NOISE REDUCTION WITH MICROPHONE ARRAYS FOR SPEAKER IDENTIFICATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Zachary G. Cohen

December 2012

COMMITTEE MEMBERSHIP

TITLE:	Noise Reduction with Microphone Arrays for Speaker Identification
AUTHOR:	Zachary G. Cohen
DATE SUBMITTED:	December 3, 2012

COMMITTEE CHAIR:	Vladamir Prodanov, Assistant Professor
COMITTEE MEMBER:	Jane Zhang, Associate Professor
COMMITTEE MEMBER:	Helen Yu, Professor

ABSTRACT

Noise Reduction with Microphone Arrays for Speaker Identification

Zachary G. Cohen

The presence of acoustic noise in audio recordings is an ongoing issue that plagues many applications. This ambient background noise is difficult to reduce due to its unpredictable nature. Many single channel noise reduction techniques exist but are limited in that they may distort the desired speech signal due to overlapping spectral content of the speech and noise. It is therefore of interest to investigate the use of multichannel noise reduction algorithms to further attenuate noise while attempting to preserve the speech signal of interest.

Specifically, this thesis looks to investigate the use of microphone arrays in conjunction with multichannel noise reduction algorithms to aid aiding in speaker identification. Recording a speaker in the presence of acoustic background noise ultimately limits the performance and confidence of speaker identification algorithms. In situations where it is impossible to control the noise environment where the speech sample is taken, noise reduction algorithms must be developed and applied to clean the speech signal in order to give speaker identification software a chance at a positive identification. Due to the limitations of single channel techniques, it is of interest to see if spatial information provided by microphone arrays can be exploited to aid in speaker identification.

This thesis provides an exploration of several time domain multichannel noise reduction techniques including delay sum beamforming, multi-channel Wiener filtering, and Spatial-Temporal Prediction filtering. Each algorithm is prototyped and filter performance is evaluated using various simulations and experiments. A three-dimensional noise model is developed to simulate and compare the performance of the above methods and experimental results of three data collections are presented and analyzed. The algorithms are compared and recommendations are given for the use of each technique. Finally, ideas for future work are discussed to improve performance and implementation of these multichannel algorithms. Possible applications for this technology include audio surveillance, identity verification, video chatting, conference calling and sound source localization.

ACKNOWLEDGMENTS

I would like to thank Dave Chambers from Lawrence Livermore National Lab for all of his support on this thesis. Without his profound knowledge and dedicated mentoring I would not have learned nearly as much as I did from this project. I would also like to acknowledge Bruce Henderer and the National Security Engineering Division of LLNL for setting up and funding the Cal Poly Graduate Internship program which made this research possible.

Table of Contents

LIST OF TABLES viii
LIST OF FIGURES ix
1. INTRODUCTION 1
1.1 Document Overview 2
2. BACKGROUND 3
2.1 Narrowband Beamforming 3
2.1.1 Delay Sum Beamforming 4
2.1.2 Other Narrowband Beamforming Algorithms 9
2.1.2.1 Maximum Signal to Noise Ratio Beamformer 10
2.2 Speaker Identification 12
3. NARROWBAND BEAMFORMING SIMULATIONS 14
3.1 Delay Sum Beamformer Simulations 14
3.1.1 Frequency Dependence 16
3.1.2 Spatially Correlated Noise 20
4. BROADBAND MULTICHANNEL NOISE REDUCTION ALGORITHMS 24
4.1 New Problem Description 24
4.2 Multichannel Wiener Filter 27
4.3 Spatio-Temporal Prediction Filter 29
5. SIMULATIONS 32
5.1 Algorithm Simulation 32
5.1.1 Matlab Implementation 32
5.1.2 Three Dimensional Noise Modeling 35
5.2 Speaker Identification Simulation 43
6. EXPERIMENTAL RESULTS AND ANALYSIS 53
6.1 Experimental Set Up and Procedure 53
6.2 Data Collections 56
6.2.1 Outdoors 5/10/2011 57
6.2.1.1 Motivation and Procedure 57
6.2.1.2 Results 58
6.2.1.3 Analysis 64

6.2.2 Indoors: Building 123 65
6.2.2.1 Motivation and Procedure 65
6.2.2.2 Results 67
6.2.2.3 Analysis 73
6.2.3 Outdoors 11/17/2011 74
6.2.3.1 Motivation and Procedure 74
6.2.3.2 Results 77
6.2.3.3 Analysis 81
6.3 Additional Observations 84
7. SUMMARY OF FINDINGS 92
7.1 Algorithm Comparison 92
7.2 Conclusion 94
7.3 Future Work 95
Bibliography 97
APPENDIX A: TIMIT PROMPTS 99
APPENDIX B: MATLAB CODE 100

LIST OF TABLES

Table Page

Table 6.1 Input SNR and delay sum output SNR for each speaker and distance case from 5/10/11 58

Table 6.2 Input SNR and delay sum output SNR for each speaker and distance case from B123 67

Table 6.3 Average input SNR and delay sum output SNR for each distance case from 11/17/11 77

LIST OF FIGURES

Figure Page Number

Figure 1.1 Model of a speech recording environment 1

Figure 2.1 Plane wave front arriving at linear array [1] 5

Figure 2.2 Theoretical directional response of a delay sum beamformer using a linear array. Parameters: f = 2 kHz signal frequency, d = .08cm element spacing, N = 9 receivers, θ = 90 degree look direction, c = 345 m/s 8

Figure 2.3 Theoretical directional response of the Maximum SNR beamformer (solid blue) and delay sum beamformer (dashed red) with desired signal at 90^o and an interfering source at 60^o [1] 11

Figure 3.1 Simulated (blue) and Theoretical (dashed green) directional responses of the delay sum beamformer to a 2 kHz signal using a .08cm linear array 15

Figure 3.2 Illustration of spatial aliasing 17

Figure 3.3 Delay sum beamformer frequency dependance 19

Figure 3.4 Delay sum beamformer steered response to spatially correlated noise 22

Figure 5.1 Matlab implementation block diagram of broadband multichannel filters ... 33

Figure 5.2 Simulation and Experimental Array Geometry 39

Figure 5.3 STP filter output SNR with varied filter length and simulated noise bandwidth..... 39

Figure 5.4 Wiener Filter output SNR with varying filter length and simulated noise bandwidth..... 40

Figure 5.5 Output SNR of broadband filters using a simulated speaker in the presence of real indoor noise..... 42

Figure 5.6 Algorithm for running speaker identification system 46

Figure 5.7 SID score vs. SNR for the 5 speakers in the target group 48

Figure 5.8 Average target group SID score vs. SNR 48

Figure 5.9 SID score vs. speech signal bandwidth for the 5 target group speakers..... 50

Figure 5.10 Average SID score vs. speech signal bandwidth of the target group 50

Figure 6.1 Microphone array used for experimental data collections [11] 55

Figure 6.2 Array microphones and specifications [11] 55

Figure 6.3 ABACUS Data Acquisition System and specifications [11] 56

Figure 6.4 Microphone (blue) and speech source positions (red) for outdoor 5/10/2011 experiment [11] 58

Figure 6.5 Wiener output SNR vs. filter length for Outdoor 5/10/11 female data 59

Figure 6.6 STP filter output SNR vs. filter length for Outdoor 5/10/11 female data 59

Figure 6.7 Wiener filter output SNR vs. filter length for Outdoor 5/10/11 male data..... 61

Figure 6.8 STP filter output SNR vs. filter length for Outdoor 5/10/11 male data 61

Figure 6.9 Female SID score vs. distance position for Outdoor 5/10/11 data 63

Figure 6.10 Male SID score vs. distance position for Outdoor 5/10/11 data..... 63

Figure 6.11 Indoor B123 experiment setup 66

Figure 6.12 Microphone (blue) and speech source (red) positions for indoor B123 experiment..... 67

Figure 6.13 Wiener Filter output SNR vs. filter length for indoor B123 female speaker data 68

Figure 6.14 STP Filter output SNR vs. filter length for indoor B123 female speaker data 68

Figure 6.15 Wiener Filter output SNR vs. filter length for indoor B123 male speaker data 69

Figure 6.16 STP Filter output SNR vs. filter length for indoor B123 male speaker data 69

Figure 6.17 Female speaker SID score vs. distance position considering maximum scores achieved by adaptive filters 70

Figure 6.18 Female speaker SID score vs. distance position considering average scores achieved by adaptive filters 71

Figure 6.19 Male speaker SID score vs. distance position considering maximum scores achieved by adaptive filters 71

Figure 6.20 Male speaker SID score vs. distance position considering average scores achieved by adaptive filters 72

Figure 6.21 Outdoor 11/17/11 experiment set up 76

Figure 6.22 Microphone (blue) and speech source (red) positions for outdoor 11/17/11 data collection 76

Figure 6.23 Average Wiener Filter Output SNR vs. filter length for all target group speakers 78

Figure 6.24 Average STP Filter Output SNR vs. filter length for all target group speakers 78

Figure 6.25 SID score vs. distance position considering the maximum score achieved by the adaptive filters when using a certain filter length. 80

Figure 6.26 SID score vs. distance position considering the average score over all filter lengths and speakers 80

Figure 6.27 Noise power vs. the time that the noise sample was taken during the 11/17/11 data collection 86

Figure 6.28 Wiener filter output SNR vs. filter length with varying number of microphones 87

Figure 6.29 STP filter output SNR vs. filter length with varying number of microphones 87

Figure 6.30 Delay sum beamformer output noise power vs. look direction for outdoor 5/10/11 data 90

Figure 6.31 Delay sum beamformer output noise power vs. look direction for indoor B123 data 90

1. INTRODUCTION

Techniques for recording and preprocessing audio have many applications in communication, surveillance and entertainment. When recording audio, it is important to eliminate all unwanted noise before further application specific processing is performed. Noise present due to the uncontrollable nature of a recording environment can be problematic to reduce as it consists of interfering sources and is statistically non-stationary. Because the characteristics of the noise change over time, classical single channel filtering techniques cannot be used to remove this noise as they will also distort the speech signal of interest. Recently, the use of multichannel processing techniques has been investigated to see if spatial information provided by microphone arrays can be exploited to improve noise reduction.

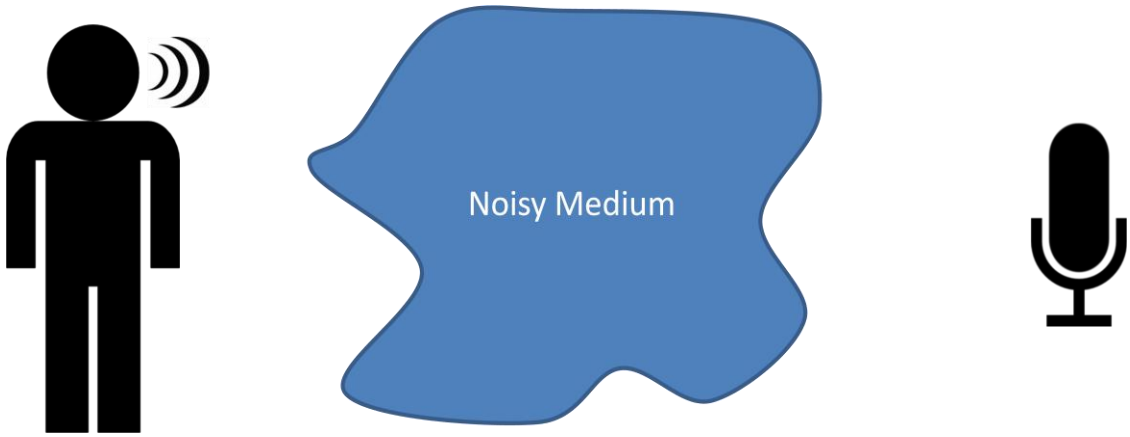

Figure 1.1 Model of a speech recording environment

One specific application where the noise environment is particularly hard to control is in the area of speaker identification. Speaker identification algorithms today are fairly accurate when speech samples are taken in a quiet environment with the speaker talking directly into the microphone. However, in applications such as surveillance, the noise

environment cannot always be controlled and the speaker will not always speak directly into a microphone. This reduction in signal to noise ratio ultimately limits the performance and confidence of speaker identification algorithms. It is therefore important to investigate the feasibility of deploying microphone arrays in conjunction with multichannel noise reduction techniques to aid in speaker identification. In particular, this thesis looks to see if these techniques can be effectively applied in different common environmental scenarios with surveillance applications in mind.

1.1 Document Overview

This document provides a thorough report documenting the progression of this thesis from start to finish. This chapter has introduced the problem that this thesis looks to address. Chapter 2 then discusses existing narrowband beamforming solutions as well as notes possible shortcomings of these approaches. Chapter 3 uses simulation to verify theoretical operation of the delay sum beamformer as well as demonstrates its possible shortcomings discussed in Chapter 2. Chapter 4 proposes two new adaptive multichannel noise reduction algorithms which provide advantages over the delay sum beamformer. Chapter 5 implements these new approaches in a Matlab simulation environment to see what kind of performance can be expected. Chapter 5 also uses simulations to characterize the performance of the speaker identification system used in this thesis. Chapter 6 provides the procedure and results of 3 field experiments performed in real environments at Lawrence Livermore National Laboratory. Finally, Chapter 7 provides an overall comparison of the filtering algorithms, a conclusion and ideas for future work.

2. BACKGROUND

In order to gain insight into the concept of array processing, it was useful to explore well known multichannel processing techniques originally developed for narrowband applications. It was also of interest to look into speaker identification algorithms and what factors limit their performance.

2.1 Narrowband Beamforming

The concept of array processing was first developed for applications in radar, sonar, seismology and communications [1]. The idea is that by using multiple receivers separated in space, you can create what's known as a "spatially selective" filter. This allows systems to receive only signals coming from certain directions and filter out interfering signals from other directions. Spatial filtering, or beamforming, is especially useful if the interfering signal is the same frequency as the signal of interest. By using multiple channels instead of a single channel, endless processing options become possible through tweaking array geometries and exploring various channel weighting schemes. This extra degree of freedom provided by multiple sensor systems lead to the development of many beamforming techniques tailored for different applications. Though narrowband techniques are well understood, issues may arise when these techniques are applied to broadband signals such as speech. Therefore, additional processing methods must be investigated that are broadband in nature.

2.1.1 Delay Sum Beamforming

The first and simplest type of beamformer explored in this thesis is the delay sum beamformer. As its name implies, this method works by first delaying the signals received at each microphone and then summing these signals to create a single "beamformed" output.

Because sound travels at a fixed speed of 345m/s in air, sound waves arrive at each microphone at different times. From this assumption, relative theoretical time delays can be calculated for each receiver that corresponds to some known signal direction of arrival or "DOA". If these delays are applied to the received signal at each microphone and these time shifted signals are summed together, any signal coming from the desired DOA is added coherently while interfering sources and noise are added incoherently resulting in noise reduction. Simply put, the delay sum beamformer can be thought of as a receiver that can adjust its listening or receiving direction electronically without mechanically shifting the array through the use of time delays.

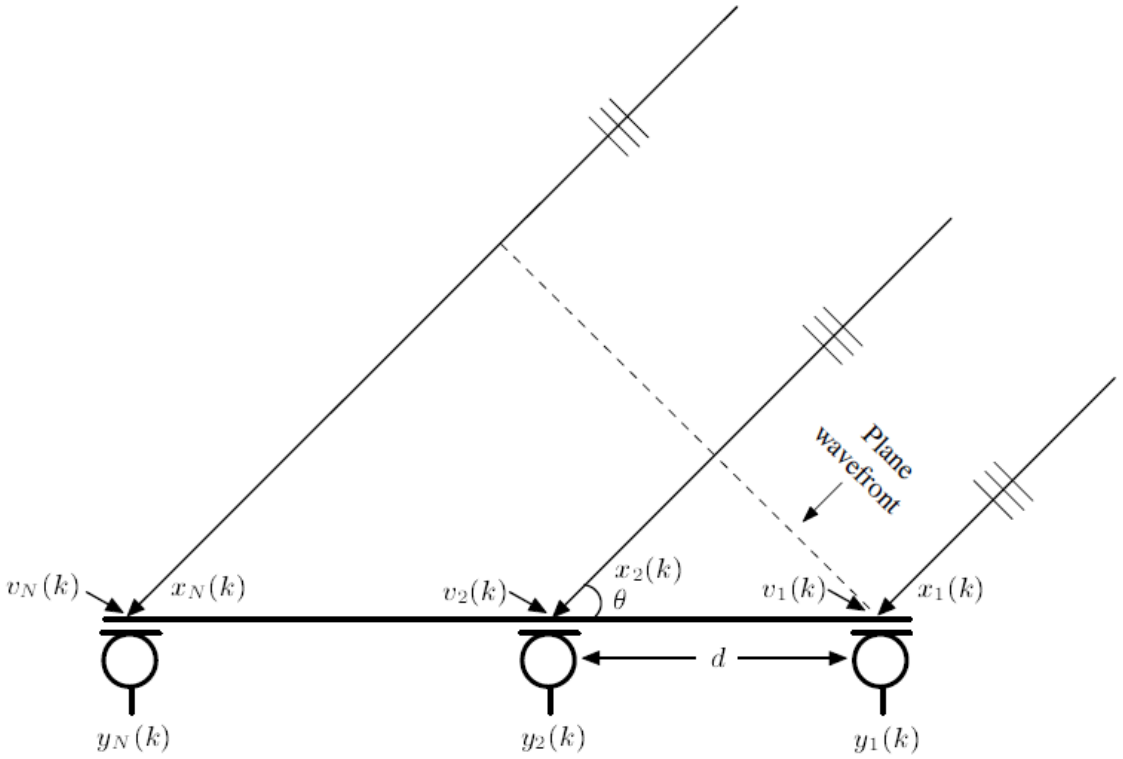

Figure 2.1 Plane wave front arriving at linear array [1]

To understand this concept analytically, consider a microphone array consisting of 'N' elements spaced a distance, 'd', apart (Figure 2.1). The signal seen at the output of microphone 'n', $y_n(k)$, can be modeled as the superposition of the desired speech signal, $x_n(k)$, and background noise, $v_n(k)$:

$$y_n(k) = x_n(k) + v_n(k) \tag{2.1}$$

Given that it is desired to only receive signals from the direction θ, the relative time delays applied to each microphone can be calculated using equation 2.2.

$$\tau_n = \frac{(n-1)d\cos\theta}{c} \tag{2.2}$$

The microphone number is denoted as ‘n’ (n = 1, 2, 3… N) and ‘c’ is the speed of sound in air (assumed to be 345 m/s). Equation 2.2 is derived simply from the array geometry and it should be noted that other array geometries may require a different time delay equation.

Once the received signals are delayed according to the desired “look direction”, these time aligned signals, $y_{a,n}(k)$, are added together to form a single channel output, z_{DS}, according to equation 2.3.

$$z_{DS}(k) = \frac{1}{N}\sum_{n=1}^{N} y_{a,n}(k) \tag{2.3}$$

where,

$$y_{a,n}(k) = y_n(k + \tau_n)$$

This single channel output theoretically recovers the signal in the specified look direction while attenuating interfering signals and noise originating from other directions because they are added out of phase. It should be noted that the $\frac{1}{N}$ factor is included in order to normalize the gain of the beamformer to unity.

To further analyze the delay sum beamformer, it is useful to look at its directional response. A directional response shows how signals from all directions contribute to the overall output of the spatial filter. The directional response can be thought of like a frequency response in traditional signal processing. The directional response of a delay sum beamformer can be derived analytically by taking the spatial Fourier Transform of equation 2.3 to get 2.4 [1]:

$$S_{DS}(\psi) = \frac{1}{N}\sum_{n=1}^{N}\left[e^{-\frac{j2\pi f(n-1)d\cos\psi}{c}}\right]e^{\frac{j2\pi f(n-1)d\cos\theta}{c}}$$

$$\downarrow \qquad (2.4)$$

$$S_{DS}(\psi) = \frac{1}{N}\sum_{n=1}^{N} e^{-j2\pi f(n-1)d[\cos(\psi)-\cos(\theta)]/c}$$

It is seen from 2.4 that the directional response of the delay sum beamformer depends on the designed look direction θ, which sets the time delays, and the actual DOA of the signal, ψ. The magnitude of the directional response can then be taken (equation 2.5 [1]) and plotted versus ψ (Figure 2.2).

$$A_{DS}(\psi) = |S_{DS}(\psi)|$$
$$= \frac{\sin\{N\pi f d[\cos(\psi)-\cos(\theta)]/c\}}{N\sin\{\pi f d[\cos(\psi)-\cos(\theta)]/c\}} \qquad (2.5)$$

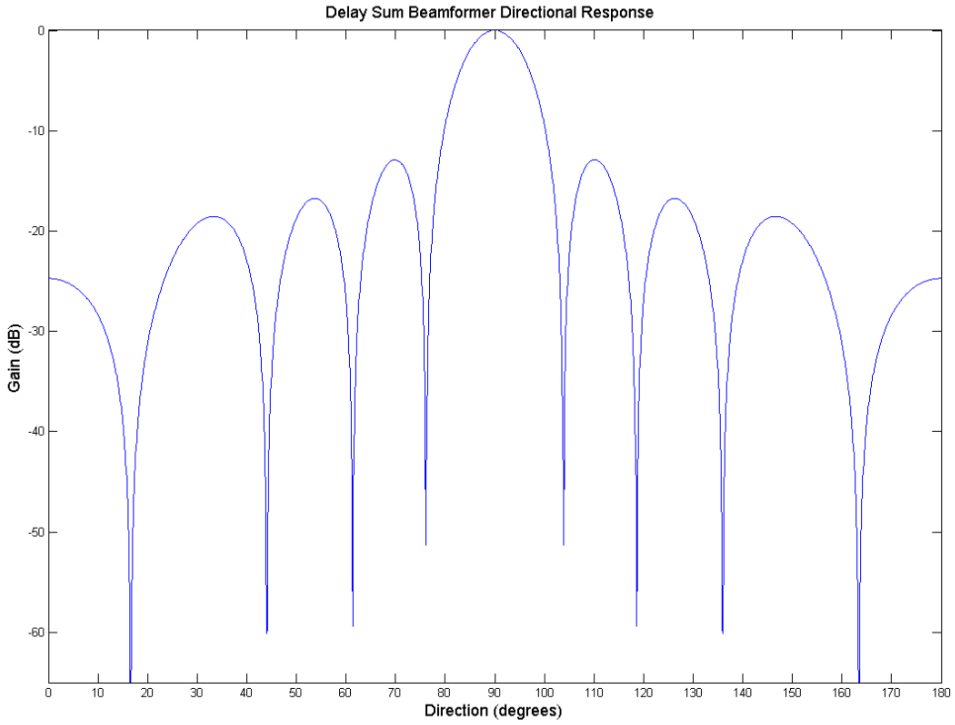

Figure 2.2 Theoretical directional response of a delay sum beamformer using a linear array. Parameters: f = 2 kHz signal frequency, d = .08 cm element spacing, N = 9 receivers, θ = 90 degree look direction, c = 345 m/s

After plotting the theoretical directional response, it is important to highlight some key aspects. The directional response exhibits a band pass shape with respect to direction with the "pass band" being referred to as the "main lobe" while the stop band characteristics are referred to as the "side lobes". Changing the designed look direction or delays of the beamformer will change the location of the main lobe while altering the channel weighting controls the shape of the main lobe and side lobes. It is also important to note that the directional response is plotted for a single frequency sinusoid. Because the frequency variable 'f' appears in the directional response equation, it is apparent that the directional response varies with frequency. This is a key observation and will be explored further in the simulation section later in this thesis.

The delay sum beamformer is a simple technique that was originally developed for narrowband applications such as radar. This beamformer is easy to realize in practice but exhibits some weaknesses when applied to broadband signals such as speech. The delay sum beamformer's frequency dependence, among other shortcomings explored later, ultimately calls for broadband multichannel noise reduction methods to be developed in order to process speech signals. Additionally, implementation of the delay sum beamformer also requires "a priori" knowledge such as the relative position of the microphones as well as the DOA of the signal of interest [1]. This information is not always known or able to be estimated for certain applications making it desirable to develop adaptive techniques that can calculate this information implicitly.

2.1.2 Other Narrowband Beamforming Algorithms

To further explore the concept of array signal processing, other narrowband beamforming techniques were looked at to see if ideas presented for single frequency applications could be extended to broadband speech. While the delay sum beamformer utilizes time delays and equal channel weighting, other beamforming algorithms utilize different channel weighting schemes to control the shape of the directional response. This concept can be thought of as analogous to a Finite Impulse Response (FIR) filter in the time domain, as each time sample is weighted differently before summing the samples together to form the output [1].

The following section briefly looks at one adaptive beamformer which utilizes some optimization criterion to calculate channel weightings. Though beamformer channel weighting schemes are abundant and very useful in certain applications, they are

ultimately not implemented experimentally in this thesis. The goal of exploring this beamformer is to gain insight into multichannel optimum-adaptive filtering that will help later when implementing other multichannel noise reduction approaches.

2.1.2.1 Maximum Signal to Noise Ratio Beamformer

The maximum SNR beamformer is an adaptive beamformer that looks to maximize the signal to noise ratio at the output of the beamformer using channel weighting [1]. This beamformer was looked at to see how information learned about the noise can aid in calculating weighting coefficients that will reduce it.

The delay sum beamformer presented in 2.1.1 is theoretically able to reduce noise based on the assumption that the noise present is equal and uncorrelated at all channels. When the noise is correlated, e.g. a directional interfering source in the same frequency range, the theoretical noise reduction of $\frac{1}{N}$ is not always achieved [2]. This is because the side lobe characteristics of the delay sum beamformer are fixed and the interfering source DOA may lie on a side lobe peak resulting in less attenuation. This scenario motivates the development of an approach that adjusts the shape of the side lobes according to the interfering source.

The solution to this problem is found in the maximum SNR approach which estimates the direction of the interfering source and places a null in its direction. The maximum SNR beamformer is an adaptive approach and achieves this desired operation by performing the following steps. Assuming there is time when only the noise or interfering source is present, the noise correlation matrix must first be estimated using equation 2.6 [1].

$$\boldsymbol{R}_{v_a v_a} = E[\boldsymbol{v_a}(k)\boldsymbol{v_a}(k)^{\boldsymbol{T}}] \qquad (2.6)$$

In the above equation, $\mathbf{v_a}(k) = [v_{a,1}(k)\ v_{a,2}(k)\ v_{a,3}(k)\ \dots\ v_{a,N}(k)]^T$ is a column vector containing the k^{th} sample at each microphone and $E[\cdots]$ represents the statistical expected value. It is shown in [1] that by maximizing the theoretical SNR, the channel weights are found by solving the generalized eigenvector problem of equation 2.7

$$\sigma_s{}^2\left(\boldsymbol{R}_{v_a v_a}{}^{-1}\right)\boldsymbol{\alpha}\boldsymbol{\alpha}^T\boldsymbol{h_{max}} = \lambda_{max}\boldsymbol{h_{max}} \qquad (2.7)$$

where $\boldsymbol{h_{max}}$ is the eigenvector corresponding to the maximum eigenvalue, λ_{max}, of $\sigma_s{}^2\left(\boldsymbol{R}_{v_a v_a}{}^{-1}\right)\boldsymbol{\alpha}\boldsymbol{\alpha}^T$. The channel weighting vector $\boldsymbol{h_{max}}$is an Nx1 column vector containing the weight for each channel. The signal attenuation vector $\boldsymbol{\alpha}$ is also an Nx1 column vector containing the attenuation of the desired signal from its source to the microphones. Since $\boldsymbol{\alpha}$ is not always known, it can be set to all ones for simplicity.

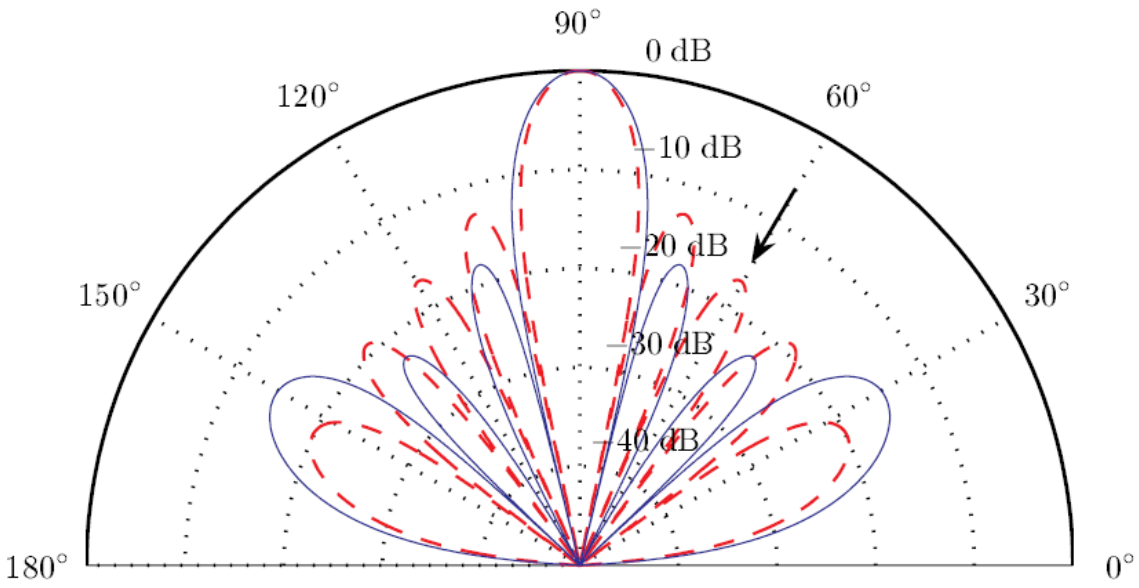

Figure 2.3 Theoretical directional response of the Maximum SNR beamformer (solid blue) and delay sum beamformer (dashed red) with desired signal at 90° and an interfering source at 60° [1]

Figure 2.3 shows an example application of the maximum SNR beamformer. The solid blue line is the maximum SNR weighting response while the dotted red line is the delay sum beamformer response. The interfering source is the same 2 kHz sinusoid as the desired signal but with a DOA of 60^o instead of 90^o. If just the delay sum beamformer is used, the 60^o interfering source would fall at the peak of a side lobe which would cause this interfering source to pass through the spatial filter with less than optimal attenuation. The maximum SNR beamformer is able to estimate the interfering sources DOA from its correlation matrix, $\boldsymbol{R}_{v_a v_a}$, and place a null at 60^o to achieve maximum signal to noise ratio at the output.

The maximum SNR beamformer shows that it is possible to reduce noise by adaptively estimating its spatial characteristics using microphone arrays. This adaptability is a very attractive feature and will be utilized in the design and implementation of the broadband multichannel noise reduction algorithms in Chapter 4. It is important to note that in order to calculate the noise statistics, there must be periods of time where there is only noise present in the system.

2.2 Speaker Identification

Speaker identification is the process of determining the identity of an individual based on the unique characteristics of their speech [3] [4]. The ability to indentify someone from their voice alone is powerful and finds many applications in access security, surveillance, and other voice operated systems.

The process of speaker identification or S.I.D. consists of taking a speech sample from an unknown speaker and matching it to a known speaker in a database. Samples of

speech can either be collected using text dependant or independent methods. Text dependant methods require the speaker to read a preselected phrase while text independent systems do not. Speech samples from known speakers in the database and unknown speech samples are classified and compared using techniques such as Gaussian Mixture Models, Vector Quantization and hidden Markov Models [3]. These methods extract features from speech that are unique to that individual. These speech features derive from anatomical features such as mouth and throat shape as well as "learned behavioral patterns" like pitch and style [3].

Though speaker identification systems have proven robust in controlled laboratory experiments, their performance suffers when used in practical application environments where the noise cannot always be controlled [5]. It is therefore useful to employ noise reduction algorithms to reduce ambient background noise before running the S.I.D. system. Specifically, this thesis looks at speaker identification for surveillance applications which require taking speech samples in the presence of various ambient noise environments without speaker cooperation. Therefore, this thesis proposes the use of microphone arrays to collect and process these speech samples in practical environments to see if S.I.D. performance can be improved.

3. NARROWBAND BEAMFORMING SIMULATIONS

In the previous chapter, narrowband beamforming techniques were introduced because they were the first type of array processing algorithms developed and are fairly well understood. Initially, one would think that simply replacing an array of receivers with an array of microphones and applying a delay sum beamformer would be effective for use with speech, but uniform operation may not be guaranteed over a broad frequency range. Many issues arise when trying to apply beamforming to broadband signals due to the frequency dependant behavior caused by applying time delays. Therefore, this chapter looks to verify the operation of the delay sum beamformer as well as explore possible shortcomings through Matlab simulations.

3.1 Delay Sum Beamformer Simulations

In order to verify single frequency operation of the delay sum beamformer, a three dimensional sound source simulator was developed in Matlab. The function "sim_3D.m" (Appendix A) simulates the signals received at a microphone array of some geometry given a user defined source position and source signal content. This was achieved by replicating the source signal for each channel of the array and then adding relative time delays derived from the source to microphone distances and the speed of sound (345 m/s). The delay sum beamformer was then implemented using the Matlab function "DS_beamformer.m" which basically corrects the delays applied by the source simulator in order to recover the signal at a user defined look direction. Ideally this look direction would be the exact location of the sound source.

The first goal of exploring the delay sum beamformer was to verify proper operation using the simulator. This was done by simulating the directional response of the delay sum or DS beamformer and comparing it with the theoretical directional response produced by equation 2.5. In order to mimic a directional response in the simulator, a sinusoidal source of some frequency was generated at various positions corresponding to different DOA angles in reference to the array. The delay sum beamformer output signal power for each source position was then calculated and plotted versus DOA for a fixed look direction. Figure 3.1 shows the simulated directional response of a delay sum beamformer to a 2 kHz sinusoid with a designed look direction of 90^{o}. The microphone array used was a 9 element linear array with element spacing of 8 cm. The theoretical directional response from equation 2.5 is also plotted (green) for comparison.

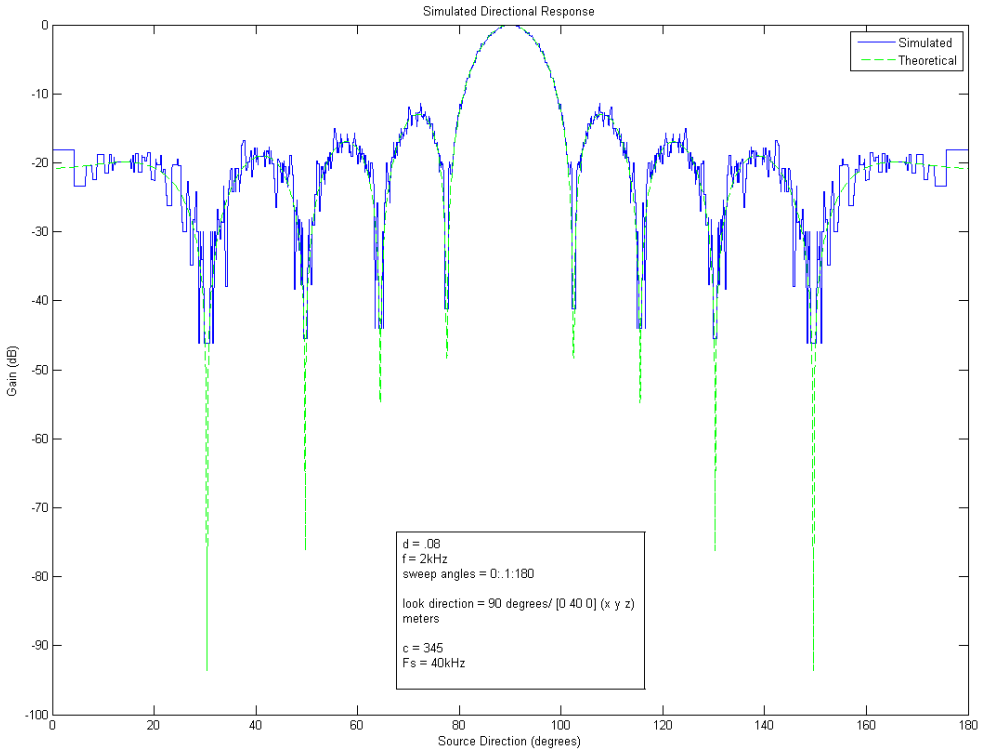

Figure 3.1 Simulated (blue) and Theoretical (dashed green) directional responses of the delay sum beamformer to a 2 kHz signal using a .08cm linear array

It is obvious from Figure 3.1that the delays sum beamformer performs as predicted from analytical methods. The slight “choppiness” of the simulated plot comes from the digitization of the source signal in terms of time sampling. This effect is more noticeable when the source DOA is farther from the designed look direction when the change in the source position cannot be resolved by the sampling period. A high sampling rate of 40 kHz was used in this simulation to minimize this effect.

