Optimizing Acoustic Array Beamforming to Aid a Speech Recognition System

A Thesis

Presented in Partial Fulfillment of the Requirements for Bachelor of Science Degree with Honors Research Distinction in Electrical and Computer Engineering

By Sergei Preobrazhensky

Electrical and Computer Engineering
The Ohio State University

May 2012

Examination Committee:

Prof. Lee Potter, Adviser
Dr. Josh Ash, Research Scientist

Abstract

The iBrutus is a pilot project at the Computer Science and Engineering (CSE) department at OSU which develops human-computer interaction via spoken dialog. The goal of the iBrutus project is to design a kiosk with a talking avatar on a screen which will answer questions at a public event like a football game at a potentially noisy environment like the Ohio Stadium.

In such an environment, the speech recognition software employed by the system would be ineffective without prior processing to obtain a cleaner speech signal. As a rule of thumb, if the iBrutus could correctly interpret 70% or more words, it could successfully to map the input to a known question/command. To improve the speech recognition rate the author has chosen to research a beamforming algorithm. Such an algorithm combines inputs of from a microphone array to minimize the interference while preserving the desired signal (i.e. speech arriving from a known direction/location).

The goal of the research has been to develop such an algorithm and a means of testing to determine which parameters associated with the algorithm – such as the spatial geometry of the microphone array – will produce the desired speech recognition rate in minimum processing time. The beamforming algorithm designed by the author in MATLAB was frequency based wideband Minimum Variance Distortionless Response (MVDR). Tests showed that at least 70% word recognition rate could be achieved under certain parameter choices. The processing time of the MATLAB-based algorithm is currently larger than desired for use with iBrutus, but there is potential for improvement.

Acknowledgements

I thank my adviser, Prof. Lee Potter for his mentoring and support from June 2011 to May 2012. He was a helpful guide during the year. I would not have made the commitment to conduct a year-long honors research project if not for him introducing to me the research opportunities he had for undergraduates.

I thank Dr. Josh Ash for being on the hearing committee alongside Prof. Potter during my Honors Thesis oral defense.

I thank Prof. Clymer of the Electrical and Computer Engineering Department at OSU for guiding me through the official steps of completing Honors Distinction Research in Electrical and Computer Engineering.

Finally, I thank my undergraduate colleagues in research from January to March 2012 for their collaboration. They are Domenic Belgiovane, David Leonard, Matt Miller, Nick Blanton, and Jonathan Lane.

The ECE Department at OSU has been generous to provide lab space and equipment for this research.

Table of Contents

List of Equations

List of Figures

1. Introduction

1.1 iBrutus Project

The iShoe project was originally developed at Purdue University under the name of “e-Stadium,” where it was then transferred to Ohio State under license. It has since been developed as a Capstone Computer Science and Engineering project. Through iShoe, Ohio State fans can enjoy real-time statistics on the game, biographies of the players and coaches. This is all accessible from any computer or other web enabled device [1].

The short-term goal of the iBrutus project is to provide an alternative front end to the iShoe and answer event-related questions during football games. The iBrutus will be set up as a kiosk at the Ohio Stadium (within the inner hallway under the stadium seats). The kiosk will feature an avatar of The Ohio State University’s mascot Brutus on a screen. This avatar will interact with human users via spoken dialog, rather than a touch screen or buttons [2].

In the long term, a system like the iBrutus could be adapted to other public venues and events. More generally, a future prospect for interaction between humans and computers is spoken dialogue and visual cues rather than a keyboard, mouse, or touch screen. The iBrutus is thus, in addition to its specific purpose, a pilot project for advancing the field of human-computer communication [2].

The iBrutus will be a complex system with multiple components including speech recognition software, Microsoft Xbox Kinect cameras, and an array of microphones [1]. The mentioned components will have a direct influence on the author's research as explained in the next section.

1.2 Problem Statement

The CSE iBrutus research team desires a means to obtain a speech signal clean enough for the employed speech recognition software to have a word recognition rate of at least 70%. English speech is redundant enough that at this rate the iBrutus could successfully map the recognized words to a sentence to which the system can respond [2].

Without additional processing, the speech recognition software will be ineffective in acoustically noisy areas such as Ohio Stadium. The author and several other ECE undergraduates have worked with the CSE department research team working on iBrutus to design a beamforming algorithm that uses a microphone array that is capable of minimizing interference.

Using the Microsoft Xbox Kinect, iBrutus will have the capability of isolating faces of individual people in a crowd in order to determine and focus on the direction from which a particular speaker's voice is coming [2]. Knowing this direction, beamforming can be used take advantage of the phase differences that the desired human speaker's signal exhibits across the microphones to "listen" in a particular direction. The algorithm will attempt to minimize interference coming from other directions. The Kinect has built-

in beamforming capability, using its four microphone array. However, the CSE iBrutus team stated that this array has not sufficiently yielded speech recognition in acoustically noisy environments [2]. Also, instantaneous beamforming done by the Xbox Kinnect will overwrite the original signal, and in case the iBrutus chooses the wrong beamforming direction for the human speaker it will not be able to reprocess the original data.

2. Background

2.1 Objectives

2.1.1 Main Goal

The author has made it his goal to develop and test a beamforming algorithm for iBrutus which can be extended to an arbitrary number of microphones. The author believes his algorithm would be more likely to exceed the 70% word recognition limit than using the beamforming capabilities of the Xbox Kinect. This is because the author's algorithm allows more microphones, and a more effective array geometry. Using more microphones is generally beneficial for beamforming as it allows more degrees of freedom for the weighted summing *(*explained in the "Solution Design" section). Certain array geometries will allow for more interference suppression than others (see "Experimental Procedure" section). Furthermore, the algorithm to be designed would be able to reprocess the original audio data, if needed, unlike the Xbox Kinect. These two benefits outweigh the fact that implementing such an algorithm will inherently cause a processing lag (the lag could still be acceptably small). The beamforming algorithm that author has developed in MATLAB is known as minimum variance distortionless response (MVDR). More specifically, the author has chosen a wideband and frequency-based implementation of MVDR (explained in the "Solution Design" section).

2.1.2 Parameters

The author's algorithm has several input parameters the values of which can be chosen as needed. The parameters that have been varied in the tests presented in this paper (explained in the "Solution Design" and "Experimental Procedure" section) are:

1. spatial placement of the microphones,
2. extent of the frequency spectrum to process (also referred to here as bandwidth),
3. the width of uniform frequency sub-bands into which the frequency spectrum of the input signal spectrum would be split for wideband processing,
4. forgetting factor associated with the adaptive nature of MVDR,
5. the ratio of the RMS of the desired component of the test audio signal to the RMS of the undesired component of the same signal - simulated interference (this ratio is referred to as the signal to interference ratio or SIR).

2.1.3 Metrics

The effectiveness of the beamforming will ultimately be measured by the speech recognition software to be used with iBrutus. Currently iBrutus is designed to use Windows Speech Recognition (WSR) [2]. However, the author has found WSR to be ineffective for achieving a 70% word recognition rate, given a speech signal processed with his algorithm. Instead the author has tested Google Speech Recognition (GSR) with success. The iBrutus is not currently interfaced with GSR, but the author's

successful tests with GSR provide a proof of concept that his algorithm could be made viable for iBrutus. The word recognition rate that the author has used as a metric was simply the number of words that GSR recognized correctly. This score is not affected by extra words incorrectly inserted in the recognized text as the iBrutus will be designed to find key words to which it can respond in a string of recognized words [2].

The other important metric of the beamforming algorithm is processing time. Generally, there is a trade-off between processing quality and time. The goal with beamforming for iBrutus can be fully restated as: process speech with small and constant delay without compromising 70% word recognition. This could be accomplished, for example, if the beamforming algorithm can process a second of noisy audio in under a second. This will allow for continuous, quasi-real-time processing of consecutive blocks of audio with a small constant lag required to process a single block. From the tests the author has conducted, there is reason to believe that a fast modern computer could, in fact, process a second of signal with his MATLAB-based algorithm in under a second.

The author has also used a less relevant metric, but one that still offers insight on the beamforming performance. This metric was the percentage is RMS error improvement when comparing the amount of noise/interference in the input and output. This will be discussed more detail in the “Non-speech Simulations” section.

2.1.4 Constraints

The primary constraint of the author’s research is that findings are to be presented by the end of the Spring 2012 academic quarter at OSU (which extends from

the end of March to the first week of June). This is the deadline by which the author is to complete this thesis.

Given limited research time, testing was limited to the particular MVDR beamforming algorithm developed. For the same reason only the algorithm parameters previously described were varied, although several others can be varied as well. Additionally, the author has narrowed down the array geometry parameter, to one-dimensional, uniformly spaced microphone arrays. As the CSE iBrutus team desires portability of the design, the length of the microphone array was constrained to 1 meter or less.

The simulations done by the author have thus far been constrained to an array of eight microphones. The existing data acquisition hardware available to the author through the ECE department is the TASCAM US-1641. Although this device has sixteen various channels, it supports a maximum of eight identical microphones (through XLR channels) [20]. Most of the author's work has relied on simulating a microphone array in MATLAB rather than relying on connecting a physical microphone array to the TASCAM hardware. However, the author would like to leave future researchers with the opportunity of recording speech in a real noisy environment with the existing TASCAM hardware. Therefore the testing the author has done was focused on using eight microphones. Results suggest that eight microphones may be an acceptable number. However the author's MATLAB algorithm supports an arbitrary number of microphones, in case future researchers choose, to acquire DAQ hardware that supports more microphones. It is not known however whether the increased processing time

associated with the larger number of microphones would be worth the possible improvement in speech recognition.

2.2 Solution Design

2.2.1 General MVDR Beamforming Algorithm Description

The software design of this project will be comprised of the wideband MVDR beamforming algorithm. The algorithm is adaptive in time. It estimates the average covariance between each microphone signal over a short time window; it then continuously updates this estimate over the time windows that follow. A covariance matrix is thus computed using the block (time window) of signal that is recorded by the microphones. This is shown in Equations 1 and 2 below.

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{bmatrix}$$

Equation 1

$$R = \frac{X X^{T}}{n}$$

Equation 2

Here, the X is an array of the input signals with n microphones and k samples, and R is the covariance matrix. Then the algorithm computes gain and phase delay weights to be applied to each microphone such as to minimize the overall average energy of the output. The output will be a weighted sum of microphone inputs. The weights are chosen such that on the average maximal destructive interference is achieved by the sum for all the interference components. However, to protect a desired

signal coming from a known direction (the human speaker, in the case of iBrutus), the weights are chosen such that there is unity gain in that direction. Optimal weights are calculated using Equation 3.

$$w = [\, w_1 \; w_2 \; \cdots \; w_n \,]^H = \frac{R^{-1} v_s}{v_s{}^H \, R \, v_s}$$

Equation 3

In Equation 3, v_s is the steering vector which indicates the delays the desired signal exhibits across the microphone array. The expression for v_s is given by Equation 4 below. In Equation 4, tau is the time delay from the desired source location to each microphone of the array.

$$v_s = e^{j\,2\,\pi\,f\,[\tau_1 \;\; \tau_2 \cdots \; \tau_n \,]^T}$$

Equation 4

Equation 5 shows how MVDR weights are applied to the data block X to form the output block.

$$x_{out} = w^H X$$

Equation 5

Note that the time delays in Equation 4 are calculated via the spherical wave propagation model from a point-source to the microphones. The steering vector depends not only on the angle of the desired sound source but also on its distance from the microphone array. This MVDR implementation method is known as near-field MVDR as the steering vector will depend more and more on the distance rather than just the angle as the source is placed nearer to the microphone array. The latter method

approximates the steering vector by the plane-wave (rather than spherical) propagation model which is a good approximation for far away sources. For the case of iBrutus, the location of the desired human speaker relative to the microphone array is expected to be close enough that near-field MVDR should show significant improvement in the SIR when compared to the far-field model implementation [3].

2.2.2 Wideband MVDR Implementation

An MVDR steering vector assumes a fixed operating frequency (denoted by *f* in Equation 4). This means that there will be unity gain and no unwanted phase offset for the sinusoidal signal component of that particular frequency. For frequency components of the desired source signal which are close to this operating frequency, MVDR will produce near unity gain and very little phase distortion. Thus MVDR works well for a narrow-band signal. The frequency range for the main bulk of human speech is 300-3400 Hz [5].

A previous iBrutus student research team at OSU determined that MVDR processing over this spectrum will yield much better results when the signal is decomposed into narrow bands of frequency components [6]. This approach is known as wideband MVDR. Separate MVDR steering vectors are chosen for multiple operating frequencies which are uniformly spaced throughout the human speech spectrum. Further modifications that need to be made for this approach are as follows: The covariance matrix paired with each steering vector is chosen to represent only a band of frequency components which are close to the operating frequency. The previous iBrutus research team filtered the microphone data (using band-pass) into multiple equal-sized

frequency sub-bands. Separate MVDR weights were calculated and applied to each sub-band, choosing the operating frequency to be at the center of each band. Then the sub-band output signals are summed to produce the processed signal for the whole frequency spectrum.

Most of the testing done by the author and previous teams working on the iBrutus project has stayed within the bounds of 300-3500 Hz [6,3,4]. However, according to [5], taking a broader spectrum of 150-7000 Hz may be better suited for speech recognition. Consonant sounds, for example, can be distinguished much better using this extended spectrum. The author has tested the MVDR algorithm with both the narrower and the wider spectrum.

2.2.3 Computing and Applying MVDR weights in Frequency Domain

Traditional time-domain MVDR, involves band-pass filtering a time window of microphone data into multiple time domain signals – one for each frequency sub-band. Then each of these signals shifted to baseband to produce a complex signal suitable for MVDR [8]. Following that the filtered signals can be possibly down-sampled to save computation time without loss to accuracy [9]. After MVDR weights have been applied, this process would need to be reversed in order to reconstruct the speech spectrum.

The author has decided to bypass band-pass filtering the noisy speech signal into multiple time domain signals, and instead perform wide-band MVDR directly in the frequency domain. This can be done by taking the FFT (Fast Fourier Transform) over a time window of microphone data; then the covariance matrix is formed from discrete

frequency bin values which represent a sub-band. MVDR weights are calculated from this covariance matrix and applied to these bins. The weights are then applied to the frequency bins of each sub-band. This approach has been outlined in [7], and the author decided to follow this description closely. With the frequency domain MVDR algorithm, the time domain MVDR steps could be bypassed likely saving on computation. Although it is not fully known what trade-offs there may be to each approach, the article [7] contends that frequency domain MVDR demonstrates better SIR results than time-domain MVDR methods, especially when it comes to non-stationary interference sources. The acoustic interference in the case of iBrutus is in fact expected to be largely non-stationary noise (rather than stationary white noise).

In the frequency domain MVDR algorithm, the FFT is taken for 1024 sample Hamming-windowed sections of the n-microphone data. Each subsequent 1024-sample window is advanced in time by 32 samples. After MVDR processing in frequency domain, the IFFT is taken and all but the 32 central samples of the resulting time domain signal are discarded. The remaining 32-sample block is appended to the processed time signal. Figure 1 depicts this process. The article [7] offers that such 32-sample increments yield in almost half the computational time almost as good a performance as the 16-sample increments which were at first used by its authors.

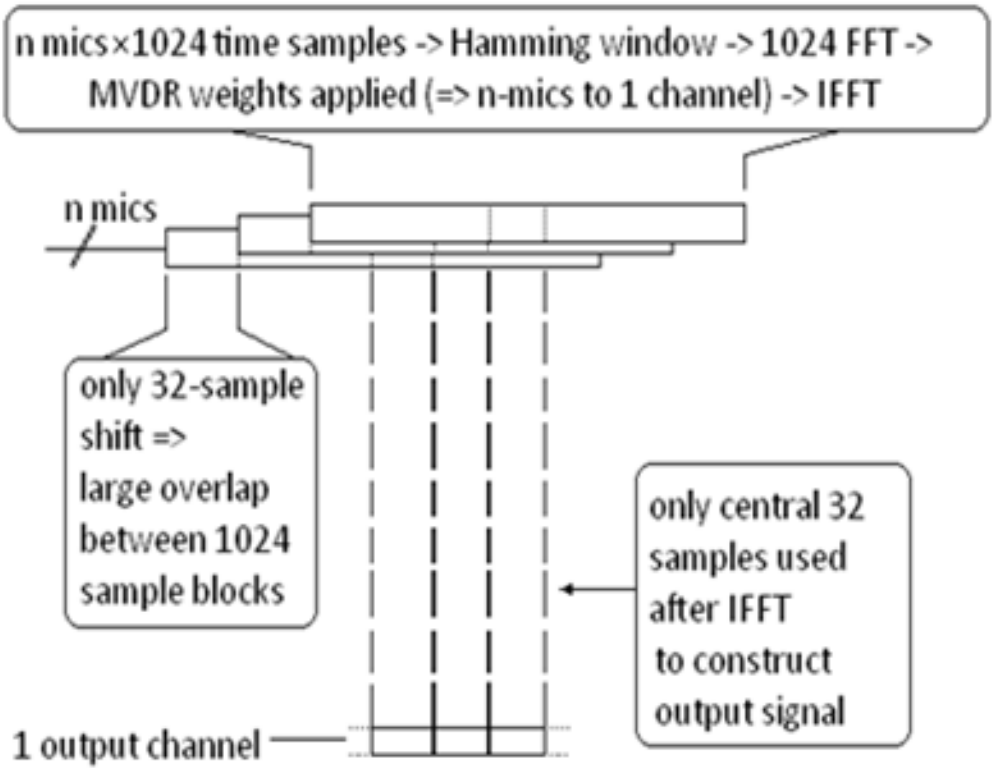


Figure 1. Frequency-based MVDR Beamforming as Used by Author.

For the DFT of each section, wideband MVDR is applied only to bins which represent the spectrum of interest. For example the one-sided bandwidth of 300-3400 Hz roughly corresponds to 144 DFT bins. This number of bins can be split into equal size sub-band numbers such as 18, 24, or 36, (the corresponding sub-band size, s, would be 8, 6, and 4 bins). The covariance matrix R for a particular sub-band is formed by the following equation:

$$R_{b,i} = \lambda R_{b,i-1} + S S^{H}$$

Equation 6

Here b is the sub-band index, i is the counter of how many 1024-sample blocks of microphone data have been transformed into frequency domain since processing began, S is an n by s matrix of frequency bin values, and $0<\lambda<1$ is the forgetting factor.

The forgetting factor helps make sure that MVDR weights applied to each 1024-sample DFT change deviate only slightly from the weights of previous blocks. As the name forgetting factor implies, the recent data is the most heavily weighted, to adapt to recent interference. The memory introduced to the processing via this factor helps the waveform reconstructed in the time domain to track the desired source signal [7]. The forgetting factor generally presents a trade-off between how quickly the algorithm adapts to changing interference (better as factor approaches zero) versus how gradually the weights change to aid in tracking the desired signal (better as factor approaches one). Optimal forgetting factor values therefore depend on the nature of the desired signal and interference and must be determined experimentally.

It must be noted that to avoid the issue of singularity when inverting the covariance matrix to compute MVDR weights, the diagonal of the covariance matrix is multiplied by the conditioning factor of 1.03 – the value cited in [7].

Once MVDR weights are computed for each sub-band they are applied to the bins corresponding to the positive frequencies in the DFT. The conjugate of each set of weights is applied to the corresponding negative frequency DFT bins due to frequency domain conjugate symmetry of a real time-domain signal (refer to Equation 5).

It must be noted that the steering vector in the algorithm implemented was designed to assign zero phase to the steering vector element corresponding to the microphone closest to the desired source. This way, MVDR beamforming would strive

to have the processed signal aligned in time with the desired component of the signal in the closest microphone. Thus MVDR performance could be derived via direct subtraction of the processed signal from the reference clean signal from the closest microphone – without having to correct for a phase offset. Of course, this is only useful in experiments as the ones described further where the waveform of the desired signal with no interference is known.

2.3 Previous Work and Collaboration

2.3.1 Previous Work by Author

The author began researching MVDR beamforming in the Autumn 2011 academic quarter at OSU (Sep-Dec 2011). During that time, several preliminary simulation tests have been developed with qualitative and quantitative metrics for the performance of different combinations of MVDR parameters. The near-field model for MVDR has been shown to be superior for the purposes of iBrutus to the classic far-field MVDR model mentioned in the “Design Solution” section. Preliminary simulations have been run to acquire insight on how placement of microphones affects performance. These tests have suggested that for a speech signal band-pass filtered from 300 to 3400 Hz, an array length somewhere between 0.5 m and 1 m produces the best improvement in signal to noise ratio (if eight microphones are used) [3]. The tests done

in the autumn of 2011 were not as sophisticated as the tests developed by the author in 2012. The experience and basic insights acquired from September to December of 2011 have nevertheless been useful.

From January to March of 2012, (Winter 2012 academic quarter at OSU), the author developed a wideband and frequency-based implementation of MVDR. Some tests for the mentioned word recognition percentage and error RMS improvement metric under various parameter choices have been carried out [4]. These tests have been improved (bugs fixed and testing procedure modified) from March to May of 2012 and new results have been obtained.

2.3.1 Collaboration with Other OSU Undergraduates

From January to March of 2012, the author has also collaborated with the ECE undergraduate students mentioned in the “Acknowledgements” section. These students have researched data acquisition possibilities to serve as the input to the beamforming algorithm. More specifically, the team was seeking audio hardware that supports continuous quasi-real time audio recording that iBrutus would require. The team also considered a software interface from the hardware to the algorithm developed in MATLAB by the author (see Figure 2). The team has not been able to design a working quasi-real time implementation of beamforming but has nevertheless laid groundwork for future research.

The team has not found a way for the TASCAM US-1641 unit to support continuous streaming for quasi-real-time beamforming. Using the TASCAM unit, the team has only been able to record audio blocks with short gaps in time which would be

detrimental to speech recognition. Only one PC process at a time can access the ASIO driver for the TASCAM, and gaps would result when a recording process exports a block of audio data to a beamforming process [4]. Again, quasi-real-time processing for iBrutus would require continuous and exporting of data for processing without interrupting the recording. Some possibilities for accomplishing this have been presented in the “Future Work” section. The team has also purchased Analog Devices ADAU1361 which was believed to support streaming quasi-real-time DAQ. Due to poor online documentation for this device, it was only discovered after delivery that this device automatically mixes the input microphone channels into one, voiding the possibility of beamforming [4].

In the end, the team has presented functioning freeware MATLAB code (also available in C++) that could at least automate recording a single block of predetermined duration using an ASIO compatible device such as the TASCAM unit [4]. This code could be useful to future researchers for automating recording in a real environment using any ASIO compatible device. The author has successfully tested this code with his beamforming algorithm during February and March of 2012.

