Sound Shield Final Report

Submitted To

Dr. Brian Evans
Dr. Gregory Allen
The University Of Texas at Austin, ECE Department

Prepared By

Yeong Choo
Jun Qi Lau
Dung Le
Mark Leatherman
Sung Park
Negin Raoof
Brandon Williams

EE 464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Spring 2016

CONTENTS

TABLES

FIGURES

EXECUTIVE SUMMARY

The purpose of this final report is to give a comprehensive account of the results of the Sound Shield Senior Design Project. This report details the design problem, solution, implementation, and test results for creating a dynamic, real-time noise-masking software package. Sound shield achieves this by performing frequency analysis on the noise outside of a room and generating a custom mask to cover that noise to the perception of the user inside of that room. Sound Shield is capable of running on multiple platforms and contains a user interface that enables the user to access the software via a web browser on any other device connected to the same wifi network. Sound Shield is designed to run in the background on the user's personal computer or a dedicated embedded system such as the Raspberry Pi. The limiting factor placed on Sound Shield's design specifications is that the output mask must create a soothing sonic environment for the user at all times. That is to say, the algorithm must track the changing frequency content of the noise in order to adapt its mask to cover that noise before the user becomes distracted by it. However, the mask must never become as jarring to the user as the noise itself. This means that if a loud, transient noise suddenly occurs, the software must be smart enough to let that noise go unmasked, so as not to disturb the user even more.

Our team was able to devise a solution that is robust enough to detect and adapt to most real-world noise situations that one would want to mask, but which is also delicate enough to let extreme noise cases go unmasked so as to prioritize a peaceful sonic environment for the user at all times. We developed this solution using the c programming language, which is capable of performing the necessary real-time DSP calculations in the background on a user's personal computer. We designed the program to interface with the user's pre-existing audio setup, so that the software could run on anything from consumer-grade audio hardware to professional audio equipment. We also used Python, Matlab, JavaScript, CSS, cmake, Websockets, and NGINX to meet all of our project's problem specifications and create a noise-masking prototype that is smart, user-friendly, and capable of being expanded into a marketable solution.

The testing data, which we present in this report, confirms that our prototype performs according to the specifications defined in our problem statement. Our masking algorithm begins to react 44 ms after the noise is detected (during the next callback function), but only adapts its characteristics slowly so as to not startle the user. Similarly, our mask reacts to adjust its volume to at least 4dB over that of the noise within all audible frequency bands, so that the noise is masked to the perception of the user, unless the noise occurs too suddenly or reaches a volume that we deem too loud to mask. In this way, the software always prioritizes the comfort of the user.

Sound Shield is a simple software solution capable of solving a real-life problem that millions of people encounter everyday: unwanted noise. In our busy and crowded modern world, unwanted noise interrupts our privacy, comfort, and productivity. Oftentimes, when we want to sleep, neighbors, traffic, trains, barking dogs, and many other elements from the surrounding environment awaken us. By creating a smart noise-masking system that knows both when and how to change its mask to cover intrusive noise, we are offering millions of people a new way to enhance their everyday lives by improving the environments in which they sleep, work, and live.

1.0 INTRODUCTION

This document reports the Sound Shield Team's design solution for a dynamic noise-masking software system and evaluates the prototype's success based on its ability to meet the project requirements. We will demonstrate how the Sound Shield prototype meets the design goals outlined in our Design Implementation Plan, as well as the performance specifications described in our Testing and Evaluation Plan.

Sound Shield is a cross-platform software solution capable of dynamically covering unwanted noise by introducing the soothing sounds of nature as sound masks into the environment. By analyzing the frequency content of the input noise in real time, Sound Shield is able to generate custom sound masks that effectively render aggressive noise imperceptible to the user.

This report culminates the Sound Shield Team's efforts to research, design, and implement such a dynamic noise-masking software prototype. In the first section we will detail the problem's design specifications, parameters, and constraints as proposed by Faculty Mentors Dr. Brian Evans and Dr. Greg Allen. Next, we will present our unique solution to the problem and demonstrate how we fulfill the specifications outlined in our team's Design Implementation Plan, including meeting real-time performance and generating masks that effectively render noise imperceptible. Subsequently, in the Design Implementation section we will enumerate the decisions, obstacles and tradeoffs that we encountered while implementing this solution. For example, we made modifications to the original filter bank design, as well as our method for reading the mask files in order to improve our design solution. Next, we will demonstrate how the tests proposed in our Testing and Evaluation Plan produced results that verified our design solution, proving that Sound Shield indeed generates a dynamic noise mask capable of tracking the frequency spectrum of the noise in real time. Following the test results, we will evaluate our team's time and cost considerations, showing how we completed our project on-schedule according to our Gantt Chart. Since we did not have an allocated budget for the project, we did not encounter any cost constraints and were able to implement the project on our personal computers and audio hardware. Next, we will present the safety and ethical considerations relevant to our project, including how Sound Shield can benefit those suffering from insomnia or tinnitus. Finally, we will present our recommendations on how future teams could improve our

Sound Shield prototype, based on the extensive research that we have conducted over the past nine months. These recommendations mainly focus on how improvements to the software could provide the user with a richer, more customizable noise-masking experience.

2.0 DESIGN PROBLEM

The Sound Shield project was founded by faculty mentors Dr. Brian Evans and Dr. Greg Allen in order to improve upon the shortcomings of traditional noise-masking devices by developing a more robust dynamic noise-masking solution. Unwanted noise interrupts concentration, disturbs rest, and inhibits productivity. Daily traffic from large highways prematurely wakes up individuals living in cities. Lawn mowers and weed whackers disturbs the peace of suburban families in their homes. Loud music propagates through walls of apartments, disturbing neighbors. Office chatter travels across the workplace, distracting employees. Counselor and doctor conversations can be overheard by passing individuals, eliminating patient privacy. Unlike active noise-cancellation, which introduces secondary sounds into the environment to cancel out noise, sound-masking attempts to solve these problems by introducing sound masks into the environment that reduce the perceptibility of these noises. Traditional sound-masking devices, such as white noise machines, limit their functionality to manual user controls: these systems are typically on-or-off state devices that require the user to physically control the mask's volume. Because of these limitations, traditional noise-masking devices often end up producing masks that are just as distracting as noise itself, since they cannot automatically respond to changing environmental noise conditions. A dynamic solution such as Sound Shield actively responds to changing noise, generating sound masks that are only as loud as necessary to cover up the noise present without requiring any user input. This insures that the system is as minimally intrusive as possible, while still effectively masking intrusive noise.

With the help of our faculty mentors, we defined clear design parameters and constraints to guide our development of a dynamic noise-masking solution:

Non-Offensive: Sound masks produced by Sound Shield must not be as distracting or annoying as the noise being masked.

Real-Time Performance: The system must perform frequency analysis on the noise without incurring a noticeable delay in response time. We have determined that 185ms is an acceptable upper bound to the system's latency [1].

Runtime Memory: The masking algorithm and user interface software may utilize a maximum of 20 MB during runtime to minimize the software's RAM footprint.

Cross-Platform: In order to be a multiplatform solution, Sound Shield must run on UNIX and UNIX-based operating systems, not limited to OS X, Ubuntu, and Raspbian.

Plug & Play: Sound Shield must be capable of interfacing with any user microphone and speaker setup.

Customizable: Sound Shield must include a real-time user interface that provides the user access to configure various functionalities of the software, including volume adjustment, type of mask to play, and remote system power control.