Once the correct operation of the delay sum beamformer was verified, it was then of interest to explore its shortcomings in order to better understand possible limitations. Some of these shortcomings come from the DS beamformer’s frequency dependence while other issues stem from the noise field environment.

3.1.1 Frequency Dependence

Because this thesis focuses on the use of multichannel filtering techniques for speech applications, it is important to note what problems can occur when the delay sum beamformer is used to process broadband signals. Once characterized, these shortcomings may provide insights into whether the delay sum beamformer is a valid broadband approach. This section explores two frequency dependant issues; spatial aliasing and low pass filtering, which are mentioned in [1].

When recalling the fundamentals of digital signal processing, one of the first topics presented is the concept of aliasing. Though aliasing is traditionally thought of as relating to time sampling, spatial sampling also suffers from a similar phenomenon. A plane wave propagating through space varies periodically in amplitude with respect to spatial position as well as time. Therefore, in order to accurately sample a signal in space,

samples must be taken in fine enough increments to uniquely represent signals with different wavelengths. Specifically, the spacing of elements in an array should be less than $\frac{\lambda}{2}$ of the shortest wavelength signal to be received [1]. This can be thought of like the Nyquist sampling criterion in the time domain.

When using the delay sum beamformer it initially seems desirable to utilize larger array spacings as this gives more spatial information and therefore a narrower main lobe for more selective directional response characteristics. But, if too large of an array spacing is used, higher frequency signals may pass through the spatial filter with no attenuation due to the spatial aliasing phenomena. To illustrate this consider a four element linear array with element spacing greater than half of a wavelength (Figure 3.2).

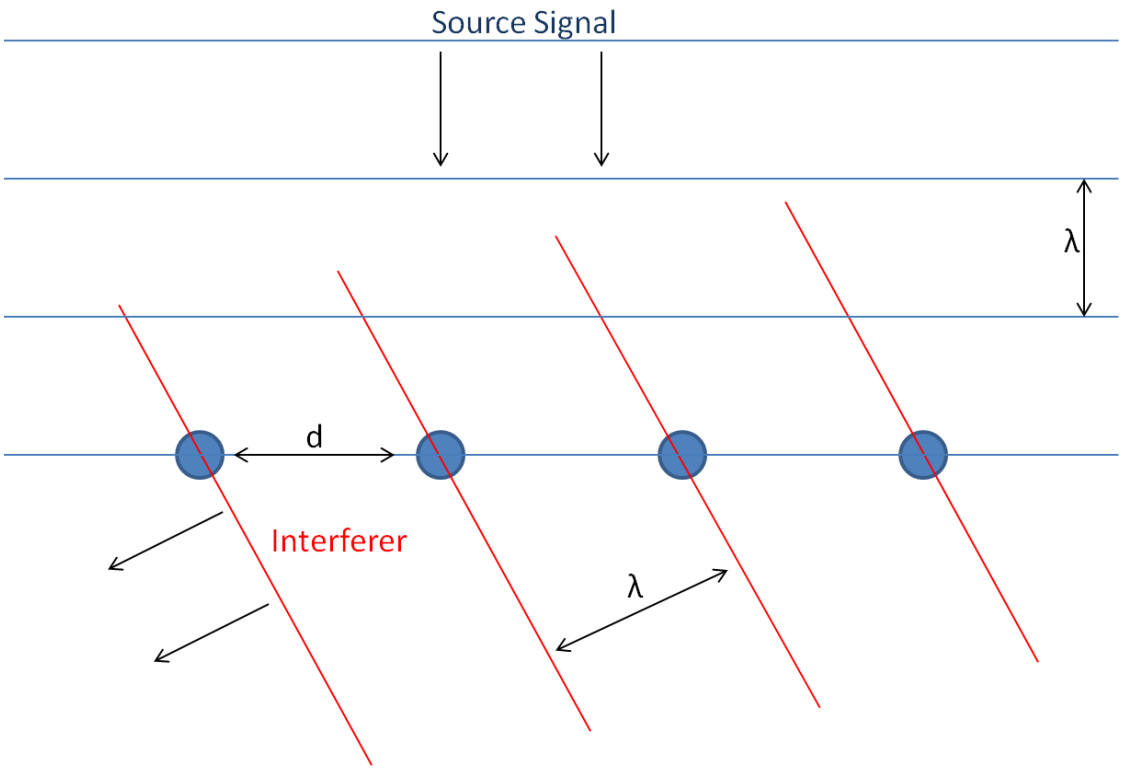

Figure 3.2 Illustration of spatial aliasing

Assuming the beamformer is designed to receive a signal coming from a DOA of 90^{o}, the applied time delays,τ_n, at each microphone would be zero according to equation 2.2. Because the array spacing is so large, there exists some other DOA where an interfering signal can line up in phase with the array elements and add together coherently, effectively passing through to the output. Even though it seems like this phenomena would only exist for periodic signals, it is an important observation to keep in mind when using the delay sum beamformer.

The second frequency dependant issue that arises from the delay sum beamformer is unwanted low pass filtering from errors in DOA estimation. This issue is important for the application of surveillance because the exact location of speakers may not be known. If the look direction of the beamformer is slightly off target, the speech signal will be low pass filtered which may inhibit speaker identification performance. This unwanted low pass filtering occurs because time delaying lower frequency, larger period, signals does not shift them enough out of phase to cause destructive interference. Higher frequency signals, on the other hand, take less time delay to cause destructive interference.

To illustrate this, the steered response of the delay sum beamformer is looked at over broad frequency range. The steered response is similar to the directional response but instead of keeping the look direction constant while moving the source, the source is kept constant while sweeping the beamformer look direction. This type of response can be useful for detecting where possible sound sources might be located. Figure 3.3 shows an example steered response using a 9 element linear array with an element spacing of 1 meter.

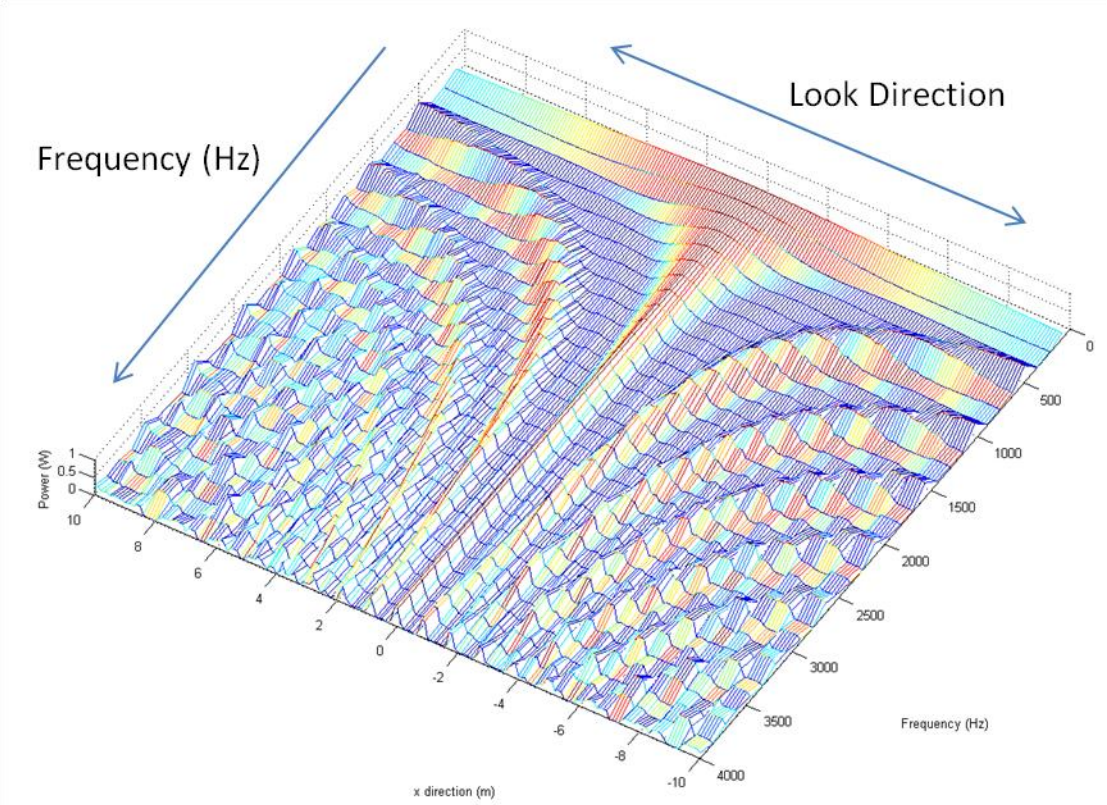

Figure 3.3 Delay sum beamformer frequency dependence

In the figure above, a sinusoidal source is located at 90^{o} (0 meters in the x direction). To obtain the frequency response of the delay sum beamformer, its steered response is swept along the horizontal plane while the source signal frequency is varied from 100Hz to 4 kHz using "DS_freq_resp.m". For each look direction and signal frequency, the output power of the delay sum beamformer is plotted in Figure 3.3. The concept of focusing error can be seen at the lower frequencies when the main lobe gets very wide. If the look direction of the beamformer is steered off of the actual source DOA just slightly, only the low frequency content will be recovered. Figure 3.3 also illustrates the effect of spatial aliasing and its affect over frequency. If the source at x = 0 meters is considered an interferer and the beamformer is not pointed towards it, it is possible for some of that interferer to pass to the output, especially if it contains high frequencies.

It is clear from the above simulations that frequency dependence is a possible limiting factor of using the delay sum beamformer with broadband signals such as speech. Though these factors may cause non-ideal performance, it is not clear how much affect they will have in practice. Intuitively, focusing error should be of most concern because of the uncertainty of source DOA estimation in surveillance applications.

3.1.2 Spatially Correlated Noise

Thus far only the delay sum beamformer's reaction to deterministic signals has been investigated. In order to predict possible short comings of the delay sum beamformer to a random noise environment, it is important to explore the effect of random noise fields on the performance of the delay sum beamformer. Because arrays consist of elements separated in space, the noise present is not only a function of time but a function of space as well. In order to simulate this spatially dependant noise, a noise model must be developed that takes into account the spatial correlation of the noise. This section therefore looks to derive a two dimensional spatially correlated noise model in order to see how random noise fields affect the delay sum beamformer.

To begin the derivation, a signal model is assumed to be a superposition of signal and noise at the output of each array element,

$$y_n(k) = x_n(k) + v_n(k). \qquad (3.1)$$

In order to find an expression for the spatial correlation of the noise, a model for $v_n(k)$ must be assumed. For this exercise, the noise is modeled as a sum of infinite plane waves with random amplitude, A, and direction, θ.

$$v_n(\overline{\boldsymbol{p}}, t) = \frac{1}{2\pi} \int_{-\pi}^{\pi} A(\theta)\, e^{j[\overline{\boldsymbol{k}}(\theta)\cdot\overline{\boldsymbol{p}} - \omega t]}\, d\theta \qquad (3.2)$$

Equation 3.2 shows the noise model in integral form where the vector $\overline{\boldsymbol{k}} = k(\cos\theta, \sin\theta)$ with $k = wavenumber = \frac{2\pi f}{c}$ and the vector $\overline{\boldsymbol{p}} = (x, y)$ with x and y being the two dimensional Cartesian spatial coordinates. The plane wave DOA, θ, is assumed to have a uniform distribution from –π to π.

In order to find the two dimensional correlation of the noise, the definition of correlation in equation 3.3 is used assuming $A(\theta)$is a zero mean random process.

$$Corr\,(v_n) = E[v_n(\overline{\boldsymbol{p_1}}, t_1) \cdot {v_n}^*(\overline{\boldsymbol{p_2}}, t_2)] \qquad (3.3)$$

The * operator in equation 3.3 denotes the complex conjugate. After substituting 3.2 into 3.3 and evaluating the expression, the correlation of the noise is found to be

$$Corr\,(v_n) = \frac{{\sigma_A}^2}{2\pi} J_0(kr)\cos(t_2 - t_1) \qquad (3.4)$$

where J_0 is the Bessel function of the first kind, r is Pythagorean distance between two points, and ${\sigma_A}^2$ is the variance of $A(\theta)$. Equation 3.4 enables modeling of the noise correlation between two receivers a distance r apart while taking into account the difference in time delays applied by the delays sum beamformer, $(t_2 - t_1)$.

From equation 3.4 an N x N correlation matrix can be constructed to simulate two dimensional spatially correlated noise. This is done with the Matlab function "plane_noise_R.m" which generates N channels of Gaussian white noise and correlates each noisy channel given a user specified array geometry. In order to see how the delay sum beamformer is affected by this spatially correlated noise, the steered response is

simulated with different arrays and noise parameters. The script “DS_noise_tests.m” was used to apply the delay sum beamformer to see how the noise frequency and array spacing affects the beamformed output.

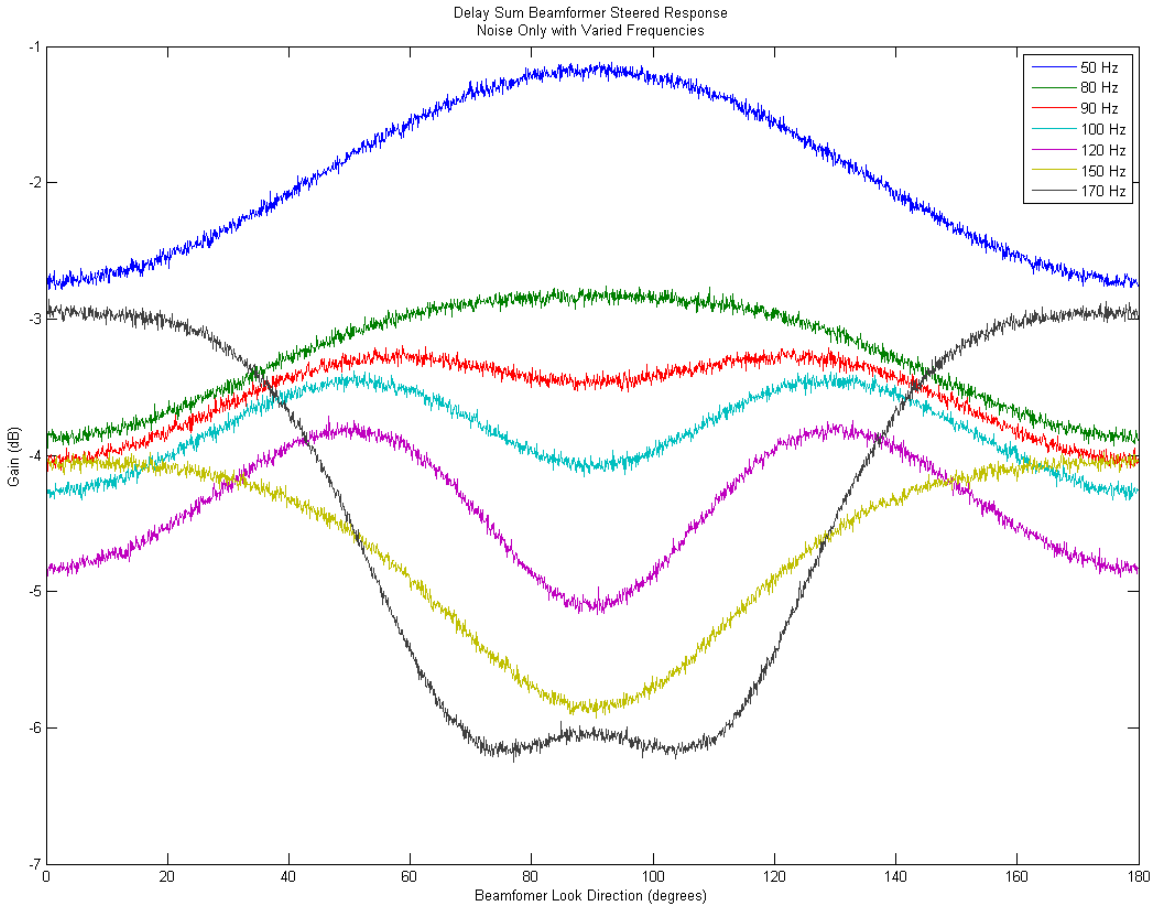

Figure 3.4 Delay sum beamformer steered response to spatially correlated noise

Figure 3.4 shows the delay sum beamformer’s steered response using a 3 element linear array with 1 meter spacing and only spatially correlated noise present. It is seen that as the beamformer look direction is swept, the output noise power changes. For some low frequency noise, the noise power peaks at 90 degrees because all time delays are zero which maximally correlates the noise. For higher frequency noise, the delay sum beamformer has the opposite effect. Keep in mind that the correlation also changes with

array spacing as well as frequency. This exercise of simulating the delay sum beamformer's reaction to spatially correlated noise highlights possible shortcomings in that noise attenuation varies with frequency, look direction and array geometry. This means that the beamformer's performance varies with each scenario and may achieve minimal noise reduction for certain look directions, both of which are not desirable in practice.

At first, the delay sum beamformer looks like a valid technique to reduce noise using an array because of its simple implementation and electronically adjustable look direction. Though it is possible to completely recover a broadband speech signal using this method, some frequency dependant issues may interfere with optimal beamformer performance as seen in this simulation section. Effects of spatial aliasing and unwanted low pass filtering from look direction focusing error vary with scenario and may cause performance degradation in practice. These issues should be kept in mind when processing broadband signals with the delay sum beamformer. Finally, a two dimensional noise model was simulated and showed that the delay sum beamformer could not guarantee consistent noise reduction for all look directions. It is obvious from the analysis above that more broadband methods should be explored to see if more robust performance can be achieved. The next chapter looks at some adaptive broadband multichannel noise reduction techniques which theoretically provide distinct advantages over the delay sum beamformer.

4. BROADBAND MULTICHANNEL NOISE REDUCTION ALGORITHMS

In Chapters 2 and 3 spatially selective filters or beamformers were analyzed and simulated as a possible means of noise reduction using microphone arrays. These techniques are well known due to their extensive use in practical applications such as radar. Because beamformers may not perform ideally in practice, it is useful to explore other types of array processing techniques that are broadband in nature due to their time domain implementation.

This chapter looks into some newer methods of microphone array processing to determine if a more suitable algorithm is available for surveillance scenarios. The two multichannel noise reduction algorithms explored are the Multichannel Wiener filter and the Spatio-Temporal Prediction filter found in [1]. Both algorithms exploit the spatial information provided by microphone arrays to reduce background noise. The two techniques are adaptive and therefore do not need any "a priori" information to use. This means that these algorithms don't require the location of the source or the relative positions of the microphones in the array and can adapt to different noise environments. These features make the adaptive algorithms ideal for use in surveillance where the sound source position is not always known and the noise environment changes.

4.1 New Problem Description

The Multichannel Wiener filter and Spatio-Temporal Prediction (STP) filter operate differently than the delay sum beamformer in the previous chapter. Therefore, a new signal model and algorithm goal must be presented. These techniques are broadband

in nature so they do not rely on time delays to achieve their filtering function. Instead these methods use single channel techniques generalized to multiple channels where each channel is treated as redundant observation of the same signal. The goal is to somehow combine this set of observation signals in such a way that reduces the noise at the output.

To illustrate this new concept, consider equation 4.1 where each microphone output of the array, $y_n(k)$, is composed of the desired speech signal to be recovered, $x_n(k)$, and noise, $v_n(k)$, sampled at discrete times k, using an N microphone array.

$$\begin{gathered} y_n(k) = x_n(k) + v_n(k) \\ n = 1, 2, 3, \dots N \end{gathered} \tag{4.1}$$

The main goal of these multichannel algorithms is to reduce $v_1(k)$ and recover a best estimate of $x_1(k)$ using N observation signals [1]. In order to achieve this goal, the signals are processed in blocks of L samples for later computations required by the algorithms. To represent these sample blocks or frames in the signal model, equation 4.1 is rewritten in vector form:

$$\begin{gathered} \boldsymbol{y}_n(k) = \boldsymbol{x}_n(k) + \boldsymbol{v}_n(k) \\ \boldsymbol{y}_n(k) = [y_n(k)\ y_n(k-1)\ y_n(k-2) \dots\ y_n(k-L+1)]^T \end{gathered} \tag{4.2}$$

In equation 4.2, the superscript T represents a vector/matrix transpose making $\boldsymbol{y}_n(k)$ a column vector consisting of the current observed sample, $y_n(k)$, in the first entry and the previous $L-1$ samples making up the rest of the vector. Block or filter length, L, is an important design parameter and will be varied later in the simulation and experiment sections. The speech vector, $\boldsymbol{x}_n(k)$, and the noise vector, $\boldsymbol{v}_n(k)$, that make up the noisy speech signal are defined in the same way as $\boldsymbol{y}_n(k)$ [1].

In order to recover the reference speech signal $\boldsymbol{x}_1(k)$ from the N observation signals, it is assumed that there is some linear transformation that can applied to the observation signals at each microphone to best estimate $\boldsymbol{x}_1(k)$. This transformation is shown and simplified in equation 4.3:

$$\boldsymbol{z}(k) = \sum_{n=1}^{N} \boldsymbol{H}_n \boldsymbol{y}_n(k) \tag{4.3}$$

$$\downarrow$$

$$\boldsymbol{z}(k) = \boldsymbol{H}\boldsymbol{y}(k)$$

where,

$$\boldsymbol{y}(k) = \boldsymbol{x}(k) + \boldsymbol{v}(k) = [\boldsymbol{y}_1{}^T(k)\ \boldsymbol{y}_2{}^T(k)\ \boldsymbol{y}_3{}^T(k) \ldots\ \boldsymbol{y}_N(k)]^T$$

$$\boldsymbol{x}(k) = [\boldsymbol{x}_1{}^T(k)\ \boldsymbol{x}_2{}^T(k)\ \boldsymbol{x}_3{}^T(k) \ldots\ \boldsymbol{x}_N(k)]^T$$

$$\boldsymbol{v}(k) = [\boldsymbol{v}_1{}^T(k)\ \boldsymbol{v}_2{}^T(k)\ \boldsymbol{v}_3{}^T(k) \ldots\ \boldsymbol{v}_N(k)]^T$$

$$\boldsymbol{H} = [\boldsymbol{H}_1\ \boldsymbol{H}_2\ \boldsymbol{H}_3 \ldots\ \boldsymbol{H}_N].$$

In the above equations from [1], $\boldsymbol{H}_n$is some LxL filter matrix corresponding to the observation signal vector, $\boldsymbol{y}_n(k)$. The simplified version of equation 4.3 is derived by organizing all of the filter matrices into one global filter matrix, $\boldsymbol{H}$, which is LxNL in dimension. The observation signals are also organized into an NLx1 column vector, $\boldsymbol{y}(k)$, containing all L samples of each observation signal vector concatenated into a single vector. The specific goal of these algorithms is to find this global filter matrix $\boldsymbol{H}$ to find the best estimate of the speech signal at the reference channel. The Multichannel Wiener Filter and STP filter both define criteria to calculate the global filter matrix given some optimization parameters. The specific criteria and methods for calculating the global filter matrix for each algorithm are described in the next sections.

4.2 Multichannel Wiener Filter

The Wiener filter is a well known classical optimal adaptive filter that was originally developed for single channel use. For our purposes, the Wiener Filter is generalized to multiple channels to be used in conjunction with microphone arrays. As stated in the previous section, the idea behind using multiple channels is that there are now multiple observable versions of the speech signal that can be combined in such a way to better estimate the desired speech signal than in the single channel case.

The Wiener Filter achieves its filtering function by utilizing the minimum mean squared error criterion. Specifically, the goal is to minimize the error between the estimated speech output signal and the desired clean speech signal. This mean squared error criterion, $J(\boldsymbol{H})$, is written in matrix form in equation 4.4 [1].

$$J(\boldsymbol{H}) = tr\{E[\boldsymbol{e}(k)\,\boldsymbol{e}^T(k)]\} \tag{4.4}$$

The error signal, $\boldsymbol{e}(k)$, is defined as the difference between the actual output of the filter and the desired speech signal, $\boldsymbol{z}(k) - \boldsymbol{x}_1(k)$. The "tr" operator represents the matrix trace operation. By substituting this error signal relation into 4.4, the mean squared error criterion is rewritten in equation 4.5 [1].

$$J(\boldsymbol{H}) = E[\boldsymbol{x}_1^T(k)\,\boldsymbol{x}_1(k)] + tr[\boldsymbol{H}\boldsymbol{R}_{yy}\boldsymbol{H}^T] - 2tr[\boldsymbol{H}\boldsymbol{R}_{yx_1}] \tag{4.5}$$

The NLxNL observation signal correlation matrix, $\boldsymbol{R}_{yy}$, is defined as $E[\boldsymbol{y}(k)\,\boldsymbol{y}^T(k)]$ and the NLxL observation-speech signal cross-correlation matrix, $\boldsymbol{R}_{yx_1}$, is defined as $E[\boldsymbol{y}(k)\,\boldsymbol{x}_1^T(k)]$. The minimum of the mean square error criterion is found by differentiating 4.5 with respect to $\boldsymbol{H}$ and setting it to zero [1]. Solving for $\boldsymbol{H}$ results in equation 4.6 [1]:

$$\boldsymbol{H}_W^T = \boldsymbol{R}_{yy}^{-1}\boldsymbol{R}_{yx_1}. \tag{4.6}$$

The global filter matrix in this case uses a 'W' subscript to specify that it was derived from the Wiener filter algorithm. It is important to note that $\boldsymbol{R}_{yx_1}$ cannot be calculated because $\boldsymbol{x}_1(k)$ is unobservable as it is the signal trying to be recovered. In order calculate the global filter matrix, $\boldsymbol{R}_{yx_1}$ must be estimated using the fact that the observation signals consist of clean speech and noise which are assumed to be uncorrelated. Assuming the noise can be observed by itself with no speech present, the cross-correlation matrix can be estimated using equation 4.7 [1].

$$\begin{aligned} \boldsymbol{R}_{yx_1} &= \left(\boldsymbol{R}_{yy} - \boldsymbol{R}_{vv}\right)\boldsymbol{U}^T \\ \boldsymbol{R}_{vv} &= E[\boldsymbol{v}(k)\,\boldsymbol{v}^T(k)] \\ \boldsymbol{U} &= [\boldsymbol{I}_{LxL}\boldsymbol{0}_{LxL} \dots \boldsymbol{0}_{LxL}] \end{aligned} \tag{4.7}$$

By subtracting the noise correlation matrix, $\boldsymbol{R}_{vv}$, from the observation correlation matrix, $\boldsymbol{R}_{yx}$ can be estimated for all channels. The LxNL $\boldsymbol{U}$ matrix is used to select out the desired cross-correlation matrix $\boldsymbol{R}_{yx_1}$. Using the relation in equation 4.7, the Wiener global filter matrix can now be calculated in 4.8.

$$\boldsymbol{H}_W^T = \left(\boldsymbol{I}_{NLxNL} - \boldsymbol{R}_{yy}^{-1}\boldsymbol{R}_{vv}\right)\boldsymbol{U}^T \tag{4.8}$$

As stated earlier, the global filter matrix can only be calculated if there are periods where there is only noise present allowing the estimation of the noise correlation matrix. Once calculated, the global filter matrix is applied to the observation signals using equation 4.3 to obtain the filtered output.

Though the Wiener filter seems like a promising approach, the spatial information provided by microphone arrays is not fully utilized. This is because it is a single channel

approach generalized to multiple channels. The next section derives a newer multichannel noise reduction algorithm called the Spatio-Temporal Prediction filter which takes advantage of the spatial information provided to calculate the global filter matrix.

4.3 Spatio-Temporal Prediction Filter

The previous section looked at the multichannel Wiener filter which is well known adaptive single channel algorithm generalized for use with multiple channels. Because the Wiener filter was designed for the single channel case, it may not exploit the spatial information provided by microphone arrays [1]. This section therefore derives a newer algorithm that was designed with multi-channel processing in mind. In fact, it can be shown that the Spatio-Temporal Prediction filter achieves no noise reduction if used in the single channel case.

The STP algorithm proposed in [6] exploits speech's predictability in time and space to clean the noisy speech signal. The STP filter is similar to the well known Linearly Constrained Minimum Variance (LCMV) filter and is implemented in two steps [1]. The first step involves calculating "optimal inter-sensor spatial-temporal prediction transforms". The second step then exploits these transforms to calculate the global filter matrix [6]. Specifically, the first step of the algorithm begins by assuming there is some linear prediction matrix $\boldsymbol{W}$ that can represent the speech signal at any channel using a linear combination of the reference speech signal samples:

$$\begin{aligned} \boldsymbol{x}_n(k) &= \boldsymbol{W}_n^T \boldsymbol{x}_1(k) \\ n &= 1, 2, 3 \ldots N \end{aligned} \tag{4.9}$$

The prediction matrix for each channel is calculated by minimizing the mean squared error caused by the transform as seen in equation 4.10.

$$J_W(\boldsymbol{W}_n) = E\left\{\left[\boldsymbol{x}_n(k) - \boldsymbol{W}_n^T\boldsymbol{x}_1(k)\right]^T\left[\boldsymbol{x}_n(k) - \boldsymbol{W}_n^T\boldsymbol{x}_1(k)\right]\right\} \tag{4.10}$$

By minimizing this function, the equation for each prediction matrix is found in 4.11:

$$\boldsymbol{W}_n^T = \boldsymbol{R}_{x_n x_1}\boldsymbol{R}_{x_1 x_1}^{-1} \tag{4.11}$$

A similar observability problem as the last section is seen in equation 4.11 in that $\boldsymbol{x}_1(k)$ is not observable in practice so it must be estimated by subtracting observation and noise signal statistics:

$$\boldsymbol{W}_n^T = \left(\boldsymbol{R}_{y_n y_1} - \boldsymbol{R}_{v_n v_1}\right)\left(\boldsymbol{R}_{y_1 y_1} - \boldsymbol{R}_{v_1 v_1}\right)^{-1} \tag{4.12}$$

Once they are calculated, all of these LxL prediction matrices are organized into an LxNL global prediction matrix $\boldsymbol{W}$ to simplify computations later:

$$\boldsymbol{W} = [\boldsymbol{I}_{LxL}\boldsymbol{W}_2\boldsymbol{W}_3 \ldots \boldsymbol{W}_N] \tag{4.13}$$

The first prediction matrix is an identity matrix because channel one is used as a reference to predict all the other channels. The fact that we are able to predict the speech signal at every channel using one channel is fundamental to the STP filter and will be utilized to calculate the global filter matrix. It is also important to point out that the global prediction matrix in equation 4.13 only needs to be calculated once if the source does not move [1] [6].

Assuming that the global prediction matrix is able to be calculated from the array observations, the global filter matrix $\boldsymbol{H}$ can now be calculated. The STP algorithm calculates filter weights by theoretically minimizing the noise power with the constraint

that the speech is not distorted. The noise power to be minimized is written in equation 4.14 where the filtered noise component of the output is defined as $\boldsymbol{e}_v(k) = \boldsymbol{H}\boldsymbol{v}(k)$.

$$J_{\boldsymbol{v}}(\boldsymbol{H}) = tr\{E[\boldsymbol{e}_v(k)\boldsymbol{e}_v^T(k)]\} \tag{4.14}$$

The speech distortion constraint is written in equation 4.15 so that the application of the filter matrix and prediction matrix do not distort the speech signal.

$$\boldsymbol{I}_{LxL} = \boldsymbol{H}\boldsymbol{W}^T. \tag{4.15}$$

By minimizing equation 4.14 with respect to 4.15 and using Lagrange multipliers, the expression for the global filter matrix is found in equation 4.16 [1] [6].

$$\boldsymbol{H}_{ST} = (\boldsymbol{W}\boldsymbol{R}_{vv}^{-1}\boldsymbol{W}^T)^{-1}\boldsymbol{W}\boldsymbol{R}_{vv}^{-1} \tag{4.16}$$

As seen in equation 4.16, calculating the STP filter's global filter matrix is more computationally intensive than the Wiener Filter as three matrix inversions are required compared to just one. The trade off here is that the STP filter implements the constraint that the speech is not distorted whereas the Wiener Filter causes more speech distortion proportionally to the amount of noise reduction achieved [6]. The theoretically minimal speech distortion that the STP approach provides is an attractive feature for the speaker identification application explored in this thesis.

5. SIMULATIONS

5.1 Algorithm Simulation

Now that the theory behind these newer multichannel noise reduction algorithms has been derived and understood, it is important to test their performance through simulation. Specifically, it is of interest to test how well these algorithms improve signal to noise ratio in different noise environments. The next few sections outline the filter implementation, noise modeling, as well as provide simulation results and interpretations.

5.1.1 Matlab Implementation

In order to test the performance of the algorithms in simulation and experiments, both the Multichannel Wiener Filter and Spatio-Temporal Prediction Filter are implemented as the Matlab functions “Wiener_filter.m” and “Spatio_Temporal_Filter.m” respectively. These functions operate using N channels of noisy speech and N channels of a noise sample (no speech present) as inputs. Filter length (L) and sample overlap are also entered as filter parameters which control the number of samples in each window as well as how much each window overlaps the previous one. Because the filter processes the data in blocks of L samples, this parameter can be varied to tune the filter. Figure 5.1 illustrates the block diagram implementation of the filters in Matlab.