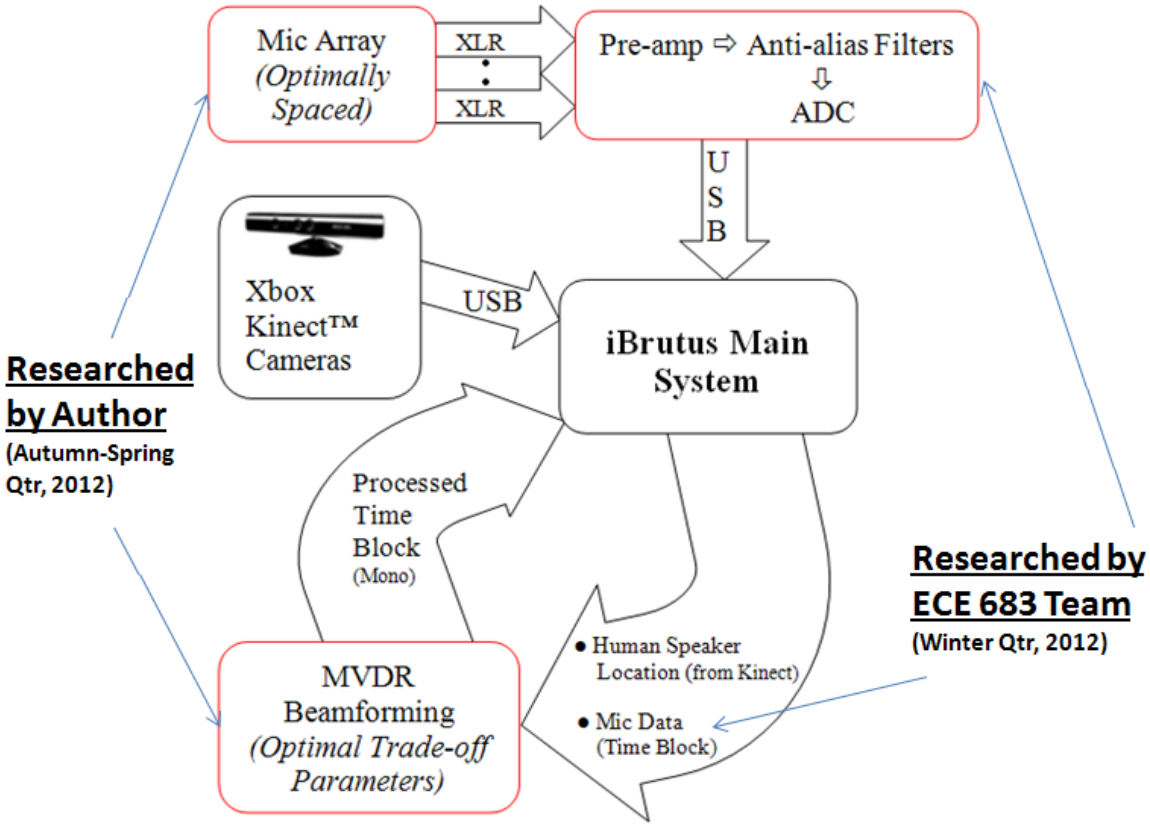


Figure 2. The iBrutus Components Directly Related to the Author's Work.

2.4 Legal, Societal and Economic Considerations

2.4.1 Legal Considerations for iBrutus

Ohio Revised Code 2933.52 states: "No person purposely shall intercept, attempt to intercept, or attempt to use an interception device to intercept or attempt to intercept a wire, oral, or electronic communication". Oral communication is defined in Ohio Revised Code 2933.51 Section A. This section states: "an oral communication uttered

by a person exhibiting an expectation that the communication is not subject to interception under circumstances justifying that expectation". The definition of an "Interception device," is also discussed in detail in Ohio Revised Code 2933.51 Section D, with many clauses [13].

Avoiding potential charges pressed against those who are in charge of iBrutus could be done via a disclaimer. A message could be placed either on a ticket to the Ohio Stadium or at the iBrutus kiosk to let people know that they are being recorded, but that iBrutus will not store the recording longer than needed for processing.

A lawyer will need to be consulted before iBrutus is put to use to determine what the exact legal concerns and solutions are. Future researchers may want to record a realistic noisy acoustic environment on the day of an event at the Ohio Stadium. They are advised to seek a legal permission from OSU officials, perhaps on the premise that the recorded audio may only be used for research purposes and not for intercepting oral communication of bystanders.

There is another potential legal issue to consider. A user may ask iBrutus a question (e.g. directions) and receive a wrong and potentially unsafe answer. As an example, there has been a court case in the U.S.: Rosenberg-v-Harwood-Google. Rosenberg, a woman in the state of Utah, sued Google after receiving bad directions from Google Maps which led to a car accident. Google happened to win the case. The court declared Google owed nothing to Rosenberg due to the fact that Google didn't have any direct legal relationship with Rosenberg [14]. To avoid such issues with iBrutus, again a disclaimer should be put on the device stating that iBrutus' responses may not always be accurate. Like in the first case, a lawyer would need to be consulted

on this particular issue. Some legal solutions should already exist, as there are many similar devices, such as Apple's Siri.

2.4.2 Societal and Economic Impact of iBrutus

A system like iBrutus is part of the effort to develop technology that can communicate with human users with buttons or a touch screen. The CSE iBrutus team has also stated that they are working on utilizing the capabilities of the Xbox Kinnect cameras to detect gestures and read a speaker's lips to augment communication [2]. Such advancements present a paradigm shift in human-computer communication. Society and the economic market of the future will likely be impacted more and more by the presence of systems similar to iBrutus.

2.5 Standards

The first standard that will be used in this project is that of the IEEE standard for a universal serial bus (USB). This method of communication between hardware and the main CPU is very well defined and the rules will simply be followed in accordance with the existing standards.

Another standard that must be acknowledged is that of the Audio Stream Input/Output (ASIO) protocol. The protocol allows for connection directly to a sound card by bypassing several layers of Microsoft software. This allows for increased speed

in processing and allows for a much more streamlined process. The protocol allows for up to 24 bit samples and as many channels as the computer will allow.

All hardware connections are using XLR and ¼ inch connectors. These are standard audio cable connectors.

3. Experimental Procedure

3.1 General Motivation Behind Tests.

Tests of the author's MVDR algorithm have been carried out to observe how varying five parameters mentioned in the "Objectives" section affects noise/interference suppression and word recognition rate.

It was discussed in the author's progress report from December 2011 how frequency content of a signal processed with wideband MVDR influences what uniform array lengths accomplish the most interference suppression [3]. To summarize this discussion, consider the higher frequency sub-band of a wideband signal. The peak point of interference suppression for this sub-band is when a relatively short array length is used (closer microphone spacing). The array length for peak suppression grows as the average frequency within a sub-band lowers. The peak suppression length for the entire bandwidth of the signal will thus be an average of peak suppression points for all the sub-bands within the bandwidth. Better interference suppression roughly translates to higher word recognition rate. The beamforming algorithm, however, can introduce distortion of its own in addition to suppressing interference, especially when not enough interference is present. This has been verified during January-March 2012 [4]. The array length parameter was varied in the tests performed to seek an optimal length for interference suppression and for speech recognition.

To have an accurate measure of what effects array length and other parameters have on speech recognition, a representation of the average frequency content of human speech would need to be known. For example, a speaker with a lower voice would probably be recognized better when beamforming with a longer array, but the goal is to find an array length which works well for a broad extent vocal ranges. The words that a human speaker utters may themselves have varying frequency content. Other factors like intonation and voice effort level (normal vs. raised voice) would have an effect on frequency content and therefore the optimal array length for interference suppression and speech recognition. Thus many words spoken by many human voices ought to be tested. However, such a wide range of speech was not tested, due to research time constraint and limited automation in testing. Instead tests that aimed to simulate and approximate such testing were carried out as described further.

Other algorithm-related parameters such as the width (number of frequency bins) of frequency sub-bands, forgetting factor, and the environment-related strength of interference parameter have been varied as well to observe effects on interference suppression and speech recognition. Some interesting results have been obtained across these parameters.

3.2 Overview of Tests.

Two types of tests were conducted each of which had advantages and limitations:

1. A test with a speech signal was performed. The advantage of this test was the fact that it used the word recognition rate metric which is a direct measure of success for iBrutus development. The limitation of this test was that only the author's voice saying certain words was recorded. Even when a few choices for were made for each parameter, due many combinations and thus many processed wave files resulted. Due to lack of automation for the code, playing of the audio files, invoking the Speech Recognizer Application in Google Chrome, and counting word recognition rate was done manually. To avoid long monotonous labour, a limited, but reasonable set of parameters was tested; only the author's voice has been used. However, automated means invoking speech recognition on processed audio can be conveniently programmed with a scripting language such as Perl, for instance. The author has learned Perl recently and written prototype code which could be used for automated computation of word error rate. Refer to the "Future Work" section.
2. Testing was also done with a non-speech signal – audio recording of a busy day environment at the Ohio Stadium. The advantage of this test was that such a signal would contain the non-stationary properties of a speech signal but would be a closer representation of the average human speech spectrum than the author's voice used in the first test. In other words, having such a non-speech

signal was used in lieu of a many of speakers saying many words. A limitation of such a test was that no direct measure of speech recognition could be used, only a measure of signal error improvement after processing (which was believed to correlate with the speech recognition rate). Another limitation of this test was that the entire frequency spectrum was weighted equally as it has not been researched how the frequencies over the average speech spectrum can be weighted in terms of importance for speech recognition.

More sophisticated testing procedures could be performed given more research time and more automation of testing. These will be discussed in the “Future Work” section. The two types of tests that were carried out by the author are described in more detail in the following two sections.

3.3 First Test: Effect of MVDR Beamforming on Speech Recognition

Simulation tests with non-speech signals which were conducted by the author from September 2011 to March 2012. From the end of February 2012 to March 2012, actual speech was successfully processed by the algorithm. Qualitative listening tests indicated that the algorithm works as the processed signal contained intelligible speech whereas the speech was too obscure in the noisy input signal. However, successful speech recognition has not been achieved until April of 2012. The GSR engine was attempted at that point in time and showed acceptable word recognition rates, whereas the previously used WSR was largely ineffective for a processed signal.

A 13 second utterance was spoken by the author: “Activate computer. Disengage. Stop Listening. Tell us about yourself. What are you? Show commands. Brutus, shut down. Yes. No. Be quiet.” For the sake of relevance to iBrutus, this utterance was a compilation of some phrases to which the system currently responds. The recording was done in a quiet room via the built in microphone array of an HP Pavilion dv4 laptop PC (See the “Hardware Used” section in the Appendix). These microphones were set up to perform weak instantaneous beamforming and reverb cancellation and were thus less sensitive to distant noise [19]. The mono output of the recording was verified with GSR and returned 100% for speech recognition; the quality of the speech sounded crisp and undistorted.

For convenience, the speech and interference was delayed across virtually spaced microphone channels. The speaker’s voice was delayed as if he stood 1 m away from the microphone array center on the line which was the perpendicular bisector of the microphone array (0°). Only this location was used for the human speaker throughout tests for the following reason. Generally, the iBrutus will be able to use the Xbox Kinect to determine the location of any speaker standing 0.8 to 4 m from the Kinect cameras [2]. However, to simplify testing the location mentioned was used as it will yield the best MVDR beamforming performance as it is close to the array and at 0° [3]. It is proposed by the author that when the iBrutus is implemented, this location should be marked on the ground as the preferred location for the human speaker to stand. Mathematically, having the speaker stand at .8 m at 0° would yield even better performance, but a .2 m safety margin is practical so the speaker does not lean out of range of the Kinect cameras. And consequently the beamforming algorithm should be

tested for the parameters to be optimized for this particular location (as it is the best candidate location in the first place).

The 13 second speech signal was mixed with 7 interferers in total drawn from the wave file of crowd noise. The reason for choosing 7 interferers related to use of 8 microphones (degrees of freedom for the MVDR weighted summing) is explained in [4]. The wave file is described in more detail in Section 3.4. The way the interference was implemented for this test (e.g. balancing of the interference frequency spectrum) is similar to the way the interference was implemented in the second test in Section 3.4, except for the duration of interference signals; see below. All interferers had random spatial locations chosen from a distribution identical to the one in the other test described in Section 3.4. Two of the interferers lasted for the entire 13 seconds, while the remaining five had durations between 3 and 10 seconds starting at random time positions within the signal. The rate at which temporary interferers became active and inactive was a guess approximation of a real interference environment like the Ohio Stadium on a busy day.

To reduce the number of output files to manually work with, only a signal to interference (SIR) value was of 2.0 was tested. This amount of interference was enough to obscure the speech signal to the point where speech recognition cannot discern any words. It should be noted that previous tests [4] and recent minor experiments showed author's algorithm to be more detrimental than useful to speech recognition when the SIR was approximately equal to or less than unity. This poses a problem to the beamforming design and will be discussed more in the "Results" and "Future Work"

sections. Minor experiments however showed that processing a signal with SIR values ranging from 1.5 to 5 yields acceptable speech recognition.

Bandpass filters with range 300-3400 Hz and 150-6300 Hz were implemented. In much of previous work, the author approximated the human speech spectrum effective for speech recognition to be 300-3400 Hz. However, starting February 2012, the author attempted to process speech using the wider bandpass filter. When GSR was invoked, it was clear that speech recognition rates were much better for the wider bandwidth; see "Results" section. The original bandwidth conveniently required 144 DFT bins (at 22050 Hz) – a number divisible into many sub-band sizes, using equal-size sub-bands. The new extended bandwidth was chosen by consulting [5] as well as doubling the number of sub-bands to preserve divisibility. Processing time would nearly double; however, the results show that the performance gained may be worth this trade-off.

For convenience, interference was virtually delayed across microphone channels. The microphone array lengths tested were: 0.6, 0.7, 0.8, 0.9, and 1 m. Note that since the author presented the oral defense of his thesis on May 11, 2012 [12], new lengths of 0.7 and 0.8 m were tested.

Based on qualitative listening test from previous work [4], forgetting factors of 0.95, 0.97, 0.98, 0.99, 0.995, and 0.999 were chosen. This range of forgetting factors yielded output that sounded better than when lower values were used. Surprisingly, the lower forgetting factors that produced a more distorted output actually worked better with speech recognition; see "Results" section.

Finally, sub-band sizes of 4, 8, 16, 24 bins were tested as well. Previous tests showed that contrary to the intuition that would be drawn from the discussion in Section

2.2.2, processing with larger sub-band sizes has yielded higher signal integrity [4]. Therefore the larger sub-bands of 16 and 24 that weren't previously tested were chosen. It would be very beneficial, if larger sub-band sizes result in acceptable speech recognition as they require less computation time; the results revealed that this was generally not the case though.

Figure 3 outlines this procedure of this speech recognition test. Note that the word recognition rate metric was computed by manually counting the number of correctly recognized words in the text strings created from every input with GSR.

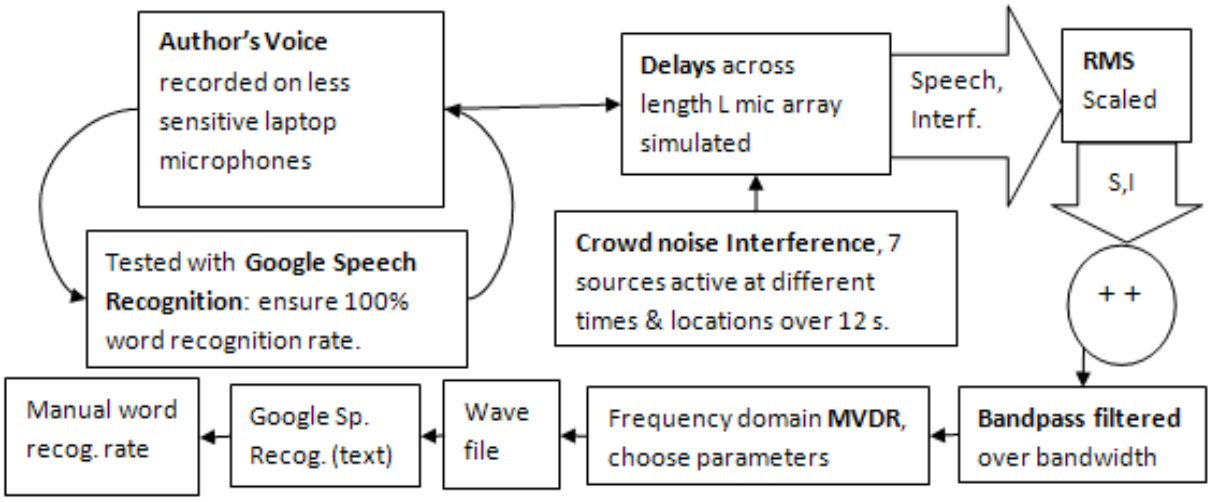


Figure 3. Speech Recognition Test Process.

3.4 Second Test: Approximating Speech Frequencies with a Non-Speech Signal

See Figure 4 for a visual description of the simulation process for this test.

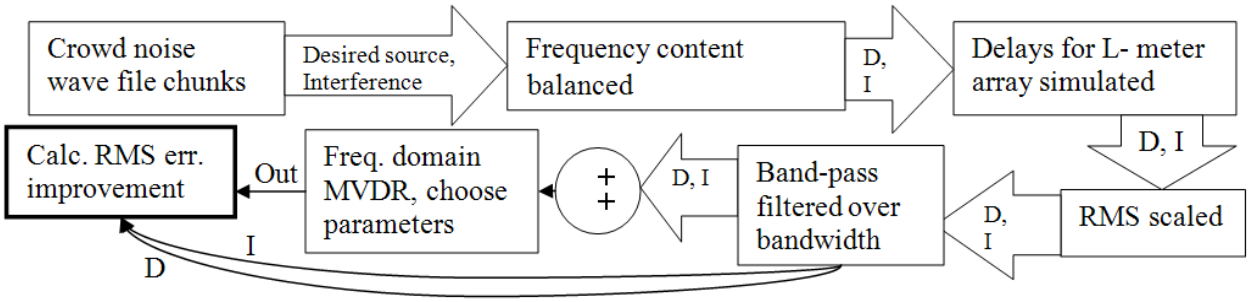


Figure 4. Non-speech Signal Simulation Test Process.

To simulate realistic non-stationary interference short segments of audio data have been randomly drawn from a nine minute wave file of acoustic interference within the hallway spaces of the Ohio Stadium and several other stadiums. This file has been put together from user-contributed recordings on the YouTube video hosting website [15, 16, 17, 18]. Fifty trials (around 0.3 sec: 6144 samples at 22050 Hz) of randomly generating short-time interference profiles were run in the simulation. Each such profile contained a total of seven interference signals active at random time intervals and simulated as if originating from random locations around a microphone array (the number 7 was chosen for the same reason as mentioned in the first test). Running this particular test several times using fifty such trials over a few parameters showed that the error scores varied little from time to time; the trends across parameters remained the same; therefore fifty random trials was sufficient. For convenience in processing, the fifty 6144-sample interference signals were consolidated into one signal. This entire test

signal was MVDR-processed after adding a desired source signal to the interference as will be further described.

The spatial locations of the interference were randomly drawn from a uniform distribution between 1.5 m and 15 m from the center of the microphone array at angles between -90 and 90 degrees.

Interferers at two locations were active throughout the entire 6144 samples. Five other interferers at fixed locations remained active for a shorter number of time samples (50-80% of the 6144 sample period) throughout the signal. It is not believed that interference sources would become active and inactive this rapidly at a location such as the Ohio Stadium. But simulation time would take several days if fifty trials of several seconds of such interference would be used; to speed up the simulations, shorter interference durations were used.

The speech recognition tests from the previous section showed that a processing over 300-3400 Hz while filtering frequencies outside this range fails to produce a 70% or better word recognition rate regardless of parameter choices. The bandwidth of 150-6350 Hz however, showed more success. Therefore, for this simulation test, only the more potent extended bandwidth was considered.

The recordings present in the interference wave file may have been done with poor frequency response microphones and far from all of the noise in the recordings was human speech. It could not be assumed that the frequency content in the recordings was a good representation of the average American English speech content. Nor has it been researched what the magnitude distribution curve of the average speech content is. Instead the frequency content of the interference drawn from the

wave file was balanced to contain rather even magnitude distribution over the 150-6350 Hz bandwidth (filtering away frequencies outside this range). This way, all frequencies in this range would have an equal effect on the performance metric (rather than performance being biased by, say, very strong frequency content in the 500 Hz region). To accomplish this balancing, each DFT bin was multiplied by a scalar which would bring that particular bin's magnitude closer to the average magnitude across the entire DFT spectrum.

The desired signal to be summed with the interference was drawn from the same wave file as the interference. It was balanced the same way over 150-6350 Hz. The desired source was simulated to originate from 1 m away and perpendicular the center of the microphone array (0°).

To test various interference strengths the RMS of the interference was adjusted to several values (whereas the RMS of the desired signal was 1.0). The RMS of the interference would thus be the reciprocal of the SIR (nominal, not dB).

The performance metric for this test was RMS error improvement. It measures how much closer to the desired signal the output is than the input. It was computed as follows:

$$E_{\%,\mathbf{time}} = [\ \mathrm{RMS}\{x_{clean}(t) - x_{processed}(t)\} - \mathrm{RMS}\{\ x_{noisy}(t) - x_{processed}(t)\}\ ] / \mathrm{RMS}\{\ x_{noisy}(t) - x_{processed}(t)\} * 100\%$$

Equation 7

In Equation 7, $x_{clean}(t)$ is the desired signal from the microphone closest to the desired source. Post-MVDR $x_{processed}(t)$ strives to track this signal in presence of interference. “RMS{…}” here is the root mean square of the vector enclosed.

An error score of 100% would mean that MVDR recovers the desired signal perfectly, where as an error score of 0% or less would indicate that MVDR makes no improvement to the signal.

Several parameters were varied. Array lengths from 0.55 to 1 m in 5 cm increments were tested. Note that additional lengths have been tested since the author’s Oral Defense presentation on May 13, 2012 [12]. The optimal array length found in preliminary simulations of September-March 2012 was observed to be around the upper length constraint of 1 m, but possibly somewhat less [3, 4]. With updates made to the simulation test and a new spectrum of 150-6350 Hz, this range of lengths was chosen to see if there what the performance trends would be over the range of lengths stated above.

Additional parameters tested were sub-band sizes of 1, 2, 4, 8, 12, 16, and 24 bins, forgetting factors .85, .9, 95, .99, and SIR values of .5, 1, and 2. If the author’s beamforming algorithm were to be implemented for iBrutus, the forgetting factor would have to be determined experimentally by testing in the acoustic environment at the Ohio Stadium, for example. Several values closer to 1 were chosen to observe performance trends, as previous simulations done by the author as well as sources of literature showed that forgetting factors in this range tend to perform better.

