A FREQUENCY-DOMAIN METHOD FOR ACTIVE ACOUSTIC CANCELLATION OF KNOWN AUDIO SOURCES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Ryan D. Rocha

June 2014

COMMITTEE MEMBERSHIP

TITLE:	A Frequency-Domain Method for Active Acoustic Cancellation of Known Audio Sources
AUTHOR:	Ryan D. Rocha
DATE SUBMITTED:	June 2014
COMMITTEE CHAIR:	Dr. Wayne Pilkington, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	Dr. Tina Smilkstein, Ph.D. Assistant Professor of Electrical Engineering
COMMITTEE MEMBER:	Dr. Xiaozheng Zhang, Ph.D. Associate Professor, Associate Dept. Chair, Graduate Coordinator of Electrical Engineering

ABSTRACT

A Frequency-Domain Method for Active Acoustic Cancellation of Known Audio Sources

Ryan D. Rocha

Active noise control (ANC) is a real-time process in which a system measures an external, unwanted sound source and produces a canceling waveform. The cancellation is due to destructive interference by a perfect copy of the received signal phase-shifted by 180 degrees. Existing active noise control systems process the incoming and outgoing audio on a sample-by-sample basis, requiring a high-speed digital signal processor (DSP) and analog-to-digital converters (ADCs) with strict timing requirements on the order of tens of microseconds. These timing requirements determine the maximum sample rate and bit size as well as the maximum attenuation that the system can achieve. In traditional noise cancellation systems, the general assumption is that all unwanted sound is indeterminate. However, there are many instances in which an unwanted sound source is predictable, such as in the case of a song. This thesis presents a method for active acoustic cancellation of a known audio signal using the frequency characteristics of the known audio signal compared to that of a sampled, filtered excerpt of the same known audio signal.

In this procedure, we must first correctly locate the sample index for which a measured audio excerpt begins via the cross-correlation function. Next, we obtain the frequency characteristics of both the known source (WAVE file of the song) and the measured unwanted audio by taking the Fast Fourier Transform (FFT) of each signal, and calculate the effective environmental transfer function (degradation function) by taking the ratio of the two complex frequency-domain results. Finally, we attempt to recreate the environmental audio from the known data and produce an inverted, synchronized, and amplitude-matched signal to cancel the audio via destructive interference. Throughout the process, we employ many signal conditioning methods such as FIR filtering, median filtering, windowing, and deconvolution. We illustrate this frequency-domain method in Native Instruments' LabVIEW running on the Windows operating system, and discuss its reliability, areas for improvement, and potential future applications in mobile technologies. We show that under ideal conditions (unwanted sound is a known white noise source, and microphone, loudspeaker, and environmental filter frequency responses are all perfectly flat), we can achieve a theoretical maximum attenuation of approximately 300 dB. If we replace the white noise source with an actual song and the environmental filter with a low-order linear filter, then we can achieve maximum attenuation in the range of 50-70 dB. However, in a real-world environment, with additional noise and imperfect microphones, speakers, synchronization, and amplitude-matching, we can expect to see attenuation values in the range of 10-20 dB.

Keywords: Digital Signal Processing, Active Noise Control, Discrete Filtering, Frequency Domain, FFT.

ACKNOWLEDGMENTS

I want to formally thank my family and friends for supporting me throughout my collegiate career, especially my parents, Rob and Char, who raised me to believe that I could achieve success through hard work and determination. I would also like to thank my thesis advisor Dr. Wayne Pilkington for meeting with me from week to week to go over my progress and providing suggestions and an expert opinion on the topics of digital signal processing. I also want to thank him for teaching me nearly all I know about signal processing and inspiring and engaging me in his undergraduate lectures.

Furthermore, I want to thank my thesis committee members Dr. Tina Smilkstein and Dr. Xiaozheng Zhang, who, in addition to Dr. Pilkington, taught me valuable information in the field of Electrical Engineering. In fact, it was in Dr. Smilkstein’s EE-521 course where I first formulated and tested the theory that this thesis builds upon. I also learned various image processing techniques in Dr. Zhang’s classes that I applied to digital audio processing in this thesis.

Lastly, I want to thank the institution of California Polytechnic State University, San Luis Obispo for providing me with a thorough education in Electrical Engineering and challenging me to apply myself to achieve my fullest potential.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Problem Statement

The cancellation of unwanted noise has and will always be an important challenge that acoustic engineering and digital signal processing can help address. Those who have a stake in the success of noise-cancellation systems include the major audio technology companies, engineers, consumers, and even healthcare providers. Studies have linked excess noise exposure with health risks such as hearing impairment, hypertension and ischemic heart disease, annoyance, sleep disturbance, and decreased school performance [1]. Today, noise cancellation is still considered a luxury, as most people would rather accept and deal with unwanted noise than pay a premium price for a system to reduce it.

Perhaps the most popular and well-known consumer products to feature noise cancellation are noise-canceling headphones. Many noise-canceling headphones employ two forms of noise reduction: passive and active. While the physical materials of the headphone itself passively attenuate noise above 500 Hz [2], an isolated microphone, digital signal processor (DSP), and headphone loudspeaker produce a canceling waveform to actively attenuate noise via destructive interference, as shown in **Figure 1-1**. To achieve substantive noise cancellation (better than 9 dB of attenuation), the unwanted noise signal and cancellation waveform must be matched with an amplitude accuracy within ±3 dB and a phase accuracy within ±20° simultaneously [3]. This method of active noise control (ANC) requires a high-speed DSP and adaptive algorithms that can be computationally demanding, and its expected performance is typically up to 25 dB of attenuation at low frequencies for a well-calibrated system [2]. The existing ANC method assumes that all unwanted noise is random or unknown, and thus the cancellation procedure requires a fast real-time architecture to sample, process, and produce a canceling waveform within strict timing requirements. Additionally, this method of ANC is not well-suited for changing, dynamic noise such as music.

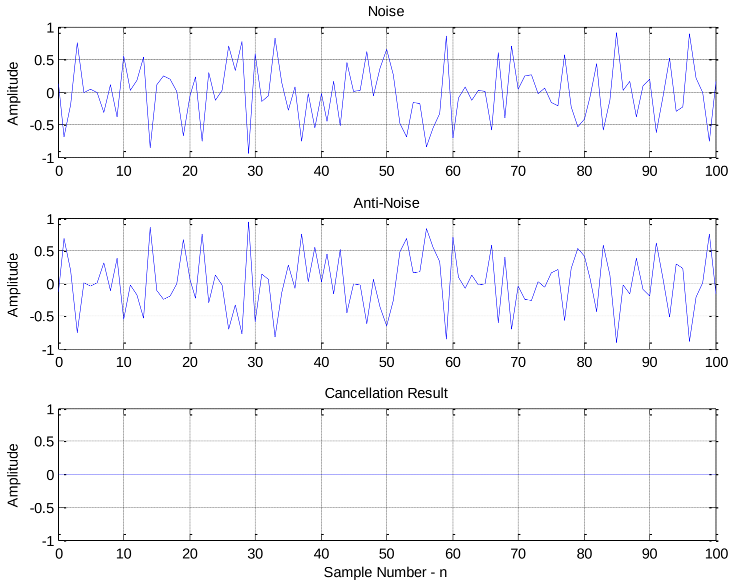


Figure 1-1: Active Noise Control (ANC) via destructive interference.

This project investigates an alternative noise-cancellation procedure in which the unwanted noise source is assumed to be known, as is in the case of a song. Advancements in song recognition technology make it possible to identify an unwanted song, and with the appropriate microphones, loudspeakers, and signal processing, it is possible to create an acoustic cancellation system in which processing occurs in chunks in “pseudo-real-time” on a non-real-time operating system such as Windows.

The ultimate goal would be to leverage technological items that many people already have – a computer, mobile phone, or tablet with a fast processor, and a good set of loudspeakers – to create an inexpensive acoustic cancellation system for known audio sources. The only additional item required would be a high-quality measurement microphone that could interface to the processing platform. This project seeks to create a system prototype that is affordable (compared to top-of-the-line active noise-canceling headphones, *i.e.* less than $300), and achieves similar cancellation performance to the active noise attenuation of ANC headphones (not including passive noise reduction, or at least 10 dB).

1.2 Project Overview

In this project, we will use National Instruments' LabVIEW running on the Microsoft Windows operating system to acquire a microphone signal of an unwanted song source, process the signal, and play back a canceling waveform through a loudspeaker. LabVIEW is well-suited for this project because it is a very visual, intuitive, and system-oriented programming language that is designed for rapid prototyping. LabVIEW also contains many built-in signal processing functions that allow us to focus more on the system architecture and less on recreating known signal processing algorithms. Although Windows is not a real-time operating system and thus is not the ideal platform for a "pseudo-real-time" noise-cancellation system, it offers some advantages: 1) It runs the basic version of LabVIEW, 2) it allows us to take advantage of the powerful multi-threaded, high-performance processors and PCs that run on Windows, 3) it allows a Windows PC to simultaneously be both a design tool and a system prototype, and 4) it allows for quick and simple interfacing with peripherals such as loudspeakers and USB microphones.

The basic procedure is as follows: First, a measurement microphone interfaced to a PC via USB records an excerpt of the unwanted song playing in the environment. We will assume that the song source has already been identified and its WAVE file data has been acquired. One example of how this could be accomplished now would be to use a song recognition application such as Shazam or Soundhound to identify the song, and ping a database to download the song file (copyright issues may apply). To minimize the complications of additional variables as much as possible, we will perform the test on a single channel (mono). With the song properly identified and its full WAVE file data acquired, we perform a cross-correlation algorithm in LabVIEW to approximate the starting location (sample index) of the excerpt within the original WAVE file, all while keeping track of the amount of time elapsed during the processing. With the approximate location found, we perform a Fast Fourier Transform (FFT) on the measured excerpt and its corresponding WAVE file excerpt to obtain a transfer function between the measured (mic recording) and actual (WAVE file) frequency responses. Assuming the transfer function represents the frequency response of the loudspeaker playing the unwanted sound coupled with an environmental filter, we apply the transfer function to a future segment of the WAVE file via frequency domain multiplication, take the Inverse Fast Fourier Transform (IFFT), and phase-shift the time-domain waveform by 180° to produce the canceling waveform. Lastly, we accumulate future playback values in a buffer, and stream the canceling waveform to the soundcard for cancellation.

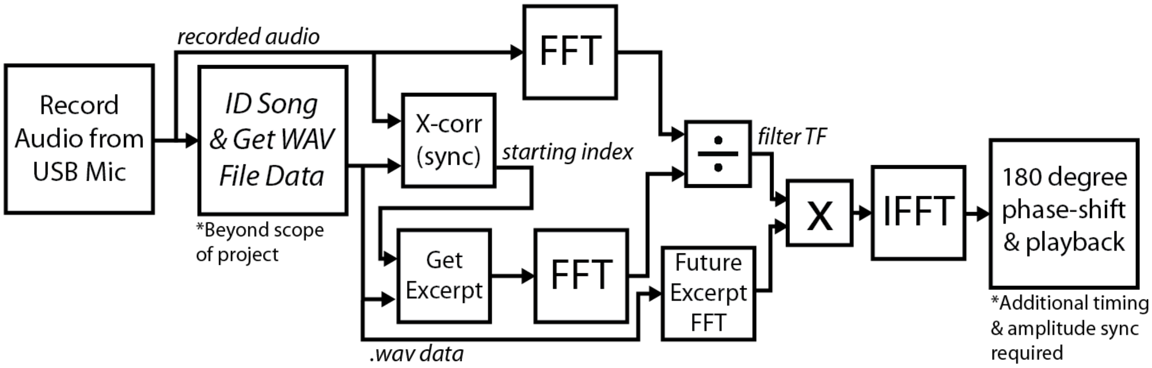

recorded audio
Record Audio from USB Mic
ID Song & Get WAV File Data
*Beyond scope of project
FFT
X-corr (sync)
starting index
÷
filter TF
Get Excerpt
FFT
Future Excerpt FFT
X
IFFT
180 degree phase-shift & playback
*Additional timing & amplitude sync required
.wav data

Figure 1-2: Basic block diagram of frequency-domain cancellation method.

During playback, the system requires additional synchronization steps: We record from the microphone again and sweep the playback index to find the location of optimal cancellation for a final timing synchronization. We can perform a similar procedure to sweep the amplitude of the playback signal to find the optimal volume for the cancellation waveform, but it is more desirable to eliminate the amplitude variable by calibrating the system beforehand to produce a 1:1 input to output audio level. In the ideal case – *i.e.* the song is easily recognizable and non-repetitive, the microphone and loudspeakers have perfectly flat frequency responses, and the transfer function impulse response behaves like a low-order linear filter – theory shows that this system can perform very well. Chapter 2 will provide a detailed proof of concept test to illustrate this best-case real-world scenario as well as an optimal scenario in which the noise source is [known] white noise and the environmental filter is perfectly flat. Chapter 4 will provide a detailed documentation of the LabVIEW system implementation.

In its final polished form, a noise-cancellation system using the frequency-domain method proposed in this project could have many potential applications and end-uses for consumers. For instance, this system could be used in a bedroom to cancel the noise (music) from the party next door so that the user can get a good night’s sleep. This system could also be used in mobile technologies in conjunction with music recognition software to cancel an unwanted song or even assist ANC headphones with supplementary cancellation.

1.3 Comparison between Existing and Proposed Concepts

The frequency-domain method has a key advantage over the existing ANC method in that it can accurately predict the future time-domain and frequency-domain values of the noise signal in advance. The predictive ability of the proposed method effectively makes it a non-causal system, which would be impossible to achieve with a real-time control system. From this key advantage, we can expect the frequency-domain method to produce better broadband attenuation performance over the entire audible frequency range as long as the cancellation waveform is properly synchronized in the time domain and its frequency response characteristics are adequately matched. We can also expect it to produce better cancellation of unwanted music.

Furthermore, the existing methods of ANC require ultra-fast analog-to-digital converters (ADCs) and digital signal processors (DSPs) comprising a control system which acquires and processes data in real time. The proposed frequency-domain method can avoid the need for high-speed DSPs to produce the cancellation signal by processing the future cancellation waveform in bigger sections at a slower rate. It is worth noting that the proposed frequency-domain method of acoustic cancellation is still technically "real time," in that there are still timing requirements that must be met for proper operation, but these processing-time constrains are on the order of milliseconds to seconds instead of microseconds.

One advantage that the current real-time ANC method has over the proposed method is that its timing is deterministic, which makes timing synchronization a much easier task. For the frequency-domain method running on a non-real-time system, the processing time will be variable, depending on system resources, which makes perfect synchronization more difficult to achieve.

Table 1-1 summarizes the aforementioned advantages and disadvantages of the existing ANC method and the proposed frequency-domain method.

Table 1-1: Comparison of existing ANC methods vs. frequency-domain method.

