18-551 Fall 2007

Group 4

Albert Lin

Jonathan Cornell

William R. Wong

Contents

INTRODUCTION

This report consists of a summary and analysis of the culminating project resulting from a semester long digital signal processing design course offered at Carnegie Mellon University. In the following in depth analysis, we will examine the theory, implementation and results of our attempt at acoustical source location. We will also heavily detail the problems encountered as part of our development as well as suggestions for future work that can be adapted from our end results.

THE PROBLEM

The original problem which began our exploration into acoustical source location was the initial desire to process acoustical waves using digital signal processing techniques. We examined previous work in this area and decided to reanalyze a successful project with the hope of creating additional improvements in efficiency and accuracy. We studied the results of Group 1 from Spring 2004's Microphone Arrays for Source Location Applications (MALSA) analysis and devised an outline to improve upon their shortcomings. We took the concept of collecting acoustical wave data from an array of microphones and expanded our proposal to include additional channels of source information. We hoped that by being able to track a noise source with additional degrees of information that we would be able to improve upon the previous analysis.

Based on our proposed improvements, we were able to associate a real-world problem that could use our developments to solve a problem. Imagine a quiet mountain just after an avalanche. After a headcount of the number of skiers, the ski rescue team realizes that they have lost a skier in the avalanche. They have reports from surviving skiers that the missing individual was last seen in a specific corner of the mountain. If the rescue team had the ability to locate the missing skier's cries for help, they would be able to pinpoint his/her exact location and could therefore reduce the amount of time necessary to conduct a search. By implementing a two dimensional array of microphones, we could improve upon the source location resolution to reduce the error in determining the location of the missing skier.

PRIOR 551 WORKS AND COMPARISONS

In the realm of microphone arrays and sound localization, there is really only one project that we found to be relevant, which was the MASLA project from Group 1, Spring 2004. Essentially, the MASLA project used the same concept and algorithm of Cross Power Spectrum Phase, except that they planned on only using two pairs of microphones in a linear array, while we planned to have four pairs of microphones in a T-shaped array. Our physical spacing was also different, where MASLA used 40cm spacing between array elements, we used 10cm spacing. We did this in order to compensate for our intentions of increasing the sampling rate and to increase the spatial accuracy we could achieve for the sound localization.

MASLA limited themselves to just a 8KHz sampling rate, claimed to be a constraint of the EVM hardware at the time, while we were able to expand out to planned 8, 24-bit, 32KHz sound recordings, thanks to the improved DSK hardware as well as the acquired M-Audio Delta1010 and PCI card.

Due to just using 2 microphone pairs, the MASLA project was only able to calculate two angles and a distance for the sound source they analyzed. However, with our 4 microphone pairs, we were theoretically able to double the accuracy of their implementation by producing twice the amount of information.

ELECTROACOUSTIC FUNDAMENTALS

When first modeling the problem of sound localization using microphone arrays, we had to analyze the geometry considerations of our implementation. In order to determine frequency and sampling rates for our acoustic setup, we have to consider the physical geometries of the microphones in our T-array.

We planned on pairing up 8 microphones to have **4 pairs** of microphones. The T is comprised of a horizontal linear array and a vertical linear array. The microphones in each pair are spaced apart **10cm**. The spacing between each *pair* in each linear array is 10cm as well.

According to our spacing and pairing decisions, we would put two pairs horizontally and two pairs vertically. The vertical pairs will be place directly centered, orthogonal, and 5cm away from the horizontal array. See the below figure:

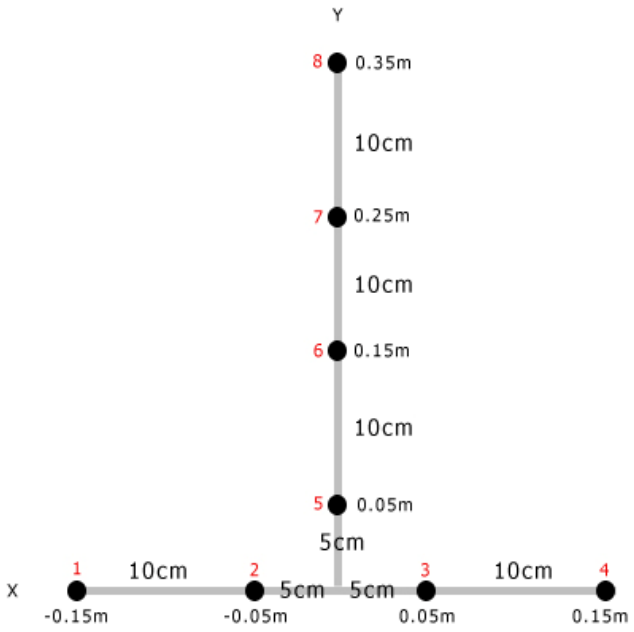


Figure 1 – Spacing and Element Pairs: 1&2, 3&4, 5&6, 7&8

Pictures of our array structure hardware are below:

Figure 2 – Close up of microphone clamp

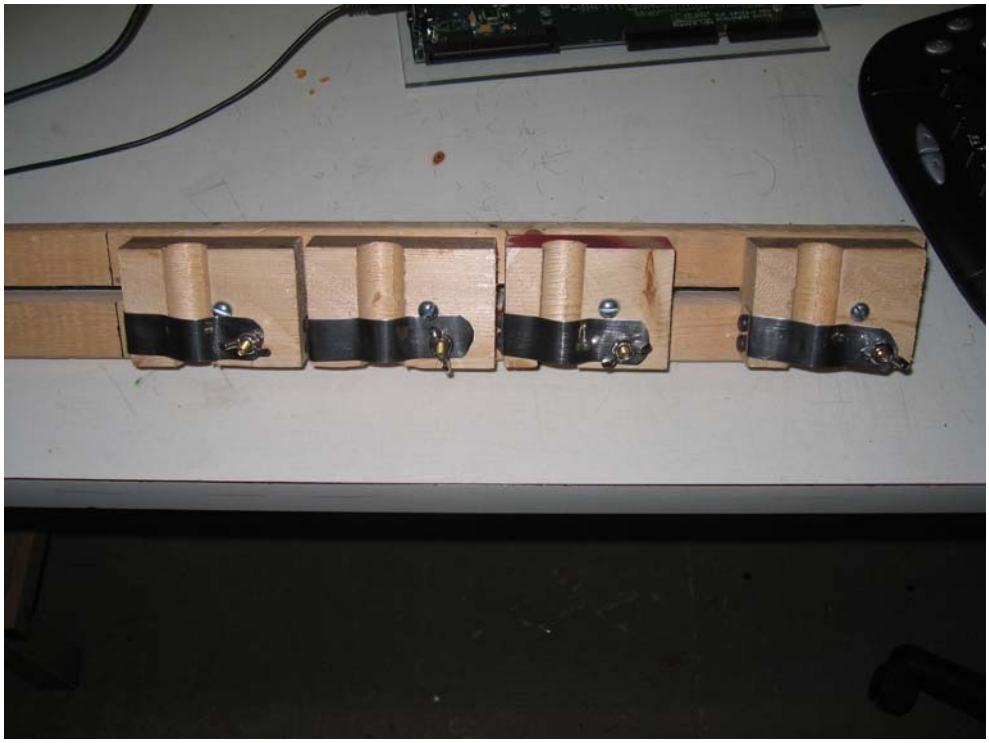


Figure 3 – Four microphone clamps

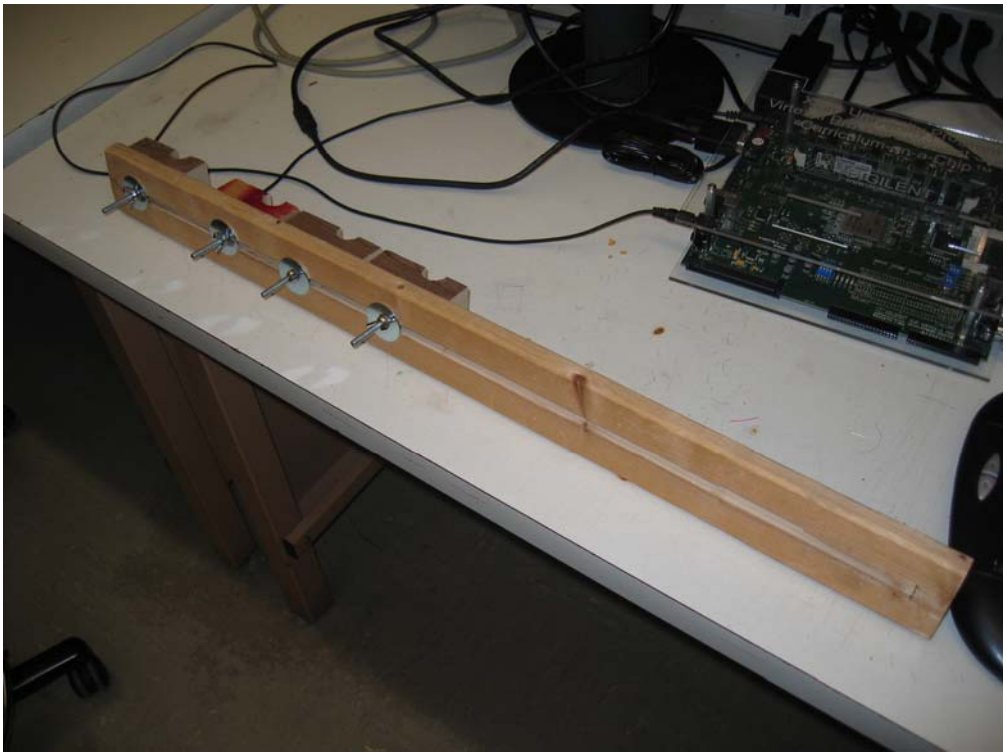


Figure 4 – Arm of microphone array

We chose these spacings with some constraints in mind:

1 – We need to accommodate the 18-551 lab room

2 – We wanted to have enough delay present between microphone array elements in each pair so that we could more accurately calculate and position a sound source

3 – The amount of delay we experience (in seconds) within a microphone pair if the acoustic signal approaches the elements on axis is then:
Delay(seconds) = (mic spacing) / (velocity of sound) , where velocity corresponds to the speed of sound = 344meters/sec at room temperature.
D = 0.1 m / 344 m/s = **0.000291** seconds

4 – We want to use a sampling frequency of 32kHz (as an improvement upon the previous 551 project)
If we have a D = 0.3 milliseconds, then the number of samples the delay would be:
D*Fs = number of samples
0.000291sec * 32000 samples/sec = 9.312 samples, or about **10** samples of delay

5 – Since we decided on **32kHz** for the sampling frequency, we need to have enough samples (delay) present to resolve in the algorithm. We want to analyze low enough frequencies to have the half wavelength (λ/2) be greater than the

spacing between our microphone elements in a pair. This is related to not wanting spatial aliasing and also considering the Nyquist frequency. The spacing relation (d) between microphone elements should then be:

$d < \lambda/2$

Due to our spacing of 10cm for the elements in a pair, we can only resolve acoustic signals up to **1.5kHz**. In deciding on addressing only audio frequencies of 1.5kHz and below, we can establish some variables and relations:

We know empirically: λ = (velocity of sound) / (frequency of audio signal)
Therefore, the maximum seen wavelength λ = (344 m/s) / (1500 cycles/s) = **0.2293m**
This satisfies the above spatial relation: 10cm < 11.4cm

In order to make calculations simpler (and more time efficient), we wanted to model far-field planar waves (rather than near-field spherical waves) that our microphone would be receiving. This is the same assumption that M. Omologo and P. Svaizer had used when developing and demonstrating their Crosspower Spectrum Phase algorithm. Thus, according to Beranek, there are two acoustical considerations to meet far field conditions:

1- $r >> L_a$ (distance from the array must be much greater than array length) , where "r" is distance from the microphone element

   Our array lengths (both horizontal and vertical) are 3*(10cm) = 30cm.
   Thus:
   r >> 30cm from array (>> = 3-10x) to be considered far field.

   "In acoustics, the size factor indicated is usually taken to be larger than 3 to 10"
   At low frequencies, assume 3-4x, which results in r> 90cm or r>1.2m.

2- $k^2r^2>>1$ or $r >> \text{sqrt}(1/k^2)$ where $k = 2\pi/\lambda$ and λ = .23m

   This results in r > 3.66cm

We take the greater constraint and assume that a distance **r > 1.2m** from the microphone elements will allow us to consider the acoustic inputs to be far-field, planar waves.

<u>Overall, we are going to be using:</u>
Sampling frequency: 32 kHz
Sound source frequencies of interest: Human speech within the 200Hz to 1600Hz range
To be considered far field, the sound source should be: 1.2m away

GEOMETRIES

For geometrical reasoning, we have the following to calculate delay between two elements in a microphone pair if assuming a planar wave:

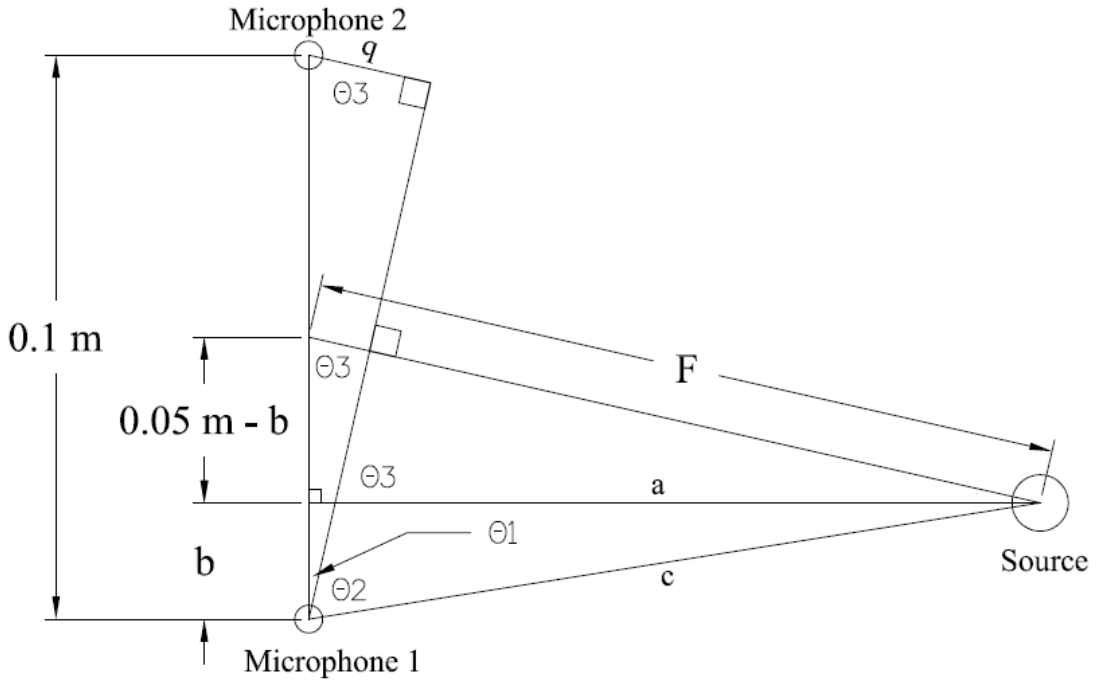


Figure 5 – The sound source on the far right is some distance "c" from microphone 1. Using the law of cosines and law of sines, we can derive the variable "q", which is the distance between the planar wave and microphone 2. Thus, "q" is the spatial delay between microphone 1 and microphone 2.

1. $a^2 + (0.05\text{-}b)^2 = F^2$
2. $\cos(\theta_3) = [(0.05\text{-}b)^2 + F^2 - a^2] / [2(a^2 + (0.05\text{-}b)^2)(0.05\text{-}b)]$
3. $\sin(\theta_1) / q = \sin(90) / 0.1$

We can now get the relation for "q" through trigonometry:

$q = 0.1 [\sin(90 – \arccos \{ [(2(0.05\text{-}b)^2)] / [(2(a^2) + 2(0.05\text{-}b)^2)) (0.05\text{-}b)] \}$

We can now use this relation for "q" to know the spatial delay and thus calculate the time delay between array elements from a planar wave using this relation and dividing by the speed of sound.

Since we now know the spatial delay, it is possible to relate a given spatial delay with an angle for which a sound source would be located using trigonometry:

1. $\theta_1 = \arcsin (q * \sin(90)/0.1)$

2. Resolvable angle = arcos(q/0.1)

The following table shows calculated resolvable angles for where a sound source could be:

Samples Delay	Delay (sec)	Delay (m)	θ_1 (deg)	Resolvable Angle (deg)
0	0	0	0	**90**
1	3.13E-05	1.08E-02	6.171221	**83.82877852**
2	6.25E-05	2.15E-02	12.41553	**77.5844727**
3	9.38E-05	3.23E-02	18.81418	**71.18581815**
4	1.25E-04	4.30E-02	25.46756	**64.53243986**
5	1.56E-04	5.38E-02	32.51361	**57.48638537**
6	1.88E-04	6.45E-02	40.16568	**49.83432453**
7	2.19E-04	7.53E-02	48.8074	**41.19259785**
8	2.50E-04	8.60E-02	59.31658	**30.68341711**
9	2.81E-04	9.68E-02	75.35253	**14.64746931**
10	3.13E-04	1.08E-01	90	**0**

Table 1 – Calculated resolvable angles from possible sample delays

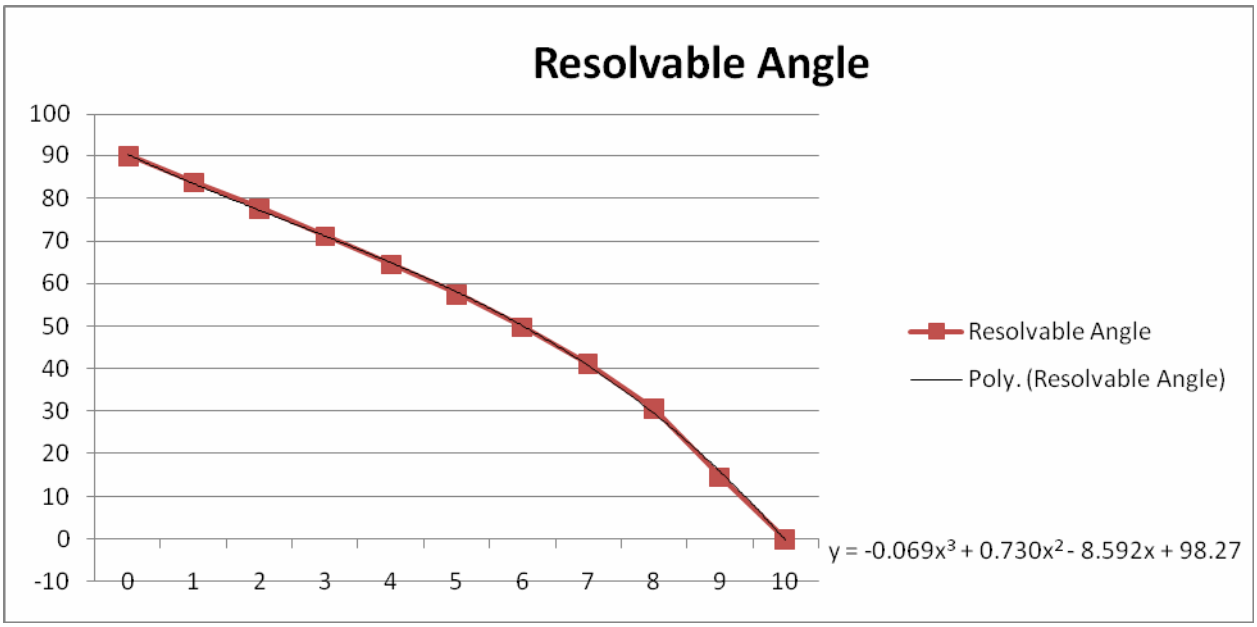


Figure 6 – Graph of resolvable angles showing non-linearity

As evident from the graphic, the relationship between the resolvable angles and the spatial delay is not a completely linear one, but actually can be well approximated with a third-order polynomial (as shown).