4. Experimental Results

4.1 Speech Recognition Test Results and Discussions

Refer to the results plots for the first test in the Appendix for this discussion. Note: that additional data was obtained (for 0.7 and 0.8 m array lengths) and results were re-examined since the author's Oral Defense on May 11, 2012 [12]. The following trends were observed regarding parameters:

- Bandwidth:
    - Only the 150-6350 Hz bandwidth achieves consistent 70+% word recognition (for most combinations of other parameters).
    - 300-3400 Hz almost always yielded a word rate of less than 70%. For brevity, results with the 300-3400 Hz bandwidth were omitted. This bandwidth is not recommended for future research. Many consonant sounds that help distinguish one word from another have important frequency content above 3400 Hz, but below 7000 Hz [5].
- Array length:
    - The only trend that clearly stands out regardless of what other parameters are used is that only 0.9 m and 1m arrays consistently achieve a word recognition rate which is greater than 70%. However, the author's voice is has lower frequency content than the average human voice. As discussed

before, lower frequencies benefit from using longer arrays for MVDR beamforming.

- Sub-band size:
    - Performance is generally better as the sub-band size decreases, although there is deviation from this trend for certain choices of other parameters. This is expected from the discussion of wideband MVDR in section 2.2.2.
    - There is not a significant improvement for doubling the computation time when the sub-band size is decreased from 8 to 4 bins. Eight-bin sub-bands still perform fairly well (above 70% word recognition) so given the results of this test alone, eight bins may be a viable trade-off between processing speed and performance for iBrutus.

- Forgetting Factor:
    - The lowest two factors of 0.95 and 0.97 would generally perform better than the four higher values tested. A qualitative listening test revealed that the signals processed with these factors sounded less pleasing and harder to understand. The signals processed with higher forgetting factors sounded better, but had more reverb as the forgetting factor increased. Perhaps GSR has more trouble recognizing speech with reverb present than the human ear.
    - Again, forgetting factors should be determined experimentally based on the acoustic environment and interference. Several values were used in

this test as it is not known what value will be chosen for beamforming for iBrutus.

4.2 Non-Speech Test Results and Discussions

Refer to the results plots for the second test in the Appendix for this discussion. Note: that additional data was obtained (for 0.55, 0.6, 0.65, 0.7, and 0.75 m array lengths) and results were re-examined since the author's Oral Defense on May 11, 2012 [5]. The following trends were observed regarding parameters:

- SIR:
    - If inference is weaker than desired signal significantly lower error improvement is observed.
    - For SIR of 0.5 (strongest interference tested) the best error improvement is observed.
    - This trend has been confirmed by the first test as well (the speech processing test only used an SIR of 0.5 as brief experiments showed that using a high SIR will not improve speech recognition after processing).

- Sub-band size:
    - Larger sub-band sizes (24, 16, 12 bins) are consistently better; this result is counterintuitive, given the discussion in Section 2.2.2.

- Peak performance point for sub-band size may be larger than the maximum size of 24 bins tested; at times however, the next largest sub-band of 16 bins shows better improvement; this is an indication that the peak may be around 24 bins.
- The same trend has been observed for the non-speech test during the January-March 2012, and a possible reason for this phenomenon is offered [4]. However, this trend seems too extreme, and definitely does not agree with the speech processing test. There may be a bug in the simulation code that has not been fixed; or the RMS error improvement may not correlate well with speech recognition performance.

- Other algorithm parameters do not show significant trends in error improvement.

It is concluded that the second test has returned some questionable results and should probably be abandoned in favor of speech recognition testing. Again, this test was carried out because it was hoped it would achieve the equivalent of testing the average human speech spectrum with many interference sources in much less time than the speech recognition test. Full automation for the process of deriving the word error rates has not been available. Therefore this test was used as a rudimentary alternative to the speech processing test, in hopes that the results would coincide well with the less comprehensive speech processing test.

4.3 Processing Time Results.

MATLAB processing time of the author's current MVDR algorithm was measured by the MATLAB tic and toc commands. This was done while running the speech processing test (which again, used only an eight microphone array). Of the parameters varied, only the sub-band size has an effect on processing time. The processing time is expressed in Figure 5 as a measure of how long it takes to process a second of signal (not counting algorithm setup time; this setup would initialize variables like the steering vector and would only occasionally be repeated when beamforming for iBrutus).

It can be seen in Figure 5 that currently the algorithm cannot process a second of signal in under a second. But for convenience, all testing was done only using the authot's HP Pavilion dv4 laptop PC – not an incredibly fast machine by today's standards. There are other potentially significantly faster machines available at the ECE department; future researchers should consult Dr. Potter. There is potential to optimize the algorithm speed without detracting from performance (see Section 5). The goal would be for example to cut the processing time of a 150-6350 Hz bandwidth using 8-bin sub-bands from 3.5 seconds to under 1 second (again, using 8-bin sub-bands has shown to have decent performance).

Note that the more sub-bands are used, the less the algorithm slows down in response to doubled bandwidth. This can be seen from the slope of the "Slow-down Ratio" curve in Figure 5.

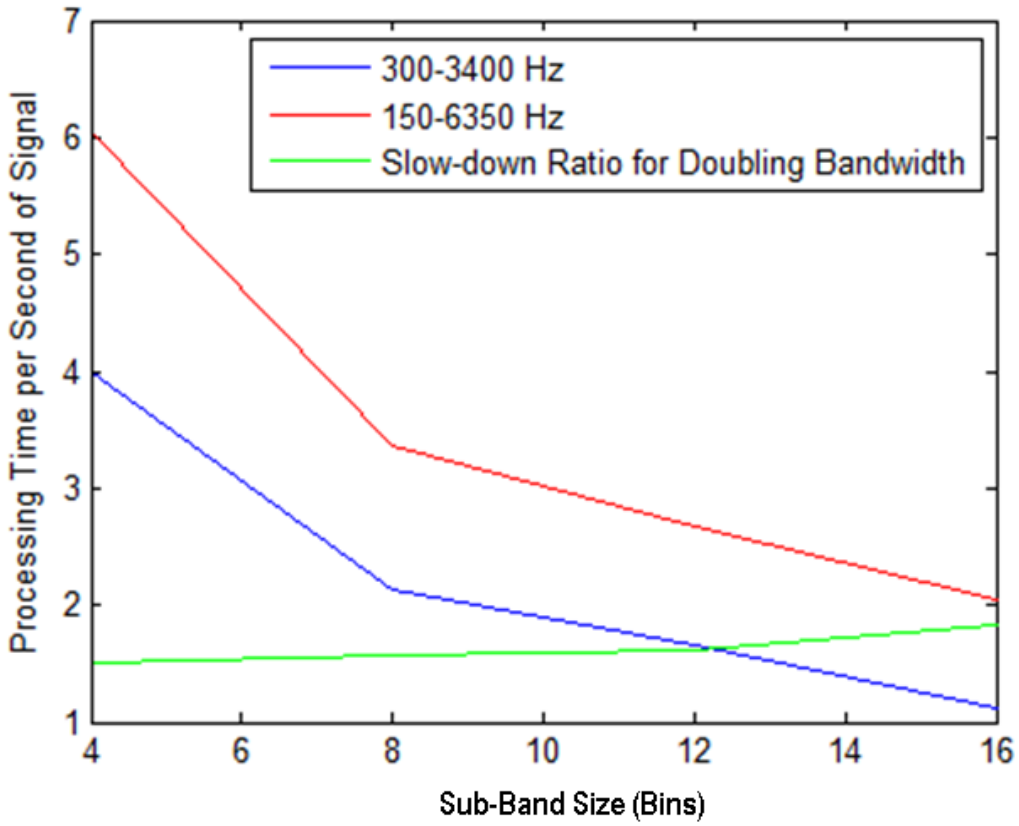


Figure 5. Effect of Sub-band Size on MATLAB Processing Time.

5. Future Work

Three areas of future advancement directly related to the author's work on iBrutus have been identified. Advice in these areas is offered to future researchers in the following sub-sections.

5.1 Improving Testing Procedure to Find Beamforming Parameters Optimal for Speech Recognition

- Simulation Tests should improve in the following general areas:
    - Accomplish more automation for speech recognition testing.
        - The author has written working code in Perl (see "Author's Code" section of the Appendix). This code uses downloadable Perl modules which interface with a 32-bit Windows system to allow mouse clicks, key presses, access to windows in the taskbar and access to the clipboard. The code plays wave files that the author's MATLAB speech processing test has generated and invokes speech recognition in Google Chrome. It then copies the recognized text and prints it to a results file.
        - No code has yet been written to automate word recognition rate computation from a string of words. Therefore the author has done this part manually. Comparing two strings of words to determine the word recognition rate could be a difficult and ambiguous process.

There may be existing freeware code which accomplishes this. Otherwise, a solution may be to create a separate wave file for every word a recorded speaker says and tally the recognition success count word by word.

- Sometimes, when audio consisting of several words is played, GSR stops listening and converts to text prematurely; to avoid this issue, it would again help to design automation to play one word at a time.
- Note: it has been observed that speech recognition engines like WSR and GSR have can produce different outputs, when the same exact audio file is played several times; therefore for testing, several trials of speech recognition are recommended for a single audio source; then the word recognition rate can be averaged.
- Sometimes GSR displays an error message saying that it cannot connect to the server; one should be set up automated tests to detect this in some way and redo the trial in such a case.

- Testing multiple human speakers of various ages, genders, and vocal qualities; more test words than the author attempted should be used.
    - Note that such testing may require a lot of hard drive space for uncompressed wave files (compressed files may be detrimental to speech recognition); MATLAB may require more memory than available so code may have to be modified to perform a section of work at a time; testing will require a lot of time.
- Recording audio which is more realistic for the purposes of iBrutus

- Interference should be recorded into a microphone array (or several physically adjusted lengths) from a real physical noisy environment, preferably the walkway belt under the seats of Ohio Stadium. This will incorporate the following realistic features into tests: moving interference sources (not previously simulated), proper durations of interference, proper levels of interference, and proper statistical properties of interference.
- Note: officials should be consulted on legality of recording.
- For convenience, the human speech may still be delayed via simulation. However, a reverb profile mimicking that of the mentioned space at the Ohio Stadium could be added to the speech.

5.2 Algorithm Improvements

5.2.1 Performance Improvements

- Research how to avoid the problem that low interference poses to beamforming; this problem may be a quirk of the frequency domain based algorithm.
    - One possible solution is to artificially mix in a minimum level of interference with the signal to maintain the SIR above a certain level. It is not known how likely this approach would be to do more harm than good in the context of iBrutus. Tests have shown that the author's algorithm

successfully extracts relatively clean speech even when SIR is as high as 5, although a ceiling SIR level has not been found.

    - Another method is to research a way to detect when the audio level is below a certain threshold (i.e. interference is low) and conditionally switch to simple delay-sum beamforming which may be less detrimental in that case than MVDR.

- Research and implement the time domain MVDR algorithm; determine whether the low interference problem is present and whether other differences exist.
- The article in which the frequency-based MVDR was described [7] offered an adaptive memory element which was somewhat different from the exponential forgetting adapted by the author for simplicity of coding. Implement the forgetting method described in the article, and observe any differences.
- An acoustic space where iBrutus is to reside (like that within the belt of hallway at the Ohio Stadium) may induce a lot of reverberation. Research an algorithm that removes reverberation from speech and determine if it is worth the extra processing time.

5.2.2. Reducing Beamforming Computation Time:

- Current algorithm is in MATLAB and could be converted to compiled languages such as C/C++ which could possibly run faster.
    - MATLAB 11 toolbox which offers automatic conversion to C/C++ is $500. This toolbox might not use proprietary fast algorithms built into MATLAB and replace them with more rudimentary slower algorithms [10].

    - Code can be converted manually
        - C/C++ libraries for faster computational routines such as FFT, fast matrix multiplication, etc. are available commercially or as freeware.

- Code could be optimized for speed while maintaining performance:
    - Decimation of the current algorithm input at 22050 Hz to a lower rate may yield similar performance while speeding up the code. For an aliasing-proof safety margin a sampling rate of less than 13 kHz Hz is not recommended if frequencies up to 6350 Hz are to be used.
    - The matrix inversion lemma for updating the inverse of the MVDR covariance matrix can be implemented. This will possibly speed up the algorithm.
    - For the lower frequency sub-bands fewer channels than eight could be processed. As lower frequencies do not require as close of microphone spacing (discussed in [3]), using every other microphone in the array for the lower frequencies could maintain performance while reducing computation.
    - It is possible that the higher frequencies of human speech (e.g. 3400-6350 Hz) do not require as accurate of processing as the lower frequencies. Sub-band sizes for these frequencies could be made large, reducing computation while possibly maintaining performance.

5.4 Interfacing Beamforming with the iBrutus System

- Research and develop methods of Data Acquisition/Transfer:
    - Research devices compatible with ASIO or MATLAB DAQ Toolbox. Either of these avenues could allow continuously recording and processing blocks without time gaps in audio.
        - MATLAB DAQ Toolbox may be more convenient as it could be directly interfaced with the author's code [11].
        - ASIO devices will likely require programming knowledge at the driver level. MATLAB has already been shown by the author's colleagues to be unable to record without gaps using the ASIO driver [4].
        - It could be beneficial to acquire hardware which could support 12 or 16 microphones. It may turn out that the multitude of interference sources in the acoustic environment of iBrutus may require a large number of microphones for successful beamforming.
        - The report of the author and his colleagues from March 2012 offers more detailed advice on DAQ hardware [4].

- Work on integrating beamforming into the iBrutus system
    - Consult the CSE iBrutus team at OSU headed by Thomas Lynch on how a beamforming algorithm and DAQ hardware could be interfaced with the system.

- The iBrutus System is currently written in C# and uses Windows Speech Recognition.
- Note again: the author was only able to obtain successful speech recognition results when using Google Speech Recognition. Though GSR may not integrate as well with iBrutus. It is slower than WSR (built into the Windows Vista and Windows 7) as it relies on an internet connection; sometimes there is a server error which requires redoing speech recognition. GSR does not support a streaming audio input like WSR and must be invoked repeatedly for recognizing words.
- Advancements in these or other speech recognition systems may be made soon. For example, the quality of WSR could be improved by Microsoft while a new release of Google Speech Recognition could be more compatible with iBrutus.
- Develop means of estimating what processing time a beamforming algorithm would require to maintain quasi-real-time processing for iBrutus.
    - For quasi-real-time processing:

$$T_{beamform,block} + t_{overhead} < t_{record,block}$$

Equation 8

In other words, the beamforming time for a block of audio combined with the overhead time it takes for the recorded audio to be accessed by the algorithm should be below the recording time of the same block.

- iBrutus needs fast, multi-core machines to be able to run several components of the system at once. Consider benchmarking processing time on machines

such as those on which iBrutus would be implemented. Consult with the iBrutus team on what computing resources could be used.

6. Conclusion

A potentially viable beamforming algorithm has been developed by the author. The algorithm meets the desired 70% word recognition level under certain parameters when using eight microphones. However the algorithm needs processing speed improvement; several avenues for this improvement have been offered. Also the issue of unsuccessful speech processing in the case of low interference case must be addressed.

To ensure that the algorithm performs well in the required processing time, more testing with beamforming followed by speech processing will need to be done as mentioned. Testing procedures have been laid out and enough testing code has been written to provide a good starting point for future researchers. Running speech recognition must be automated as many audio files are required to be processed for comprehensive testing. For this, prototype Perl code has been provided; it could be modified as needed.

For interfacing beamforming with iBrutus, proper DAQ hardware must be acquired; the algorithm must also be interfaced with the system in software. Eventually, a working iBrutus component must be able to process a noisy input to aid speech recognition while incurring a time delay small enough for users of iBrutus to be satisfied. Advice on how to accomplish these goals has been offered.

References

[1] Ramnath, Dr. R. (2011). *iShoe: Mobile* [Online]. Available: http://iss.osu.edu/iShoe/site/mobile/about/index.php

[2] Lynch, T. (2011-2012, October-February). Technical meetings regarding iBrutus.

[3] Preobrazhensky S. (2011, December). *Acoustic Interference Suppression for iBrutus Project.* [Online] Available: https://sites.google.com/site/osuacoustics/classroom-news/ece699reportdec2011

[4] Belgiovane D, Blanton N, Lane J, Leonard D, Miller M, Preobrazhensky S. (2012, March.) Winter Quarter Progress Report. [Online] Available: https://sites.google.com/site/osuacoustics/classroom-news/march2012reportonibrutusmicrophonearrayproject

[5] Cisco. (2007, December). *Wideband Audio and IP Telephony.* [Online] Available: http://www.cisco.com/en/US/prod/collateral/voicesw/ps6788/phones/ps379/ps8537/prod_white_paper0900aecd806fa57a.html

[6] Bednar J, Jender S, Ehret A, Kondo Y. (2011, June 7). The Ohio State University. Columbus, Ohio, USA. *iBrutus Acoustic Array: Team Gray ECE 682: Final Report.*

[7] Lockwood E, Jones D, Lansing C, O'Brien W, Wheeler B, Feng S. (2003, July). *Effect of Multiple Nonstationary Sources on MVDR Beamformers.* [Online] Available: http://www.brl.uiuc.edu/Publications/2003/Lockwood-Asilomar-7302003.pdf

[8] Boonstra A. (2007, Novemeber 29). *Digital Signal Processing and Beamforming.* [Online] Available: http://www.astron.nl/other/workshop/MCCT/ThursdayBoonstra.pdf

[9] Schniter, Dr P. (2008, March 24). *ECE 700 Digital Signal Processing.* [Online]. Available: http://www2.ece.ohio-state.edu/~schniter/ee700/index.html

[10] MathWorks. (2012). *MATLAB Coder Toolbox.* [Online]. Available: http://www.mathworks.com/products/matlab-coder/

[11] MathWorks, (2012). *Data Acquisition Toolbox Supported Hardware.* [Online] Available: www.mathworks.com/products/daq/supportedio.html

[12] S. Preobrazhensky. (2012, May). *Undergraduate Honors Thesis Defense ECE 683H.* [Online]. Available: https://sites.google.com/site/osuacoustics/classroom-news/may2012oraldefensepresentationibrutusmicrophonearrayproject

[13] LAW Writer® Ohio Laws and Rules. (2012). *Chapter 2933: PEACE WARRANTS; SEARCH WARRANTS* [Online]. Available: http://codes.ohio.gov/orc/2933

[14] Searchengineland.com. (2011, June). *Court Says No, You Can't Sue Google For Bad Walking Directions* [Online]. Available: http://searchengineland.com/court-dismisses-google-walking-directions-lawsuit-claims-82312

[15] YouTube. (2010, November). *How to sneak into OHIO STADIUM.* [Online]. Available: http://www.youtube.com/watch?v=tmaQPTRf-2Y.

[16] YouTube. (2008, September). *OSUMB Drum Line Entering Skull Session.* [Online]. Available: http://www.youtube.com/watch?v=OFXsrdiPr24.

[17] YouTube. (2008, January) *Penn State White Out - Tunnel Walk.* [Online]. Available: http://www.youtube.com/watch?v=DN6SCrU681o.

[18] YouTube. (2009, October). *OSUMB - The Ohio State University Marching Band pre-game.* [Online]. Available: http://www.youtube.com/watch?v=7pTr8JMvsAY.

[19] Hewlett Packard. (2008, November). *HP Pavilion dv4 Entertainment PC Maintenance and Service Guide.* [Online] Available: http://h10032.www1.hp.com/ctg/Manual/c01597750.pdf

[20] TASCAM. (2012). *Product: US 1641 | TASCAM.* [Online]. Available: http://tascam.com/product/us-1641/

[21] CUI Inc. (2012). *CMC-2742WBL-25L | Microphones | CUI, Inc.* [Online]. Available: http://www.cui.com/Product/Components/Microphones/Electret_Condenser_Microphone/CMC-2742WBL-25L

Appendices

A1. Results Data Plots

A1.1 First Test Results: Effect of MVDR Beamforming on Speech Recognition

Note: only a bandpass filter of 150-6350 Hz was used in this test.

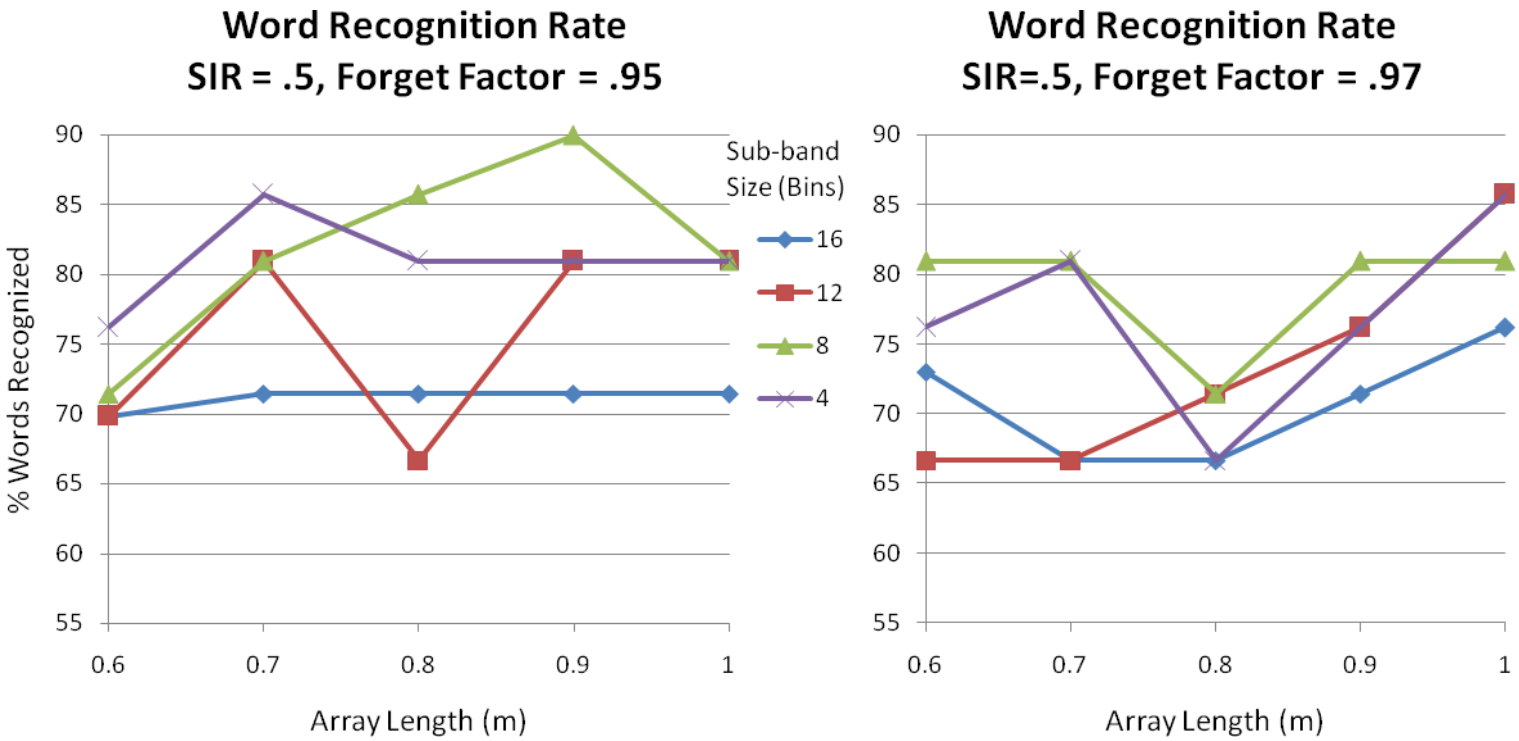


Figure 6. Test 1 Results 1.