Category	**Existing ANC Method**	**Freq. Domain Method**
Processor requirements	High speed, high power DSP chip required	Dedicated DSP chip not required – Processor can be PC, mobile phone, or tablet processor
Timing	Real-time processing restricts calculations to be within microseconds (<30μs) [2]	Piecewise processing allows for more lenient constraints (milliseconds to seconds)
Reliability	Real-time DSP produces deterministic timing	Non-real-time operating system results in variable processing time and inconsistent results
Low-Frequency (<400 Hz) Attenuation	10−25 dB [4]	10−70 dB
Wide Frequency Attenuation	0−10 dB [4]	5−20 dB
Cost (mid-range)	>$300	~$200
Source Best At Canceling	Any constant, random noise signal (low frequency)	Known audio signal (such as a song)

CHAPTER 2: BACKGROUND

2.1 Current Approaches to Noise Cancellation

The current approach to noise cancellation is to use headphones as both a passive and active noise-canceling device. The passive attenuation behaves like a low-pass filter, greatly attenuating frequencies above a few hundred Hertz. The active attenuation behaves like a high-pass filter, performing well at frequencies below a few hundred Hertz, but not at higher frequencies. Together, both forms of attenuation provide sufficient noise cancellation over the entire audible frequency range.

2.1.1 Passive Attenuation

The existing noise-cancellation method relies heavily on passive attenuation for noise reduction above a few hundred Hertz. In a headphone system, the headset shell and earphone cushions provide a seal between the user's ear and the noisy environment. These physical materials absorb high-frequency noise in the vibrations of the shell. **Figure 2-1** shows a simple illustration of a passive noise system [4].

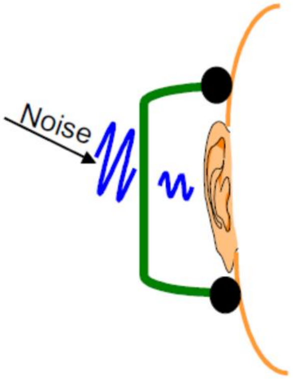


Figure 2-1: Passive noise system used in noise-canceling headphones [4].

This passive attenuation system behaves as a second-order mechanical system characterized by the following equation:

$$ATT = \frac{K_v}{K_v + K_c + j\omega R - \omega^2 M} \qquad \textbf{Eq. 2-1}$$

where M is the mass of the shell, R is the cushion damping, and K_v and K_c are the stiffness of the air in the shell volume and cushion respectively [4]. We observe that the attenuation is inversely proportional to the mass of the shell and the cushion damping, and that the attenuation is better at higher frequencies. **Figure 2-2** shows a simulated passive filter attenuation curve described by **Eq. 2-1**, with a 6-dB cutoff frequency of 300 Hz [4].

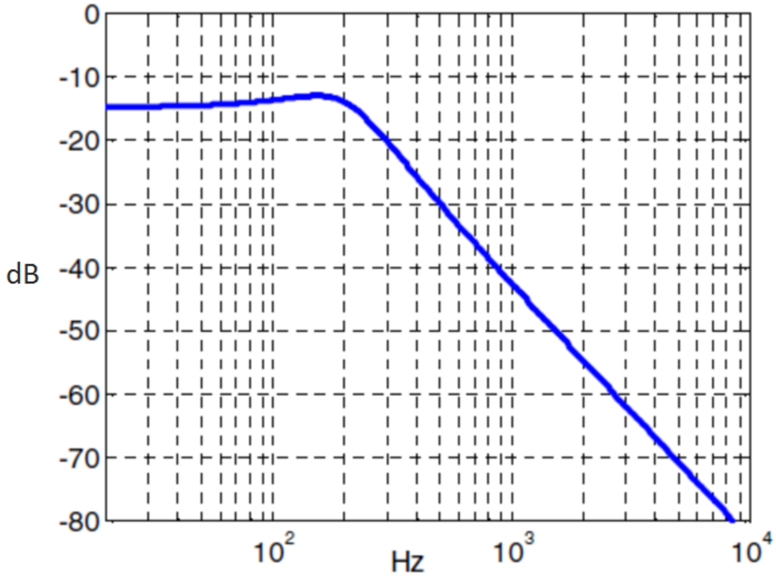


Figure 2-2: Simulated typical passive attenuation [4].

2.1.2 Active Attenuation

Active attenuation within a noise-canceling headphone relies on a control system that continuously monitors the incoming noise and seeks to minimize an error signal. Active noise-cancellation systems feature two different control system architectures: feedback and feedforward [4] [3] [5]. **Figure 2-3** shows the simplified diagrams of each control method [3].

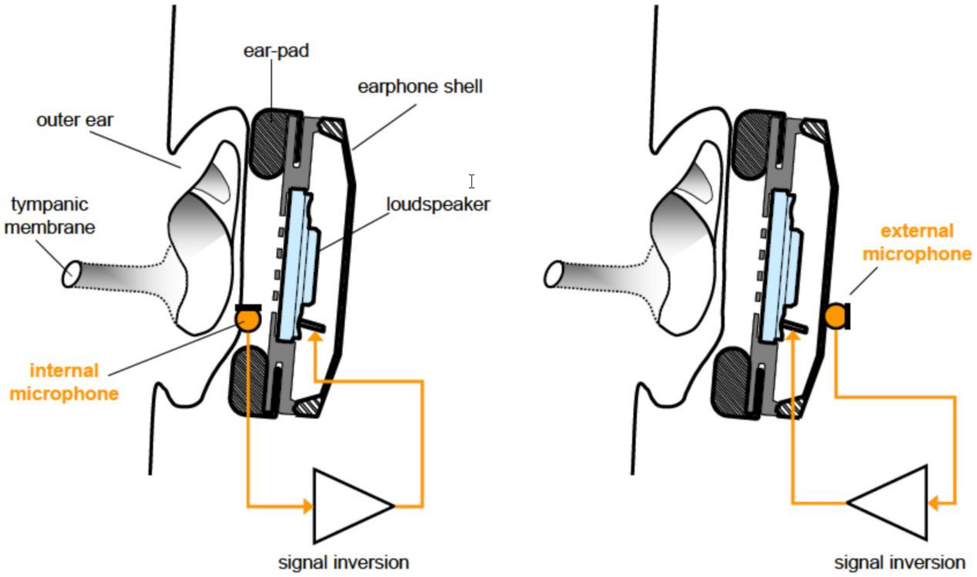


Figure 2-3: Feedback (left) and feedforward (right) noise-cancellation configurations [3].

A. Feedback Method

The feedback control system uses an internal microphone that is located directly in front of the headphone loudspeaker and is coupled to the loudspeaker by a negative feedback control loop [3]. **Figure 2-4** shows the basic block diagram for a negative feedback analog control system. Here, *P* is the plant (response from the loudspeaker to the microphone output), *C* is the analog controller, *d* is the disturbance signal (noise), and *e* is the error signal [4].

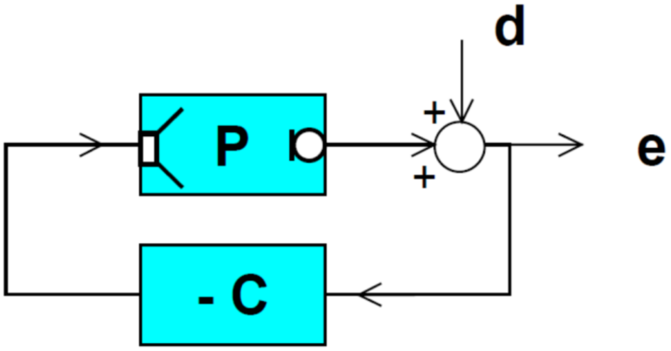


Figure 2-4: Feedback control diagram [4].

Eq. 2-2 shows the frequency-domain representation of the error signal.

$$e = d\frac{1}{1+CP} \qquad \textbf{Eq. 2-2}$$

Eq. 2-3 describes the response of the closed-loop system.

$$S = \frac{1}{1+CP} \qquad \textbf{Eq. 2-3}$$

Figure 2-5 shows a simulation of a typical magnitude response of active attenuation using feedback control [4]. We observe from **Figure 2-5** that frequencies below 500 Hz are attenuated by more than 10 dB, while frequencies above a few kHz show no attenuation. The controller amplifies frequencies in the transition region near 1 kHz, because according to [6], the Bode integral must be equal to zero.

$$\int_0^\infty \log|S(\omega)|d\omega = 0 \qquad \textbf{Eq. 2-4}$$

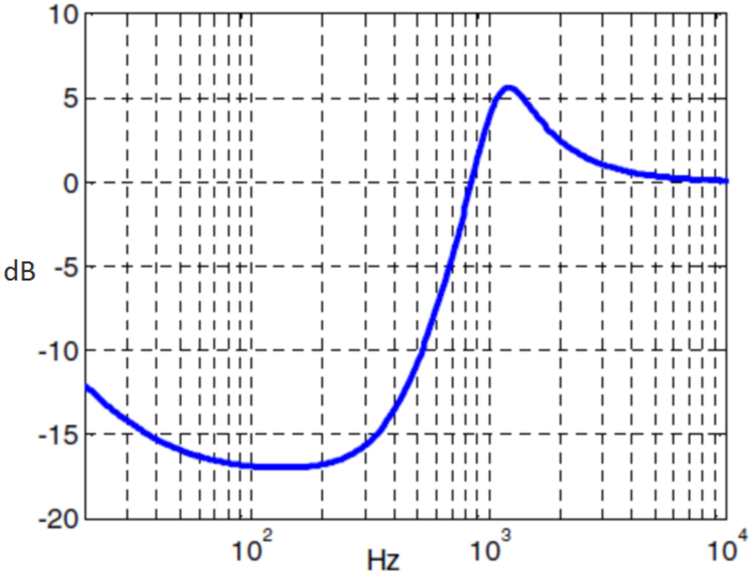


Figure 2-5: Simulated typical active attenuation using feedback method [4].

B. Feedforward Method

The feedforward method is the most popular ANC method used in modern active noise-canceling headphones [5]. Contrary to the feedback approach, the feedforward method uses an external microphone to detect a reference of the noise, which gets inverted and added to the headphone drive signal, while an internal error microphone tunes the digital controller (adaptive filter) [4] [3]. **Figure 2-6** shows the feedforward control diagram where *W* is the adaptive filter [4].

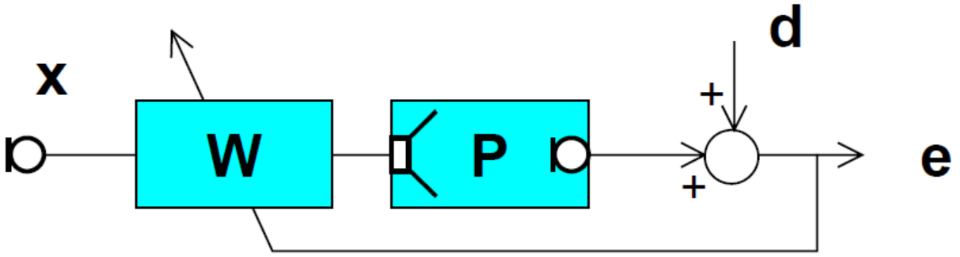


Figure 2-6: Feedforward control diagram [4].

The goal of the adaptive filter *W* is to minimize the mean-squared error of the signal from the internal error microphone. Some well-known algorithms used to achieve this goal include: LMS (Least Mean Squares), Filtered-x LMS, and Adjoint LMS, with the main differences between algorithms being their rates of convergence and computational efficiencies [5] [7].

One drawback of both digital feedback and feedforward control is that they are both subject to additional delay due to sampling delay in the DSP and DAC, and the phase delay of the low-pass filters [4] [3]. If the total electric delay exceeds the acoustic delay from the reference microphone to the loudspeaker, then the optimal filter will be non-causal, and prediction will be required to attenuate broadband signals [4]. **Figure**

2-7 shows how this delay can affect performance by limiting both the control bandwidth and attenuation level of an active noise-cancellation system [4].

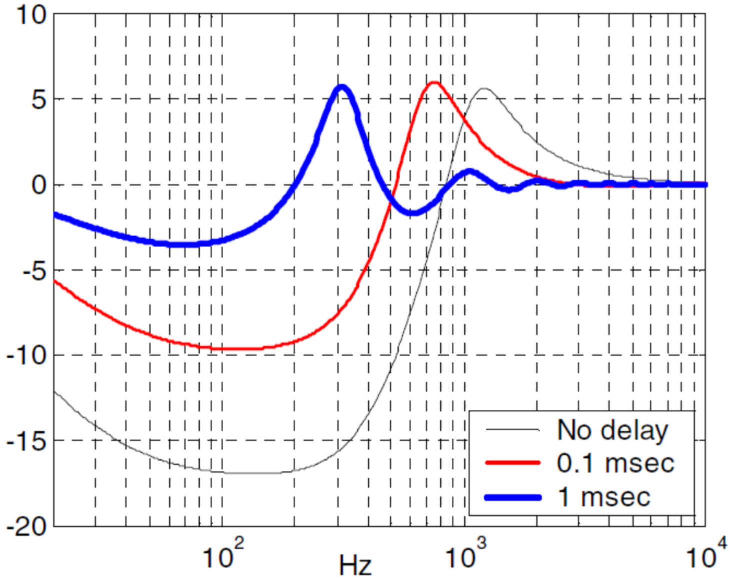


Figure 2-7: Attenuation with a feedback controller for a plant with additional delay of 0.1 msec and 1 msec [4].

Figure 2-7 shows that to achieve best performance, the delay must be minimized. Using a very high sampling frequency can minimize the delay but this will generally require powerful DSP processors and increased cost [4]. Typically, only band-limited or predictable signals can be successfully attenuated. Performance is thus limited to narrow band or tonal noise when using conventional DSP systems [5].

2.2 Frequency-Domain Method: Proof of Concept

Here, we discuss the original proof of concept and the fundamental theory behind the frequency-domain method of active noise control introduced in this report. We demonstrate two test examples in LabVIEW of the basic procedure using the parameters listed in **Table 2-1** for both an actual song and white noise (known, not random) as the unwanted noise sources. The goal of Example 1 is to describe the key steps in the procedure and provide results for an ideal real-world implementation. In Example 2, we show the theoretical optimal performance of the system.

Table 2-1: Parameters for frequency-domain method in Examples 1 and 2.

Parameter	Example 1: Song	Example 2: White Noise
Sample Rate	44.1 kHz	44.1 kHz
Bit Depth	16-bit	16-bit
Channels	1-Mono	1-Mono
Noise Source Frequency Response	Not all frequencies well represented	All frequencies strongly represented
Noise Source Autocorrelation	1 defined autocorrelation peak with several smaller peaks (due to repetitiveness of beat structure of song)	1 strong autocorrelation peak
Recorded Sample Length	Half of original song length	Half of original signal length
Microphone Frequency Response	Perfectly flat over all frequencies	Perfectly flat over all frequencies
Loudspeaker Frequency Response	Perfectly flat over all frequencies	Perfectly flat over all frequencies
Environmental Filter Frequency Response	Low-order, linear band-pass filter response	Perfectly flat over all frequencies

2.2.1 Example 1: Song as Unwanted Noise Source

To make the basic explanation of this method more clear and concise, we will define two key terms: 1) we will refer to the known audio track data (from a song file) as the *original signal*. We use a 10 second (approximate) excerpt of Michael Jackson's "Beat It" as an example of an original signal, shown in **Figure 2-8** below.