This is given just a pair of microphones, and if the other microphone pairs are accounted for, then it is possible to estimate a distance as to how far the sound source would be. This technique is the basis of beam forming, which we had considered but could not confidently implement within the given scope and time constraints of the project.

Since we now know the angles to the detected sound source, we may use further geometry to calculate an estimated distance to the sound source. The resolved angles in the following example are θ_1 and θ_2 from the following figure:

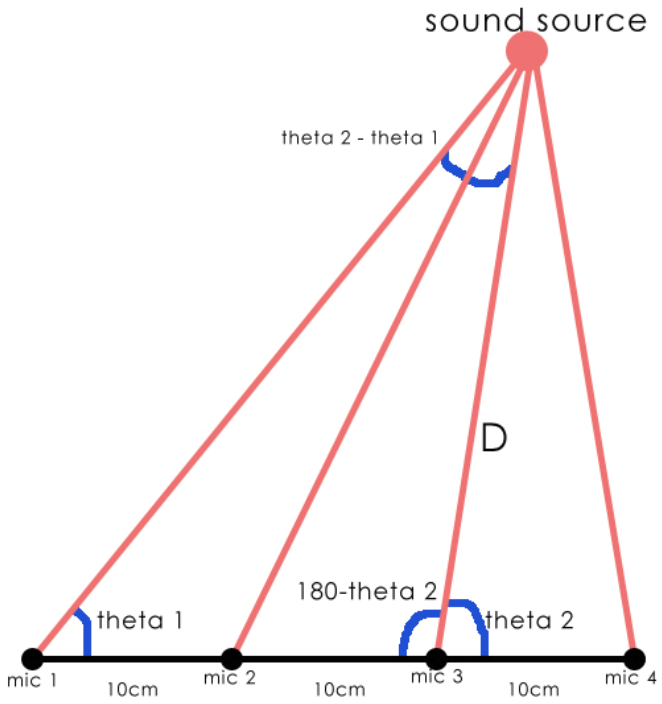


Figure 7 – Angles to compute distances D for sound localization

Using the Law of Sines for calculation, we know that:

1. $\text{Sin}(\theta_2 - \theta_1) / 0.2 = \text{Sin}(\theta_1) / D$
2. $D = 0.2 * \text{Sin}(\theta_1) / \text{Sin}(\theta_2 - \theta_1)$

From this, we can see that from pure geometry, we are able to provide basic sound localization estimates.

ALGORITHM

Array processing is not a new concept in the realm of acoustical source location. The concept of array processing has been discussed in numerous contexts some of

which include underwater acoustical imaging[1], speech recognition improvement[2] as well as radio frequency performance improvements.[3] In most implementations of microphone array processing, the additional microphones are used in obtaining additional information based on knowledge of the location of the additional elements. For our implementation, we will be applying the array concept to derive delay information from acoustical waves to aid us in calculating the approximate location of a sound source. As mentioned previously, the basis for most of our research has been the MASLA project from Spring 2004. We took the concept of setting the spacing between elements as a constant and deriving an approximation for delay of number of samples from a correlation of phase between pairs of microphones.[4] Applying this analysis for each pair of microphones, we propose that it is possible to minimize the realized error at the output of our final calculations such that we can approximate the source location to within 5 centimeters.

The basis for algorithm relied heavily upon the Cross Power Spectrum Power Phase (CPSP) algorithm developed by Omologo and Svaizer[5], which is a time delay of arrival (TDOA) technique. The MASLA group used this technique as well and achieved somewhat acceptable results.

After browsing through literature surrounding sound localization techniques, we determined that CPSP was our algorithm of choice, not just for its simplicity but also for its accuracy. The reason that a straightforward correlation of the sound information is not completely reliable and useful in our project is because a standard correlation includes information about the magnitude of a sound, as well as its phase information, and the main distinguishing factor for sound localization is phase difference. Thus, a standard correlation incorporates superfluous and redundant information in an algorithm and thereby dilutes the results and decreases resulting accuracy. The CPSP algorithm normalizes the Fourier Transform analysis and thus is able to focus more on the phase difference information and provide more accurate results for sound localization.

The basic detailing of CPSP is as follows:

---

[1] *Array Processing of Underwater Acoustic Sensors Using Weighted Fourier Integral Method I.* S. *D. Solomon and A. J. Knight,* Defense Science and Technology Organization, IEEE 2000 (http://ieeexplore.ieee.org/iel5/6991/18842/00870218.pdf)

[2] Thomas M. Sullivan, *Multi-Microphone Correlation-Based Processing for Robust Automatic Speech Recognition* Ph.D. Thesis, ECE Department, CMU, August 1996.

[3] Power Control with Antenna Array Processing for UMTS (http://tte.ele.tue.nl/radio/publications/ECR%20pubs%202004/Jevrosimovic%20VTC04Fall%20Power%20Control%20with%20Antenna%20Array%20Proc.pdf)

[4,5] Omologo, M.; Svaizer, P., Speech and Audio Processing, IEEE Transactions on, Vol.5, Iss.3, May 1997, Pages:288-292

1. Compute the Fourier transforms of each sound source signal acquired for analysis.
2. Compute the conjugate Fourier transform of the second source signal acquired.
3. Follow the equation below to obtain a delay in the number of samples:

$$CPSP = IDFT\left(\frac{\mathbf{X}_1\mathbf{X}_2^*}{|\mathbf{X}_1||\mathbf{X}_2|}\right)$$

4. Once the CPSP output is found (delay in number of samples, sigma), the angle of the observed sound source is:

> the local wavefront arrival direction (indicating the angle between normal of wavefront and x-axis) is derived as
>
> $$\hat{\theta}_i = \arccos\left(\frac{c\hat{\delta}_i}{d_i}\right) \quad (2)$$
>
> where c is the speed of sound and d_i is the distance between the two microphones.

In our case, we took the speed of sound at sea level and at room temperature, which gave us an approximate value of 344 meters/second. Our distance between the two microphones was set at 10 centimeters. From this, we obtain the sound direction angle and can subsequently obtain the distances to the sound source from the microphone pairs as discusses in the geometries section.

In total, we calculate three FFTs and one IFFT per microphone pair per frame of processing. Thus, we have twelve FFTs and four IFFTs total (accounting for all four microphone pairs) for each frame of processing. For every processed frame, the main component of our algorithm is the FFT and IFFT operation.

DATAFLOW

Our dataflow can be summarized by the following figure:

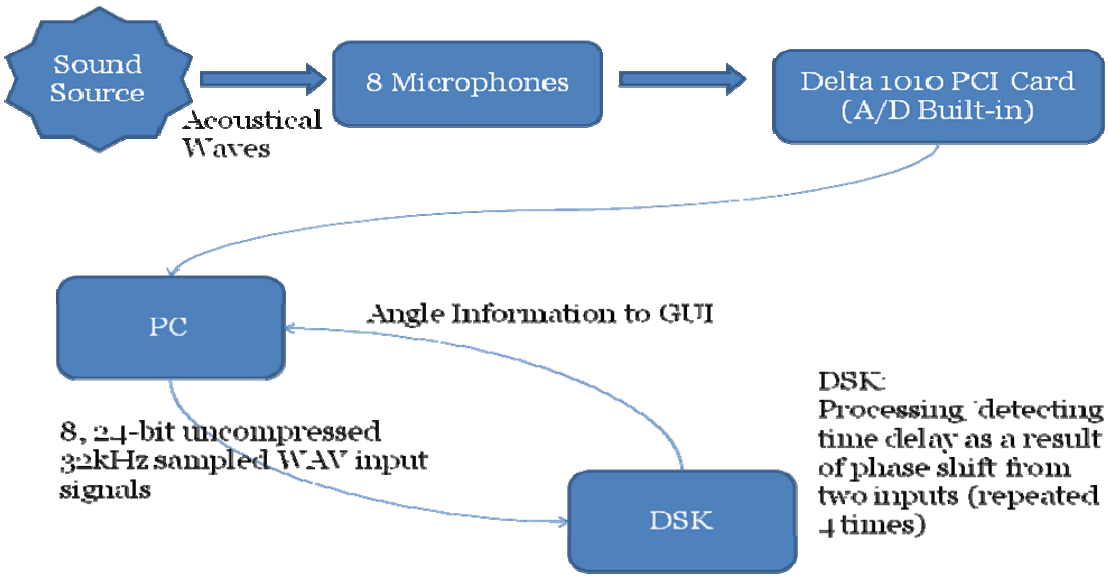


Figure 8 – Dataflow for project

We had a sound source, which was someone talking or making noises and moving across the lab room. Due to our algorithm frame size and setup, we required at least 6.4 seconds of sound to be recorded, so generally we took in about 7 to 10 seconds of total sound source recording. Because we did not have our hardware operating optimally by the time of our demo, we were left to only 2 microphones for our implementation. Thus, the two microphones were laid down on a wooden rack and held in place with metal clips. The microphones then went to a preamp that would boost the captured signal. The preamp was plugged into the M-Audio Delta1010 rack, which then sent the recorded signals to the Delta1010 PCI card located within the PC. At this point, no computation has occurred, only translation of the sound source signal from analog to digital.