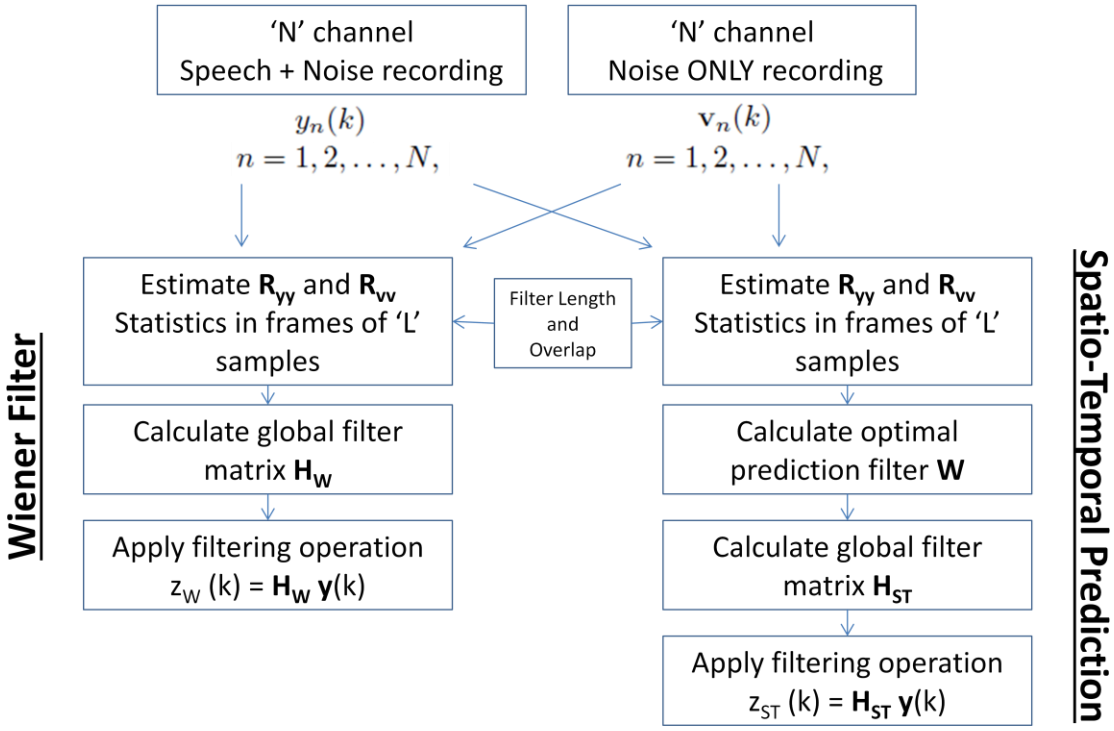

Figure 5.1 Matlab implementation block diagram of broadband multichannel filters

Though the algorithms differ in their computations, their overall implementation is fairly similar. Once the observation and noise only signals are obtained, the algorithms first calculate the correlation matrices $\boldsymbol{R}_{yy}$ and $\boldsymbol{R}_{vv}$ in blocks of L samples. This is achieved by taking the running average of the calculated correlation matrix for each frame according to equation 5.1 and 5.2 for the observation and noise signals respectively.

$$\boldsymbol{R}_{yy}{}' = \left(\frac{m-1}{m}\right)\boldsymbol{R}_{yy} + \frac{\boldsymbol{y}\boldsymbol{y}^T}{m} \tag{5.1}$$

$$\boldsymbol{R}_{vv}{}' = \left(\frac{m-1}{m}\right)\boldsymbol{R}_{vv} + \frac{\boldsymbol{v}\boldsymbol{v}^T}{m} \tag{5.2}$$

The original NxL observation and noise only signals are organized into NLx1 column vectors $\boldsymbol{y}$ and $\boldsymbol{v}$ in order to calculate these correlation matrices with a simple matrix multiplication for the current frame *m*. It is worth noting that equation 5.1 and 5.2 can be generalized to the problem of iteratively updating a variable by replacing the $\left(\frac{m-1}{m}\right)$ and $\frac{1}{m}$ factors with λ and $1-\lambda$ respectively. Using this scheme, is it possible to adjust the contributions of the old value (e.g. $\boldsymbol{R_{yy}}$) and the current value (e.g. $\boldsymbol{yy^T}$) to the new value (e.g. $\boldsymbol{R_{yy}}'$) in order to enhance or suppress the contribution of the current frame. Since one could spend a long time evaluating the effects of different updating schemes on filter performance, the running average method was chosen for simplicity.

Once the functions have run through both sets of input signals frame by frame to estimate the statistics, the global filter matrices are then calculated. The Wiener global filter matrix is calculated according to equation 4.8 while the STP global filter matrix is calculated in two steps using equation 4.12 and 4.16. Though it seems like the extra step of calculating the prediction matrix in the STP approach seems significant, it is actually fairly trivial because all of the information needed for equation 4.12 is embedded in $\boldsymbol{R_{yy}}$ and $\boldsymbol{R_{vv}}$ which must also be calculated for the Wiener Filter.

Once the global filter matrix is calculated, the last step is to perform the filtering operation. This is done using a simple matrix multiplication of the NLx1 column vector of the current frame with the global filter matrix. This multiplication results in an Lx1 vector which is used to construct the output signal frame by frame. This output signal is considered a best estimate of the speech signal at the reference channel (microphone 1 in this case).

5.1.2 Three Dimensional Noise Modeling

In order to verify the operation of these multichannel filtering methods, a noise model must be developed to provide a noise input to the adaptive filter functions in Matlab. As opposed to Chapter 3, this section will derive a three dimensional noise model because these filters are implemented using a three dimensional array.

After some research, it was found that a model was proposed by Cook et al. in [7] using a similar approach as was used to derive the two dimensional noise model in Section 3.1.2. The results from [7] provide an equation for spatial correlation for three dimensional plane wave noise containing a band of frequency content. This equation was simplified and rearranged to give a more implementable relation for the purposes of this thesis.

$$R(r) = \int_0^{\omega_c} \frac{\sin kr}{kr}\, d\omega$$
$$\downarrow \tag{5.3}$$
$$R(r) = \frac{Si(k_c r)}{k_c r}$$

Equation 5.3 gives the correlation between two points in space which are a distance *r* apart given noise with a cutoff frequency or bandwidth of $\omega_c = 2\pi f_c$, $k_c = \frac{\omega_c}{c}$. The speed of sound, *c*, is assumed to be 345 m/s in air. The function *Si* denotes the "Sine Integral" which is defined as: $Si(z) = \int_0^z \frac{\sin u}{u}\, du$. This expression for the spatial correlation of the noise can now be used to construct noise with these characteristics at each microphone in

the array. It is worth noting that $R(r)$ is normalized by k_c so that $R(0) = 1$ (note: $Si(z) \approx z$ for $z \ll 1$).

Noise with the correlation described by equation 5.3 is generated using the Matlab function "plane_noise_R_3D.m". The function first generates N channels of Gaussian white noise with user specified duration. This noise is then correlated in time using a Butterworth low pass filter of specified order and cut off frequency to establish the noise bandwidth. An NxN noise correlation matrix is then calculated according to equation 5.3, using the distance between each microphone as values for *r*. This matrix is then used to spatially correlate the low pass filtered noise through simple matrix operations. The resulting N channels of noise are now spatio-temporally correlated according to the above model. This output noise from this function is then used as the "noise only" input of the filtering functions for simulation.

5.1.3 Simulation Results

Now that a model for background noise has been developed in three dimensions, it is of interest to verify and evaluate the performance of the Wiener and STP filters in a simulation environment. The first set of simulations in this section will look at the ability of the filters to reduce noise with different bandwidths generated by the model in the previous section. The second set of simulations briefly explores how well the filters are able to reduce background noise recorded from a real environment. Filter length, L, will be used as the main adjustment parameter for each simulation in order to observe its affects. The goal of these simulations is to see what kind of noise reduction can be achieved by applying these multichannel filters.

In both simulations, a clean speech sample is simulated at a location in space using the Matlab function “sim_3D.m”. These N channels of clean speech are added to the appropriate background noise at a relatively low signal to noise ratio. These N channels of observation signals seen at each microphone are used as the noisy speech input to the filters, while the generated or recorded background noise is applied to the noise only input. The filtering functions use these two inputs to adaptively calculate and apply the best global filter matrix to the noisy speech input as derived in Chapter 4. The filtered single channel output will then be analyzed to see how much noise reduction was achieved.

In order to quantify and compare the performance of the Wiener and STP filters, output signal to noise ratio (SNR) will be used as the primary metric. More specifically, global SNR will be calculated because local or frame based SNR estimations methods may be sensitive to changes in frame length [8]. Due to the nature of this experiment, the clean speech signal is not observable at the output so its power cannot be calculated. Therefore, output SNR will be estimated under the assumption that the output signal is a superposition of filtered speech and residual noise. The output noise power can be estimated by applying the global filter matrix to the noise only sample and calculating its variance. The output speech signal power can then be estimated by taking the difference between the power of the filtered speech output and the filtered noise. Equation 5.4 analytically illustrates this SNR estimation method.

$$SNR_{out} = \frac{P_{sig_{out}}}{P_{noise_{out}}} = \frac{P_{total_{out}} - P_{noise_{out}}}{P_{noise_{out}}} \tag{5.4}$$

The Matlab function “BB_metrics.m” estimates this output SNR by calculating the variance of the total output and the filtered noise only output. It should be noted that this estimation technique assumes that the noise statistics are the same for the noisy speech signal and the noise only signal. This is a valid assumption for these simulations because the same noise is used to corrupt the speech signal that is also used to estimate noise statistics in the filtering functions. Non-stationary noise may prove problematic in the experimental portion of this thesis where these filters are applied to real data. Limitations of this SNR estimation technique will be discussed more in Chapter 5.

As mentioned, the first set of simulations looks to evaluate the performance of the Wiener and STP filters in the presence of the noise from the model developed in the previous section. For each simulation, noise of varying bandwidths are generated and added to a clean male and female speech source simulated 40 meters directly in front of the array. A relatively low input SNR of 0dB is used so that the speech and noise signals are competing when listened to.

The array used in this simulation consists of 9 elements arranged vertically in a 3 by 3 square with each microphone spaced roughly 1 meter apart (Figure 5.2). This is the same array geometry as the physical array used for the field experiments in the next chapter. In this simulation, both filters are applied to the simulated noisy speech with varying filter lengths to observe affects on the output SNR. The Matlab script “Wiener_Simulation_test1.m” is used to initialize and run iterations of this simulation with different noise bandwidths and filter lengths for both multichannel filters. The output SNR results of this set can be seen in Figure 5.3 and Figure 5.4.

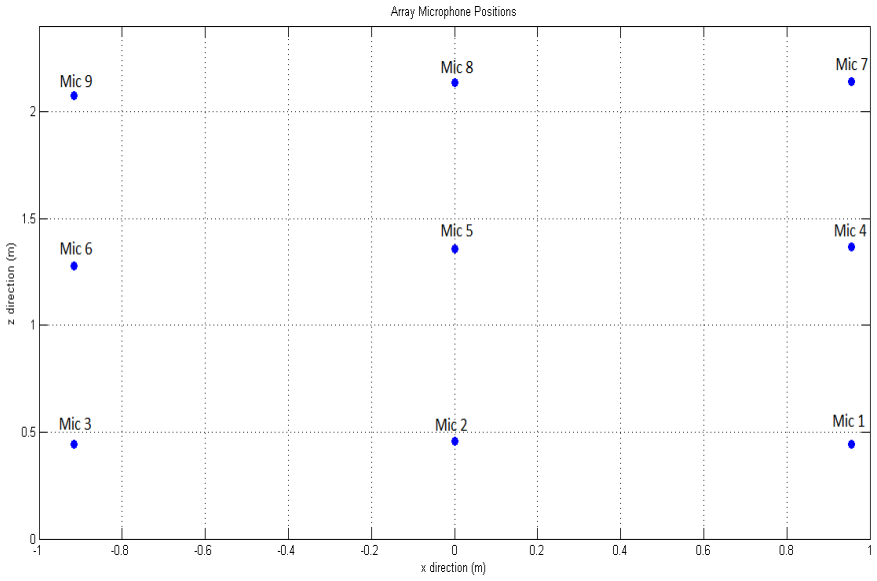

Figure 5.2 Simulation and Experimental Array Geometry

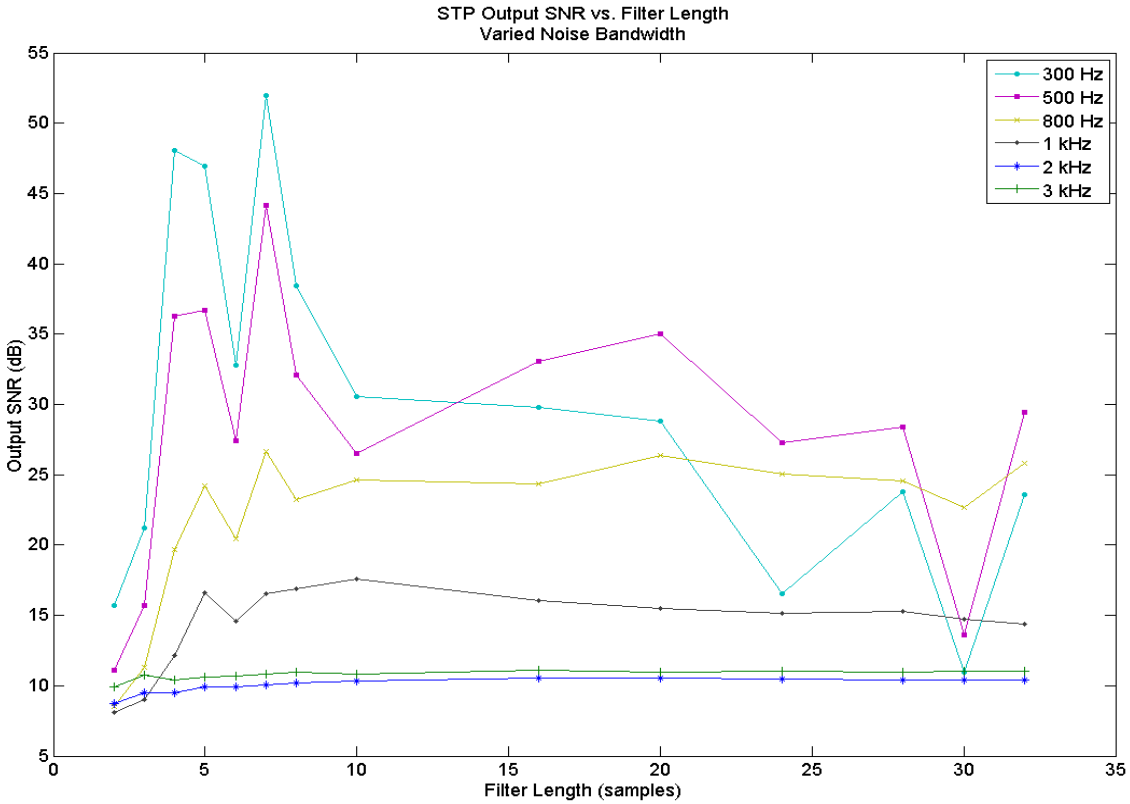

Figure 5.3 STP filter output SNR with varied filter length and simulated noise bandwidth

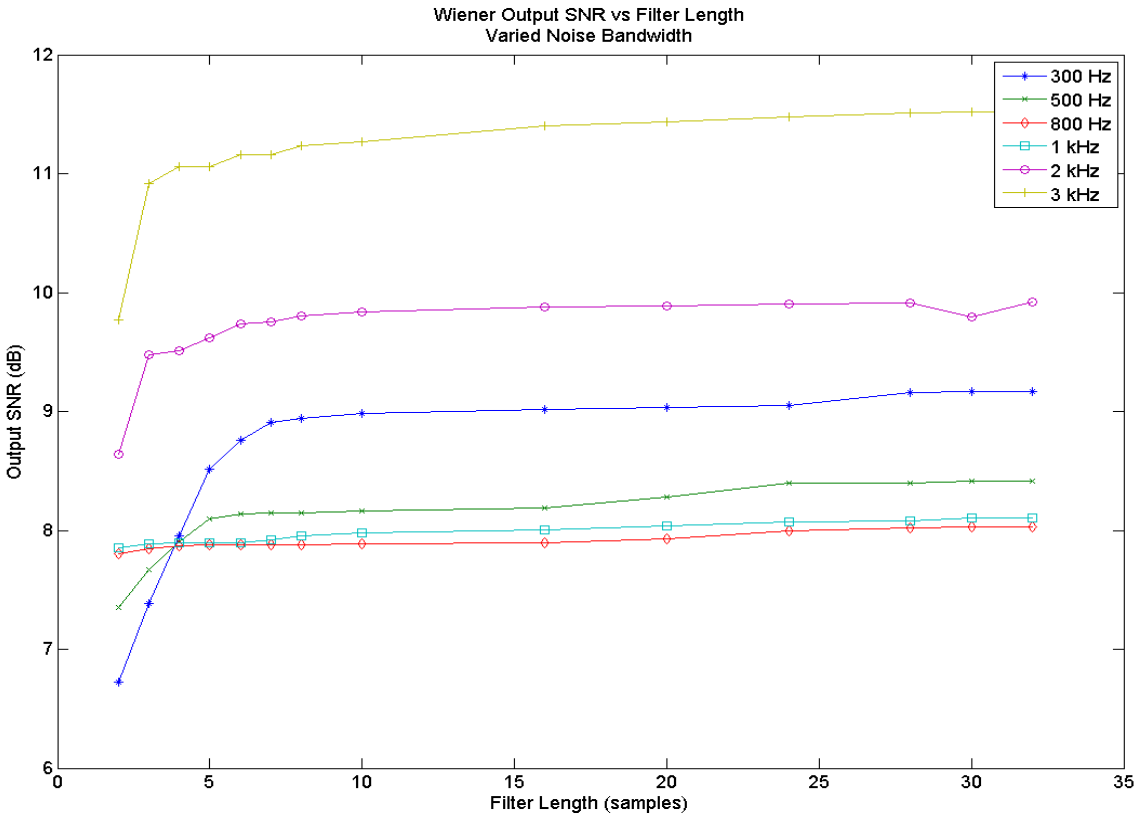

Figure 5.4 Wiener Filter output SNR with varying filter length and simulated noise bandwidth

It is clear that the output SNR performance of each filter varies significantly with noise bandwidth and filter length. The STP filter achieves a high output SNR for lower noise bandwidths and levels off around 10dB for higher bandwidths. This lower SNR may be contributed to the increased amount of frequency overlap of the speech and noise as the bandwidth increases. At lower noise bandwidths, the STP filter also exhibits more sporadic performance when the filter length is varied. The Wiener Filter, on the other hand, performs more predictably over filter length but has a significantly lower output SNR potential of 7-11.5 dB when considering all cases. From this simulation it is seen that both approaches achieve reasonable SNR gains with the STP filter showing the most potential for noise reduction.

The second set of simulations uses noise recorded in a real indoor environment to see how the performance of the multichannel filters changes with respect to the first simulation set. The noise used is a 1 minute recording of background noise inside an empty auditorium at Lawrence Livermore National Laboratory. The recording was performed using a physical microphone array with the same geometry as the previous simulation. The experimental set up and equipment will be discussed in more detail in the next chapter. The recorded background noise is added to the same clean speech source as before with an SNR of -3dB. Both filters are again applied to this noisy speech signal with varying filter lengths using the Matlab scripts "Wiener_Test3_filt_length_max overlap.m" and "STP_Test3_filt_length_max_overlap.m" for each filter respectively. The resulting output SNR performance can be seen in Figure 5.5.

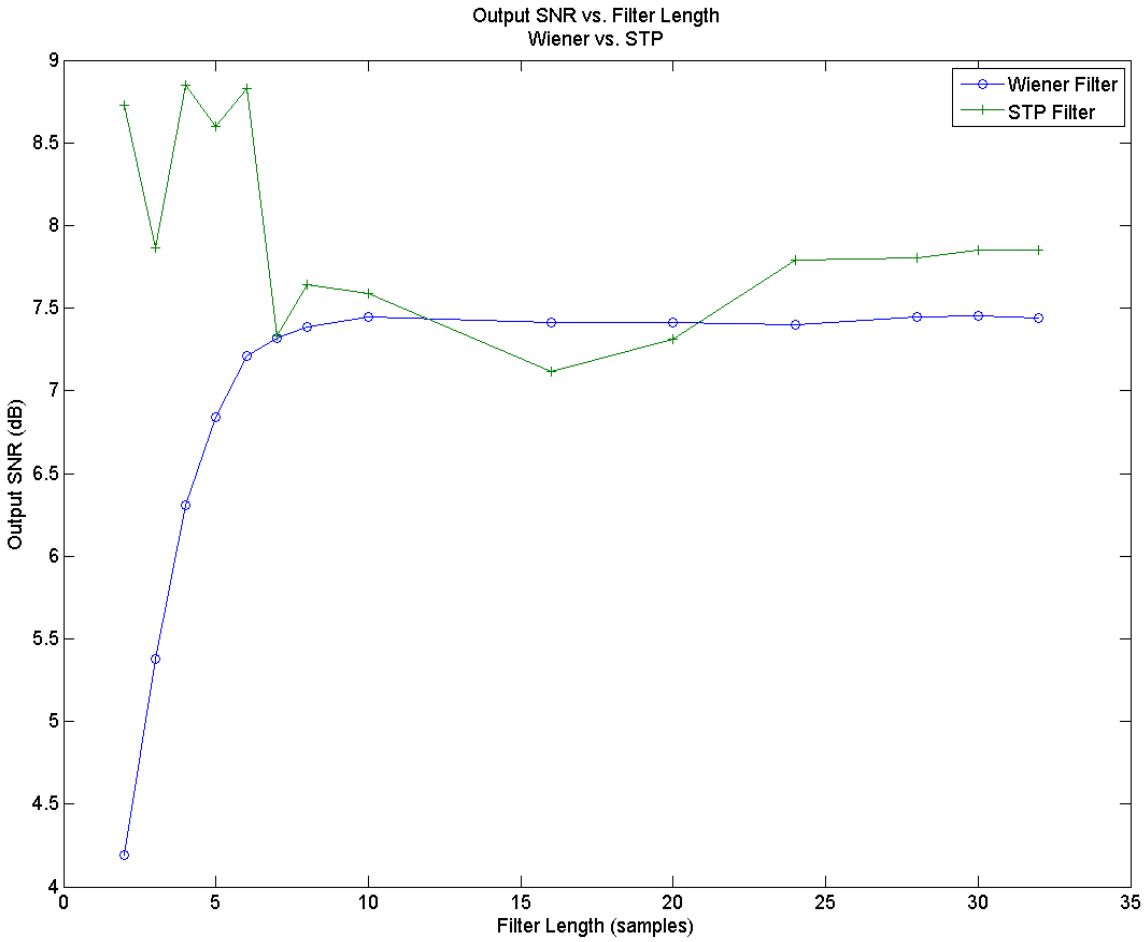

Figure 5.5 Output SNR of broadband filters using a simulated speaker in the presence of real indoor noise

Figure 5.5 exhibits similar trends to the previous simulation that are worth noting. First, the STP filter achieves higher output SNR at lower filter lengths and also exhibits similar seemingly unpredictable SNR trends as the first simulation. The Wiener filter also follows a similar pattern as the first simulation showing initially increasing SNR which levels off after a filter length of around 10 samples. The lower output SNR of both filters may be caused by the lower input SNR used in this simulation. Overall, the STP filter has consistently higher output SNR but tweaking its filter length does not guarantee a predictable increase or decrease in SNR as opposed to the Wiener filter.

Through the use of simulation in this section, it is clear that both filters are able to achieve noise reduction with different performance characteristics. The STP filter shows promise for greater output SNR while the Wiener filter exhibits a more predictable performance as the length of the filter is varied. The similarities between the two sets of simulations suggests that the generated noise is a reasonable model for real acoustic background noise in a three dimensional space. In the next chapter, these multichannel filtering algorithms will be put to the test using real field data recorded in various environments. It will be interesting to see how the performance of these filters in real life scenarios compares to what was observed in simulation.

5.2 Speaker Identification Simulation

In order to equate the results from the above simulations to speaker identification performance, this section explores the affect of SNR on SID confidence. The speaker identification software used in this thesis is the Advanced ID plug-in for the Rome Audio Processing Tool (RAPT-R). RAPT-R is an all purpose audio processing tool developed by the Air Force Research Laboratory for government and military use.

The Advanced ID plug-in enables target identification as well as cross verification using a variety of algorithms. The RAPT-R recommended SID system, Open-Set SID 2010, will be utilized to evaluate the performance of the noise reduction algorithms presented in this thesis. The Open Set SID system utilizes the Super Vector Classifier with “a set of feature extractors including GSV Linear Predictive Coding Coefficients (LPCC), Mel Frequency Cepstral Coefficients (MFCC), and Perceptual Linear Prediction

(PLP)" [9]. The goal of this system is to match an unknown speaker from an audio recording to a member of a target group of known speakers or "suspects".

In order to quantify confidence, or how well the unknown speaker matches a known speaker from the target group, the SID system assigns a numerical score to each member of the target group corresponding to how closely they match the unknown speaker. A more useful metric was developed for this thesis in order to better quantify SID in terms of overall confidence as well as score differentiation between speakers. This "SID discrimination metric" or SID score is calculated by simply taking the individual score of the actual speaker in the "unknown" recording and subtracting the next highest speaker score from it. For example, if the actual speaker in the recording scores a 1.5 and the highest scored speaker from the rest of the target group receives a 1.0, the SID discrimination metric is calculated to be 0.5. This calculated metric is shown in equation 5.5 and will herein be referred to as the "SID score" or "SID metric" not to be confused with the "confidence score" given to each member of the target group by the RAPT-R program.

$$SID\ Score = Actual\ Speaker\ Score - Highest\ Score\ of\ Other\ Speakers \quad (5.5)$$

The SID metric in equation 5.5 is useful because it takes into account the relative confidence scores of other targets rather than just the absolute confidence score of the actual speaker. This differential approach makes it so that a high SID score is calculated if the actual speaker receives a much higher confidence score than the other targets, while

a low SID score is calculated if the actual speaker has a similar score to other targets. A negative score is calculated if the SID system identifies the wrong target as the unknown speaker. This case corresponds to the actual speaker receiving a lower confidence score than another target. The SID metric developed here will be used in this simulation section as well as in the next chapter to compare the performance of the different noise reduction algorithms.

In order to test the SID system with various speaker input cases, Lawrence Livermore Lab employees were used to compile a database of 13 speakers. Both male and female speakers were used in the dataset including individuals with slight accents. Each speaker in the database was recorded in a low noise, office room environment using a computer microphone. Speakers are recorded uttering multiple phrases taken from TIMIT prompts. These phrases are used because they are “phonetically diverse” and were developed for use with speaker recognition research [10]. A list of these phrases can be found in Appendix A. Speakers in the database are labeled alphabetically from ‘A’ to ‘M’ due to privacy restrictions.

Before testing the SID system with different inputs, various setup steps must be performed to calibrate the system. A Universal Background Model or UBM must first be generated using as many speakers as possible in order to train the system on the channel characteristics, speaker/population mix (male or female) and environment of the speech recordings [9]. The Advanced ID manual emphasizes that speakers used to create a UBM should never be used as targets to be matched on. The UBM for simulation and later experiments was created using 8 of the 13 speakers available in the Lawrence Livermore National Laboratory speaker database designed for this research. Next, a target group is

constructed using the remaining 5 speakers in the database. Speaker identification can then be performed by running an "unknown" speaker wav file against all of the speakers in the target group using Open Set SID 2010. The output of the system is a set of 5 confidence scores corresponding to how closely each speaker in the target group matches the unknown speaker. From these scores, the SID discrimination metric is calculated by hand using equation 5.5. The steps used to perform speaker identification are outlined in Figure 5.6.

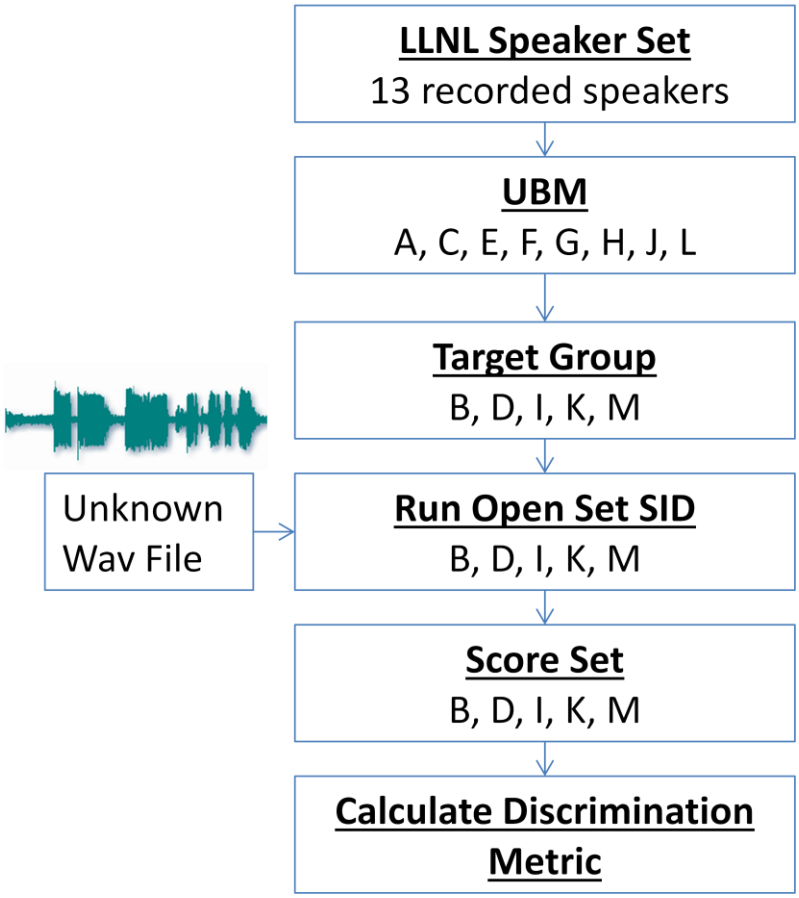

Figure 5.6 Algorithm for running speaker identification system

With the UBM and target group set up, it is important to see how corruption of the speech signal with noise and distortion affects SID scoring. To examine the influence of

noise, the clean speech sample for each of the speakers in the target group is corrupted with different levels of additive Gaussian white noise. These noisy speech signals are then passed through the SID system and scored. The SID results of this simulation are shown in Figure 5.7 using various input SNR levels for each speaker. The average score over all speakers is also plotted versus input SNR in Figure 5.8.

When looking at SID scoring, it is important to note that any score above 0 is a positive identification while anything below 0 is a negative identification, meaning the SID system identified the wrong individual. Within the positive scoring area, any score from 0-0.5 is considered a “weak positive”, 0.5-1.0 is a “moderate positive” and scores above 1.0 are “strong positives” indicating a highly confident identification. A moderate positive should be considered fairly sufficient while a weak positive should be interpreted with caution.

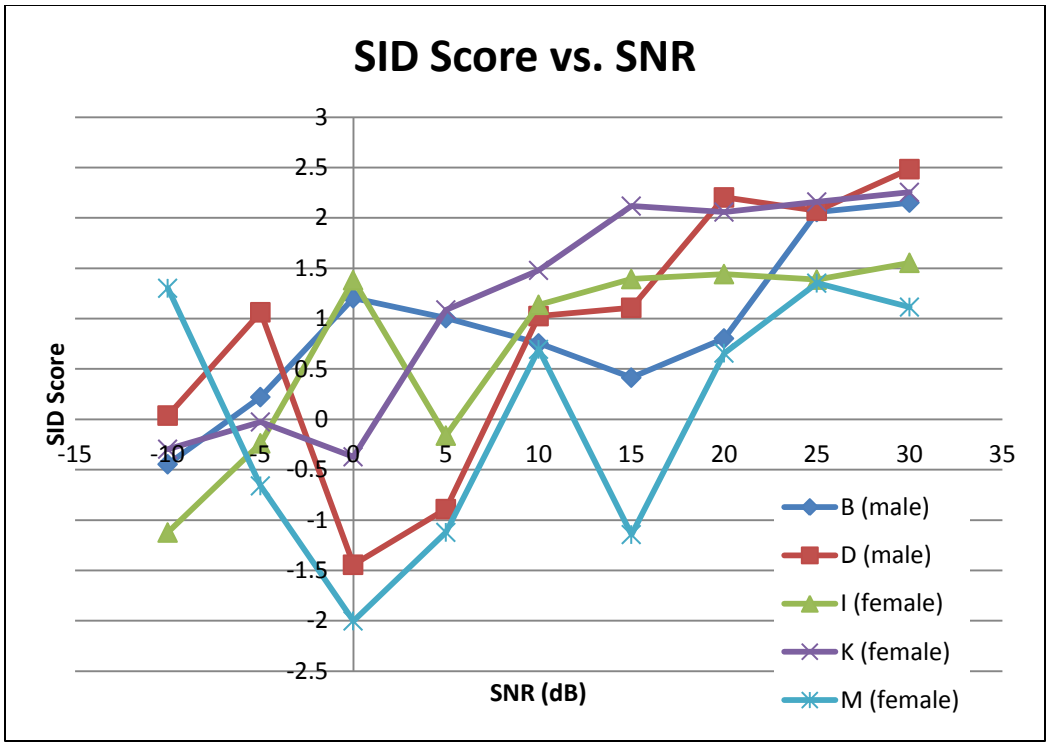

Figure 5.7 SID score vs. SNR for the 5 speakers in the target group

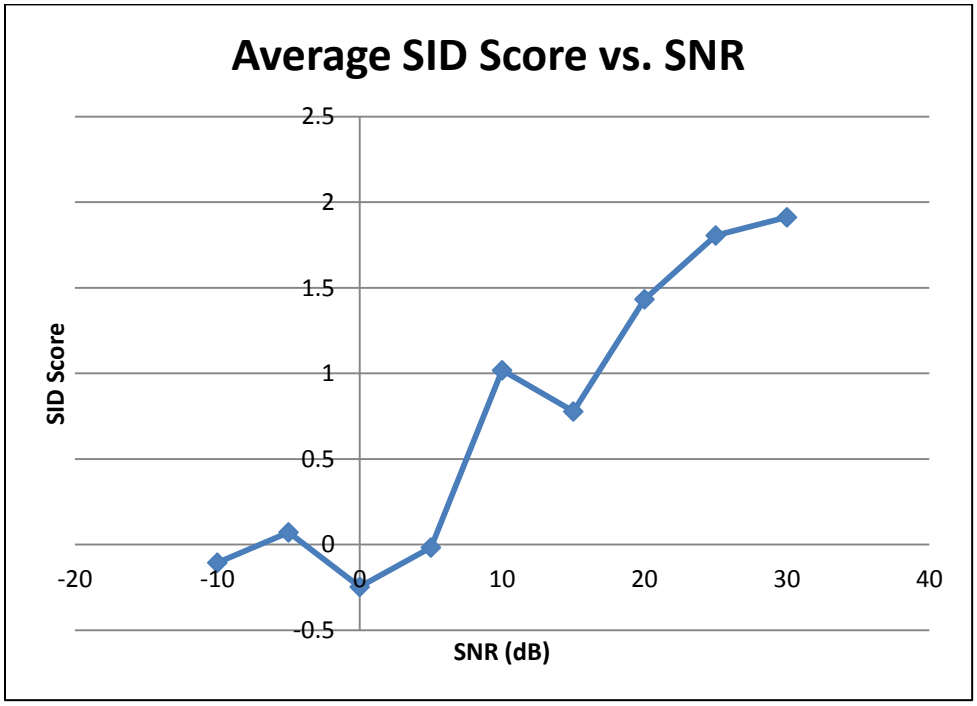

Figure 5.8 Average target group SID score vs. SNR

As seen in the figures above, SID score does seem correlated with SNR but trends do fluctuate between speakers. When all speaker scores are averaged for each noise level, it is clear that SID score does increase with SNR. From Figure 5.8, it seems that about a 10 dB SNR is needed to achieve a positive identification. It is important to keep in mind that the effect of SNR does vary from speaker to speaker so the trend may not always hold for every individual case. Though SNR seems like a good performance predictor for speaker identification, it is not the whole story.

Distortion of the speech signal caused by filtering may also cause performance limitations in speaker identification scoring. Signal corruption via distortion is simulated by taking the clean speech samples and applying band pass filters of varying bandwidths to them. The lower the bandwidth of the filter, the more “speech distortion” is applied by altering the frequency content of the speech. Figure 5.9 plots SID score versus the bandwidth of the band pass filter applied to each clean speech signal.