Masking Level: The intensity of the sound mask in each frequency band must exceed the intensity of the noise by at least 4dB in order to mask the sound [2].

Using the above criteria, the sound shield team made design decisions, delineated and explained later in this report that enabled the team to achieve the ultimate goal of developing a fully functional dynamic noise-masking software prototype.

3.0 DESIGN SOLUTION

Sound Shield's design is simple, yet the software that achieves the desired functionality is refined and optimized. Sound Shield receives noise signals via a microphone, analyzes the frequency content of those signals over a short window of time, and synthesizes a sound mask to cover up those intruding noise signals when played out through speakers. For physical system setup, the microphone is placed in the presence of the noise which the user desires to mask, and the computer running Sound Shield is placed inside of the room where the user is present, connected to the speakers. The peripherals are placed in separate environments in order to avoid feedback loops and to prevent Sound Shield from attempting to mask its own sound masks. This example setup is illustrated in the figure below.

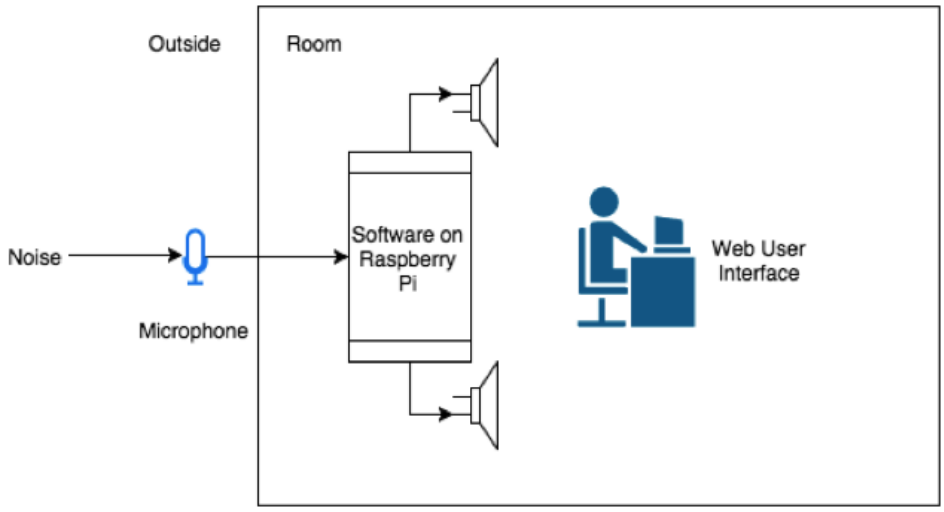


Figure 1. User Setup Diagram

Sound Shield is able to generate sound masks that more comprehensively conceal noise than traditional noise-masking devices because of the analysis and processing it executes in order to generate its sound masks. Specifically, the software accomplishes this by tailoring the frequency content of the generated sound mask to the frequency content present in the input noise signal. By sampling a microphone, Sound Shield can identify how much power is present in each frequency band of the noise, and can then calculate the proper amount of gain to apply to those same bands of the sound mask's frequency spectrum. This method enables Sound Shield to lower its volume when noise is absent, raise its volume when there are increased amounts of noise, and in both cases boost only the necessary mask frequencies required to effectively cover the noise.

3.1 Software Design

Sound Shield's software is organized into several subsystems that work together in a modular fashion. As the Modular System Diagram below illustrates, the main program is divided into three subsystems: the noise-analysis subsystem, the mask-level computation subsystem, and the mask-synthesis subsystem.

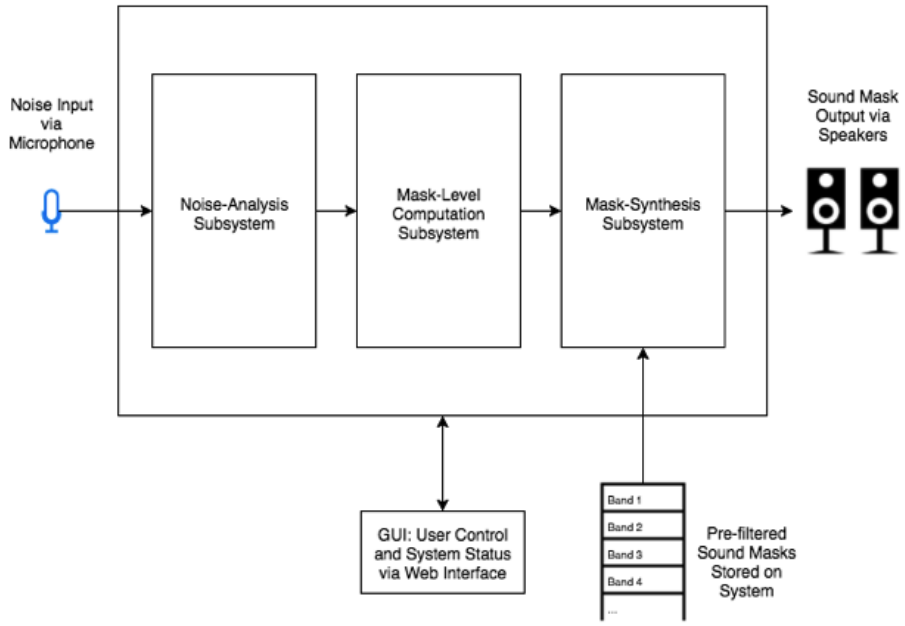


Figure 2. Modular System Diagram

3.1.1 *The Noise-Analysis Subsystem*

The first module of Sound Shield, the noise-analysis subsystem, is responsible for receiving the input noise audio stream from the microphone and analyzing that signal's frequency content. Specifically, this subsystem is responsible for finding the power present throughout the frequency spectrum of the noise signal. Our design solution groups the noise and output sound mask frequency spectrums into ten octave-spaced frequency bands. Therefore, while this subsystem's Fast Fourier Transform (FFT) algorithm yields 2048 frequency bins containing the magnitude and phase of the noise signal throughout its entire frequency spectrum, it groups those bins into ten separate octave-spaced frequency bands for analysis. Since half of the generated 2048 frequency bins is a reflection of the other half, Noise-Analysis Subsystem discards the first 1024 bins of the FFT calculation. Then the system takes the remaining 1024 bins, groups those bins into ten octave-spaced bands and performs a Power Spectral Density (PSD) calculation on each band. This method generates scalar values for the noise power present in each frequency bin, which can be used in future mask-gain algorithms. These values are specifically passed as inputs to the Mask-Level Computation Subsystem. The figure below illustrates how this FFT/PSD method works within the entire scheme of Sound Shield.