Figure 7. Test 1 Results 2.

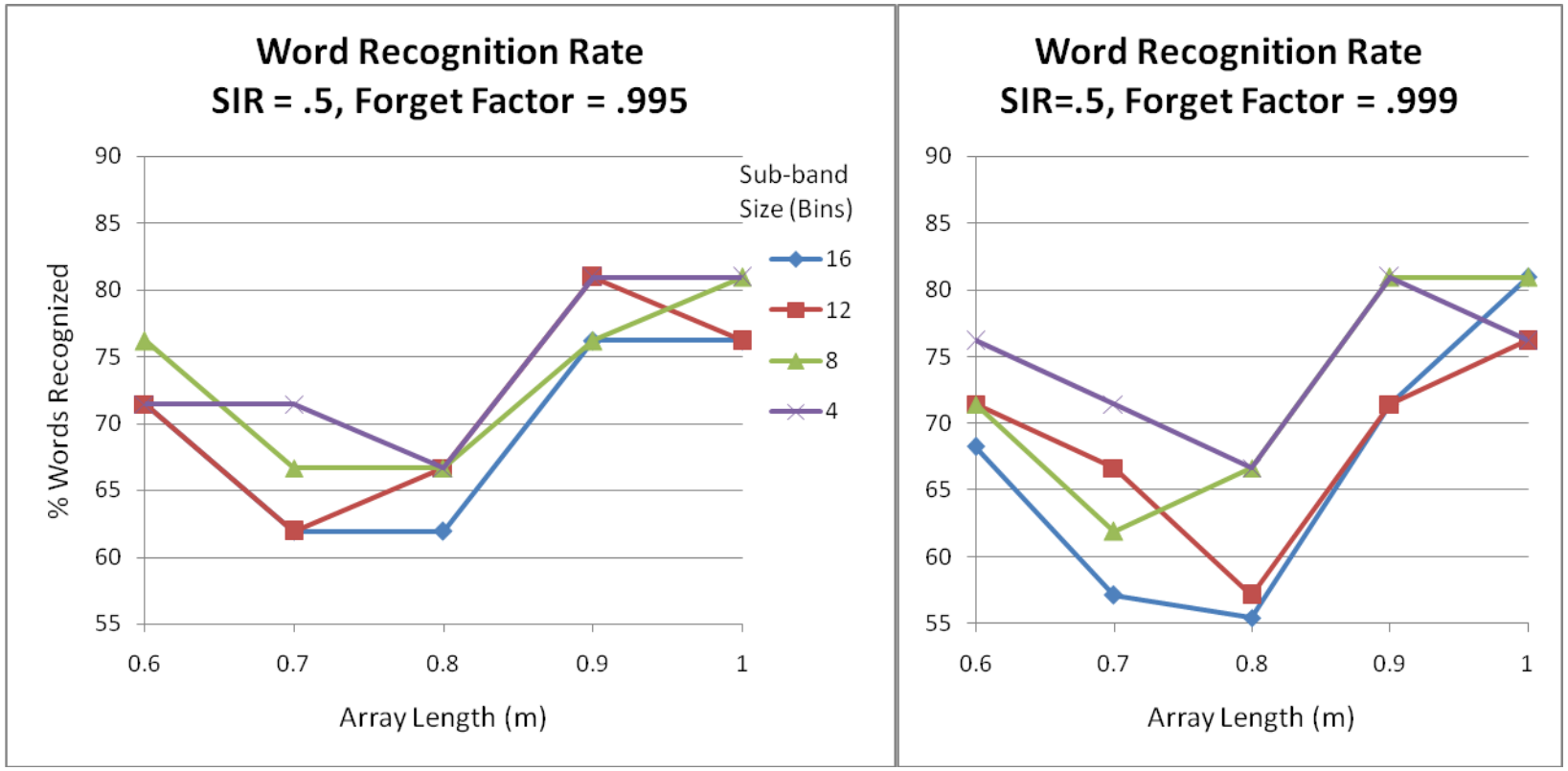


Figure 8. Test 1 Results 3.

A1.2 Second Test Results: Approximating Speech Frequencies with a Non-Speech Signal

Note that again only the 150-6350 Hz bandpass filter has been used in these tests.

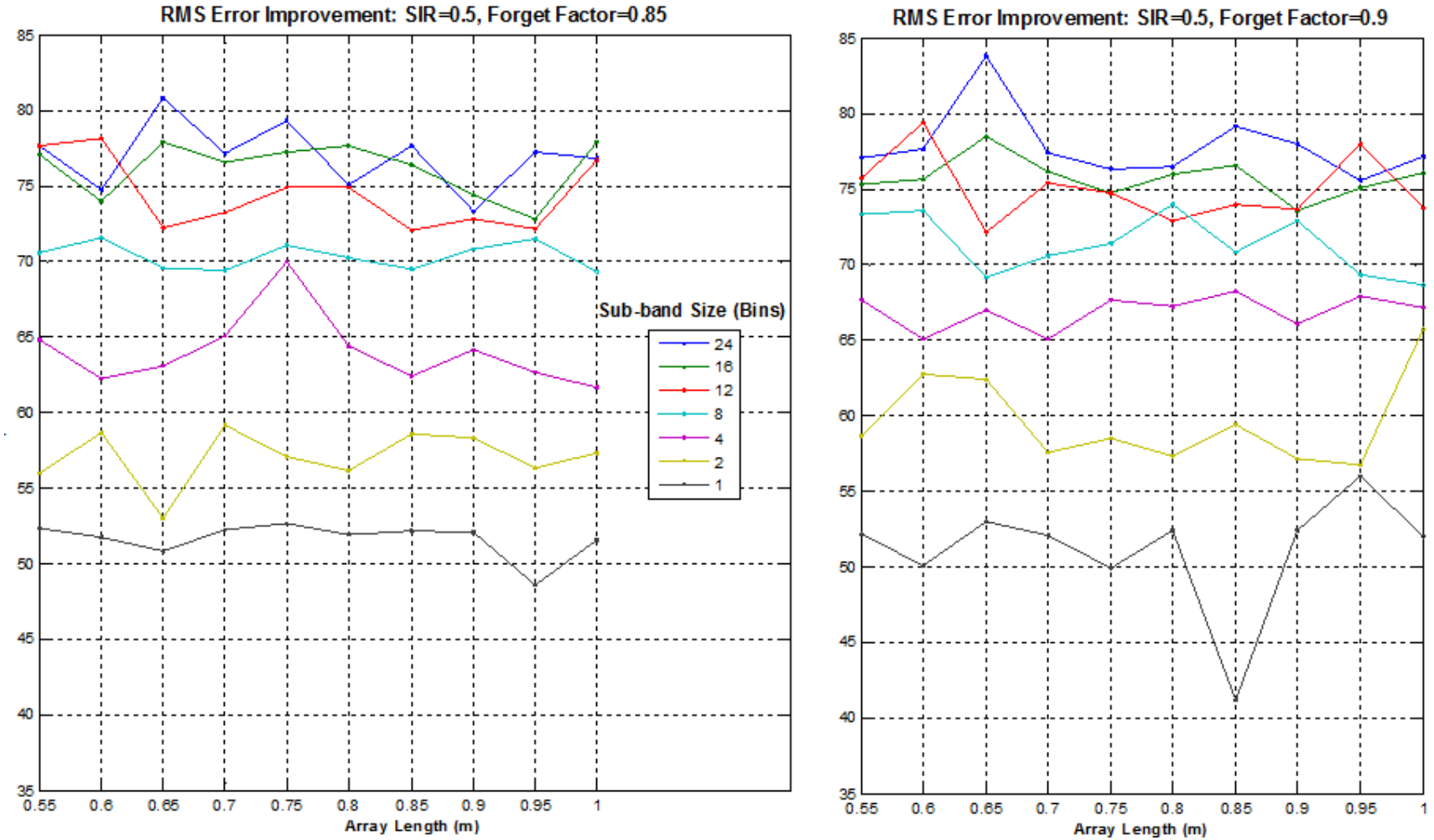


Figure 9. Test 2 Results 1.

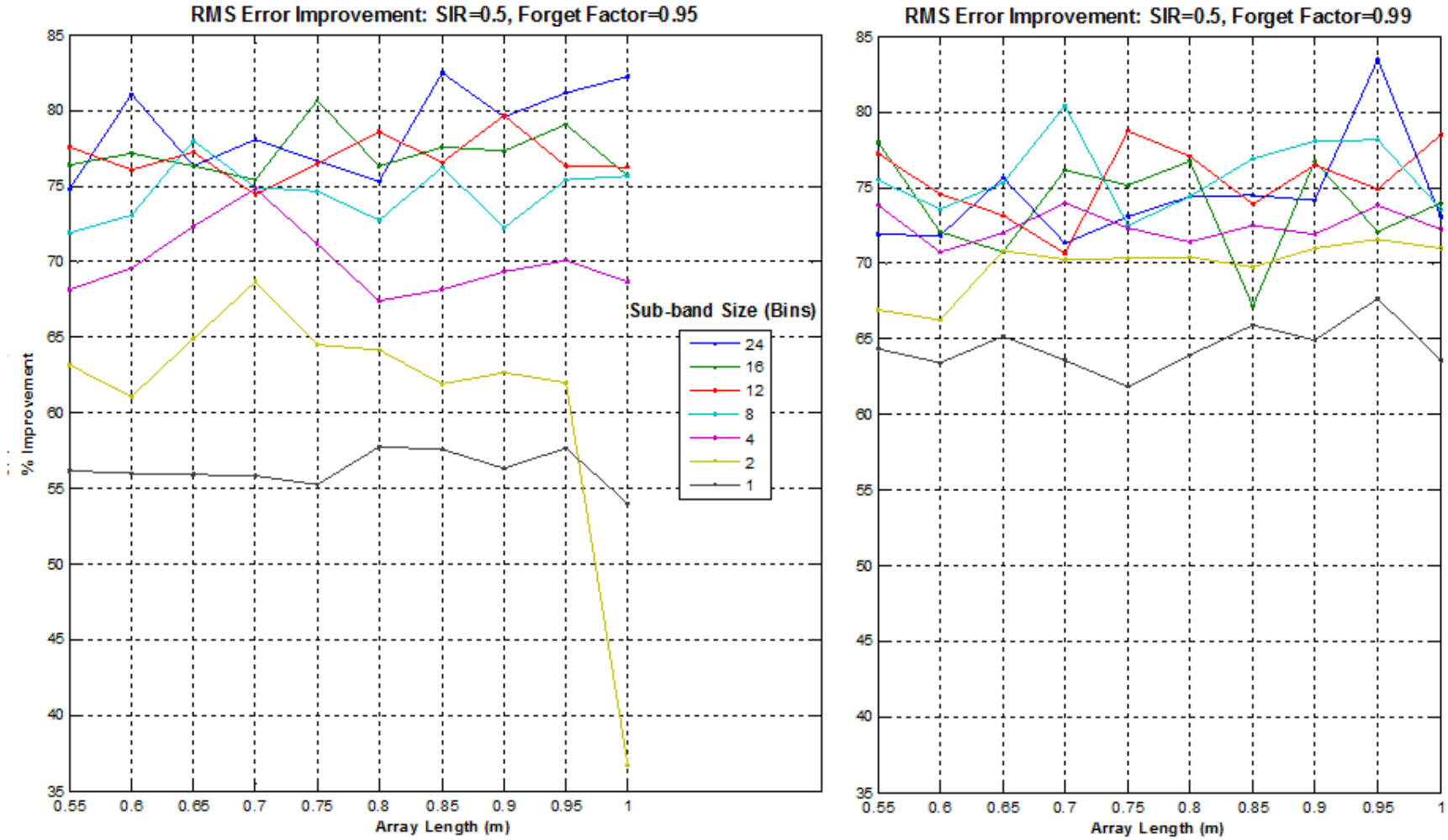


Figure 10. Test 2 Results 2.

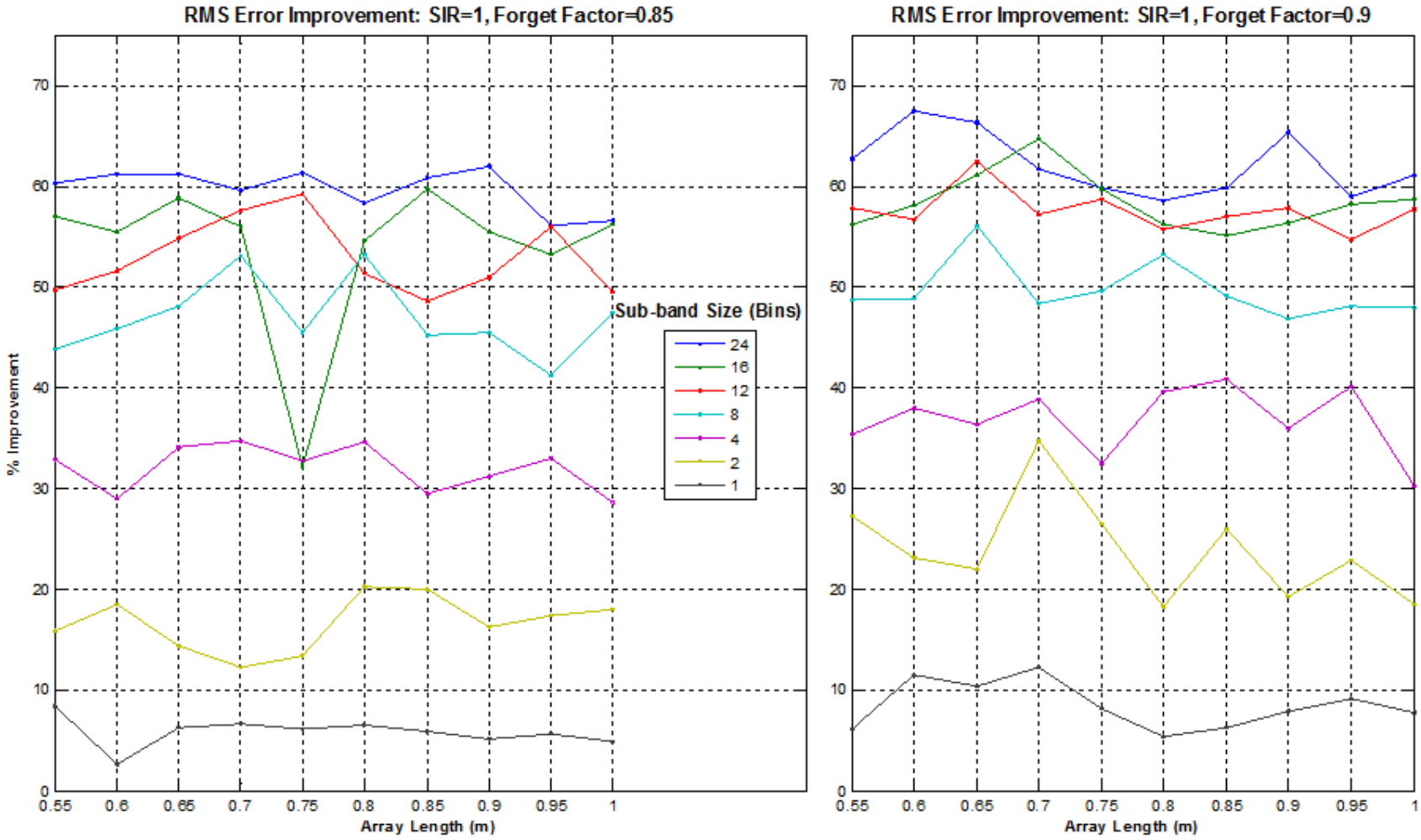


Figure 11. Test 2 Results 3.

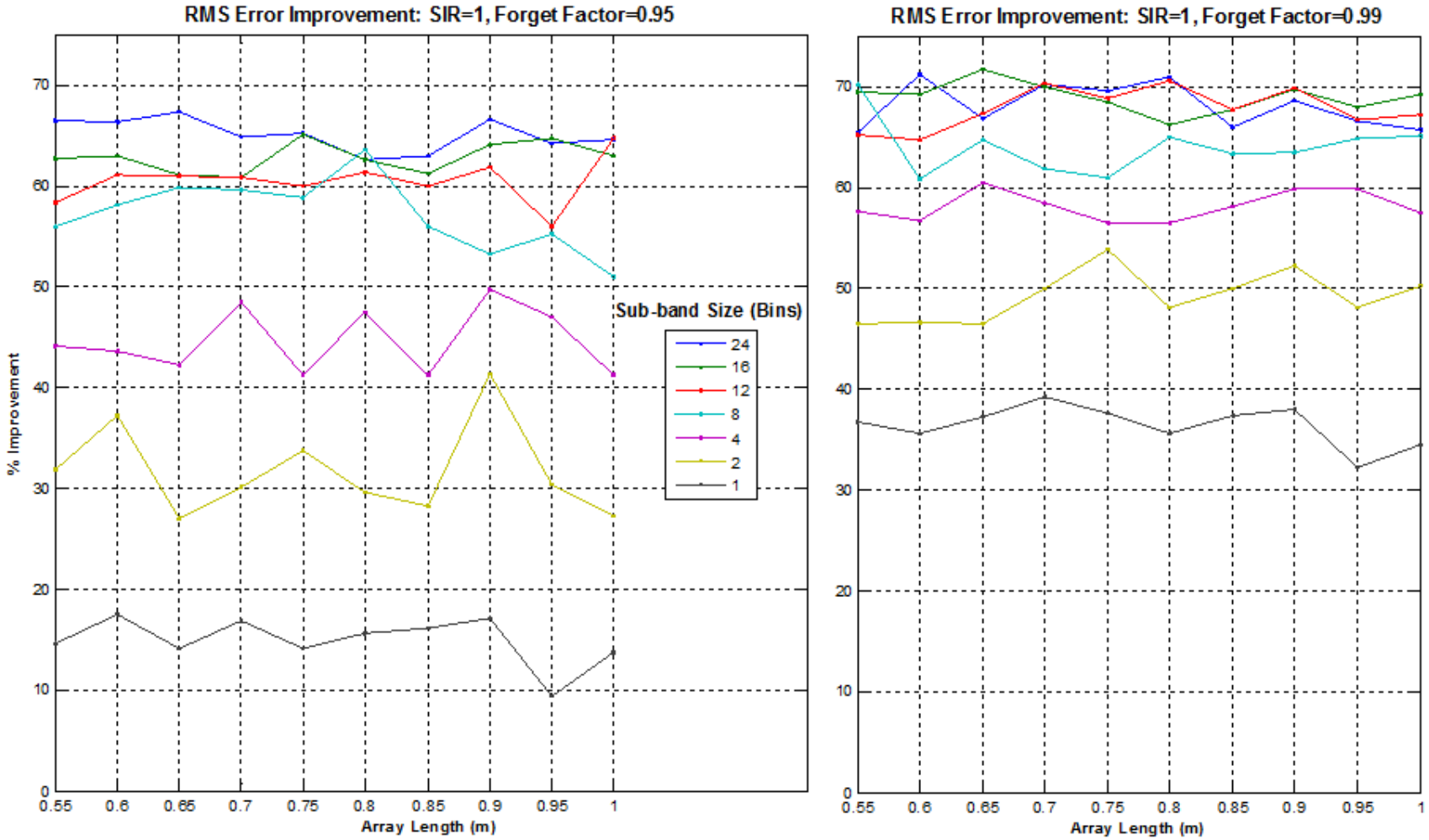


Figure 12. Test 2 Results 4.

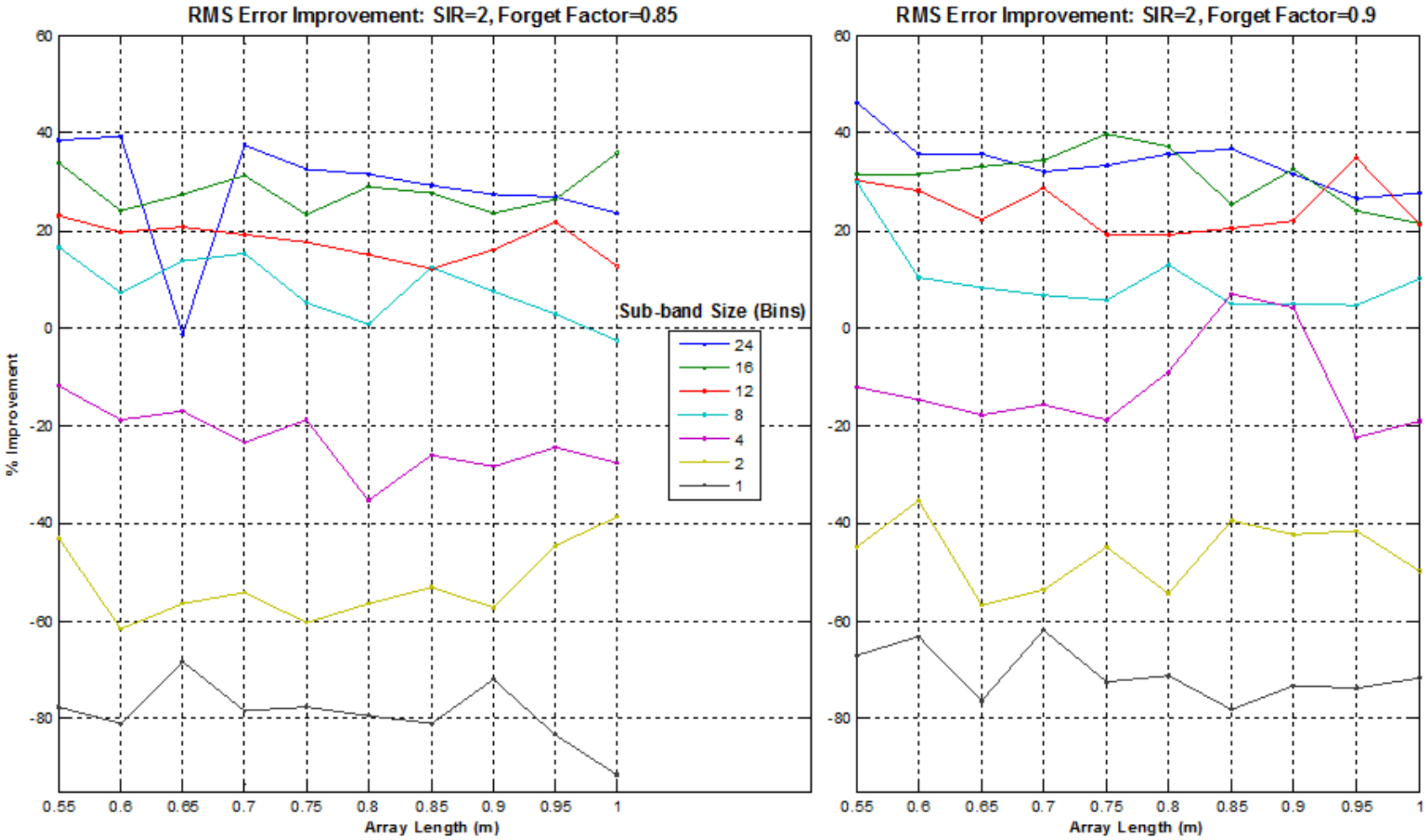


Figure 13. Test 2 Results 5.

