Array Signal Processing

By:

Jeremy Bass
Claiborne McPheeters
James Finnigan
Edward Rodriguez

Array Signal Processing

By:
Jeremy Bass
Claiborne McPheeters
James Finnigan
Edward Rodriguez

Online:
< http://cnx.org/content/col10255/1.4/ >

C O N N E X I O N S

Rice University, Houston, Texas

Table of Contents

Chapter 1

Array Signal Processing: An Introduction[1]

1.1 Introduction and Abstract

Array signal processing is a part of signal processing that uses sensors that are organized in patterns, or arrays, to detect signals and to determine information about them. The most common applications of array signal processing involve detecting acoustic signals, which our project investigates. The sensors in this case are microphones and, as you can imagine, there are many ways to arrange the microphones. Each arrangement has advantages and drawbacks based on what it enables the user to learn about signals that the array detects. We began the project with the goal of using an array to listen to relatively low frequency sounds (0 to 8 kHz) from a specific direction while attenuating all sound not from the direction of interest. Our project demonstrates that the goal, though outside the capabilities of our equipment, is achievable and that it has many valuable applications.

Our project uses a simple but fundamental design. We created a six-element uniform linear array, or (ULA), in order to determine the direction of the source of specific frequency sounds and to listen to such sounds in certain directions while blocking them in other directions. Because the ULA is one dimensional, there is a surface of ambiguity on which it is unable to determine information about signals. For example, it suffers from 'front-back ambiguity,' meaning that signals incident from 'mirror locations' at equal angles on the front and back sides of the array are undistinguishable. Without a second dimension, the ULA is also unable to determine how far away a signal's source is or how high above or below the array's level the source is.

[1]This content is available online at <http://cnx.org/content/m12561/1.6/>.

Uniform Linear Array

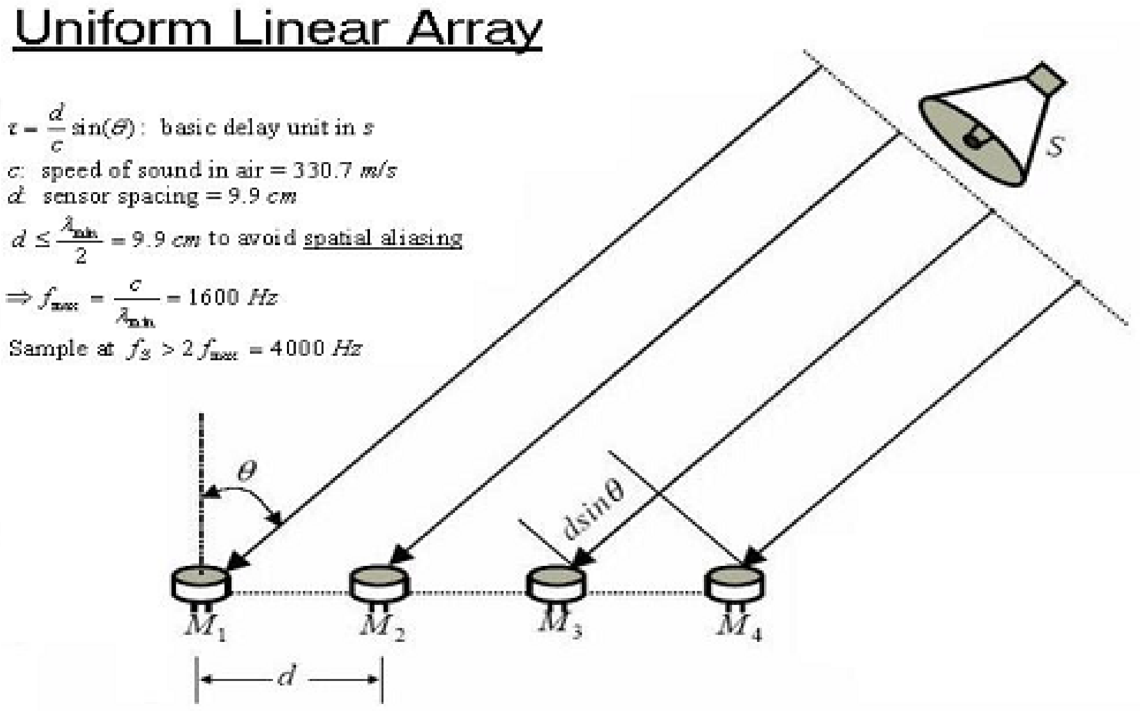

Uniform Linear Array
$\tau = \frac{d}{c}\sin(\theta)$: basic delay unit in s
c: speed of sound in air = 330.7 m/s
d: sensor spacing = 9.9 cm
$d \le \frac{\lambda_{min}}{2}$ = 9.9 cm to avoid spatial aliasing
$\Rightarrow f_{max} = \frac{c}{\lambda_{min}}$ = 1600 Hz
Sample at $f_S > 2 f_{max}$ = 4000 Hz
S
θ
dsinθ
M_1
M_2
M_3
M_4
d

Figure 1.1: The ULA is the simplest array design, though it has limitations.

When constructing any array, the design specifications should be determined by the properties of the signals that the array will detect. All acoustic waves travel at the speed of sound, which at standard temperature and pressure of 0 degrees celsius and 1 atm, is defined as :

$$c \equiv 330.7\frac{m}{s} \tag{1.1}$$

The physical relationship describing acoustic waves is similar to that of light: $\lambda f = c$. The frequencies of signals that an array detects are important because they determine constraints on the spacing of the sensors. The array's sensors sample incident signals **in space** and, just as aliasing occurs in analog to digital conversion when the sampling rate does not meet the Nyquist criterion, aliasing can also happen in space if the sensors are not sufficiently close together.

A useful property of the ULA is that the delay from one sensor to the next is uniform across the array because of their equidistant spacing. Trigonometry reveals that the additional distance the incident signal travels between sensors is $d\sin(\theta)$. Thus, the time delay between consecutive sensors is given by:

$$\tau = \frac{d}{c}\sin(\theta) \tag{1.2}$$

Say the highest narrowband frequency we are interested is $f_{\max}$. To avoid spatial aliasing, we would like to limit phase differences between spatially sampled signals to π or less because phase differences above π cause incorrect time delays to be seen between received signals. Thus, we give the following condition:

$$2\pi\tau \mathrm{f}_{\max} \le \pi \tag{1.3}$$

Substituting for τ in (2), we get

$$d \leq \frac{c}{2f_{\max}\sin(\theta)} \tag{1.4}$$

The worst delay occurs for $\theta = 90\,^\circ$, so we obtain the **fundamentally important condition**

$$d \leq \frac{\lambda_{\min}}{2} \tag{1.5}$$

for the distance between sensors to avoid signals aliasing in space, where we have simply used $\lambda_{\min} = \frac{c}{f_{\max}}$.

> We refer to the direction perpendicular to the length of the array as the **broadside** of the array. All angles to the right, or clockwise from the broadside are considered positive by convention up to $+90\,^\circ$. All angles to the left, or counter-clockwise from the broadside are considered negative up to $-90\,^\circ$.

Just think of spatial sampling in a similar sense as temporal sampling: the closer sensors are, the more samples per unit distance are taken, analogous to a high sampling rate in time!

Ambiguity of the ULA

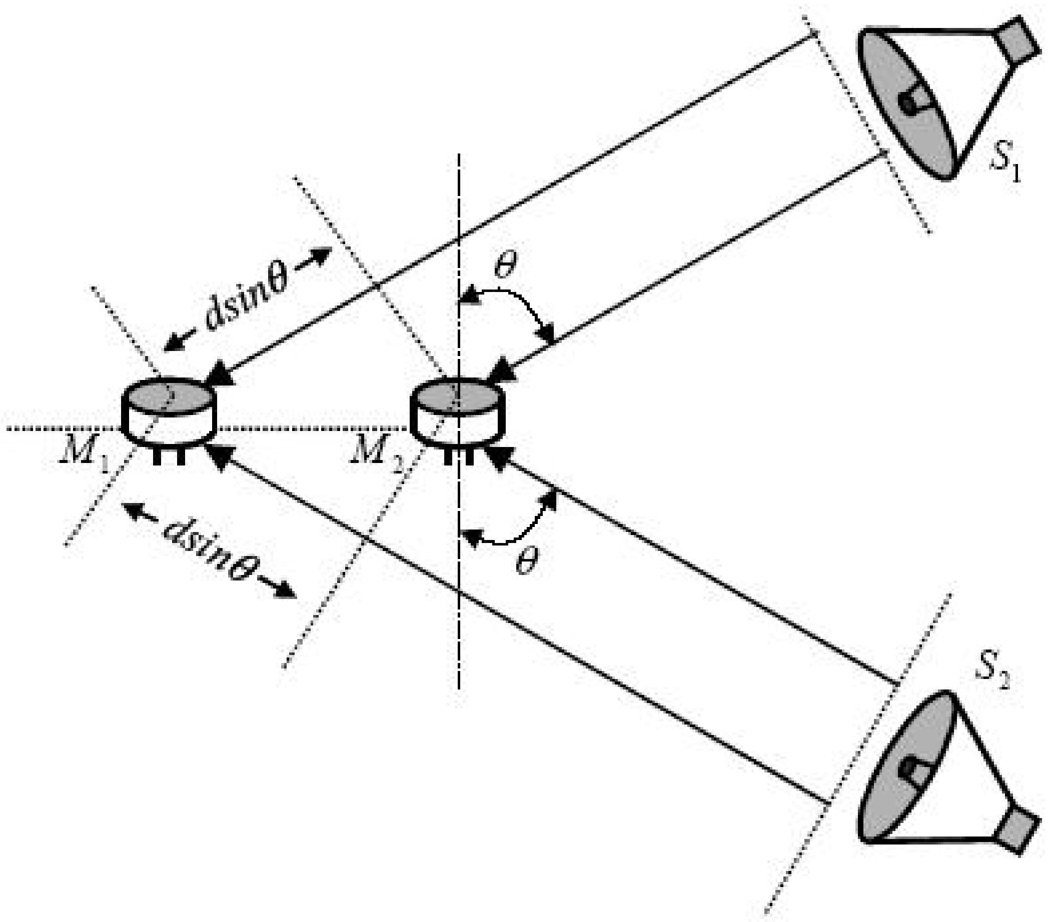


Figure 1.2: The ULA is unable to distinguish signals from it's front or back side, or signals above or below it.

The limitations of the ULA obviously create problems for locating acoustic sources with much accuracy. The array's design is highly extensible, however, and it is an important building block for more complex arrays such as a cube, which uses multiple linear arrays, or more exotic shapes such as a circle. We aim merely to demonstrate the potential that arrays have for acoustic signal processing.

Chapter 2

Beamforming Basics[1]

2.1 Introduction to Beamforming

Beamformer Basics
Beamforming is just what the name sounds like, no pun intended. Beamforming is the process of trying to concentrate the array to sounds coming from only one particular direction. Spatially, this would look like a large dumbbell shaped lobe aimed in the direction of interest. Making a beamformer is crucial to meet one of the goals of our project, which is to listen to sounds in one direction and ignore sounds in other directions. The figure below, while it accentuates what we actually accomplished in Labview, it illustrates well what we want to do. The best way to not listen in 'noisy' directions, is to just steer all your energy towards listening in one direction. This is an important concept, because it is not just used for array signal processing, it is also used in many sonar systems as well. RADAR is actually the complete opposite process, so we will not deal with that.

[1]This content is available online at <http://cnx.org/content/m12563/1.6/>.