On the PC, we used an open-source recording software called Audacity to save our sound source signals as .WAV files. Audacity was able to capture each microphone channel separately and could export them as separate sound files and thus facilitated our processing. We recorded the sound files as 24-bit, 32 KHz WAVs. Because of this, our demonstration could be considered simulated real-time. We noticed after playback and analysis of the raw sound recording, we were experiencing much more noise than anticipated. We knew that there was also an inherent DC offset present in the sound recordings, and so we needed to minimize the noise and extraneous data as much as possible through filtering.

We decided to use two Butterworth filters implemented in MATLAB as part of our preprocessing phase. We first high pass the signals to remove any blatant DC offset and then use a low pass to try to remove any white noise. We did the filtering in MATLAB in the interest of time, though we had originally planned to incorporate it into our DSK code. If we had implemented the filters in C, we would have had to create a lookup

table for the filter coefficients and would have had to deal with issues of overlap-add in order to compute the correct, filtered result from the frame processing.

After filtering, we send the new, filtered sound files using Microsoft Visual C++ C code sockets to the DSK, which then performs the Cross Power Spectrum Phase algorithm in the respective C code through Code Composer and returns the angles information in a data file, which is then read back into MATLAB for the GUI. The GUI interprets the angle information and draws a line on an image representation of our microphone pair to show the sound localization angle to the user.

IMPLEMENTATION & PROBLEMS ENCOUNTERED

Our final implementation was based on only one microphone pair (two microphones) due to time and hardware constraints. We were able to get everything else working as originally planned in terms of code.

As part of our implementation it was necessary to develop software to provide sampled information to the DSK so as to make it possible to analyze the discrete samples of each input file. From the Delta 1010 PCI interface card, Windows XP drivers supplied by M-Audio provided a software interface to access eight simultaneous channels. We found it easier to collect and process the sound information that we retrieved from the Delta 1010 by using a freeware program instead of M-Audio's proprietary capture program. We obtained a copy of Audacity[6] which allowed us to interface directly with the Delta 1010 drivers to capture multiple channels. As discussed in our analysis of problems, we decreased the number of channels analyzed to two but found the ease of use that Audacity provided should remain part of our implementation. We also used Audacity to configure the D/A converter on the Delta 1010 and saved our configuration settings for sampling rate, quantization size and number of channels in a configuration file to simplify the process of repeating our analysis. Once we obtained the recorded waveforms in Audacity, the sampled data was exported to multiple wav files with each file corresponding to a specific channel number as described in the numbering convention figure. These files were exported to wav files and numbered as mentioned.

Once we had a separate wav files, it was necessary to send the data to the DSK for processing. Prior to processing the data, we wanted to filter the waveforms as much as possible to reduce the computation time on the DSK. We implemented a filtering function in the MATLAB code that provided our graphical user interface such that MATLAB would filter the signal to remove the DC offset present in the original wave file as well as the unnecessary components above 1500Hz that provided noise as a result of not being able to completely resolve frequencies above the physical acoustical threshold. Once the files were filtered, they were resaved to wav files using MATLAB's wavwrite() command. From here, we implemented PC side code in C that read in each

[6] http://audacity.sourceforge.net/

file consecutively and skipped the header information that gave details about the information in the file. This was accomplished by incrementing the file pointer to skip over the first 44 bytes of information and then reading in the sample data.[7] The sample data was sent to the DSK as part of the process outlined from our previous work in Lab 3. Once a request to send data is received by the PC, the Win32 Winsock library function net_send_ready() is used to send TCP data to the DSK to ensure proper delivery of all the sampled data. The C code on the PC then waits for a response packet from the DSK with the calculated information. From here, the received data is written to an out file that contains the angle information that is to be displayed by the graphical user interface.

The file read function consists of opening file pointers and reading in set blocks of data into allocated memory in an array set up for each file. The file pointers remain open and are incremented to skip the header information. Once opened, each time the DSK calls for a new block of data, the file pointer is incremented and new data is read in and then sent to the DSK for processing. Once done processing, the DSK sends the data back to the PC where a connection is listening for a response. Once that data is received, it is written to a buffer and then written to a file containing all of the angle information. The Graphical User Interface then opens this data file and reads in the information so as to display the calculated angle measurement graphically in the GUI. No additional processing is done on the PC side.

We initially decided on implementing eight channels of data in real time to increase the resolution of the previous group's project. Once we found out that the interfacing process to get data out from the PCI interface into a data stream that we could interpret to send to the DSK was very involved, we decided to modify our proposal to make the system simulate real time by making the input information available completely. By providing a file to the PC side code, the system emulated a real time implementation. Once we started developing code to read and send frames to the DSK, we realized that allocation of memory resources to dedicate buffers to each file would require a significant amount of storage off chip. We decided to use only the on chip memory in interest of preserving processing time because we were sending data in small blocks very frequently and wanted the data back as quick as possible to allow for later development in real time.

Once we decided on changing the code into a simulated real time we encountered the problem of dealing with a DC offset in our test data in MATLAB. When we implemented the CPSP algorithm in MATLAB with sample files with an artificially induced sample delays, we continuously saw a correlation peak at zero referring to a DC component. In theory, after removing the DC component in MATLAB our algorithm worked properly and successfully found the delay in terms of samples. However, when we ran our lab recorded sound files through the algorithm in MATLAB, we noticed that the noise level

[7] http://www.sonicspot.com/guide/wavefiles.html

was directly related to the accuracy of the outputs. We then decided to examine other sources of noise and found that if we high pass filtered any input data that we saw better accuracy results (see **APPENDIX E**). We then decided to implement a filter to remove the DC offset as well as to remove higher frequency components as shown in the frequency response figure in the appendix. This filtering step removed higher frequency components that could not be completely resolved because of the physical spacing and frequency resolution limitations. Frequencies higher than 1500Hz would only contribute to the increase in noise. Running this filtering step significantly improved the accuracy of the data as shown in the results.

Upon obtaining a properly functioning algorithm our next step was to implement our code on the DSK. After porting the functions over to C code, we encountered several problems with verifying the algorithm and implementation issues. Initially we were having problems with viewing the actual data using Code Composer's view memory functions. Because we allocated quite an amount of data for global variables, our data overwrote into other sections of memory that are not readily accessible by Code Composer. After much investigation, we finally realized that given the memory setup code that we had ported over from Lab 3, our global variables had written over into memory locations labeled as cache memory. Because of this, Code Composer would return zeros for any data that was there. However, the data was actually still there and printed out correctly using printf functions.

Another problem we were having was dealing with inputs of zero magnitude. In the CPSP algorithm, there is a division by the magnitude of the two input signals. When the code was ported from MATLAB to C, we needed to check for zero magnitude. In this case, we had decided to not perform the division (which would return NAN). After we had done this, our algorithm worked 100% on two perfectly identical delayed signals. However it did not perform as well with noise and inherently different physical microphone characteristics.

A minor problem we had experienced is through the use of data alignment. The FFT and IFFT both required a double word alignment. Initially we had problems with aligning data. But after much consultation with the teaching assistants, they had introduced us to the #pragma DATA_ALIGN compiler directives that simplified the process of data alignment.

MATLAB Code:

Our project involved modeling the algorithm through MATLAB, and that code and some results can be seen in **APPENDIX C**. Essentially, it takes in the recorded sound files (we modeled with just one microphone pair to make sure the concept was correct), filters them with the Butterworth filters, and performs the Cross Power Spectrum Phase analysis on it. Then it computes the resulting sound source angle from the derived time delay using trigonometry as previously described.

We used the same filtering techniques in our final implementation as part of a preprocessing stage. The GUI was implemented in MATLAB, and the code for it can be seen in **APPENDIX D**. Essentially, it was able to take in a file of angle data (not directly linked to C/C++) and compute and display the necessary calculations resulting from the Cross Power Spectrum Phase algorithm and geometry calculations. Then it would playback the filtered sound file and would draw a line according to the derived angle of arrival information for a microphone pair.

TIMING, MEMORY & RATES

When we were timing the amount of time that was taken to complete execution of the PC side code, we saw that it was necessary to leave at least one printf statement in our code otherwise the DSK would not successfully connect to the PC. We believe this to be due to relatively small amount of time necessary to process one frame of data (1024 samples). The amount of time allocated to the connection initialization process was extended enough by one printf call such that the program had enough time to complete the negotiation process to set up a connection. The results we retrieved with one printf are as follows:

For two channels of 9.856 second sound files, the total processing time from file opening to cleanup of the socket libraries was 28.8280 seconds. This equates to approximately 2.9249 seconds of analysis per one second of input data for two channels. Therefore, approximately 1.46 seconds to process one second of sound data. We realize this is including many printf statements that execute once for every block of sampled data and we also noticed subjectively that the actual execution time without the printf was almost ½ of the calculated time. We do not doubt the ability of the code to perform in real time but we were unable to properly measure time using the time function. However, we were able to profile the following using the sample 9.856 input files:

> One FFT: 32540 clock cycles Iterated: 400 times
>
> One IFFT: 32543 clock cycles Iterated: 200 times
>
> Complete algorithm (Inclusion of one twiddle factor generation): 2,316,021 clock cycles

In terms of optimization, we attempted to implement the cache-optimized FFT, but could not get the function working properly with our implementation. Thus, we needed to go to the radix-2 FFT provided in the TI libraries. We used code hoisting for writing our loops, which performed operations outside of the loop and also performed parallel operations of array computations. We turned on Code Composer's OptLevel to File optimization 3 and Program LevelOpt to optimization level 2.