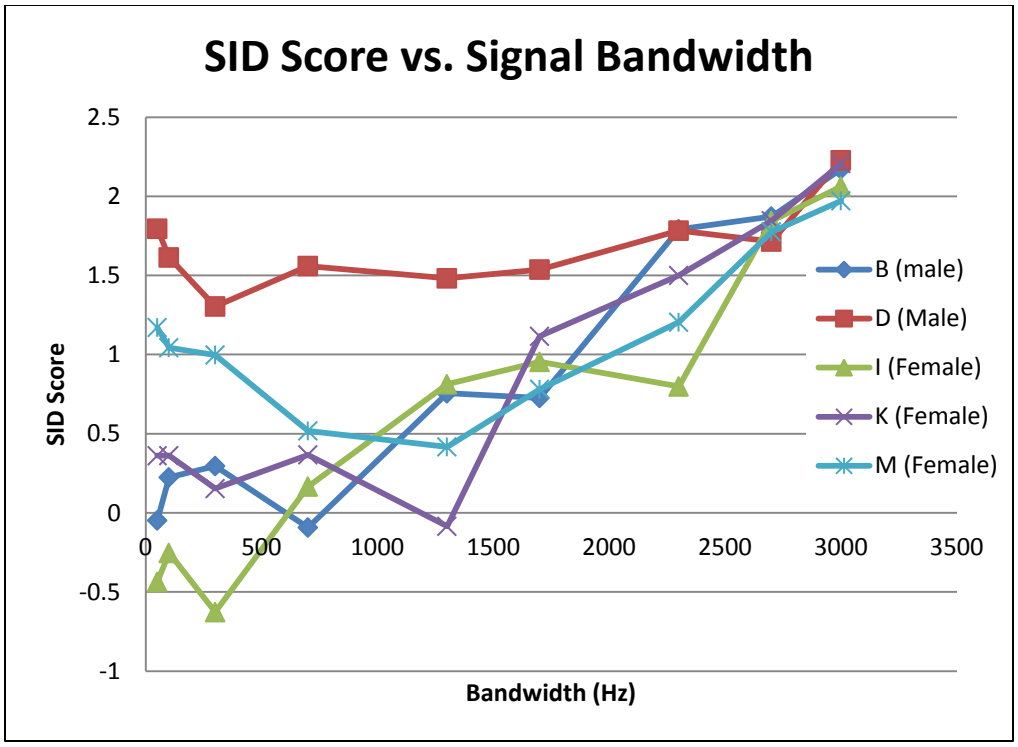

Figure 5.9 SID score vs. speech signal bandwidth for the 5 target group speakers

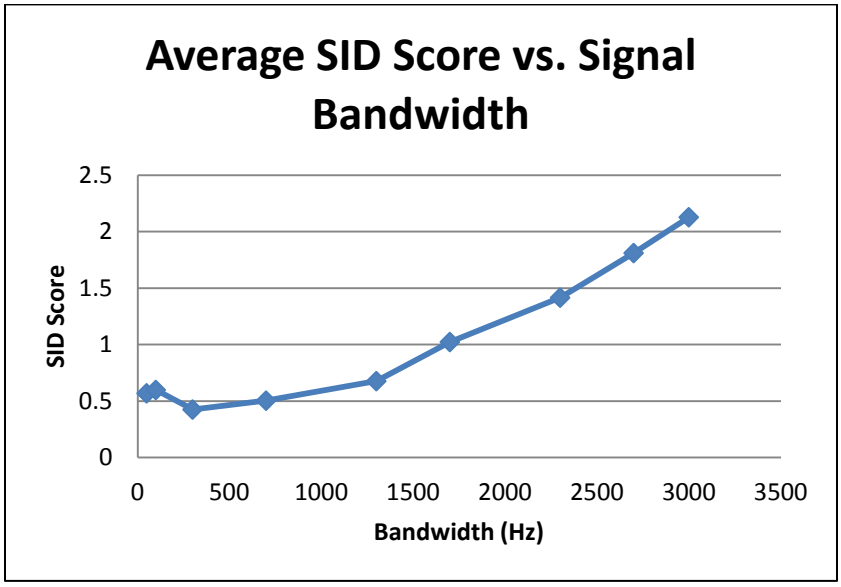

Figure 5.10 Average SID score vs. speech signal bandwidth of the target group

As expected, altering the frequency content of the speech does affect the SID score. As the bandwidth of the clean speech is decreased, the SID score also decreases on

average. Like the SNR simulation, this trend varies from speaker to speaker. For example, speaker ‘I’ is directly affected by this simulated speech distortion while speaker ‘D’ is barely affected and maintains a strong positive score. This speaker variation may be due to the differences in the location of the strongest frequency content for each speaker relative to the location of the center frequency of the band pass filter. Overall, speech distortion does seem to affect SID scoring but does not have as drastic of an effect as SNR.

The effect of speech distortion is explored here because the noise reduction algorithms are not perfect and may distort the speech signal while reducing noise. Therefore, it is important to know its effect on SID scoring if distortion does occur. Due to the adaptive nature of the noise reduction algorithms, it is difficult to estimate their effect on frequency content for each speaker case. In the experiments in the next chapter, speech distortion measurement was attempted using theoretical approaches from [1], but did not result in reasonable values. Because only experimental output SNR measurements will be available for comparison, the findings of this speech distortion simulation should just be kept in mind when analyzing the results of experiments in the next chapter.

Through various simulations in this chapter, the use of multichannel noise reduction algorithms has proven to be a reasonable approach to aid in speaker identification. Section 5.1 verified that noise reduction is possible using multichannel techniques for different types of noise, including noise recorded in a real environment. It was seen that the STP filter had the highest output SNR potential, while the Wiener Filter performed more predictably as the length of the filter was varied. Section 5.2 demonstrated the connection between SNR and SID scoring for the white noise case. Finally, the effect of

speech distortion on SID performance was investigated and should be noted as a potential performance inhibitor. The next chapter looks to put the multichannel filters to the test in real environments to see if they can indeed enable speaker identification in practice.

6. EXPERIMENTAL RESULTS AND ANALYSIS

Thus far, this thesis has presented three multichannel noise reduction techniques with the goal of aiding speaker identification algorithms. The delay sum beamformer, multichannel Wiener filter and Spatio-Temporal Prediction filter were all derived and their performance was investigated using simulation. Now that the proper operation of each approach has been verified and potential shortcomings have been noted, this thesis will conclude by applying these filters to multichannel speaker data from real environments.

The results of three data collections will be presented in this chapter as a final performance test of these multichannel noise reduction techniques. Of the three data collections, one was performed indoors while the other two were performed outdoors to create a more "challenging" noise environment. During each experiment a nine element microphone array is used to record up to five speakers in various locations. Output SNR and ultimately SID scoring will be used as metrics to judge the performance of each filter. The overall goal is to compare the filters and see if one approach is optimal under certain conditions [11]. The next section describes the experimental set up and procedure in greater detail.

6.1 Experimental Set Up and Procedure

Since the goal of this thesis is to see if a speaker identification system can be improved with multichannel pre-processing for practical surveillance applications, it is important for these experiments to mimic real scenarios while still controlling as many variables as possible. Each experiment uses speech recordings of individuals played back

through computer speakers on a mobile cart. The “speech source” or cart location is varied and differs for each data collection. Using recordings rather than live speakers allows for consistent control of speaker volume, position and speech content for each trial. Different numbers of speakers, source locations and trials are used for each data collection and are specified in their respective sections below. Each experiment also uses different methods to sample background noise for the adaptive filters. While each experiment differs slightly in procedure, they all follow a similar outline and use the same equipment.

In order to properly record and store speech data from experiments, a microphone array and data acquisition system (DAQ) is utilized. Figure 6.1 shows the array, constructed at LLNL for this research, which consists of nine elements arranged in a 3x3 pattern with each microphone spaced roughly 1 meter apart. The custom array frame enables mounting of nine Brüel & Kjær 4958 20 kHz Precision Array Microphones on adjustable microphone holders (Figure 6.2). These microphones feed to the Abacus DAQ made by Data Physics which streams the data to a Panasonic Toughbook computer via Ethernet (Figure 6.3). The nine channels of data are converted to Matlab (.mat) format for post-processing back in the lab. After applying the filtering techniques presented in this thesis, output metrics such as output SNR and SID score are calculated using Matlab and RAPT-R respectively. From there, the performance of the algorithms will be compared and analyzed.

Figure 6.1 Microphone array used for experimental data collections [11]

4958 20 kHz Precision Array Microphones (Brüel & Kjær)

- Frequency Range: 10 – 20,000 Hz
- Dynamic Range: 28 – 140 dB
- Built in preamplifier
- 34 mm long, 7 mm diameter

Figure 6.2 Array microphones and specifications [11]

DAQ system: ABACUS by DataPhysics
- 32 channels
- Dedicated 24 bit ADC per channel
- Real-time signal processing

Figure 6.3 ABACUS Data Acquisition System and specifications [11]

6.2 Data Collections

In this section, the three data collections performed at Lawrence Livermore Lab are outlined and the results analyzed. Each data collection has a different motivation as well slight procedural nuances which are described in each sub section. In general, several speakers are recorded at various locations using the microphone array. Back in the lab, the filters are applied to each multichannel speech recording and the output SNR is calculated. Next, the single channel output of each speech recording is subjected to speaker identification using the Advanced SID program in the RAPT-R software. The SID score is then calculated manually for every speaker. The results are plotted and analyzed and conclusions are derived. The goal of these experiments is to see if these multichannel filters can, in fact, enable positive, more confident identifications than the raw recordings.

6.2.1 Outdoors 5/10/2011

6.2.1.1 Motivation and Procedure

The first data collection was recorded on May 10, 2011 in an outdoor environment at Lawrence Livermore National Laboratory. The location of the collect was on the North West corner of the LLNL campus near the corner of Vasco and Patterson Pass Road. The proximity to these busy streets provided a realistic traffic noise environment. It is important to note that this collect was not performed for the purposes of this thesis but for a related project. The data from this experiment was used for this thesis to provide preliminary results to get an idea of what to expect out of the filters and speaker identification system.

Two speaker sources were used in this collect, one male and one female. The position of the speech sources was varied straight out in front of the array at distances of 10 ft., 25 ft., 50 ft. and 100 ft. using computer speakers mounted on a mobile equipment cart. The array and source positions are plotted in Figure 6.4. At each position, both speech samples are played back at a moderate volume. Speakers ‘D’ (male) and ‘I’ (female) are used in this experiment. A one minute sample recording of the noise environment was taken at the beginning of the experiment for later analysis and use in the adaptive filtering algorithms. Noise sources in this environment include wind, trees rustling, traffic, airplanes, generators and birds chirping. The output SNR and SID scoring results are provided in the next section.

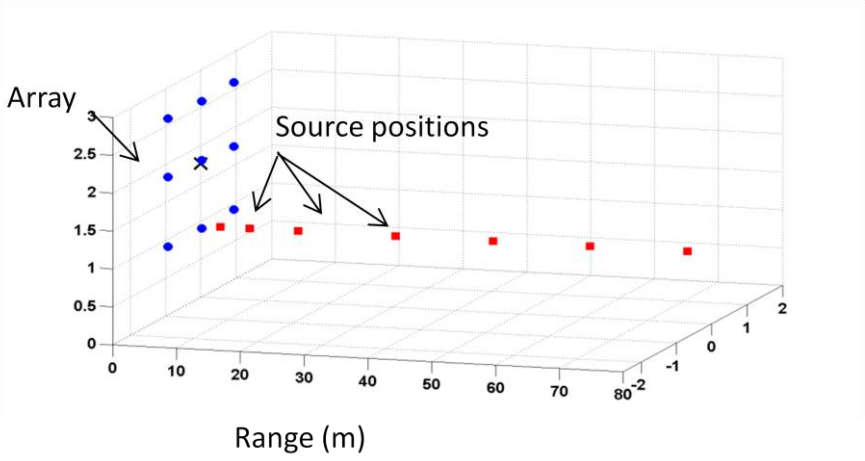

Figure 6.4 Microphone (blue) and speech source positions (red) for outdoor 5/10/2011 experiment [11]

6.2.1.2 Results

Plotted below are the results from the outdoor data collection performed on 5/10/11. Output SNR is plotted first versus filter length for each speaker using the Wiener and Spatio-Temporal Prediction Filter. Speaker cases where output SNR was not measureable are not included in the plots. The output SNR plots are included to examine any correlation between SNR and SID score. A table of calculated input SNRs and delay sum beamformer output SNRs is also provided below. Any cases where SNR was unable to be estimated are marked as "N/A" and are omitted from the figures below.

Table 6.1 Input SNR and delay sum output SNR for each speaker and distance case from 5/10/11

Distance from array (ft)	Male Speaker		Female Speaker	
	Measured Input SNR (dB)	Delay Sum Output SNR (dB)	Measured Input SNR (dB)	Delay Sum Output SNR (dB)
10	-4.5942	-1.549	-17.258	N/A
25	N/A	N/A	-4.266	-4.038
50	-1.7698	-2.660	-7.525	-8.564
100	.1444	0.440	N/A	-9.615

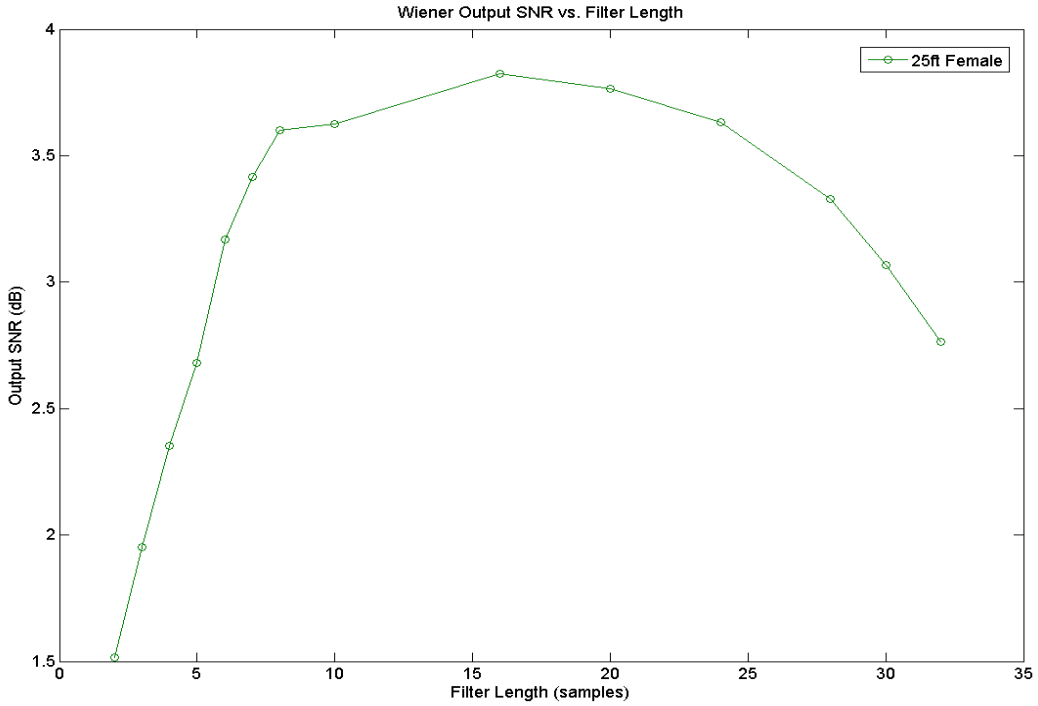

Figure 6.5 Wiener output SNR vs. filter length for Outdoor 5/10/11 female data

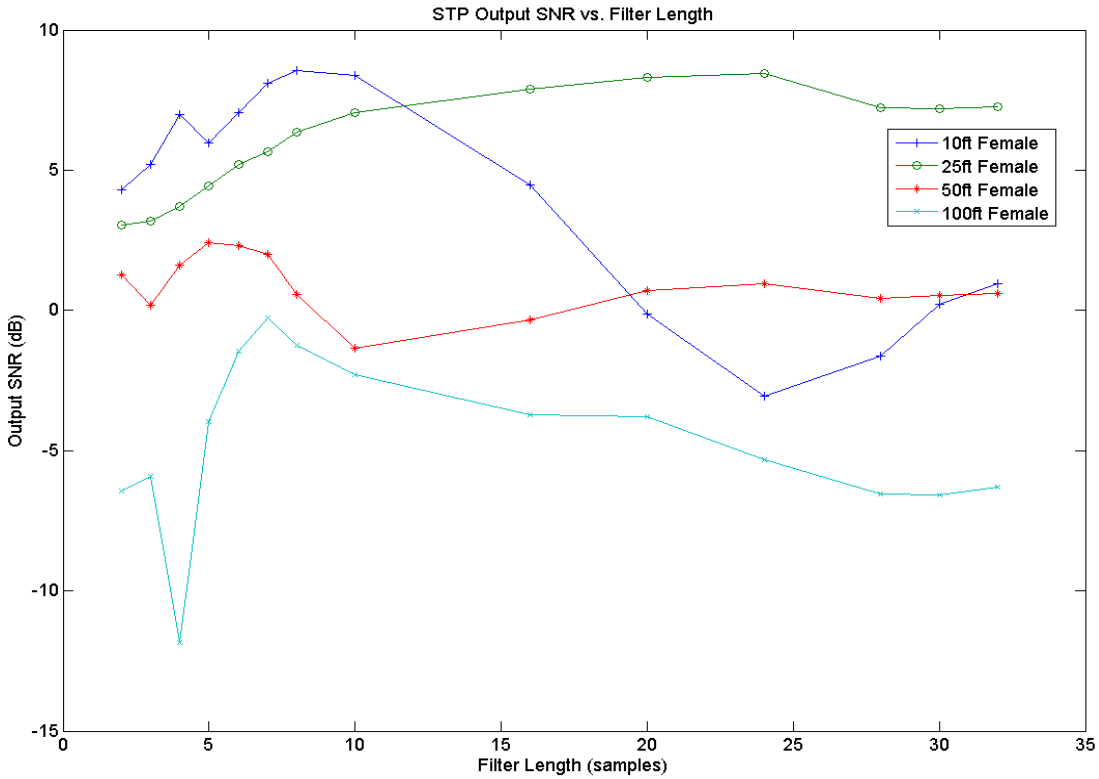

Figure 6.6 STP filter output SNR vs. filter length for Outdoor 5/10/11 female data

In the figures above, it is seen that the output SNR of the Wiener Filter for the female speaker was not measureable for all cases except at 25 feet. For the 25 ft. case, the output SNR increases with filter length but starts to drop off after 20 samples. This is in contrast to the simulation in the previous chapter where the output SNR stayed fairly stable for longer filters. The Spatio-Temporal Prediction filter on the other hand, shows more promise. All cases for the female speaker produced measureable output SNR and are larger in magnitude than the Wiener filter for all filter lengths. Like in simulation, there does not seem to be an extractable trend for the STP filter as the filter length is varied. For output SNR, it seems that a lower filter length is better for the 10, 50 and 100 feet cases while a longer filter length is better for the 25 ft. case. At this point, it is useful to examine the results of the male speaker and note any similarities or differences to the female case.

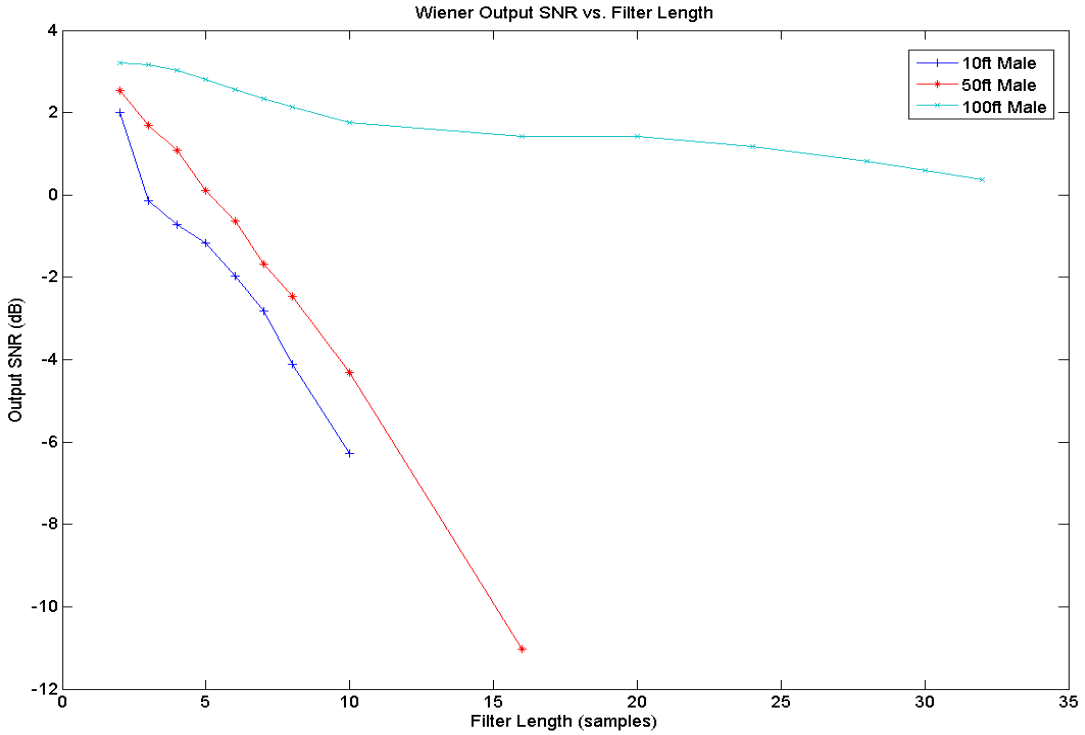

Figure 6.7 Wiener filter output SNR vs. filter length for Outdoor 5/10/11 male data

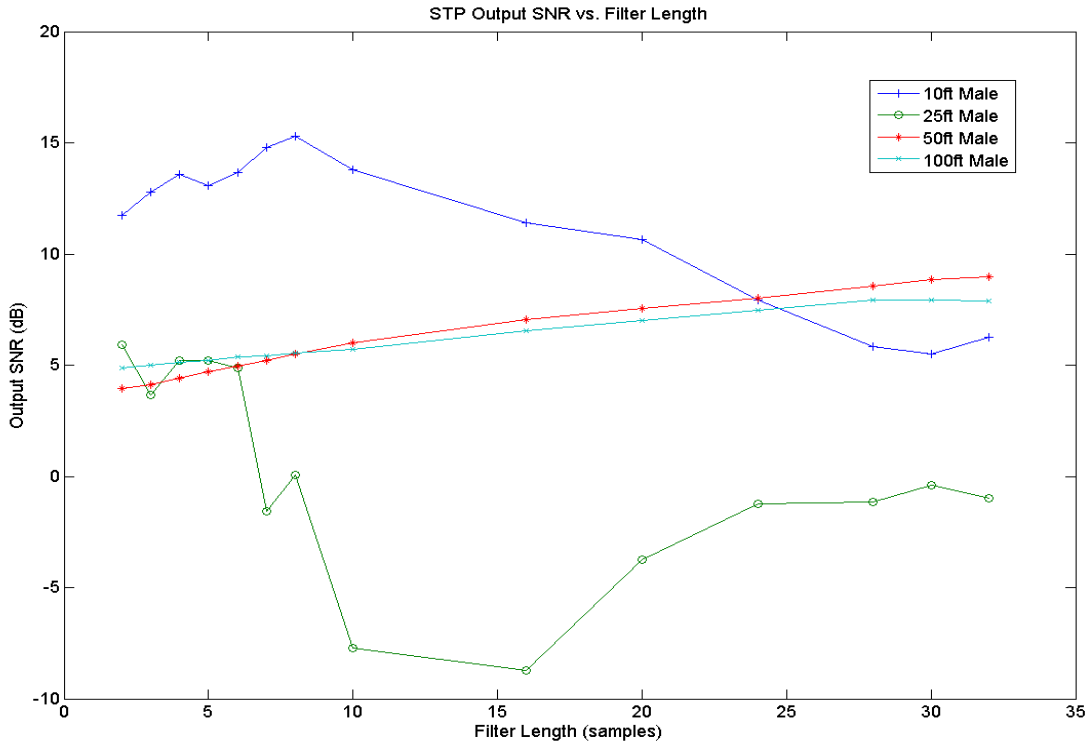

Figure 6.8 STP filter output SNR vs. filter length for Outdoor 5/10/11 male data

For the male speaker case, the Wiener Filter still has difficulties while the STP filter achieves even better output SNR performance. This time, all cases produced measureable output SNR except the 25 ft. case for the Wiener Filter. For each case, output SNR decreases almost linearly with filter length. This result greatly contrasts with earlier simulations where output SNR increased and then leveled off as filter length increased. The STP filter again achieves much higher output SNR than the Wiener Filter but varies in performance between cases. For the 50 and 100 ft. cases, output SNR increases almost linearly with filter length while the 10 and 25 ft cases perform best at lower filter lengths. At this point, it seems that the performance trends of these filters is highly dependent on speaker and position cases.

The second set of plots below show the speaker identification scores achieved by each filtering approach for each speaker and position. For the adaptive filters, the maximum SID scores achieved for all filter lengths used are plotted. Because the delay sum beamformer does not have an “adjustable parameter” like filter length, its one and only SID score is plotted for each position. For comparison, the score achieved by the raw data from a single microphone in the array (microphone 1) is also plotted.

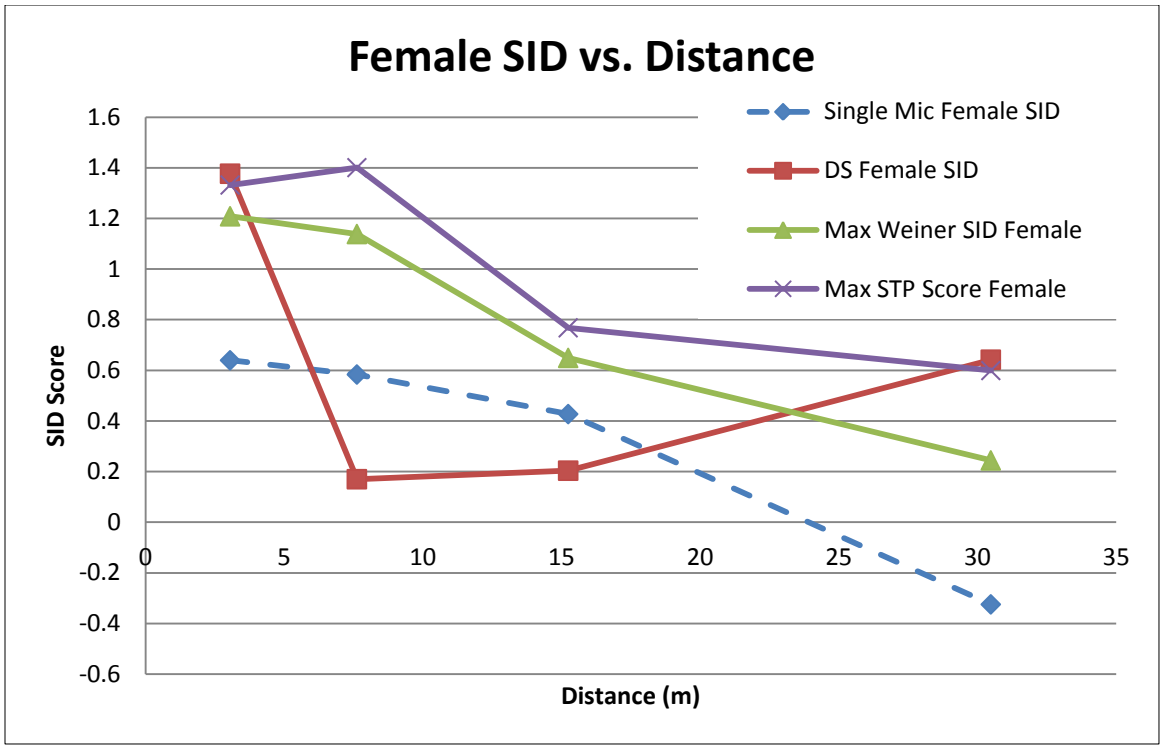

Figure 6.9 Female SID score vs. distance position for Outdoor 5/10/11 data

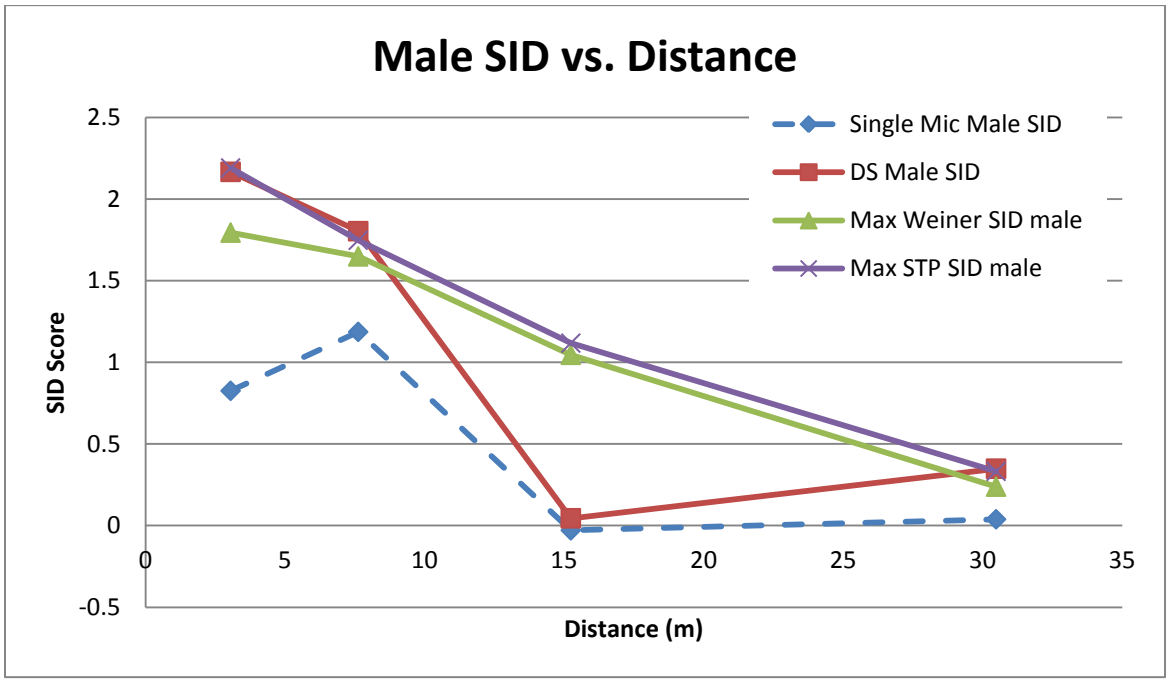

Figure 6.10 Male SID score vs. distance position for Outdoor 5/10/11 data

In figures above, it is seen that all multichannel processing techniques have the potential to enable higher SID scores than the single microphone. The STP method seems

to consistently achieve the highest SID scores out of all the multichannel filters. Despite its difficulties with output SNR, the Wiener filter follows a similar trend to the STP filter but with a slightly lower score. Finally, the Delay Sum beamformer actually registers slightly higher scores than the STP filter for some cases but is inconsistent, registering lower scores than the single microphone for other cases. For both male and female speakers, all three methods can enable positive scores up to 100 feet.

6.2.1.3 Analysis

The results of this experiment showed promise for multichannel filtering technique's ability to aid in speaker identification but also brought complicated problems to the surface. Much time and energy was spent trying to determine why output SNR was so hard to estimate for the Wiener Filter and why its output SNR trend differed so greatly from simulation. Cases where SNR was hard to estimate correspond to negative calculated SNR values. Negative SNR obviously does not exist and is an artifact of the SNR estimation technique. From equation 5.4 a negative SNR can be calculated if the estimated output noise power is greater than the actual total speech and noise power. Because the noise sample was taken at the beginning of the experiment and is not necessarily the exact noise in the noisy speech recording, difficulties measuring SNR can occur. Because of this observation, it was theorized that the Wiener Filter SNR degradation could be a result of the changing noise environment and estimation method used in this experiment as well as the high noise power relative to the recorded speech. The relatively high noise power gives little room for error when using this SNR estimation technique.

Despite the low or even un-measureable output SNR of the Wiener Filter, reasonable SID scores were still achieved. This shows that high output SNR does not always equate to higher SID score though it does seem to give the SID system a better chance. The SID score vs. SNR simulation in Section 5.2 shows similar results in that some speakers register positive identifications even for low SNRs. This simulation also showed that low SNRs can cause unpredictable SID scoring so it is still important to attempt to achieve a high output SNR.

Overall, the STP filter shows the most promise in both SID and SNR performance. The Wiener Filter seems sensitive to changing noise and the noise estimation technique used while the STP filter is more robust to these variations. The delay sum beamformer should not be totally disregarded as it achieves comparable SID scores even with consistently low SNRs but is "hit or miss" for some distance cases. More experiments should be performed to address some of the issues seen in this data collect.

6.2.2 Indoors: Building 123

6.2.2.1 Motivation and Procedure

In order to address the noise related performance concerns from the first outdoor collection, the second experiment was performed indoors. The lower, more controlled noise environment of the building 123 auditorium at LLNL was used to combat changing noise statistics and provide more "headroom" for SNR estimation. By controlling the acoustic environment, better filter performance will hopefully be achieved and comparisons can be made between the outdoor and indoor environments.

Figure 6.11 Indoor B123 experiment setup

For this experiment, the same two speakers, ‘D’ and ‘I’, are played back at various positions in the auditorium. The auditorium allows for a greater variety of source positions in both the vertical and horizontal directions. Figure 6.12 shows the microphone and source positions in the auditorium. A one minute noise sample is taken at the beginning of the experiment just like the outdoor data collection. Noise sources in this experiment include the building’s HVAC system and the data acquisition system’s fan. This type of noise is more stationary and should eliminate performance variations due to changing noise. It is important to note that while the recordings from this collect were used for this thesis, this specific experiment was designed for a different, but related project at LLNL.

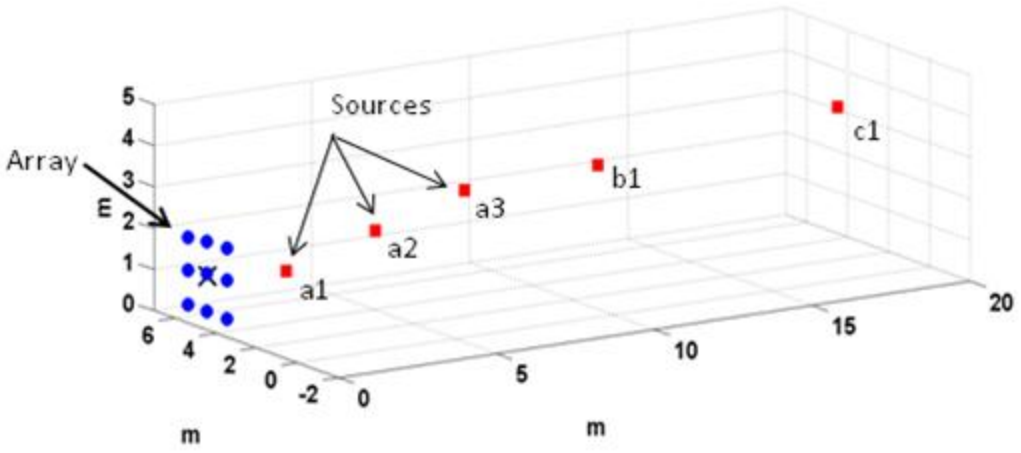

Figure 6.12 Microphone (blue) and speech source (red) positions for indoor B123 experiment [11]

6.2.2.2 Results

As done in the previous experiment, output SNR is measured and plotted for both speakers for each position case. The input SNR and delay sum beamformer output SNR are included in the table below for comparison. Un-measureable, negative, output SNRs are labeled as "N/A" in the table and are not included in the plots.