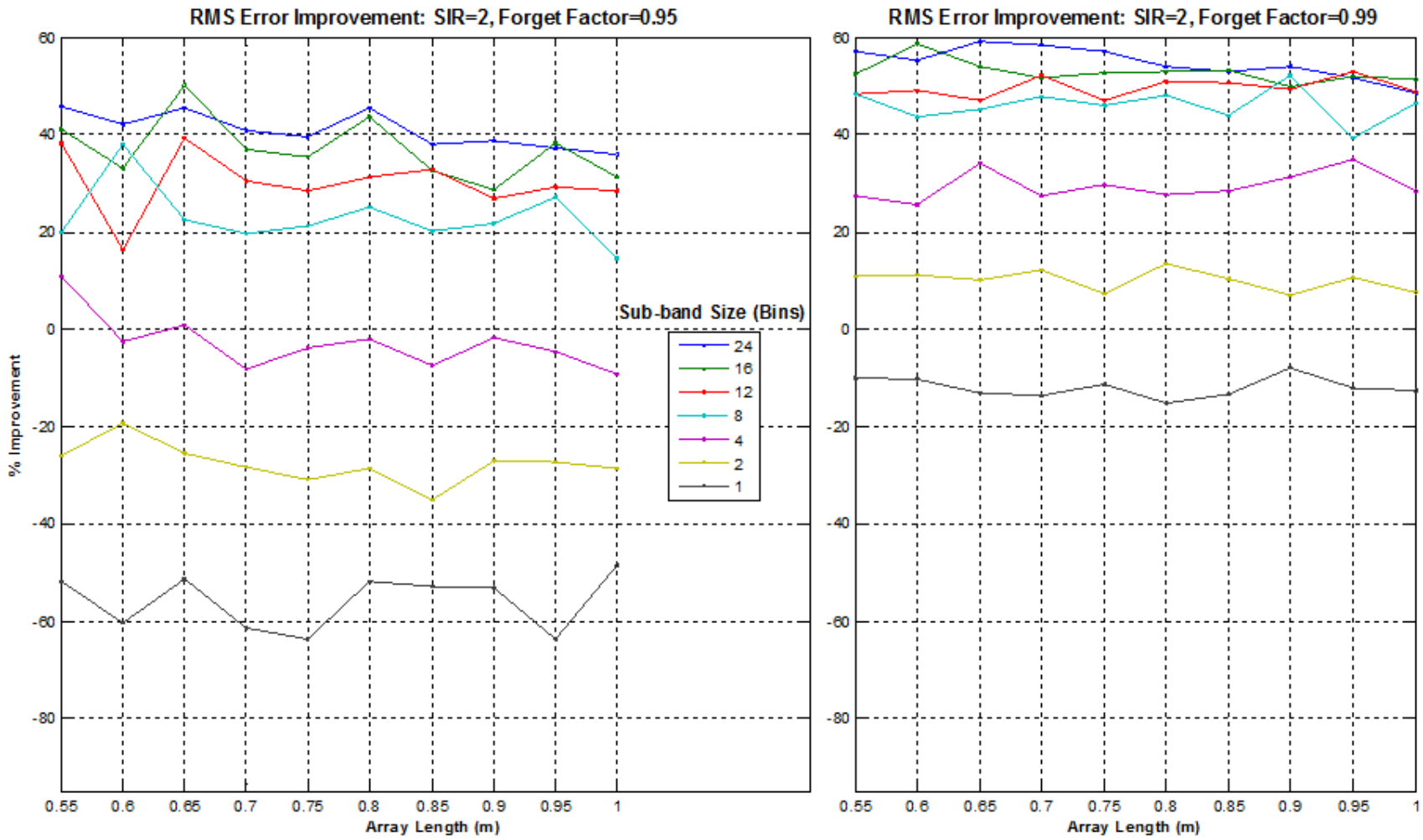


Figure 14. Test 2 Results 6.

A2. Hardware and Materials Used

Note: For OSU-based research, Consult Dr. Potter of the ECE department at OSU; all hardware and materials listed except for HP Laptop PC are available through the ECE department. Other computers are available through the ECE department.

A2.1. HP Pavilion DV 4 Entertainment Laptop PC

Figure 15. HP Pavilion DV 4 Laptop.

Courtesy: http://www.notebooknotes.com

- Used for recording and running tests.
- Dual array microphones used. See [19] for more information.

A2.2 Hardware and Materials used for DAQ only during Jan-Mar 2012

These were used by author and colleagues for microphone array recording [4].

A.2.2.1 TASCAM US 1641 audio ADC/DAC

TASCAM

US 1641

Technical Documentation

PRODUCT OVERVIEW

Utilizing the latest high-speed USB 2.0 technology, the TASCAM US-1641 packs the interfacing power of a big console into a single rack space.

The eight mic/line inputs utilize TASCAM's Professional Mic Preamps providing 60 dB of gain with phantom power. Two, front panel, balanced TRS balanced line inputs; four, rear panel, line inputs; and a S/PDIF pair (switchable to AES/EBU) complete the US-1641's 16-channel inputs. Rear panel outputs consist of four, balanced (line level) and a left/right channel (balanced) monitor output.

MSRP $ 499.00

1-U Rack-mountable

Eight, TASCAM Pro Mic Pre-Amps w/phantom power.

Six, balanced ¼ TRS line inputs (two on front panel)

Four, balanced line outputs

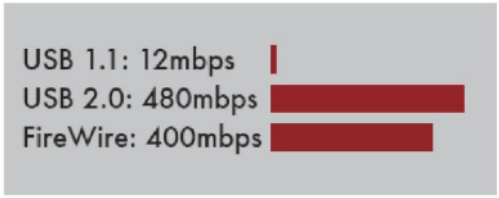
Stereo S/PDIF digital input and output

Headphone Out w/volume control

High Speed USB 2.0

Built in Power Supply (No wall wart!)

96kHz/24-bit support on all inputs and outputs



TASCAM 7733 Telegraph Road Montebello, CA 90640 (323) 726-0303 http://www.tascamcontractor.com

Figure 16. TASCAM Data Sheet. (For more datasheet information: [20]).

A2.2.2 CUI, Inc. Microphones.

CUI INC

date 06/2008

PART NUMBER: CMC-2742WBL-25L **DESCRIPTION:** electret condenser microphone

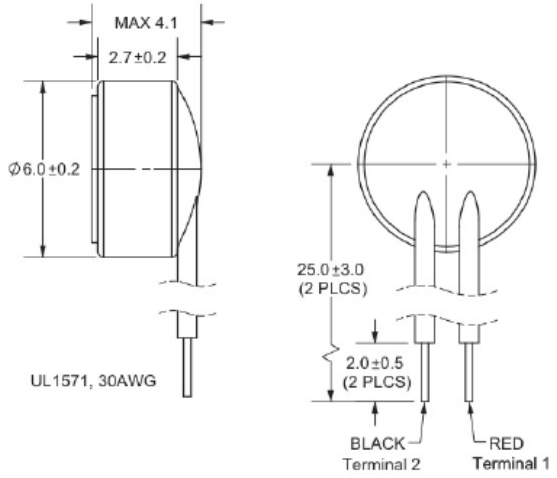
SPECIFICATIONS

directivity	omnidirectional	
sensitivity (S)	-42 ±3 dB	f = 1KHz, 1Pa 0dB = 1V/Pa
sensitivity reduction (ΔS-Vs)	-3 dB	f = 1KHz, 1Pa Vs = 2.0 ~ 1.5 V dc
operating voltage	2 V dc (standard), 10 V dc (max.)	
output impedance (Zout)	2.2 KΩ	f = 1KHz, 1Pa
operating frequency (f)	100 ~ 20,000 Hz	
current consumption (IDSS)	0.5 mA max.	Vs = 2.0 V dc RL = 2.2KΩ
signal to noise ratio (S/N)	57 dBA	f = 1KHz, 1Pa A-weighted
operating temperature	-20 ~ +70° C	
storage temperature	-20 ~ +70° C	
dimensions	ø6.0 x 2.7 mm	
weight	0.22 g max.	
material	Al	
terminal	wire type (hand soldering only)	
RoHS	yes	
dustproof and waterproof level	IP57	

note: We use the "Pascal (Pa)" indication of sensitivity as per the recomendation of I.E.C. (International Electrotechnical Commission). The sensitivity of "Pa" will increase 20dB compared to the "ubar" indication. Example: -60dB (0dB = 1V/ubar) = -40dB (1V/Pa)

APPEARANCE DRAWING

tolerances not shown: ±0.3mm



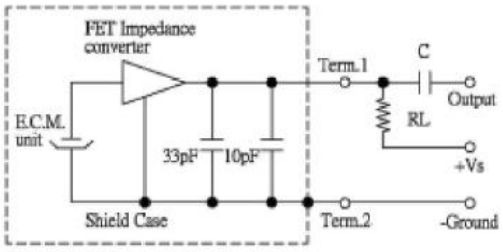
20050 SW 112th Ave. Tualatin, Oregon 97062 phone 503.612.2300 fax 503.612.2383 www.cui.com

Figure 17. CUI, Inc. Microphone Data Sheet Page 1.

PART NUMBER: CMC-2742WBL-25L **DESCRIPTION:** electret condenser microphone

MEASUREMENT CIRCUIT

RL = 2.2KΩ



Schematic Didagram

FREQUENCY RESPONSE CURVE

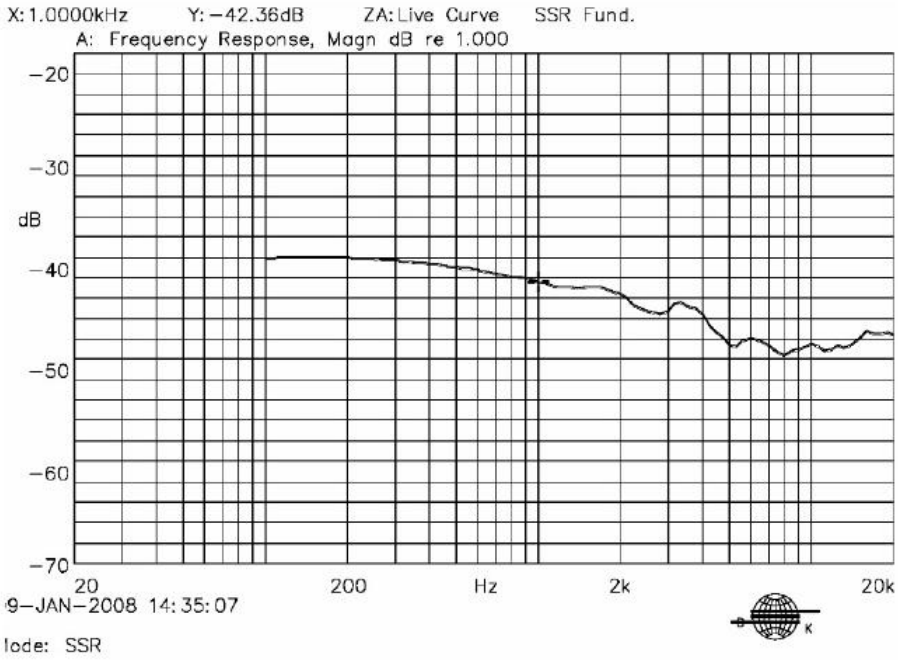


Figure 18. CUI, Inc. Microphone Data Sheet Page 2.

(For more datasheet information see: [21]; for wiring: see [6]).

A.2.2.3 Acoustic Foam Board for Microphone Array Support

- Board is approximately 1.2 by .8 m, depth approx. 1 cm
- Holes can be punctured in the board at microphone spacing of choice.

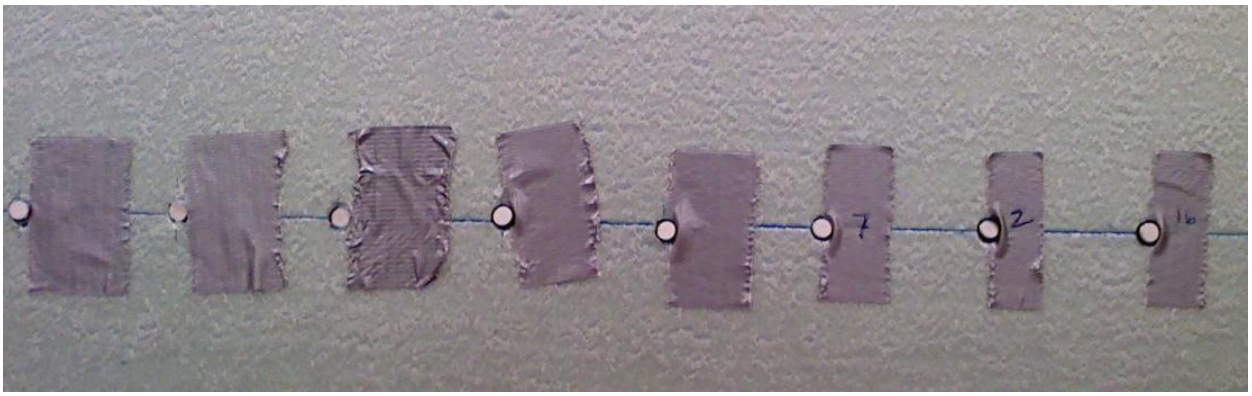


Figure 19. Sample Set Up with Acoustic Foam Board.

Microphones (the mentioned C.U.I. Inc. model) are seen as white circular protrusions.

A2. Software Used

Four main software packages were used to aid the experiments.

Audacity

- Version 1.2.6
- Freeware.
- Used for recording audio.

MATLAB

- Version R2011a.
- Free student edition available through the Ohio State University.
- Used for all code for the MVDR beamforming algorithm and testing.

Microsoft Windows Media Player

- Version 12.
- Freeware.
- Used for playing back audio for speech recognition.

Google Chrome: Speech Recognizer App

- Chrome version 11, Speech Recognizer version 3.
- Freeware; must have a Google account to install GSR App.
- Used as the speech recognition engine.

A3. Author's Code

As the code requires certain files which are not text files, all the material relevant to the code can be found at: https://sites.google.com/site/osuacoustics/classroom-news/ece683hauxiliaryfiles

A3.1 MATLAB for First test (Speech Processing)

MVDR_speech_test.m

```
%This is the main script for testing MVDR beamforming on given (1-channel)
%clean speech signal under various paramters;

%Required function files:
%gen_BPF.m, gener_delayed_interf.m, gener_interf.m, MVDR_speech.m,
%reprow.m, fix_src.m

%Also required: some recorded input audio file at sampling rate
%which is a multiple of 22050 Hz

%By Sergei Preobrazhensky

%NOTE:
%Here input clean source (human speech) is
%delayed across a microphone array as if the source (human speaker) stood
%1 m from the array center on a line which is perpendicular to the array
%and intersects the array at its center.
%Therefore the input recorded speech signal is delayed
%across various microphone array lengths using this assumption.

%Parameters varied:

%Mic array length - vector of lengths 'Lv';
%(assume uniformly spaced, 1-d array of 8 mics)

%Interference strength - vector of values 'interf_rms_v';
%the clean speech signal is scaled to have an RMS of 1 while the RMS
%of the interference is chosen by this parameter;
%the interference RMS value therefore equals to
%1/SIR where SIR is the signal to interference amplitude ratio

%MVDR forgetting factor - vector of factors 'fg_v';
%this is a factor ranging 0<fg<1 used for the memory element of
%adaptive part of the MVDR algorithm

%number of MVDR sub-bands to use - vector of integer values 'num_bands_v';
```

```
%make sure that the number of DFT bins spanning the bandwidth used
%(given Fs=22050 and a 1024-FFT) is divisible by this number;
%this will make each sub-band an equal integer number of bins
%e.g.: 300-3400 Hz => 144 DFT bins, so num_bands_v could be [24,16,12]
%(resulting in corresponding sub-band sizes of 6,9,and 12 bins respectiv.)

%hifi - hifi==0 => bandpass filter of 300-3400 is used for processing;
%effectively reducing the signal to this bandwidth;
%otherwise bandwidth of 150-6350Hz is used


%This script processes the input file under all the combinations of
%parameters and saves the results in a series of wave files
%the names of these output files correspond to the parameters used;

%speech recognition can then separately be run on these files
%and then the word recognition rate can be computed
%to determine which parameter choices results in better rates.

%The input file and save files path is given by 'path' below.

clc
clear all

%preliminary constant setup for test
num_mics=8;
fft_siz=1024;%fft-size used by MVDR algorithm
shift_siz=32;%shift size used by MVDR algorithm
inpad_len=(fft_siz-shift_siz)/2;%pad input by this # samples for MVDR
inpad=zeros(1,inpad_len);
Fs=22050;%sampling rate

Lv=.6:.1:1;
interf_rms_v=[2];
num_bands_v=[9 12 18 36];
fg_v=[.95 .97 .98 .99 .995 .999];

path=...
 'C:\Users\Serge\Documents\z_college\AAiBrutus\MATLAB\Audio_Files\Speech\';
name=input('wav file name of speech signal: \n','s');
Fs_old=input('data sampling frequency, press enter for default of 44.1 k');
if isequal(Fs_old,'')
    Fs_old=44100;
end
[clean_mono,Fs_old]=wavread([path name '.wav']);

%decimate the source signal and cut it to be divisible into fft_siz
%blocks for testing purposes
clean_mono=fix_src(clean_mono',fft_siz,Fs_old,Fs);

sig_len=size(clean_mono,2);
num_blocks=sig_len/shift_siz;

if mod(num_blocks,1)
   error('make sure input wave file fit an integer number of blocks')
end
ham_block=hamming(fft_siz,'periodic')';

%error score arrays setup
err_decr_percs=zeros(length(Lv),length(interf_rms_v),...
                        length(num_bands_v),length(fg_v));
ERR_DECR_PERCs=err_decr_percs;
```

```
%processing time array setup
tm_v=zeros(length(Lv),length(interf_rms_v),...
           length(num_bands_v),length(fg_v),2);


%permute all parameter choices; do additional variable setup within loops
for hifi=0:1
    hifi
    %assign value to string to be included in output file names
    if hifi
       hifstr='hifi';
    else
       hifstr='';
    end

%generate band_pass filter; also return vector which contains bin numbers
%corresponding to frequencies which span the chosen bandwidth;
[BPF,SPECTR_WHOLE]=gen_BPF(Fs,fft_siz,hifi,sig_len,num_mics);
DIRT_ERR_blocks=zeros(1,length(SPECTR_WHOLE));
CLEAN_blocks=DIRT_ERR_blocks;


for L_ind=1:length(Lv)
    L=Lv(L_ind)
    clean0=delay_src(clean_mono,num_mics,Fs,L);

    %compute normalized version of clean signal for error score purposes
    for mic_num=1:num_mics
     clean0(mic_num,:)=clean0(mic_num,:)./sqrt(mean(clean0(mic_num,:).^2));
    end
    clean_ref=[inpad clean0(4,:) inpad];
    cleanfile=[name 'clean_L-' num2str(L) '.wav'];
    clean_norm=clean_ref/max(abs(clean_ref))*.95;


for interf_rms_ind=1:length(interf_rms_v)
    interf_rms=interf_rms_v(interf_rms_ind);

    %generate interference which is properly delayed across mics
    dirty=gen_delayed_interf(num_mics,L,sig_len,interf_rms,BPF,hifi);

    %combine clean signal with interference
    dirty=clean0+dirty;

    %compute normalized noisy signal for error score purposes
    dirty_ref=[inpad dirty(4,:) inpad];
    sqrt(mean(dirty_ref.^2))
    dirty_norm=dirty_ref./max(abs(dirty_ref))*.95;

    %save noisy signal to file
    dirtfile=[name 'dirty_L-' num2str(L) '_rms-' ...
              num2str(interf_rms) hifstr '.wav'];
    wavwrite(dirty_norm,Fs,24,[path dirtfile]);



for num_bands_ind=1:length(num_bands_v)
    num_bands=num_bands_v(num_bands_ind)*(hifi+1)

for fg_ind=1:length(fg_v)

    fg=fg_v(fg_ind)
```

```
    %run MVDR algorithm with chosen parameters;
    [output,tm]=MVDR_speech(dirty,Fs,L,num_bands,fg,hifi);
    tm_v(L_ind,interf_rms_ind,num_bands_ind,fg_ind,hifi+1)=tm;
    output_norm=output/max(abs(output))*.95;

    %save output file
    outfile=[name 'out_L-' num2str(L) '_rms-' num2str(interf_rms) ...
    '_bands-' num2str(num_bands) '_fg-' num2str(fg) hifstr '.wav'];
    wavwrite(output_norm,Fs,24,[path outfile]);

    %pad output signal to compare to clean and noisy signals
    %to compute error score
    output=[inpad output inpad];

    %time-domain error score computed
    proc_err_rms=sqrt(mean( (output-clean_ref).^2  ) );
    dirt_err_rms=sqrt(mean( (dirty_ref-clean_ref).^2 ) );
    err_decr_percs(L_ind,interf_rms_ind,num_bands_ind,fg_ind,hifi+1)=...
        (dirt_err_rms-proc_err_rms)/dirt_err_rms*100;

    %frequency-domain error score computed
    DIRT_ERR_RMS_blocks=zeros(num_blocks,1);
    PROC_ERR_RMS_blocks=zeros(num_blocks,1);

    for block=1:num_blocks

       CLEAN_block=abs(fft(...
        clean_ref(shift_siz*(block-1)+1:shift_siz*(block-1)+fft_siz)...
        .*ham_block,100*fft_siz));

       DIRT_block=abs(fft(...
       dirty_ref(shift_siz*(block-1)+1:shift_siz*(block-1)+fft_siz)...
        .*ham_block,100*fft_siz));

       DIRT_ERR_RMS_blocks(block)=...
       sqrt(mean((CLEAN_block(SPECTR_WHOLE)-DIRT_block(SPECTR_WHOLE)).^2));

       PROC_block=abs(fft(...
         output(shift_siz*(block-1)+1:shift_siz*(block-1)+fft_siz)...
        .*ham_block,100*fft_siz));

       PROC_ERR_RMS_blocks(block)=...
       sqrt(mean((CLEAN_block(SPECTR_WHOLE)-PROC_block(SPECTR_WHOLE)).^2));
    end

    ERR_DECR_PERCs(L_ind,interf_rms_ind,num_bands_ind,fg_ind,hifi+1)=...
           mean((DIRT_ERR_RMS_blocks-PROC_ERR_RMS_blocks)./...
            DIRT_ERR_RMS_blocks*100);
end
end
end
end
end

%%%%%%%%%% END OF FILE %%%%%%%%%%
```