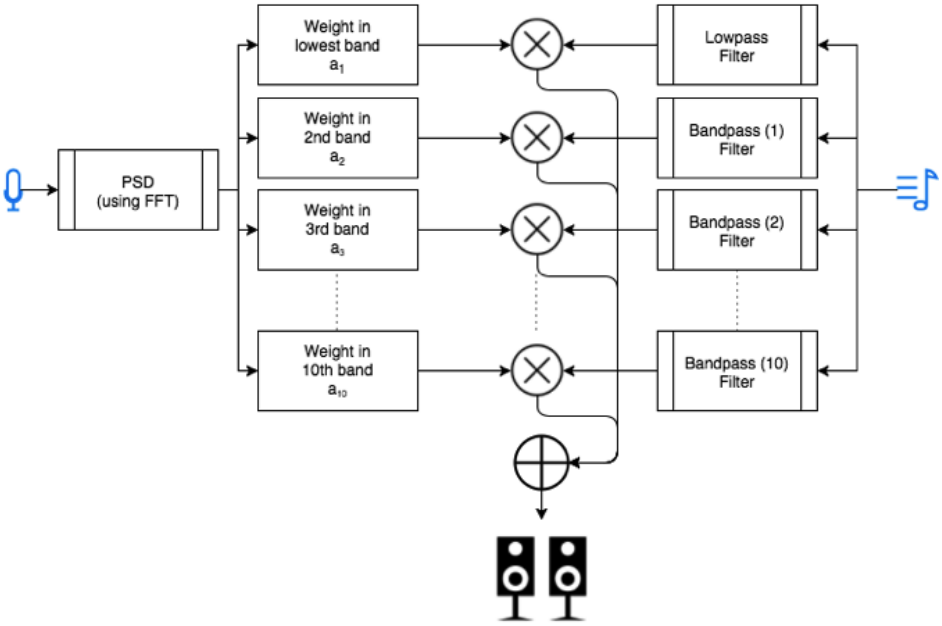


Figure 3. The FFT/PSD Method

The following table analyzes Sound Shield's FFT/PSD algorithm, broken down from the number of additions, multiplications and other arithmetic operations.

Table 1. Complexity of our FFT/PSD Function

	Real Multiplications	Real Additions	Comparisons
FFT calculation (buffer size N = 1024)	4*(N/2)*log2(N)= 20480	[(N/2) + 2N]*log2(N)= 25600	
Calculating |FFT magnitude|^2	2N = 2048	N = 1024	
Normalization (Dividing by buffer size)	2N = 2048		
Peak picking gains for the 8 lower frequency bands			(1/4)N = 256
Averaging gains for the 2 higher frequency bands	2	(3/4)N = 768	
Averaging filter over 10 bands (B = 10)	B = 10	2B = 20	
TOTAL	4N + 2N*log2(N) + B =	2N + 2.5*log2(N) + 2B =	(1/4)N =
	24588	27412	256

The number of clock cycles that it takes to run each call of this function depends on the platform's architecture and the optimization process of the compiler.

3.1.2 *The Mask-Level Computation Subsystem*

Once the power present in each frequency bin has been computed, the noise-level measurement subsystem passes that information to the mask-level computation subsystem, which is responsible for assigning a weight to each subband of the output mask. Furthermore, this subsystem is responsible for governing how the mask's spectrum and intensity are allowed to change over time. The design decisions involved in implementing this module were determined through empirical process. In our first implementation, we utilized the power spectral coefficients from each frequency band in the noise-analysis module and directly applied it to the corresponding bands of the mask. This resulted in a solution that was just as distracting to the user as the noise itself, since the mask was allowed to change instantaneously along with the changing noise signal. However, once we applied certain constraints to how the frequency bands of the sound mask were allowed to change, it drastically improved the quality of the resultant mask.

The first change we implemented was to apply a 15-frame averaging filter to the output signal, where the amplitude of the next output sample depends on the previous 15 input frames. This causes the mask to change at a slightly slower rate than the noise itself. Since the output masks we are using are meant to emulate the sounds of ocean waves as the rise and fall, or a bubbling creek as it rushes through the woods, this time-averaging effect greatly improves the quality of the sound mask.

The second change we implemented in this mask-level computation module was to gang together the neighboring bands of the mask. Under this approach, when one band changes in response to a change in the noise level within that band, the neighboring bands will also change, although by a smaller amount. We determined that the most convincing results occurred when we allowed the neighboring bands to change by 25% the amount as the primary band of interest. By loosely ganging together the frequency bands of our output masks, we were able to produce more realistic sound masks.

3.1.3 *The Mask-Synthesis Subsystem*

In the third module of our software, the mask-synthesis subsystem, the normalized weights for the individual frequency bands of the sound mask are received in a vector and applied as gains to each subband of the mask. These mask subbands have already been pre-filtered offline from the original mask audio file and stored on the system, using the naming convention b0.wav - b9.wav. The mask-synthesis subsystem adds these weighted subbands together sample-by-sample, normalizes the result to prevent clipping, and plays out the resulting stereo signal via the speakers interfaced with the system.

In order to be able to separate the original mask sound file into its constituent subbands, we first had to design a filter bank. We began by designing a linear-spaced filterbank in MATLAB. However, when Dr. Evans informed us that the human ear more closely resembles an octave-spaced filterbank, we redesigned it to more closely match the natural sensitivity of the human ear. Using MATLAB, we created a Butterworth IIR filter bank whose phase response approximates linear phase within the passband of each filter, so as to avoid phase distortion. The figure below shows the magnitude response of this initial filterbank, which we used to filter the original mask files.

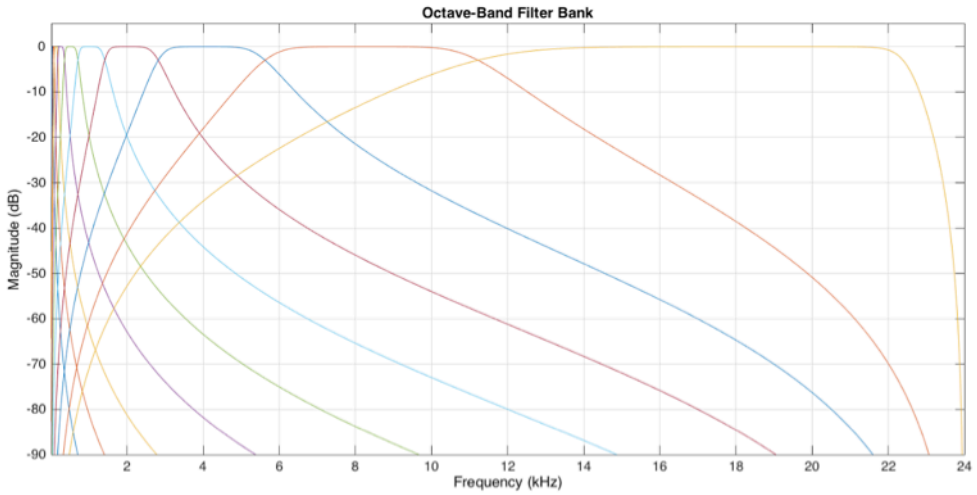


Figure 4. Octave Filterbank

As one can see, the bands are separated by octaves. In other words, the second band spans twice as many frequencies as the first, and the third band spans twice as many frequencies as the

second, and so forth. When the filtered files are reconstructed, they recreate the original sound file perfectly enough so the ear cannot discern the difference between them.

Although we already had a functional filterbank in MATLAB, near the end of the project we proceeded to design an implementation of the same filterbank in Python. In this way, the user can filter their own mask files without having to purchase MATLAB. We experienced many setbacks while designing the filter bank in Python, because the Python DSP packages were all written for mono files. Since we are using stereo files for higher-quality masks, we had to modify all of the Python packages to correctly manipulate the mask files in stereo, using an interleaved data format. Once we had verified the quality of audio files produced by this new filter bank, we proceeded to package this Python script into a standalone executable, which we also include in our code so that users can pre-process their own mask files. Once these filtered mask files are placed in the *currently_playing* mask directory, Sound Shield will begin to read them and use them in the mask-synthesis subsystem.