Table 6.2 Input SNR and delay sum output SNR for each speaker and distance case from B123

Position	Male Speaker		Female Speaker	
	Measured Input SNR (dB)	Delay Sum Output SNR (dB)	Measured Input SNR (dB)	Delay Sum Output SNR (dB)
A1	-3.393	-1.8006	-3.614	-3.15
A2	N/A	N/A	N/A	N/A
A3	-12.815	N/A	-10.11	N/A
B1	-19	N/A	-22.0761	N/A
C1	-8.526	N/A	N/A	N/A

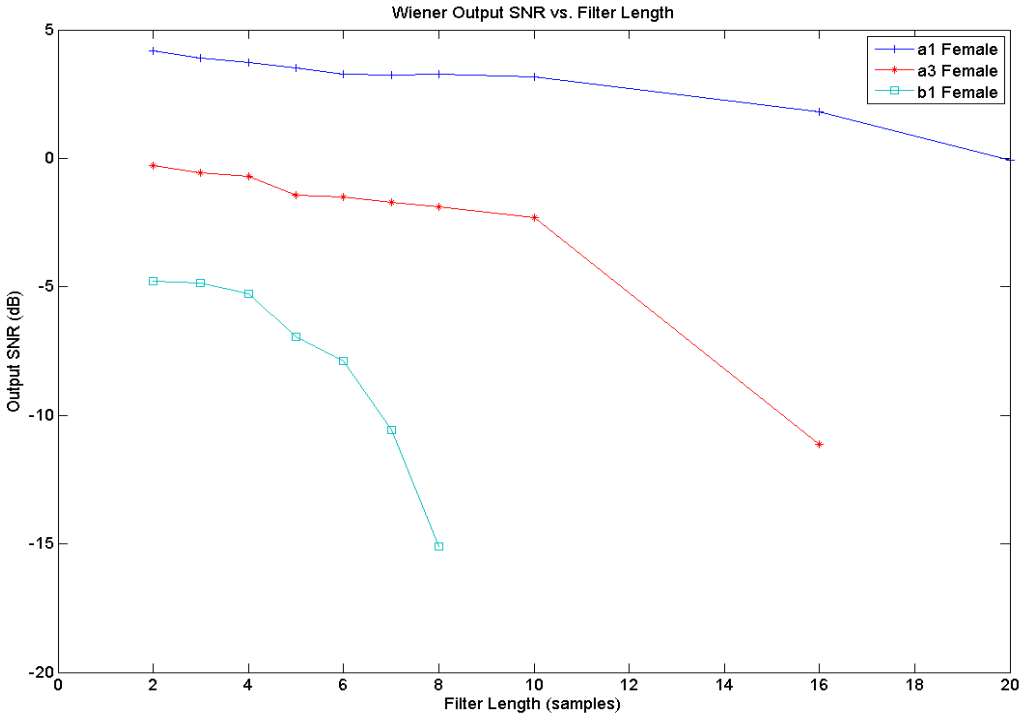

Figure 6.13 Wiener Filter output SNR vs. filter length for indoor B123 female speaker data

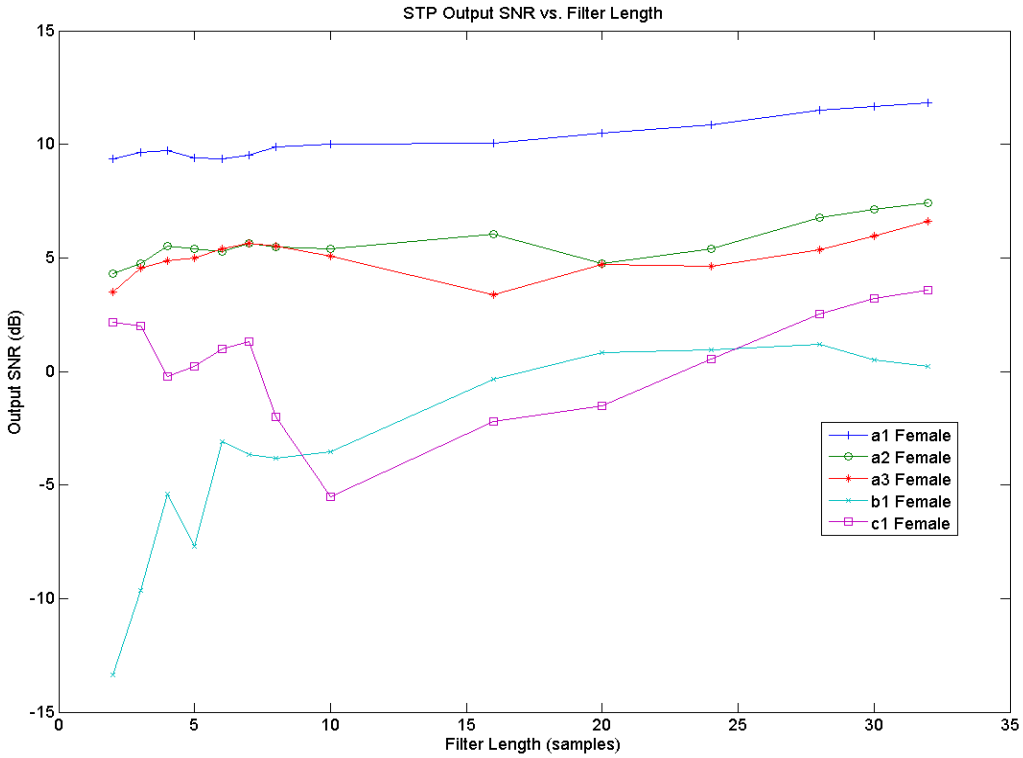

Figure 6.14 STP Filter output SNR vs. filter length for indoor B123 female speaker data

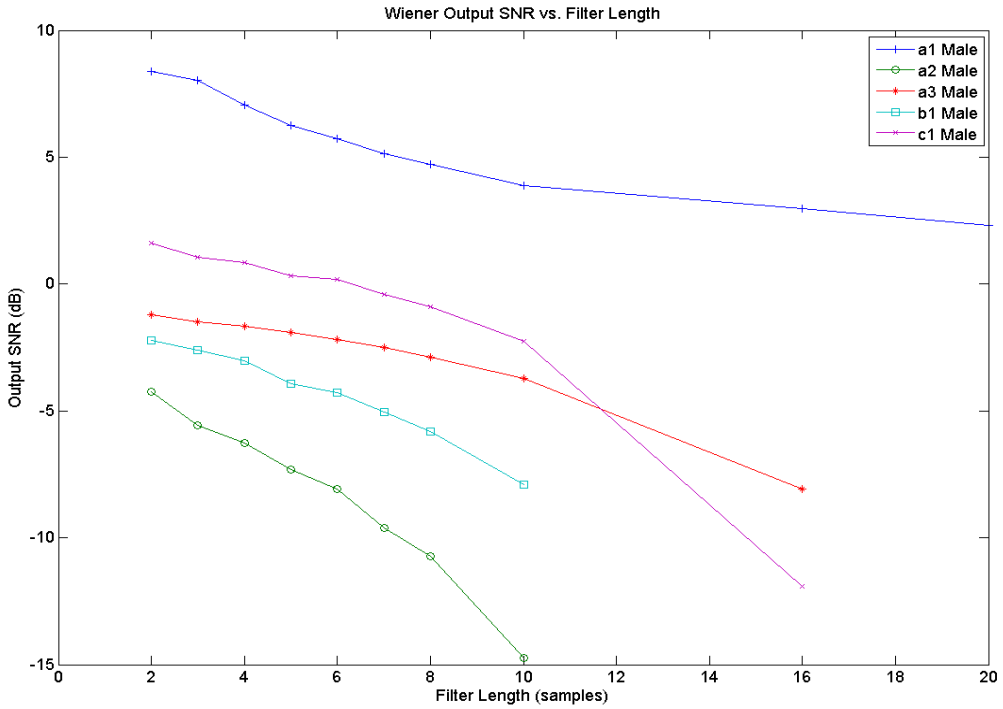

Figure 6.15 Wiener Filter output SNR vs. filter length for indoor B123 male speaker data

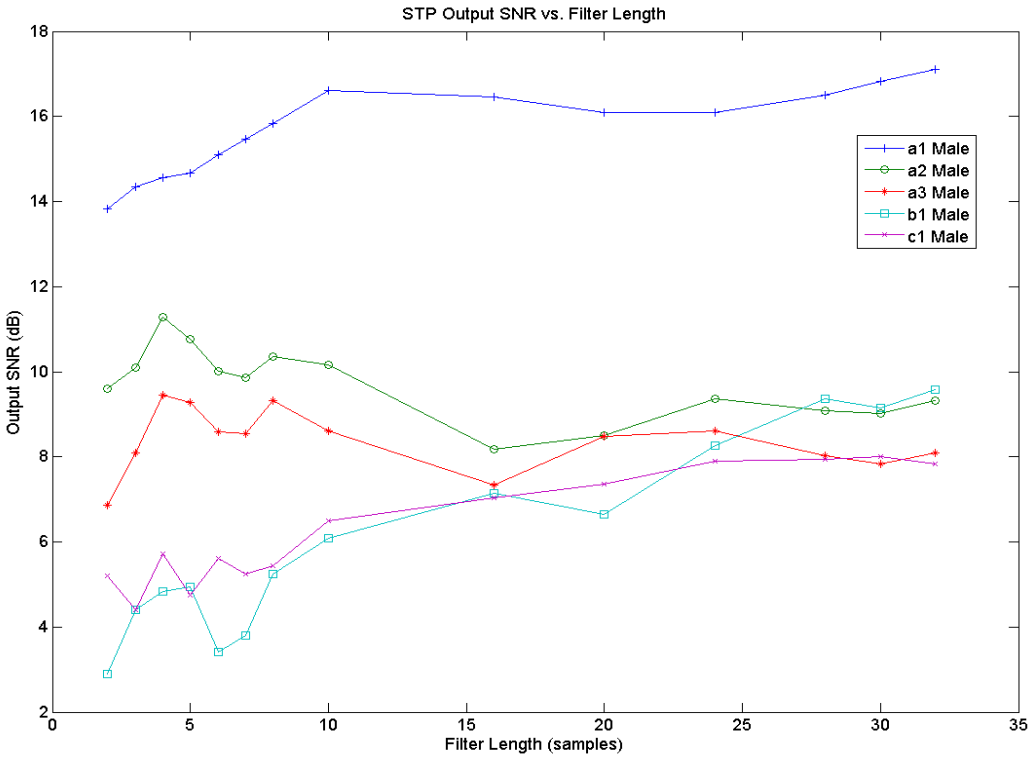

Figure 6.16 STP Filter output SNR vs. filter length for indoor B123 male speaker data

Output SNR for both filters in this experiment was greatly improved in magnitude and measurability. The Wiener filter output SNR was measureable for three out of five cases for the female speaker and all five cases for the male speaker. However, for each case the output SNR decreases monotonically as the Wiener filter length was increased. The STP filter achieves greater output SNR for both speakers with maximum SNRs of 11.82 dB for the female and 17.1 dB for the male speaker at the closest position, "a1". In general, the STP output SNR seems to increase or stay about the same as filter length is increased.

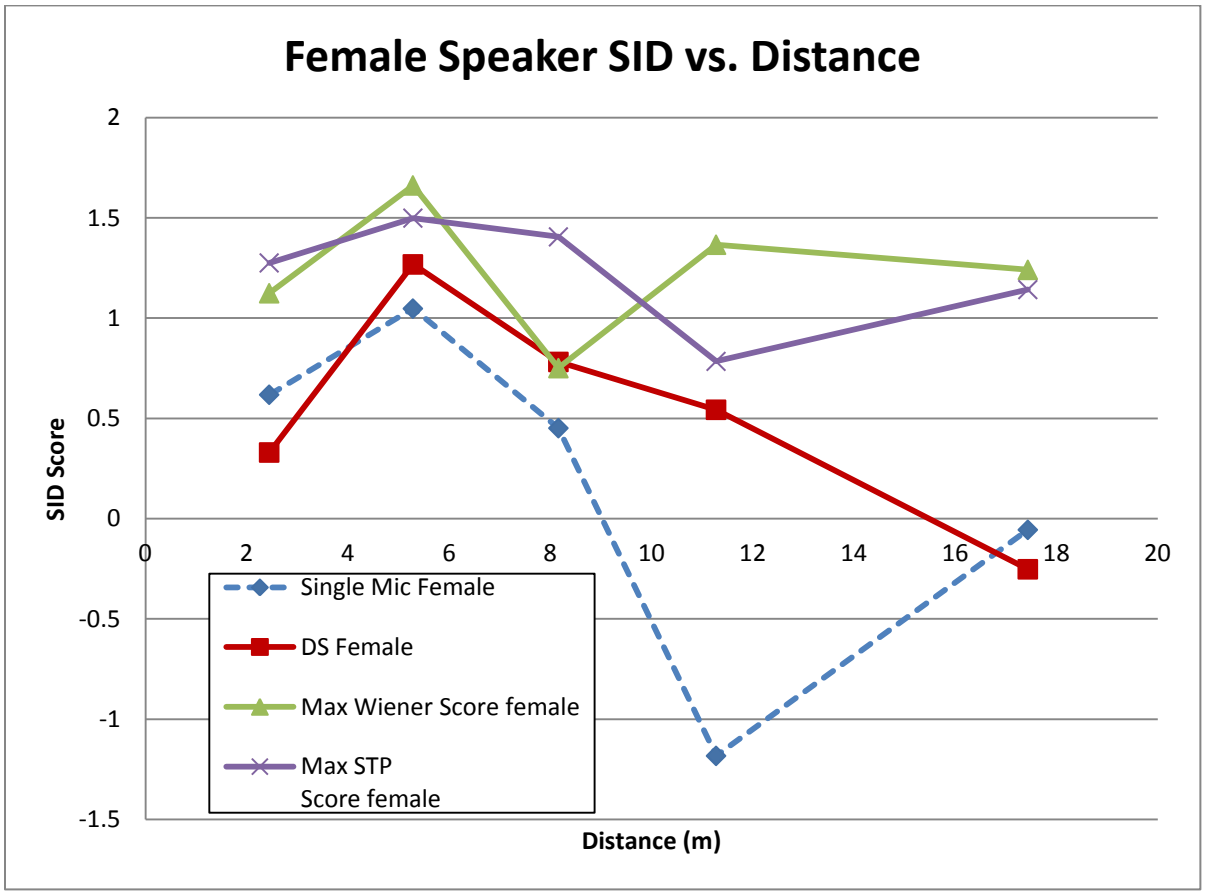

Figure 6.17 Female speaker SID score vs. distance position considering maximum scores achieved by adaptive filters

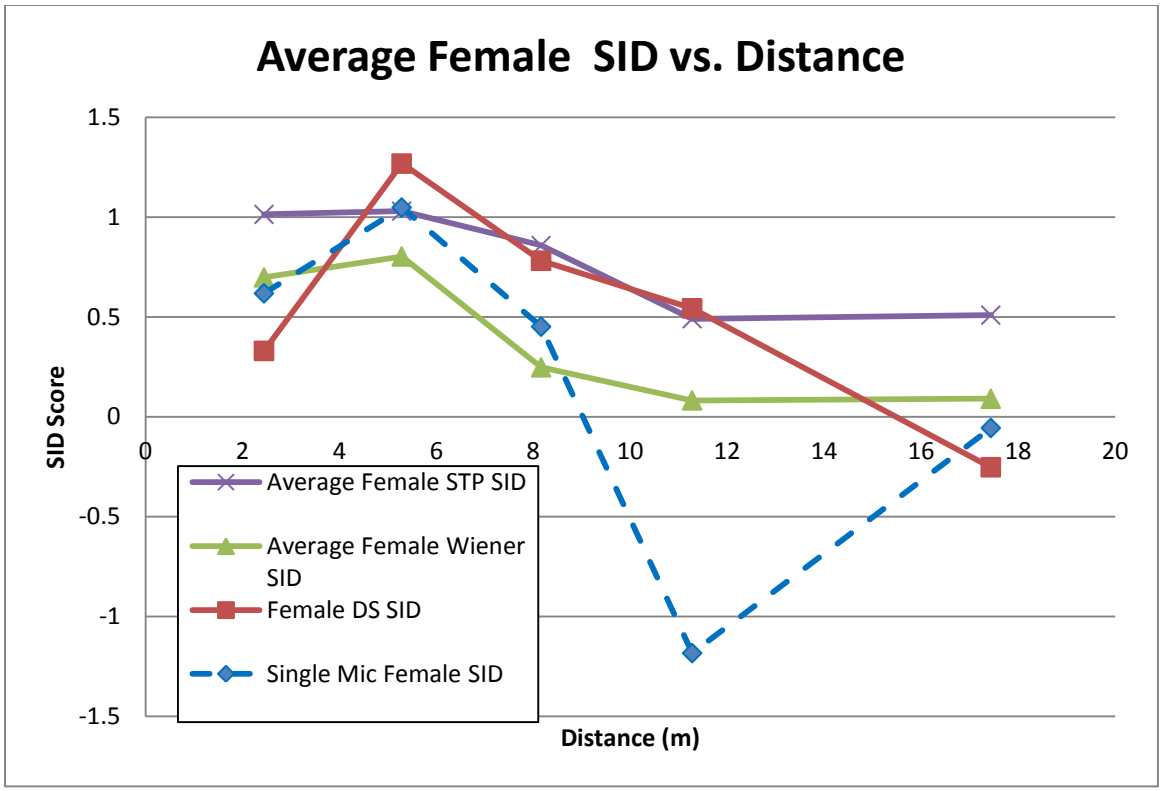

Figure 6.18 Female speaker SID score vs. distance position considering average scores achieved by adaptive filters

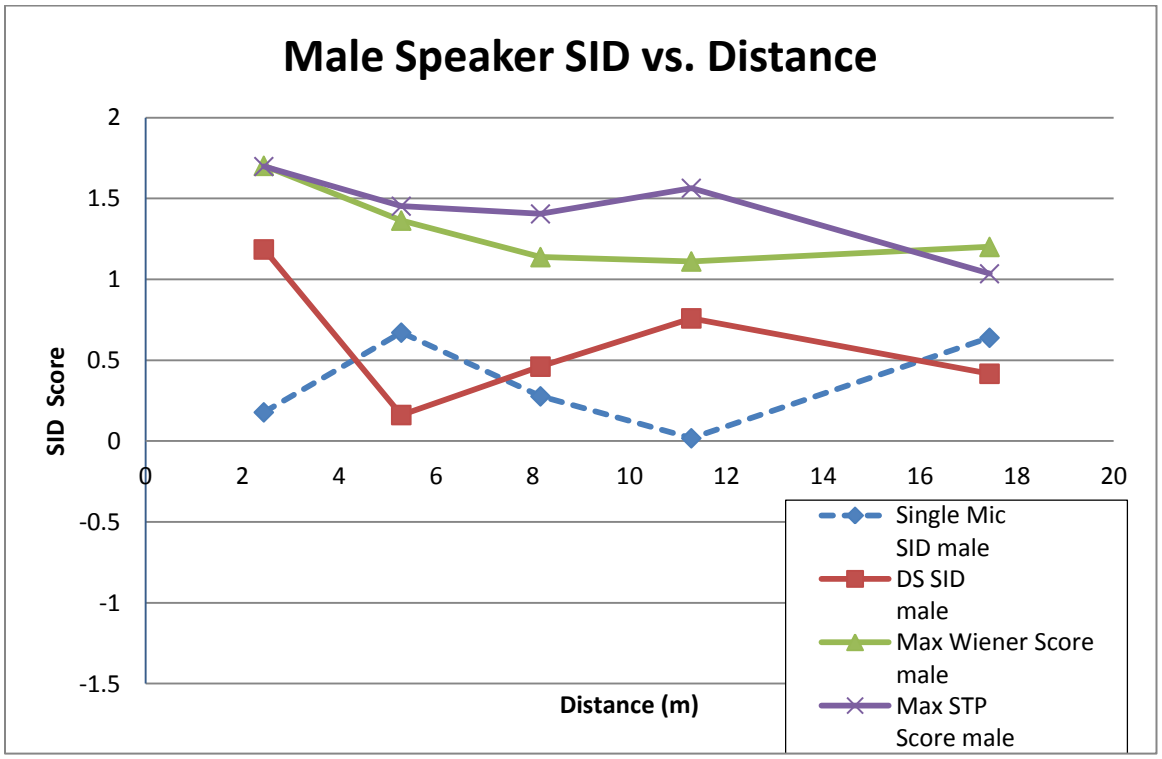

Figure 6.19 Male speaker SID score vs. distance position considering maximum scores achieved by adaptive filters

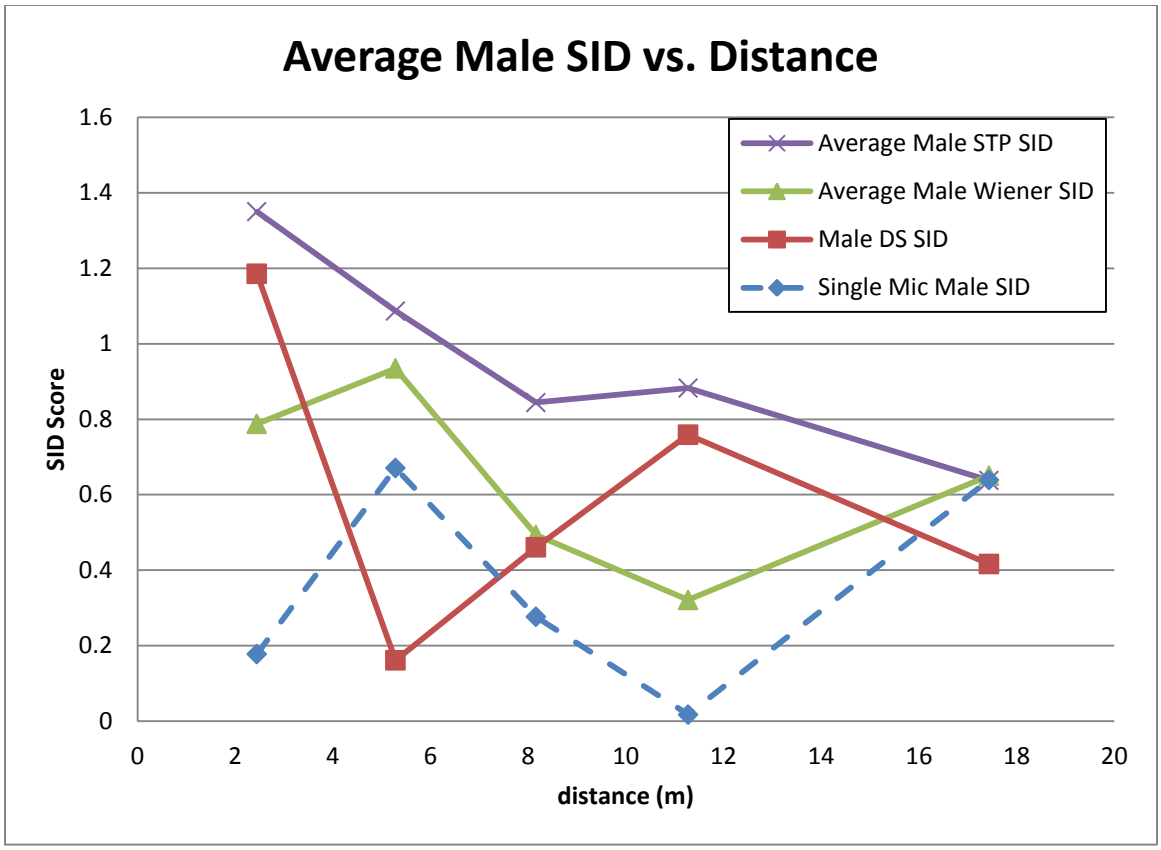

Figure 6.20 Male speaker SID score vs. distance position considering average scores achieved by adaptive filters

When looking at SID scoring for each approach, the adaptive filters maintain moderate to strong positives for each case. For the female speaker, the max SID scores are comparable for the adaptive filters, staying above a score of 1 for the most part. When all of the SID scores are averaged over all of the filter lengths used, the STP filter maintains higher SID scores for both speakers. For the female speaker, both adaptive filters show a similar trend but the STP filter maintains a strong to moderate score while the Wiener filter does not. With respect to the average scores of the adaptive filters, the delay sum beamformer registers a comparable score for the middle three female cases but registers a negative score for the longest distance case, "c1". The delay sum beamformer is below the average STP score for all cases but also seems to "mirror" the Wiener filter.

6.2.2.3 Analysis

Overall, the data from the indoor data collection provides more interpretable results than the outdoor experiment. From the SNR results, it is clearly seen that the STP filter performs best for all cases as the filter length is varied. The STP output SNR tends to increase over filter length but is also sporadic for some cases and can rise or drop sharply especially for the longer distance cases. Despite this, the output SNR seems to level off and become more predictable for longer filter lengths. The Wiener filter, on the other hand, still seems to have issues as there is a relatively sharp drop off in output SNR as filter length increases for all cases. This trend is the same for the outdoor data collect with the exception of one case. Despite this, the indoor noise environment caused more cases to result in reasonable SNR values than in the previous outdoor experiment. These observations suggest that the noise sampling method may cause the Wiener filter performance issues as the decreasing SNR trend continues even with the indoor noise environment.

As with the previous outdoor experiment, the multichannel noise reduction techniques allow for higher SID scoring. For both male and female speaker cases, both adaptive filters were able to achieve consistently strong maximum scores for every source position. The difference between the two is seen when the SID scores provided by all of the different filter lengths are averaged for each position. After this averaging is performed, the Wiener filter scores fall to moderate to weak positives while the STP filter maintains strong to moderate positives in Figure 6.18 and Figure 6.20. This result may be due to the low output SNR of the Wiener filter which causes sporadic SID scoring as seen in the Section 5.2 SNR SID simulation. Though the Wiener filter can enable positive

scores comparable to the STP filter, it also registers some lower scores that bring its average score down.

From this indoor experiment, more results and trends were able to be extracted from the data than in the previous collect. Observing these trends has allowed for generalizations on SNR and SID performance to be hypothesized. Heavy speaker dependence is still observed in this experiment which inhibits complete generalization of performance to all cases. If this dependence can be removed, performance prediction may be possible which will help when making recommendations for using these filters in practice. To accomplish this, the third and final experiment is designed for the purposes of this thesis with the results from the past two data collections in mind.

6.2.3 Outdoors 11/17/2011

6.2.3.1 Motivation and Procedure

Thus far, this thesis has presented and developed 3 multichannel noise reduction techniques and has proven their potential to aid in speaker identification. The previous two data collections demonstrated the usefulness of these algorithms in a post processing scenario by enabling identification of two different speakers in two different environments. These experiments showed capable results but it is unclear how much these results can be generalized for predicting performance in real applications. Because this system could be applied to infinite scenarios (speakers, positions, environments) this final experiment must incorporate as many conditions as possible to get a good idea of which filtering approach is best.

As opposed to the previous two experiments, data collection was designed specifically to obtain results that will enable more generalizations to be made about the use of these multichannel filters. Instead of just a male and female speaker, all five speakers in the target group will be used as speech sources. Averaging the results over more speakers will give better insight into the general performance of the filters by attempting to remove the speaker dependence of the output SNR and SID. For this experiment, each speaker source will be played back at seven different distances with finer spacing increments. In order to address the concerns about noise sampling, a new noise recording will be taken every time the source position moves. The noise sample taken at each source position will be used to filter the speech signals at that corresponding location. This method will hopefully reduce the effects of changing noise over the course of the experiment which can take up to 3 hours. Finally, the same outdoor environment as the first experiment is chosen for this last data collection. This outdoor location provides more realistic background noise that mimics practical application environments.

For this experiment, the microphone array is deployed at the North West corner of LLNL as seen in Figure 6.21. Each of the five speakers are played back at distances of 10ft, 20 ft, 30 ft, 40 ft, 50 ft, 75ft and 100ft. As mentioned, an additional 30 second noise sample is taken after the source position is moved, before the speakers are played back. The noise sample corresponding to each distance is used as the “noise only” input to the filtering functions during post processing. After filtering, SNR and SID are averaged over all five speakers to extract and analyze performance trends.

Figure 6.21 Outdoor 11/17/11 experiment set up

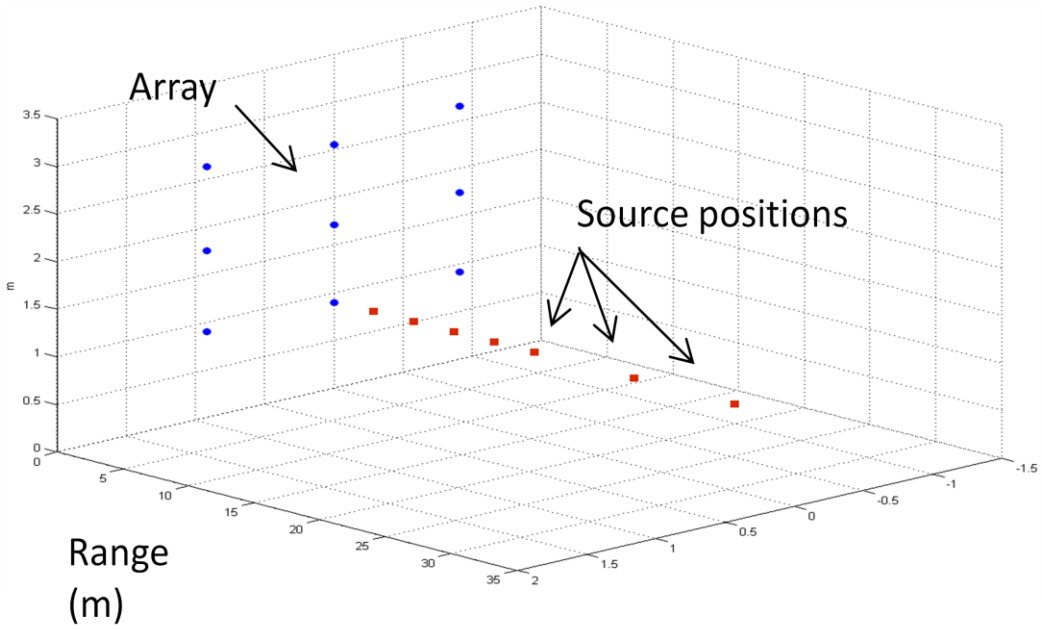

Figure 6.22 Microphone (blue) and speech source (red) positions for outdoor 11/17/11 data collection

6.2.3.2 Results

Below are the plots for output SNR and SID scores. A table including the input SNR and delay sum output SNR is included for comparison. Again, cases where output SNR is unable to be estimated are marked as "N/A" in the table and are omitted from the plots as well.

Table 6.3 Average input SNR and delay sum output SNR for each distance case from 11/17/11

Distance from array (ft)	Average Input SNR (dB)	Average Delay Sum output SNR (dB)
10	3.156	8.227
20	5.881	5.831
30	-0.821	-1.967
40	-1.531	2.503
50	-1.778	-2.134
75	-0.974	-2.111
100	-4.716	-4.056

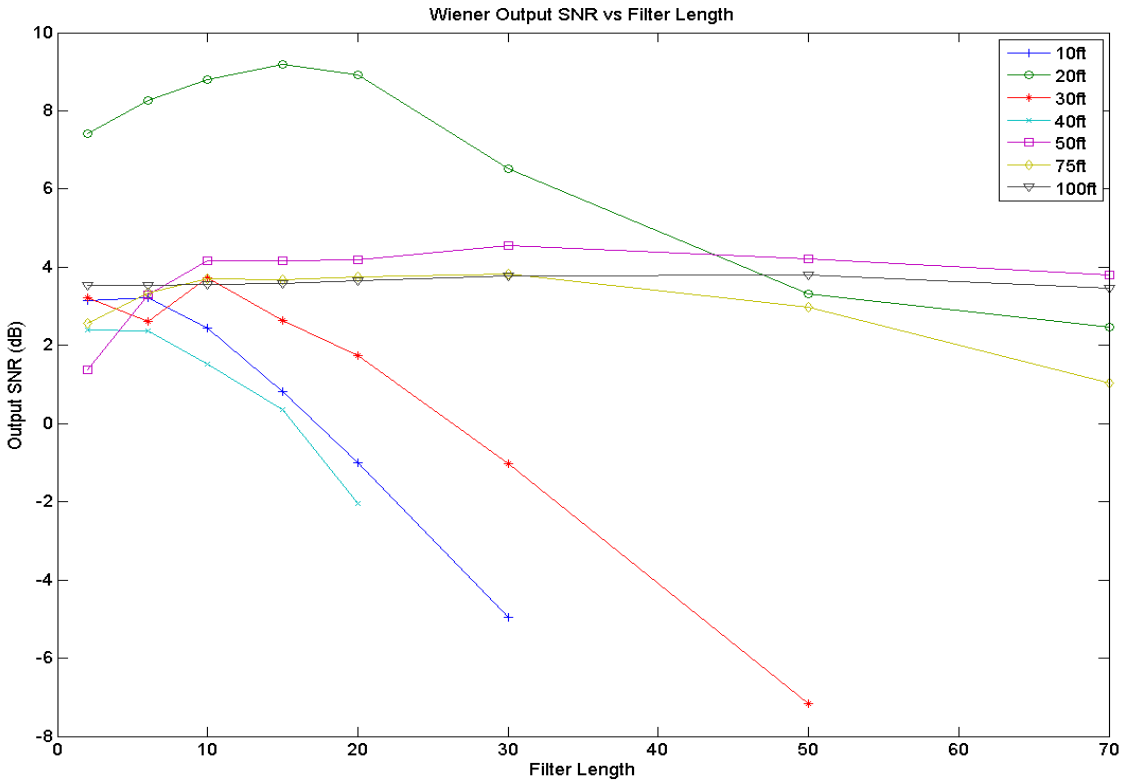

Figure 6.23 Average Wiener Filter Output SNR vs. filter length for all target group speakers

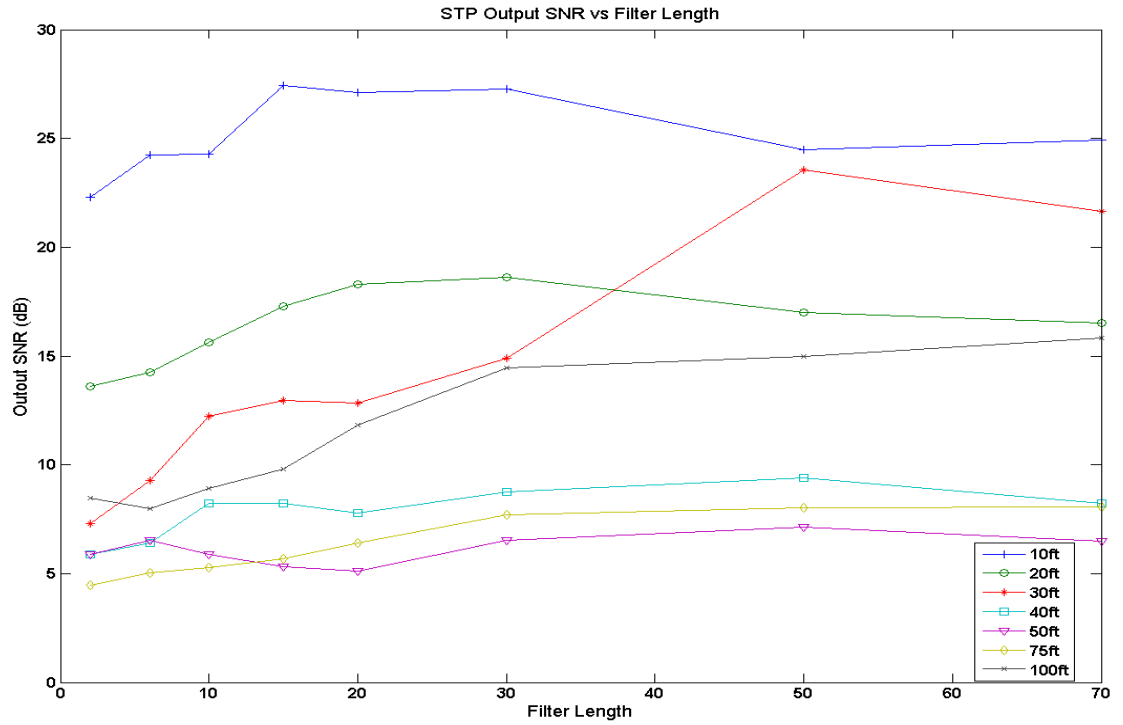

Figure 6.24 Average STP Filter Output SNR vs. filter length for all target group speakers

The output SNR results for the two adaptive filters are shown in the above figures. These plots depict the average output SNR over all five speakers for each filter length used. Filters of length 2 to 70 samples are used in this experiment to see how even longer filters affect SNR. From Figure 6.23 it is seen that the Wiener filter exhibits similar SNR trends to the simulations in Chapter 5 for 50 ft, 75 ft and 100 ft cases but still has issues with decreasing SNR for the other cases. For the 20 ft case, the Wiener filter registers the highest SNR of 9.21dB for a filter length of 15, but subsequently decreases for longer filters. For all cases, the Wiener filter does not seem to benefit from longer filter lengths and in general achieves the best output SNR for filter lengths of 10-30. The STP filter, on the other hand, maintains or increases its output SNR with filter length and achieves much larger absolute SNRs for each case when compared to the Wiener filter. The STP filter produces a maximum SNR of 27.44 dB for the 10 ft case with a filter length of 15 samples. Though the STP filter does not seem to benefit from the longest filter length of 70 samples, it still achieves its best output SNR results for filter lengths of around 20-50 in general.