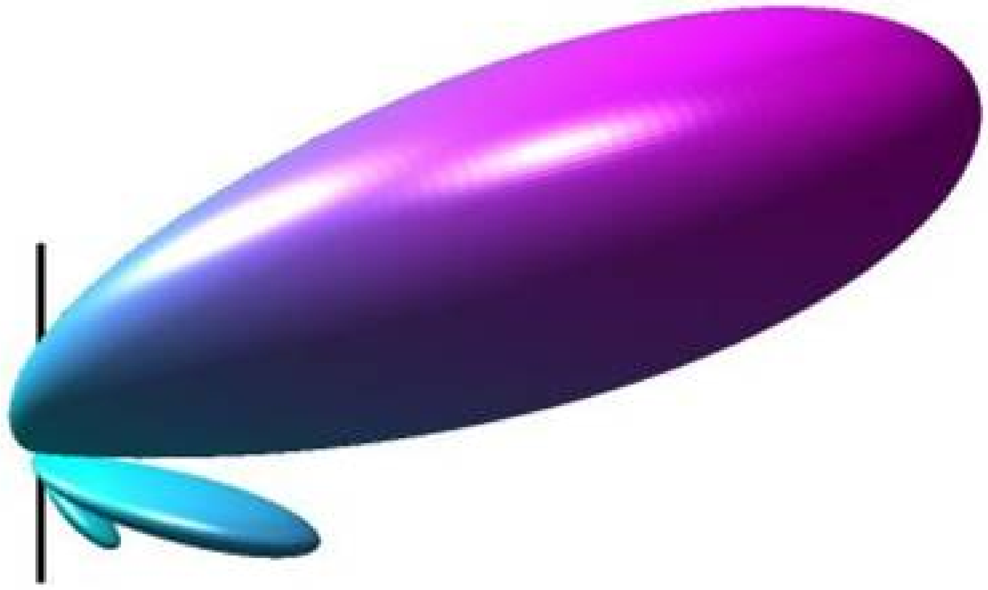


Figure 2.1: Visualization of a Beamformer

Delay & Sum Beamformers
Even though we did not use a delay and sum beamformer for the implementation of our project, it is a good first step to discuss, because it is the simplest example. While, we were doing research for this project one of the first beamformers we learned about was the delay and sum beamformer because of its simplicity. The delay and sum beamformer is based on the idea that if a ULA is being used, then the output of each sensor will be the same, except that each one will be delayed by a different amount. So, if the output of each sensor is delayed appropriately then we add all the outputs together the signal that was propagating through the array will reinforce, while noise will tend to cancel. In the Introductory module, we discussed what the time delay is for a linear array, so since the delay can be found easily, we can delay each sensor appropriately. This would be done by delaying the first sensor output by nτ, where n is the sensor number after the first. A block diagram of this can be seen below.

Image not finished

Figure 2.2: Block Diagram for a Delay & Sum Beamformer

Does this Really Work?
This seems too simple and to easy to work in practice. Delay and sum beamformers are not very commonly used in practical applications, because they do not work to well, but they do explain a tricky concept simply,

which is why they are often used to introduce beamforming. The problem with further development of time-domain beamformers, such as the delay and sum is that time-domain beamformers are often difficult to design. It is often easier to look at the frequency domain to design filters, which we can then use to steer the attention our array. However, this is no ordinary frequency analysis, this is Spatial Frequency![2]

[2]http://cnx.rice.edu/content/m12564/latest/

Chapter 3

Developing the Array Model and Processing Techniques[1]

We continue to develop the properties for the uniform linear array (ULA) that has been discussed previously[2] . With the important relationship that we found to avoid spatial aliasing, $d \leq \frac{\lambda_{\text{min}}}{2}$, we now consider the theoretical background of the ULA. Once we understand how the array will be used, we will look at a method called 'beamforming' that directs the array's focus in specific directions.

3.1 Far-field Signals

We saw previously that a very nice property of the ULA is the constant delay between the arrival of a signal at consecutive sensors. This is only true, however, if we assume that plane waves arrive at the array. Remember that sounds radiate spherically outward from their sources, so the assumption is generally not true! To get around that problem, we only consider signals in the **far-field**, in which case the signal sources are far enough away that the arriving sound waves are essentially planes over the length of the array.

Definition 3.1: Far-field source
A source is considered to be in the far-field if $r > \frac{2L^2}{\lambda}$, where r is the distance from the source to the array, L is the length of the array, and λ is the wavelength of the arriving wave.

If you have an array and sound sources, you can tell whether the sources are in the far-field based on what angle the array estimates for the source direction compared to the actual source direction. If the source is kept at the same angle with respect to the broadside and moved further away from the array, the estimate of the source direction should improve as the arriving waves become more planar. Of course, this only works if the array is able to accurately estimate far-field source directions to begin with, so use the formula first to make sure that everything works well in the far-field, and then move closer to see how distance affects the array's performance.

Near-field sources are beyond the scope of our project, but they are not beyond the scope of array processing. For more information on just about everything related to array processing, take a look at **Array Signal Processing: Concepts and Techniques**, by Don H. Johnson and Dan E. Dudgeon, Englewood Cliffs, NJ: Prentice Hall, 1993.

[1]This content is available online at <http://cnx.org/content/m12562/1.3/>.
[2]http://cnx.rice.edu/content/m12561/latest/

3.2 Array Properties

Depending on how the array will be used, it may be important (as it was in our project) that the microphones used be able to receive sound from all directions. We used **omni-directional** microphones, which are exactly what they sound like – microphones that hear in all directions. If you don't need this ability, you can look into other options, but keep in mind that the theoretical development here assumes omni-directional capability, so you will need to do some research on array processing techniques that suit your needs. In fact, it would be a good idea no matter what! Our project used a simple array design, but it took a while to learn all of the theory and to figure out how to implement it.

Our array comprises six generic omni-directional microphones. We built an array frame out of PVC pipe to hold each microphone in place with equidistant spacing between the sensors. Physical limitations of the microphones and PVC connecting pieces prevented us from using a very small spacing; for our array, we had a sensor spacing of d=9.9 cm. Since we know that we need to have $d \leq \frac{\lambda_{\min}}{2}$ to avoid spatial aliasing, we are able to calculate the highest frequency that the array is capable of processing: $f_{\max} = 1600\text{Hz}$. (Actually, $f_{\max}$ is a little higher than 1600 Hz as you can verify, but to be on the safe side we kept it a bit lower.)

If you want to have any chance of figuring out some useful information about a signal, particularly in real-time, you're going to have to ditch the pencil and paper for some electronic equipment. We used National Instruments' LabVIEW 7.1 to do all of our signal processing, although we performed some analog conditioning on the received signal before the analog to digital conversion (ADC). We also used National Instruments' 6024e data acquisition card to digitize the signal. This is a multiplexed ADC with a total sampling capacity of 200 kHz that divides between the number of inputs. Therefore, with six sensor inputs, we could sample the signals received at each microphone at a maximum rate of 33.3 kHz. Since twice the Nyquist rate for speech is about 44.1 kHz, this is not a good DAQ for speech applications; however, it would have worked for our original plan to listen to low frequency sound in the 0 to 8 kHz range. As it turns out, since our array can process a maximum frequency of 1600 Hz, we chose to sample at $f_s = 4000\text{Hz}$, which exceeds the Nyquist requirement and is well within the capability of our DAQ.

All of these properties generealize to determining the design of any ULA or the design of any array, though other designs may have greater capabilities and thus would require that you consider additional signal properties (e.g., signal elevation above or below the array) and how they affect the array. If you need a starting point, think about the range of frequencies that you are interested in and get equipment that is capable of processing them. That includes an ADC that can sample at a high enough rate to avoid temporal aliasing and the materials to construct an array such that spatial aliasing will not occur. You will probably have to do some pre-conditioning of the signal before you digitize it, such as lowpass filtering to reject frequencies above those you are interested in and applying a gain to the input signals. These are all important things to think about when you design your array!

3.3 ULA Processing Fundamentals

Now it's time to look at the theory that we need to implement for a ULA that enables us to figure out where signals come from and to listen to them. We are considering narrowband signals (i.e., sinusoids) of the form

$$x\left(t\right) = e^{\mathrm{j}2\pi \mathrm{f}t} \tag{3.1}$$

where f is the frequency of the signal. If we have N sensors numbered from n=0,...,N-1, then the delayed versions of x(t) that arrive at each microphone n are

$$x_n\left(t\right) = e^{\mathrm{j}2\pi \mathrm{f}(t-\mathrm{n}\tau)} \tag{3.2}$$

Thus, the first sensor (n=0) has zero delay, while the signal arrives at the second sensor (n=1) one unit delay later than at the first, and so on for each sensor. Then, we sample the signal at each microphone in the process of ADC, and call $x_n\left(r\right) = x_n\left(mT\right)$, where m is the integers. This gives us the sampled sinusoids at each sensor:

$$x_n\left(r\right) = e^{\mathrm{j}2\pi \mathrm{f}(r-n\tau)} \tag{3.3}$$

Now we need to do some Fourier transforms to look at the frequency content of our received signals. Even though each sensor receives the same frequency signal, recall that delays $x\left(t - n\tau\right)$ in time correspond to modulation by $e^{-jn\tau}$ in frequency, so the spectra of the received signals at each sensor are not identical. The first Discrete Fourier Transform (DFT) looks at the temporal frequency content at each sensor:

$$X_n\left(k\right) = \frac{1}{\sqrt{R}}\sum_{r=0}^{R-1} e^{\mathrm{j}2\pi\mathrm{f}\left(r-n\tau\right)}e^{-\frac{\mathrm{j}2\pi\mathrm{kr}}{R}} \tag{3.4}$$

$$\frac{e^{-\mathrm{j}2\pi fn\tau}}{\sqrt{R}} \tag{3.5}$$

for k=fn, and zero otherwise. Here we have used the definition of the normalized DFT, but it isn't particularly important whether you use the normalized or unnormalized DFT because ultimately the transform factors 1/Sqrt(R) or 1/R just scale the frequency coefficients by a small amount.

Now that we have N spectra from each of the array's sensors, we are interested to see how a certain frequency of interest fo is distributed spatially. In other words, this **spatial Fourier transform** will tell us how strong the frequency fo for different angles with respect to the array's broadside. We perform this DFT by taking the frequency component from each received signal that corresponds to fo and concatenating them into an array. We then zero pad that array to a length that is a power of two in order to make the Fast Fourier Transform (FFT) computationally efficient. (Every DFT that we do in this project is implemented as an FFT. We use the DFT in developing the theory because it applies always, whereas the FFT is only for computers.)

NOTE: When we build the array of frequency components from each of the received signals, we have an N length array before we zero pad it. Let's think about the **resolution** of the array, which refers to its ability to discriminate between sounds coming from different angles. The greater the number of sensors in the array, the finer the array's resolution. Therefore, what happens when we zero pad the array of frequency components? We are essentially adding components from additional 'virtual sensors' that have zero magnitude. The result is that we have improved resolution! What effect does this have? Read on a bit!.

Once we have assembled our zero padded array of components fo, we can perform the spatial DFT:

$$\Omega\left(k\right) = \frac{1}{\sqrt{\mathrm{NR}}}\sum_{n=0}^{N-1} e^{-\mathrm{j}2\pi\left(\frac{k}{N}+f_0\tau\right)} \tag{3.6}$$

where N for this equation is the length of the zero padded array and R remains from the temporal DFT. The result of the summation is a spectrum that is a digital sinc function centered at $f_0\tau$. The value of the sinc represents the magnitude of the frequency fo at an angle theta. Because of the shape of the lobes of the sinc, which look like beams at the various angles, the process of using the array to look for signals in different directions is called **beamforming**. This technique is used frequently in array processing and it is what enabled us to detect the directions from which certain frequency sounds come from and to listen in different directions.

Recalling that we zero padded our array of coefficients corresponding to f_0, what has that done for us in terms of the spatial spectrum? Well, we have improved our resolution, which means that the spectrum is smoother and more well-defined. This is because we are able to see the frequency differences for smaller angles. If we increase the actual number of sensors in the array, we will also improve our resolution and we will improve the beamforming by increasing the magnitude of the main lobe in the sinc spectrum and decreasing the magnitudes of the side lobes.

Chapter 4

Spatial Frequency[1]

4.1 Temporal Frequency

Problems with Temporal Frequency Analysis

We are accustomed to measuring frequency in terms of (1/seconds), or Hz. Sometimes we may even measure frequency in rad/sec, which is often called angular frequency. More information about temporal frequency can be found here[2] . The reason we often use frequency domain techniques is because it allows for filtering[3] of noise and various signals. If were just interested in listening to a particular tone, say a 500 Hz sine wave, it would be easy to just tune in to that one frequency, we would just bandpass filter out all the other noise. However, when we do this we get no information about where the signal is coming from. So, even though we could easily ignore noise, we could not steer our array to just listen in one direction. It would be more like giving it 'selective hearing.' It hears what it wants to, which in this case would be signals at 500 Hz.

4.2 Propagating Waves

Nature of Waves

As Dr. Wilson discusses in his modules[4] on waves propagating down a transmission line, waves carry two forms information in two domains. These domains are the time and space domains, because the wave equation is usually written in terms of s(x,t) because it propagates in space at a particular time, and if one looks at standing wave at a particular point in space, one should notice that it still moves up and down in a similar manner. An example of this illustrated below. So, if we only look at the temporal frequency component, we are missing out on half the information being transmitted in the propagating signal! If we really want to be able to steer our array in a direction, then we need to analyze the spatial frequency components.