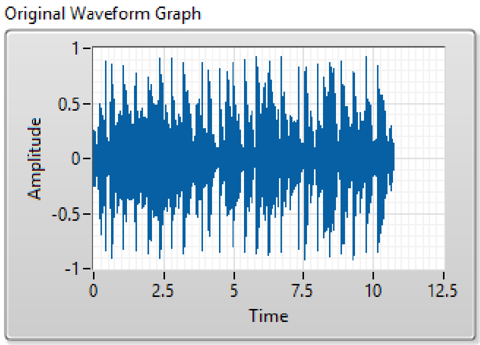


Figure 2-8: Excerpt of "Beat It" by Michael Jackson, used as the original signal.

2) We will refer to the sampled and filtered version of the original signal as the *sampled signal*. The sampled signal is assumed to be a filtered version of the original signal since the speakers and environment act as synthetic and natural filters to frequencies in the audible range of 20 Hz to 20 kHz. For this example, we acquire the sampled signal with an ideal microphone (noiseless, distortionless, perfectly flat frequency response). **Figure 2-9** shows an example of a sampled signal, which is a filtered and attenuated version of the original signal from **Figure 2-8**. Here, we will assume that the music has been recognized correctly and the original signal data has been fetched from a database and is ready for processing.

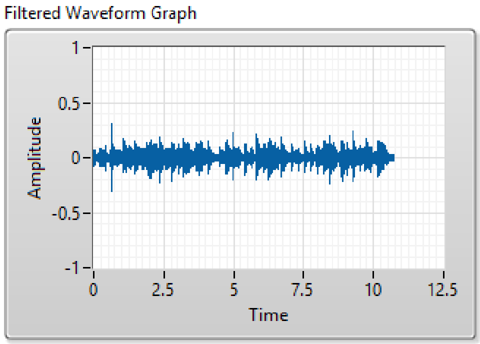


Figure 2-9: Filtered version of waveform in **Figure 2-8**, used as sample signal.

The following list illustrates the basic cancellation procedure using the frequency-domain method:

A. Determine Sample Location within Original Signal

In order to properly determine the transfer function, or frequency response, of the environmental filter, we must first locate the starting index of the sampled signal within the original signal. We achieve this with the cross-correlation function, which is a convolution-like function used to determine the similarity between two waveforms. We can find the cross-correlation mathematically using

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f^{*}[m] \cdot g[n+m] \qquad \textbf{Eq. 2-5}$$

We can determine the cross-correlation in LabVIEW using the built-in correlation virtual instrument (VI). **Figure 2-10** below shows the result of the correlation of the original signal shown in **Figure 2-8** and the first half of the sampled signal shown in **Figure 2-9**.

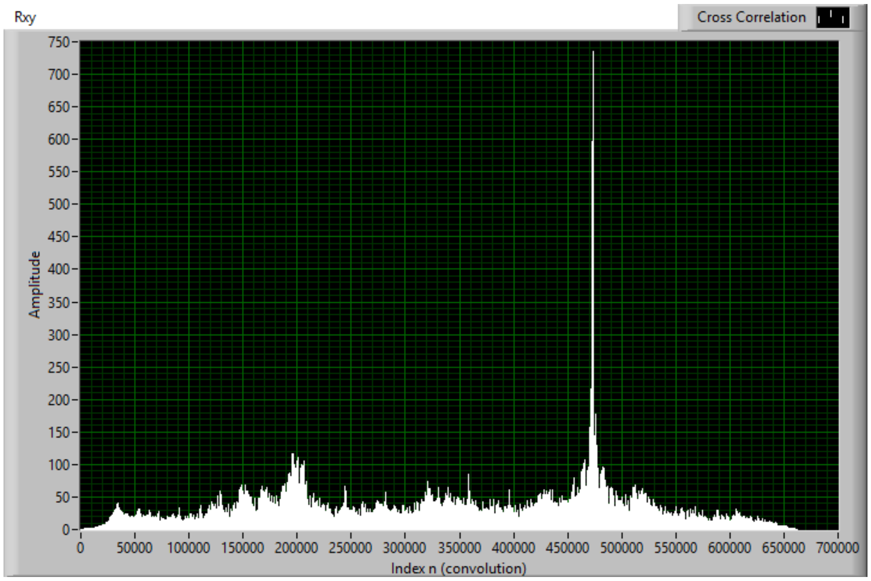


Figure 2-10: Cross-correlation of original and sampled signals in LabVIEW.

Figure 2-10 shows a clear spike at the index near *n* = 475000. To find the index of the original signal that corresponds to the starting index of the sampled signal, we use

$$i = n - m. \qquad \textbf{Eq. 2-6}$$

In this equation, *i* is the index of the original signal that corresponds to the sampled signal's starting index, *n* is the index at which the maximum cross-correlation value occurs, and *m* is the length (total number of samples) of the original signal. This method of locating and synchronizing waveforms must be optimized in order to synchronize waveforms as accurately as possible, since the cancellation capabilities are highly dependent on proper alignment.

B. Get Frequency Content of Each Signal

Next, we take the Fast Fourier Transform of each signal (both the sampled signal and the located excerpt of the original signal). **Eq. 2-7** shows the equation for the Discrete Fourier Transform (DFT) of a signal. The FFT is an efficient algorithm to calculate the DFT.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi k \frac{n}{N}} \text{ for k = 0,…, N -1.} \qquad \textbf{Eq. 2-7}$$

As in the previous step, we use built-in LabVIEW VIs to calculate the FFT for each signal. **Figure 2-11** and **Figure 2-12** show the FFTs of the original and sampled audio waveforms from **Figure 2-8** and **Figure 2-9**. We plot each FFT in the frequency domain over the audible frequency range (20 Hz−20 kHz).

In a product implementation of this system, we should choose to acquire a sampled signal of length 2^n because the FFT is optimized for signals that are powers of 2 in length.

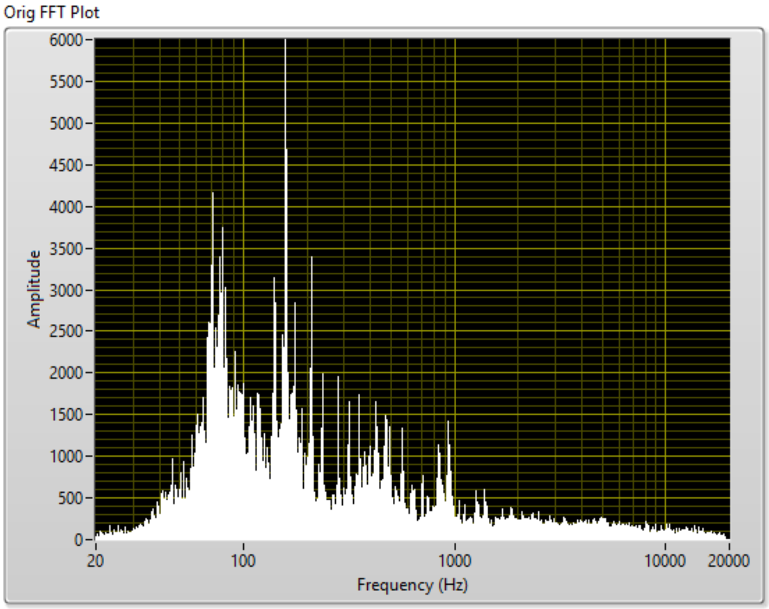


Figure 2-11: FFT of original signal.

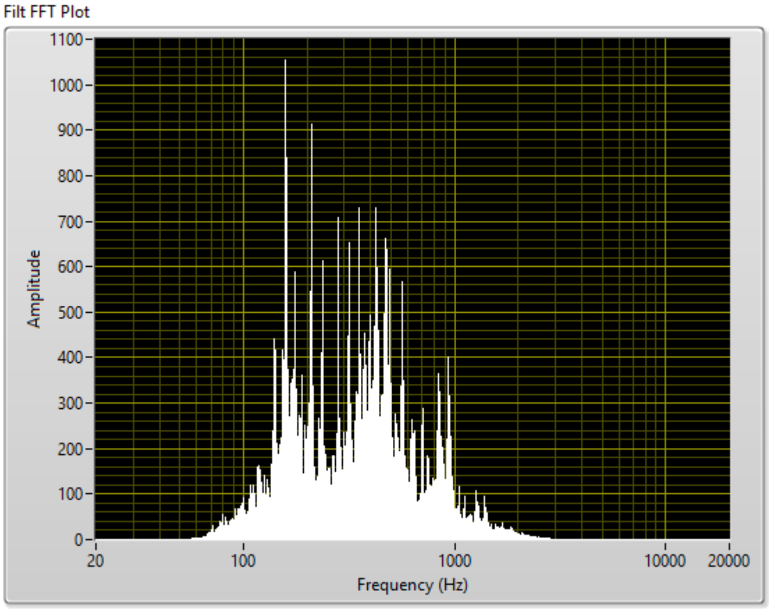


Figure 2-12: FFT of sampled signal.

Figure 2-12 shows the FFT of the entire filtered version of the original signal. However, only the first half of the original signal will be used to take the FFT. We show the full length FFTs so that we can compare the filter transfer functions of the full length signals to that of the half-length signals.

C. Determine and Condition the Filter Transfer Function

We can easily determine the effective filter response applied to the original signal by the sound source and intervening environment by dividing the sampled signal's complex FFT by the original signal's FFT. It is important to maintain both the magnitude and phase information for each frequency; and so, you cannot simply divide the magnitude responses to obtain the effective filter response. **Figure 2-13** shows the result of the FFT division from the full-length data in **Figure 2-11** and **Figure 2-12**. The resulting filter shows a band-pass response centered at approximately 450 Hz.

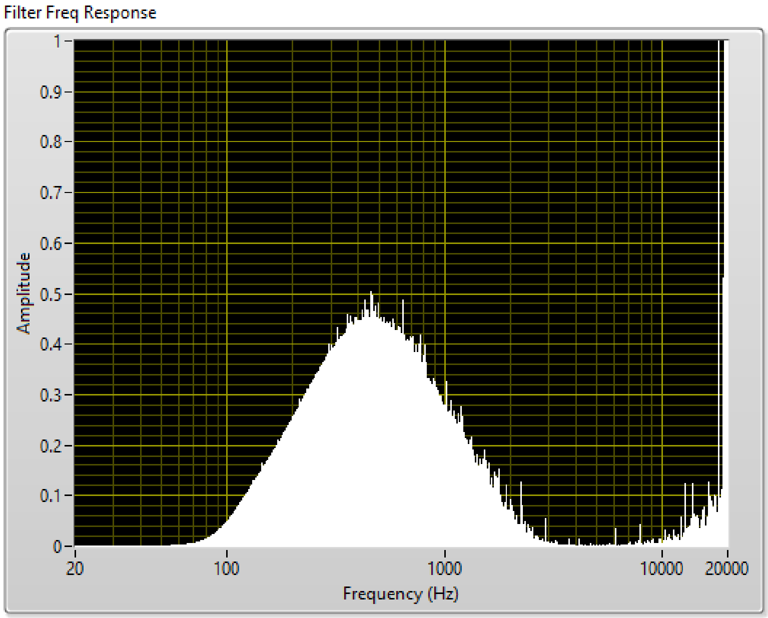


Figure 2-13: Full-length filter frequency response.

Figure 2-14 shows the filter derived from a sample signal whose length is half of the original sample, and whose index starts at 0 (the beginning of the original signal).

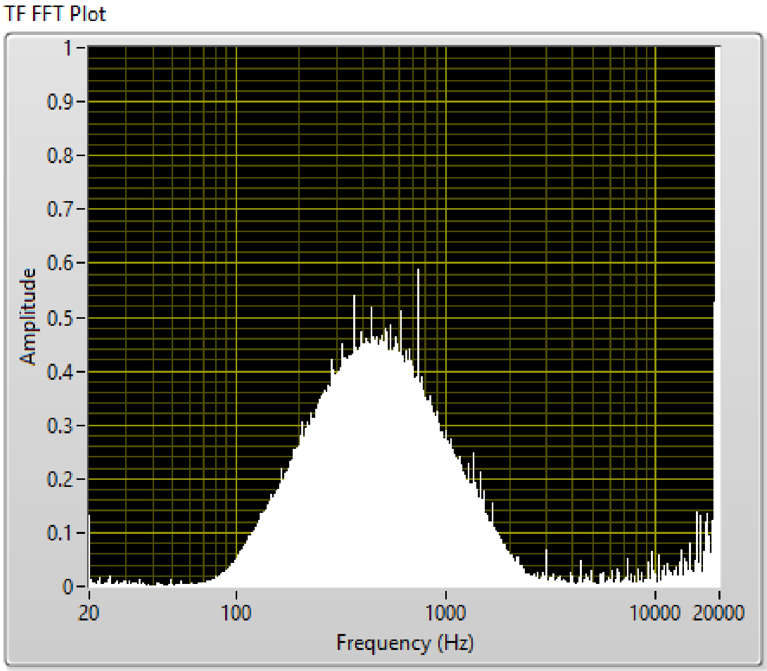


Figure 2-14: Filter frequency response from first half of the total sample.

We can see from **Figure 2-13** and **Figure 2-14** that significant noise exists on the filter magnitude responses. If we assume that environmental filters are relatively smooth in nature, we can apply a low-order median filter to the resulting filter FFT values to reduce the noisy spikes on the output. Although median filtering is a non-linear process, it is shown to be more effective at limiting the impulse-like noise on the transfer function without “blurring” the time-domain response. **Figure 2-15** shows the result of passing the data from **Figure 2-14** through a median filter of size 10.

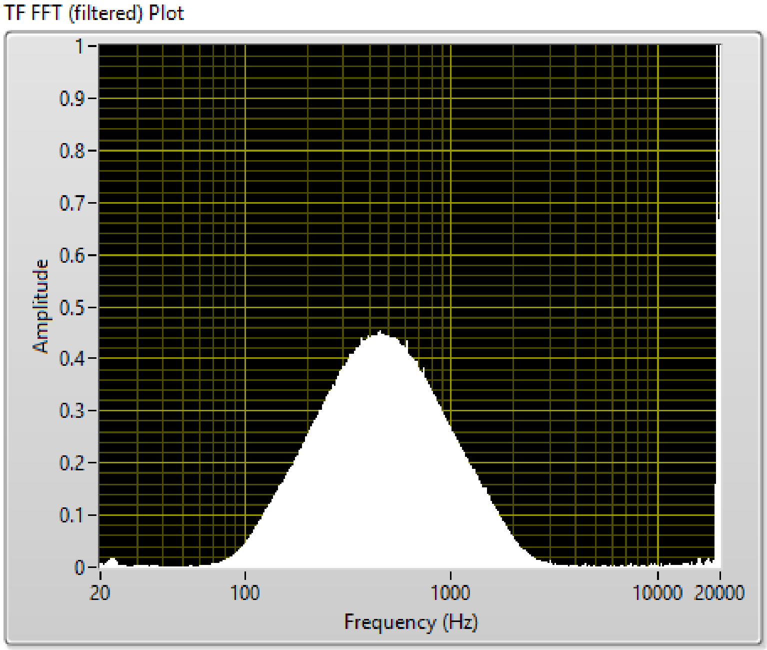


Figure 2-15: median-filtered version of **Figure 2-14**.