gen_BPF.m

```
%Function which generates band_pass filter;
%this function is used in MVDR_speech_test.m

%By Sergei Preobrazhensky

%Inputs:

%Fs - sampling rate

%fft_siz - fft_size used in MVDR algorithm in preperation for which this
%function is called; this value is used for generating the output vector
%SPECTR_WHOLE

%hifi - =0 to use 300-3400 Hz bandpass filter, nonzero to use 150-6350 Hz

%sig_len - length of signal to be filtered (and later MVDR processed)

%num_mics - the number of microphones used for the current MVDR test


%Outputs:

%BPF - bandpass filter vector which is roughly a twentieth
%of the signal length;

%SPECTR_WHOLE - vector which contains bin numbers
%corresponding to frequencies which span the chosen bandwidth;

function [BPF,SPECTR_WHOLE]=gen_BPF(Fs,fft_siz,hifi,sig_len,num_mics)

%if Fs==44100
%    fbin_btL=7+1;%bin~300 Hz
%    fbin_tpL=78+1;%~3400 Hz
if Fs==22050

    if ~hifi
        fbin_btL=14+1;%~300 Hz
        fbin_tpL=157+1;%~3400 Hz
    else
        fbin_btL=7+1;
        fbin_tpL=294+1;
    end

%elseif Fs==7350
%    fbin_btL=42+1;%~300 Hz
%    fbin_tpL=473+1;%~3400 Hz
else
    error('only Fs=22050 supported')
end

%compute actual frequency values which represent the bottom and the top bin
f_bt=fbin_btL/fft_siz*Fs;%~=300Hz
f_tp=fbin_tpL/fft_siz*Fs;%~=3400Hz
F=...
[0 (f_bt+10)/(Fs/2) (f_bt+10)/(Fs/2) (f_tp-10)/(Fs/2) (f_tp-10)/(Fs/2) 1];
A=[0        0           1               1               0             0];
BPF=reprow( fft(...
            [firls(ceil(sig_len/20)+mod(ceil(sig_len/20),2),F,A)...
    zeros(1,sig_len-1)]), num_mics);

FBIN_btL=round(f_bt/Fs*fft_siz*100+1);
```

```
FBIN_tpL=round(f_tp/Fs*fft_siz*100+1);
FBIN_tpR=fft_siz*100-FBIN_btL+2;
FBIN_btR=fft_siz*100-FBIN_tpL+2;

SPECTR_WHOLE=[FBIN_btL:FBIN_tpL FBIN_tpR:FBIN_btR];

%%%%%%%%%%% END OF FILE %%%%%%%%%%%%%%%%
```

gen delayed_interf.m

```
%function which returns profiles of interference simulated as if several
%interference sources originate at various random locations;
%the locations of origin are apparent from the delays that the interference
%exhibits across a microphone array.
%One delayed interference profile is generated for each array length
%passed as input;

%By Sergei Preobrazhensky

%INPUTS:

%num_mics: number of microphones in the arrays (only one scalar value can
%be used per function call)

%Lv: vector of array lengths to simulate

%sig_len: time length of each interference profile returned

%itnerf_rms: the RMS value to which the interference amplitude will be
%approximately scaled; used for adjusting signal to interference ratio

%BPF: bandpass filter to apply to interference and thus restrict
%processed bandwidth (tip: use the BPF returned by gen_BPF.m)

%hifi: 0 if bandwidth is 300-3400; otherwise 150-6350 Hz bandwidth assumed


%OUTPUTS:

%interf_delayed - 3-d array: num_mics rows, sig_len cols, length(Lv) pages;
%each page represents a delayed interference profile for each array length
%in given in Lv input.

function [interf_delayed]=...
    gen_delayed_interf(num_mics,Lv,sig_len,interf_rms,BPF,hifi)

c=339;%approx. speed of sound at altitude of Columbus, OH

%GENERATE INTERFERENCE:

%zero matrix to pad sections of interference to accomodate time delays
sect_pad=zeros(num_mics,size(BPF,2)-sig_len);
num_perm=2;%number of interferers active throughout whole profile
num_temp=5;%number of interferers active during only part of profile
num_interf=num_perm+num_temp;
rms_coef=1.2;%used to rescale rms after filtering off some energy;

%reset random stream to generate the same interference profile each time
%function is called (to maintain consistent interference profile
%while testing different parameters in routine that calls on this function)
stream = RandStream.getGlobalStream;
reset(stream);

%full address of interference data bank (crowd noise wave file)
filename=[...
      'C:\Users\Serge\Documents\z_college\AAiBrutus\MATLAB\Audio_Files',...
          '\Ohio_Stadium_noise24.wav'];

%the code below calls on function that randomly returns variables to use
%for building profiles of random interference throughout the rest of this
%function
```

```
%interf and interflocs_cel contain randomly generated interference samples
%and spatial locations respectively; strtpos contains starting time
%positions of short-time interference sources;
%the rest of the outputs of gener_interf are used for delaying in time;
[interf,fft_fctrs,fft_fctr,interflocs_cel,strtpos,k,shift_smp] =...
       gener_interf(filename,22050,num_perm,num_temp,1,sig_len);


%DELAY INTERFERENCE:

%All mic arrays simulated have num_mics mics uniformly spaced along x axis,
%centered at origin; they vary only in length and thus the mic spacing).

%preallocate output array of interference profiles
interf_delayed=zeros(num_mics,sig_len,length(Lv));

for L_ind=1:length(Lv)
    L=Lv(L_ind);
    miclocs=linspace(-L/2,L/2,num_mics)';

    interflocs=interflocs_cel{1};
    interf_delays=zeros(num_mics,num_interf);

    xmicinterf=zeros(num_mics,num_interf);
    for interf_ind=1:num_interf
        for mic_num=1:num_mics
            xmicinterf(mic_num,interf_ind)=...
            interflocs(1,interf_ind)-miclocs(mic_num);
            %calculate horiz dist from each source and interferer to
            %current mic in the current mic array
        end
    end

    ymicinterf=reprow(interflocs(2,:),num_mics);
    %y coords of mic locations are all 0 so y_interf_i-y_mic_n=y_interf_i

    interf_delays(:,:,L_ind)=sqrt(xmicinterf.^2+ymicinterf.^2)/c;
    input_sect=zeros(num_mics,k);

    %delay and condition interference sources active throughout whole
    %profile
    for interf_ind=1:num_perm

        INTERF0=fft( interf{interf_ind});
        %balance frequency spectrum for each interference source
        INTERF0=INTERF0.*...
                (9*(hifi+1)*mean(abs(INTERF0))+abs(INTERF0))/...
                (9*(hifi+1)+1) ./ abs(INTERF0);

        %normalize not to make some interference sources
        %disproportionately loud or quiet after the balancing
        INTERF0=INTERF0./max(abs(INTERF0));

        %preallocate copy 1-channel interference source
        %to form num_mics channels
        INTERF_MICS=reprow(INTERF0,num_mics);

        %apply delays in frequency domain
        interf_mics=real(ifft(INTERF_MICS.*...
        exp(interf_delays(:,interf_ind)*...
         fft_fctr),[],2));

        %add interference source to profile:
```

```
        input_sect=input_sect+interf_mics;
    end

    %delay and condition interference sources active during only parts of a
    %profile
    for interf_ind=interf_ind+1:num_temp
        el=length(interf{interf_ind});
        strt=strtpos(interf_ind-num_perm);

        INTERF0=fft(interf{interf_ind});
        %balance frequency spectrum for each interference source
        INTERF0=INTERF0.*...
                (9*(hifi+1)*mean(abs(INTERF0))+abs(INTERF0))/...
                (9*(hifi+1)+1)    ./ abs(INTERF0);
        %normalize not to make some interference sources
        %disproportionately loud or quiet after the balancing
        INTERF0=INTERF0/max(abs(INTERF0));

        %apply delays in frequency domain
        INTERF_MICS=reprow(INTERF0,num_mics);
        interf_mics=real(ifft(INTERF_MICS.*...
            exp(interf_delays(:,interf_ind)*...
             fft_fctrs{interf_ind-num_perm}),[],2));

        %add interference source to profile:
        input_sect(:,strt:strt+el-1)=...
            input_sect(:,strt:strt+el-1)+interf_mics;

    end
        %adjust RMS before filtering

        %the strength of the interference is controlled by interf_rms,
        %but a wider bandwidth will generally have a larger RMS
        %so that interference energy isn't spread thinner over
        %the larger number of sub-bands
        for mic_num=1:num_mics
            input_sect(mic_num,:)=input_sect(mic_num,:)*...
            rms_coef*interf_rms/sqrt(mean(input_sect(mic_num,:).^2));
        end

        %skip the silence (due to time delays) at beginning of profile
        input_sect=[input_sect(:,shift_smp:shift_smp+sig_len-1), sect_pad];
        %bandpass filter each profile
        input_sect=real(ifft(fft(input_sect,[],2).*BPF,[],2));
        input_sect=input_sect(:,size(input_sect,2)/2-sig_len/2+1:...
                        size(input_sect,2)/2+sig_len/2);
        interf_delayed(:,:,L_ind)=input_sect;

end
%%%%%%%%%%%% END OF FILE %%%%%%%%%%%%%%%%
```

gener_interf.m

```
%generates interference by randomly drawing chunks from crowd noise file
%(or any given audio file) for simulating interference recorded into
%a mic array

%NOTE: this file is used by the speech signal simulation test
%such as MVDR_speech_test.m via the gen_delayed_interf function

%By Sergei Preobrazhensky

%Inputs:

%filename - full path of audio data file from which to draw interfernce

%Fs - sampling frequency (should match that of file preferably)

%num_perm - number of interference sources which are active throughout
%entire time length of one trial
%(one trial a single profile of randomly generated interference
%see 'random_trials' input)

%num_temp - number of short term interference sources. These are active
%for only a fraction of the trial time length (specified by Lmin, Lmax
%below)

%rand_trials,sig_len - rand_trials # of interference profiles,
%each sig_len time samples long will be generated. Each of these profiles
%will contain the num_perm and num_temp type interferers.

%NOTE: interference profiles are not delayed across a microphone array
%in this function. Only random audio data is drawn, as well as random
%time positions, durations, and random spatial coordinates are chosen.
%An outside routine must take care of delaying these interference sources
%depending on microphone locations chosen

%Outputs:

%interf - cell_array of all the interference data drawn from audio file

%fft_fctrs - cell array of vectors used for delaying 'num_temp'-type
%interferers in frequency domain (must be done by outside routine).

%fft_fctr - vector used for delaying any 'num_perm'-type
%interferer in frequency domain (must be done by outside routine).

%interflocs_cel - cell array of
%randomly generated spatial location of all the interferers;
%see 'space and physical constraints' section of code below.

%strtpos - array of starting position (in time samples)
%of each 'num_temp'-type interferer in each trial;
%used by outside routine to place these interferers at the given positions

%k - maximum possible time length of one interference profile after delays;
%this is based on the maximum distance between a microphone and an
%interferer; the maximum mic_array length is assumed to be 1 m;
%the maximum distance inteference can be generated is defined in
%'space and physical constraints' section of code below.
%k = sig_len + maxsmplsdelay;
%this value is used by an outside routine for delaying purposes
```

```
%shift_smp - since the all interference sources will be delayed there will
%by default be silence at the beginning of an interfrence profile;
%to avoid this silence, the profile will start at sample # shift_smp
%of length k signal and will continue for sig_len samples;
%this effectively produces a window of interference
%which is length sig_len and has little to no silence at the beginning;
%this operation is done by an outside routine.

function [interf,fft_fctrs,fft_fctr,interflocs_cel,strtpos,k,shift_smp]=...
            gener_interf(filename,Fs,num_perm,num_temp,rand_trials,sig_len)
%hold on
%rng('shuffle')
num_interf=num_perm+num_temp;

%reading interference wav file
data = wavread(filename)';
LENG = length(data);

%signal sample constraints
Lmin=round(8/48*sig_len);
Lmax=round(24/48*sig_len);
%e.g. for a 2400 sample signal, temp interferer length will be chosen
%randomly to be between 50 and 400 samples

%signal amplitude constraints
%amplLo=.25;
%amplHi=2.5;
ampl=1;%^no longer used...

%space and physical constraints
c=339;
rmin=1.5;
rmax=15;
thetamin=-90;
thetamax=90;

%compute zero-padding for delay
distmax=rmax+.5;
maxsmplsdelay=ceil((distmax./c)*Fs);
maxsmplsdelay=maxsmplsdelay+mod(maxsmplsdelay,2);
pad=zeros(1,maxsmplsdelay);
shift_smp=round(maxsmplsdelay*.7);
k=sig_len+maxsmplsdelay;

%preallocating
interf=cell(num_interf,rand_trials);
strtpos = zeros(num_temp,rand_trials);
fft_fctrs=cell(num_temp,rand_trials);
interflocs_cel=cell(1,rand_trials);

%preparing additional auxilary vectors
fft_fctr=-2i*pi*[0:k/2 -k/2+1:-1]/k*Fs;

%populating interference array
for trial=1:rand_trials

    %random spatial locations
    interflocs = [rand(1,num_interf)*(rmax-rmin)+rmin;
            rand(1,num_interf)*(thetamax-thetamin)+thetamin];
    interflocs_cel{trial} = [interflocs(1,:).*sind(interflocs(2,:))
                             interflocs(1,:).*cosd(interflocs(2,:))];
    %Easier to generate interf in polar coords to have all interference be
    %at least 1.5 m away from mic array center and more sparse as the distance
```

```
    %increases. Then interference locations are converted to cartesian coords
    %sind x=r*sind(theta) as theta measured from y-axis

    for i=1:num_perm

        %ampl=(rand*(amplHi-amplLo)+amplLo);

        pos=randi(LENG-sig_len+1);
        chunk=data(pos:pos+sig_len-1);

        nonstat=ampl*chunk;
        white=0;
        interf{i,trial}=[nonstat+white pad];
    end
    %lin='-:+o*.xsd^v><ph';
    %cel={};
    for i=i+1:i+num_temp

        leng=randi([Lmin,Lmax]);
        leng=leng+mod(leng,2);
        padded_leng=leng+maxsmplsdelay;
        %ampl=(rand*(amplHi-amplLo)+amplLo);

        pos=randi(LENG-leng+1);
        chunk=data(pos:pos+leng-1);

        nonstat=ampl*chunk;
        white=0;
        interf{i,trial}=[nonstat+white pad];

        strtmin=round(maxsmplsdelay*2/3);
        strtmax=sig_len-leng+1;
        strt=randi([strtmin,strtmax]);
        strtpos(i-num_perm,trial)=strt;

        fft_fctrs{i-num_perm,trial}=...
         -2i*pi*[0:(padded_leng)/2,-(padded_leng)/2+1:-1]/(padded_leng)*Fs;

        %plot(strt:strt+leng-1+length(pad),interf{i,trial},...
        %'color',rand(1,3),'linestyle',lin(i-num_perm))
        %cel{end+1}=[num2str(i-num_perm) ' ' num2str(leng)];
    end
    %legend(cel);
end
end

%%%%%%%%% END OF FILE %%%%%%%%
```

MVDR_speech.m

```
%this function is an MVDR beamforming algorithm
%which can be to process for an n-channel signal;
%it was designed for use with processing a noisy speech signal
%recorded into a microphone array to aid with speech recognition

%By Sergei Preobrazhensky

%NOTE:
%Here it has been assumed that a desired source (e.g. human speaker) is
%1 m from the array center on a line which is perpendicular to the array
%and intersects the array at its center.
%See 'STEERING VECTOR SETUP' section of code. The steering vector can
%be modified as needed, if the desired source is at a different location.

%Inputs:

%dirty - N-channel matrix of audio input; M columns => M time samples

%Fs - sampling rate in Hz (only 22050 supported currently)

%L - length of uniformly spaced, 1-dim, mic array

%num_bands - number of sub-bands to use;
%make sure that the number of DFT bins spanning the bandwidth used
%(given Fs=22050 and a 1024-FFT) is divisible by this number;
%this will make each sub-band an equal integer number of bins
%e.g.: 300-3400 Hz => 144 DFT bins, so num_bands can be 24,16,12, etc.
%(resulting in corresponding sub-band sizes of 6,9,and 12 bins respectiv.)

%fg - forgetting factor value for memory element of adaptive part of
%algorithm 0<fg<1; optimal value must be determined experimentally;
%usually values of 0.9 or more perform better.

%hifi - hifi==0 => bandpass filter of 300-3400 is used for processing;
%effectively reducing the signal to this bandwidth;
%otherwise bandwidth of 150-6350Hz is used

%Outputs:

%output - 1xM vector of processed signal, M is input signal time length

%tm - MATLAB processing time of algorithm (setup time not included)

function[output,tm]=MVDR_speech(dirty,Fs,L,num_bands,fg,hifi)

%close all

%NOTE:
%all lengths/space coordinates in meters, frequencies in Hz, time in sec

for misc_prelim=1:1
%MISC PRELIMINARY SETUP:
%clc
c=339;%approx. speed of sound at altitude of Columbus, OH

num_mics=size(dirty,1);
sig_len=size(dirty,2);
half_num_bands=fix(num_bands/2);

fft_siz=1024;
block_siz=32;
```

```
num_blocks=(sig_len)/(block_siz)-1;
if mod(num_blocks,1)
    error('make sure a section of test signal fits integer num of blocks')
end

epsi=1e-13;
cndfctr=1.03;

if Fs==44100
    fbin_btL=7+1;%bin~300 Hz
    fbin_tpL=78+1;%~3400 Hz
elseif Fs==22050
    if hifi
    fbin_btL=7+1;%~300 Hz
    fbin_tpL=294+1;%~6300 Hz
    else
    fbin_btL=14+1;%~300 Hz
    fbin_tpL=157+1;%~6500 Hz
    end
elseif Fs==7350
    fbin_btL=42+1;%~300 Hz
    fbin_tpL=473+1;%~3400 Hz
else
    error('only Fs=7350, 22050, sand 44100 supported')
end
fbin_btR=fft_siz-(fbin_btL-1)+1;
f_bt=fbin_btL/fft_siz*Fs;%~=300Hz
f_tp=fbin_tpL/fft_siz*Fs;%~=3400Hz

j = sqrt(-1);
pi2=2*pi;
j2pi = j*pi2;
end


for steering_vect=1:1
%STEERING VECTOR SETUP:

%for each mic array: precalculate time delays from the desired source
%to each mic in the array.

%All mic arrays here have num_mics mics uniformly spaced along x axis,
%centered at origin; they vary only in length and thus the mic spacing).

%Desired Source is assumed to be at coordinates: (0,1)
%assume mics are unifromly spaced
miclocs=linspace(-L/2,L/2,num_mics);
%assume human speaker coordinates of (0,1)
xmicsrc=miclocs;
ymicsrc=ones(1,num_mics);

src_delays=sqrt(xmicsrc.^2+ymicsrc.^2)'/c;

wop_L=zeros(num_mics,num_bands);

band_siz=(fbin_tpL-fbin_btL+1)/num_bands;
if mod(band_siz,1)
    error('make sure subband size comes out to be an integer')
end

%populate array of steering vectors
fv=zeros(1,num_bands);
for f_ind=1:num_bands
```

```
    %find center freqs of sub-band to later plug into the steering vectors
    %only need to compute the positive center freqs (left half of dft)
    fbin_L=((fbin_btL+(f_ind-1)*band_siz)+...
           (fbin_btL+(f_ind-1)*band_siz+band_siz-1))/2;
    fv(f_ind)=(fbin_L-1)/fft_siz*Fs;
end

%only compute steering vector for positive center freqs as steering vectors
%for the negative freqs are computed by taking the conjugate
vssL=exp((src_delays-min(src_delays))*-j2pi*fv);
%vssR=conj(vssL);
end


for other_misc=1:1
%OTHER MISC SETUP:

F=...
[0 (f_bt+20)/(Fs/2) (f_bt+30)/(Fs/2) (f_tp-30)/(Fs/2) (f_tp-20)/(Fs/2) 1];
A=[0        0           1               1               0             0];
BPF=reprow( fft(...
            [firls(ceil(sig_len/40)+mod(ceil(sig_len/40),2),F,A)...
    zeros(1,sig_len-1)]), num_mics);
%fir2(sig_len,F,A) is length sig_len+1 by default
outpad=zeros(num_mics,size(BPF,2)-sig_len);
%hamming window will be applied to one time block at a time before fft
ham=reprow(hamming(fft_siz,'periodic'),num_mics);

%other prelim pre-allocation
dirtpad=zeros(num_mics,(fft_siz-block_siz)/2);

%conditioning matrix for MVDR
CND=(cndfctr-1)*diag(ones(1,num_mics))+ones(num_mics);

R_L=zeros(num_mics,num_mics,num_bands);

output=zeros(1,sig_len);
end



%perform MVDR
tic


dirty2=[dirtpad dirty dirtpad];

%{
%zero pad and perform fast convolution with band pass filter
dirty=real(ifft(fft([dirty outpad],[],2).*BPF,[],2));
        dirty=dirty(:,size(dirty,2)/2-sig_len/2:...
                        size(dirty,2)/2+1+sig_len/2);
%}



for block_num=1:num_blocks

    timepos=(block_siz*(block_num-1)+1);
    %if timepos+fft_siz>sig_len3
    %--remove secur. measure for speed
    %   timeblock=inputs(:,timepos:end);
    %else
```

```
dirty_block=dirty2(:,timepos:timepos+fft_siz-1).*ham;
%end

%convert interference block to frequency domain
DIRTY_BLOCK=fft(dirty_block,[],2);

PROCESSED=DIRTY_BLOCK(4,:);


for f_ind=1:length(fv)

    %work with one subband at a time;
    %define subband edges for pos. & neg. freq pairs:
    fbin_loL=fbin_btL+(f_ind-1)*band_siz;%left edge
    fbin_hiL=fbin_loL+band_siz-1;%right edge
    fbin_loR=fbin_btR-(f_ind-1)*band_siz;%-left edge
    fbin_hiR=fbin_loR-band_siz+1;%-right edge



    S_L=DIRTY_BLOCK(:,fbin_loL:fbin_hiL);
    S_R=DIRTY_BLOCK(:,fbin_loR:-1:fbin_hiR);
    vsL=vssL(:,f_ind);
    %vsR=vssR(:,f_ind);
    %{
    if f_ind==1
        EXTRA=0;
    elseif f_ind==2
        EXTRA=0;%.5*R_L(:,:,f_ind-1);
    else
        EXTRA=0;%.5*EXTRA;
    end
    %}
        R_L(:,:,f_ind)=R_L(:,:,f_ind)*fg+...
        +S_L*S_L'.*CND;

    %R_R(:,:,f_ind)=R_R(:,:,f_ind)*forgetfctr(f_ind)...
    %+S_R*S_R'.*CND;
    %covariance matrix and MVDR steering vectors are made
    %for both the pos. freqs (_L) and neg. freqs (_R)

    %MVDR weights computed
    if (mod(block_num,2) && f_ind<=half_num_bands)||...
       ((~mod(block_num,2)||block_num==1) && f_ind>half_num_bands)
    wop_L(:,f_ind)=...
        ( R_L(:,:,f_ind) )\vsL ./...
        (vsL'/ (R_L(:,:,f_ind) )*vsL+epsi);
    end


    %weights are applied to both the pos. and neg. freq
    %subband
    WS_L=wop_L(:,f_ind)'*S_L;
    WS_R=wop_L(:,f_ind).'*S_R;

    PROCESSED([fbin_loL:fbin_hiL, fbin_loR:-1:fbin_hiR]) = [WS_L WS_R];

end

    processed=real(ifft(PROCESSED));
    processed=processed(fft_siz/2-(block_siz)/2+1:...
                        fft_siz/2+(block_siz)/2);
```

```
        output(timepos:timepos+(block_siz)-1)=processed;

end


%FINAL RESULTS:
%zero pad and perform fast convolution with band pass filter

output=real(ifft(fft([output outpad(1,:)]).*BPF(1,:)));
output=output(:,length(BPF)/2-sig_len/2+1:...
                        length(BPF)/2+sig_len/2);
output=output/max(abs(output))*.9;

tm=toc;

end
%END OF FILE
%================================================================================
```

reprow.m

```
%Faster repmat function strictly for repeating rows.
%Used by MVDR_speech.m and gen_delayed_interf.m
function M=reprow(v,rows)
M=zeros(rows,length(v));
for row=1:rows
    M(row,:)=v;
end
%%%%%%%% END OF FILE %%%%%%%%%%
```

fix_src.m

```
%decimate the source signal and cut it to be divisible into fft_siz
%blocks for testing purposes

function y=fix_src(x,shift_siz,Fs_old,Fs_new)
x=x(:,1:end-mod(length(x),shift_siz));
y=[];

for i=1:size(x,1)
    z=decimate(x(i,:),Fs_old/Fs_new,'FIR');
    z=z(1:end-mod(length(z),shift_siz));
    y=[y; z];
end
%y=.95*y./max(abs(y));

%%%%%%%%%%%%% END OF FILE %%%%%%%%%%%%%%%
```