[1]This content is available online at <http://cnx.org/content/m12564/1.4/>.
[2]http://cnx.rice.edu/content/m0038/latest/
[3]http://cnx.rice.edu/content/m0533/latest/
[4]http://cnx.rice.edu/content/m10095/latest

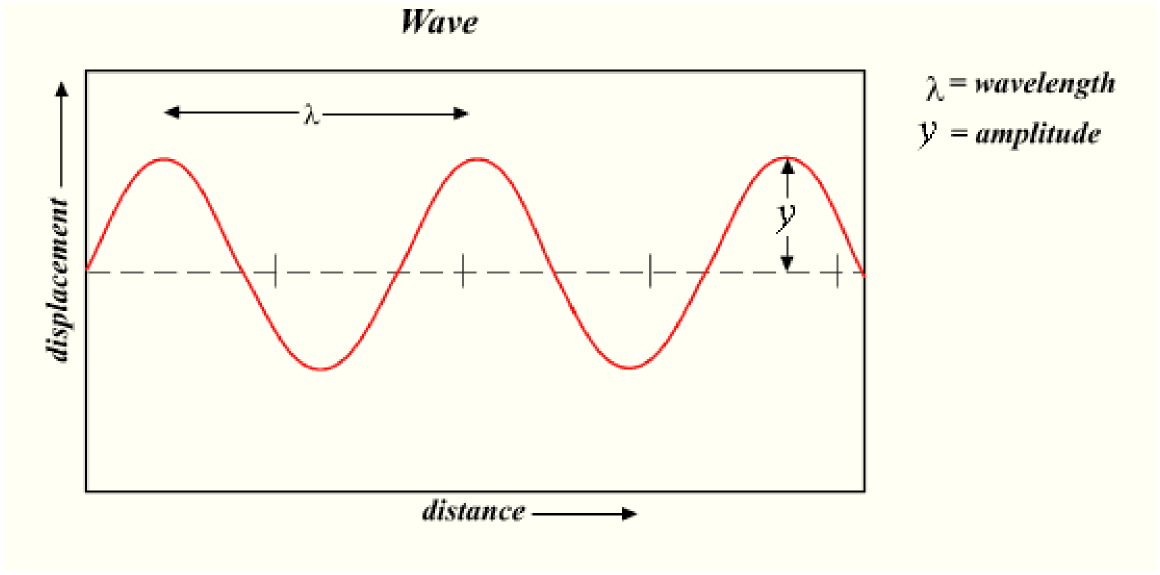


Figure 4.1: Illustration of a wave propagating in space

4.3 Spatial Frequency

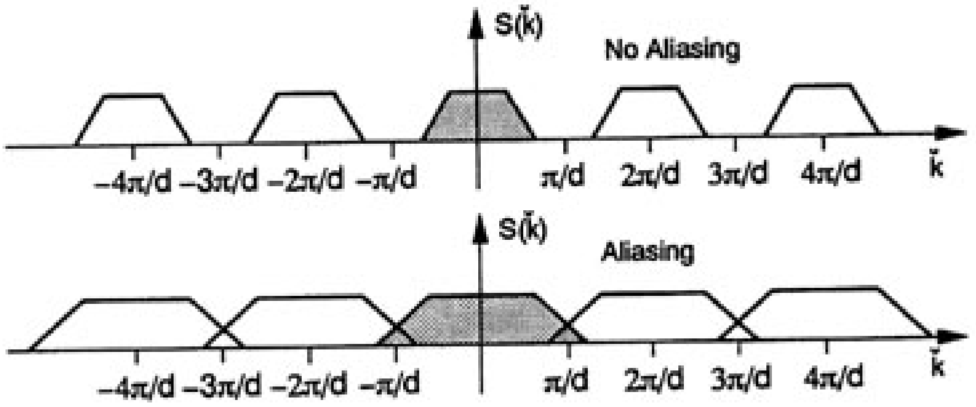
Introduction to Spatial Frequency
While we were investigating the time domain, we were able to accomplish such operations as filtering by taking $2\pi\ /\ T$, where T is the period of the signal, to get the temporal frequency denotated ω. We can use similar reasoning to obtain k, the wavenumber, which is the measure of spatial frequency. Instead of using the period of the signal, we now use the wavelength, which is the spatial equivalent to the period. This makes sense, because a period is the length of time it takes to complete one cycle, whereas the wavelength is the amount of distance the wave covers in one cycle. We there are able to change the temporal frequency equation $\omega = 2\pi$ / T into k = 2π / λ.

Chapter 5

Spatial Frequency Analysis[1]

5.1 Aliasing in the Spatial Frequency Domain

Avoiding Spatial Aliasing
As we saw was the case in the time domain, a phenomenon known as aliasing[2] can occur in the frequency domain if signals are not sampled at high enough rate. We have the same sort of considerations to take into account when we want to analyze the spectrum of the spatial frequency as well. As was discussed in the introduction[3] , the Nyquest equivalent of the sampling rate is 1/2 of the **minimum** wavelength. This comes about from the relationship between speed, frequency and wavelength, which was discussed in the introduction as well. The figure below demonstrates the effects of aliasing in the spatial domain; it looks identical to filtering the time domain except that instead of the x-axis being related to pi/T it is now pi/d, where d is the distance between sensors. So, if we bandlimit our signal in temporal frequency, so that we can sample as two times the maximum **temporal** frequency, and if we design the sensors so that half of the minimum wavelength is greater than distance between sensors, we can avoid aliasing in both time and space!



$S(\check{k})$
No Aliasing
$-4\pi/d$ $-3\pi/d$ $-2\pi/d$ $-\pi/d$ π/d $2\pi/d$ $3\pi/d$ $4\pi/d$ $\check{k}$
$S(\check{k})$
Aliasing
$-4\pi/d$ $-3\pi/d$ $-2\pi/d$ $-\pi/d$ π/d $2\pi/d$ $3\pi/d$ $4\pi/d$ $\check{k}$

Figure 5.1: Spatial Aliasing

[1]This content is available online at <http://cnx.org/content/m12557/1.5/>.
[2]http://cnx.rice.edu/content/m10793/latest/
[3]http://cnx.rice.edu/content/m12561/latest/

5.2 Spatial Frequency Transform

Introduction to the Spatial Frequency Transform
Analogous to the DFT[4] , is the sampled and windowed spatial equivalent, which is what we used to be able to filter our signal in frequency. The reason we want the information in the spatial frequency or wavenumber domain is because it is directly correlated to the angle the signal is coming from relative to the ULA. The spatial DFT is computed as the FFT of the first FFT. The first FFT represents the time domain frequency response and the second FFT represents the wavenumber response. This seems strange this would work, but let's explore this a little more fully. Let's look at theoretical example.

Example 5.1: Mentally Visualizing the Spatial Frequency Transform
The 2-D Transform
Consider a box filled with numbers. The box is labeled on one edge time and on the other edge space. The first FFT we are taking is to obtain the temporal frequencies, so this would be like looking at a row along the box and taking the FFT of the numbers going across, while the spatial FFT would be calculated by looking at the numbers going down the columns. This is done repeatedly on each row and column, so the first FFT would go across each row, while the 2nd one would go down each column. This is easier to comprehend with a picture like the one below.

[4]http://cnx.rice.edu/content/m10249/latest/

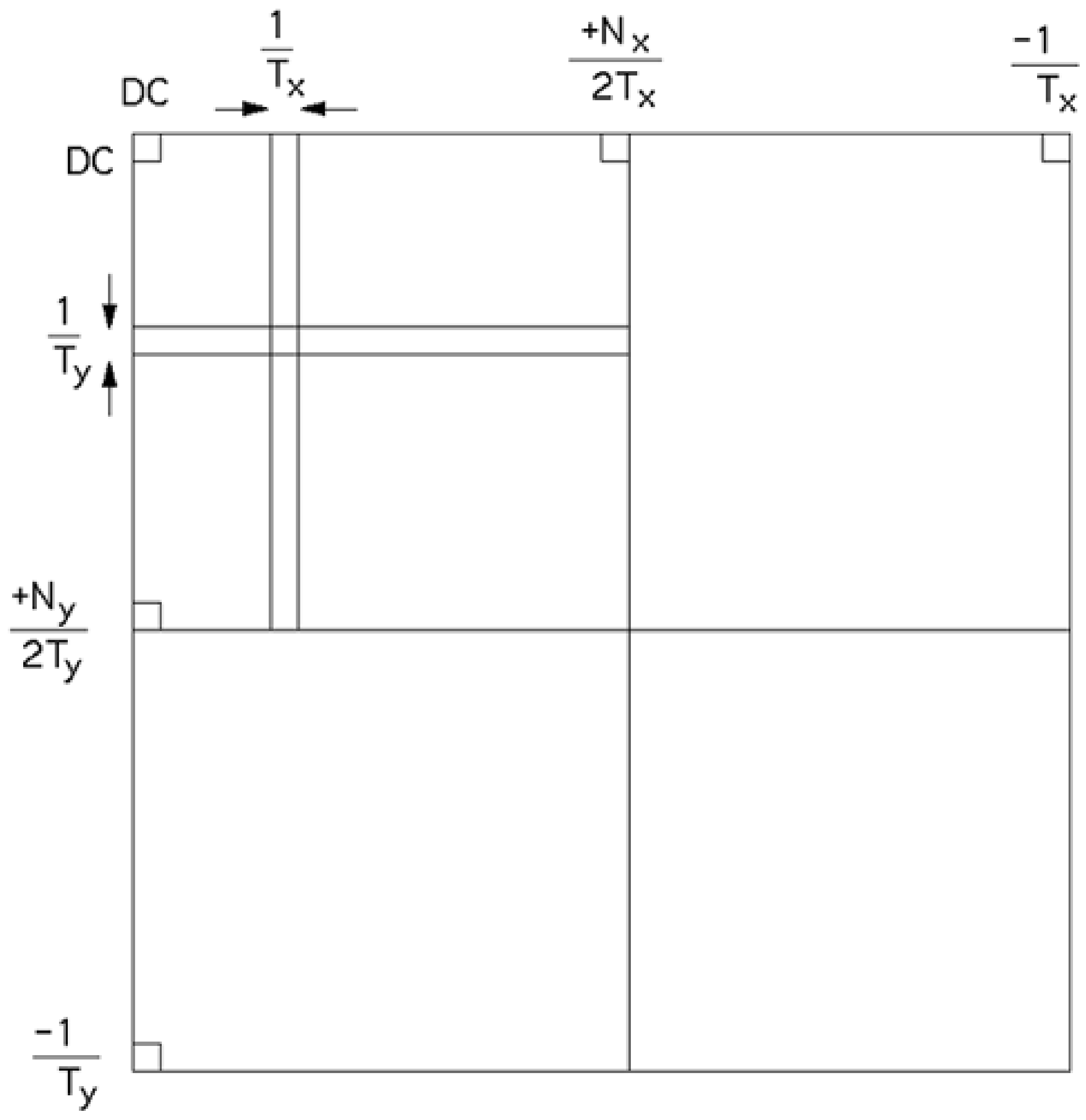


Figure 5.2: Visualization of mapping a signal into Spatial & Temporal Frequencies

SFT with Sinusoids
Since we were interested in detecting sinusoids, it would be interesting to consider what this kind of "double" Fourier Transform would do to a sinusoid. From our list of Fourier Transforms[5] we know that the FFT of a sinusoid will give us a delta function shifted by the frequency of the sinusoid. We then see that the FFT of a delta function is 1, which would mean that we get the equivalent of white noise in spatial frequency! Fortunately, this is not exactly how the spatial FFT works. We are basically taking the FFT across one set of vectors followed by the FFT down the columns of those vectors, we are NOT taking the FFT(FFT(f(x,t)). So, when we accomplish this sort of arrangement on our signal, f(x,t), we get:

[5]http://cnx.rice.edu/content/m10099/latest

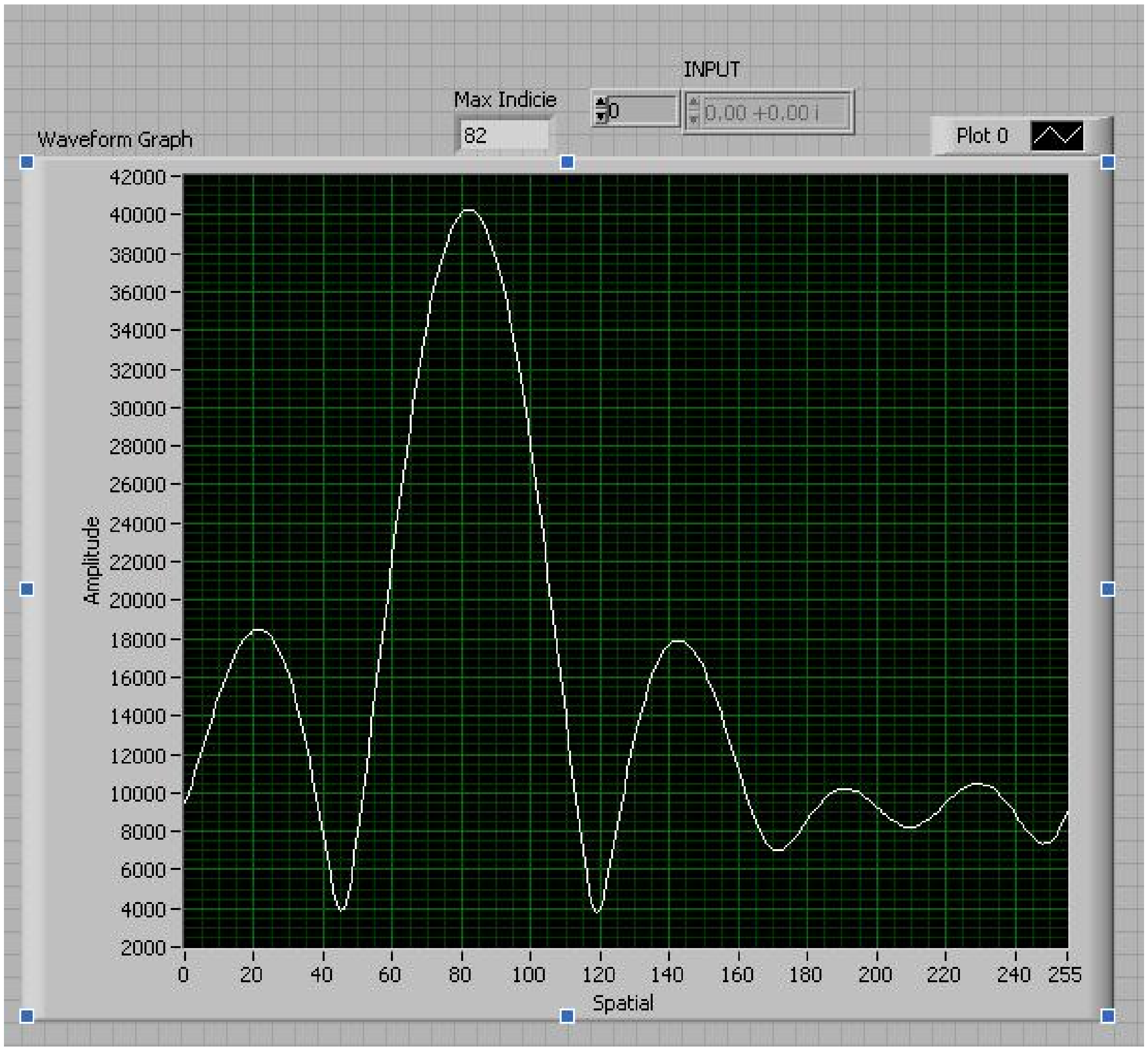


Figure 5.3: Spatial FFT of a Sinusoid

A sinc function!

5.3 Spatial Domain Filtering

Just as we are able to filter signals in **temporal** frequency, we can filter signals in **spatial** frequency. In fact, the way we accomplished the direction detecting algorithm in labview used a graph very similiar as the one above and then looking for the largest magnitude part of the signal. Once, this value is known, quick computation can then find the angle that signal came from! Ta da! We're done! Well, sort of.

Chapter 6

Labview Implementation[1]

6.1 Labview VIs Used in Simulation

Top Level Diagram and Overall Organization

The figure below is a top level diagram of a six microphone array simulation VI. This VI is named "Multiple Frequency Simulation" Starting at the left with the signal generation, the code flows to the right. This top VI is made out of smaller VIs that each perform a specific function. Detailed descriptions of each of these can be found in the following paragraphs.

Top Level Diagram (Multiple Frequency Simulation)

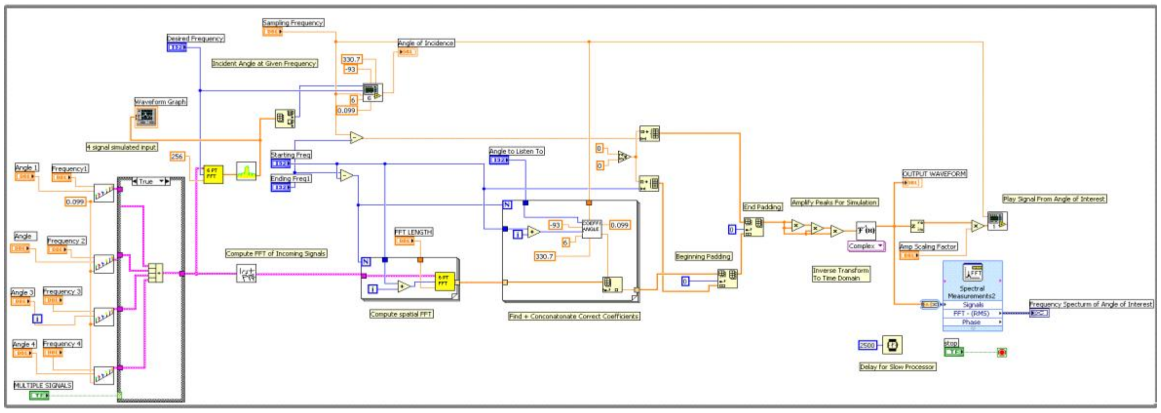


Figure 6.1: Top Level Diagram

[1]This content is available online at <http://cnx.org/content/m12565/1.10/>.

6.2 Simulated Signal Generation VI

Simulate Signal Icon

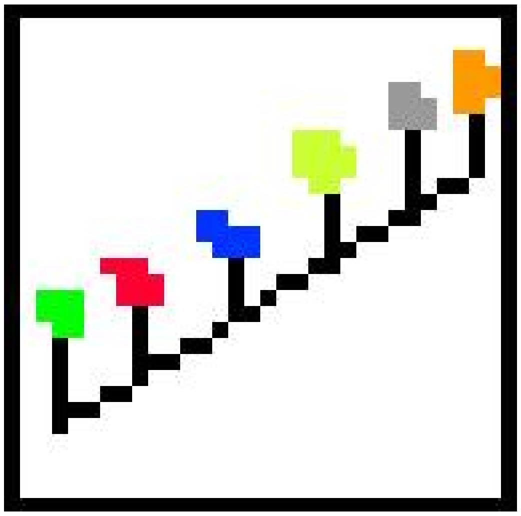


Figure 6.2: Simulate Signal Icon

Signal Generation VI

The above icon corresponds to the Signal Generation VI. At the far left are four signal generation VIs. Each of these VIs produces six different signals that simulate the delayed signals that each of the microphones would hear in real life. These six signals are then bunched together in a cluster (pink wire) to keep the amount of wires down. The three inputs to this VI are the frequency of the sinusoid desired, the direction of the signal, and the distnace between microphones (9.9 cm in this example). The user sets these paramaters on the front panel and is free to change them at any time.

Our VI uses the formula discussed in previous modules that relates the time delay of signals to the distance between microphones. Using the desired angle that the signal is coming from, the speed of sound (C), the distance between microphones, and a conversion between degrees and radians, the VI first computes the time delay between microphones. Becuase we are interested in the delay between the first microphone and all others, the amount of delay is multiplied by zero for the first mic, one for the second mic, two for the third, and so on. These delays are then used as phase inputs to the six identical Simulate Signal VIs. Adding phase to a periodic signal like the sinusoids being used here has the same effect as delaying the signal in time. Finally the six signals are merged together into a cluster so that they will be easier to deal with as a group in other VIs.

Simulate Signal Block Diagram

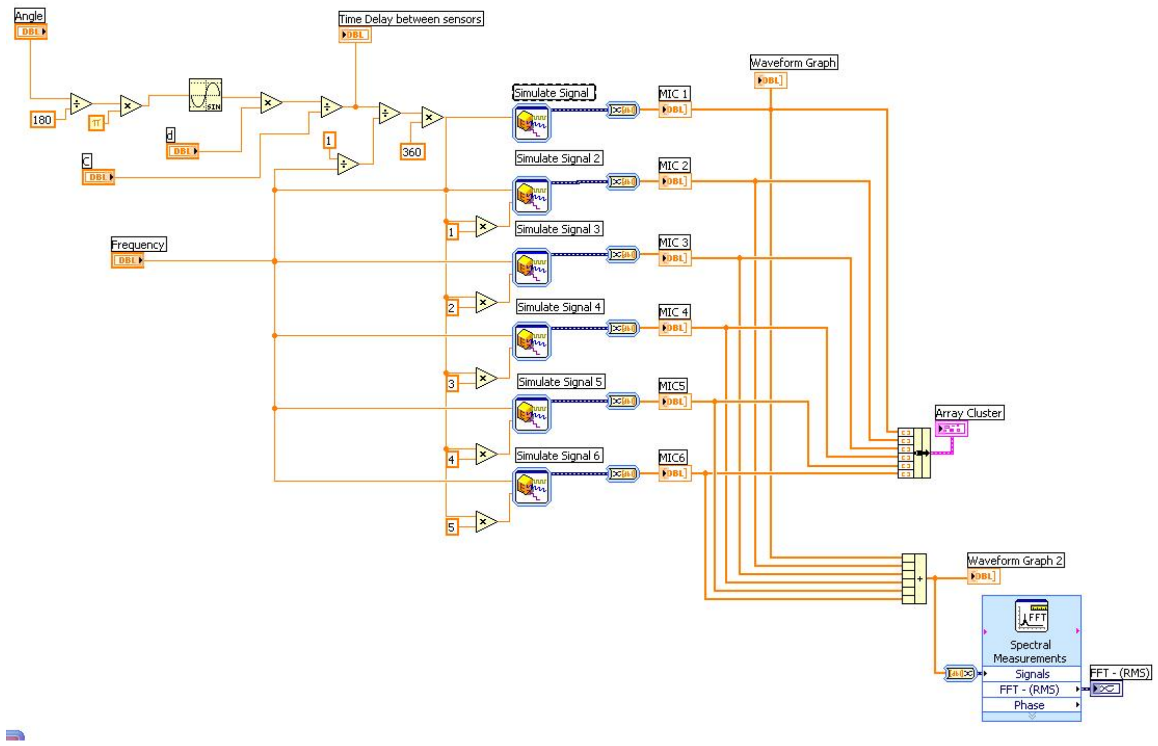


Figure 6.3: Simulate Signal Block Diagram

In this simulation we simulate four different frequencies and directions. Once we have the data of these four separate signals, we sum the signals on each of the channels to get the final output on each of the microphones. Doing so leaves us solely with what we would hear on the six microphones if this were set up in real life. If more signals are required for future testing or other applications, onc can simply copy and paste the signal generation VI and then sum it with the other signals.

Once this signal is comple, we move to the right in our VI. From here we branch into two separate areas. Following the pink line up takes us to where we calculate the angle that a given frequency is coming from,and to the right is where we perform the calculations to listen to signals from a given direction.

6.3 Frequency and Spatial FFT Computation VIs

1rst FFT Icon (Time - Frequency)

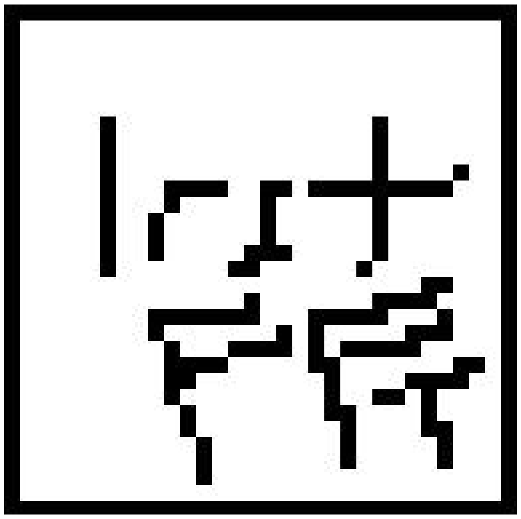


Figure 6.4: 1rst FFT Icon

1rst FFT VI
In order to listen to a certain direction we first need to transform our signals from the microphone in to the frequency domain. To do this we made the "1rst FFT" Vi (see icon above). This sub-VI is fairly simple so there is no need to show the block diagram. It takes the FFT of each of the six channels channels, transfroming the data from the time domain to the freqeuncy domain For simulation, we looked at 1 second samples at 4000hz. This VI then takes the FFTs of the six 4000 element long signals from the simulated microphones.

Six Point FFT Icon (Frequency - Spatial)

Figure 6.5: Six Point FFT Icon(Frequency - Spatial)

Six Point FFT VI

Moving to the right after the first FFT, we find the "6 pt FFT" VI. This VI is used to transform our frequency domain data into the spatial domain. This VI needs to run for every frequency of interest, so in the top level diagram this VI is found inside of a for loop. The user can control what range of frequencies are of interest on the front panel of the main VI, and the loop will run for those frequencies.

Inside this VI, the complex frequency coefficient for the given frequency is extracted from the array for each microphone channel. These six values are then concantonated into a lengh six array. Next, the array is zeropadded to a user specified length (we used 256 in our simulation) so more resolution can be observed in the resulting spatial transform. Finally, the FFT is performed on these values transforming them into the spatial domain. With the data in the spatial domain we are easily able to figure out the magnitude of any frequency from any direction in our 180 degrees of interest. How to do this can be found in the magnitude VI.