3.2 Graphical User Interface

The user can communicate with the main software via a standalone web application that we designed using JavaScript and CSS. This user interface is hosted through a web application and can therefore be accessed via any device that is connected to the same wifi network as the computer running the main Sound Shield software. It interacts with the main program using client-server communication through the Websockets library and NGINX. This powerful combination of Websockets and NGINX allows the UI to achieve full-duplex communication with the main software in real time, thus allowing the user to remotely start and stop the system and adjust the volume of the overall system, as well as the maximum volume of each of the individual frequency bands. Moreover, this UI allows the user to monitor the mask's efficiency via overlaid magnitude plots that show the difference in dB between the noise input and Sound Shield's mask output. In this way, the user can see if they need to turn the volume of the mask up or down in order to cover the noise by at least 4dB, as specified in our design specifications.

The figures below show the home page of our graphical user interface.

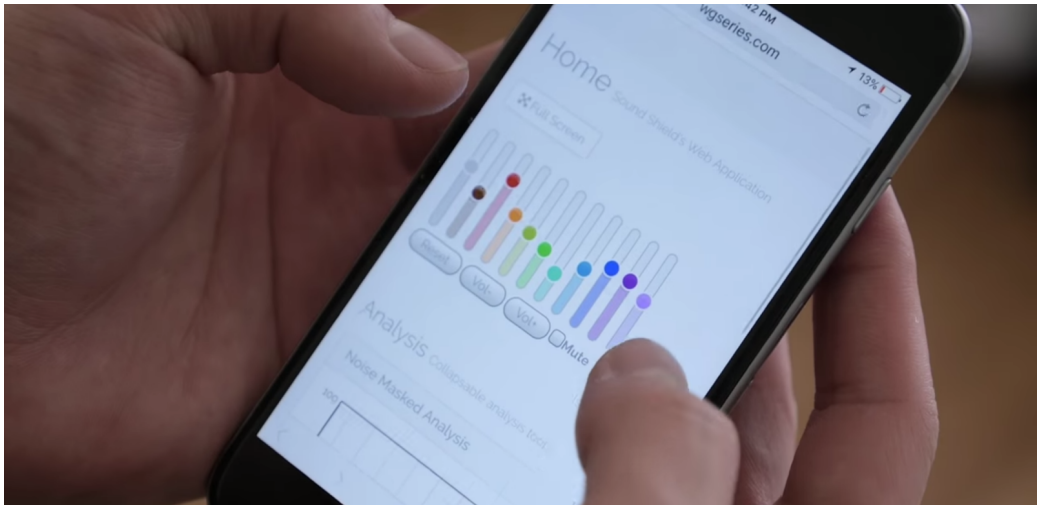


Figure 5. Graphical User Interface

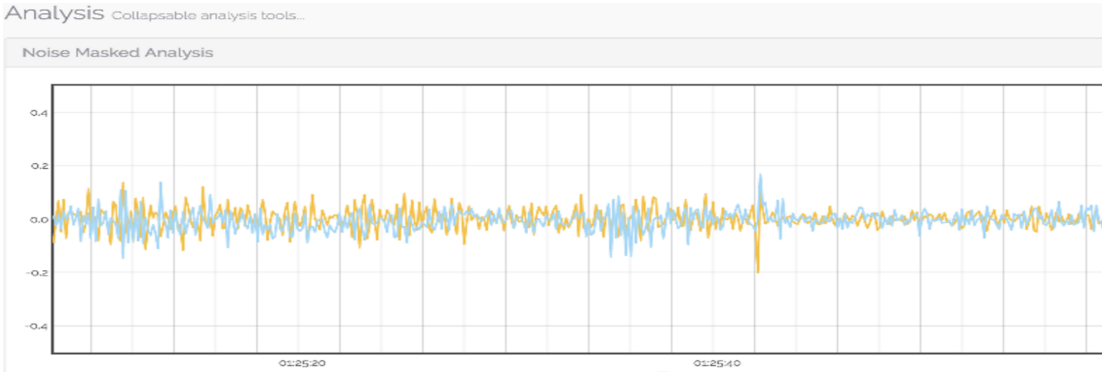


Figure 6. Input Noise & Mask Response in Time Domain

3.3 System Overview

By carrying out the design solution detailed above, we were able to meet all of the problem specifications enumerated by our faculty mentors. We created a non-offensive mask that achieves real-time performance and stays within the memory constraints that we specified. We also implemented a graphical user interface that communicates with the main software through a web application where the user can customize the mask and monitor the masking status as it runs in real time. The only feature in the UI that we did not implement is the ability to let the user change masks through the web interface. Instead of giving the user access to specify which mask they will play through the web application, he or she must instead copy and paste the filtered bands of the mask file into the system's *currently_playing* mask directory. Since we use a simple naming convention for the filtered band files, the new mask will automatically start playing on

the next startup. While this solution would not be acceptable for a marketable software package, we decided to implement mask selection in this way for our prototype so that the user can filter their own mask files using our standalone python executable. In summary, the following table gives a synopsis of the tools and methods that we used to fulfill each of our project's design specifications:

Table 2. Design Tools & Methods

Design Specification	Tools & Methods used to achieve the Specification
Non-offensive	15-frame averaging to smooth the mask's response
	Gang neighboring bands together to maintain natural character of sound masks
	Impose mask ceiling
Real-time performance	Port Audio performs callback function every 44 ms. (minimum delay for our system to begin responding to noise)
Runtime Memory	Main program written in C for efficiency and ability to run in the background on any OS
Cross-platform	Cmake manages compiling/linking of all code
	Port Audio manages I/O audio interface
Plug and Play	Port Audio reads and writes through standard I/O ports of computer
Customizable	GUI features written in JavaScript, CSS, NGINX, Websockets, Python
Masking Level	Calibration of gains applied to mask bands

4.0 DESIGN IMPLEMENTATION

Throughout the design process, our team was faced with implementation choices, and forced to make educated and well-founded decisions in order to manage development time and optimize system performance. Our main focus was to find solutions that optimized real-time performance.

We faced our first implementation choice when designing the mask filterbank. Initially, our solution was based on splitting the noise and mask signals into linearly-spaced bands. However, after conducting more research into audio filters and studying audio systems and equalizers, we found that the human ear automatically divides audio signals into octave-spaced frequency bands. As we saw in Figure 4, the octave filterbank, this frequency spectrum division results in filters with finer resolution in the lower frequencies, and wider filters in the higher frequencies. Since human ears are more sensitive to frequencies from 1kHz to 5kHz, dividing the frequency spectrum into octave-spaced bands enabled us to generate masks with increased resolution in the frequency ranges where the human ear is more sensitive. As we will describe in the Recommendations section below, our design could be further improved by providing even more resolution in the middle frequencies where the human ear is most sensitive. However, given the time constraints of our problem, we decided that the octave-spaced filterbank constituted a significant improvement over the linearly-spaced filterbank and that we should therefore leave the further fine-tuning to the human ear's response to a later implementation of the project.

Another important decision that we made was to provide users with the opportunity use their own audio files as the mask sound. Initially, we had a few pre-processed sound masks stored in the mask directory. These masks recreated the soothing sounds of nature such as ocean waves, light rain, rushing water, and a meadow breeze. In order to give users the possibility of uploading their own masks, Sound Shield had to be able to process the chosen audio file upon user command. Our initial preprocessing script was written in MATLAB. However, the Raspberry Pi's operating system, Raspbian, does not support MATLAB. In order to remain platform independent, we decided to rewrite the script using Python, which is compatible with many modern operating systems, including the Raspbian. Since very few Python 3 packages exist for manipulating stereo audio signals, this process required us to do more research on the structure of WAV audio files. Similarly, we were forced to modify the packages we decided to use

(ThinkDSP, PyFIlterbank) so that they could run on Raspbian and operate on stereo, instead of mono, audio files.