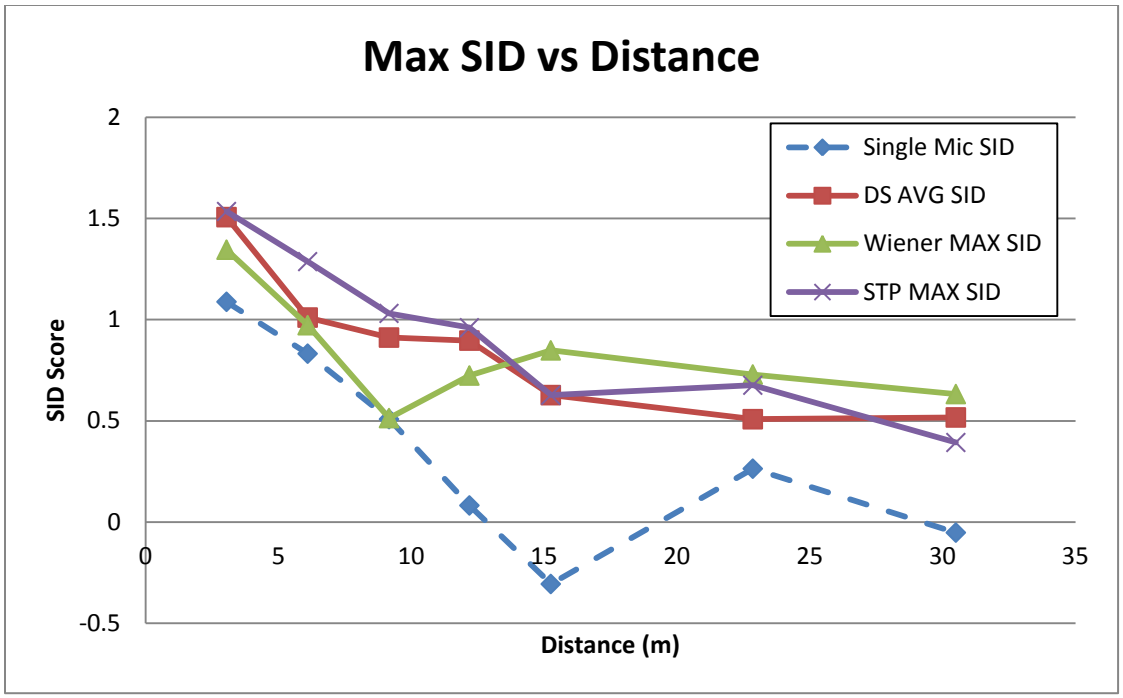

Figure 6.25 SID score vs. distance position considering the maximum score achieved by the adaptive filters when using a certain filter length.

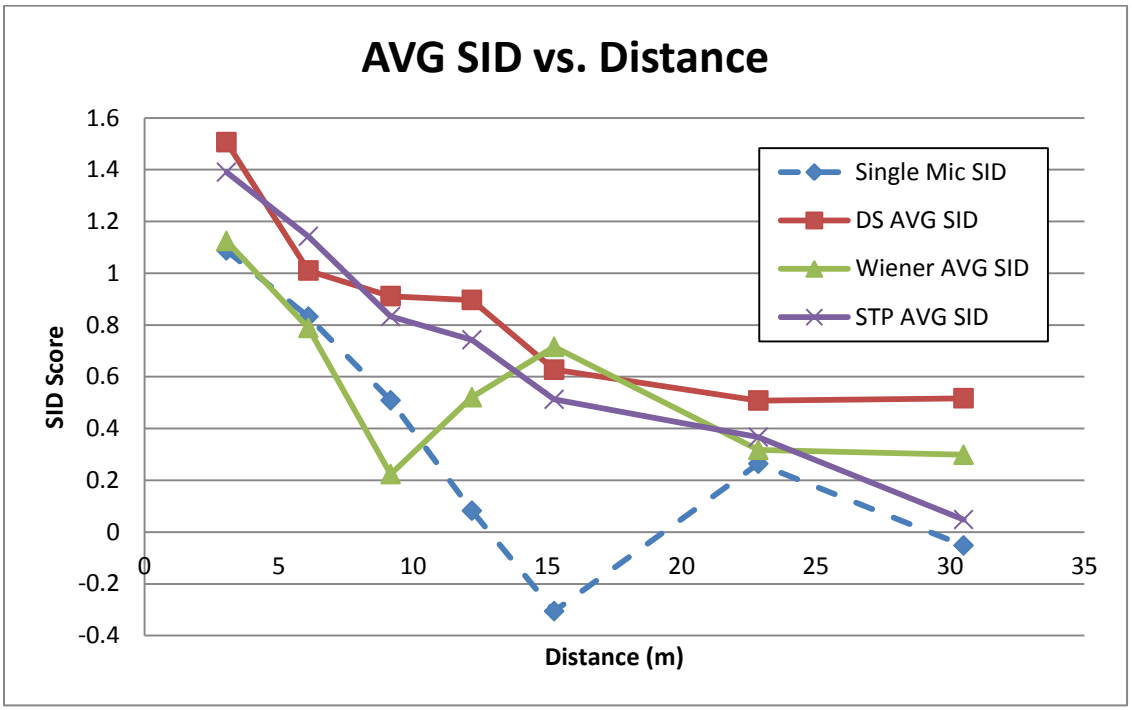

Figure 6.26 SID score vs. distance position considering the average score over all filter lengths and speakers

Similarly to SNR, the SID results above are obtained by averaging SID scores over all speakers for each filter length used. For each distance case the maximum and average SID score of all the filter lengths are plotted in the figures above. In general, it is seen that SID score decreases with distance. When looking at the maximum SID score, all three multichannel filters enable higher SID scores with the STP filter registering higher scores below 50 ft while the Wiener filter registers higher scores at the longer distances. The delay sum beamformer produces comparable SID scores to both adaptive filters. When the SID scores are averaged over all speakers as well as filter lengths for each distance, both of adaptive filters scores drop. In general, the STP filter keeps higher scores than the Wiener filter but decreases almost linearly with distance. The Wiener filter even drops below the single microphone score for some distances. All three approaches enable a positive identification at 100 ft when the single microphone causes a negative identification.

6.2.3.3 Analysis

Through the use of five speakers, the results of this experiment allow the performance of the three multichannel noise reduction techniques to be better compared and generalized. Also, the new method of noise sampling produced better adaptive filter output SNR than in the previous two experiments. The Wiener filter produced flatter output SNR trends similar to what was seen in simulation for the three longest distances. Because some distance cases still suffer from decreasing output SNR even after speaker averaging, it is clear that Wiener performance degradation is not speaker dependant. From further exploration, it seems as though the Wiener filter is highly sensitive to noise sampling procedures. Though taking more noise samples throughout the experiment

helped with some cases, other position cases still produce decreasing SNR trends. Taking noise samples before every speaker is played back may further improve performance as these noise samples would be more representative of the actual noise in the speech recording. The STP filter SNR performance also seemed to be improved by taking more noise samples. The STP filter produces a maximum output SNR of 27.44 dB, highest of all the experiments. By averaging over all five speakers, it is seen that the STP filter performs better at medium to high filter lengths of 20-50. The output SNR also increases or stays about the same as filter length is increases, making filter length a good predictor of output signal to noise ratio for the STP approach. From this experiment, it is clear that the Spatio-Temporal Prediction filter provides the best output SNR while the Wiener filter achieves modest SNR that may decrease with filter length due to its sensitivity to noise estimation.

By averaging SID scores over all speakers for each position, new trends are observed. The most surprising result is that the delay sum beamformer is comparable to the adaptive filters when considering their maximum SID score achieved for all filter lengths. When the SID scores of the adaptive filters are averaged over all speakers and filter lengths used, their SID scores drop below the delay sum beamformer for the most part. This shows that even though the delay sum beamformer has proven unreliable in the past two experiments, it can be comparable to the adaptive techniques on average. Another explanation for this boost in delay sum beamformer performance could be contributed to more accurate source position measurements in this experiment.

When comparing the adaptive filters, the STP filter registers higher maximum SID scores for distances below 50 ft while the Wiener filter can achieve higher maximum

scores for the longer distances. When the SID scores are averaged over all speakers and filter lengths, the Wiener filter actually scores lower than the single microphone for distances less than 40 ft. For the longer distance source positions, the fully averaged Wiener filter SID scores are comparable to the STP filter but maintains a higher positive score at 100 ft. This sporadic behavior could be caused by the low output SNRs of the Wiener filter. It is also plausible that both adaptive filters could cause speech distortion which might explain why their average scores are below the delay sum beamformer in Figure 6.26. It is also important to remember that the delay sum beamformer does not have a "filter length" and is therefore only averaged over all speakers for both Figure 6.25 and Figure 6.26.

The results of this experiment showed that with precise source location the delay sum beamformer can perform similarly if not better than the adaptive approaches in terms of SID. Though the delay sum beamformer produces relatively low output SNRs of 8.23 dB at 10 ft and -4.06 dB at 100 ft, it is not prone to speech distortion if the exact source position is known. The adaptive filters are more likely to distort speech due to their estimative nature which could cause a drop in SID performance. Overall, this experiment has shown that the delay sum beamformer should not be discounted as a viable multichannel noise reduction technique even though the Spatio-Temporal Prediction approach achieves the highest output SNR. This outcome greatly contrasts the past two experiments where the delay sum beamformer was inconsistent in SID performance. It is evident that the use of more speakers and positions is the best way to further evaluate these approaches.

6.3 Additional Observations

In order to verify assumptions and test hypotheses related to the performance of multichannel filters presented in this thesis, supplementary exploration is necessary. From earlier simulations and data collects certain questions arose such as:

"Do the noise statistics actually vary over time?"

"How does the number of microphones used affect filter performance?"

"Does the delay sum beamformer achieve the same noise reduction for all look directions?"

These questions were saved and compiled in this final section to better understand aspects of multichannel noise reduction which may not have been fully explored or explained earlier in this thesis. Hopefully these brief explorations will help clear up any lingering curiosities before the conclusion of this thesis.

After performing the three experiments in Section 6.2 along with further exploration, it is apparent that changing noise may inhibit the performance of the adaptive filters. If the noise sample used to calculate noise statistics differs from the actual background noise present in the noisy speech recording, the filter may not achieve as much noise reduction and may even add noise. In the first two experiments, only a single 60 second sample recording of noise was taken in order to provide an estimate of the background noise for post processing and analysis. This noise sampling method may be insufficient if the noise changes over the course of the experiment and may cause undesirable results. The third experiment aimed to reduce the affect of changing noise by taking more noise

samples over the course of the experiment. It is from this data that we can actually test the hypothesis that background noise changes over time and would therefore affect filter performance. In Figure 6.27, the noise power for each 30 second noise sample is calculated and plotted against the time it was taken. From Figure 6.27, it is apparent that the noise power changes over the course of the data collection. Because the power changes over time, it can also be inferred that the frequency content of the noise changes as well. This result further cements the idea that more noise samples taken over an experiment would result in more accurate noise estimation during filtering, leading to better noise reduction. The overall solution to this problem would be a real-time implementation of these adaptive filters using a voice activity detector (VAD) to re-sample the noise during every interval where no speech is present.

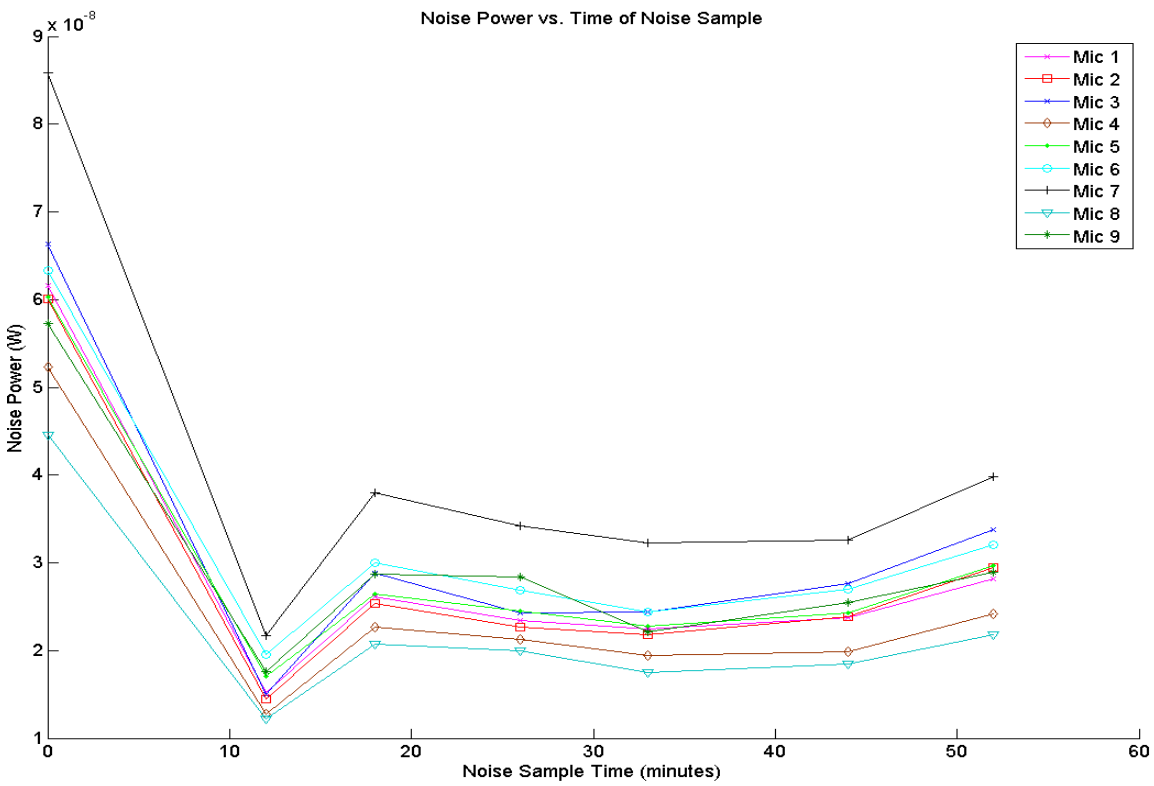

Figure 6.27 Noise power vs. the time that the noise sample was taken during the 11/17/11 data collection

Another area of multichannel noise reduction that was not emphasized but is worth noting is the number of array elements used in the system. In general, one would assume that more receivers produce better performance due to the added spatial information. This is especially true for delay sum beamforming where noise is theoretically reduced proportionally to the number of receivers, assuming white noise [1]. This generalization does not hold true for the two adaptive filters. Figure 6.28 and Figure 6.29 show how the output SNR vs. filter length trend changes for the Wiener and STP filters as the number of microphones is varied. For this exploration, the closest female speaker data from the indoor data collection at position “a1” is used as an example.

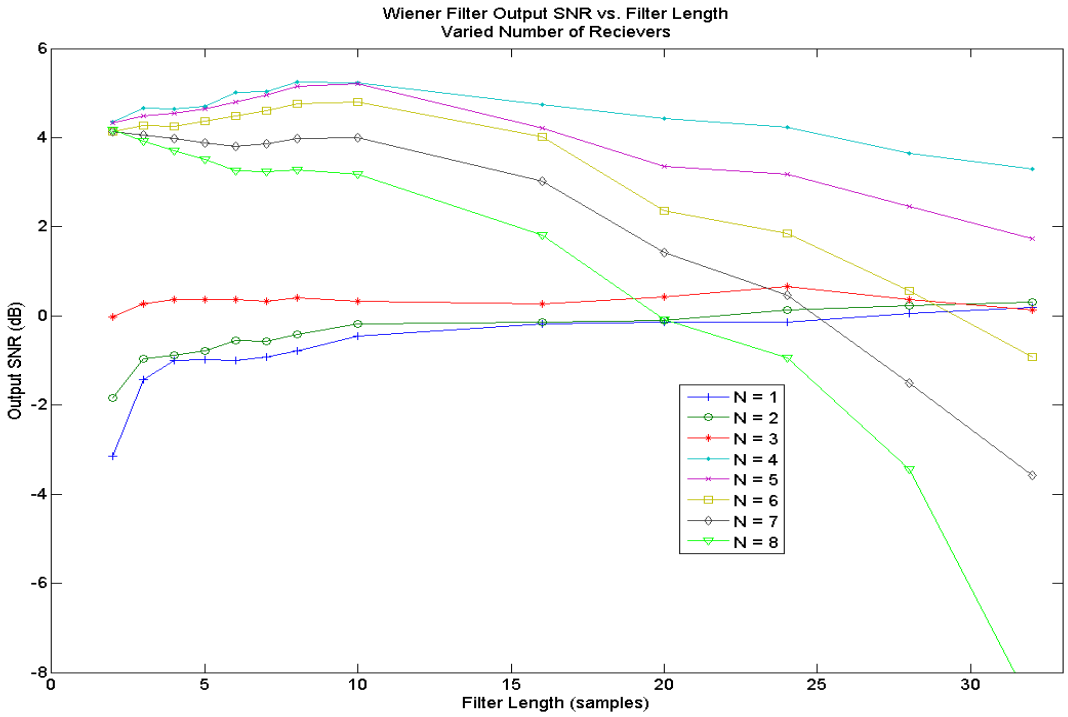

Figure 6.28 Wiener filter output SNR vs. filter length with varying number of microphones

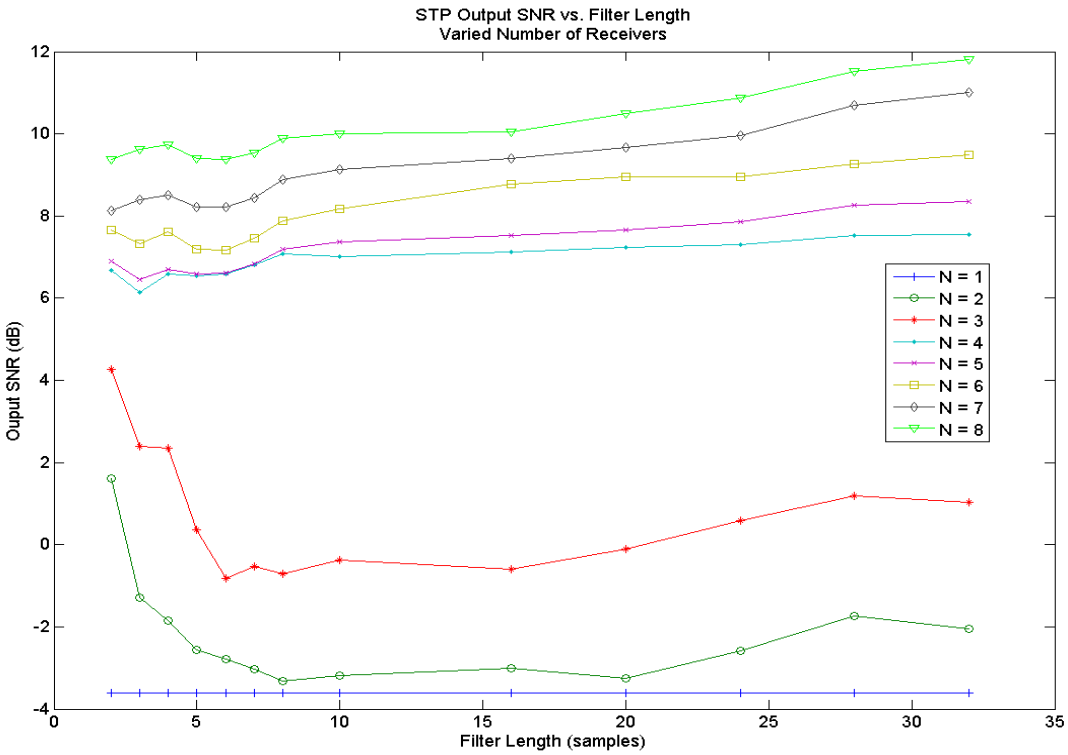

Figure 6.29 STP filter output SNR vs. filter length with varying number of microphones

From the above results it is apparent that the number of microphones greatly affects the output SNR performance of both of the adaptive filters. For the Wiener filter, it seems that adding too many microphones may cause a decrease in output SNR. In Figure 6.28 it is seen that using the bottom three microphones produces a stable output SNR trend (similar to earlier simulations) but adding a 4^{th} microphone completely changes the output SNR trend. Recalling that the array in the field utilized a 3x3 element pattern, adding this 4^{th} microphone to the system changes the microphone configuration from a 2D linear array to a 3D array pattern. Adding this second row to the array causes such a significant rise in SNR because it adds another spatial dimension and therefore more spatial information. The downside of adding the extra spatial dimension is that it may cause output SNR to decrease with filter length. It is seen that this affect gets worse as more microphones are added. From this exploration it is evident that the Wiener filter is optimal for smaller 3D arrays with small to medium filter lengths.

When looking at the results of the STP filter, almost the opposite trend is observed. As more microphones are added, output SNR is increased. As with the Wiener filter, adding the 4^{th} microphone creates a drastic increase in output SNR and also changes how SNR varies with filter length. For the STP filter though, the output SNR trend is improved so that SNR increases with filter length. It is also worth noting that unlike the Wiener filter, the STP filter cannot achieve any noise reduction with a single microphone. This makes sense as the STP algorithm requires a prediction matrix to relate each channel which becomes an identity matrix for the single channel case. In general, it seems that the Spatio-Temporal Prediction filter performs better with more array elements as well as

longer filters. This intuitive performance predictability makes the STP filter an attractive approach.

In the final exploration of this section, the question of the delay sum beamformer's dependence on look direction and noise environment is investigated. It was seen through simulation in Section 3.1.2 that spatially correlated noise can cause the delay sum beamformer to not achieve uniform noise reduction for all look directions. Because of this observation, it was of interest to see if a similar effect occurs with the real data obtained in the above experiments. Figure 6.30 and Figure 6.31 plot the steered response of the delay sum beamformer using only the sampled background noise as the input. These results prove that the delay sum beamformer does not achieve uniform noise reduction in all directions. For certain directions the spatially correlated nature of the background noise may cause peaks in noise power due to the applied time delays of the beamformer. This effect may cause problems in practical applications if the desired speech source happens to be located at or near the position of the peak noise power. Depending on the noise, a situation like this may cause the delay sum beamformer to achieve minimal noise reduction.

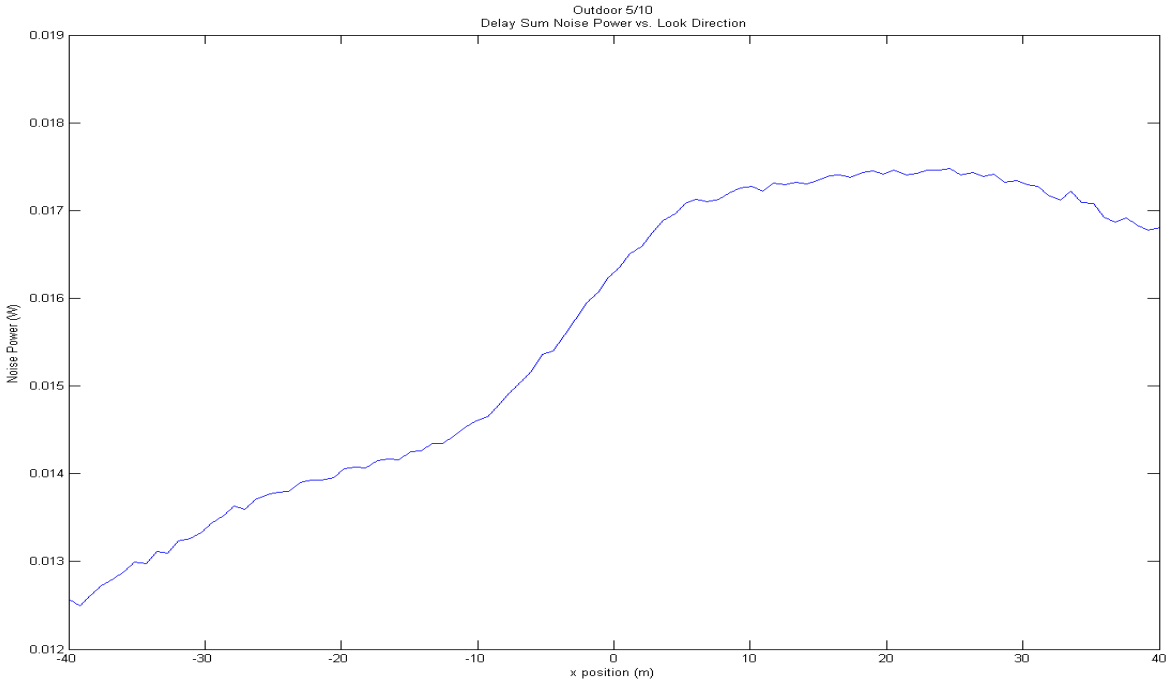

Figure 6.30 Delay sum beamformer output noise power vs. look direction for outdoor 5/10/11 data

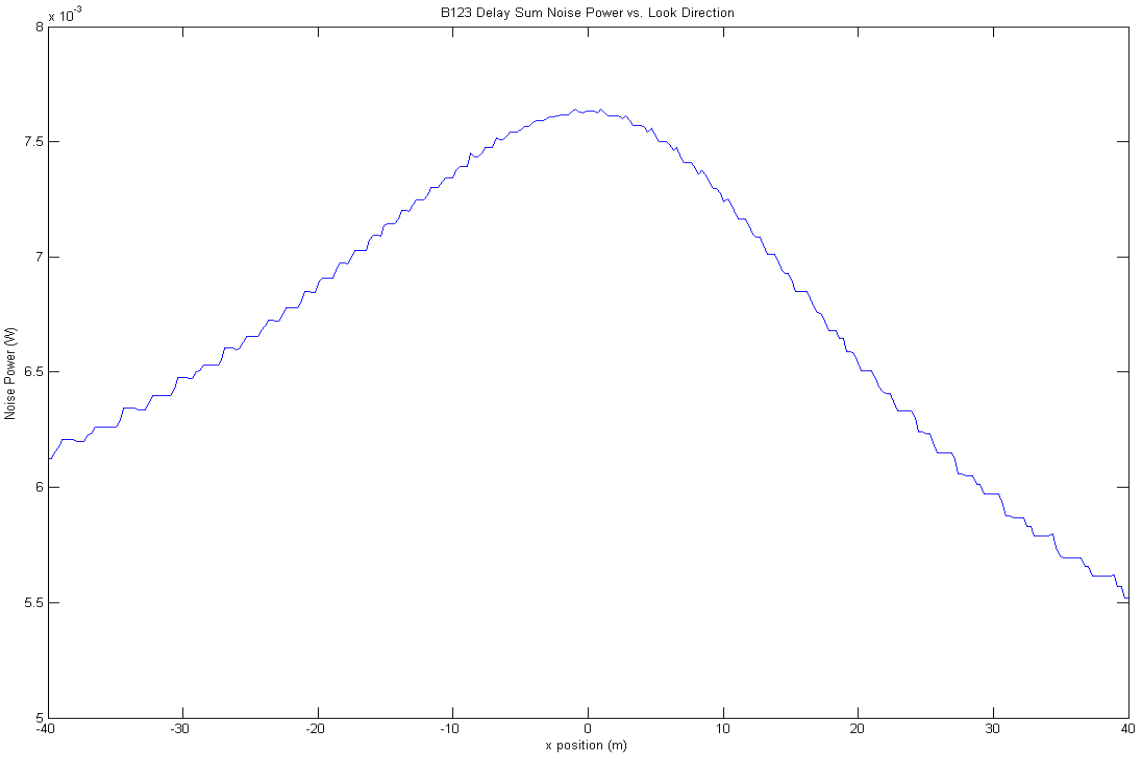

Figure 6.31 Delay sum beamformer output noise power vs. look direction for indoor B123 data

Though this section may have produced more questions than answers, these findings provide valuable insights into optimal filter implementation for practical applications. To summarize, real acoustic noise changes over time requiring frequent noise sampling, the number of microphones and relative spatial positions affect adaptive filter performance and the delay sum beamformer does not achieve the same noise reduction over all directions. Though each of the ideas were only briefly explored, it is conceivable that more extensive research could be done in these areas to better understand multichannel noise reduction.

7. SUMMARY OF FINDINGS

In the final chapter of this thesis, the work performed and the knowledge gained from this project will be summarized. First, a comparison of the three multichannel noise reduction approaches is given along with recommendations for implementation. Next an overall conclusion of the project highlights key "take aways" and lessons learned. Finally, ideas for future work are provided to inspire continued research in the area of multichannel noise reduction with microphone arrays.

7.1 Algorithm Comparison

In this thesis, three multichannel noise reduction algorithms were presented, implemented and tested using simulation and experiments. Specifically of interest in this project is determining which approach works best for speaker identification and under what circumstances. This section will therefore provide a final comparison of the delay sum beamformer, Wiener Filter and Spatio-Temporal Prediction filter as well as recommendations for implementation in real applications.

From this thesis, it is apparent that the delay sum beamformer is a simple and capable approach with some drawbacks. Theoretically, the delay sum beamformer can achieve complete signal recovery without distortion and achieve low to moderate noise reduction. Caution must be used with this approach as noise reduction can be inconsistent depending on the look direction of the beamformer due to the spatial correlation of the noise. Also, when using the delay sum beamformer, the exact position of the speech source and relative microphone positions must be known to avoid distorting the speech. Despite these potential drawbacks, the delay sum beamformer performed extremely well

compared to the two adaptive filters in the final outdoor experiment in terms of SID scoring. The delay sum beamformer has proved to be a valid approach and would be a good choice for applications where processing resources are limited and moderate to low noise reduction is sufficient. Again, accurate source and microphone positions must be known to assure optimal performance.

The Wiener filter showed promise in simulation but came up short in the practical experiments. The Wiener filter may be more attractive than the delay sum beamformer because it is adaptive but it does not always guarantee better performance as seen in Chapter 6. The Wiener filter performance seems sensitive to implementation parameters such as noise sampling, filter length and number of microphones, causing varied performance. Overall, the Wiener filter should be used when an adaptive filter with moderate noise reduction is desired but should only be used with a small number of microphones and smaller filter lengths to avoid unexpected output SNR degradation.

Finally, the Spatio-Temporal Prediction filter consistently achieved the best noise reduction and showed the most potential for enabling higher SID scores. This adaptive filter allows for deployment in any noise environment and does not require the microphone or speaker positions to be known. This attractive feature paired with its excellent experimental noise reduction, robustness to changing noise and overall solid SID performance for all experiments makes the STP filter the most recommended choice. The STP filter should be deployed where maximum noise reduction is desired and computational resources are abundant. Due to its predictable performance, the STP filter should be implemented with as many microphones as possible using longer filter lengths.

7.2 Conclusion

Through the work of this thesis, microphone array processing techniques have proven viable for implementation at the front end of speaker identification systems. The main highlight of this work is that all three algorithms enabled a positive identification at 100 ft where the raw data produced a negative score. Though the results were not as "clean" as expected, much was learned about each of the three noise reduction algorithms and their implementation advantages and drawbacks. While the adaptive filters do not need to know microphone and speaker positions, their performance is highly dependent on noise sampling, number of microphones and filter length. If tuned correctly these parameters can optimize the performance of the adaptive filters but if set improperly can cause performance degradation. The proper parameters found for each filter shows that each one is specialized for certain situations given application specifications and resources available.

Another key point learned is that SID scoring is not deterministic but stochastic in nature. While improving SNR gives a better chance of achieving a higher SID score, it is not always guaranteed. Speech distortion may cause a reduction in SID scoring even if a large SNR is achieved through filtering. In general, dealing with audio and speech is not an exact science. If a certain experiment is performed once, the same results cannot always be expected consistently. There are just too many variables that are hard to control for. When dealing with real speech and noise there are no limit to the combination of inputs to the system, making it difficult to generalize results. This is why it is important to run as many experiments and test cases as possible in order to extract general trends.

Throughout this thesis, the Spatio-Temporal Prediction Filter has proven to be the best algorithm considering all simulations and experiments. Though the STP filter is most recommended, the Wiener filter and delay sum beamformer are still valid approaches especially if application resources are constrained. In engineering, there is not always a clear cut result; there are always performance, design and application tradeoffs that have to be balanced to achieve the best solution possible.

7.3 Future Work

Through the progression of this thesis, many new ideas for future work have arisen that are outside the scope of this project. Some of the following ideas could be used to build on this project or are simply curiosities that warrant further research.

One project that would be worth looking into is actually implementing the two adaptive filters in real time using a voice activity detector. This approach would better estimate the noise because the system would update noise statistics in every non-speech interval. Care should be taken in order to make sure the desired speech signal is never classified as a noise interval by the VAD. If this occurs, the speech might end up being reduced along with noise.

If one was to build on this project in the future, use of a larger speaker database is recommended in order to have more test speakers for use in experiments. Having a larger speaker database would allow training of a more solid Universal Background Model for the SID system as well as having more speakers left over to use in experiments (UBM speakers cannot be used in experiments for Open Set SID 2010). This methodology

would allow more test cases to average over in order to generalize results for better comparisons. For further evaluation and comparison, use of other speaker identification programs is also recommended.

Other areas of future work include testing the filters in an anechoic chamber to control the noise environment, implementing frequency domain multichannel filtering methods, exploring the effect of statistical estimations such as “forgetting factor” as well as using subjective human listening tests to provide another comparison metric. While multichannel signal processing has been around for awhile, there is still much research to be done for use with broadband speech signals. Overall, multichannel noise reduction for speaker identification has powerful potential and will continue being developed for applications from video chatting to top secret surveillance.

Bibliography

[1] J. Benesty, J. Chen and Y. Huang, Microphone Array Signal Processing, vol. 1, Heidelberg: Springer, 2008.

[2] S. P. Applebaum, "Adaptive Arrays," *IEEE Transactions on Antennas and Propogation,* vol. 24, no. 5, pp. 585-598, 1976.

[3] Contributors, "Speaker recognition," Wikipedia, 11 5 2012. [Online]. Available: http://en.wikipedia.org/wiki/Speaker_recognition. [Accessed 29 6 2012].

[4] S. Furui, "Speaker Recognition," Scholarpedia, 16 10 2007. [Online]. Available: http://www.scholarpedia.org/article/Speaker_recognition. [Accessed 29 6 2012].

[5] R. Rose, H. E. M. and D. Reynolds, "Integrated Models of Signal and Background with Application to Speaker Identification in Noise," *IEEE Transactions on Speech and Audio Processing,* vol. 2, no. 2, pp. 245-257, 1994.

[6] J. Chen, J. Benesty and Y. Huang, "A Minimum Distortion Noise Reduction Algorithm With Multiple Microphones," *IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING,* vol. 16, no. 3, pp. 481-493, March 2008.

[7] R. Cook, R. Waterhouse, B. R.D., S. Edelman and M. Thompson, "Measurement of Correlation Coeficients in Reverberent Sound Fields," *The Journal of the Acoustical Society of America,* vol. 27, no. 6, pp. 1072-1077, November, 1955.

[8] M. Vondrasek and P. Pollak, "Methods for Speech SNR estimation: Evaluation Tool and Analysis of VAD Dependency," *RADIO ENGINEERING,* vol. 14, no. 1, pp. 6-11, 2005.

[9] Air Force Research Laboratory, "Rome Audio Processing Tool (RAPT) Help System," Rome, NY, 2010.

[10] Anonymous, "The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus," University of Pennsylvania, [Online]. Available: http://www.ldc.upenn.edu/Catalog/readme_files/timit.readme.html.

[11] D. Chambers, "Array Beamforming," in *Biometrics Review,* Livermore, 2011.

[12] B. D. Jeffs, "Beamforming," Brigham Young University, Provo, Utah, March, 2008.

[13] S. Fischer and K. Simmer, "Beamforming Microphone Arrays for Speech Acquisition in Noisy Environments," *Speech Communication,* vol. 20, pp. 215-227, 1996.