Six Point FFT block Diagram (Frequency - Spatial)

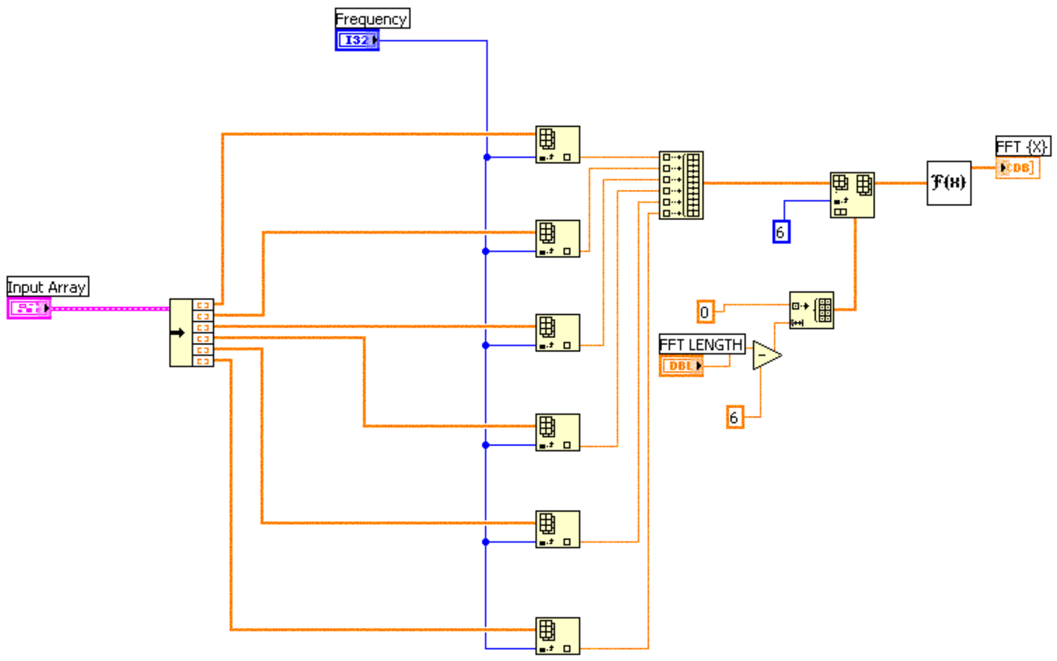


Figure 6.6: Six Point FFT block Diagram (Frequency - Spatial)

6.4 Coefficient, Angle, and Magnitude Calculation VIs

Coefficient Calculator

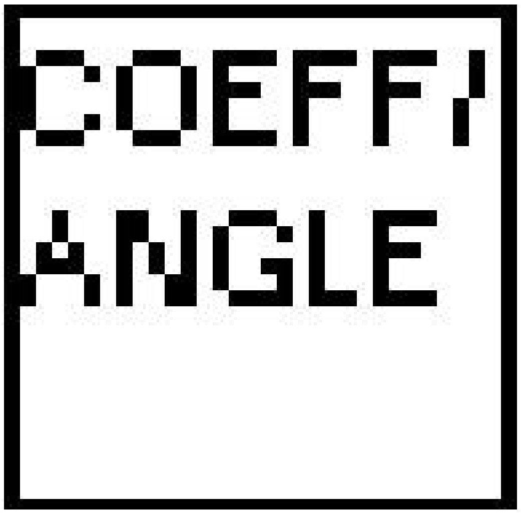


Figure 6.7: Coefficient Calculator

Coefficient Calculator
Now that the signals have been transformed to their spatial representations, the correct coefficients to construct the signal of interest must be computed. To do this we created the Coefficient Calculator VI.. This VI takes in the angle of interest, a frequency, the number of microphones, a constant factor (determined by the setup of the microphones and the amount of zero padding...in this example it is -93), and our original sampling frequency to compute the correct coefficient to look at for the given angle. Once this coefficient is found, we extract the value at that coefficient and append it to our ouptut array as the value at that given frequency. Below is the block diagram for this VI. It consists of a basic formula node and some logic on the front and back to make sure that the calculations are correct when working with angles from negative directions.

Because this formula is dependent on the frequnecy of interest, we are required to run this VI for every frequency we are intereseted in. In order to do this, we put this VI inside a for loop that is controlled by our frequency range. Any coefficient for frequencies outside of this range are simply given a value of zero. The array modules ouside of this for loop are used to do just that. They append arrays with value zero on the front and back of the output of the for loop to return our vector to its original size. From here, we run this vector through a few multiplies to amplify the differnce between the coefficients with high and low magnitudes, and finally we inverse FFT it to get our output array. This array represents a signal in the time domain and is graphed on the front panel along with a graph of its frequency components. We also included a small VI that will play the output waveform on computer speakers. This VI uses a matlab script and requires the user to have matlab 6.5 or earlier.

Magnitude Graph of Coefficients After Spatial FFT

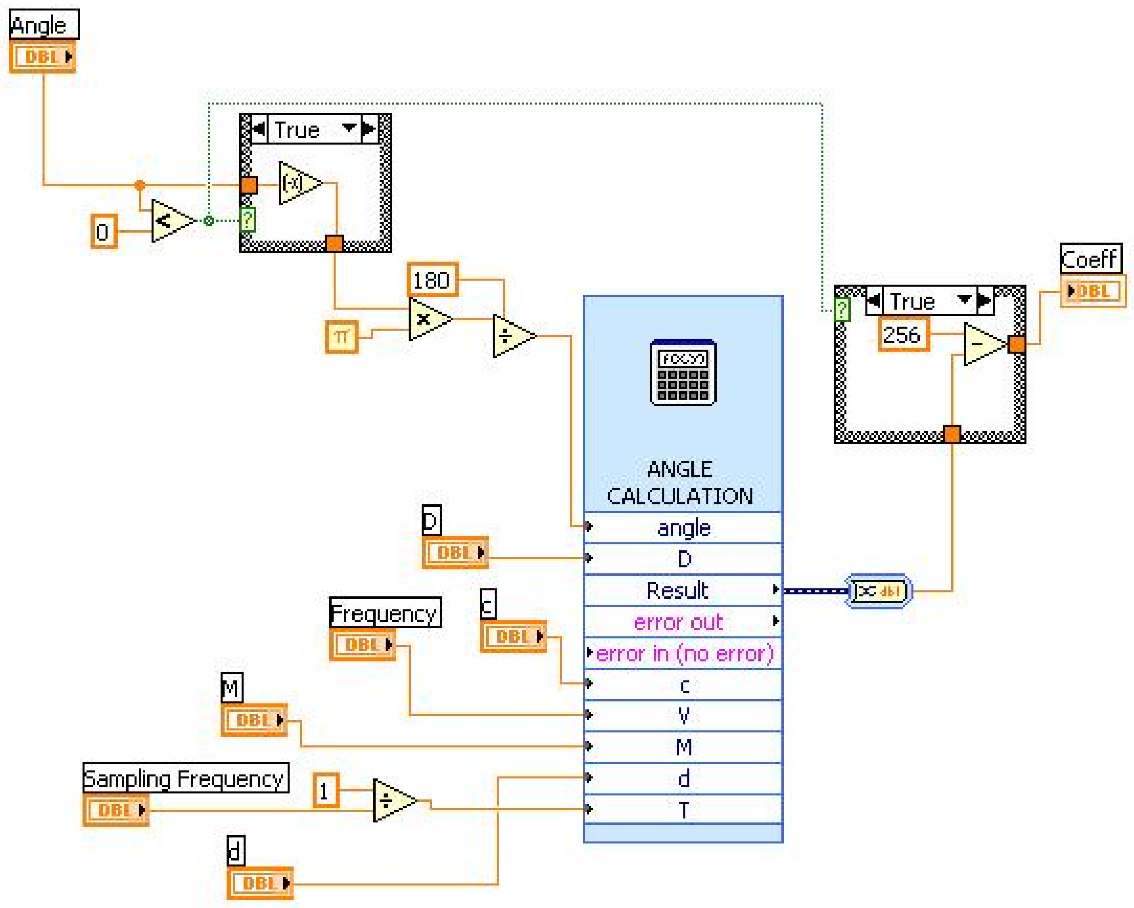


Figure 6.8: Magnitude Graph of Coefficients After Spatial FFT

Magniutde Calculation VI

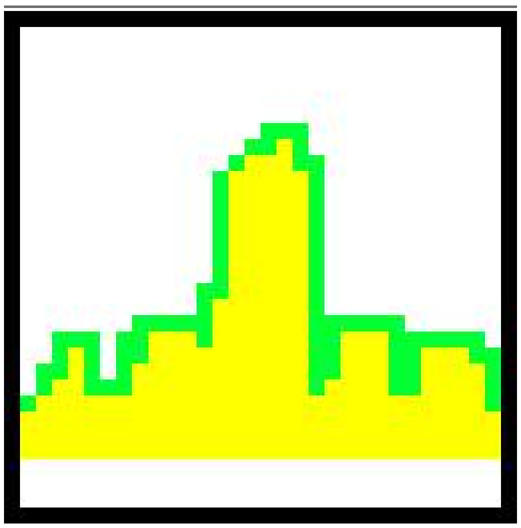


Figure 6.9: Magnitude Calculation VI

Magnitude Calculation

If we go back to the branch in the pink wire immediatly after the signal generation VIs and move upwards instead of to the right, we come across the code that is used to calculate the angle at which the largest magnitude signal of a given frequency is approaching the array. Another "6 pt fft" VI is used, but this one is slightly modified. It also includes the initial FFTs of all 6 channels. We grouped these two VIs together because ony one spacial FFT is being computed (that at the frequeny of interest).

The resulting vector of the previous six point FFT is immediately used as the input to the Magnitude Calculation VI. The vecor of spatial coefficients from the "six pt FFT" vis are complex, so this VI is used to calculate the magnitudes of the coefficients so the maximum coefficient can be found. The output of this VI is also used to create a visual representation of what direction the specified frequency is coming from. Below is a graph of the magnitude of the coefficients of the spatial FFT. As discussed before, we see the peak correspoinding to the incoming direction and the smaller ripples to each side.

Magnitude Graph of Coefficients After Spatial FFT

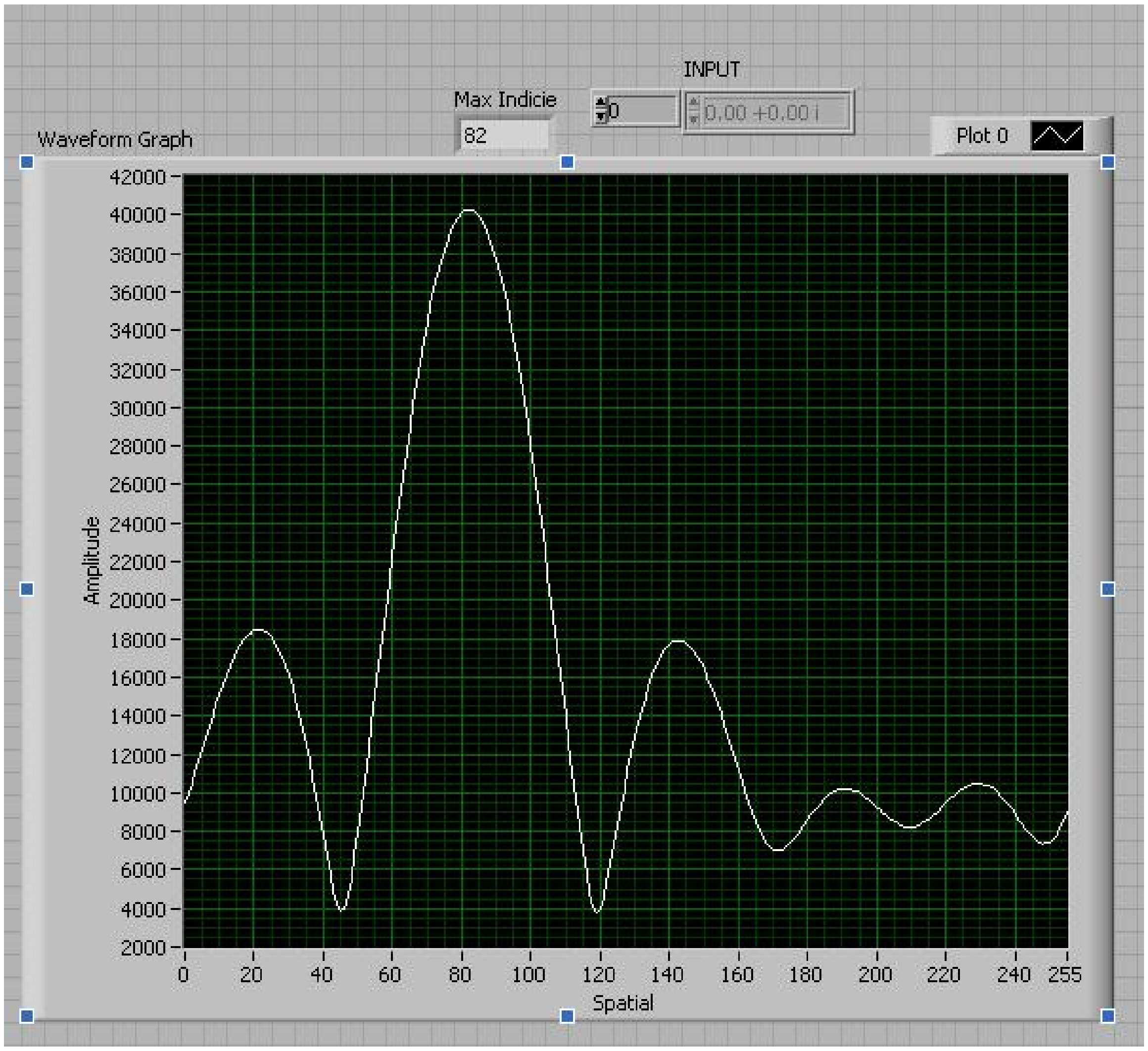


Figure 6.10: Magniutde of Coefficients after Spatial FFT

Example 6.1: Magnitude Graph Example
As we can see in the previous figure, the magnitude of the spatial FFT is greatest around coefficient 82. There are also smaller ripples that die off around each end. This graph tells us that the the direction that the given frequency is coming from corresponds to the angle represneted by coefficient 82. To figure out what angle this was, we would use our Coefficient to Angle VI.