For memory, we utilized a small frame size to try to make the transfers and computations inside the algorithm much faster and more efficient. Our PC/ DSK request methodology minimizes memory utilization for the entire process, as discussed above in the implementation section. We found that the amount of data necessary for

calculations fits on chip if it is sent in series, and that is part of the reasoning for our decision to send our data in series rather than in parallel. We changed to serial transfer as a result of memory caching issues, and the memory allocation is fixed at:

1024 samples * 32 bit float = 4 kilobytes/block analysis
Using 200 blocks for our buffer analysis = 200*4Kb = 800 kilobytes total

Also, our memory is cleared after each calculation in order to avoid any memory-full issues.

The transfer rates are derived from our timing calculations as follows:

- 24 bits * 32kHz = **~768 kilobits/sec** to DSK
- Using 96 KiloBytes/sec per microphone input
- Data will travel from PC to DSK over Network Interface Card via packet transfers
- 4 angles * 4 Bytes = 16 Bytes for each angle
- Actual results:
  - PC to DSK (average over all files): 105 kilobytes/sec
  - DSK to PC (average over all angle transfers): varies as a result of DSK requesting but average has been 518 Bytes/sec (16Bytes/ .032seconds + overhead).

These observed results matched our theoretical calculations almost exactly, probably as a result of the size of the data transfers remaining relatively the same. Our theoretical calculations did not account for transmission overhead which can explain the variations we observed.

Update speed was another rate that we considered in the design of our system. It was unnecessary to continuously process every frame based on the speed that a stranded skier would move would be close to zero but we wanted to show proof of concept that future work would be able to develop our implementation further with other applications. Since we did not provide a true real time analysis but instead a simulated analysis, we processed every frame of data. Theoretically, this value can be changed in our code to update an integer value of samples (i.e. every 20^{th} frame) to adjust to processor availability and desire to change update speed. Our processed data was made available to show the possibilities of our implementation but could have easily been scaled back to processing every 50^{th} frame. Again, this minimum value is highly dependent upon moving speed of the sound source as well as the desire of how often to update information. In our final demo, we processed every frame and displayed the calculated results to show what the system is capable of. The update speed was simulated at approximately 1 update per second as previously discussed as a result of debugging printf statements in our code (see timing discussion).

MATLAB MODEL RESULTS

Our initial MATLAB modeling of the algorithm showed how our delay theory was correct. When we delayed a stock .WAV sound file by a set number of samples (in this

example, 10), we saw that the algorithm was able to detect that 10-sample delay quite well:

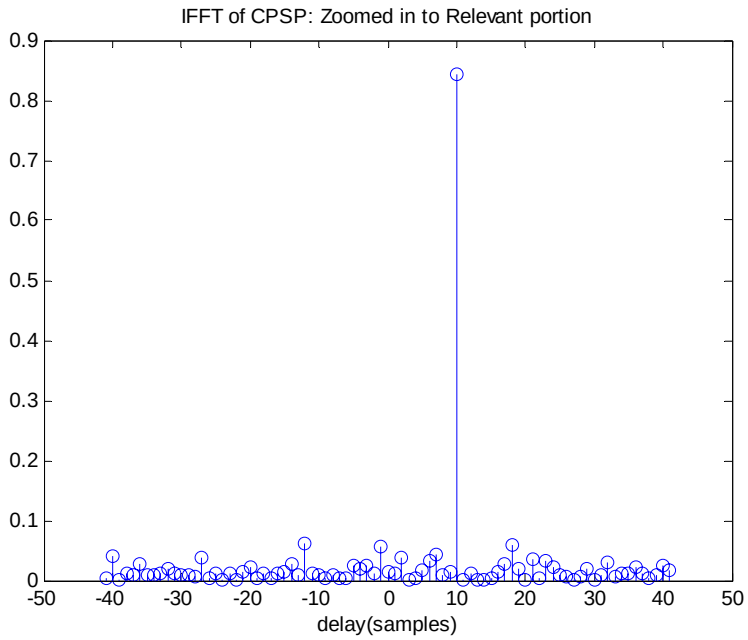


Figure 9 – MATLAB graph showing correct delay detection for contrived situation

We also found in our MATLAB model that there were cases where the results were not obvious and had some quirks about them. Waveform 1 and 2 are actual recorded signals from two microphones from our testing phase, of someone saying "Help, help! (pause) I'm lost on a mountain!"

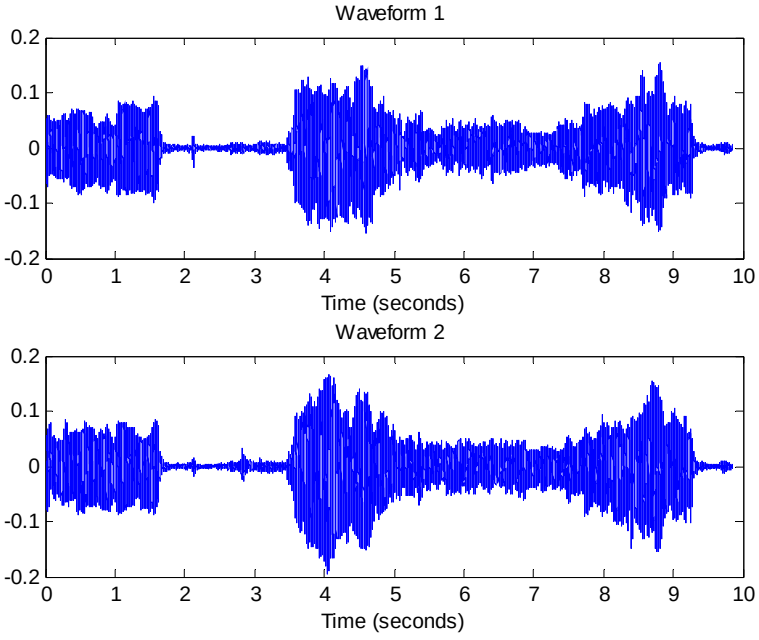


Figure 10 – Waveforms of input sound signals. 2 is 3-sample delayed version of 1.

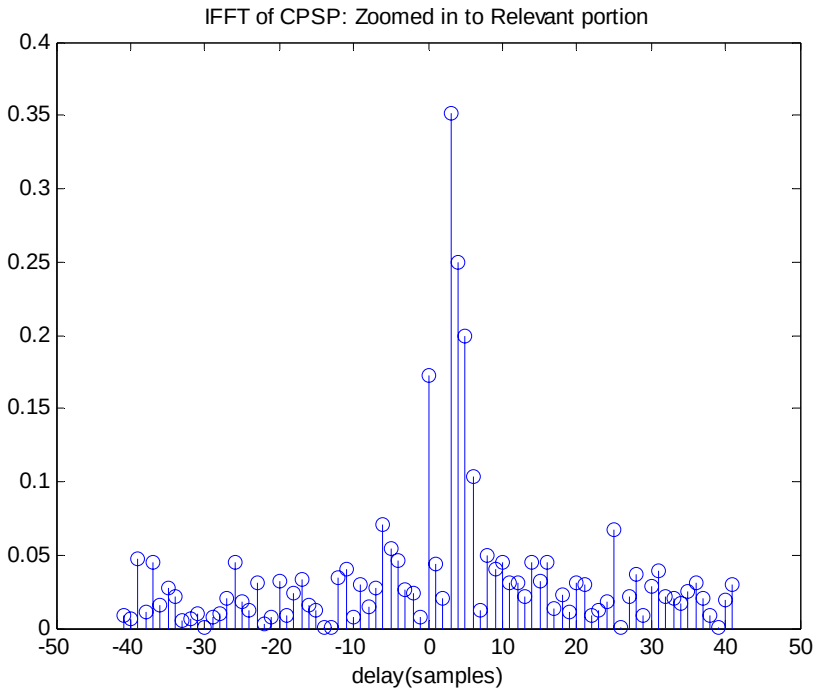


Figure 11 – Detected output delay is correct at 3 samples, but sees lots of noise

We see that there is a lot more noise and extraneous information in the resulting correlation, but there is still a good peak detected when the sound is delayed by 3 samples, as is shown in the output graph.

HARDWARE

To accomplish our acoustical source analysis, it was necessary to take hardware into consideration when proposing an implementation for our signal processing algorithm. To collect the acoustical waves and transform them into signals that we can interpret, we initially proposed utilizing eight omni-directional microphones but decided upon using two Sennheiser MD518 microphones connected to a Symetrix SX202 two channel pre-amplifier. From the pre amplifier, the microphone input signals were sent to the external interface of the Delta 1010 where the signals were sampled and then sent to the PCI interface card installed in a lab desktop computer.