D. Recreate & Cancel the Filtered Signal from the Determined Filter Response

Now that we have the effective environmental filter response, we can apply the filter to an equal-length future segment of the original signal. We do this by taking the FFT of future values of the original signal and multiplying the resulting FFT by the environmental filter response FFT values. Because we are performing frequency-domain multiplication, it is extremely important that the lengths of both FFT results are the same. The length of the example filter FFT in **Figure 2-15** is half of the original sample length (~5 seconds out of ~10 seconds), which means we can recreate a filtered version of the second half of the original signal for cancellation. In **Figure 2-16** below, the second half of the waveform from **Figure 2-9** is almost perfectly recreated by multiplying the FFT of the equivalent portion of the signal from **Figure 1-1** and the filter FFT from **Figure 2-15**, and then taking the inverse FFT of the result.

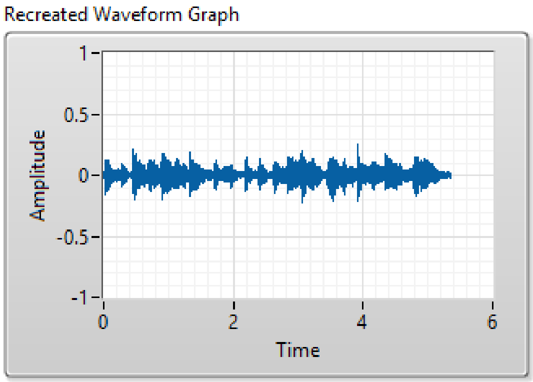


Figure 2-16: Recreated ~5 sec. portion of waveform in Figure 2-9 (2nd half).

Once we have recreated the expected values of future sampled data, we can invert the waveform (180° phase shift) and output the data to destructively interfere with the future sampled values. **Figure 2-17** shows the result of adding the second half of the waveform data in **Figure 2-9** (which has yet to be played) to the inverted waveform in **Figure 2-16**. We can see that this method nearly perfectly cancels the unwanted audio signal.

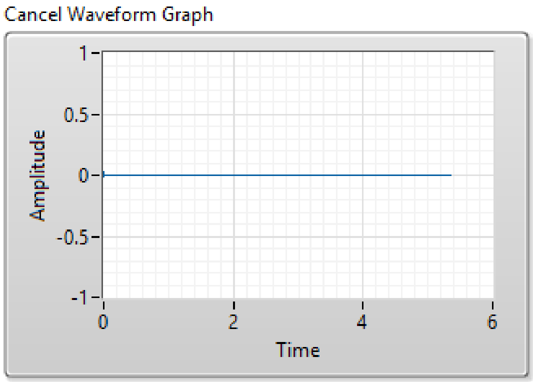


Figure 2-17: Canceled ~5 sec. future segment of unwanted audio.

Figure 2-18 below shows the attenuation in dB of the canceled waveform over the audible frequency range. Since the environmental filter behaves like a band-pass filter centered at 450 Hz, we see the best attenuation of roughly 70 dB near 450 Hz.

Conversely, we see no attenuation at the frequencies that the environmental filter had filtered out (below 50 Hz and above 2.5 kHz). This makes sense from an intuitive standpoint because if the sampled signal is already void of frequency content above 2.5 kHz and below 50 Hz, then it is already attenuated at those frequencies and an ANC signal will not help to further attenuate those frequencies.

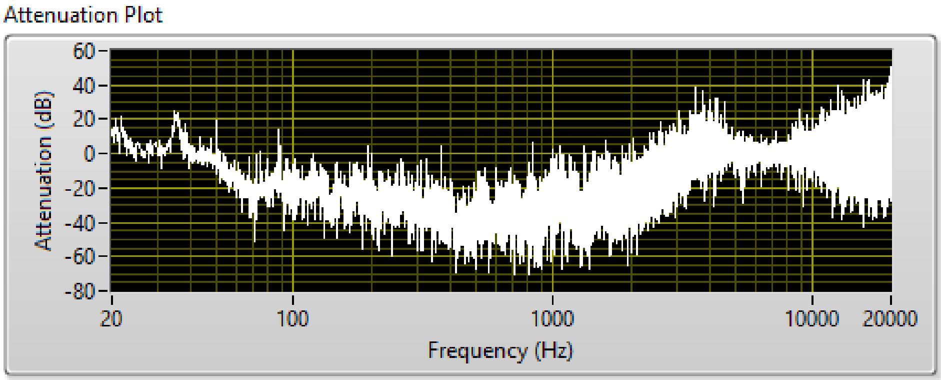


Figure 2-18: Attenuation in dB of the canceled waveform.

2.2.2 Example 2: White Noise as Unwanted Noise Source

This example uses a 6-second white noise signal as the unwanted noise source. We will use the same white noise signal for both the original and sampled signals, effectively simulating a perfect environment. **Figure 2-19** shows the FFT of the white noise signal. Note that because the white noise source is time-limited, the magnitude response of the FFT is not equal-valued for all frequencies.

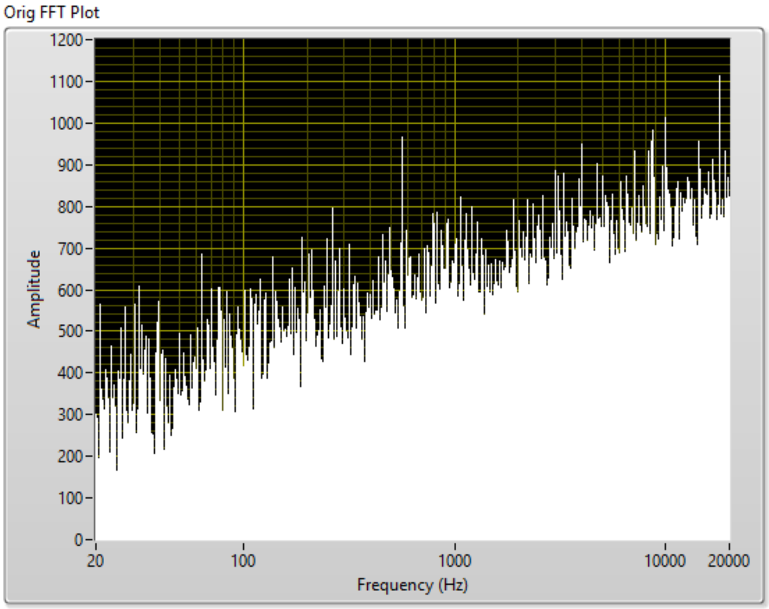


Figure 2-19: FFT of white noise source (original and sampled).

Here, obtaining the transfer function of the environmental filter is trivial, since the two waveforms are identical. The resulting transfer function will have a magnitude of 1.0 and a phase of 0 for all frequencies. When we apply this transfer function to the original signal, we create a perfect copy of the noise signal to be canceled, which we phase-shift by 180° and play back for cancellation.

With perfect synchronization, we can attenuate the noise by approximately 300 dB, as shown in the attenuation plot in **Figure 2-20**.

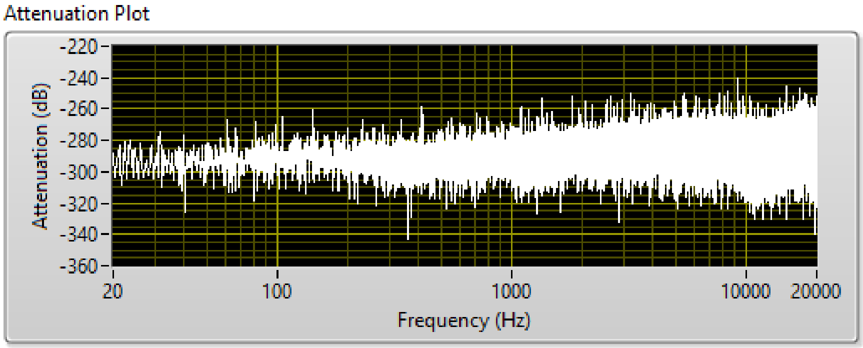


Figure 2-20: Attenuation plot of known white noise source under ideal conditions.

Any residual noise is due to quantization error and rounding error from floating point calculations within LabVIEW, which is insignificant because the human ear would not be able to distinguish it from ambient noise.

As previously mentioned, the plot in **Figure 2-20** is the attenuation result in the case of perfect synchronization. **Figure 2-21** shows the attenuation vs. frequency when the cancellation waveform is 1, 2, 3, and 4 samples out of sync (at $f_s = 44.1$ kHz). For a 1-sample misalignment, the best theoretical attenuation reduces to 50 dB (at 20 Hz) and further reduces logarithmically by approximately 20 dB/decade as the frequency increases. Increasing the offset effectively shifts the y-intercept (referenced to 20 Hz) up on the plot in **Figure 2-21**. The notches at 14.7 kHz and 11 kHz are due to the fact that the sample rate is 44.1 kHz and therefore 14.7 kHz is one full period out of phase (360°) for an offset of 3, and 11 kHz is one full period out of phase for an offset of 4. Further increasing the offset will introduce more notches, as the offset and frequency of one period required for complete cancellation are inversely related.

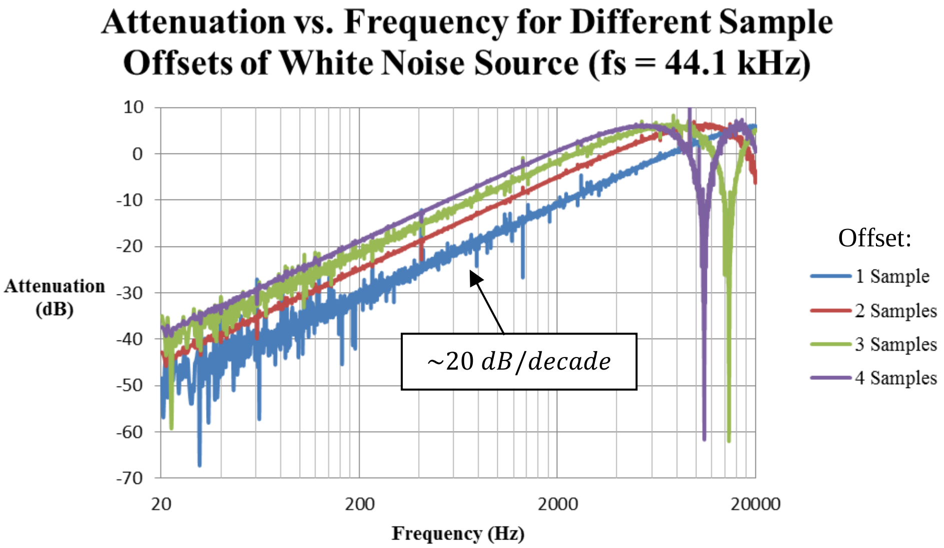


Figure 2-21: Attenuation vs. frequency for different sample offsets of white noise source at 44.1 kHz sample rate.

The frequency-domain method described here is very robust as long as we properly synchronize the original and sampled signals and acquire a sufficient length sample. Due to the high sensitivity to time-alignment, we must continue improving and developing the synchronization methods, since the cross-correlation method does not always produce an exact alignment.

The minimum sample length required varies from song to song, as each song contains unique frequency content over time. However, tests show that sample lengths down to 2 seconds in length at a 44.1 kHz sample rate still produce significant attenuation. Increasing the sample size not only increases the number of frequency bins of the FFT, but also the odds that more frequencies in the audible range are represented within that sample. As the sample length increases, the system will not only take longer to begin the cancellation process, but the algorithm itself will take longer. Thus, we must simultaneously maximize the representation of frequencies in the audible range to increase the reliability of this frequency-domain cancellation method, while minimizing the length so that we can start the cancellation procedure quicker and perform all necessary computations efficiently.

CHAPTER 3: DESIGN REQUIREMENTS

3.1 Introduction

This project seeks to create a system prototype that is affordable compared to existing high-performance active noise-canceling headphones, *i.e.* less than $300), and achieves similar cancellation performance to the active noise attenuation of ANC headphones without microphone isolation. Since the microphone and loudspeaker will not be isolated and neither the microphone nor loudspeaker will be ideal devices, the goal is to achieve at least 10 dB of noise reduction.

Again, the aim is to leverage widely-available technological items such as a personal computer and loudspeakers to create an inexpensive, almost do-it-yourself acoustic cancellation system for known audio sources. The only additional item required is a high-quality measurement microphone that can take accurate acoustic measurements.

3.2 Acoustic Measurements

This sound canceling method relies heavily on precise acoustic measurements. If these measurements do not accurately represent the real-world environmental sound, then all future signal processing computations and algorithms based on these inaccurate measurements will neither be reliable nor useful. Thus, the project requires a high-quality measurement microphone. Electret condenser microphones (ECM) are the most commonly used microphone type for measurement purposes due to their stability and their wide and flat frequency response characteristics [8] [9]. ECMs can also achieve reasonable signal-to-noise ratios and good maximum sound pressure levels, although distortion and a high noise floor can be an issue. The two main disadvantages of ECMs are that 1) they require a large DC bias voltage (usually phantom power at +48V) and (2) they must be coupled directly to an amplifier because of their low capacitance [8]. Most importantly, ECMs are relatively inexpensive devices, with quality microphones in the price range of $50-$100. The best (and most expensive) measurement microphones cost several thousands of dollars and these would be the ideal devices for this project. However, the goal of the project is to not only to create a sound cancellation system that performs well, but one that is also affordable. Therefore, we must make some sacrifices in quality with a less expensive microphone.

3.2.1 Dayton Audio EMM-6 Electret Measurement Microphone

To compromise price and quality as best as possible, this project utilizes the Dayton Audio EMM-6 electret condenser measurement microphone for acoustic measurements, shown in **Figure 3-1** below. **Table 3-1** on the following page shows the microphone requirements compared to the EMM-6 datasheet specifications.

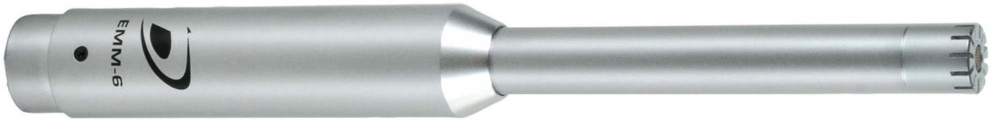


Figure 3-1: Dayton Audio EMM-6 Electret Measurement Microphone.

Table 3-1: Microphone requirements compared to the Dayton Audio EMM-6 datasheet specifications [10].

Parameter	**Microphone Requirements**	**Dayton Audio EMM-6**
Capsule type	electret condenser	6 mm electret condenser
Polar pattern	Omnidirectional	omnidirectional
Frequency response	20 Hz – 20kHz	18 Hz - 20 kHz
Impedance	200 Ω	200 Ω between pins 2 and 3
Sensitivity at 1 kHz into 1 kΩ	<10 mV/Pa	10 mV/Pa (-40 dBV, re. 0 dB = 1V/Pa)
Max. SPL for 1% THD @ 1000 Hz	>120 dB	127 dB
S/N ratio	>70 dB	70 dB A-weighted
Connector	USB	gold plated XLR
Power	phantom power	phantom power: +15 V to +48 V
Weight	Any	144 grams
Price	<$100	$79.99

The EMM-6 offers many advantages over other similarly-priced measurement microphones. First, the microphone boasts a very flat frequency response from 18Hz to 20 kHz with a sensitivity deviation within 2dB of the 1 kHz value over the entire frequency range [10]. Furthermore, Dayton Audio provides a file of the calibration data attributed to each microphone’s serial number. **Figure 3-2** shows the frequency response calibration data for the microphone used in this project (serial number 6521), which is located on Dayton Audio’s website. The frequency response appears to be reasonably flat over the entire frequency range, with more variation in the sensitivity occurring at frequencies above 2 kHz. **Table 3-2** summarizes key values from **Figure 3-2**.