[14] I. McCowan, J. Pelecanos and S. Sridharan, "Robust Speaker Recognition using Microphone Arrays," in *A Speaker Odyssey-The Speaker Recognition Workshop,* Creete, Greece, 2001.

[15] T. Brand, "Speech Intelligibility," in *Handbook of Signal Processing in Acoustics*, New York, Springer , 2009, pp. 197-204.

[16] Z. Cohen, "Noise Reduction with Microphone Arrays for Speaker Identification," LLNL report LLNL-PRES-562491, Livermore, 2011.

[17] Z. Cohen, "Noise Reduction with Microphone Arrays for Speaker Identification," LLNL report LLNL-TR-524752, Livermore, 2011.

APPENDIX A: TIMIT PROMPTS

She had your dark suit in greasy wash water all year.

Don't ask me to carry an oily rag like that.

This was easy for us.

Jane may earn more money by working hard.

She is thinner than I am.

Bright sunshine shimmers on the ocean.

Nothing is as offensive as innocence.

Why yell or worry over silly items?

Where were you while we were away?

Are your grades higher or lower than Nancy's?

He will allow a rare lie.

Will Robin wear a yellow lily?

Swing your arm as high as you can.

Before Thursday's exam, review every formula.

The museum hires musicians every evening.

A roll of wire lay near the wall.

Carl lives in a lively home.

Alimony harms a divorced man's wealth.

Aluminum silverware can often be flimsy.

She wore warm, fleecy, woolen overalls.

APPENDIX B: MATLAB CODE

Array_Geometry.m

```
%% .08m Spaced 10 Sensor 2D Linear Array (book)
prec = [ -.36 0 0 ;
-.28 0 0 ;
-.20 0 0 ;
-.12 0 0 ;
-.04 0 0 ;
.04 0 0 ;
.12 0 0 ;
.20 0 0 ;
.28 0 0 ;
.36 0 0 ];

%% 1m Spaced 2D linear array

prec = [ -4.5 0 0 ;
-3.5 0 0 ;
-2.5 0 0 ;
-1.5 0 0 ;
-.5 0 0 ;
.5 0 0 ;
1.5 0 0 ;
2.5 0 0 ;
3.5 0 0 ;
4.5 0 0 ];

%% Experimental 3D Array setup
prec = [ .956   0   .445;
         0      0   .458;
         -.915  0   .445;
         .956   0   1.368;
         0      0   1.358;
         -.915  0   1.278;
         .956   0   2.139;
         0      0   2.133;
         -.915  0   2.075];
%% .01m spaced linear array
prec = [ -.045 0 0 ;
-.035 0 0 ;
-.025 0 0 ;
-.015 0 0 ;
-.005 0 0 ;
.005 0 0 ;
.015 0 0 ;
.025 0 0 ;
.035 0 0 ;
.045 0 0 ];

%% .05m Spaced linear array
prec = [ -.225 0 0 ;
-.175 0 0 ;
```

```
-.125 0 0 ;
-.075 0 0 ;
-.025 0 0 ;
.025 0 0 ;
.075 0 0 ;
.125 0 0 ;
.175 0 0 ;
.225 0 0 ];

%% .02m spaced linear array
prec = [ -.09 0 0 ;
-.07 0 0 ;
-.05 0 0 ;
-.03 0 0 ;
-.01 0 0 ;
.01 0 0 ;
.03 0 0 ;
.05 0 0 ;
.07 0 0 ;
.09 0 0 ];

%% .015m spaced lin array
prec = [ -.0675 0 0 ;
-.0525 0 0 ;
-.0375 0 0 ;
-.0225 0 0 ;
-.0075 0 0 ;
.0075 0 0 ;
.0225 0 0 ;
.0375 0 0 ;
.0525 0 0 ;
.0675 0 0 ];

%% .012m Spaced Linear Array
prec = [ -.054 0 0 ;
-.042 0 0 ;
-.030 0 0 ;
-.018 0 0 ;
-.006 0 0 ;
.006 0 0 ;
.018 0 0 ;
.030 0 0 ;
.042 0 0 ;
.054 0 0 ];

%% 1m Spacing 3 Sensor linear array

prec = [-1 0 0 ;
        0 0 0;
        1 0 0];
%% .8m Spaced 3 Sensor linear array
prec = [-.8 0 0 ;
        0 0 0;
        .8 0 0];
%% .5 Spaced 3 Sensor linear array
prec = [-.5 0 0 ;
```

```
        0 0 0;
        .5 0 0];
 %% .3 Spaced 3 Sensor linear array
prec = [-.3 0 0 ;
        0 0 0;
        .3 0 0];
%% .2 Spaced 3 Sensor linear array
prec = [-.2 0 0 ;
        0 0 0;
        .2 0 0];
%% .7 Spaced 3 Sensor linear array
prec = [-.7 0 0 ;
        0 0 0;
        .7 0 0];
%% .6 Spaced 3 Sensor linear array
prec = [-.6 0 0 ;
        0 0 0;
        .6 0 0];
```

Array_preprocess.m

```
function [] = Array_preprocess(datadir,datafile,FSout,fcutoff)
% function [] = Array_preprocess(datadir,datafile,Fsout,fcutoff)
% This function preprocesses the array data, by downsampling and high
pass
% filtering. Inputs are:
%      datadir: data directory
%      datafile: data file (including .mat extension)
%      FSout: final sampling frequency
%      fcutoff: cutoff frequency for detrending (high pass filter)
% Output file name is input file name with _PP appended.

    dfheader = datafile(1:(end-4));
    outfile = [dfheader '_PP.mat'];

    load(fullfile(datadir,datafile));
    FSin = 1/hDelta;
    ndet = 2*round(.5*FSin/fcutoff);

    [TimeDatadt,~] = detrend_filt(TimeData,ndet);
    TimeDataPP = fractional_downsample(TimeDatadt,FSin,FSout);
    [nt,~] = size(TimeDataPP);
    time = (0:(nt-1))'/FSout;
    FS = FSout;

    save(fullfile(datadir,outfile),'time','FS','TimeDataPP');

end
```

BB_Filter_Metrics.m

```
function [ z_v_filt, SNR_in, SNR_out, SSNR_out, nr_factor, sd_factor,
P_out, P_out_n, P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src, noise,
L,overlap, H, z_k )
```

```
%Calculates broadband filter performance metrics given noisy speech and
%noise only mic array outputs.
%   src = N rows/channels of noise+speech signal with any length
%   noise = N rows/channels of noise only signal with any length
%   L = length of frame
%   overlap = # of sample overlap of frame
%   H = calculated filter matrix
%   z_k = single channel beamformed filter output (1x..)

N = size(src,1); % Calculate number of mics in array



for i = 1:L-overlap:length(noise)

    if(L+i-1 <= length(noise)) %if current block will exceed length of
input array, break loop

    v_L = noise(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

    %Organize NxL matrix containing sample blocks into NLx1 matrix
    v_k = v_L(:);

    z_v_filt(i:i+L-1,1) = H*v_k;
    end
end

P_out_n_SEG = 0;
P_out_SEG = 0;
m = 0;
n = 0;
for i = 1:L-overlap:max(length(noise),length(src))

    if(L+i-1 <= length(z_v_filt)) %if current block will exceed length
of input array, break for loop
        m = m+1;
        P_out_n_SEG = ((m-1)/m)*P_out_n_SEG + var(z_v_filt(i:L+i-
1))./m;
    end

    if(L+i-1 <= length(z_k))
        n = n+1;
        P_out_SEG = ((n-1)/n)*P_out_SEG + var(z_k(i:L+i-1))./n;
    end

end
SSNR_out = (P_out_SEG - P_out_n_SEG)./P_out_n_SEG;

P_out = var(z_k);
P_out_n = var(z_v_filt);
P_sig1 = var(src(1,:)) - var(noise(1,:));

SNR_in = P_sig1./var(noise(1,:));
```

```
SNR_out = (P_out./P_out_n)-1;

nr_factor = var(noise(1,:))./P_out_n;



P_out_sig1 = P_out - P_out_n;

sd_factor = abs(P_out_sig1 - P_sig1)./P_sig1;

end
```

dir_response.m

```
function [ dir_resp, tau_scan ] = dir_response( look_dir, W, prec, f,
x_scan )
%[ dir_resp, tau_scan ] = dir_response( look_dir, W, prec, f, x_scan )
%   Creates beamplot directional response for an arbitrary array
geometry
if length(look_dir) == 1
    look_angle = look_dir.*(pi/180);
else
    look_angle = atan(look_dir(2)/look_dir(1));
end

c = 345;
N = size(prec, 1); % number of microphones

for i=1:N
    d(i) = sqrt( (prec(i,1)-look_dir(1))^2 + (prec(i,2)-look_dir(2))^2
+ (prec(i,3)-look_dir(3))^2);
end

%calc time delay between source and receivers
td = d./c;
% Calc the minimum delay corresponding to the closet microphone to the
source
td_min = min(td); % Calc the minimum delay corresponding to the closet
microphone to the source

tau = td - td_min;

for k = 1:length(x_scan)
    for l = 1:N
    d_scan(k,l) = sqrt(sum((prec(l,:) - [x_scan(k) look_dir(2)
look_dir(3)]).^2));
    end
    d_min_scan = min( d_scan(k,:));
    tau_scan(k,:) = (d_scan(k,:) - d_min_scan)./c;
end
```

```
for k = 1:length(x_scan)
    for l = 1:N
        dir_resp_W(k,l) = (1./N)*W(l).*(exp(-j*2*pi*f.*(tau_scan(k,l)-
tau(l))));
    end
end

dir_resp = sum(dir_resp_W, 2);

figure()
plot(x_scan, 10.*log10(abs(dir_resp)))


end
```

DS_beamformer.m

```
function [ ya, z, pout ] = DS_beamformer( x, Fs, look_dir, W, prec )
%[ ya, z, pout ] = DS_beamformer( x, Fs, look_dir, W, prec )
%   x = actual samples at each time step (NxL matrix) N = # of
receivers L
%       = # of samples. Each column corresponds to one sample time.
%   Fs = sample rate
%   look_dir = coordinates [x y z] for beamformer to focus on
%   W = channel weighting vector 1xN (# of mics) ex: [1 2 1]
%   prec = 3D coordinate postions of each microphone [x y z]

%   ya = delayed/aligned samples corresponding to beamformer look
direction
%   z = summed output of all delayed/aligned samples [1xL vector]
%   pout = output power of z for designated look direction
%

%   generate time series

L = size(x,2); % # of samples L

n = (0:(L-1)); %sample index #

Ts = 1./Fs; %sampling period

t_tx = n.*Ts; % create source time series starting at t = 0

c = 345;
N = size(prec, 1);  %calculate number of receivers

%calc distance between look point and current receiver
for i=1:N
d(i) = sqrt( (prec(i,1)-look_dir(1))^2 + (prec(i,2)-look_dir(2))^2 +
(prec(i,3)-look_dir(3))^2);
end
```

```
%calc time delay between source and receivers
td = d./c
% Calc the minimum delay corresponding to the closet microphone to the
source
td_min = min(td)

tau = td - td_min; % convert to time delay rewlative to closest mic

k = round(tau./Ts) % convert relative time delay to equivalent sample
delay
k_max  = max(k);

%align samples according to specified BF look direction
for i = 1:N
    ya(i,:) = W(i) .* x(i, ((k(i)+1):(L- k_max + k(i))));
end

%sum all aligned samples to calculate output of beamformer
z = sum(ya)./N;

pout = var(z) % calculate output power of signal

end
```

DS_freq_response.m

```
function [ beam_pwr ] = DS_freq_response( f_sweep, x_scan, Fs, psrc,
prec, W )
%[ beam_pwr ] = DS_freq_response( f_sweep, x_scan, Fs, psrc, prec, W )
%   Detailed explanation goes here

L = length(f_sweep);


 for i = 1:L % generate sinusoid at each frequency of f_sweep
    sine = sine_gen(1, f_sweep(i), Fs, 3);

    % simulate each sinusoid with mic array
    [x, ~, ~] = sim_3D( sine, Fs, psrc, prec, 0);

    %scan all x axis values on a line and look at BF output power
    for j = 1:length(x_scan)                          %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x, Fs, [x_scan(j) psrc(2)
psrc(3)], W, prec);
     beam_pwr(i,j) = pout_scan; %beampower matrix dependant on x
position and frequency
    end
 end

 figure()

 mesh(x_scan, f_sweep, beam_pwr./.5)
```

```
 title('Beamformer Frequency Response')
 xlabel('x direction (m)')
 ylabel('frequency (Hz)')
 zlabel('Gain')

 figure()

 title('Broadband Directional Response')
 plot(x_scan, sum(beam_pwr./.5)./length(f_sweep))

 xlabel('x direction (m)')
 ylabel('Gain')

end
```

DS_lin_dir_response.m

```
function [ P_sig, P_ds ] = DS_lin_dir_response( look_dir, sweep_angles,
N, d ,f, var_n )
%Plots 2D directional response beampattern of an equispaced linear
array with
%equal weights (DS beamformer)
%   [ P_sig, P_ds ] = lin_dir_response( look_dir, sweep_angles, c, N, d
,f, var_n )
% look_dir = look direction /steered direciton of beamformer (degrees)
% sweep_angles = directions evaluated for beampattern response 1xL
(degrees)
% N = # of receivers
% d = distance between each microphone
% f = signal frequency
% var_n = noise variance for noise field model
% P_sig = signal power for each sweep angle
% P_ds = signal + noise for each sweep angle


c = 345;
k = (2*pi*f/c);
theta = look_dir*(pi/180); % beamformer look direction
psi = sweep_angles*(pi/180); %beamformer directional response sweep
angles

% calculate directional signal power from derived formula
P_sig = abs(sin(pi*f*N*d.*(cos(psi)-
cos(theta))./c)./(N*sin(pi*f*d.*(cos(psi)-cos(theta))./c))).^2;

% calculate noise field correlation matrix
for m = 1:N
    for n =1:N
        d_mn = d*abs(m-n);
        R_mn(m,n) = (var_n/(2*pi))* besselj(0,k*d_mn);
    end
end
```

```
%Calculate noise power
P_n = (1/N)^2 * sum(sum(R_mn));

%Add signal and noise power together to get total DS beamformer power
P_ds = P_sig + P_n;

figure()
plot(psi.*(180/pi), 10.*log10(P_ds),psi.*(180/pi), 10.*log10(P_sig))
title('Delay Sum Beamformer Directional Response w/ Added Noise')
xlabel('direction (degrees)')
ylabel('Gain (dB)')
legend ('Signal + Correlated Noise','Signal','Location', 'South')

end
```

DS_metrics.m

```
function [ SNR_out, nr_factor, sd_factor, SNR_in, P_out, P_out_n,
P_out_sig1, P_sig1, z_DS, z_n ] = DS_metrics( src, noise, psrc, prec,FS
)
%Calculates output metrics for DS beamformer run on source and noise
%   only signals with known look direction
%
%Run DS Beamformer on src and noise
[ ~, z_DS, pout ] = DS_beamformer( src, FS, psrc, ones(1,9), prec );
[ ~, z_n, pout_n ] = DS_beamformer( noise, FS, psrc, ones(1,9), prec );

%Calculate metrics
    P_in = var(src(1,:));
    P_n = var(noise(1,:));
    P_sig1 = P_in-P_n;
    SNR_in = P_sig1./P_n;

    P_out = pout;
    P_out_n = pout_n;
    P_out_sig1 = pout-pout_n;



    SNR_out = (pout-pout_n)./pout_n;
    nr_factor = var(noise(1,:))./P_out_n;
    sd_factor = abs(P_out_sig1 - P_sig1)./P_sig1;


end
```

DS_metrics_sweep.m

```
%%% DS metrics Sweep

Gain = 1;
```

```
% set path where all preprocessed test cases are located
datadir = fullfile('Outdoor_Perimeter_11172011');

% set path where all noise only recording are found
datadir_noise = fullfile('Outdoor_Perimeter_11172011','Noise');

% set path where array measurments are found
datadir_src = fullfile('Outdoor_Perimeter_11172011','Array
Measurements');
load(fullfile(datadir_src,'Array_Meas_11_17'))

% Load file containing the list of the names of all of the case files
load(fullfile(datadir,'DataList.mat'))

for i = 1:length(datalist)
    current_case = datalist(i).name; %extract the name of the current
case
    display(['Processing ' current_case])

    load(fullfile(datadir,current_case)) %load the data from the
current case

    src = Gain.*TimeDataPP; %amplify signal
    clear TimeDataPP
    clear time

    noise_header = current_case(1:end-8);
    load(fullfile(datadir_noise, [noise_header 'noise_PP.mat'])) %load
noise that corresponds to the current case

    noise = Gain.*TimeDataPP; %amplify noise signal
    clear TimeDataPP
    clear time

    if i == 1
        src_pos = psrc(1,:);
    end
    if i ==6
        src_pos = psrc(2,:);
    end
    if i == 11
        src_pos = psrc(3,:);
    end
    if i == 16
        src_pos = psrc(4,:);
    end
    if i == 21
        src_pos = psrc(5,:);
    end
    if i == 26
        src_pos = psrc(6,:);
    end
    if i == 31
        src_pos = psrc(7,:);
```

```
    end

    [ SNR_out, nr_factor, sd_factor, SNR_in, ~, ~, ~, ~, z_DS, z_n ] =
DS_metrics( src', noise', src_pos, prec,FS );

    save(fullfile('DS Beamformer', ['DS_' current_case(1:end-7)]
),'SNR_in','SNR_out','nr_factor','sd_factor','z_DS','z_n','FS','src','n
oise','Gain','current_case','src_pos');
    wavwrite((1./max(abs(z_DS))).*z_DS, fullfile('BF out
wavs',[current_case(1:end-7) '_DSBF']))
    clear SNR_in SNR_out nr_factor sd_factor z_DS z_n current_case src
noise FS
end
```

DS_noise_tests.m

```
%% Noise Analysis for DS Beamformer Script

c = 345; % speed of sound m/s
N = 9; % # of receivers
d = .08; % distance between receivers in linear array
f = 100:4000; %frequency for beamplot Hz
var_n = 1; % noise variance
theta = 90*(pi/180); % beamformer look direction
psi = (0:.99:180)*(pi/180); %beamformer directional response sweep
angles

for i = 1:length(f)
k = (2*pi*f(i)/c)

% calculate directional signal power from derived formula
P_sig = abs(sin(pi*f(i)*N*d.*(cos(psi)-
cos(theta))./c)./(N*sin(pi*f(i)*d.*(cos(psi)-cos(theta))./c))).^2;

%calculate noise correlation matrix      samples     var  L
%[ noise_corr, R_corr ] = Spacial_noise_R( 1000, prec, var_n,  10);

for m = 1:N
    for n =1:N
        d_mn = d*abs(m-n);
        R_mn(m,n) = (var_n/(2*pi))* besselj(0,k*d_mn);
    end
end

R_un = var_n.* eye(N);

%Calculate noise power
P_n = (1/N)^2 * sum(sum(R_mn));
P_n_un = (1/N)^2 * sum(sum(R_un));

%Add signal and noise power together to get total DS beamformer power
```

```
P_ds(i,:) = P_sig + P_n;
P_ds_un(i,:) = P_sig + P_n_un;

% figure()
% plot(psi.*(180/pi), 10.*log10(P_ds),psi.*(180/pi),
10.*log10(P_sig),psi.*(180/pi), 10.*log10(P_ds_un))
% title('Delay Sum Bemaformer Directional Response w/ Added Noise')
% xlabel('direction (degrees)')
% ylabel('Gain (dB)')
% legend ('Signal + Correlated Noise','Signal', 'Signal + Uncorrelated
Noise','Location', 'South')

end

figure()
 mesh(psi*(180/pi),f,10*log10(P_ds));
 title('new noise model')
 xlabel('Direction (degrees)')
 ylabel('frequency (Hz)')
 zlabel('gain (dB)')

 figure()
 mesh(psi*(180/pi),f,10*log10(P_ds_un));
 title('white noise')
 xlabel('Direction (degrees)')
 ylabel('frequency (Hz)')
 zlabel('gain (dB)')
```

DS_Simulation1_noiseBW.m

```
%%% DS Simulation Test 1 %%
%%% varying noise BW to be compared with wiener and ST approach.

prec = [ .956   0   .445;
         0      0   .458;
         -.915  0   .445;
         .956   0   1.368;
         0      0   1.358;
         -.915  0   1.278;
         .956   0   2.139;
         0      0   2.133;
         -.915  0   2.075];

psrc = [0 40 0];
[ src, ~, src_pwr ] = sim_3D( bftest0, FS, psrc, prec );
fc = [ 50 100 200 300 500 800 1000 2000 3000 ];

pout = zeros(1,length(fc));
pout_n = zeros(1,length(fc));
SNR_out = zeros(1,length(fc));
for i = 1:length(fc)
    current_fc = fc(i)
    [ noise_corr, R_vv ] = plane_noise_R_3D( prec, src_pwr,
length(src), fc(i),7,FS );
```

```
    x = src+noise_corr;
    [ ~, z, pout ] = DS_beamformer( x, FS, psrc, ones(1,9), prec );
    [ ~, z_n, pout_n ] = DS_beamformer( noise_corr, FS, psrc,
ones(1,9), prec );
    pout_all(i) = pout;
    pout_n_all(i) = pout_n;
    SNR_out(i) = (pout-pout_n)./pout_n;

end
```

DS_Simulation2_B123_noise_bftest0.m

```
%%% DS Simulation Test 2 %%
%%% Simulation With bftest0 src + B123 noise (recorded)

%Inititalize
FS = 8000;
prec = [ .956   0   .445;
         0      0   .458;
         -.915  0   .445;
         .956   0   1.368;
         0      0   1.358;
         -.915  0   1.278;
         .956   0   2.139;
         0      0   2.133;
         -.915  0   2.075];

psrc = [0 10 0];
[ x1, ~, src_pwr ] = sim_3D( bftest0, FS, psrc, prec );

src = .5.*x1+noise(1:length(x1),:)';

%Run DS Beamformer on src and noise
[ ~, z, pout ] = DS_beamformer( src, FS, psrc, ones(1,9), prec );
[ ~, z_n, pout_n ] = DS_beamformer( noise', FS, psrc, ones(1,9), prec
);

%Calculate metrics
    P_out = pout;
    P_out_n = pout_n;
    P_out_sig1 = pout-pout_n;
    P_sig1 = var(src(1,:))-var(noise(:,1));
    SNR_in = (var(src(1,:))-var(noise(:,1)))./var(noise(:,1));

    SNR_out = (pout-pout_n)./pout_n;
    nr_factor = var(noise(:,1))./P_out_n;
    sd_factor = abs(P_out_sig1 - P_sig1)./P_sig1;
```

fractional_downsample.m

Code credit: Dave Chambers, LLNL

```
function yout = fractional_downsample(yin,Fsin,Fsout,tol)
% function yout = fractional_downsample(yin,Fsin,Fsout,tol)
% This function combines decimate and resample to downsample yin from
Fsin
% to Fsout. Inputs are:
%     yin: input signal
%     Fsin: sample rate for yin (Hz)
%     Fsout: desired output sample rate (Hz)
%     tol: tolerance for rational approximation to Fsin/Fsout
(optional)

    if nargin < 4
        tol = .001;
    end
    [~,ncol] = size(yin);
%    yin = yin(:);
%    ny = length(yin);
    nd = floor(Fsin/Fsout);
    [p,q] = rat(nd*Fsout/Fsin,tol);

    if nd>0
        ydec = decimate(yin(:,1),nd);
    else
        ydec = yin(:,1);
    end
    yout1 = resample(ydec,p,q);
    nyout = length(yout1);
    if ncol>1
        yout = zeros(nyout,ncol);
        yout(:,1) = yout1;
        for j=2:ncol
            if nd>0
                ydec = decimate(yin(:,j),nd);
            else
                ydec = yin(:,j);
            end
            yout1 = resample(ydec,p,q);
            yout(:,j) = yout1;
        end
    else
        yout = yout1;
    end

end
```

Init_3D_SIM.m

```
%% Initialize Sournce and receiver locations
clear variables
%%
psrc = [0 40 0];
pnoise = [0 10 1];
prec = [ .956   0   .445;
         0      0   .458;
```

```
          -.915  0   .445;
          .956   0   1.368;
          0      0   1.358;
          -.915  0   1.278;
          .956   0   2.139;
          0      0   2.133;
          -.915  0   2.075];
W = [1 1 1 1 1 1 1 1 1];
x_scan = linspace(-20,20,300);


 %% load sounds
 load('handel');
 handel = y;
 bftest0 = wavread('bftest0');
 bftest1 = wavread('bftest1');
 sine = sine_gen(1, 2000, 40000, 1);                          %var
%gausian width
 [ noise_corr, R_corr ] = Spacial_noise_R( length(x1), prec, .1,  1.5);

 %% Execute Simulation

 %% Single source
 [ x1, ~, src_pwr ] = sim_3D( bftest0, Fs, psrc, prec );

 %% Dual Sources
 %src 1
 [ x1, ~, src_pwr ] = sim_3D( bftest0, Fs, psrc, prec );

 %src2
 [ x2, ~, src_pwr2 ] = sim_3D( handel, Fs, pnoise, prec );

 min_L = min([size(x1,2) size(x2,2)]);

 x1 = x1(:,1:min_L);
 x2 = x2(:,1:min_L);
 x = x1+x2;

 %% Plot
 x_scan = linspace(-10,10,300);               % y_plane z_plane
weighting vector
 [beam_pwr] = x_beam_plot( noise, Fs, prec, x_scan, psrc(2), psrc(3),
W);

 %% Run DS beamformer focused on each source

 [ ~, z_DS, ~ ] = DS_beamformer( x, Fs, psrc, W, prec );

 [ ~, z_SNR, ~ ] = DS_beamformer( x, Fs, psrc2, h_max', prec );

 sound(x(1,:))
 sound(z_DS)
 sound(z_SNR)
```

```
 %% Run Freq Resp
 f_sweep = linspace(80, 3000, 20);
 x_scan = linspace(-10, 10, 300);
 [ beam_pwr ] = DS_freq_response( f_sweep, x_scan, Fs, psrc, prec,
h_max');
```

Init_SIM_script.m

```
%% Load Hallelujah
load handel

%% Initialize Source Location
psrc = [0 20]

%% Initialize Receiver Locations
p1= [-2.5 0]
p2= [-1.5 0]
p3 = [-.5 0]
p4 = [.5 0]
p5 = [1.5 0]
p6 = [2.5 0]

%% Run Simulator
xy_sim_6( y, Fs, psrc, p1, p2, p3, p4, p5, p6 );
```

lin_dir_response.m

```
function [ P_bf, z ] = lin_dir_response( look_dir, sweep_angles, f, W,
N  )
%Computes and plots the 2D directional repsonse of a linear array with
equal
%spacing amnd user defined channel weightings W.
%
%NOTE: some code used courtesy of Brian D. Jeffs
% Associate Professor
% Dept. of Electrical and Computer Engineering
% Brigham Young University
% March 2008
%
%[ output_args ] = lin_dir_response( look_dir, sweep_angles, f, W, N  )
%
% N =  % no. of array elements
% f = frequency (Hz)
c = 345;

d = c/f/2; % element spacing
look_dir = look_dir*(pi/180);
d_s = exp(j*2*pi*f*d/c*cos(look_dir)*[0:N-1]).';
R_s = d_s*d_s';

% compute steering vector samples for beam resp. plot
psi = sweep_angles.'*pi/180;
```

```
D_b = exp(j*2*pi*f*d/c*cos(psi)*[0:N-1]).';

% conventional windowed beamformer case
h = d_s.*W;
z = h'*D_b./sum(abs(W)); % beam response after weighting
P_bf = abs(z).^2;

figure()
plot(sweep_angles,10*log10(P_bf))
title( 'Directional Response for Weighted Beamformer')
xlabel('direction (degrees)')
ylabel( 'Gain (dB)')
end
```

max_SNR.m

```
function [h_max, SNR_max, SNR_DS, SNR_mSNR, z_src ] = max_SNR( psrc,
prec, src, noise_corr, Fs, x_scan)
% %[h_max, SNR_max, SNR_DS, SNR_mSNR ] = max_SNR( psrc, prec, src,
noise_corr, Fs, x_scan)
%     psrc = source position [x y z]
%     prec = receiver positions Nx3 [x y z]
%     src = source sound 1xL
%     noise_corr = NxL matrix of spatially correlated noise located at
x = 0
%     Fs = sampling rate
%     x_scan = x position samples for output plots
%
%     h_max = Nx1 array of calculated weights for each receiver channel
%     SNR_max = eigenvalue corresponding to h_max eigen vector
%     SNR_DS = signal to noise ratio of unity weighted DS beamformer at
%       source look direction
%     SNR_mSNR = signal to noise ratio of Max SNR algorithm using h_max
as
%       DS weights

N = size(prec,1);

a (1:N) = 1;
aat = a'*a;
W(1:N) = 1;

%src simulation
[ x1, ~, ~ ] = sim_3D( src, Fs, psrc, prec );

%[ noise, ~, ~ ] = DS_beamformer( noise_corr, Fs, -1.*pnoise, W, prec
);

%superposition source and noise signals
%force matrices to be same length
min_L = min([size(x1,2) size(noise_corr,2)]);

x1 = x1(:,1:min_L);
```

```
x2 = noise_corr(:,1:min_L);

x = x1+x2;

% align/delay noise source samples in direction of the source
[ va, ~, ~ ] = DS_beamformer( noise_corr, Fs, psrc, W, prec );

%STAT CALCS
% Find Eigenvector that corresponds to max eigenvalue of constraint eqn
R_vv = corr( va');
R_vv_inv = inv(R_vv);
A = var(src).*R_vv_inv*aat; %Matrix for eigenvalue calc.... A*x = SNR*x
[h_max, SNR_max] = eigs(A,1)

% Evaluate DS response
%calculate received power from the source as a function of position
for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x1, Fs, [x_scan(i) psrc(2)
psrc(3)], W, prec );
     beam_pwr_sDS(i) = pout_scan;
end

%calculate received power from the noise as a function of position
for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x2, Fs, [x_scan(i) psrc(2)
psrc(3)], W, prec );
     beam_pwr_nDS(i) = pout_scan;
end

%calculate total received power from at the output of the beamformer
for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x, Fs, [x_scan(i) psrc(2)
psrc(3)], W, prec );
     beam_pwr_DS(i) = pout_scan;
end

% Calculate DS SNR
[ ~, ~, pout_sDS ] = DS_beamformer( x1, Fs, psrc, W, prec );
[ ~, ~, pout_nDS ] = DS_beamformer( x2, Fs, psrc, W, prec );

SNR_DS = 10.*log10(pout_sDS./pout_nDS);

% Evaluate Max SNR response

for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x1, Fs, [x_scan(i) psrc(2)
psrc(3)], h_max', prec );
     beam_pwr_sSNR(i) = pout_scan;
end

for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x2, Fs, [x_scan(i) psrc(2)
psrc(3)], h_max', prec );
     beam_pwr_nSNR(i) = pout_scan;
end
```

```
for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x, Fs, [x_scan(i) psrc(2)
psrc(3)], h_max', prec );
     beam_pwr_SNR(i) = pout_scan;
end

%Calc actual Max SNR SNR
[ ~, z_src, pout_sSNR ] = DS_beamformer( x1, Fs, psrc, h_max', prec );
[ ~, z_noise, pout_nSNR ] = DS_beamformer( x2, Fs, psrc, h_max', prec
);

SNR_mSNR = 10.*log10(pout_sSNR./pout_nSNR);

% Plot DS response

figure()
plot(x_scan, beam_pwr_sDS, x_scan, beam_pwr_nDS, 'r')
title('DS Source and Noise directional response')
xlabel('x location (m)')
ylabel('Beam power (W)')

% figure()
% plot(x_scan, beam_pwr_DS)
% title('DS total output directional response')
% xlabel('x location (m)')
% ylabel('Beam power (W)')

%Plot Max SNR response

figure()
plot(x_scan, beam_pwr_sSNR, x_scan, beam_pwr_nSNR, 'r')
title('Max SNR Source and Noise directional response')
xlabel('x location (m)')
ylabel('Beam power (W)')


% figure()
% plot(x_scan, beam_pwr_SNR)
% title('Max SNR total output directional response')
% xlabel('x location (m)')
% ylabel('Beam power (W)')

% figure()
% plot(x_scan, (beam_pwr_nSNR./beam_pwr_nDS))
% title('Noise Gain Directional Response')
% xlabel('x position (m)')
% ylabel('noise gain')

figure()
subplot(2,1,1)
plot(0:1/Fs:(length(src)-1)*(1/Fs),src)
title('Original Source Signal')
xlabel('time (s)')
```

```
ylabel('Amplitude')

subplot(2,1,2)
plot(0:1/Fs:(length(z_src)-1)*(1/Fs), z_src)
title('Source Signal After Beamforming')
xlabel('time (s)')
ylabel('Amplitude')

end
```

plane_noise_R.m

```
function [ R_vv ] = plane_noise_R( prec, look_dir, var_n, f )
%Builds a NxN correlation matrix for random directional plane wave
noise
%model.
%   prec = receiver postions [x1 y1 z1; ... xN yN zN]
%   look_dir = point/ direction that the beamformer is looking [x y z]
%   var_n = noise variance scale factor for correlation coeficients
%   f = operating frequency of  beamformer

% Calculate constants
N = size(prec,1); % number of mics
c = 345; %speed of sound
k = 2*pi*f/c; %wave number

%calculate distances between receivers and put into NxN matrix
for m = 1:N
    for n = 1:N
        d_mn(m,n) = sqrt( sum((prec(m,:) - prec(n,:)).^2) );
    end
end

%calculate distances from each mic to look direction [x y z]
for i = 1:N
    d(i) = sqrt( sum((look_dir - prec(i,:)).^2));
end

%convert distances to each mic into relative time delays for each mic
td = (d-min(d))./c;

% calculate noise field correlation matrix
for m = 1:N
    for n =1:N
        R_vv(m,n) = var_n*
besselj(0,k.*d_mn(m,n)).*cos(2*pi*f*abs((td(n)-td(m))));
    end
end

end
```

plane_noise_R_3D.m

```
function [ noise_corr, R_vv ] = plane_noise_R_3D( prec, var_n, length,
fc,order,Fs )
%Creates spatially correlated low pass filtered (correlated in time)
noise
%   prec = receiver postions [x1 y1 z1; ... xN yN zN]
%   var_n = variance of the noise
%   length = # oof samples of the created noise
%   fc = upper cutoff frequency of the desired noise (Hz)

% Calculate constants
N = size(prec,1); % number of mics
c = 345; %speed of sound
kc = 2*pi*fc/c; %wave number of upper cutoff frequency

%Correlate random white noise in time by low pass filtering
[B_lp,A_lp] = butter(order, (2/Fs)*fc);

noise =  randn(N,length);
noise_lp = (filtfilt(B_lp, A_lp ,noise'))';


%calculate distances between receivers and put into NxN matrix
d_mn = zeros(N,N);
for m = 1:N
    for n = 1:N
        d_mn(m,n) = sqrt( sum((prec(m,:) - prec(n,:)).^2) );
    end
end

% calculate noise field correlation matrix
R_vv = zeros(N,N);
for m = 1:N
    for n =1:N
        if d_mn(m,n) == 0
            R_vv(m,n) = 1;
        else
            R_vv(m,n) = sinint(kc.*d_mn(m,n))./(kc*d_mn(m,n));
        end
    end
end

% Spatially Correlate the noise using R_vv
norm = mean(var(noise_lp'));
noise_corr = sqrt((1/norm)).*sqrt(var_n).*sqrtm(R_vv)*noise_lp;
end
```