Moreover, we faced another implementation choice on whether we wanted to incorporate the preprocessed mask as a data structure to be compiled along with Sound Shield or a completely decoupled mask from the Sound Shield's core engine. During our earliest stages of development, given the limited amount of data structures available in C, our most logical solution for producing an audio output was to traverse through an array. So, with the help from MATLAB, we generated our preprocessed mask by cascading the floating point values from the WAV file into ten different arrays with each array corresponding to each of our ten octave-spaced frequency bands. Each array contained data for audio signals about 10 seconds long, with a sampling frequency of 44100 Hz. With this implementation, we successfully created our first prototype on Linux and OS X. When we migrated our development to the Raspberry Pi, we found that Sound Shield would crash after a certain number of callbacks. We found a trend that at each consecutive execution, Sound Shield was able to make approximately twice as many callback before crashing, compared to the previous execution. By analyzing the number of callbacks at each execution, we were able to identify this problem as a page fault. We verified the page fault by profiling Sound Shield during execution, which would dump the page fault count before crashing. We resolved the page fault by traversing through all of the arrays at initialization in order to page the needed real-time data into the system's RAM.

Furthermore, the fact that we were storing all of our mask data as raw floating-point values in arrays became a concern when we compiled Sound Shield on the Raspberry Pi. The files containing these arrays were large and the time required to compile the mask on the Raspberry Pi was excessive. We found that the long compilation time limited us in our implementation process, and hence we decided to change the format of the preprocessed audio samples to WAV files. We found a powerful C library for reading and writing audio samples in real-time called libsndfile, which is compatible with Linux and Unix systems. The use of this library enabled us to compile our program in a much shorter time (less than 1 minute).
Although decoupling the mask from Sound Shield drastically reduced the compilation time, we ran into another page fault that could not be resolved by the previous solution. To address this

problem, we created an additional buffer and interleaved the output patterns of the two buffers. As one buffer read the mask data needed in the next callback, the other buffer would send its data to the audio output for the current callback. At the end of each callback, the roles of the two buffers would be interchanged.

The last major implementation decision we made was to incorporate libwebsocket and NGINX into the user interface. We needed a server that could support data transmission and reception. Initially, we found that libwebsocket was our best option in terms of hardware resource consumption. However, upon implementation we found that Libwebsocket was not able to properly serve all the files needed to the web client, so we included NGINX to act as the web server. While NGINX served the files needed to the web client, libwebsocket created the necessary websockets for the web client to communicate with Sound Shield in a full-duplex manner.

5.0 TEST AND EVALUATION

In our Testing and Evaluation Plan we listed the National Instruments myDAQ as the core-testing device that would fulfill the needs of handling the audio interface for data acquisition. Throughout the testing process, the testing team learned that the Data Acquisition Toolbox present in the MATLAB environment provided similar functionality to acquire and playback audio data in real time [3]. This switch ultimately saved the testing team from having to learn the nuances of Labview and allowed for the integration of data acquisition and data analysis in the same environment. The decision to change to MATLAB data acquisition toolbox also allowed us a broader selection of testing devices. Instead of being limited to the myDAQ with NI, we could now run the testing scripts using any personal computing device capable of running MATLAB and equipped with an audio I/O interface.

In this section, we will present our system's test results to showcase the success of our current implementation in meeting our design specifications as outlined above [4]. Quantitative tests included system latency test and system performance test, and qualitative tests included subjective feedback gauged at the Senior Design Open House.

5.1 System Latency Test

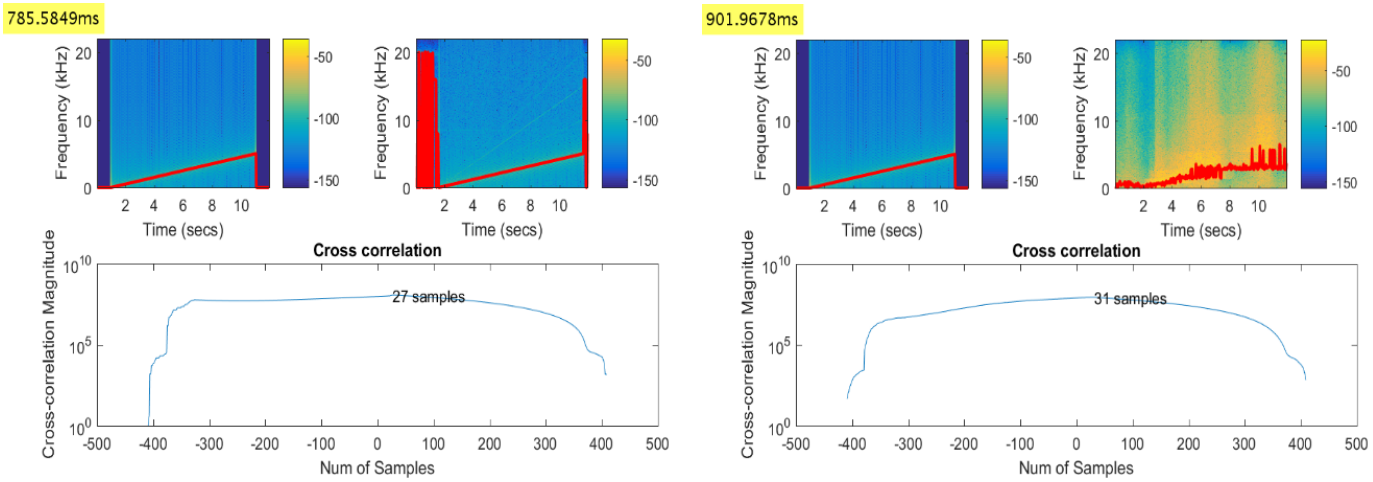
We have indicated in the Design and Implementation Plan as well as in the Testing and Evaluation Plan that humans could tolerate audio lag of up to 185 ms in our system without noticing the noise too distinctly. As we increased the length of averaging filter, the delay until first noticeable mask response also increased.

Our subject for testing is the Raspberry Pi 3 (Model B) running the most recent version of Sound Shield. The testing platform is a personal laptop running Windows 8.1 and the most recent release of MATLAB (R2016a) with the data acquisition toolbox (version 16.1.0) dedicated to acquiring and analyzing data from the DirectSound Audio driver.

The system latency test consists of two parts: first in identifying a known delay through the MATLAB software interfaced with the audio driver, and second in the overall delay upon addition of the Sound Shield system. The difference between the two measurements is the ultimate result of interest that we will document as the Sound Shield system delay. The system latency test also uses a test signal of known frequency content. Our testing team has selected a chirp signal that sweeps through the frequencies from 0 Hz to 5 kHz over ten seconds and another chirp signal that sweeps through the frequencies from 0 Hz to 22 kHz in ten seconds. Both frequency chirp signals are zero-padded to simulate one second long of silence at the beginning and end of the test signals to ensure test system reliability.