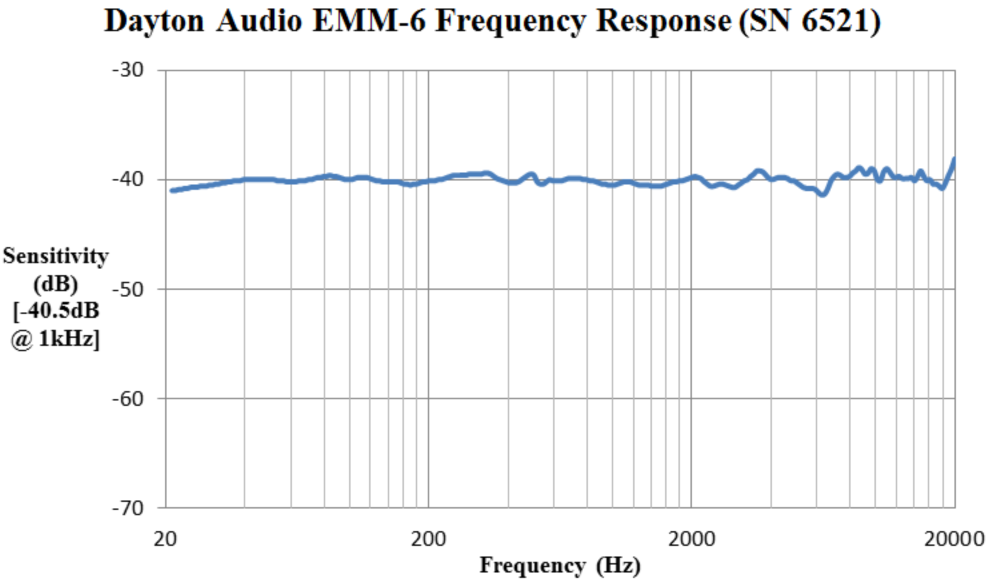


Figure 3-2: Frequency Response of Dayton Audio EMM-6 Electret Measurement Microphone.

Table 3-2: Dayton Audio EMM-6 SN6521 key frequency response values.

Parameter	**Sensitivity**	**Frequency**
Sensitivity at 1 KHz into 1 kΩ	-40.5 dBV	1 kHz
Maximum Sensitivity	-38.1 dBV	20 kHz
Minimum Sensitivity	-41.4 dBV	6.24 kHz

3.2.2 Blue Icicle XLR to USB Microphone Converter/Preamp

The Dayton Audio EMM-6 microphone has an XLR connector and requires phantom power. This project necessitates a USB microphone interface to a PC and therefore requires an XLR-to-USB converter which supplies phantom power. In this project, we use the Blue Icicle converter and preamp shown in **Figure 3-3** [11].

Figure 3-3: Blue Icicle XLR to USB Microphone Converter/Preamp [11].

Figure 3-4 shows the connection diagram provided by Blue Microphones on how to properly interface an XLR, phantom-powered microphone to a PC via USB [11].

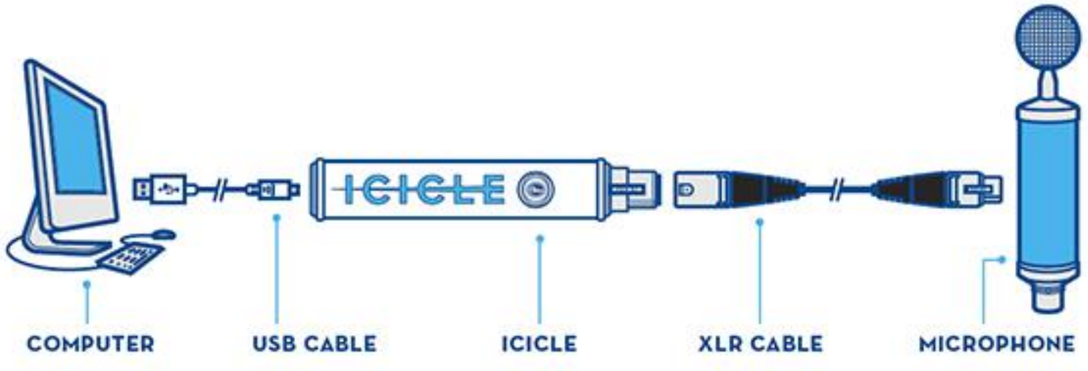


Figure 3-4: Connection diagram for Blue Icicle interface between mic and PC [11].

According to the Blue Icicle datasheet, it uses a 44.1 kHz sample rate with a bit depth of 16 bits. It requires 200 mA from the USB bus to supply 48 V phantom power to the microphone [11].

3.3 Loudspeaker

The ideal loudspeaker for this project has a perfectly flat frequency response from 20 Hz to 20 kHz and has no distortion at any frequency in that range. As with the microphone, we cannot expect to achieve ideal characteristics with an affordable device. However, this project will attempt to overcome the shortcomings of non-ideal loudspeakers with inverse filtering. First, we will characterize the frequency response of the loudspeaker using a frequency sweep source. An inverse filter attempts to "undo" the frequency response of the loudspeaker by boosting the underrepresented frequencies and attenuating the overrepresented frequencies.

In theory, inverse filtering can perfectly equalize the frequency response of the loudspeaker. In practice, however, the process will most likely introduce distortion because the magnitude of the inverse filter response goes to infinity as the measured magnitude response approaches zero. One solution is to use Wiener deconvolution, which can compensate for these instances where the inverse filter response goes to infinity by introducing a term in the denominator of the filter [12]. Unfortunately, if the loudspeaker physically cannot reproduce particular frequencies, then no amount of frequency response correction will be able to achieve a flat response. We will further discuss the performance of inverse filtering in this project in Chapter 5.

3.3.1 M-Audio AV-30 Reference Monitor Loudspeakers

This project will use the M-Audio AV-30 reference monitor loudspeakers for audio playback. These are mid-range studio monitors designed to reproduce accurate sound with a relatively flat frequency response for under $100. The main drawback of the AV-30 is its low-frequency response. We can attempt to improve the low-frequency response with a subwoofer. See **Table 3-3** below for the loudspeaker requirements and **Figure 3-5** for the AV-30 frequency response according to [13].

Table 3-3: Loudspeaker requirements compared to the M-Audio AV-30 datasheet specifications [13].

Parameter	**Loudspeaker Requirements**	**M-Audio AV-30**
Frequency response	50 Hz – 20kHz	90 Hz – 20kHz
RMS SPL	100 dB @ 1 meter	101.5 dB @ 1 meter
S/N ratio	>80 dB	>90 dB A-weighted
Input connectors	1/8” audio jack	Left and right RCA line input and 1/8" aux input
Price	<$100	$79.99

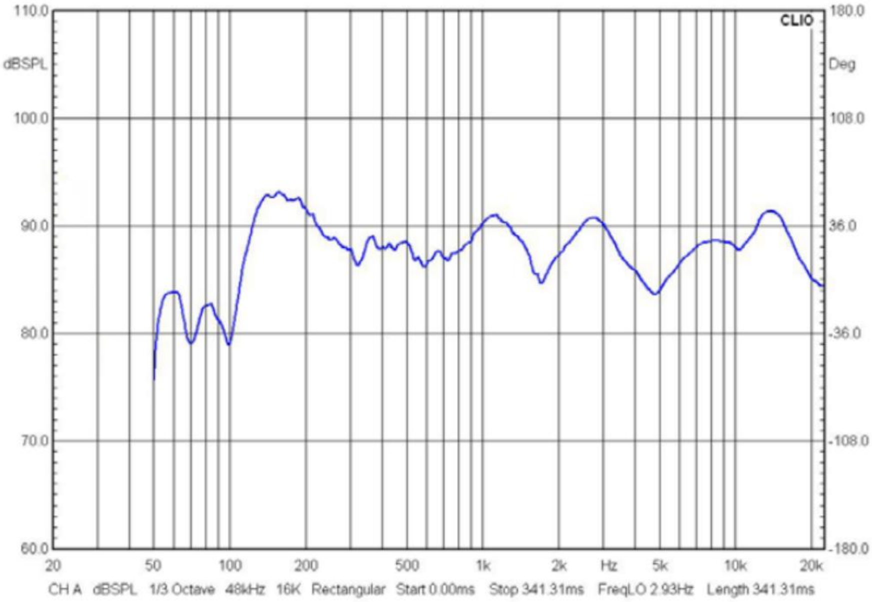


Figure 3-5: Typical frequency response of M-Audio AV-30 reference monitor loudspeakers according to [13].

CHAPTER 4: IMPLEMENTATION

This chapter will provide a detailed block-by-block discussion of the project implementation in LabVIEW according to the diagram shown **Figure 4-1** below. We will highlight important features and LabVIEW functions, and explain the pros and cons of the chosen parameters.

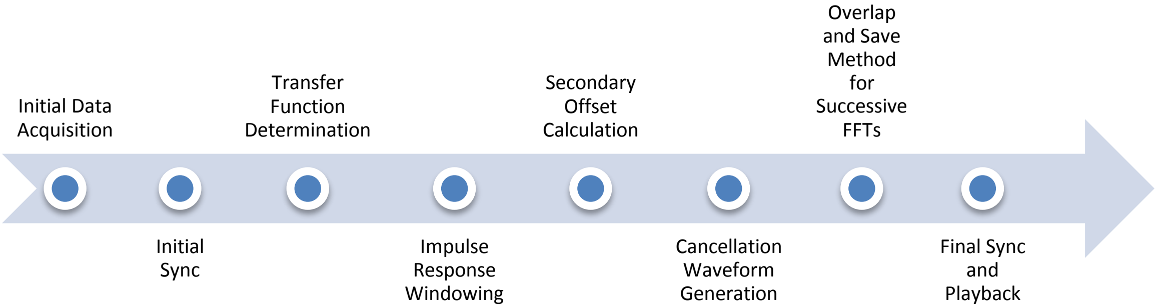

Initial Data Acquisition
Initial Sync
Transfer Function Determination
Impulse Response Windowing
Secondary Offset Calculation
Cancellation Waveform Generation
Overlap and Save Method for Successive FFTs
Final Sync and Playback

Figure 4-1: High-level system flow diagram.

4.1 Initial Data Acquisition

4.1.1 Overview

The first block in the system block diagram is the data acquisition block. This block records a segment of the unwanted audio via a microphone and sends the measured data to the next block. In this project, we make use of LabVIEW's built-in Sound VIs for microphone input. The three main sub-VIs required for microphone data acquisition are "Sound Input Configure.vi," which configures the soundcard to record audio from a specified microphone source at a desired sound format; "Sound Input Read.vi" which reads the input data and stores the values to a waveform; and "Sound Input Clear.vi" which stops acquisition of data and clears the buffer. See APPENDIX A for more information on the inputs and outputs of low-level VIs used in this project. Combined together, these three sub-VIs create a custom VI (function) called “Record From Mic.vi,” shown in **Figure 4-2**. This VI will record a specified number of samples of audio from the microphone and output a 1-dimensional array of wave values.

The function “Acquire Microphone Data (6sec).vi” shown in **Figure 4-3** records approximately 6 seconds of audio at 44.1 kHz and 16 bits per sample. It also initializes a microsecond timer, which we will use later for synchronization purposes.

4.1.2 LabVIEW Block Diagrams

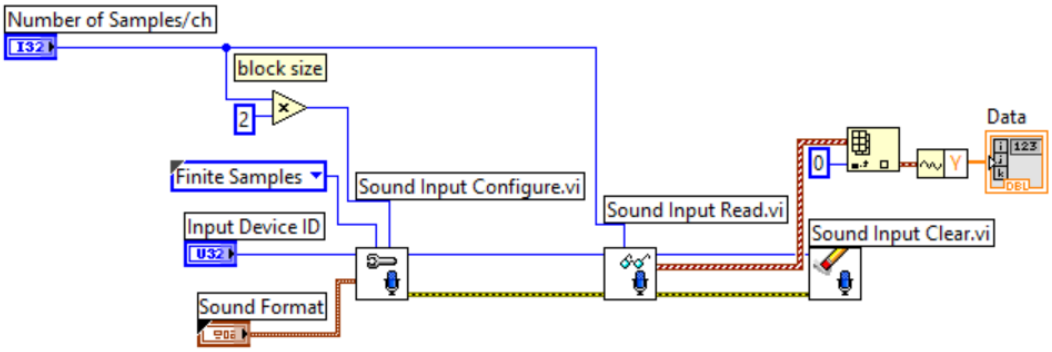


Figure 4-2: Record From Mic.vi block diagram.

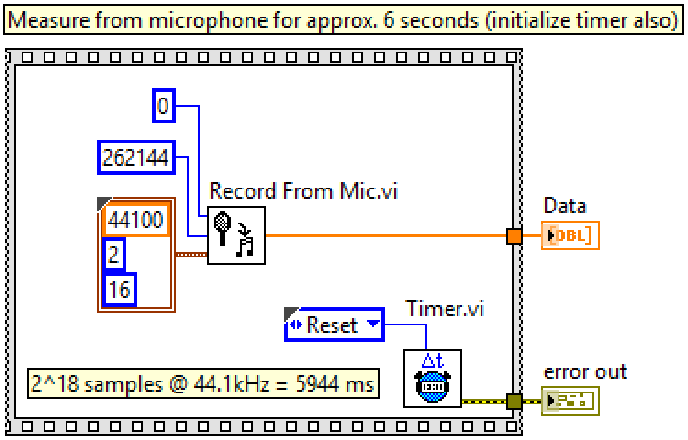

Measure from microphone for approx. 6 seconds (initialize timer also)
0
262144
44100
2
16
Record From Mic.vi
Data
DBL
Reset
Timer.vi
2^18 samples @ 44.1kHz = 5944 ms
error out

Figure 4-3: Acquire Microphone Data (6sec).vi block diagram.

4.1.3 Parameters Chosen

For the sound format, we choose a sample rate of 44.1 kHz and a bit depth of 16 bits per sample because these are the values compatible with the Blue Icicle [11]. As mentioned earlier, this project operates on only a single channel (mono), but we must choose 2 channels in the sound format to keep the system compatible with stereo microphones. To obtain a single channel, we simply extract one channel from the resulting 2-D array.

We choose to record 262144 samples of audio for a few reasons: 1) the number is a power of 2 (2^{18}), which will make future FFT operations more efficient; 2) we will achieve a good resolution in the frequency domain (~0.17 Hz) after taking an FFT; 3) the approximately 6-second (5.944 s) duration of audio is long enough to increase the likelihood that more frequencies will be present in the waveform; and 4) the large value will increase the likelihood that the cross-correlation operation will accurately locate the starting index of the excerpt within the original WAVE file.

The disadvantages of using such a large number of samples for the initial data acquisition are that 1) the block will produce a minimum time delay of 6 seconds and 2) the processing time will be longer than it would with a smaller sample length, since processing time increases as sample length increases. **Table 4-1** below summarizes the key parameters used in the initial data acquisition block.