A3.2 MATLAB Code for Second Test: Non-Speech Signal

MVDR_sp_sim_batch.m

```
%Main Script: MVDR simulation for non-speech signal
%under various parameters, March-May 2012

%By Sergei Preobrazhensky

%Required function files:
%MVDR_test_apr20.m, gener_desired_noise.m, gener_interf_old.m,

%Also required: Ohio_Stadium_noise24.wav crowd noise wave file

clear all
close all
clc

path='C:\iBrutus';
%path=input('input noise file path \n');
%if isequal(path,'')
%   path='C:\Users\Serge\Documents\z_college\AAiBrutus\MATLAB\Audio_Files';
%end

%lines below not used: parallel computing toolbox found to be ineffective:
%matlabpool close force
%matlabpool open local 2

%set up paramter vectors and other fixed inptus to MVDR_test_apr20 function
%see MVDR_test_apr20.m file for help
num_mics=8;
Lv=.55:.05:1;
rand_trials=50;
band_siz_v=[24 16 12 8 4 2 1];
Fs=22050;
sect_len=1024*6;
shift_siz=32;
fg_v=[.85 .9 .95, .99];
interf_rms_v=[.5 1 2];

%preallocate results data arrays
%originally these were made into cell arrays to be compatible with
%parallel computing toolbox.
%This toolbox is no longer used, but the slightly
%awkward format of an array within a cell array remains.
err_sub_cel=repmat({[]},[length(band_siz_v) length(Lv) length(fg_v)]);
err_decr_percs_cel=repmat({err_sub_cel},1,length(interf_rms_v));
ERR_DECR_PERCs_cel=err_decr_percs_cel;
update=0;%explained in MVDR_test_apr20
hifi=1;%test with 'hifi' bandwidth of .15-6.35 KHz rather than .3-3.4 KHz
i=0;%counter for saving data files


%the for loops below permute all parameter choices
for fg_ind=1:length(fg_v)

    fg=fg_v(fg_ind)

    for L_ind=1:length(Lv)
```

```
        L=Lv(L_ind)

    for band_siz_ind=1:length(band_siz_v)

        band_siz=band_siz_v(band_siz_ind)


        for interf_rms_ind=1:length(interf_rms_v);

            fprintf(['interf_rms_ind=' num2str(interf_rms_ind) '\n'])


            %reset MATLAB random number stream every time to
            %have consistent resutls across parameter combinations
            stream = RandStream.getGlobalStream;
            reset(stream);

            %run MVDR simulation test for one chosen set of parameters
            %(see function file MVDR_test_apr20.m for help)
            %MVDR_test_apr20 fucntion can handle a vector of parameters
            %for L, band_siz, and fg, but to avoid running out of memory,
            %only one choice is given to the function for each of these
            %parameters
            [err_decr_percs ERR_DECR_PERCs]=...
            MVDR_test_apr20(num_mics,L,rand_trials,...
                            band_siz,Fs,update,...
                            hifi,sect_len,shift_siz,...
                            interf_rms_v(interf_rms_ind),fg,path);

            err_decr_percs_cel...
                {interf_rms_ind}{band_siz_ind,L_ind,fg_ind}=...
                err_decr_percs;

            ERR_DECR_PERCs_cel...
                {interf_rms_ind}{band_siz_ind,L_ind,fg_ind}=...
                ERR_DECR_PERCs;

        end

        %incrementally save results as a percaution
        i=i+1;
        save(['C:\iBrutus\May29b' num2str(i) '.mat'],...
            'err_decr_percs_cel','ERR_DECR_PERCs_cel');

    end

    end

end
%line below not used: parallel computing toolbox found to be ineffective:
%matlabpool close

%%%%%% END OF FILE %%%%%%%%%%
```

MVDR_test_apr20.m

```
%Function for simulating MVDR performance with a virtual test signal
%Used for development for beamforming to human speech.
%This function calls on subroutines 'gener_interf_old' and
%'gener_desired_noise' to generated interference and desired signal
%by drawing chunks from a large wave file of crowd noise,
%'Ohio_Stadium_noise24.wav'
%Path of this file can be changed as needed below where 'filename' is set.
%These chunks are used to form interference with of relatively
%short time durations throughout the signal
%Multiple interferers at a time simulated at random spatial locations.
%All interference is balanced to contain even energy levels across
%the processed spectrum (e.g. 300-3400 Hz)

%---INPUTS---:

%num_mics:   must be at least 2%Lv:    is a vector of the uniform linear array lengths
tested

%rand_trials:    how many trials of interference to test.
    %e.g. rand_trials=50: a clean test signal will be processed 50 times
    %with a new randomly generated interfence profile added to it each time.

%band_siz_v:   a vector of multiple numbers of sub-bands to test

%Fs:    only use 22050 in this version

%update:
%--NOTE:   during March-May 2012 research, only update=0 has been used
%        if update=0:
    %MVDR weights updated only for lower freq. sub-bands for one block
    %before weights are applied to all sub-bands in that block.
    %Then, MVDR weights updated only for higher sub-bands in the next block
    %before weights are applied to all sub-bands in that block.
    %This pattern is then repeated.
%       otherwise:
    %all MVDR weights are updated for all sub-bands in every block.


%sect_len:      time length of a one trial e.g. a short 1024-sample signal.
%shift_siz:     how many samples each 1024-sample FFT is shifted for every
            %block of appying MVDR weights
%interf_rms:    each interference profile is adjusted to have this RMS
                %this allows testing different inteference strengths
                %the RMS of the clean signal is set to 1.
%fg_v:  a vector of different forgetting factors to test.
        %each factor >0, but <1.



%---OUTPUTS---:

%err_decr_percs:
    %time domain error score:
    %measures improvement in RMS of error in the time domain
    %after processing. 'Error' means deviation from clean signal.
    %100=> perfect recovery of clean signal.
    %0 or less=>MVDR made no improvement to error.

%ERR_DECR_PERCs:
    %Similar to time domain score but more complex.
    %measures improvements in RMS error in magnitude response
```

```
    %by transforming 1024-sample sections of processed and noisy signal
    %to freqeuncy and computing the average improvement in error
    %for the processed signal across DFT of the many 1024-sample sections.

%--NOTE: each of error score arrays above is 3 dimensional.
%Rows correspond to values from Lv,
%columns to values from band_siz_v,
%pages to values from fg_v.


%
function [err_decr_percs,ERR_DECR_PERCs]=...
         MVDR_test_apr20(num_mics,Lv,rand_trials,...
        band_siz_v,Fs,update,hifi,sect_len,shift_siz,interf_rms,fg_v,path)
%}

%NOTE:
%for debugging and development purposes, remove the '{' below and put '{'
%on the line with '%' directly above line 'function [...' to run this
%as a script rather than a function. This will be slower, but will store
%all variable in workspace (useful for debugging).

%{
clear all
num_mics=8;
Lv=[1];
rand_trials=2;
band_siz_v=[8];
Fs=22050;
sect_len=1024*6;
shift_siz=32;
interf_rms=2;
fg_v=[.97];
hifi=1;
update=1;
path='C:\iBrutus';
%}
%%
%NOTE:
%all lengths/space coordinates in meters, frequencies in Hz, time in sec

for misc_prelim=1:1
%MISC PRELIMINARY SETUP:
%clc
%rng('shuffle')

j = sqrt(-1);
pi2=2*pi;
j2pi = j*pi2;

c=339;%approx. speed of sound at altitude of Columbus, OH

%MVDR conditioning constants
epsi=1e-13;
cndfctr=1.03;
CND=ones(num_mics)+((cndfctr-1)*diag(ones(1,num_mics)));

%blocks and sections setup
fft_siz=1024;
cut=shift_siz*32;
sig_len=(sect_len-cut*2)*rand_trials;%all rand interf. trials put in 1 long signal
num_blocks=(sig_len)/(shift_siz);
if mod(num_blocks,1)
```

```
    error('make sure a section of test signal fits integer num of blocks')
end

inpad_len=(fft_siz-shift_siz)/2;
inpad=zeros(num_mics,inpad_len);
ham_sect=hamming(fft_siz,'periodic')';

%setup of frequency bins and frequencies in Hz

%Number of DFT bins which approx. represent the speech spectrum
%should be divible by 36 or 18 (to test with 18 or 36 sub-bands)

%Thus for each sampling frequency, particular bin numbers that satisfy
%above condition are chosen:
if hifi
    Flo=150;
    Fhi=6350;

    if Fs==22050
        fbin_btL=7;%~150 Hz
        fbin_tpL=294;%~6350 Hz
    else
        error('only Fs=22050 supported')
    end

else

    Flo=300;
    Fhi=3400;
    %if Fs==44100
    %    fbin_btL=7+1;%bin~300 Hz
    %    fbin_tpL=78+1;%~3400 Hz
    if Fs==22050
        fbin_btL=14+1;%~300 Hz
        fbin_tpL=157+1;%~3400 Hz
    %elseif Fs==7350
    %    fbin_btL=42+1;%~300 Hz
    %    fbin_tpL=473+1;%~3400 Hz
    else
        error('only Fs=22050 supported')
    end
end


fbin_btR=fft_siz-(fbin_btL-1)+1;%lowest DFT bin in bandwidth computed

%compute actual frequency values which represent the bottom and the top bin
f_bt=fbin_btL/fft_siz*Fs;%~=150Hz when hifi==1 or 300Hz when hifi==0
f_tp=fbin_tpL/fft_siz*Fs;%~=6350Hz when hifi==1 or 3400Hz when hifi==0;

%for corresponding bin numbers for a sect_len-point fft used for measuring
%frequency domain error for each output
FBIN_btL=round(f_bt/Fs*fft_siz*100+1);
FBIN_tpL=round(f_tp/Fs*fft_siz*100+1);
FBIN_tpR=fft_siz*100-FBIN_btL+2;
FBIN_btR=fft_siz*100-FBIN_tpL+2;
WHOLE_SPECTR100=[FBIN_btL:FBIN_tpL  FBIN_tpR:FBIN_btR];

%Bandpass filter Setup
filt_ord=1024;
filt_len=filt_ord+1;
Fs2=Fs/2;
F=[0 (f_bt+20)/(Fs2) (f_bt+30)/(Fs2) (f_tp-30)/(Fs2) (f_tp-20)/(Fs2) 1];
```

```
A=[0        0           1                1               0             0];
BPF= firls(filt_ord,F,A);
BPF_interf= repmat(fft([BPF zeros(1, sig_len-1)] ),num_mics,1);
BPF_desir= repmat(fft([BPF zeros(1, sig_len-1+inpad_len)] ),num_mics,1);
%firls(sig_len,F,A) returns a length sig_len+1 filter unit pulse repsonse
%even order => odd length filter


%balancing factor setup for interference and desired signal
balance_factor=9*(hifi+1);
end


for generate_interf=1:1
%GENERATE INTERFERENCE
num_perm=2;
num_temp=5;
num_interf=num_perm+num_temp;

filename=[path '\Ohio_Stadium_noise24.wav'];
[interf,fft_fctrs,fft_fctr,interflocs_cel,strtpos,k,shift_smp] =...
   gener_interf_old(filename,22050,num_perm,num_temp,rand_trials,sect_len);

end


for interf_signals=1:1
%INTERFERENCE SIGNALS SETUP:

%for each mic array: precalculate time delays from the
%interferers to each mic in the array.

%All mic arrays here have num_mics mics uniformly spaced along x axis,
%centered at origin; they vary only in length and thus the mic spacing).
input=zeros(num_mics,sig_len);
inputs=zeros(num_mics,sig_len+inpad_len*2,length(Lv));


for L_ind=1:length(Lv)
    L=Lv(L_ind);
    miclocs=linspace(-L/2,L/2,num_mics)';

for sect=1:rand_trials
    interflocs=interflocs_cel{sect};
    interf_delays=zeros(num_mics,num_interf,length(Lv),rand_trials);

    xmicinterf=zeros(num_mics,num_interf);
    for interf_ind=1:num_interf
        for mic_num=1:num_mics
            xmicinterf(mic_num,interf_ind)=...
            interflocs(1,interf_ind)-miclocs(mic_num);
            %calculate horiz dist from each source and interferer to
            %current mic in the current mic array
        end
    end

    ymicinterf=repmat(interflocs(2,:),num_mics,1);
    %y coords of mic locations are all 0 so y_interf_i-y_mic_n=y_interf_i

    interf_delays(:,:,L_ind,sect)=sqrt(xmicinterf.^2+ymicinterf.^2)/c;
    input_sect=zeros(num_mics,k);

%delay and condition longer interference sources (they are 1 section long)
```

```
    for interf_ind=1:num_perm
        INTERF0=fft( interf{interf_ind});
        fft_len=length(INTERF0);
        spectr_bins=[round(Flo/Fs*fft_len)+1:round(Fhi/Fs*fft_len)+1 ...
         fft_len-round(Fhi/Fs*fft_len)+2: fft_len-round(Flo/Fs*fft_len)+2];
        INTERF_spectr_mag=abs(INTERF0(spectr_bins));
        %balance frequency spectrum for each interference source
        INTERF0(spectr_bins)=INTERF0(spectr_bins).*...
            (balance_factor*mean(INTERF_spectr_mag)+INTERF_spectr_mag)/...
            (balance_factor+1)./INTERF_spectr_mag;
        %normalize not to make some interference sources
        %disproportionately loud or quiet after the balancing
        INTERF0=INTERF0./max(abs(INTERF0));

        INTERF_MICS=repmat(INTERF0,num_mics,1);

        interf_mics=real(ifft(INTERF_MICS.*...
        exp(interf_delays(:,interf_ind,L_ind,sect)*...
         fft_fctr),[],2));

        input_sect=input_sect+interf_mics;

    end

    %delay and condition shorter interference sources (length <1 section)
    for interf_ind=interf_ind+1:num_temp
        el=length(interf{interf_ind});
        strt=strtpos(interf_ind-num_perm);

        INTERF0=fft( interf{interf_ind});
        fft_len=length(INTERF0);
        spectr_bins=[round(Flo/Fs*fft_len)+1:round(Fhi/Fs*fft_len)+1 ...
         fft_len-round(Fhi/Fs*fft_len)+2: fft_len-round(Flo/Fs*fft_len)+2];
        INTERF_spectr_mag=abs(INTERF0(spectr_bins));
        %balance frequency spectrum for each interference source
        INTERF0(spectr_bins)=INTERF0(spectr_bins).*...
                (9*mean(INTERF_spectr_mag)+INTERF_spectr_mag)/10./...
                INTERF_spectr_mag;
        %normalize not to make some interference sources
        %disproportionately loud or quiet after the balancing
        INTERF0=INTERF0/max(abs(INTERF0));

        INTERF_MICS=repmat(INTERF0,num_mics,1);
        interf_mics=real(ifft(INTERF_MICS.*...
            exp(interf_delays(:,interf_ind,L_ind,sect)*...
             fft_fctrs{interf_ind-num_perm}),[],2));

        input_sect(:,strt:strt+el-1)=...
            input_sect(:,strt:strt+el-1)+interf_mics;
    end

        input_sect=input_sect(:,shift_smp+cut:shift_smp+sect_len-cut-1);
        input(:,(sect_len-2*cut)*(sect-1)+1:(sect_len-2*cut)*sect)=...
            input_sect;


end
    %adjust interference RMS to value specified by input

    for mic_num=1:num_mics
        input(mic_num,:)=input(mic_num,:)*...
                            interf_rms/sqrt(mean(input(mic_num,:).^2));
    end
```

```
    input=real(ifft(...
        fft([input zeros(num_mics,filt_len-1)],[],2 ).*BPF_interf,[],2));
    input=input(:,length(input)/2-sig_len/2+1:length(input)/2+sig_len/2);

    %pad interf. so MVDR algorithm can output the first 32
    %samples of interference+clean as the first block.
    inputs(:,:,L_ind)=[inpad input inpad];

end


end


for clean_signal=1:1
%GENERATING CLEAN (OR DESIRED) SIGNAL

%the desired test signal will be added to the interference
%with appropriate delays.

%clean signal will be simulated as if recorded from a source
%at (0,1) into at a mic at (0,0) to compare to MVDR-processed noisy signal
src_delays=zeros(num_mics,length(Lv));
clean_refs=zeros(num_mics,2*inpad_len+sig_len,length(Lv));
clean_ref=zeros(num_mics,sig_len+inpad_len);

%clean signal will actually be randomly drawn from the same crowd noise
%wave file as interference
[desired_noise, shift_smp, des_fft_fctr]=...
          gener_desired_noise(filename,Fs,rand_trials,sect_len);

for L_ind=1:length(Lv)

    for sect=1:rand_trials
    clean_bloc=desired_noise{sect};
    CLEAN_BLOC=fft(clean_bloc);
    fft_len=length(CLEAN_BLOC);
        spectr_bins=[round(Flo/Fs*fft_len)+1:round(Fhi/Fs*fft_len)+1 ...
         fft_len-round(Fhi/Fs*fft_len)+2: fft_len-round(Flo/Fs*fft_len)+2];
    CLEAN_BLOC_spectr_mag=abs(CLEAN_BLOC(spectr_bins));
    %just like with interf, balance freqs of clean signal across bandwidth
    CLEAN_BLOC(spectr_bins)=CLEAN_BLOC(spectr_bins).*...
     (balance_factor*mean(CLEAN_BLOC_spectr_mag)+CLEAN_BLOC_spectr_mag)/...
     (balance_factor)./CLEAN_BLOC_spectr_mag;
    CLEAN_NODELAY_MICS=repmat(CLEAN_BLOC,num_mics,1);

    L=Lv(L_ind);

    miclocs=linspace(-L/2,L/2,num_mics)';
    xmicsrc=zeros(1,num_mics);

    for mic_num=1:num_mics
        xmicsrc(mic_num)=0-miclocs(mic_num);
    end

    ymicsrc=ones(1,num_mics);

    src_delays(:,L_ind)=sqrt(xmicsrc.^2+ymicsrc.^2)/c;

    CLEAN_DELAY_MICS=CLEAN_NODELAY_MICS.*...
                        exp(src_delays(:,L_ind)*des_fft_fctr);
```

```
        clean_ref_block=real(ifft(CLEAN_DELAY_MICS,[],2));
        clean_ref_block=clean_ref_block(:,cut+1:end-cut);
        timepos=((sect_len-2*cut)*(sect-1)+1);
        clean_ref(:,timepos:timepos+(sect_len+shift_smp-2*cut)-1)=...
                                        clean_ref_block;
    end

    clean_ref=real(ifft(...
    fft([clean_ref zeros(num_mics,filt_len-1)],[],2).*BPF_desir,[],2));
    clean_ref=clean_ref(:,length(clean_ref)/2-(sig_len+inpad_len)/2+1:...
                        length(clean_ref)/2+(sig_len+inpad_len)/2);
    clean_ref=clean_ref./...
        repmat(sqrt(mean(clean_ref.^2,2)),1,size(clean_ref,2));
    clean_refs(:,:,L_ind)=[inpad clean_ref];

end

end


for steering_vector=1:1
%STEERING VECTOR SETUP:

vs_cel=cell(length(Lv),length(band_siz_v));

for band_siz_ind=1:length(band_siz_v)
    band_siz=band_siz_v(band_siz_ind);
    num_bands=(fbin_tpL-fbin_btL+1)/band_siz;
    if mod(num_bands,1)
        error(['make sure number of bands',...
               num2str(band_siz_ind) 'comes out to be an integer'])
    end

    fv=zeros(1,num_bands);
    for f_ind=1:num_bands
        fbin_L=((fbin_btL+(f_ind-1)*band_siz)*2+band_siz-1)/2;
        fv(f_ind)=(fbin_L-1)/fft_siz*Fs;
    end

    j2pif=j2pi*fv;

    %preallocate array of steering vectors
    vs_cel{L_ind,band_siz_ind}=zeros(num_mics,num_bands);

    for L_ind=1:length(Lv)
        vs_cel{L_ind,band_siz_ind}=...
        exp(  (src_delays(:,L_ind)-min(src_delays(:,L_ind)))  *-j2pif);
    end

end

end


for results_arrays=1:1
%OUTPUT (and auxiliary) ARRAYS PRELLACOTION
output=zeros(1,sig_len);

DIRT_ERR_RMS_blocks=zeros(num_blocks,1);
PROC_ERR_RMS_blocks=zeros(num_blocks,1);
err_decr_percs=zeros(length(Lv),length(band_siz_v),length(fg_v));
ERR_DECR_PERCs=err_decr_percs;
```

```
end

%%
%TEST ALL MIC ARRAYS with MVDR
for L_ind = 1:length(Lv)
    L_ind;

    %look up properly delayed clean signal
    clean=clean_refs(:,:,L_ind);

    %various setup for time domain error scores
    clean_reff=clean(4,inpad_len+1:end-inpad_len);
    dirt_err=inputs(4, inpad_len+1:end-inpad_len, L_ind);
    dirt_err_rms=sqrt(mean(dirt_err.^2));

    %various setup for frequency magnitude error scores
    dirty_REF=inputs(4,:,L_ind)+clean(4,:);

    %try different number of sub-bands
    for band_siz_ind=1:length(band_siz_v)
    band_siz=band_siz_v(band_siz_ind);
    num_bands=(fbin_tpL-fbin_btL+1)/band_siz;
    half_num_bands=fix(num_bands/2);

        %try diffent forgetting factors
        for fg_ind=1:length(fg_v)
            fg=fg_v(fg_ind);
            R_L=zeros(num_mics,num_mics,num_bands);
            wop_L=zeros(num_mics,num_bands);

            %process signal with MVDR block by block
            for block_num=1:num_blocks

                %%
                timepos=(shift_siz*(block_num-1)+1);

                dirty_block=inputs(:,timepos:timepos+fft_siz-1,L_ind);
                clean_block=clean(:,timepos:timepos+fft_siz-1);

                %convert interference block to frequency domain
                DIRTY_BLOCK0=fft(dirty_block,[],2);
                CLEAN_BLOCK=fft(clean_block,[],2);

                %add interference to clean signal
                DIRTY_BLOCK=DIRTY_BLOCK0+CLEAN_BLOCK;


                PROCESSED=DIRTY_BLOCK(4,:);

                %calculate and apply MVDR weights one sub-band at a time
                for f_ind=1:num_bands
                %%
                    %define subband edges for pos. & neg. freq pairs:
                    fbin_loL=fbin_btL+(f_ind-1)*band_siz;%left edge
                    fbin_hiL=fbin_loL+band_siz-1;%right edge
                    fbin_loR=fbin_btR-(f_ind-1)*band_siz;%-left edge
                    fbin_hiR=fbin_loR-band_siz+1;%-right edge

                    S_L=DIRTY_BLOCK(:,fbin_loL:fbin_hiL);
                    S_R=DIRTY_BLOCK(:,fbin_loR:-1:fbin_hiR);
```

```
        vsL=vs_cel{L_ind,band_siz_ind}(:,f_ind);

        %covariance matrix: cov=(cov_old*forget_factor)+cov_new
        R_L(:,:,f_ind)=R_L(:,:,f_ind)*fg+...
                       S_L*S_L'.*CND;

        %If update is 0, new MVDR weights computed:
        %for only half the sub-bands in one block,
        %other half in next block, and so on.
        %Otherwise, new weights are computed for all bands in
        %every block.
        if (   (mod(block_num,2) && f_ind<=half_num_bands)||...
                ( (~mod(block_num,2)||block_num==1)&&...
                      f_ind>half_num_bands )              )...
            ||update

            wop_L(:,f_ind)=...
            (R_L(:,:,f_ind)\vsL) ./...
            (vsL'/R_L(:,:,f_ind)*vsL+epsi);
        end

        %weights are applied to both the pos. and neg. freq
        %subband
        WS_L=wop_L(:,f_ind)'*S_L;
        WS_R=wop_L(:,f_ind).'*S_R;

        PROCESSED([fbin_loL:fbin_hiL,...
                   fbin_loR:-1:fbin_hiR]) = [WS_L WS_R];

    end
    %{
    %to plot clean, noisy and processed blocks in frequency for
    %observation, remove '{' two lines above
    plot(abs(CLEAN_BLOCK(4,:)))
    title('clean')
    pause(1)
    plot(abs(DIRTY_BLOCK0(4,:)))
    title('interf')
    pause(1)
    plot(abs(DIRTY_BLOCK(4,:)))
    title('b4')
    pause(1)
    plot(abs(PROCESSED))
    pause(1)
    %}
    processed=real(ifft(PROCESSED));
    processed=processed(fft_siz/2-(shift_siz)/2+1:...
                        fft_siz/2+(shift_siz)/2);
    output(timepos:timepos+(shift_siz)-1)=processed;

end

%calculate time domain error score:
proc_err_rms=sqrt(mean((output-clean_reff).^2));
err_decr_percs(L_ind,band_siz_ind,fg_ind)=...
(dirt_err_rms-proc_err_rms)/dirt_err_rms*100;

%calculate frequency magnitude error score
%(as it was observed that there are similarities
%with time domain error score,
%this metric was not presented in May 2012 report):
proc_REF=[inpad(1,:) output inpad(1,:)];
```

```
            for block_num=1:num_blocks

                timepos=(shift_siz*(block_num-1)+1);

                CLEAN_block_ref=abs(fft(...
                    clean(4,timepos:timepos+fft_siz-1).* ham_sect...
                                  ,100*fft_siz));

                DIRT_block_ref=abs(fft(...
                 dirty_REF(timepos:timepos+fft_siz-1).* ham_sect...
                               ,100*fft_siz));

                DIRT_ERR_temp=...
                   CLEAN_block_ref(WHOLE_SPECTR100)-...
                   DIRT_block_ref(WHOLE_SPECTR100);

                DIRT_ERR_RMS_blocks(block_num)=...
                    sqrt(mean(DIRT_ERR_temp.^2));

                PROC_block_ref=abs(fft(...
                    proc_REF(timepos:timepos+fft_siz-1).* ham_sect...
                             ,100*fft_siz));

                PROC_ERR_temp=...
                    CLEAN_block_ref(WHOLE_SPECTR100)-...
                    PROC_block_ref(WHOLE_SPECTR100);

                PROC_ERR_RMS_blocks(block_num)=...
                    sqrt(mean(PROC_ERR_temp.^2));

            end

            ERR_DECR_PERCs(L_ind,band_siz_ind,fg_ind)=...
            mean((DIRT_ERR_RMS_blocks-PROC_ERR_RMS_blocks)./...
            DIRT_ERR_RMS_blocks*100);

        end
    end
end

%END OF FILE
%=========================================================================
```