Calculation of Angle
Finally, we isolate the index of the maximum angle and use it to compute the angle of incidence. Using the same formula used in the Coefficient Angle Calculator, the Coefficient to Angle VI deduces the angle of incidence. This VI uses the same formula found in the Coefficient Angle Calculator, but is arranged differently so that we can find the angle based on the coefficient instead of the coefficient based on the angle. Once the VI computes this value, the result is output on the front panel.

6.4.1 Links to Labview Code

- Multiple FrequencySimulation (Top Level VI)[2]
- Coefficient Calculator[3]
- Incident Angle Calculator[4]
- Play Array[5]
- Six Point FFT (with first fft)[6]
- Six Point FFT (without first fft)[7]
- Magnitude Calculation[8]
- First FFT[9]
- Simulate Signal[10]

6.5 Code Remarks

Overall, the code involved in this method of array signal processing can be broken up into smaller parts that are easy to understand. By combining these smaller parts we are able to create an upper level VI that performs a complicated task that would be difficult to get working using other methods. The major problem with this VI is that it requires a large number of calculations. To increase performance (without upgrading computers) one could decrease the frequency range of interest, or they could lower the amount of zeropadding. They could aslo look at a smaller time period.

[2]http://cnx.org/content/m12565/latest/Multiple_Frequency_Simulation.vi
[3]http://cnx.org/content/m12565/latest/Coefficient_Calculator.vi
[4]http://cnx.org/content/m12565/latest/Incident_Angle_Calculator.vi
[5]http://cnx.org/content/m12565/latest/play_array_anything.vi
[6]http://cnx.org/content/m12565/latest/Six_Point_FFT_with_first_fft.vi
[7]http://cnx.org/content/m12565/latest/Six_Point_FFT_without_first_fft.vi
[8]http://cnx.org/content/m12565/latest/Magnitude_Calculation.vi
[9]http://cnx.org/content/m12565/latest/First_FFT.vi
[10]http://cnx.org/content/m12565/latest/Simulated_Input.vi

Chapter 7

Microphone Array Simulation[1]

7.1 Why Labview Simulation

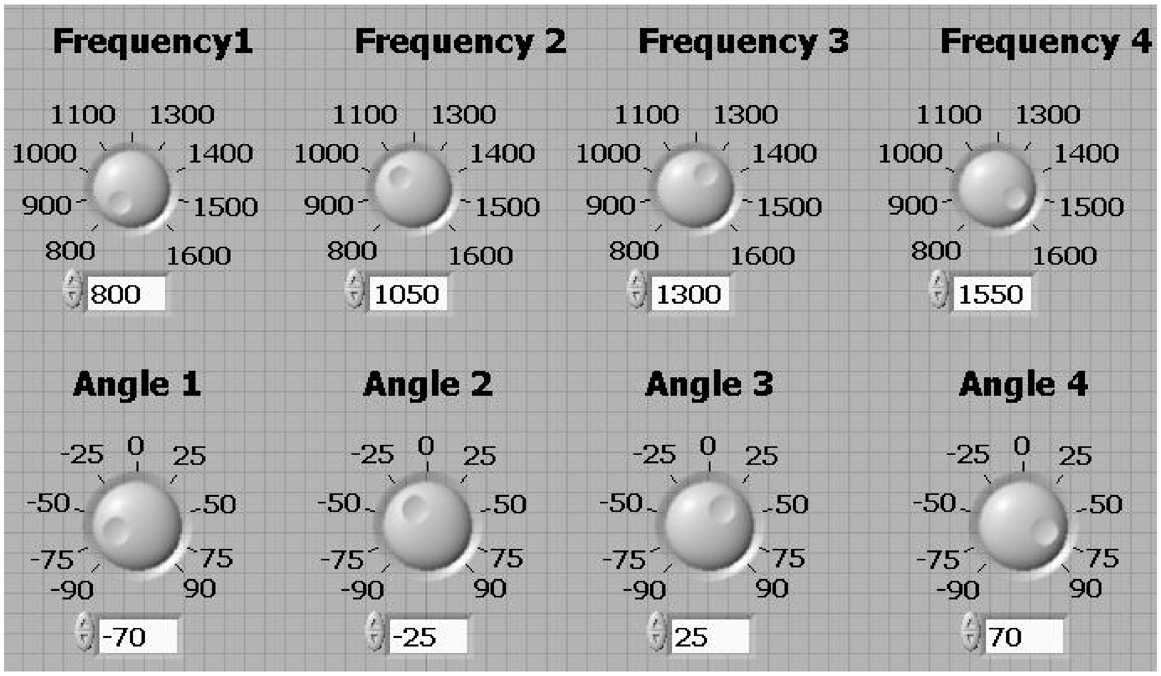
Reasons For Simulation
Simulation of the microphone array is an integral step in verifying that the design and method of implementation behind the beamforming system are correct. Additionally, Simulation allows one to easily change the paramaters of the system (number of microphones, type of signals, distance between microphones) so that the system can be optimized to the desired needs. Simulation also allows for easy testing free of noise, uncertainty, and other errors. Finally, simulation allows one to modify input signals on the fly and lets one instantaenously see if the system is working as planned.
Reasons for Simulation in Labview
There are many practical reasons why Labview is the ideal program to simulate array processing in. Labviews graphical programming environment is perfect for this type of application. The simple to use signal generator VIs and convenient output displays make controlling the results and modifying the inputs easly. Additionally, with Labview the Simulation can be easily modified to work in real life (see next module). By replacing the simulated signals with real life data acquiition, the same code is used for real-life implemenation of the array.

7.2 Simulation Inputs

Four independent sinusoids are used as the inputs to the simulator. Each of these sinusoids has its own frequency and direction of incidence. The user of the simulator can modify these signals while the VI is running by simply rotating the knobs or entering new values. To cut down on processing time, we decided to bandlimit the frequencies of the sinusoids from 800 to 1600 Hz. There are other controls in the VI that allow the user to change this range, but we found that the simulations runs most smoothly at this range. With four signals the simulator can be used to examine the major combinations of sinusoids that should be tested....different frequencies coming from different directions, similar frequencies coming from different directions, similar frequencies coming from the same direction, and varied frequencies coming from the same direction. These four major categories can be used to show that the simulator does indeed work. The figure below shows how the paramaters of the sinusoids are input in the VI.

[1]This content is available online at <http://cnx.org/content/m12568/1.3/>.

Input Signal Selection



Frequency1	Frequency 2	Frequency 3	Frequency 4
800	1050	1300	1550

Angle 1	Angle 2	Angle 3	Angle 4
-70	-25	25	70

Figure 7.1: Input Signal Selection

7.3 Simulation Front Panel

Below is a copy of the front panel of the simulation VI. The "Angle to Listen To" knob is where the user selects the direction that they want to listen to. The graph on top shows the magnitude of the various frequency components from the angle that they are looking at. If a user wants to know from which direction a certain frequency compnent is propogating (or if there are two signals, the one witht he greatest amplitude), they can enter that frequency on the "Desired Frequency Knob". The direction that the frequency of interest is coming from will then be displayed on the "Angle of Incidence" Meter. Overall, this Front panel alllows the user to listen in different directions and determine the direction of an incoming frequency.

Example 7.1: Simple Example Using of Simulation
On the "Front Panel" figure below, we can see that the user is listening to signals coming from thirty degrees. We can then deduce from the graph that a signal at 1300Hz is propograting from thirty degrees. Based on the inputs above, this is exactly what we expect to see. We can also tell from looking at the "Angle of Incidence" meter that a signal at 1050Hz is comign from rougly -25 degrees. Again, this makes sense based on the above inputs.

FRONT PANEL

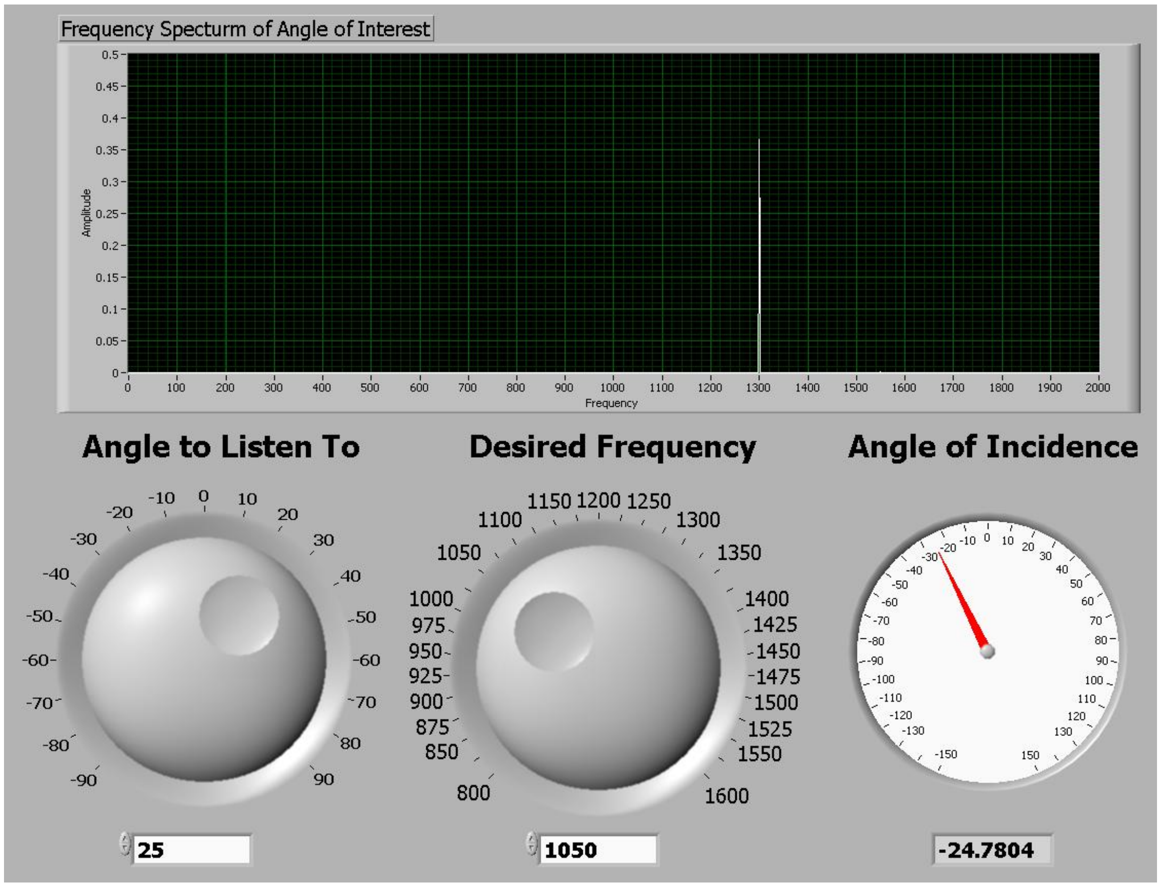


Figure 7.2: FRONT_PANEL

7.4 Simulation Output

The final part of this Simulation VI is the ouptut. The VI will display the incoming signal coming from the desired direction on the "Ouput Waveform" Graph. In addition to to this graph, the Simulation will also play this output waveform as audio. Doing so allows the user to hear the diffent sounds as they change their angle of interest.