The specifications of the Sennheiser microphones are as follows:

FEATURES

- Good feedback and handling noise rejection
- "Powerful" sound
- "Waterproof" sound inlet
- All-metal housing
- Unbreakable stand holder

TECHNICAL DATA

Acoustical mode of operation	pressure gradient receiver
Frequency response	50 - 16000 Hz
Directional characteristic	cardioid
Open circuit output voltage at 1000 Hz	1,3 mV/ Pa±3dB(= -77 dBV)
Magnetic interference factor up to 16 kHz	≤ 5 µV / 5 µT
Electrical impedance at 1 kHz	200 Ohm
Min. load impedance	200 Ohm
Connector	3-pin. XLR-3-plug (Canon)
Delivery	1 microphone, 1 stand holder MZA 431 (seulement MD 518)

[8]

The microphones were a key component in the hardware setup as they were the elements responsible for translating the acoustical waves into electrical signals that we would be sampling once received by the Delta 1010 D/A converter. The Sennheiser microphones provided a relatively high quality input signal which would assist us in minimizing error in the sampling and processing stages later on in our implementation.

The pre-amplifier that we used was necessary to amplify the signals coming in from the microphones such that amplitude of the translated electrical wave would be able to be interpreted by the input ports on the Delta 1010 external interface. The gains were adjustable on the pre-amplifier for each channel and we decided to set the gain at 40dB after realizing that for the sound signals that we were inputting into the system had various signal to noise ratios depending upon the uncooperative test environment in the lab. We decided on leaving the amplification lower in the interest of reducing the amplification of noise that we were experiencing in testing our implementation in real time. The method used to decide upon this figure was trial and error. We wanted to obtain the highest signal to noise ratio without clipping the input signal by amplifying it unnecessarily. The specifications for the pre amplifier are as follows:

---

[8] http://www.sennheiser.com/sennheiser/old_manual.nsf/resources/MD_518_5188%20FE_49882_Sp3.pdf/$File/MD_518_5188%20FE_49882_Sp3.pdf

[9]

Figure 11 - SX202

Features:

- Input levels to +14 dBV Left/Right and Left + Right outputs
- Uncompromising sonic performance
- +48 volt phantom powering
- Polarity reversal
- Compact (1/2 rack) and lightweight

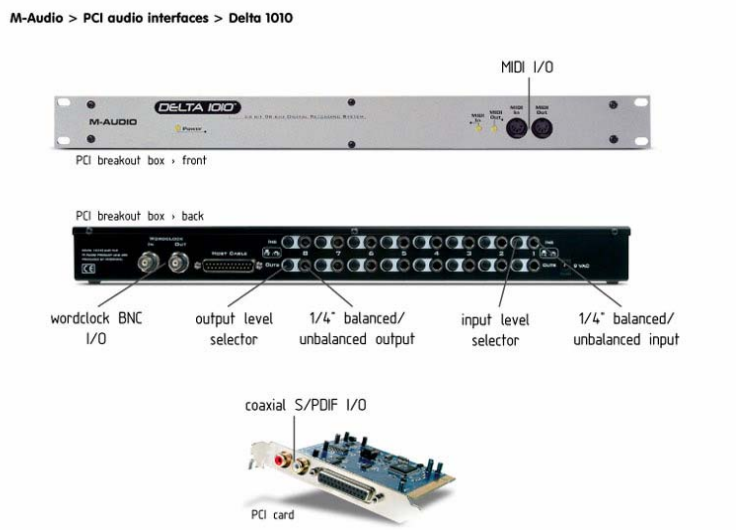
Specifications

- Maximum Gain 60 dB Minimum Gain 20 dB
- Outputs Type low-Z Output Source Impedance 600 ohms balanced 300 ohms unbalanced
- Maximum Output Level (600 ohms) +24 dBm balanced +18 dBm unbalanced
- Connectors 1/4" TRS balanced/unbalanced
- Power Requirements 16V ac, 200 ma
- Physical Size 1/2 rack unit
- Size (HWD) 1.75 x 8.5 x 6.5 in

The Delta 1010 was originally purchased to provide us with a means for simultaneously recording eight channels at a time using a single interface. The previous group that used four channels stated that they had a difficult time in synchronizing inputs from different interfaces so we searched for an eight channel interface that could give us synchronized inputs. We later only implemented two channels but still took advantage of the onboard codec to sample the input waveforms as it was easier and more efficient to some processing work off of the DSK. Sampling the waveforms on the Delta 1010 instead of on the DSK allowed us to take further advantage of

[9] http://anonemusic.com/music/node/747

the DSP processor without processing interrupts. The specifications of the M-Audio Delta 1010 Digital Recording System are as follows:



[10]

Figure 12 – Delta1010

Full Features

- 10 x 10 24-bit/96kHz full-duplex recording interface
- 8 x 8 analog I/O on balanced/unbalanced 1/4" TRS
- S/PDIF digital I/O (coaxial) with 2-channel PCM
- SCMS serial copy management control
- digital I/O supports surround-encoded AC-3 and DTS pass-through
- up to 24-bit/96kHz fidelity
- 1 x 1 MIDI I/O
- analog outs can directly drive up to 7.1 surround
- +4dBu/-10dBV operation individually switched on rack-mount unit
- word clock I/O for sample-accurate device synchronization
- software-controlled 36-bit internal DSP digital mixing/routing
- zero-latency monitoring

Specifications

- Frequency Response: 20Hz-22kHz, +/-0.3dB
- Dynamic range (A-weighted): 109dB (A/D), 117dB (D/A)
- Signal-to-noise ratio (A-weighted): -109 dB (A/D), -117dB (D/A)
- THD +N: 0.00072% (A/D), 0.00200% (D/A)
- Size/weight (breakout box): 1-3/4" x 19" x 6-1/8"; 4.9 lbs.

We also had to develop a means for keeping the distance between array elements constant as this was a major assumption necessary for our algorithm to function

[10] http://www.m-audio.com/images/en/callouts/big/delta_1010.jpg

properly. To keep the microphone spacing constant but still provide a means for changing the separation distance if necessary later in the project directed us to create a variable setting spacing mechanism. Constructed of wood, we developed an adjustable spacing mechanism that slid eight individual securing mechanisms along a meter long axis. This allowed us to achieve a spacing variation from 10cm to approximately ¾ meter if necessary. Each microphone was secured to these mechanisms with a stainless steel strap that was adjustable for various widths of microphones. The microphone array is pictured below.

The standard TI 6713 DSK board was used as the center of the signal processing operations for our project. We took advantage of the additionally provided network interface card configuration to send data to the DSK. We configured the DSK to request information only after it was prepared to accept new information so as to avoid any buffer or racing complications from sending data faster than the DSK could have received it. The software that accomplished this integration process will be explained in detail in the software analysis of the DSK code.

FINAL DEMO RESULTS

For our final demonstration and implementation, we scaled back our project immensely. Due to our DSK algorithm code issues and not being able to successfully implement the full eight microphones correctly, we ensured that we had a two-microphone setup (one pair) working correctly, and demoed that implementation. Thus, our final output on the GUI is shown below:

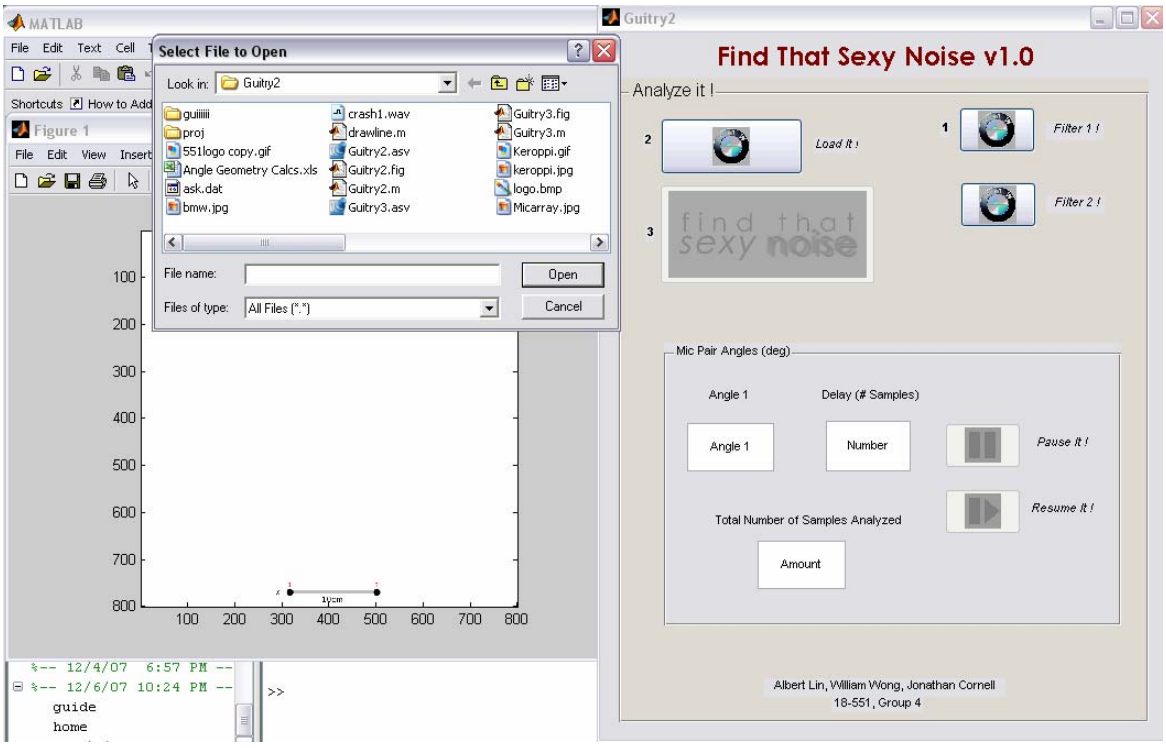


Figure 13 – MATLAB GUI Loading results file

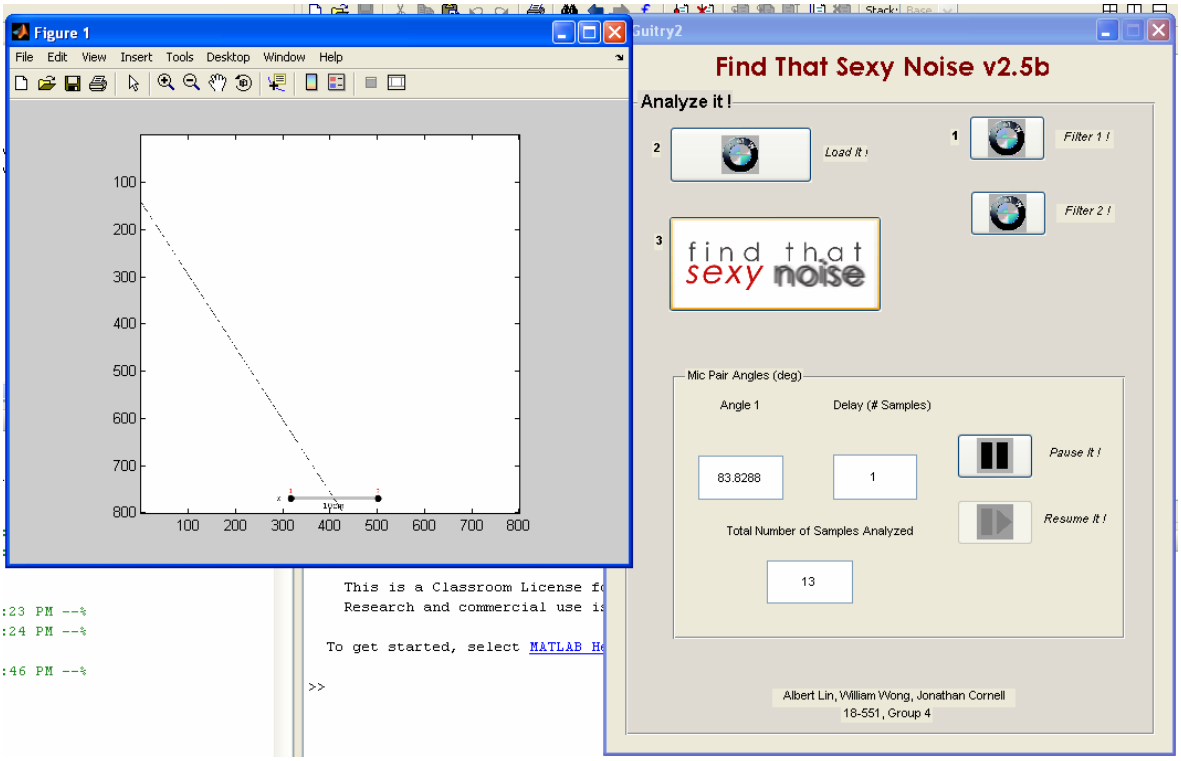


Figure 14 – MATLAB GUI showing the angle, sample delay, and projected direction of the detected sound source

We could then track a person talking while moving across the room with some moderate success. The noise in the 551 lab was a major factor in throwing off our algorithm calculations, as the resulting angles and lines being drawn jumped all over the place, though it did show a general trend of moving with the speaker. The implementation was in simulated real time, meaning the sound was prerecorded and then processed. Taking into account our array element spacing of 10 centimeters, we could achieve a relatively acceptable degree of accuracy, assuming we were staying in the far field conditions outlined in the geometries discussion.

In the end, our demonstration was with success in terms of calculating the obtained sound correctly but was not so successful in filtering out the excessive noise that resulted in a pseudo-random angle and line output. The DSK was able to successfully calculate and detect appropriate angles from the delays in the sound recordings, and the PC-DSK connection was successful in transferring the information.

ERROR

Our implementation depended greatly upon the ability of our algorithm to distinguish between different states of delay. As described in terms of our spacing considerations, the ability to distinguish between signals is given in terms of the delay

relative to an input signal. With our current spacing of 10cm, due to the propagation delay of acoustical waves we can distinguish a maximum of 10 samples of delay. This is *far* from the ideal case of being able to distinguish at least 90 samples. In only being able to resolve 10 possible delay values, we are taking 90 degrees of information and providing only 10 possible values for a location. Currently with spacing of 10cm, our error is at best 9 degrees. We initially considered this as part of our proposal phase such that we would be able to rely upon information from other arrays to narrow down the error from 10% to something less than 5% with overlapping position information. After modifying our initial proposal, we found that this error estimate was very conservative. The approximation of 10% was based solely on optimal data being processed into our algorithm. In reality with the ambient noise of the lab environment introduced into our sound signals, the CPSP algorithm was seeing delay values that were completely inaccurate (see geometry analysis for possible values of theta).

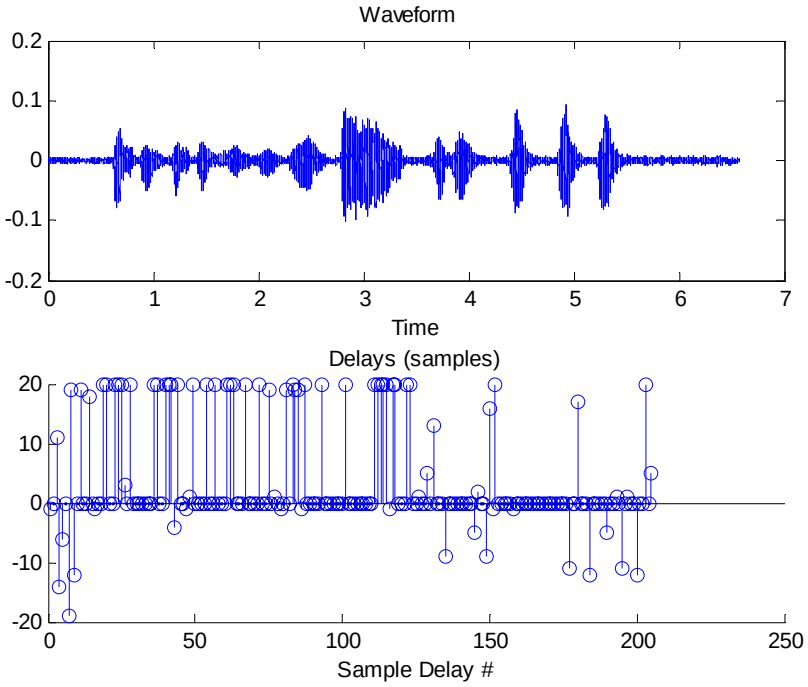


Figure 15 – Waveform and detected delays (error prone)

As described in our demo analysis, the values successfully followed a moving target in a relatively quiet environment but the results from a noisy sound clip gave results that were very error prone. The introduced noise caused errors in our correlation function and gave approximate delays that were incorrect. The incorrect delay results provided incorrect angle measurements which were incorrectly displayed on our GUI. We attempted to filter noise on a few test samples using Audacity's noise filtering algorithm in an attempt to optimize our results as much as possible early in the processing stage but were unable to improve the filtering of the randomization of the error found in the final results. Error could be improved upon by spacing the microphones further apart to

make it possible to resolve more possible values for sample delays. Another possibility could be to conduct up sampling of the data to resolve sub sample delays. We did not consider this in our initial proposal because we were relying upon error reduction to be conducted in the redundancy of additional arrays. Since we had to scale our design down to one pair of microphones in a short time, we were not able to implement any additional error correction techniques to compensate for the increased error.

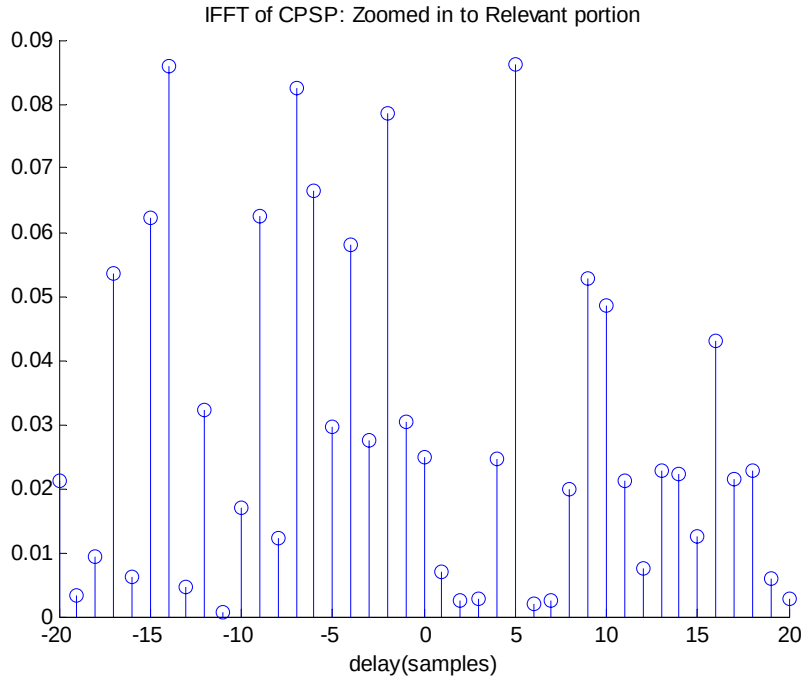


Figure 16 – CPSP result that shows there is lots of error in the technique

Error was also introduced in the physical characteristics of our array implementation in the way that our calculations assumed exactly 10 centimeters of spacing as well as exactly 344 m/s for the speed of sound in air. Our key assumptions rely heavily upon these numbers to calculate distances and angles as part of the trigonometric analysis and there is little room for error in the calculations on such a small scale. It is possible that even in the best case of all assumptions being true that our values could be 10 degrees off. In a far worse case, possibly because of noise the resultant delay could be completely inaccurate even to the point that with a multiple array implementation the calculated angles would not intersect.