gener_interf_old.m

```
%generates interference by randomly drawing chunks from crowd noise file
%(or any given audio file) for simulating interference recorded into
%a mic array

%NOTE: this file is used by non-speech signal simulation test
%such as MVDR_test_apr20.m

%By Sergei Preobrazhensky

%Inputs:

%filename - full path of audio data file from which to draw interfernce

%Fs - sampling frequency (should match that of file preferably)

%num_perm - number of interference sources which are active throughout
%entire time length of one trial
%(one trial a single profile of randomly generated interference
%see 'random_trials' input)

%num_temp - number of short term interference sources. These are active
%for only a fraction of the trial time length (specified by Lmin, Lmax
%below)

%rand_trials,sig_len - rand_trials # of interference profiles,
%each sig_len time samples long will be generated. Each of these profiles
%will contain the num_perm and num_temp type interferers.

%NOTE: interference profiles are not delayed across a microphone array
%in this function. Only random audio data is drawn, as well as random
%time positions, durations, and random spatial coordinates are chosen.
%An outside routine must take care of delaying these interference sources
%depending on microphone locations chosen

%Outputs:

%interf - cell_array of all the interference data drawn from audio file

%fft_fctrs - cell array of vectors used for delaying 'num_temp'-type
%interferers in frequency domain (must be done by outside routine).

%fft_fctr - vector used for delaying any 'num_perm'-type
%interferer in frequency domain (must be done by outside routine).

%interflocs_cel - cell array of
%randomly generated spatial location of all the interferers;
%see 'space and physical constraints' section of code below.

%strtpos - array of starting position (in time samples)
%of each 'num_temp'-type interferer in each trial;
%used by outside routine to place these interferers at the given positions

%k - maximum possible time length of one interference profile after delays;
%this is based on the maximum distance between a microphone and an
%interferer; the maximum mic_array length is assumed to be 1 m;
%the maximum distance inteference can be generated is defined in
%'space and physical constraints' section of code below.
%k = sig_len + maxsmplsdelay;
%this value is used by an outside routine for delaying purposes

%shift_smp - since the all interference sources will be delayed there will
```

```
%by default be silence at the beginning of an interfrence profile;
%to avoid this silence, the profile will start at sample # shift_smp
%of length k signal and will continue for sig_len samples;
%this effectively produces a window of interference
%which is length sig_len and has little to no silence at the beginning;
%this operation is done by an outside routine.


function [interf,fft_fctrs,fft_fctr,interflocs_cel,strtpos,k,shift_smp]=...
        gener_interf_old(filename,Fs,num_perm,num_temp,rand_trials,sig_len)
%hold on
num_interf=num_perm+num_temp;

%reading interference wav file
data = wavread(filename)';
LENG=length(data);

%signal sample constraints
Lmin=round(24/48*sig_len);
Lmax=round(40/48*sig_len);
%e.g. for a 2400 sample signal, temp interferer length will be chosen
%randomly to be between 50 and 1200 samples


%signal amplitude constraints
%^no longer used...
%amplLo=.25;
%amplHi=2.5;
%ampl=1;

%space and physical constraints
c=339;%speed of sound at altitude of Columbus Ohio
rmin=1.5;
rmax=15;
thetamin=-90;
thetamax=90;

%compute zero-padding to accomodate delays
distmax=rmax+.5;
maxsmplsdelay=ceil((distmax./c)*Fs);
maxsmplsdelay=maxsmplsdelay+mod(maxsmplsdelay,2);
pad=zeros(1,maxsmplsdelay);
shift_smp=round(maxsmplsdelay*.7);
k=sig_len+maxsmplsdelay;

%preallocating
interf=cell(num_interf,rand_trials);
strtpos = zeros(num_temp,rand_trials);
fft_fctrs=cell(num_temp,rand_trials);
interflocs_cel=cell(1,rand_trials);

%preparing additional auxilary vectors
fft_fctr=-2i*pi*[0:k/2 -k/2+1:-1]/k*Fs;

%populating interference array
for trial=1:rand_trials

    %random spatial locations
    interflocs = [rand(1,num_interf)*(rmax-rmin)+rmin;
            rand(1,num_interf)*(thetamax-thetamin)+thetamin];
    interflocs_cel{trial} = [interflocs(1,:).*sind(interflocs(2,:))
                             interflocs(1,:).*cosd(interflocs(2,:))];
    %Easier to generate interf in polar coords to have all interference be
```

```
    %at least 1.5 m away from mic array center and more sparse as the distance
    %increases. Then interference locations are converted to cartesian coords
    %sind x=r*sind(theta) as theta measured from y-axis

    for i=1:num_perm

        %ampl=(rand*(amplHi-amplLo)+amplLo);

        pos=randi(LENG-sig_len+1);
        chunk=data(pos:pos+sig_len-1);

        %nonstat=ampl*chunk;
        %rms_nonstat=sqrt(mean(nonstat.^2));
        %white=0;%rms_nonstat/3*randn(1,sig_len);
        interf{i,trial}=[chunk pad];
    end
    %lin='-:+o*.xsd^v><ph';
    %cel={};
    for i=i+1:i+num_temp

        leng=randi([Lmin,Lmax]);
        leng=leng+mod(leng,2);
        padded_leng=leng+maxsmplsdelay;
        %ampl=(rand*(amplHi-amplLo)+amplLo);

        pos=randi(LENG-leng+1);
        chunk=data(pos:pos+leng-1);

        %nonstat=ampl*chunk;
        %rms_nonstat=sqrt(mean(nonstat.^2));
        %white=0;%rms_nonstat/3*randn(1,leng);
        interf{i,trial}=[chunk pad];

        strtmin=round(maxsmplsdelay*2/3);
        strtmax=sig_len-leng+1;
        strt=randi([strtmin,strtmax]);
        strtpos(i-num_perm,trial)=strt;

        fft_fctrs{i-num_perm,trial}=...
         -2i*pi*[0:(padded_leng)/2,-(padded_leng)/2+1:-1]/(padded_leng)*Fs;

        %plot(strt:strt+leng-1+length(pad),interf{i,trial},...
        %'color',rand(1,3),'linestyle',lin(i-num_perm))
        %cel{end+1}=[num2str(i-num_perm) ' ' num2str(leng)];
    end
    %legend(cel);
end
end
%%%%%%%%%%%% END OF FILE %%%%%%%%%%%%%%
```

gener_desired_noise.m

```
%%%%%%% gener_desired_noise.m %%%%%%%%%%%%

%this generates desired signal from given crowd noise (or some audio) file

%By Sergei Preobrazhensky


%Inputs:

%filename - full path of audio data file from which to draw interfernce

%Fs - sampling frequency (should match that of audio file preferably)

%rand_trials,sig_len - rand_trials # of desired signal sections,
%each sig_len time samples long will be generated.

%NOTE: desired signal sections are not delayed across a microphone array
%in this function. Only random audio data is drawn.
%An outside routine must take care of delaying
%depending on the microphone locations chosen.

%Outputs:

%desired_noise - cell_array of the data drawn from crowd noise audio file
%to be used for the desired signal; each cell contains a section of desired
%signal corresponding to one trial (each such section will have an
%interference profile added to it; the interference profile is generated
%by outside routines;)

%shift_smp - since the desired signal will be delayed, there will
%by default be silence at the beginning of a desired signal section;
%to avoid this silence, the section will start at sample # shift_smp
%of the extended desired signal (extended to accomodate delays)
%and will continue for sig_len samples;
%the samples before and after will be discarded;
%this effectively produces a window of desired signal
%which is length sig_len and has little to no silence at the beginning;
%this operation is done by an outside routine.

%des_fft_fctr - cell array of vectors used for delaying
%desired signal in frequency domain (must be done by outside routine).

function [desired_noise,shift_smp,des_fft_fctr]=...
          gener_desired_noise(filename,Fs,rand_trials,sig_len)
%hold on
rng('shuffle');

%reading interference wav file
data = wavread(filename)';
LENG=length(data);

%space and physical constraints
c=339;
des_y=1;
des_x=0;
arrayLmax=1;

%compute zero-padding for delay
distmax=sqrt((des_y^2+(arrayLmax/2-des_x)^2));
maxsmplsdelay=ceil((distmax./c)*Fs*1.5);
```

```
shift_smp=maxsmplsdelay+mod(maxsmplsdelay,2);
pad=zeros(1,shift_smp);
k=sig_len+shift_smp;
des_fft_fctr=-2i*pi*[0:k/2 -k/2+1:-1]/k*Fs;



%preallocating
desired_noise=cell(rand_trials);

%drawing random audio from file for each trial
for trial=1:rand_trials

    chunk=0;

    for i=1:10
        pos=randi(LENG-sig_len+1);
        chunk=chunk+data(pos:pos+sig_len-1);
    end

        desired_noise{trial}=[chunk pad];

end


end
%%%%%%%%%%%% END OF FILE %%%%%%%%%%%%%%%%
```

A3.3 Prototype Perl Code

This code was drafted and briefly tested to confirm correct functionality. Read the comments section at the beginning of the code for help. This code only converts a wave file to text via speech recognition; no computation of the word error rate is done. Further development is needed to automate speech recognition tests.

```
############### WavtoText.pl ###############

=pod


Written by Sergei Preobrazhensky, May 2012

The following perl script is a prototype for
automating key presses and mouse clicks to
run Google Speech Recognition on many wave
files. It is implied that each of these
wave files has been processed (to suppress
noise/interference) under various parameters
of the MVDR beamforming algorithm written
in MATLAB by the author. These parameters
are described in step 9.

Steps to run this script:
1.      Make sure you are running 32-bit Windows, as the
        auxiliary Perl modules this script relies on are for Win-32 systems.

2.      Download, install Strawberry Perl v. 12; this is a freeware perl
        interpreter available online and has the convenient cpan module
        installation support via the command line.

3.      Install cpan modules.
        a.      Open the Command Prompt (Start->Run->enter “cmd”
                OR: Start->enter “cmd” in search bar)
        b.      Make sure you are connected to the internet.
        c.      Enter “cpan Win32::GuiTest“
        d.      Enter “cpan Win32:Clipboard”
        e.      Enter “cd [PATH]” where the [PATH] is the full path of this file.
                You first may have to first type “[X]:” where [X] is the drive containing this file.
        f.      Keep the command prompt open.

4.      Make sure Windows Media Player (WMP) v. 12 or similar
 version is installed; Perl code specifically works with
 the WMP window and uses the File->Open URL command

5.      Make sure G Chrome 11 with Google Speech Recognizer (GSR)
App v. 3 is installed (other versions may work). Open GSR in
a tab, and keep focus on the tab.

6.      Make sure Google Speech Recognizer in unobstructed from
```

view when running Perl script; at least make sure that the
“microphone” button and text box are completely unobstructed
for automated mouse clicks.

7. WMP can be minimized or on the screen. There are no mouse clicks involved with WMP

8. The wave file path/name convention in this script is:
C:\Speech\Male1out_L-#_rms-#_bands-#_fg-#hifi.wav where each
“#” stands for the parameter value using which the chosen
wave was created. Make sure you have all the wave files
located at this path prior to running script. The path
and naming conventions can be modified below.
a. It is assumed that wave files are 15 s or less, or
the timing will not work, unless you change SLEEP_SECS
to a higher value.

9. Modify the constant declarations of parameters correspond
to what parameters the wave files are and algorithm below as
needed. A wave for each of the possible parameter choice
combinations must be accessible.
   a. LEN_A is array of lengths of microphone array
   b. RMS_A is the array of interference to signal ratios
   (RMS of interference when RMS of signal is 1).
   c. BANDS_A is the array of numbers of sub-bands
   (144/band_size for 300-3400 Hz; for 150-6350 Hz,
   num_bands=2*288/band_size will be automatically
   computed).
   d. FG_A is the array of forgetting factors
   e. HIFI_A is the array indicating a ‘hifi’ extended
   150-6350 Hz bandwidth or ‘’ (empty string) for a
   300-3400 Hz bandwidth. A wave file for each of these
   cases must exist. Note that if only ‘hifi’ is used,
   the line “my $hifi_ind=#;” must have 1 in place of “#”;
   if both ‘’ and ‘hifi’ are used (declared in HIFI_A in
   that order), the said “#” must be 0;

10. Make sure that you are still connected to the internet.
Enter “perl wavtotext.pl” in command prompt to run Script,
and follow the two setup instructions. Move the command
prompt window as needed not to obstruct GSR. Do not click
with the mouse; only point and press enter; the command
prompt window must be active when you press enter.

11. Do not use the PC while the script is running. If you
must stop the script, click on the command prompt window
and press Ctrl+C. You may be interrupted by automated
mouse clicks or key strokes, so try until you succeed.
The best time to interrupt is while the audio is playing
in WMP. It takes approximately half a minute or so per
to convert one wave file to text so you may want to split
the work into reasonable sections if you have many files
to convert.

12. The output text file with resulting strings of speech
is saved in C:\recog_text given the name which corresponds
to the date and time of start of script execution. In the
file each string of speech is written under a line that
details the parameters of the wave file used.
=cut

use strict;

```
use warnings;
use 5.010;
use Win32::GuiTest qw(FindWindowLike SetForegroundWindow
                      GetCursorPos MouseMoveAbsPix SendLButtonDown
                      SendLButtonUp SendKeys);
use Win32::Clipboard;

#modify file path and format as needed:
use constant PATH_S => 'C:\\speech\\';
use constant PATH2_S => 'C:\\recog_text\\';
use constant PREF_S => 'Male1out';
use constant SUFF_S => '.wav';
use constant SEP1_S => '_';
use constant SEP2_S => '-';

#modify setup parameters as needed:
use constant SLEEP_SECS => 20;
use constant NUM_TRIALS => 1;

#modify speech processing parameter values as needed:
use constant LEN_A => (.8);
use constant RMS_A => (2);
use constant BANDS_A => (9, 12, 18, 36);
use constant FG_A => (.95, .97, .98, .99, .995, .999);
use constant HIFI_A => ('hifi');

sub Setup(); #asks user to indicate two cursor positions for clicking  in
             #GSR window. returns two pairs of coordinates.
sub WavText($); #plays audio file at input address, returns recognized text
sub TimeStamp();#returns date+time in format acceptable for folder name

my ($x1, $y1, $x2, $y2) = Setup();
my $loctime=TimeStamp();
my $num_bands;

#run speech recognition on audio files with all given parameter combos
foreach my $len (LEN_A){

        foreach my $rms (RMS_A){

                foreach my $bands (BANDS_A){

                        foreach my $fg (FG_A){
                                my $hifi_ind=1;

                                foreach my $hifi (HIFI_A){
                                        $hifi_ind++;
                                        $num_bands=($bands * $hifi_ind);
                                        my $file_addr=
                                        PATH_S.PREF_S.
                                                SEP1_S.'L'.SEP2_S.$len.
                                                SEP1_S.'rms'.SEP2_S.$rms.
                                                SEP1_S.'bands'.SEP2_S.$num_bands.
                                                SEP1_S.'fg'.SEP2_S.$fg.
                                                $hifi.SUFF_S;
                                        my $band_siz= 144/$bands;
                                        open my $fh, '>>', PATH2_S.$loctime.'--'.PREF_S.'.txt';
                                        say $fh "L=$len rms=$rms bandsize=$band_siz ".
                                                "forget=$fg $hifi";
                                        say $fh "\n";
                                        close $fh;
```

```
                                        for (my $trial=1; $trial<=NUM_TRIALS; $trial++){

                                                my $text = WavText($file_addr);
                                                open $fh, '>>', PATH2_S.$loctime.'--'.PREF_S.'.txt';
                                                say $fh $text."\n";
                                                close $fh;
                                        }
                                        open $fh, '>>', PATH2_S.$loctime.'--'.PREF_S.'.txt';
                                        say $fh '';
                                        close $fh;
                                }
                        }
                }
        }
}

sub WavText($){
                my ($window)=FindWindowLike(0, 'Windows Media Player');
                SetForegroundWindow($window);#focus on Windows Media Player
                SendKeys("%fu".$_[0]);#enter address into File -> Open URL...

                SendKeys("~");

                ($window)=FindWindowLike(0, 'Speech Recognizer - Google Chrome');
                SetForegroundWindow($window);#focus on Speech Recognizer

                MouseMoveAbsPix($x1,$y1);
                SendLButtonDown();
                SendLButtonUp(); #click speech recognition button

                sleep(SLEEP_SECS); #wait for speech recogntion to generate text

                MouseMoveAbsPix($x2,$y2);
                SendLButtonDown();
                SendLButtonUp(); #click field of recognized text

                SendKeys("{END}");
                SendKeys("{SPACE}");
                SendKeys("^a");
                SendKeys("^x"); #cut text

                return my $text = Win32::Clipboard::GetText();
}


sub Setup(){
                say 'Move cursor to speech recognition button, enter when ready';
                my $a=<STDIN>;
                my ($x1,$y1) = GetCursorPos();
                say 'Move cursor to left inner edge of recognized speech text field, enter when ready';
                $a=<STDIN>;
                my ($x2,$y2) = GetCursorPos();
                return($x1,$y1,$x2,$y2);
}

sub TimeStamp(){
                my $loctime = scalar localtime();
                $loctime = join ('_', split (':', $loctime));
                return $loctime = join ('-', split (' ', $loctime));
}

###############     end of file     ########################
```