Table 4-1: Parameters chosen for data acquisition.

Parameter	**Value**
Sample Rate	44.1 kHz
Number of Channels	2
Bit Depth	16-bit
Number of Samples	262144 (or 2^{18})

4.2 Initial Synchronization

4.2.1 Overview

After the initial data acquisition stage, it is critical to locate the sample index within the original WAVE file that the excerpt begins so that the frequency content of each FFT match as best as possible. We accomplish the initial synchronization using the cross-correlation function, which is a convolution between the recorded excerpt and the original waveform. The length of the original waveform subtracted from sample index of the peak of the cross-correlation function returns the starting index of the excerpt.

However, with large WAVE files, it is not efficient to perform the cross-correlation with the entire waveform because this will require an unreasonably large amount of memory and slow down the processor. To avoid using excess RAM, we parse the WAVE file into more manageable sizes and perform the cross-correlation between the excerpt and each parsed section.

4.2.2 LabVIEW Block Diagram

Figure 4-4 shows the entire block diagram used in the initial synchronization process.

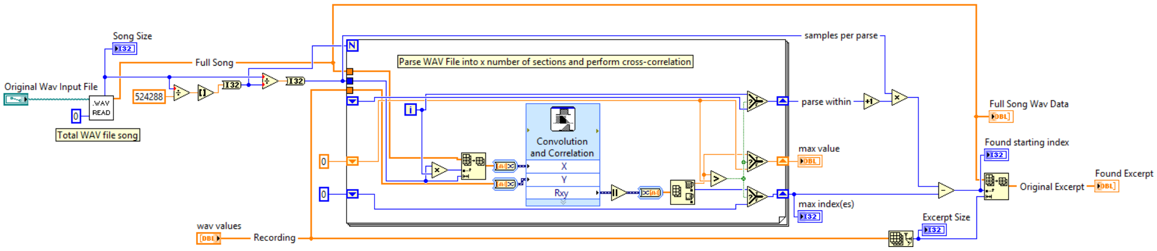


Figure 4-4: Cross-correlation.vi block diagram.

Figure 4-5 shows a zoomed-in view of the input side of the block diagram, in which we read the WAVE file and specify the parse size based on the WAVE file size.

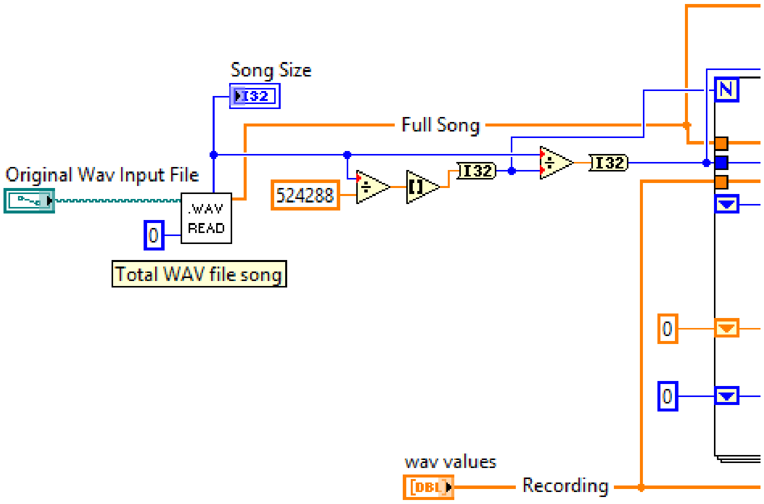


Figure 4-5: Input side of cross-correlation.vi block diagram.

Figure 4-6 shows the parse loop, which sweeps through the WAVE file, performs the cross-correlation on each section, and saves the maximum correlation value, index, and loop iteration in which the max value occurs using three case-select structures and shift registers, which send data to the next iteration of the FOR loop.

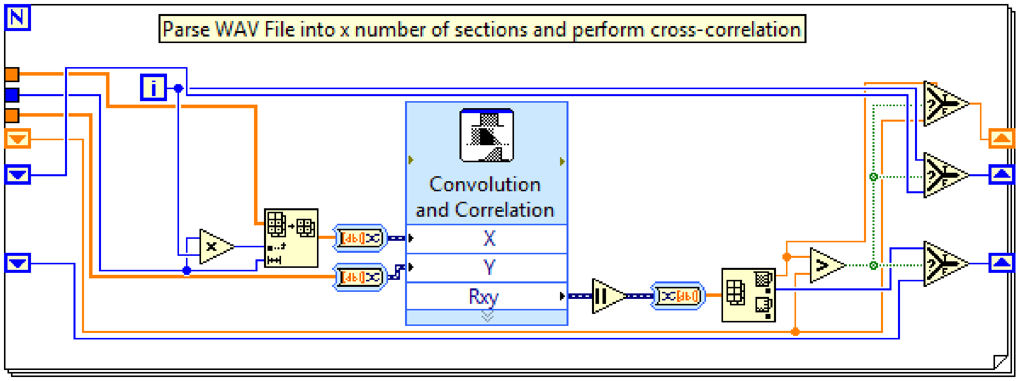

N
Parse WAV File into x number of sections and perform cross-correlation
i
Convolution and Correlation
X
Y
Rxy

Figure 4-6: Correlation parse loop within cross-correlation.vi.

Figure 4-7 shows the output side of the block diagram, which computes the starting index (n) based on 1) the parse size ($size_{parse}$), 2) the index within the parse where the max value occurs (i_{max}), and 3) the parse (loop iteration) in which the max value occurs (i_{parse}) according to **Eq. 4-1** below.

$$n = (size_{parse})(i_{parse} + 1) - i_{max} \qquad \textbf{Eq. 4-1}$$

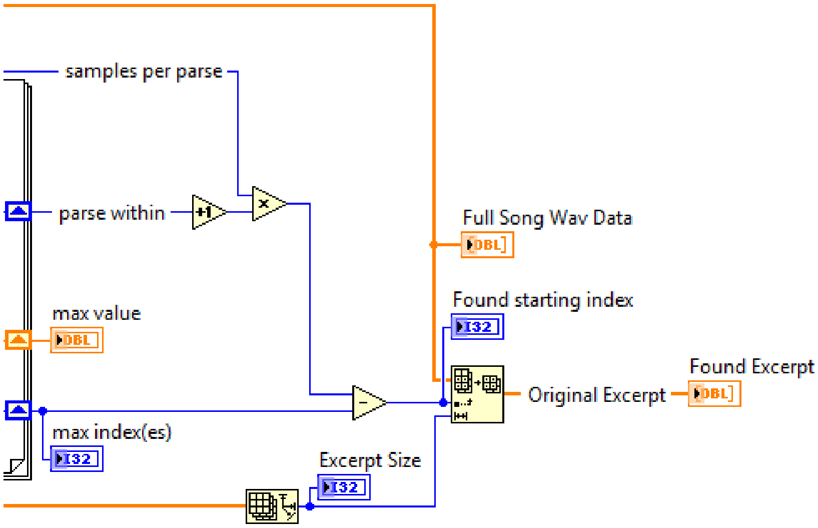


Figure 4-7: Output side of cross-correlation.vi.

4.2.3 Parameters Chosen

The initial synchronization process requires only one important parameter: the parse size. This parameter affects both the memory usage required for processing, the processing time, and the reliability of the correlation function. If the sections are too large, then both the memory usage and the processing time will increase. If parse size is too small, then the correlation result becomes unreliable. Tests show that parsing the WAVE file into sections approximately twice the length of the excerpt produces the best results. This means that the cross-correlation is between the 6-second recorded excerpt and WAVE file sections of approximately 12 seconds.

Table 4-2: Parameters chosen for initial synchronization.

Parameter	**Value**
Parse size	524288 (or 2^{19})

4.3 Transfer Function

4.3.1 Overview

With the approximate starting index of the recorded excerpt located, the next step is to determine the transfer function of the environmental filter (degradation function) by comparing the FFTs of both the measured excerpt and the found WAVE file excerpt. Prior to taking the FFT of the measured excerpt, we filter the waveform with a band-pass filter. The band-pass filter prevents the transfer function from over-amplifying the extreme low and high frequencies. We choose to filter out the low-frequency content because the loudspeaker will not be able to adequately reproduce these frequencies and also because including this frequency content in the canceling waveform could introduce unnecessary distortion. Additionally, we filter out the high-frequency content for a couple of reasons: 1) active noise-cancellation performance worsens at higher frequencies (recall **Figure 2-7** and **Figure 2-21**), and we can thus reduce the risk of adding to the high-frequency noise by pre-filtering the measured sample; and 2) for this project, we can assume that the environmental filter has some passive, low-pass behavior and therefore has already filtered out much of the high-frequency content.

After computing the environmental filter transfer function, we must filter the impulse-like noise in both the magnitude and phase response with a low-order median filter. Although median filtering is a non-linear operation, it works best to eliminate high-variance noise without greatly disturbing the original signal.

4.3.2 LabVIEW Block Diagram

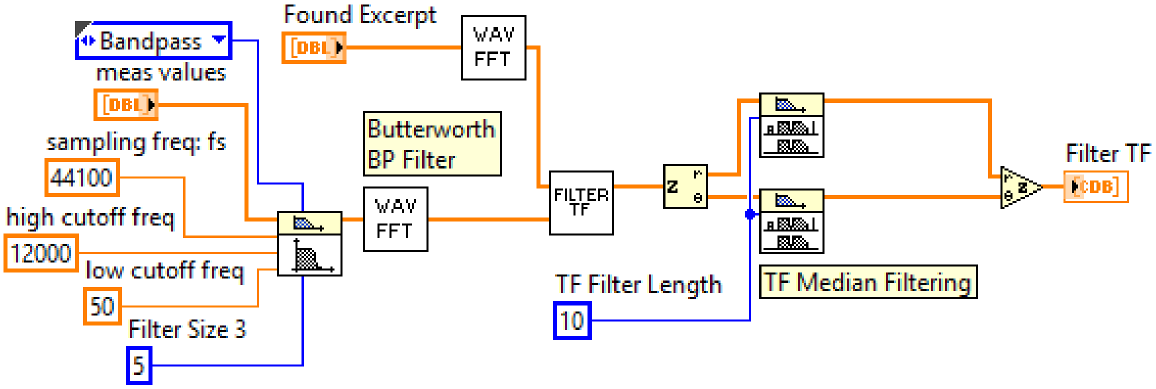


Figure 4-8: Transfer Function.vi block diagram.

4.3.3 Parameters Chosen

The parameters for this block include the band-pass and median filter specification. For the band-pass filter, we choose a 5th order Butterworth filter due to its smooth transition region and linear phase response. We choose 50 Hz for the low cutoff frequency because the loudspeaker will not be able to properly reproduce frequencies below this value. We choose 12 kHz for the high cutoff frequency because we want to avoid adding high-frequency noise during cancellation. Lastly, we choose a median filter size of 10 based on experimental testing. Median filter sizes over 20 caused noticeable and undesirable artifacts in the sound of the waveform. The chosen parameters are summarized in **Table 4-3** below.

Table 4-3: Parameters chosen for transfer function block.

Parameter	Value
Band-pass filter type	Butterworth
Band-pass filter order	5
Band-pass low cutoff frequency	50 Hz
Band-pass high cutoff frequency	12 kHz
Median filter size	10

4.4 Impulse Response Windowing

4.4.1 Overview

One undesirable result of the environmental filter transfer function is that, due to the non-ideal loudspeaker and microphone, it behaves like an infinite impulse response (IIR) filter. This project requires the filter to behave like a relatively low-order finite impulse response (FIR) filter to minimize the impact of circular convolution when rebuilding the output waveform from multiple FFTs. Tests show that the impulse response (found by taking the IFFT of the transfer function) typically contains significant information beyond 100000 samples. In LabVIEW, the impulse response shows peaks at indices 0 and 262144 and nearly symmetrically fades out toward index 131072, as shown in **Figure 4-10**. To properly window the impulse response, we must swap the first and second halves of this waveform, set the first and last 131072 – $M/2$ values to zero, and apply the window to the middle section of M values, where M is the window size. Then, we rebuild the waveform and swap the two halves back. **Figure 4-9** shows the LabVIEW implementation of this process.

After windowing the transfer function's impulse response, we again take its FFT to convert it back to the frequency domain.

4.4.2 LabVIEW Block Diagram

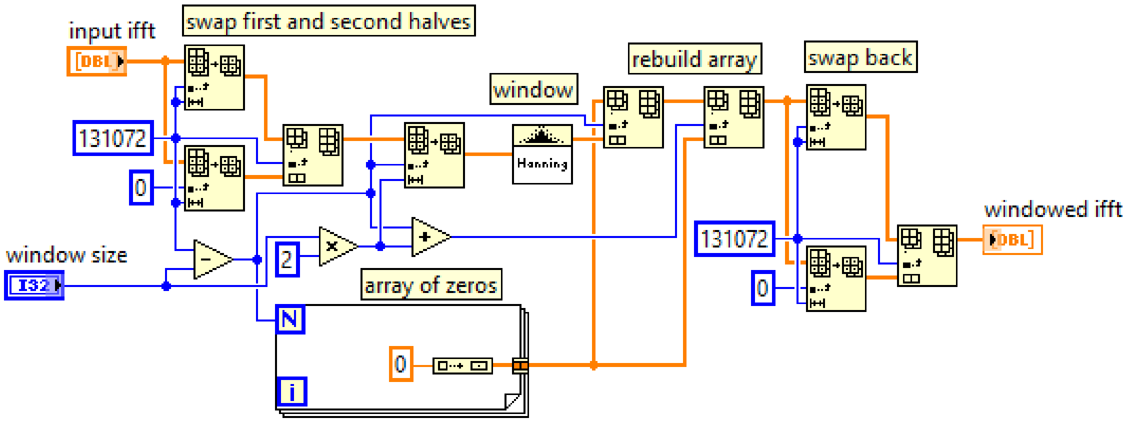


Figure 4-9: Impulse Response Windowing.vi block diagram.

4.4.3 Parameters chosen

The two main parameters in this subsystem are the window type and window size. We choose to use a Hanning Window with a size of 5000. The equation for the Hanning Window is shown in **Eq. 4-2** below, where *M* is the window size.

$$y_i = \frac{1}{2}x_i[1 - \cos\left(\frac{2\pi i}{M}\right)] \qquad \textbf{Eq. 4-2}$$

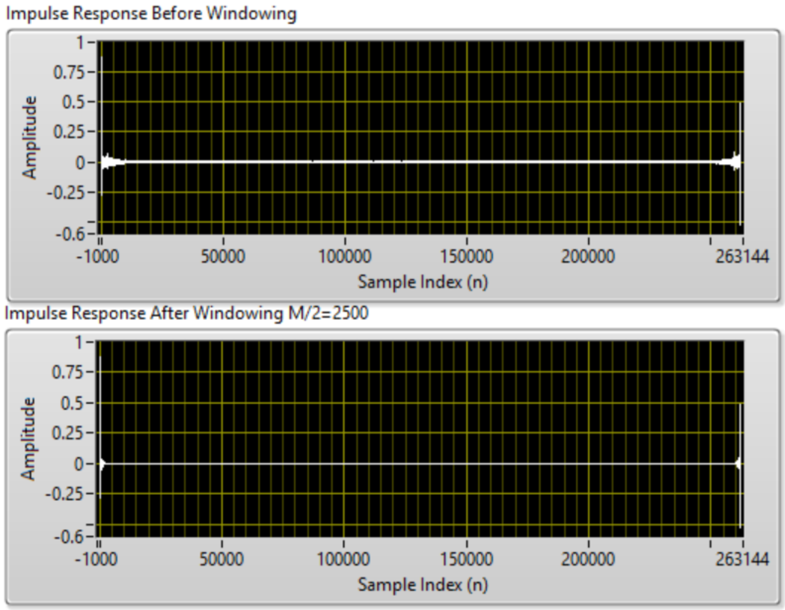


Figure 4-10: Impulse response of environmental filter before and after windowing.