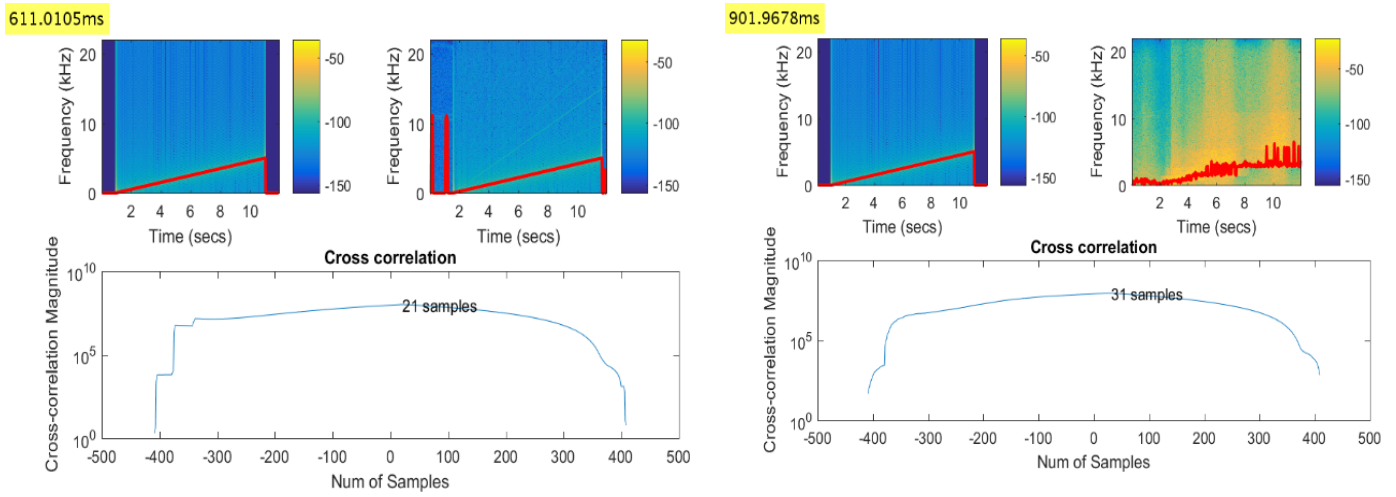
The first part of the system latency test was performed by feeding the output of test system directly back to the input of the test system, also known as the talk-through step of the system test. The second part of the system latency test includes the Sound Shield system to be tested. Both steps of the system latency test employed data logging of audio signals for post processing purposes. We took advantage of using the Short-Time Fourier Transform (STFT) to break audio signals into Kaiser Windows of 5120 samples, each with overlapping of 3840 samples [5]. The principal frequency component is then identified from every window on both audio input and output signals, which are then marked with red markers in figures 7 and 8. A cross-correlation method is then used on the principal frequency component to identify the temporal delay between the audio signals. The method is made useful by the assumption that both audio signals

carry very similar frequency contents but they are separated by certain delay. Two sample scenarios of our system's delay are shown in the test results below:



(a) Talk-through test of 785.58ms (b) Overall test of 901.97ms

Figure 7. Minimum Sound Shield Delay



(a) Talk-through test of 611.01ms (b) Overall test of 901.97ms

Figure 8. Maximum Sound Shield Delay

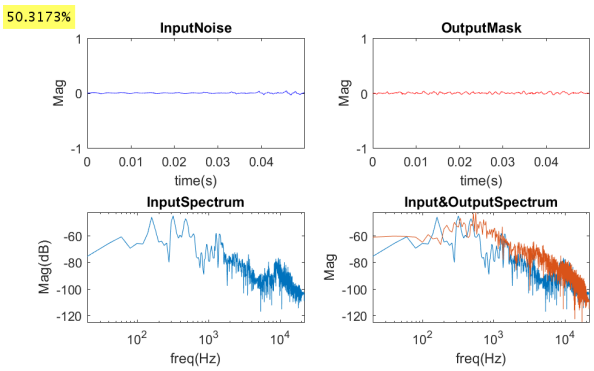
After observing the results of 10 test runs, the talk-through tests generally give a rather unreliable latency result, but the overall tests with the Sound Shield system delay yield a more consistent test result. It is inferred that the talk-through tests mainly cover the scheduling of the PC's I/O ports and are heavily influenced by the scheduling process, which is beyond the capability of testing team. The overall testing result on Sound Shield delay varied in the range of [116.93, 290.96] ms and averages about 227 ms which is close to the original goal of 185ms.

This average result of 227 ms constitutes the amount of time the user must wait to hear the mask's full response, given that the mask-level computation subsystem is limiting how fast the mask is allowed to respond to changes in the noise. On the other hand, we know that our system actually begins to respond to changes in the noise level within 46.44 ms because that is the scheduled callback rate within our software.

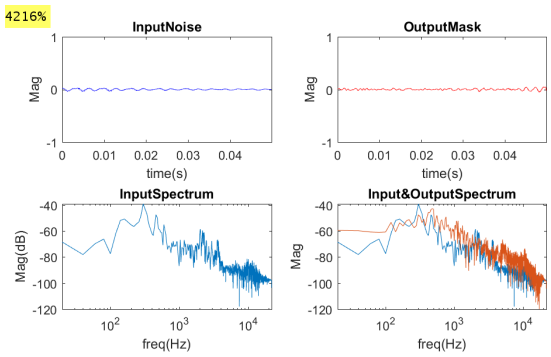
5.2 System Performance Test

Our system performance test closely followed our proposal in the testing and evaluation plan [4]. At any given time when the system is running, it is important to ensure that the synthesized output mask follows a similar frequency pattern to that in the input noise. The system performance test is designed to quietly listen to both microphone input and speaker output of the hardware system running Sound Shield. The presence of system-performance testing suite is hidden from Sound Shield's perspective.

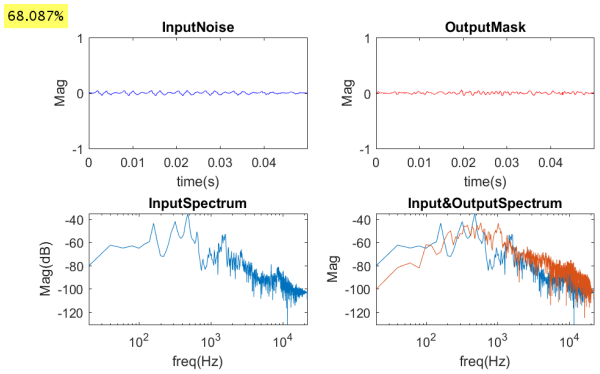
Similarly, the system performance test was coded to handle data logging of both audio input and output streams for post processing. A short time window of 50 ms, which corresponds to 2205 digital audio samples without overlapping, is used for computation of frequency analysis. Figure 9 below showcases several scenarios of the generated mask in response to the received intrusive noise. The order of the frames is from left to right and top to bottom: upon meeting the presence of intrusive noise, Sound Shield starts responding to the noise, in which the frame capture only shows about 45 percent success rate per frequency coefficients on mask performance, and gradually the percentage of success rate increases to about 80 percent upon fully covering the frequency coefficients across the spectrum. Also note that due to the limitation of our microphone hardware, whose dynamic range only reaches down to 50Hz, there is no input information for our system to tailor its mask response down to 20Hz.