OUTPUT WAVEFORM

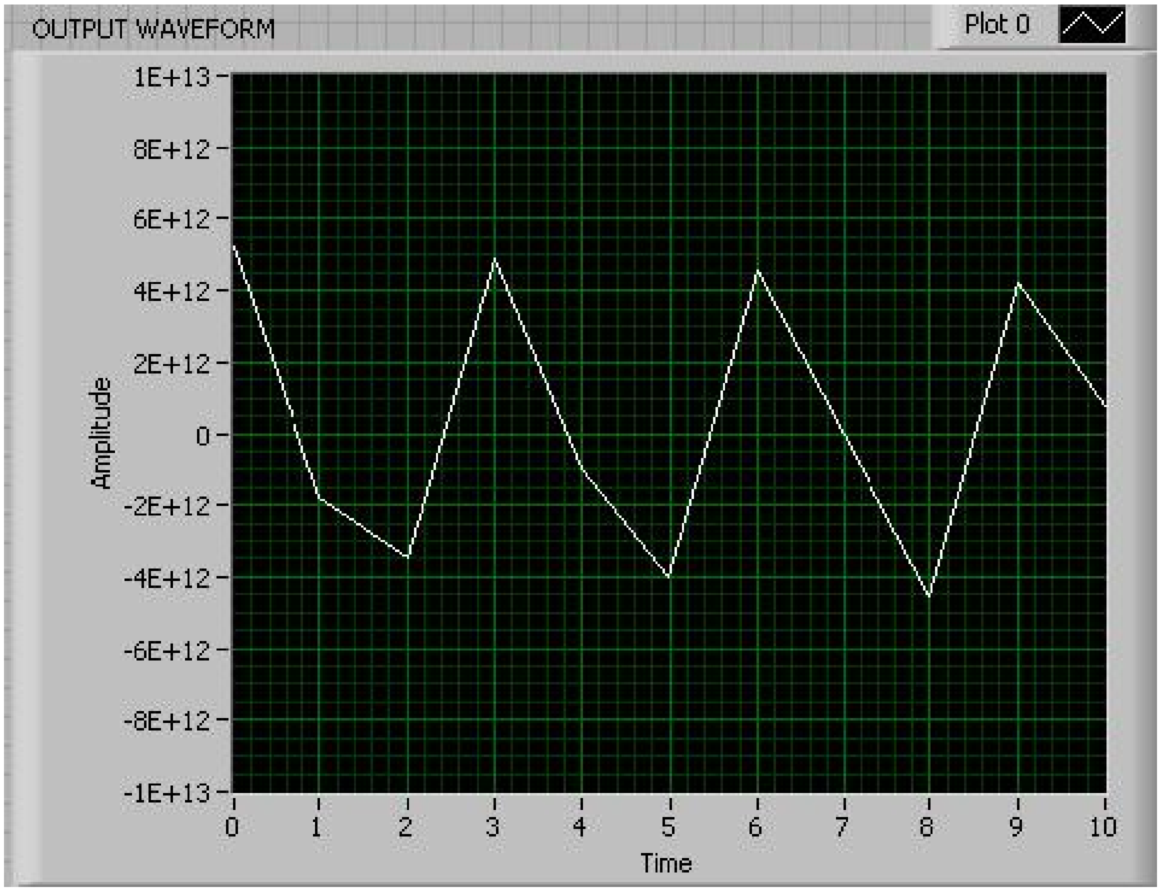


Figure 7.3: OUTPUT WAVEFORM

Chapter 8

Hardware[1]

8.1 Hardware

Our array was built using six Sony F-V100 omnidirectional microphones, spaced at 9.9 cm apart. We built a frame out of precisely cut 1" I.D. PVC pipe in order to keep the microphones spaced accurately apart and minimize phase error in our measurements. These microphones produced an average 1 mV p-p signal from the sinusoids that were used for the test program. The signals were fed into a six-channel amplifier to increase the voltage to .5 V p-p in order to achieve a usable range of the DAQ card. The amplifier was built using common 741 op-amps and care was taken to insure all the path lengths were approximately the same length. Following the amplifier, the signals were fed into a National Instruments 6024e DAQ card. The DAQ card was set to sample at 4000 Hz for a length of 2 seconds.

8.1.1 Test Setup

Once the signals were digitized, we were able to use the Labview code that we had developed using the simulated test signals and apply the same algorithm to the real life signals that we were recording. To generate test signals we used a second laptop to generate different frequency sinusoids on the left and right channel of the sound output and then used a set of speakers with extended cables to place them at different locations around the array. For the actual location of the tests, we set up the array in the middle of an open outdoor area in order to avoid sounds echoing off the walls of an enclosed space and causing unwanted interference.

[1]This content is available online at <http://cnx.org/content/m12569/1.3/>.

Chapter 9

Limitations to Delay and Sum Beamformers[1]

9.1 Implementing a Delay and Sum Beamformer

As we discovered in the section discussing beamformers[2] , that when an array of sensors record a signal there is an implicit delay between the signal arriving at the different sensors because the signal has a finite velocity and the sensors are not located at the same location in space. We can use this to our advantage, by exploiting the fact that the delay among the sensors will be different depending on which direction the signal is coming from, and tuning our array to "look" in a specific direction. This process is know as beamforming. The traditional way of beamforming is to calculate how much delay there will be among the sensors for sound coming from a direction that you are interested in. Once you know this delay you can delay all the corresponding channels the correct amount and add the signals from all the channels. In this way, you will constructively reinforce the signal that you are interested in, while signals from other directions will be out of phase and will not be reinforced.

[1]This content is available online at <http://cnx.org/content/m12570/1.4/>.
[2]http://cnx.rice.edu/content/m12563/latest/

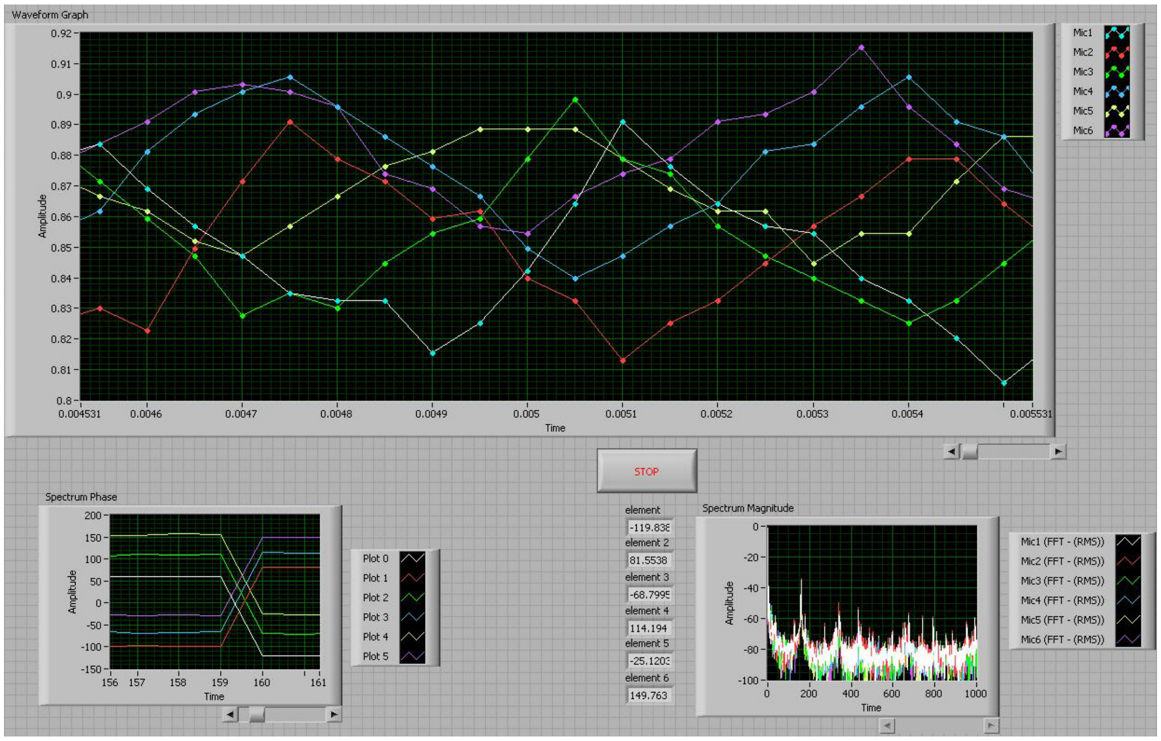


Figure 9.1

The figure above illustrates a sinusoid captured on a six channel linear array. Though, the sinusoids look crude because of the implicit noise of real signals, you can see by the spectrum that it is indeed there and in also that the phase is different for each of the sensors. The phase difference is determined by the delay between the sensors and is given by a simple geometrical calculation which we discuss later.

The problem with this method is that the degree of resolution that you can distinguish is determined by the sampling rate of your data, because you can not resolve delay differences less than your sampling rate. For example if the sampling period is 3 milliseconds, then you would have a range of say 20 degrees where the delay would be less than 3 milliseconds, and thus they would all appear to be coming from the same direction because digitizing signals from anywhere in this range would result in the same signal.

This is very significant because the spacing of the sensors is usually quite small, on the order of centimeters. At the average rate of sound, 330 m/s, it only takes .3 milliseconds to move 10 cm. However, the Nyquist Sampling Theorem states that we can derive all the information about a signal by sampling at only twice the highest frequency contained in the signal. With a 10 cm spacing between sensors the highest sinusoid we can capture is 1600 Hz, for reasons we discuss elsewhere. So ,we should be able determine the phase of all the sinusoids by only sampling at 3200 Hz rather than at tens of kilohertz that is required with delay and sum beamforming. In order to do this, however, we must implement a better method of beamforming in the frequency domain.

Chapter 10

Results[1]

The tests of the real-world implementation were successful. Using the second laptop, we positioned a 1000 Hz signal at about 40 degrees and a 1600Hz signal at about -25 degrees relative to the axis perpendicular to our array. Both signals were on simultaneously and at equal volume. As you can see from the spectrum of the output of our program, changing the dial to tune to the array to different directions results in the expected behavior.

Tuned towards 40 degrees

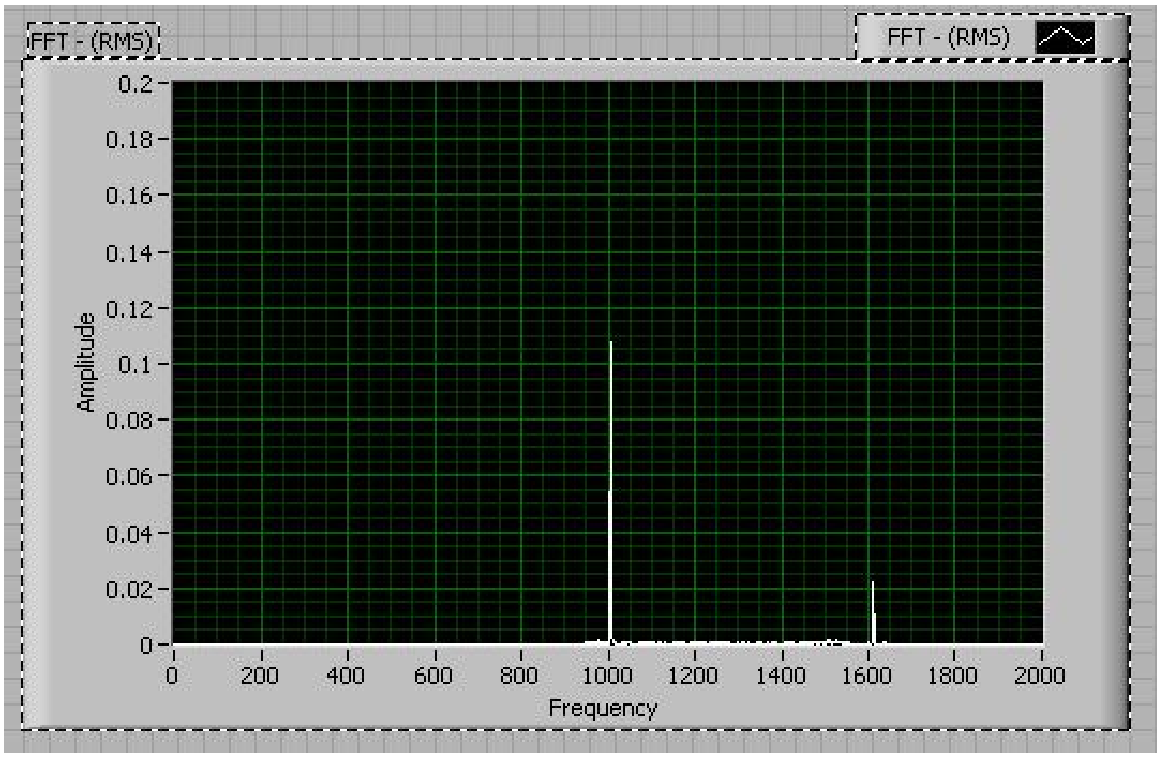


Figure 10.1

[1]This content is available online at <http://cnx.org/content/m12571/1.4/>.