FUTURE WORK AND POTENTIAL

An immediate possibility for future work would be to successfully implement 8 microphones using the same M-Audio setup that we had. It is just a matter of obtaining 8 quality microphones and to fully process the sound files. Our code actually has the flexibility (if tweaked a little bit) to receive 8 separate channels of sound information, so the realization of a T-array is not far off.

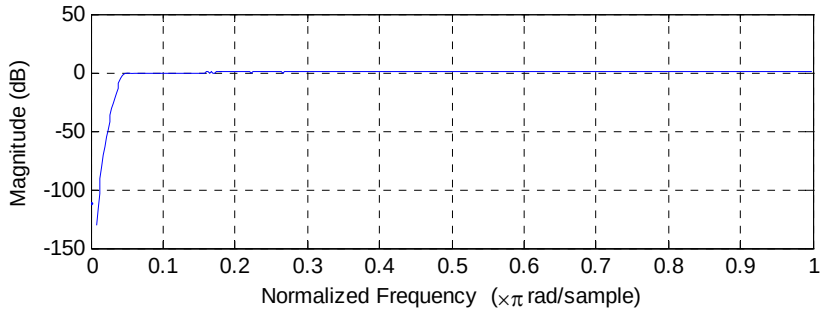
Another issue that could be improved would be the method of reading in and transferring data. We used sockets in series rather than in parallel, and so information transfer and thus subsequent calculations could have been much faster if done in parallel with the capabilities of the DSK.

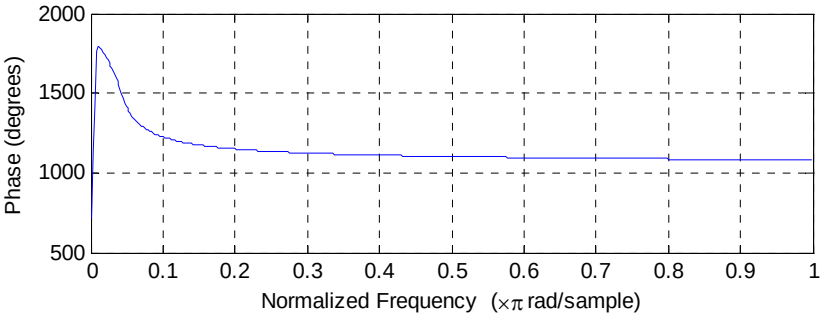
Up sampling could be a definite possibility to look at in order to improve the accuracy for tracking, but one might run into memory issues or might not be able to find a good enough up sampling method to warrant the extra calculations and padding. This was the case with our experimentation with up sampling, where we couldn't find a good enough padding buffer to justify implementing up sampling on the DSK.

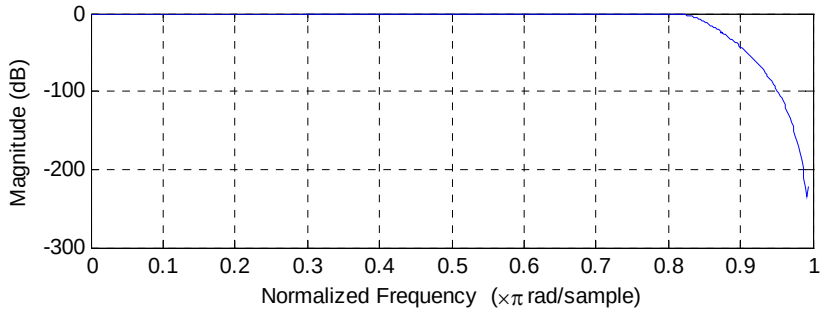
A more obvious expansion to the work would be to include more microphones in the array, maybe considering 12 or 16 microphones in an array, as the array geometry would be very interesting in terms of giving improved resolution and maybe even provide tracking in three dimensional space rather than just to dimensions (with some level of reasonable accuracy).

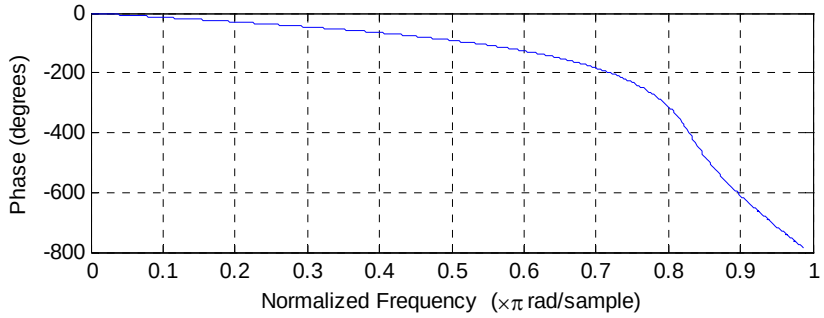
Another way to improve the concept would be to have a sound floor incorporated into the algorithm. Basically, by predetermining a frequency at which all frequencies lower than it would be considered extraneous, one could set those digital sample values to zero (silence) in order to create a better resulting CPSP or correlation result. The same could be said of having a sound ceiling, where the frequencies above a certain, predetermined frequency would be considered extraneous and be set to zero. If we had more time for the project, we would have attempted to incorporate these sound floor and ceiling ideas into the algorithm to provide superior results.

Butterworth filter responses

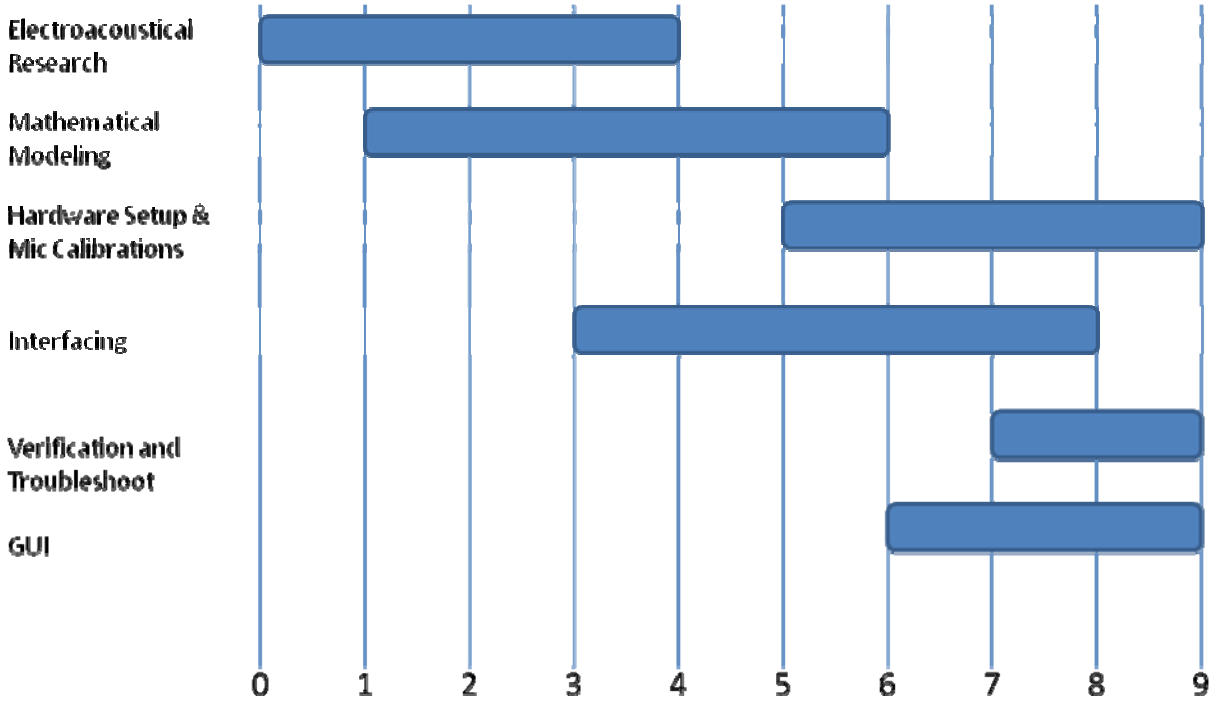








1. Electroacoustical Research – JC
2. Mathematical Modeling & Algorithms –
    - MATLAB Algorithm – WW
    - Geometries – AL
3. Hardware Setup & Mic Calibrations – JC
4. Implementing DSK Coding Algorithms – WW
5. Implementing PC-side Coding - JC
6. GUI – AL



References

1. Use of the Crosspower-Spectrum Phase in Acoustic Event Location, M. Omologo and P. Svaizer, IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING, VOL. 5, NO. 3, MAY 1997
    - Basis for implemented algorithm
2. MASLA – Spring 2004, 18-551
    - Starting point for our implementation
3. Acoustics, L. Beranek, 2nd Ed., Amer. Inst. Of Physics, 1987
    - Acoustical requirements and considerations for physical implementation
4. Issues in Acoustic Signal Processing and Recognition, C.H.Chen, Springer-Verlag, 1983
    - Filtering concepts for FFTs to reduce error
5. TMS320C67x Digital Signal Processing Library (DSPLIB) –
    - Non Cache - Optimized FFT, the center point of our algorithm
6. C.L. Luengo Hendriks - http://clluengo.lbl.gov/MATLAB/drawline.m
    - MATLAB script for drawing lines on images
7. http://audacity.sourceforge.net/
    - Audacity User Manual and Software Download for PC Recording Program
8. http://www.sonicspot.com/guide/wavefiles.html
    - Wave File Header References