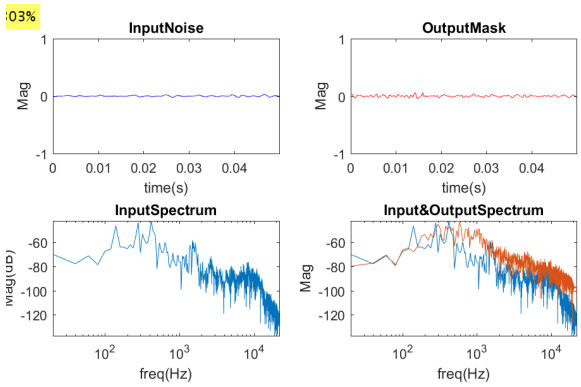
(a) Mask response at time instant, t0



(b) Mask response at time instant, t0 + Δt



(c) Mask response at time instant, t0 + 2*Δt



(d) Mask response at time instant, t0 + 3*Δt

Figure 9. Various Noise & Frequency Response

The difference between each time window, Δt is 50ms. As we have mentioned, our Sound Shield system creates an artificial delay by averaging its input of intrusive noise. Therefore, the system waited several packets of input audio noise before it responded to the intrusive noise. The impulses seen in the time-domain plot of input noise in Figure 10 below correctly shows that Sound Shield does not immediately respond to transient impulses that could potentially irritate the end user. The performance of Sound Shield is judged by differencing each frequency coefficient of the intrusive noise and that of the synthesized mask from the frequency analysis. Whether or not the difference in intensity is greater than 4 dB for the effectiveness in every frequency component depends on this grouping. In every frame of frequency analysis, it shows

the success rate, by percentage of successful coverage of synthesized mask on the intrusive noise. Out of the 20 test runs, the range of success rate falls between [66.6, 86.38] percent.

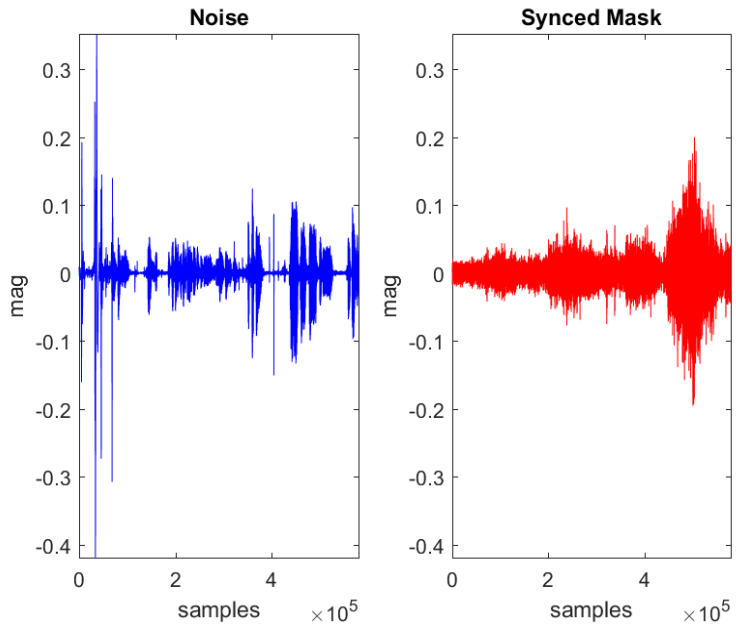


Figure 10. Time Domain Audio Signal Waveform

To summarize our test results, our system begins to respond to input noise after 46.44ms, which is the rate of the callback function that processes input and output buffers. However, due to our constraints on the mask-level computation subsystem that tells the software not to make the mask change abruptly, the user will notice the mask adapting to the noise within an average of 227 ms. While 227 ms is above our initial goal of reaching 185 ms, we believe that it is still an acceptable result because we arrived at this conclusion through calibration with our 15-frame averaging operation, which makes the mask retain a much more natural, soothing characteristic. Similarly, our test results show that the mask indeed adjusts its frequency spectrum to cover that of the noise by at least 4dB in every subband, except those which are out of reach of the microphone's frequency response. This result also demonstrates that the quality of the mask in its ability to adapt and cover the entire range of human hearing (20 Hz - 20 kHz) is limited by the quality of the microphone responding to the noise.

6.0 TIME AND COST CONSIDERATIONS

Our team has proposed a Gantt chart in the first semester, highlighting the main objectives to be accomplished over the 2015 - 2016 academic year. As we approach the end of the Spring 2016 semester, looking back, we have successfully accomplished our objectives within the proposed time frame. The main driving factors that allowed our group to achieve the deadlines were the willingness of our members to work as a group and dedication to the project, which allowed our group to meet during weekends, spring break, and winter break.

Since we designed Sound Shield as a purely software solution, this alleviated us from having to create a budget aside from the tools we bought for testing the software's accuracy and the Raspberry Pi. We want the end user to be able integrate Sound Shield on any audio setup that includes a pair of speakers and a microphone. Our team's job is to provide the best user-experience possible, only limited by the hardware of the end user's audio setup. Ultimately, the cost of Sound Shield is determined by the end user's choice of how much they want to spend on their audio setup.

Within our given time frame, we faced many different unexpected challenges and we effectively mitigated these problems by working in an agile fashion. As we immersed ourselves deeper into the problem, the number of bugs and issues we encountered grew. However, as an agile team, we practiced constant communication in order to properly discuss and resolve the problems as they arose. We thoroughly documented each issue in our lab notebook on Google Drive and used tables and backlogs to assign individuals to assess carry out each task.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

As with any engineering application destined for public use, it is important to weigh the potential benefits and risks associated with using the Sound Shield product. We believe that Sound Shield provides the great benefit of enhancing the quality of everyday life for those who use it. As demonstrated in this report, Sound Shield is able to transform unwanted noise into the soothing sounds of nature, enabling the user to remain focused on their work or keep resting, even in the presence of distracting noise outside their room. Such technology plays an important role for the good of our society: in an increasingly crowded and noisy world, Sound Shield helps safeguard

the user's privacy, personal space and productivity. This is especially important in cities where people's lives overlap in many ways. For example, a nurse working night shifts needs to rest during the day, while his or her family and neighbors are busy going about their lives and making noise. In such a situation, Sound Shield would greatly enhance the quality of life for the Nurse, enabling him or her to rest during the day and perform well at work.

Another benefit of Sound Shield is its potential to enhance the everyday lives of those who suffer from insomnia or tinnitus. Consumer Reports states that 43% of insomniacs surveyed achieved deeper sleep with the help of a white noise machine. Similarly, mynoise.net, a popular noise-generation website, features testimonials from users suffering from tinnitus who say that white noise masks help them achieve better sleep because it blocks their awareness of their tinnitus, which is perceived as a constant "ringing or buzzing" in the ears. Sound Shield could help those who suffer from these conditions in a similar way by preparing at atmosphere conducive to sleeping.

As with any marketable audio device, we must warn users not to turn up the volume to loud on their speakers. Although Sound Shield calibrates the mask intensity to that of the incoming noise signal, the user still has the ultimate control over the volume of the mask, through their speaker volume control. In order to avoid the risk of hearing damage, one should never turn up the speakers too loudly. In other words, if some noises in the environment are simply too loud to mask, one should not risk the health of their hearing in order to turn their speakers up so loud so as to mask the noise.

8.0 RECOMMENDATIONS

Throughout the process of designing Sound Shield, we researched and became familiar with the many concepts that constitute the Sound Shield prototype, including real-time digital signal processing, the psychoacoustics of the human ear, and user interface design. This research yielded many ideas that would greatly improve the quality of our software solution. However, due to time constraints and the limited scope of our design problem, we were not able to incorporate all of the features that we would have liked to include in our prototype. Therefore, in

this section of the report we will present these ideas and their underlying research as recommendations for the future improvement of the Sound Shield project.

The first feature that we would like to recommend is improved mask-spectrum controls in the user interface. Since our software dynamically alters the spectrum of the noise mask, it would be ideal to provide the user with some input to customize the overall tone of the output to their liking. For instance, one could design a knob to control the tone of the device in an intuitive way, so that the user could tell the software to output masks that are warmer or brighter in tone. Then, the software could still dynamically alter the mask spectrum, but keep it within the parameters specified by the user in the web interface.