Preprocess_data.m

Code credit: Dave Chambers, LLNL

```
% Preprocess_data_11172011
% Preprocessing script for array data for outdoor 11/17/2011 data
collect
```

```
datadir = fullfile('Outdoor_Perimeter_11172011','Noise');
load(fullfile(datadir,'DataList.mat'))
ndata = length(datalist);
for j=1:ndata
    datafile = datalist(j).name;
    disp(['Processing ' datafile])
    Array_preprocess(datadir,datafile,8000,150);
end
clear j datafile ndata
```

receiver.m

```
function [ t_tx, t_rx, snd_rx ] = receiver( snd_tx, Fs, d, c, a)
%[ t_tx, t_rx, snd_rx ] = receiver( snd_tx, Fs, d, c)
%   sound = amplitude samples of sound file from wave (-1 : 1)
%   Fs = sampling rate of source sound (samples/second)
%   d = distance between source and receiver (m)
%   c = speed of sound wave (345 m/s)
%   a = attenuation facor 0:1
%   t_tx = transmitted time series (s)
%   t_rx = received signal time series
%   td = time delay from source to receiver

%   generate time series

N = length(snd_tx); % of samples N

n = (0:(N-1)); %sample index #

Ts = 1./Fs %sampling period

t_tx = n.*Ts;

%   aquire received signal

td = d./c; %time delay from source to receiver

t_rx = t_tx + td %calc received signal time series

snd_rx = a.*snd_tx; % account for attenuation

figure (1)
title('source to receiver time delay')

subplot(2,1,1)
plot (t_tx, snd_tx)
xlabel('time [sec]')
ylabel('Amplitude')
title('Source Signal')

subplot(2,1,2)
plot(t_rx, snd_rx)
xlabel('time [sec]')
ylabel('Amplitude')
```

```
title('received signal')

end
```

rect_polar.m

```
function [ xy ] = rect_polar( r, angle_sweep )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

theta = angle_sweep*(pi/180);

x = r*cos(theta);
y = r*sin(theta);

xy = [x; y]';

end
```

semilogx_spectrum.m

```
function [ f, Y_f ] = semilogx_spectrum( y_t,FS )
%Plots the Spectrum of y_t with same number of points as y_t
%   Detailed explanation goes here

f = FS*(0:1./(length(y_t)):1-(1./length(y_t)));
Y_f = abs(fft(y_t))./length(y_t);
semilogx(f,Y_f);
end
```

sim_3D.m

```
function [ x, ya_sim, src_pwr ] = sim_3D( snd_tx, Fs, psrc, prec,
dist_atten)
%[ x, ya_sim, src_pwr ] = sim_3D( snd_tx, Fs, psrc, prec, dist_atten)
%  Calculates outputs of a microphone array with one source in 3D
%   snd_tx = samples of source signal (Lx1)
%   Fs = Sampling rate or sound source
%   psrc = source position [x y z]
%   prec = microphone positions Nx3 matrix: row = source #, columns = x
y z
%   dist_atten = accounts for 1/r attenuation due to source to array
%       distance. Leave empty if no attenuation is desired. Enter '1'
if
%       attenuation is desired
%
%   x = received sound samples at each corresponding mic (each row is a
different receiver
```

```
%       each column corresponds to sample times)
%   ya_sim = simulated aligned output values after added noise (if any)
%   src_pwr = calculates the power of the source signal
if nargin == 4
    dist_atten = 0;
end

%   generate time series

L = length(snd_tx); % # of samples in source signal

n = (0:(L-1)); %sample index #

Ts = 1/Fs; %sampling period

t_tx = n.*Ts; % create source time series starting at t = 0

%Calculate source power
src_pwr = var(snd_tx);

%Calc source -> receiver distances
c = 345;
N = size(prec, 1); % number of microphones

for i=1:N
    d(i) = sqrt( (prec(i,1)-psrc(1))^2 + (prec(i,2)-psrc(2))^2 +
(prec(i,3)-psrc(3))^2);
end

if(min(d) < 1)
    error('Source cannot be closer than one meter to the array!')
end

% Calculate attenuation factor
a = 1./d;
if(~dist_atten)
a(:) = 1;
end

%calc time delay between source and receivers
td = d./c;

% Calc the minimum delay corresponding to the closet microphone to the
source
td_min = min(td); % Calc the minimum delay corresponding to the closet
microphone to the source

%calculate time series for each receiver where t = 0 is when the source
signal first
% hits the closest mic to the source
for i = 1:N
    t(i,:) = td(i) + t_tx - td_min;
end
```

```
%shift data to correspond with time indicies
%data starts when source signal first hits the closest microphone
%removes preceding '0's due to initial source to receiver delay
for i = 1:N
    x(i, round((t(i,:)/Ts) +1)) = a(i).*snd_tx;
end

K = length(x);
t_rec = (0:Ts:(K-1)*Ts);

%add noise here
% y = x + n
y = x;

%align array of signals corresponding to the source look direction
for i = 1:N
    ya_sim(i,:) = y(i,round((t(i,:)/Ts) +1));
end

%Plot locations of source and receivers
% figure (1)
%
% hold on
% grid on
%
% title('Source and Receiver Loacations')
%
% s = scatter3(psrc(1),psrc(2), psrc(3), 'g', 'filled');
% r = scatter3(prec(:,1),prec(:,2),prec(:,3), 'b', 'filled');
view(30,20);
%
% xlabel('x-direction (m)')
% ylabel('y-direction (m)')
% zlabel('z-direction (m)')
% legend([s r], 'Source', 'Receivers')
% hold off
%
% %plot received signals for each mic
% figure (2)
% hold on
% title('Received Signals')
%
% for i = 1:N
% ax(i) = subplot(N,1,i);
% plot(t_rec,x(i,:))
% label = ['Receiver ' num2str(i)];
% ylabel(label)
% end
%
% linkaxes(ax, 'xy')
% hold off
 end
```

sim_dir_resp.m

```
function [ P_out,P_sig, P_out_n, P_in_n, src_pwr ] = sim_dir_resp( src,
Fs, look_dir, prec, W, angle_sweep, var_n, f )
%Plots the actual simulated directional resopnse of an input signal
%(preferreably a sinusoid) by varying the actual sourec location while
%keeping the beamformer looking in a specified direction.
%   src = source signal of any length
%   Fs = sampling rate of source signal (samples/sec)
%   look_dir = beamformer look point [x y z] beamformer focuses on
%   prec = postion of receivers [x1 y1 z1; ...xN yN zN]
%   W = weighting of each channel. Must be length N
%   angle_sweep = direction points (degrees) to be tested and plotted
for the directcional
%       response
%
%   P_out = output power array for every x_scan point
%   P_sig = signal power at output of beamformer
%   P_out_n = noise pwer at the output of the beamformer
%   P_in_n = input noise power at the receivers
%   src_power = orignal source power to be compared with the output of
the
%       beamformer

%Calculate number of sensors in array
N = size(prec,1);

%Calculates distance of look point to origin
r = sqrt(look_dir(1).^2 + look_dir(2).^2);

%Converts sweep angles to rectangular coordinates for simulator
xy = rect_polar(r, angle_sweep);

%Calculate noise correlation matrix based on time delays and receiver
%positions and frequency
[ R_vv ] = plane_noise_R( prec, look_dir, var_n, f );


for i = 1:length(angle_sweep)
    %simulate source from each angle in angle sweep
    [ x, ~, src_pwr ] = sim_3D( src, Fs, [xy(i,1) xy(i,2) look_dir(3)],
prec);

    %Beamform source signal for designed beamformer look direction (not
    %actual source location)
    [ ya, z_sig, P_sig(i) ] = DS_beamformer( x, Fs, look_dir, W, prec
);

    %generate random noise corresponding to R_vv correlation matrix
    L = size(ya,2);
    va = sqrtm(R_vv)*randn(N,L);

    %Weight and sum noise with same weights used to beamform the signal
    %NOTE: noise does not need to be delayed due to the correlation
matrix
    %taking that into account
    z_n = (W'*va)./N;
```

```
    %Superposition source and noise signals after beamforming
    z = z_sig + z_n;

    % Calculate input noise power
    P_in_n(i) = mean(var(va'));
    %Caluclate output noise power after weighting
    P_out_n(i) = var(z_n);
    %Calculate total output power of noise and signal combined
    P_out(i) = var(z);
end

figure()
plot(angle_sweep, 10.*log10(P_out./src_pwr),'b',
angle_sweep,10.*log10(P_sig./src_pwr),'r',
angle_sweep,10.*log10(P_out_n./P_in_n),'g' )
title('Simulated Directional Response')
xlabel('direction (degrees)')
ylabel('Gain (dB)')
end
```

sim_steered_resp.m

```
function [ P_sig, P_out_n, P_in_n, P_out, src_pwr] = sim_steered_resp(
src, Fs, psrc, angle_sweep, prec, var_n, f, max_snr_weights )
%Sweeps beamformer look direction at each angle of angle sweep and
%calculates the signal, noise and total power at the output of the
%beamformer.
%   src = source signal of any length
%   Fs = sampling rate of source signal (samples/sec)
%   look_dir = beamformer look point [x y z] beamformer focuses on
%   prec = postion of receivers [x1 y1 z1; ...xN yN zN]
%   angle_sweep = direction points (degrees) to be tested and plotted
for the directional
%       response
%   max_snr_weights-> if = to'1' then max_snr weights will be
calculated
%   and applied. If not = '1', regular DS weights will be used (1/N)
%
%   P_out = output power array for every angle_sweep point
%   P_sig = signal power at output of beamformer
%   P_out_n = noise power at the output of the beamformer
%   P_in_n = input noise power at the receivers
%   src_power = orignal source power to be compared with the output of
the
%       beamformer

%Calculate number of sensors in array
N = size(prec,1);

%Calculates distance of look point to origin
r = sqrt(psrc(1).^2 + psrc(2).^2);

%Converts sweep angles to rectangular coordinates for simulator
```

```
xy = rect_polar(r, angle_sweep);

%simulate source at actual source location
[ x, ~, src_pwr ] = sim_3D( src, Fs, psrc, prec);

for i = 1:length(angle_sweep)
    %Calculate noise correlation matrix based on time delays (look
angle) and receiver
    %positions and frequency
    [ R_vv ] = plane_noise_R( prec, [xy(i,1) xy(i,2) psrc(3)], var_n, f
);
    R_vv_sum(i) = sum(sum(R_vv));

    %Beamform source signal at every look direction in angle sweep
    [ ya, ~, ~ ] = DS_beamformer( x, Fs, [xy(i,1) xy(i,2) psrc(3)],
ones(N,1), prec );
    L = size(ya,2);

    %generate random noise corresponding to R_vv correlation matrix
    va = sqrtm(R_vv)*randn(N,L);

    %Calculate maximum SNR algorithm coeffiecients
    if max_snr_weights == 1 && var_n ~= 0
    [h_max, ~] = eigs(src_pwr.*inv(R_vv)*ones(N),1);
    W = N.*h_max./sum(h_max);

    else W = ones(N,1); % Use delays sum weights
    end

    %Weight and sum signal and noise with same weights
    %NOTE: noise does not need to be delayed due to the correlation
matrix
    %taking that into account
    z_sig = (W'*ya)./N;
    z_n = (W'*va)./N;

    %Superposition source and noise signals after beamforming
    z = z_sig + z_n;

    %Calculate output signal power
    P_sig(i) = var(z_sig);
    % Calculate input noise power
    P_in_n(i) = mean(var(va'));
    %Caluclate output noise power after weighting
    P_out_n(i) = var(z_n);
    %Calculate total output power of noise and signal combined
    P_out(i) = var(z);
end

figure()
plot(angle_sweep, 10.*log10(P_out./src_pwr),'b',
angle_sweep,10.*log10(P_sig./src_pwr),'r',
angle_sweep,10.*log10(P_out_n./P_in_n),'g' )
title('Simulated Steered Response')
xlabel('Beamformer Look Direction (degrees)')
```

```
ylabel('Gain (dB)')
end
```

sine_gen.m

```
function [ sine ] = sine_gen( amp, freq, Fs, length_t )
%[ sine ] = sine_gen( amp, freq, Fs, length_t )
%   Detailed explanation goes here

t = 0:1/Fs:length_t;
sine = amp.*sin(2.* pi.*freq.*t);

end
```

Single_Mic_Wavwrite.m

```
%%% Single Mic wav file extractor

Gain = 1;

% set path where all preprocessed test cases are located
datadir = fullfile('Outdoor_Perimeter_11172011');

% Load file containing the list of the names of all of the case files
load(fullfile(datadir,'DataList.mat'))

for i = 1:length(datalist)
    current_case = datalist(i).name; %extract the name of the current
case
    display(['Processing ' current_case])

    load(fullfile(datadir,current_case)) %load the data from the
current case

    src = Gain.*TimeDataPP(:,1); %amplify signal
    clear TimeDataPP
    clear time

    wavwrite((1./max(abs(src))).*src, fullfile('BF out wavs',
current_case(1:end-7)))
    clear current_case src FS
end
```

sinint.m

```
function y = sinint(x)
% function y = sinint(x)
% This function evaluates the sine integral function using the identity
```

```
%  Si(x) = (1/2*i) (expint(i*x)-expint(-i*x)) + pi/2
% For x<0.001, use polynomial approximation Si(x)~x - x^3/18

    indr = find(abs(x)>=.001);
    inds = find(abs(x)<.001);
    y = zeros(size(x));
    c = sqrt(1/18);
    if ~isempty(indr)
        y(indr) = pi/2+ (expint(1i*x(indr))-expint(-
1i*x(indr)))/(2*1i);
    end
    if ~isempty(inds)
        y(inds) = x(inds).*(1-c*x(inds)).*(1+c*x(inds));
    end

end
```

Spacial_noise_R.m

```
function [ noise_corr, R_corr ] = Spacial_noise_R( n_samples, prec,
n_var,  L)
%[ noise_corr, R_corr ] = Spacial_noise_R( n_samples, prec, n_var,  L)
%   calculates N channels of spatially correlated noise according to:
%   R_corr(i,j) = exp(-mag(prec(i)-prec(j))^2/L^2)
%
%   n_samples = # of columns of spatially correlated noise
%   prec =receiver positions N x3 [x y z]
%   n_var = nosie variance
%   L = gaussian decay factor

N = size(prec,1);

for i = 1:N
    for j = 1:N
        d_rec = sqrt(sum((prec(i,:)-prec(j,:)).^2));
        R_corr(i,j) = exp(-(d_rec.^2)./(L.^2));
    end
end

noise_corr = sqrt(n_var).*sqrtm(R_corr)*randn(N, n_samples);

end
```

Spatio_Temporal_Filter.m

```
function [ z_ST, W_o,H_ST, R_vv, SSNR_in ] = Spatio_Temporal_Filter( y,
v_only,L,overlap )
%Applies a multichannel Spatio-Temporal Prediction algorithm to a
%multichannel signal given signal+noise samples and noise only samples.
%   y = signal +noise array (N channels x ..)
%   v_only = noise only array (N channels x...)
%   L = frame size of filter;
%   overlap = number of sample overlap between frames
```

```
%Verify that the sample overlap does not exceed block length
if(overlap>L)
    error('Sample block "L" must be larger than sample overlap')
end
N = size(y,1); % Calculate number of mics in array

%Initialize
R_vv = zeros(N*L,N*L); % Initialize noise correlation matrix NLxNL
R_yy = zeros(N*L,N*L); % Initialize signal + noise correlation matrix
m = 0;%intialize noise correlation matrix calculation counter for
averaging later
n = 0;% intialize signal+noise correlation matrix calculation counter
for averaging later
P_in_n_SEG = 0; % initialize average noise power of each frame
P_in_SEG = 0;


%Calculate statistics for each block
for i = 1:L-overlap:max(length(v_only),length(y)) %initialize index to
start at beginning of each block of length L taking sample overlap into
account

        % Calculate noise statistics
    if(L+i-1 <= length(v_only)) %if current block will exceed length of
input array, break for loop

        v = v_only(:,i:L+i-1)'; % Take a block of L samples at starting
at current index i

        %Organize NxL matrix containing sample blocks into NLx1 matrix

        v_L = v(:);

        %Calculate current correlation matrix for N blocks of L samples
        m = m+1; % number of times correlation matrix is calculated

        %Average correlation matrix over time
        R_vv = ((m-1)/m)*R_vv + (v_L * v_L')./m;
        %R_vv = lambda*R_vv + (v_L * v_L').*(1-lambda);
        P_in_n_SEG = ((m-1)/m)*P_in_n_SEG + mean(var(v))./m;
    end

    % Calculate source + noise statistics
    if(L+i-1 <= length(y))

        y_L = y(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

        %Organize NxL matrix containing sample blocks into NLx1 matrix

        y_k = y_L(:);

        n = n+1;
```

```
        R_yy = ((n-1)/n)*R_yy + (y_k*y_k')./n;
        %R_yy = lambda*R_yy + (y_k*y_k').*(1-lambda);
        P_in_SEG = ((n-1)/n)*P_in_SEG + mean(var(y_L))./n;
    end
end

SSNR_in = (P_in_SEG-P_in_n_SEG)./P_in_n_SEG;



%Calculate the optimal spatio-temporal prediction matrix W_o

W_o = ((R_yy(:,1:L)-R_vv(:,1:L))/(R_yy(1:L,1:L)-R_vv(1:L,1:L)))';

H_ST = inv(W_o*inv(R_vv)*W_o')*W_o*inv(R_vv);

for i = 1:L-overlap:length(y)
    if(L+i-1 > length(y)) %if current block will exceed length of input
array data, break for loop
        break
    end
    y_L = y(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

    %Organize NxL matrix containing sample blocks into NLx1 matrix

    y_k = y_L(:);

    %perform filtering operation using matrix multiply
    z_ST(i:i+L-1,1) = H_ST*y_k;

end

end
```

ST_Test1_filt_length_0_overlap.m

```
%% ST Filter Test 3- Varying filter lengths with 0 overlap

L = [2     3     4     5     6     7     8    10    16    20    24
28    30    32];
overlap = L -L;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_ST_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
    L_current = L(i)
    [ z_ST, ~,H_ST, ~ ] = Spatio_Temporal_Filter( src', noise', L(i),
overlap(i)  );
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(i),overlap(i), H_ST, z_ST );
```

```
    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_ST_all(1:length(z_ST),i) = z_ST;
end
```

ST_Test2_overlap.m

```
%% ST Filter Test 2- Varying sample overlaps with L =  10 filter
tic;
L(1:10) = 10;
overlap = 0:9;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_ST_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
    current_overlap = overlap(i)
    [ z_ST, ~,H_ST, ~ ] = Spatio_Temporal_Filter( src', noise', L(i),
overlap(i)  );
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(i),overlap(i), H_ST, z_ST );

    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_ST_all(1:length(z_ST),i) = z_ST;
end
toc;
```

ST_Test3_filt_length_max_overlap.m

```
%% ST Filter Test 3- Varying filter lengths with max overlap
tic
L = [2     3     4     5     6     7     8    10    16    20    24
28    30    32];
%L = [40 70 100 ]; % extended version
overlap = L-1;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_ST_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
```

```
    L_current = L(i)
    [ z_ST, ~,H_ST, ~, SSNR_in] = Spatio_Temporal_Filter( src', noise',
L(i), overlap(i)  );
    [ z_v_filt, SNR_in, SNR_out, SSNR_out, nr_factor, sd_factor, P_out,
P_out_n, P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(i),overlap(i), H_ST, z_ST );

    SSNR_in_all(i) = SSNR_in;
    SSNR_out_all(i) = SSNR_out_all;
    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_ST_all(1:length(z_ST),i) = z_ST;
end
toc
```

STP_Test4_N.m

```
%% STP Filter Test 4- Varying Mic #s
tic
N = 1:9;
L = [2 3 4 5 6 7 8 10 16 20 24 28 32] ;
overlap = L-1;
z_v_filt_all = zeros(max(length(noise),length(src)),length(N));
z_ST_all = zeros(max(length(noise),length(src)),length(N));
for i = 1:length(N)
    current_receiver_num = N(i)

    for j= 1:length(L)
    [ z_ST, ~,H_ST, ~ ] = Spatio_Temporal_Filter( src(:,1:N(i))',
noise(:,1:N(i))', L(j), overlap(j)  );
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src(:,1:N(i))',
noise(:,1:N(i))', L(j),overlap(j), H_ST, z_ST );

    SNR_in_all(i,j) = SNR_in;
    SNR_out_all(i,j) = SNR_out;
    nr_factor_all(i,j) = nr_factor;
    sd_factor_all(i,j) = sd_factor;
    P_out_all(i,j) = P_out;
    P_out_n_all(i,j) = P_out_n;
    P_sig1_all(i,j) = P_sig1;
    P_out_sig1_all(i,j) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_ST_all(1:length(z_ST),i) = z_ST;
    end
end
toc
```

Test3_case_sweep.m

```
%%% Algorithm Test 3 Case script
% Opens and executes each case from the 11/17/2011 outdoor data collect
% using the Spatio Temporal Predictions and Wiener Filters.
%
%Set Up:
% 1. Preprocess all cases using "Array_Preprocess" function
% 2. Put all preprocessed cases in their own folder "datadir"(change
code to match)
% 3. Put all noise only files in a separate subfolder "datadir_noise"
(change code to match)
% 4. Create "DataList.mat" file using datalist = dir (make sure only
case
%    files are in the current folder)
% 5. Save "DataList.mat" in the folder with all of the cases to be
tested
% 6. Make sure all noise file are in this format: 10ft_noise.mat
% 7. Make sure all case files are in this format: 10ft_B.mat
% 8. Verify location wher output wavs will be saved

L = [ 2 6 10 20 30 40 70]; % set filter lengths to be used in sweeps
overlap = L-1;
Gain = 1000; % Set signal gain before filtering

% set path where all preprocessed test cases are located
datadir = fullfile('Array_data','Outdoor_Perimeter_11172011'); %%% <---
-- 2

% set path where all noise only recording are found
datadir_noise =
fullfile('Array_data','Outdoor_Perimeter_11172011','Noise'); %%% <-----
3

% Load file containing the list of the names of all of the case files
load(fullfile(datadir,'DataList.mat'))


for i = 1:length(datalist)

    current_case = datalist(i).name; %extract the name of the current
case
    display(['Processing ' current_case])

    load(fullfile(datadir,current_case)) %load the data from the
current case

    src = Gain.*TimeDataPP; %amplify signal
    clear TimeDataPP
    clear time

    noise_header = current_case(1:end-8);
```

```
    load(fullfile(datadir_noise, [noise_header 'noise_PP.mat'])) %load
noise that corresponds to the current case

    noise = Gain.*TimeDataPP; %amplify noise signal
    clear TimeDataPP
    clear time

    %Run STP Sweep
    z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
    z_ST_all = zeros(max(length(noise),length(src)),length(L));
    for j = 1:length(L)
        L_current = L(j)
        [ z_ST, ~,H_ST, ~, SSNR_in] = Spatio_Temporal_Filter( src',
noise', L(j), overlap(j)  );
        [ z_v_filt, SNR_in, SNR_out, SSNR_out, nr_factor, sd_factor, ~,
~, ~, ~ ] = BB_Filter_Metrics( src', noise', L(j),overlap(j), H_ST,
z_ST );

        SSNR_in_all(j) = SSNR_in;
        SSNR_out_all(j) = SSNR_out;
        SNR_in_all(j) = SNR_in;
        SNR_out_all(j) = SNR_out;
        nr_factor_all(j) = nr_factor;
        sd_factor_all(j) = sd_factor;
        z_v_filt_all(1:length(z_v_filt),j) = z_v_filt;
        z_ST_all(1:length(z_ST),j) = z_ST;
    end

    %save output metrics to appropriate Test 3 folder
    save(fullfile('Spatio-Temporal
Prediction','ST_Test3_L_max_overlap', ['ST_Test3_vars_'
current_case(1:end-7)]
),'SSNR_in_all','SSNR_out_all','SNR_in_all','SNR_out_all','nr_factor_al
l','sd_factor_all','z_v_filt_all','z_ST_all','FS','src','noise','Gain',
'current_case','L','overlap');
    clear SSNR_in_all SSNR_out_all SNR_in_all SNR_out_all nr_factor_all
sd_factor_all z_v_filt_all z_ST z_v_filt SSNR_in SSNR_out H_ST SNR_in
SNR_out nr_factor sd_factor j


    % Run Wiener Filter Sweep
    z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
    z_W_all = zeros(max(length(noise),length(src)),length(L));
    for j = 1:length(L)
        L_current = L(j)
        [z_W, ~, ~, H_W, SSNR_in] = Weiner_filter( src', noise', L(j),
overlap(j)  );
        [ z_v_filt, SNR_in, SNR_out, SSNR_out, nr_factor, sd_factor,
P_out, P_out_n, P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(j),overlap(j), H_W, z_W );


        SSNR_in_all(j) = SSNR_in;
        SSNR_out_all(j) = SSNR_out;
        SNR_in_all(j) = SNR_in;
```

```
        SNR_out_all(j) = SNR_out;
        nr_factor_all(j) = nr_factor;
        sd_factor_all(j) = sd_factor;
        z_v_filt_all(1:length(z_v_filt),j) = z_v_filt;
        z_W_all(1:length(z_W),j) = z_W;
    end

    %save output metrics to appropriate Test 3 folder
    save(fullfile('Weiner Filter','Weiner_Test3', ['Weiner_Test3_vars_'
current_case(1:end-7)]
),'SSNR_in_all','SSNR_out_all','SNR_in_all','SNR_out_all','nr_factor_al
l','sd_factor_all','z_v_filt_all','z_W_all','FS','src','noise','Gain','
current_case','L','overlap');
    clear SSNR_in_all SSNR_out_all SNR_in_all SNR_out_all nr_factor_all
sd_factor_all z_v_filt_all z_W z_v_filt SSNR_in SSNR_out H_W SNR_in
SNR_out nr_factor sd_factor j

    %write all filtered output signals to wav files in the appropriate
folder
    for j = 1:length(L)
        wavwrite((1./(max(abs(z_ST_all(:,j))))).*z_ST_all(:,j),FS,
fullfile('BF out wavs', [current_case(1:end-7) '_STBF_L'
num2str(L(j))]))
        wavwrite((1./(max(abs(z_W_all(:,j))))).*z_W_all(:,j),FS,
fullfile('BF out wavs', [current_case(1:end-7) '_WBF_L'
num2str(L(j))]))
    end
    clear current_case
end
```

Weiner_filter.m

```
function [z_W, R_vv, R_yy, H_W, SSNR_in] = Weiner_filter( y, v_only, L,
overlap  )
%Applies a multi channel weiner filter noise reduction algorithm to a
multi
%channel input given a noise only segment and a signal + noise segment
%   y = NxP matrix to be beamformed. Rows correspond to each mic
channel and columns
%       correspond to sample numbers. N = # of mics, P = length of data
%       NOTE: y is not time shifted or aligned. Raw data from mics
%   v_only = NxQ matrix of each channel of noise recording with no
speech
%       present. Does not need to be same length as y_k. Used to
calculate
%       Rvv
%   Fs = sampling rate of data
%   L = filter length to be used in algorithm
%   overlap = sample overlap for each length L frame


if(overlap>L)
    error('Sample block "L" must be larger than sample overlap')
end
N = size(y,1); % Calculate number of mics in array
```

```
% Calculate "U" matrix for future H_w calculations
U = zeros(L,N*L);
U(1:L,1:L) = eye(L,L); % U = [I(LxL) 0(LXL)... 0(LXL)] (LXNL matrix)


%%% Initialize Variables
R_vv = zeros(N*L,N*L); % Intialize noise correlation matrix NLxNL
m = 0; % intialize noise correlation matrix calculation counter for
averaging later
P_in_n_SEG =0;
R_yy = zeros(N*L,N*L);
n = 0;
P_in_SEG =0;


%%% Estimate statistics by breaking inputs into frames of samples
for i = 1:L-overlap:max(length(v_only),length(y)) %initialize index to
start at beginning of each block of length L taking sample overlap into
account

    %%%% Calculate R_vv from v_only %%%%
    if(L+i-1 <= length(v_only))
    v = v_only(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

    %Organize NxL matrix containing sample blocks into NLx1 matrix

    v_L = v(:);


    %Calculate current correlation matrix for N blocks of L samples
    m = m+1; % number of times correlation matrix is calculated

    %Average correlation matrix over time
    R_vv = ((m-1)/m)*R_vv + (v_L * v_L')./m;
    %R_vv = lambda*R_vv + (v_L * v_L').*(1-lambda);
    P_in_n_SEG = ((m-1)/m)*P_in_n_SEG + mean(var(v))./m;
    end

    %%%% Calculate R_yy for each L sample block of (signal + noise)
data %%%%
    if(L+i-1 <= length(y))
    y_L = y(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

    %Organize NxL matrix containing sample blocks into NLx1 matrix
    y_k = y_L(:);

    n = n+1;
    R_yy = ((n-1)/n)*R_yy + (y_k*y_k')./n;
    %R_yy = lambda*R_yy + (y_k*y_k').*(1-lambda);
    P_in_SEG = ((n-1)/n)*P_in_SEG + mean(var(y_L))./n;
    end
```

```
end

SSNR_in = (P_in_SEG-P_in_n_SEG)./P_in_n_SEG;


%%%% Calculate Filter matrix H_w and execute filtering operation %%%%
H_W = ((eye(N*L,N*L) - (R_yy\R_vv))*U')';


for i = 1:L-overlap:length(y)
    if(L+i-1 > length(y)) %if current block will exceed length of input
array data, break for loop
        break
    end
    y_L = y(:,i:L+i-1)'; % Take a block of L samples at starting at
current index i

    %Organize NxL matrix containing sample blocks into NLx1 matrix

    y_k = y_L(:);

    z_W(i:i+L-1,1) = H_W*y_k;

end

end
```

Weiner_Simulation_test1.m

```
%%% Weiner Simulation test 1 %%%
% varying noise cutoff frequencies with spacial correlation
% with block lengths also varied
prec = [ .956   0   .445;
         0      0   .458;
         -.915  0   .445;
         .956   0   1.368;
         0      0   1.358;
         -.915  0   1.278;
         .956   0   2.139;
         0      0   2.133;
         -.915  0   2.075];

psrc = [0 40 0];
[ src, ~, src_pwr ] = sim_3D( bftest0, FS, psrc, prec );

L = [2     3     4     5     6     7     8    10    16    20    24
28    30    32];
overlap = L - 1;
fc = [ 50 100 200 300 500 800 1000 2000 3000 ];
for h = 1:length(fc)
    current_fc = fc(h)
```

```
    [ noise_corr, R_vv ] = plane_noise_R_3D( prec, src_pwr,
length(src), fc(h),7,FS );
    y = src+noise_corr;

    for i = 1:length(L)
    L_current = L(i)
    [z_W, ~, ~, H_W] = Weiner_filter( y, noise_corr, L(i), overlap(i)
);
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( y, noise_corr,
L(i),overlap(i), H_W, z_W );

    SNR_in_all(h,i) = SNR_in;
    SNR_out_all(h,i) = SNR_out;
    nr_factor_all(h,i) = nr_factor;
    sd_factor_all(h,i) = sd_factor;
    P_out_all(h,i) = P_out;
    P_out_n_all(h,i) = P_out_n;
    P_sig1_all(h,i) = P_sig1;
    P_out_sig1_all(h,i) = P_out_sig1;

    end
end
```

Weiner_Test1_Filt_length.m

```
%% Weiner Filter Test 1- Varying filter lengths with 0 overlap

L = [2     3     4     5     6     7     8    10    16    20    24
28    30    32];
overlap = 0;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_W_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
    L_current = L(i)
    [z_W, ~, ~, H_W] = Weiner_filter( src, noise', L(i), overlap(i)  );
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src, noise', L(i),overlap(i),
H_W, z_W );

    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_W_all(1:length(z_W),i) = z_W;
end
```

Weiner_Test2_overlap.m

```
%% Weiner Filter Test 2- Varying sample overlaps with L =  10 filter

L(1:10) = 10;
overlap = 0:9;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_W_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
    current_overlap = overlap(i)
    [z_W, ~, ~, H_W] = Weiner_filter( src', noise', L(i), overlap(i)
);
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(i),overlap(i), H_W, z_W );

    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_W_all(1:length(z_W),i) = z_W;
end
```

Weiner_Test3_filt_length_max_overlap.m

```
%% Weiner Filter Test 3- Varying filter lengths with max overlap
tic

L = [2     3     4     5     6     7     8    10   16  20 24 28 30 32
];
overlap = L-1;
z_v_filt_all = zeros(max(length(noise),length(src)),length(L));
z_W_all = zeros(max(length(noise),length(src)),length(L));
for i = 1:length(L)
    L_current = L(i)
    [z_W, ~, ~, H_W, SSNR_in] = Weiner_filter( src', noise', L(i),
overlap(i)  );
    [ z_v_filt, SNR_in, SNR_out, SSNR_out, nr_factor, sd_factor, P_out,
P_out_n, P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src', noise',
L(i),overlap(i), H_W, z_W );


    SSNR_in_all(i) = SSNR_in;
    SSNR_out_all(i) = SSNR_out;
    SNR_in_all(i) = SNR_in;
    SNR_out_all(i) = SNR_out;
    nr_factor_all(i) = nr_factor;
    sd_factor_all(i) = sd_factor;
    P_out_all(i) = P_out;
    P_out_n_all(i) = P_out_n;
    P_sig1_all(i) = P_sig1;
```

```
    P_out_sig1_all(i) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_W_all(1:length(z_W),i) = z_W;
end
toc
```

Weiner_Test4_N.m

```
%% Weiner Filter Test 4- Varying Mic #s

N = 1:9;
L = [2 3 4 5 6 7 8 10 16 20 24 28 32] ;
overlap = L-1;
z_v_filt_all = zeros(max(length(noise),length(src)),length(N));
z_W_all = zeros(max(length(noise),length(src)),length(N));
for i = 1:length(N)
    current_receiver_num = N(i)

    for j= 1:length(L)

    [z_W, ~, ~, H_W] = Weiner_filter( src(:,1:N(i))', noise(:,1:N(i))',
L(j), overlap(j)  );
    [ z_v_filt, SNR_in, SNR_out, nr_factor, sd_factor, P_out, P_out_n,
P_sig1, P_out_sig1 ] = BB_Filter_Metrics( src(:,1:N(i))',
noise(:,1:N(i))', L(j),overlap(j), H_W, z_W );

    SNR_in_all(i,j) = SNR_in;
    SNR_out_all(i,j) = SNR_out;
    nr_factor_all(i,j) = nr_factor;
    sd_factor_all(i,j) = sd_factor;
    P_out_all(i,j) = P_out;
    P_out_n_all(i,j) = P_out_n;
    P_sig1_all(i,j) = P_sig1;
    P_out_sig1_all(i,j) = P_out_sig1;
    z_v_filt_all(1:length(z_v_filt),i) = z_v_filt;
    z_W_all(1:length(z_W),i) = z_W;
    end
end
```

x_beam_plot.m

```
function [ beam_pwr ] = x_beam_plot( x, Fs, prec, x_scan, y_plane,
z_plane, W)
%[ beam_pwr ] = x_beam_plot( x, Fs, prec, x_scan, y_plane, z_plane, W)
%   x = raw (un-aligned) data from mic array (NxL)
%   Fs = sampling rate of source signal
%   prec = postion of receivers [x1 y1 z1;x2 y2 z2...]
%   x_scan = array of points used to scan the x axis
%   y_scan = defines the y plane for the x direction sweep
%   z_scan = defines the z plane for the x direction sweep
%   W = weighting vector for each mic channel

%   beam_pwr = vector of calculated signal power at each x_scan point
```

```
%   src_pwr = calculated source power over all inoput samples



 %% Find Max power point
 for i = 1:length(x_scan)                              %look direction
     [ ~, ~, pout_scan ] = DS_beamformer( x, Fs, [x_scan(i) y_plane
z_plane], W, prec );
     beam_pwr(i) = pout_scan;
 end

 %% Plot Beam Power vs X axis
 figure ()

 subplot(2,1,1)
 plot(x_scan, beam_pwr);
 xlabel('x position (m)')
 ylabel('beam power (W)')

 subplot(2,1,2)
 plot(x_scan, 10.*log10(beam_pwr))
 xlabel('x position (m)')
 ylabel('beam power (dB)')


end
```