Tuned towards -25 degrees

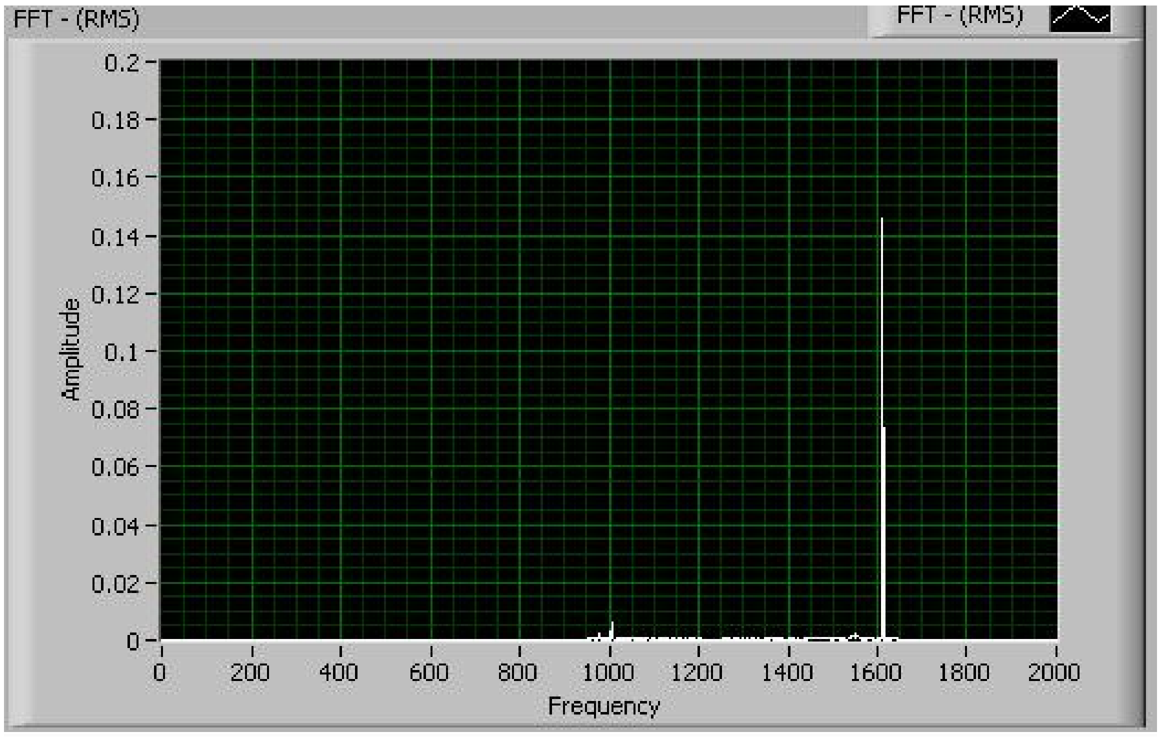


Figure 10.2

These first two figures show that our output signal consists of the two test sinusoids. Tuning the software to look in the appropriate directions shows that the magnitude of the corresponding sinusoid is more powerful than that of the power of the other sinusoid. Focusing on the 1000 Hz sinusoid enhances the power of that sinusoid to about 5 times that of the other sinusoid. Focusing on the 1600 Hz sinusoid gives even better results. The difference in power here is more than 10 times.

Tuned straight ahead

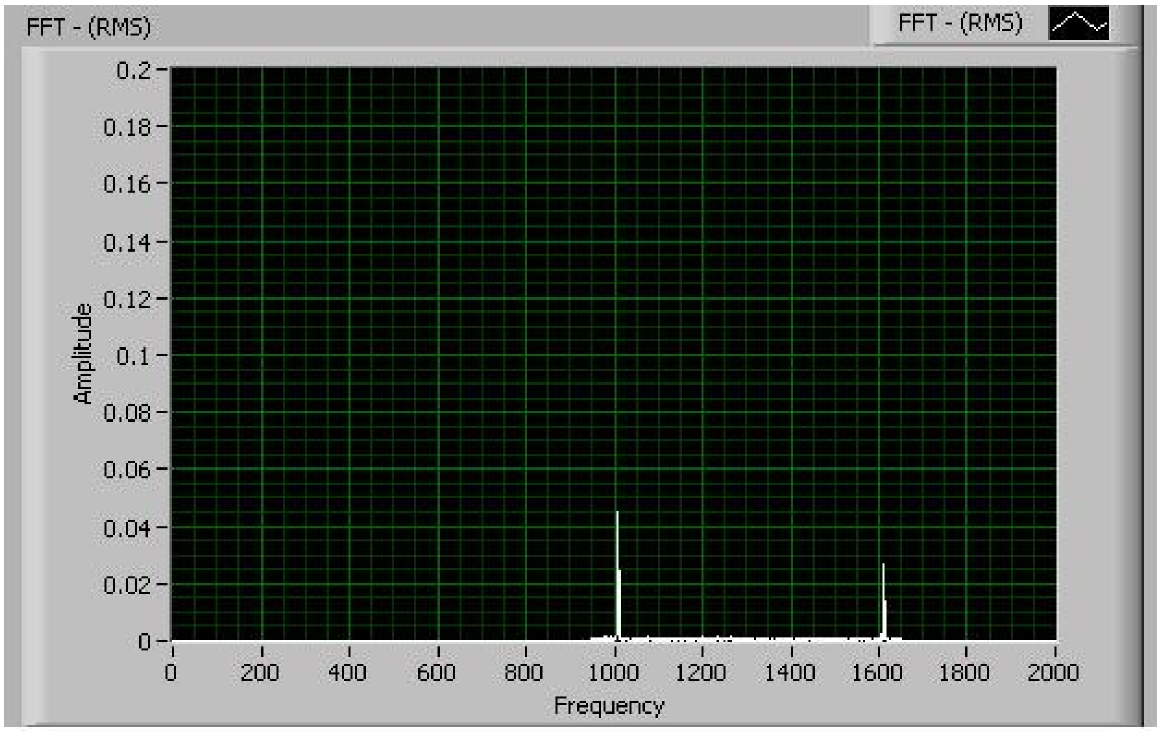


Figure 10.3

Tuned towards -83 degrees

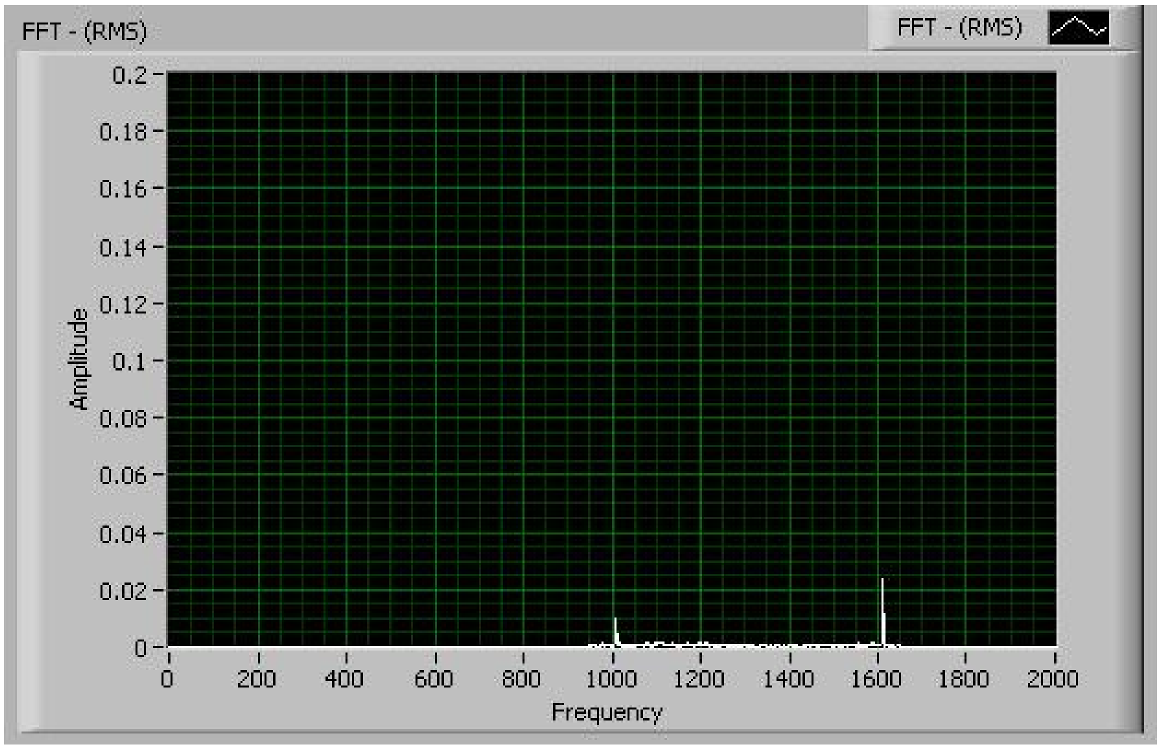


Figure 10.4

When we tune the array to a direction which does not have a source we get scaled down versions of anything close to that angle. For example, when we steer it at the middle we get small versions of the two sinusoids, and when we steer the beam at a direction that's way off, we get much smaller peaks from our two sinusoids.

10.1 Conclusion

We were able to demonstrate this processing technique in both simulation and real life. The main problem with our system is the amount of processing that is needed to make this a realtime process. For example, using a 1.6GHz processor we were capturing 1 second of data and taking about 2 seconds to process it. Due to this restriction in processing power, we are only processing a band of spectrum from 800 - 1600 Hz. This is not enough to process voice data. Another problem is that we have to space the sensors closer together in order to sample a higher frequencies because we have to avoid spatial aliasing in addition to temporal aliasing. Therefore the main improvements to this system would be ways to decrease the amount of processing or to use a much more powerful system. Although we have hammered out a good chunk of this project, there is definitely room for improvement in the optimization of processing.

Chapter 11

The Team[1]

11.1 Group Members

The group work was distributed to follow the strengths of the individual members. Jim Finnigan was responsible for most of the LabVIEW code, while Clay McPheeters and Jeremy Bass worked on a couple of supplementary VIs. Ed Rodriguez built the actual hardware microphone array and the filters for the microphone signals going into the DAQ. In addition, we all got together and tested the hardware and software to make sure it worked. Jeremy and Clay created the poster for the presentation, while all members of the group worked on the Connexions modules. We all spent many hours and exchanged uncountable blank stares trying to understand array theory. Many thanks to Dr. Don H. Johnson for helping us along the way. Dr. Richard Baraniuk also gave us helpful ideas and LabVIEW, which we could never have afforded.

[1]This content is available online at <http://cnx.org/content/m12566/1.4/>.

Jim Finnigan (finnigan@rice.edu)

Figure 11.1

Ed Rodriguez (edrod@rice.edu)

Figure 11.2

Clay McPheeters (claym@rice.edu)

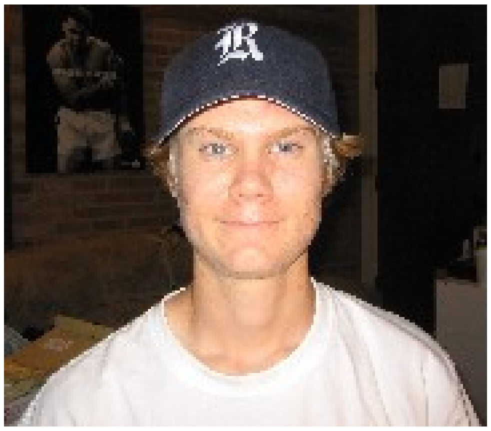


Figure 11.3

Jeremy Bass (jbass1@rice.edu)

Figure 11.4

11.2 Acknowledgements

Dr. Don H. Johnson for consultation & book: Array Signal Processing: Concepts & Techniques

Dr. Richard Baraniuk for LabVIEW and direction

Varma, Krishnaraj: "Time-Delay-Estimate Based Direction-of-Arrival Estimation for Speech in Reverberant Environments"

Glossary

F **Far-field source**

A source is considered to be in the far-field if $r > \frac{2L^2}{\lambda}$, where r is the distance from the source to the array, L is the length of the array, and λ is the wavelength of the arriving wave.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Array Signal Processing
This is our ELEC 301 Project for the Fall 2004 semester. We implemented a uniform linear array of six microphones. We then sampled the data and analyzed it in LabVIEW in order to to listen in one direction. We also explored listening for a particular frequency and its direction.