We also recommend that a future iteration of Sound Shield redesign the filter bank and FFT bin-grouping algorithm to more closely match the psychoacoustic curve of the human ear's sensitivity. Our solution utilizes octave-spaced bands, and therefore gives greater precision to the lowest bands of the noise and output mask. However, in reality, the human ear has less resolution in the lowest and highest bands, instead containing its greatest resolution in the 800-4000 Hz range. For this reason, redesigning the filterbank and FFT bin-grouping algorithm to give greater precision in this range would increase the precision and accuracy with which the software could mask noises that fall within this middle range of the human ear and increase the effectiveness of the mask, given that these noises are most likely to disturb the user.

Another idea that was simply outside the scope of our problem was to design a feature that would allow the user to calibrate the software to room's frequency response. If such a customization feature were added, the masking algorithm would take into account the frequency response of the room and synthesize a mask creates the most soothing, natural sound possible for the user. In this way, the user could feel as if he or she is truly outside enjoying the sounds of nature because the room's frequency response has been removed from the mask.

A similarly difficult yet interesting problem to solve would be to allow the user to scale the software for multiple microphones and speakers in different locations around the room. This would require the software to know the location of each microphone and speaker, so that it knows the direction from which the noise is being emitted, and how loud it needs to play the

mask out of each speaker. Such a scalable implementation would allow Sound Shield to be used in office settings or larger rooms of a house, and would give the user a much more immersed experience in the mask.

Our final recommendation is to make is to package the software for distribution. Cmake, the tool with which we decided to compile and run Sound Shield, is capable of packaging the software for streamlined installation on another computer. However, this could be a very time-consuming project because one would need to either figure out how to include all of the necessary libraries utilized by our software or tell Cmake to automatically download them onto the user's computer while Sound Shield is being installed. Either way, the difficulty in this task would lie in maintaining the cross-compatibility between different operating systems.

The recommendations for improvements on our Sound Shield prototype can be grouped into two categories: enriching the user experience and creating a software package that is ready for distribution and easy installation. Since Negin, one of our team members, has decided to continue development on Sound Shield as her Digital Arts and Media capstone project, she will attempt to implement those ideas which have to do with improving the user interface. In our opinion, the rest of the recommendations would constitute good opportunities for future senior design teams to build on our prototype and create a more robust and effective noise-masking experience for the user.

9.0 CONCLUSION

In conclusion, this report details how Team 17 was able to successfully deliver a functional dynamic noise-masking software prototype. While this prototype still lacks the enhancements and packaging that would allow it to compete in the consumer noise-masking market, it fulfills all of the specifications given in our design problem, as well as some of our stretch goals.

The Sound Shield noise-masking solution is robust enough to accurately detect and adapt to changing noise frequencies in real time, yet delicate enough to avoid generating masks that are just as disruptive to the user as the noise itself. In other words, Sound Shield does not react so quickly to sudden noises so as to startle the user, but instead it gradually ramps its response

within the time requirements specified in our Design and Implementation Plan in order to cover the noise across all audible frequencies by at least 4dB in the most soothing manner possible. Moreover, when noise is not present in the environment, Sound Shield's mask returns to a low volume level which is barely noticeable, only ramping up its response again when noise intrudes upon the user's privacy.

The stretch goals which our team accomplished include a user interface that can access the software via a web page on any device sharing the same wifi connection as the software, as well as the ability to filter one's own custom masks and integrate them for use in Sound Shield. These features greatly enhance the user's experience of Sound Shield, enabling them to have more control over their noise-masking experience and enhance their rest, relaxation and productivity.

Finally, Team 17 has released Sound Shield's source code for free download online, making it available for personal use, as well as for continued community development. Any enhancements, added features and customizations that the public comes up with are welcome, with the hope that together we will create a truly unique noise-masking experience that can run in the background on anyone's personal computer. The final Sound Shield code repository is located at https://github.com/shjpark92/SoundShield and may be accessed for free by anyone. Moreover, comments, suggestions and questions may be sent to soundshieldUT@gmail.com and will be answered in as timely a manner as possible.

REFERENCES

[1] P. J. C. Audrey C. Younkin, "Determining the Amount of Audio-Video Synchronization Errors Perceptible to the Average End-User," *Broadcasting, IEEE Transactions on*, vol. 54, no. 3, pp. 623 – 627, 2008.

[2] "2015-10-30_Acoustics Seminar with David Nelson," *Google Docs*. [Online]. Available: https://docs.google.com/document/d/1FEj2xsBObKw2VPXsZWBTHuvki7dokimbPS3WMtmvUJA/edit?usp=sharing&pli=1&usp=embed_facebook. [Accessed: 02-Dec-2015].

[3] "Making Measurements with DAQ Hardware," The MathWorks, Inc. [Online]. Available: http://www.mathworks.com/products/daq/features.html [Accessed: 25-April-2016].

[4] SoundShield 2015/2016 DIP/TEP reports

[5] "Kaiser window," The MathWorks, Inc. [Online]. Available: http://www.mathworks.com/help/signal/ref/kaiser.html [Accessed: 23-April-2016].

APPENDIX A – Minimum Hardware Specifications

Table A1. Minimum Hardware Specification

Hardware	Tech Specification	Minimum Requirement	Description
Microphone	Sensitivity	-60dB +/-2dB	Electrical response at its output to a given standard acoustic input (1kHz at 94dB-SPL)
	Direction	Omni direction	Describes the pattern in which the microphone's sensitivity changes when the sound source changes position in space
	Signal-to-Noise Ratio	>60dB	Specifies the ratio of a reference signal to the noise level of the microphone output
	Dynamic Range	>60dB	Measure of the difference between the loudest and quietest SPL to which the microphone responds linearly
	Frequency Response	50Hz - 20kHz	Describes its output level across the frequency spectrum
	Total Harmonic Distortion	>90dB	Measurement of the level of distortion on the output signal for a given pure tone input signal
	Power Supply Rejection	>90dB	Indicate a microphone's ability to reject noise present on the power supply pins from the signal output
	Acoustic Overload Point	>90dB	Sound pressure level at which the THD of the microphone's output equals 10%
Speaker	Sensitivity	-80dB +/12dB	Electrical response at its output to a given standard acoustic input (1kHz at 94dB-SPL)
	Frequency Response	50Hz - 20kHz	Describes its output level across the frequency spectrum
	Signal-to-Noise Ratio	>60dB	Specifies the ratio of a reference signal to the noise level of the microphone output
	Dynamic Range	>60dB	Measure of the difference between the loudest and quietest SPL to which the microphone responds linearly
	THD+N	>40dB	Measurement of the amount of unwanted impurities in a given signal
	IMD+N	>40dB	Measurement of all impurities in an amplified audio signal that are not harmonically related to the source signal
	Stereo Crosstalk	>40dB	Measurement of how much an output signal on one channel crosses into separate output channel
	Impedance	8 ohms	AC resistance of the loudspeaker to the audio signal from the speaker
	Power Rating	10W subwoofer, 5W speaker	RMS or peak value a loudspeaker can handle before destroying the speaker
Processor	CPU Clock speed	700Mhz	Amount of instructions that could be completed in one second
	CPU # of Core	1	Number of instructions that could run together in one instance
	RAM	512MB	Read Access memory or runtime memory (including OS)
	extra free storage space	4GB	Storage Space (including OS)
	Connectivity	802.11b or Ethernet	Internet Connectivity
	DAC	12 bits	Digital to Analog Conversion
	ADC	12 bits	Analog to Digital Conversion
	Operating System	Linux, OS X	Unix based