Figure 4-10 shows the impulse response of the environmental filter before and after windowing. We can see that without windowing, the impulse response does not decay to zero. The following table summarizes the parameters chosen for this block.

Table 4-4: Parameters chosen for impulse response windowing block.

Parameter	**Value**
Window type	Hanning
Window size	5000

4.5 Secondary Offset Calculation

4.5.1 Overview

The goal of this block is to determine the amount of time that has elapsed since the initial data acquisition block so that it can compensate by adding an offset to the starting index of the future cancellation waveform. Recall from **Figure 4-3** that "Acquire Microphone Data (6sec).vi" initializes a microsecond timer. Before we can apply the environmental filter to the WAVE file to generate the cancellation waveform, we must find the total elapsed time and convert it to its equivalent number of samples. **Eq. 4-3** shows the conversion from the total elapsed microseconds (*t*) to the equivalent sample offset value.

$$\text{offset} = \frac{t \times f_s}{10^6} \qquad \textbf{Eq. 4-3}$$

4.5.2 LabVIEW Block Diagram

Figure 4-11 shows the LabVIEW implementation of **Eq. 4-3**. The purpose of "FFT values in" and "FFT values out" is to guarantee that LabVIEW processes this block only after the FFT values have been determined, since the VI will not be executed until all inputs are ready.

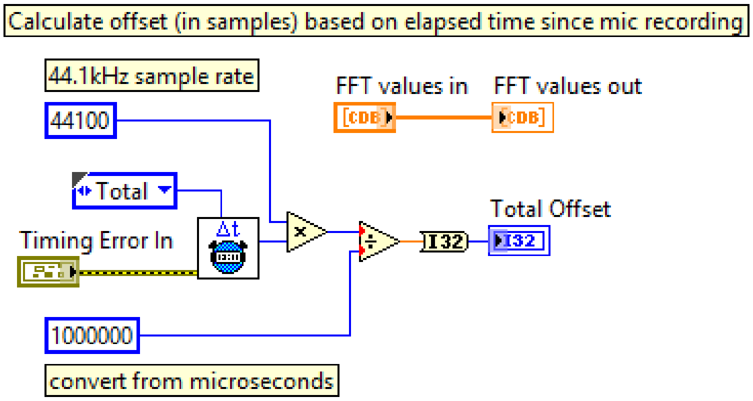


Figure 4-11: Initial Offset.vi block diagram.

4.6 Cancellation Waveform Generation

4.6.1 Overview

The next step is to generate the cancellation waveform from the environmental filter transfer function. The LabVIEW implementation of this process is quite simple: Given the full song WAV data, the found starting index, and the calculated secondary offset, we extract the future excerpt and take its FFT. We then perform complex multiplication with the FFT of the future WAV excerpt and the filter transfer function. Lastly, we take the IFFT of the result and negate the time-domain signal. The reason we cannot simply convolve the impulse response of the environmental filter with the rest of the WAVE file is that this operation would take up too much memory. This is the same reason we had to parse the WAVE file for the cross-correlation in the initial synchronization block.

4.6.2 LabVIEW Block Diagrams

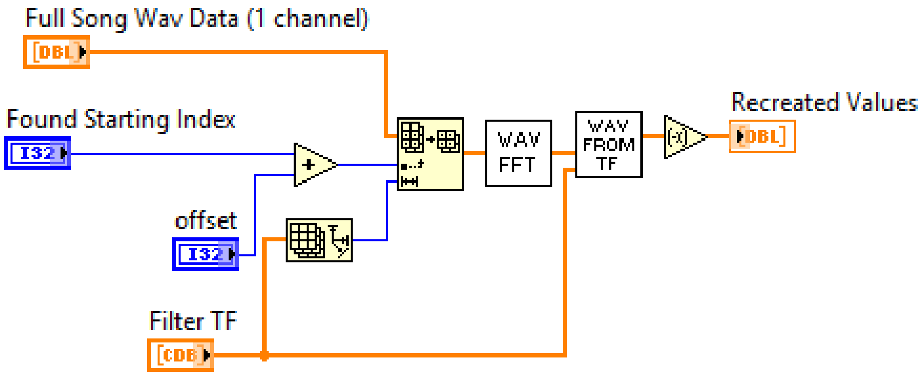


Figure 4-12: Recreate Waveform.vi block diagram.

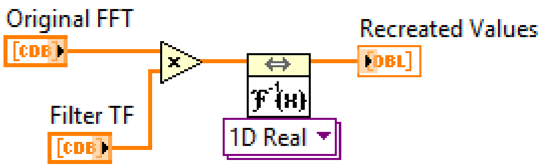


Figure 4-13: WAV from TF.vi block diagram.

4.7 Overlap and Save Method for Successive FFTs

4.7.1 Overview

The previous step generates a cancellation waveform of length 262144, or about 6 seconds of cancellation audio at 44.1 kHz. In order to continue the cancellation, we must repeat the same process to subsequent portions of the WAV data. However, due to the effects of circular convolution, we cannot simply compute the next 262144 values and append the result to the previous computation. Multiplication in the frequency-domain multiplication is equivalent to convolution in the time-domain, and the convolution has some lead-in and lead-out depending on the length of the filter's impulse response. In the overlap and save method, we overlap the successive FFTs, throw away the lead-in and lead-out of each convolution result, and save the parts of the waveform unaffected by the wrapping effect of the circular convolution.

Figure 4-14 shows an illustration of the overlap and save method implemented in this project [14].

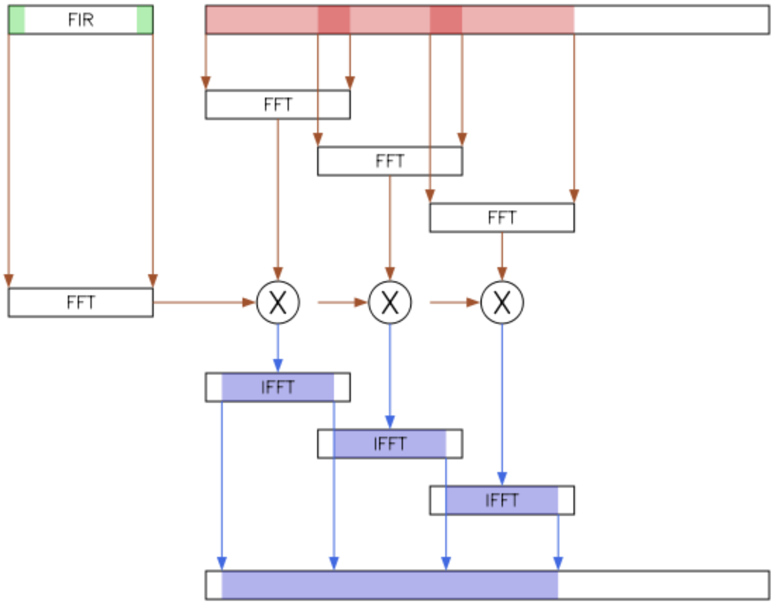


Figure 4-14: Overlap and save method schematic diagram [14].

Using half of the window length (M/2) from the impulse response windowing block as the number of samples to throw away from the beginning and end of each convolution result, we can make use of LabVIEW's array subset function to extract the desired portion of each convolution result. **Figure 4-15** shows the basic LabVIEW implementation of the overlap and save method. We convert each saved portion of the convolution result from WAV values to 32-bit integers (required for WaveIO playback [15]), and we then fill a buffer by concatenating each result to dynamically build an output array. In this implementation, we separately compute the first section so that it is immediately available for playback. This gives the overlap and save block a 6 second head start to fill up the output buffer. The output buffer array, called “I32 values” is a local variable, which can be accessed from other locations within the same VI.

4.7.2 LabVIEW Block Diagram

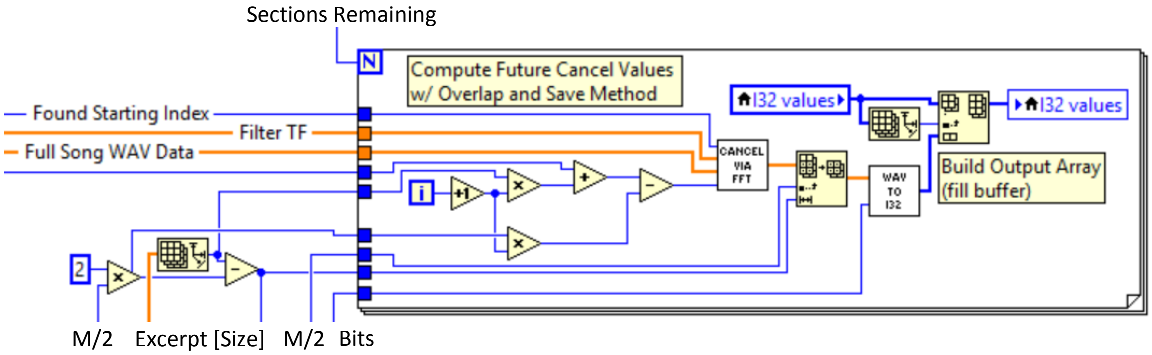


Figure 4-15: Overlap and save method block diagram.

4.7.3 Parameters chosen

The only parameter in the overlap and save block is the overlap size, or the number of samples to throw away from the beginning and end of the signal. We choose M/2 for this parameter, where M is the window length chosen in Section 4.4.3. M/2 is the smallest value for which you can guarantee avoiding the effects of circular convolution, because it is also the maximum length of the environmental filter’s impulse response.

4.8 Final Synchronization and Playback

4.8.1 Overview

As the output buffer fills up, the system can begin playing the cancellation waveform. This block attempts to find the optimal index offset to synchronize the cancellation waveform with the noise signal. We achieve this by first calculating an updated offset compensation value based on the delay since the previous calculation in Section 4.5. Then, we perform a coarse sweep (Sweep 1) of the sample index above and below the approximate best index value while measuring the resulting audio. We locate the region within the recorded waveform where it exhibits the best cancellation, and repeat with a fine sweep (Sweep 2) around the located region. This procedure will only work if 1) the sample index that produces perfect synchronization/cancellation falls within the range of the first sweep, 2) the step size of Sweep 1 is not larger than the width of the cancellation "sweet spot," and 3) the total sweep range of Sweep 2 is at least twice the step size of Sweep 1.

Figure 4-16 shows an illustration of the principle behind the dual-sweep algorithm for the final synchronization block. The red region represents the indices where there is no cancellation, while the green region represents the cancellation "sweet spot," or the range of indices where partial and complete cancellation occurs. The origin index labeled as "0" is the approximate pre-sweep sample index including the initial offset compensation. Note that we may require a constant offset index value if the pre-sweep value is consistently off by the same (or nearly the same) value.

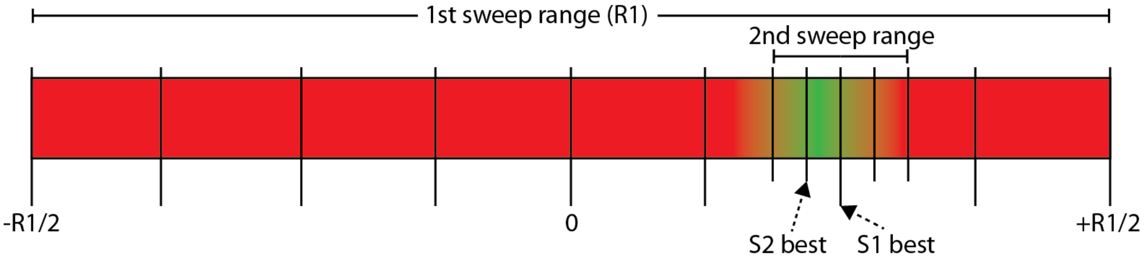


Figure 4-16: Basic principle of sweep algorithm for final synchronization.

4.8.2 LabVIEW Block Diagrams and Detailed Description

While the concept of the dual-sweep algorithm is fairly straightforward, its implementation in LabVIEW is complicated. This block consists of two important loops that must communicate between one another. **Figure 4-17** shows the block diagram of first important loop in this final synchronization process. This loop sets the sweep parameters for each sweep, performs both Sweep 1 and Sweep 2, adjusts the sample index offset value based on the value found in the second loop (shown in **Figure 4-19**), and plays the cancellation waveform.

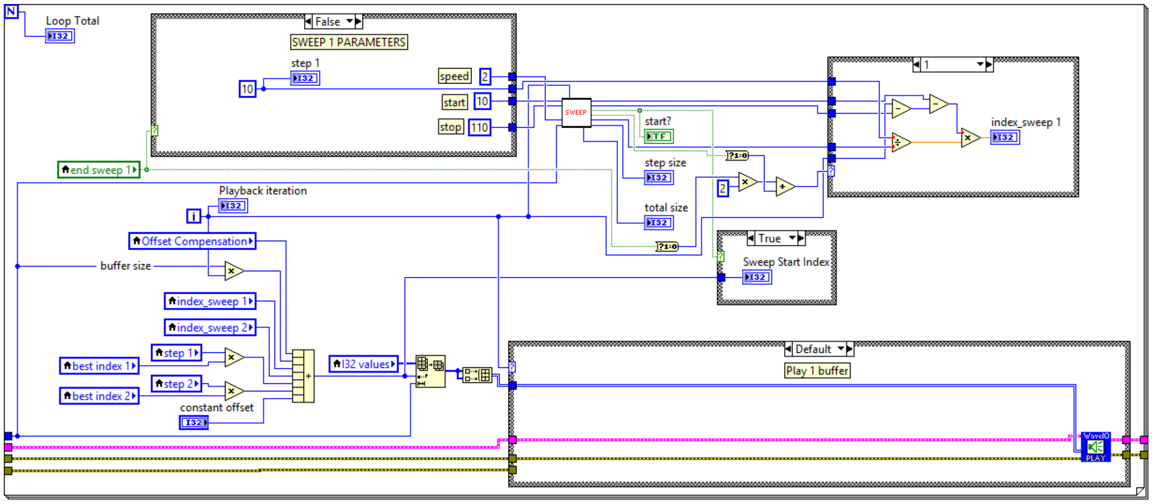
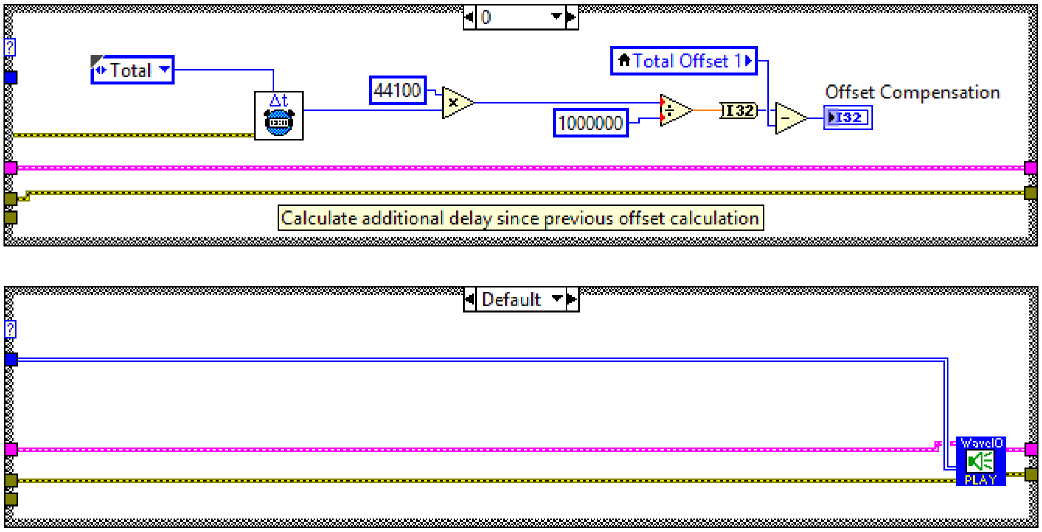


Figure 4-17: Block diagram of first loop in final synchronization.

Figure 4-18 shows the case structure that calculates the best approximate pre-sweep index (origin of **Figure 4-16**) during the first iteration of the loop shown in **Figure 4-17** (bottom right) and plays the buffered cancellation audio during all subsequent iterations.



Calculate additional delay since previous offset calculation

Figure 4-18: Offset compensation calculation on i=0, buffer playback on subsequent iterations.

In **Figure 4-17**, we define the parameters for each sweep. There are 4 different sweep parameters: start iteration, stop iteration, speed adjust, and step size. The start and stop iterations define the iteration range within the first loop that the sweep will occur. The speed-adjust parameter defines the rate at which the index will sweep by skipping iterations within the specified range that do not divide evenly into the difference between the current iteration and the start iteration. For example, a speed-adjust value of 2 would make it so that the index will only step during even iterations. The step size is simply the number of samples the sweep steps by during each iteration. With a speed adjust value of 2, the index will step by the step size every 8000 samples (2 × buffer size), or every 181 milliseconds of real time.

For this project, we choose the following parameters in **Table 4-5** below.

Table 4-5: Parameters chosen for dual-sweep algorithm.

Parameter	**Sweep 1**	**Sweep 2**
Start iteration	10	160
Stop iteration	110	220
Speed adjust	2	2
Step size	10	2

With these parameters, Sweep 1 will step from ±250 samples by 10 samples. This is because with a speed-adjust value of 2, there will only be 50 steps during the 100 iterations, which equates to a total Sweep 1 range of 500 samples. Similarly, Sweep 2 will step from ±15 samples by 1 sample. We wait 50 iterations between Sweep 1 and Sweep 2 to allow the second important loop to process the recorded audio and return the best found index.

The sweep parameters are set in a case structure with the Boolean variable "end sweep 1" as the case selector. For Sweep 1, "end sweep 1" is False. The variable is set to True after processing the measured audio from Sweep 1 in the second loop.

When the first loop iteration equals the start iteration, this triggers a Boolean variable "start?" to be True. When "start?" is True, the sample index at start of the sweep is stored to the variable "Sweep Start Index" and the processing within the second main loop begins.

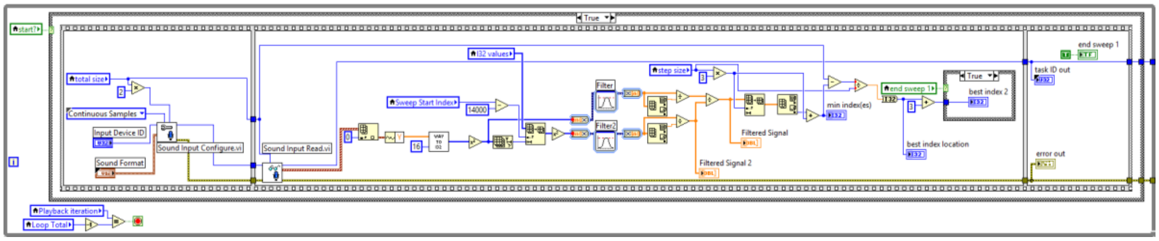


Figure 4-19: Block diagram of second important loop in final synchronization process.

Figure 4-19 shows the block diagram of the second loop, which records audio during each sweep, compares the averaged and normalized measured waveform to the averaged and normalized playback waveform, and determines the index location where the best cancellation occurs. The entire process must be contained within a while loop because it must check the values of the local variables at the start of each iteration.

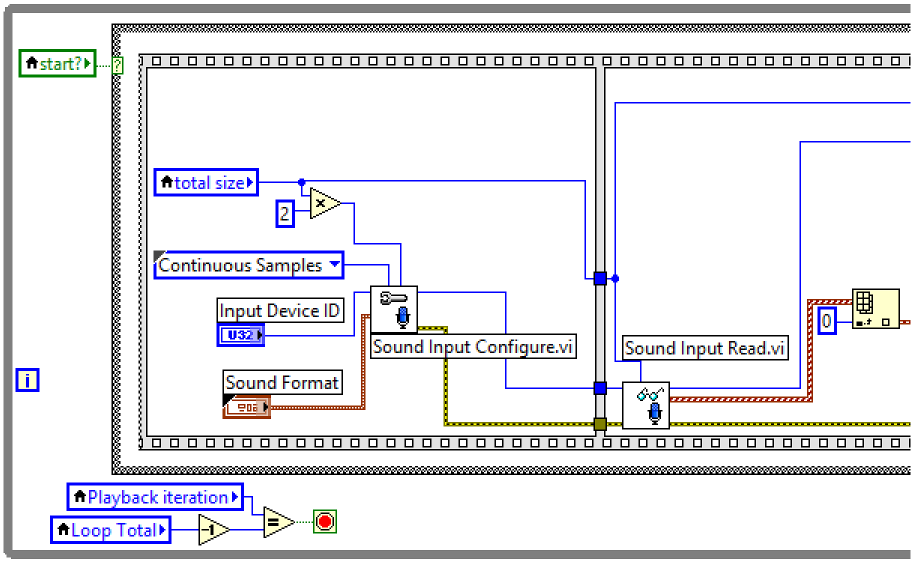


Figure 4-20: Left part of block diagram in **Figure 4-19**.

Figure 4-20 shows the left side of the block diagram in **Figure 4-19**. This is where the sound input is configured and the microphone data is read similarly to the initial data acquisition step in Section 4.1.

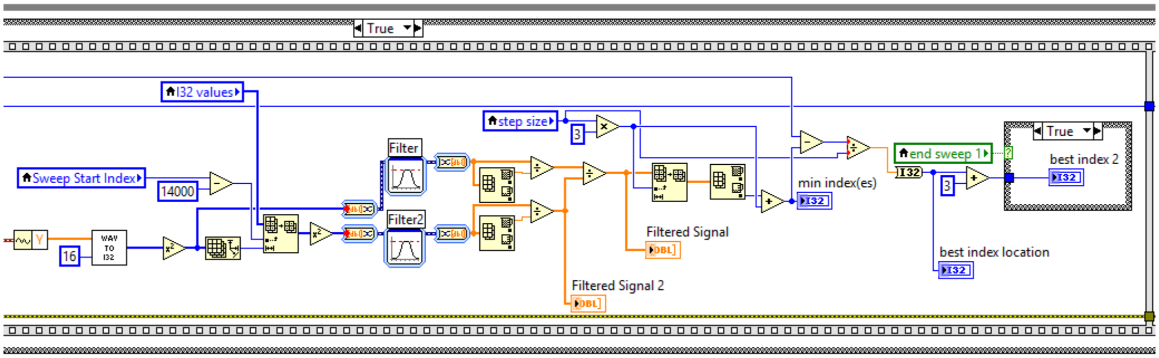


Figure 4-21: Middle part of block diagram in **Figure 4-19**.

Figure 4-21 shows the portion of the block diagram that compares the measured audio to the playback audio (expected audio), and finds the best cancellation index. We must compare the measured and expected audio because songs typically have a variable dynamic range and thus by only taking the minimum value of the measured audio after smoothing, we increase the risk of finding a false positive. This project uses a moving average filter size of 10000. We locate the index where the optimal cancellation occurs within the averaged result by finding the location of its minimum value.

The following are some important details within **Figure 4-21**:

- We subtract an experimentally-determined constant value (14000 in this case) from “Sweep Start Index” to account for the delay in **Figure 4-20**. This better aligns the measured audio with the playback audio.
- We disregard the first (24000 = 3 × step size) samples of the averaged result, where: step size = buffer size × speed adjust = 4000 × 2 = 8000 samples.

  This is because the beginning portion of the outcome of the moving average filter returns small values, which will likely result in false positives. Additionally, depending on the synchronization of the microphone recording and the sweep, it is possible that the microphone might record a cancellation signal at the beginning if it begins the recording slightly before the start of the sweep and if the optimal cancellation index is near the origin of the sweep. See Section 5.2.2 for an example of why disregarding the first 24000 samples is important.

- Because both sweeps are designed to step the sample index value down from $+R/2$ to $-R/2$, the final index of each sweep is $R/2$ below the initial approximation (origin of Sweep 1). We must *add* the appropriate number of indices to $(start - R/2)$ to produce the best index. **Figure 4-22** shows an example where the best cancellation index is 108 samples above the origin. Using the sweep parameters from **Table 4-5**, we expect the best coarse cancellation index to be 36 steps (step size of 10) above the final sweep value of -250. Sweep 2 centers around the sum of the two values, which will be 110 samples above the initial approximation. Lastly, we expect the best Sweep 2 cancellation index to be 13 steps above the final sweep value of -15, producing a final found index value of -2 with respect to the origin of Sweep 2, and +108 with respect to the origin of Sweep 1. The total range of the loop is 400,000 samples in real-time, since there are 50 total steps with (2 × buffer size = samples/step).

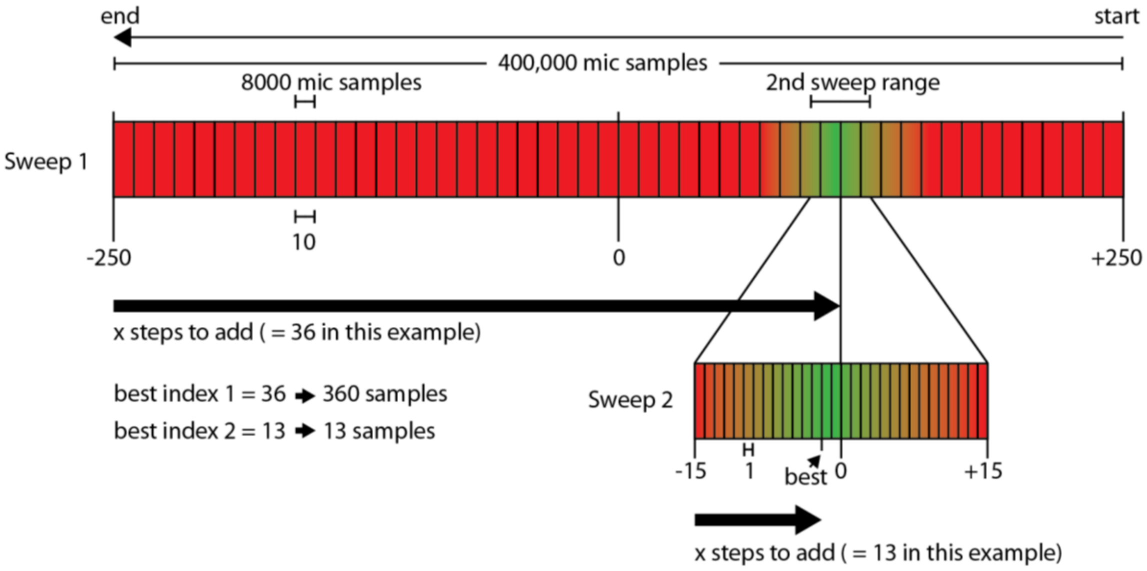

end
start
400,000 mic samples
8000 mic samples
2nd sweep range
Sweep 1
10
-250
0
+250
x steps to add (= 36 in this example)
best index 1 = 36 → 360 samples
best index 2 = 13 → 13 samples
Sweep 2
-15
1
best
0
+15
x steps to add (= 13 in this example)

Figure 4-22: Dual-sweep synchronization example.

The final part of the second loop is to set "end sweep 1" to True, shown in **Figure 4-23**. Recall, that this value will trigger the sweep parameter case structure to switch from the Sweep 1 parameters to the Sweep 2 parameters.

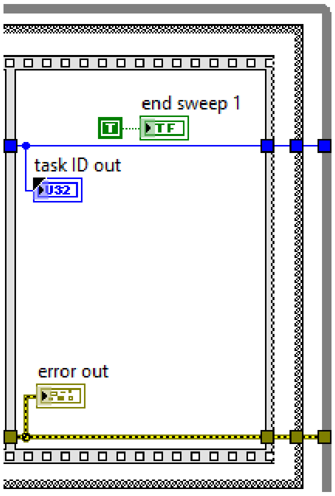


Figure 4-23: Right part of block diagram in **Figure 4-19**.

Figure 4-24 summarizes the basic flow diagram of the final synchronization step. Note that the blue blocks represent the processing that takes place in **Figure 4-17**, while the green blocks represent the processing that occurs in **Figure 4-19**.

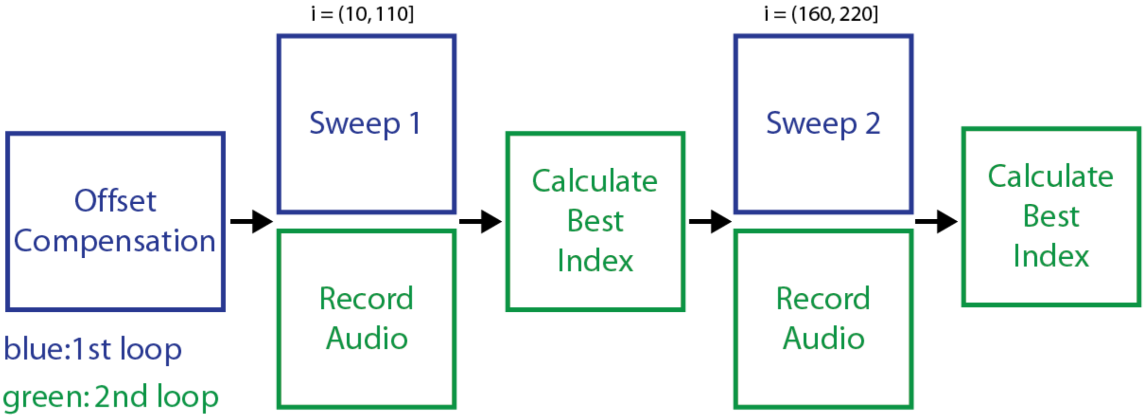


Figure 4-24: Summary of final synchronization block diagram.

Figure 4-25 shows the accumulative offset correction within the first loop in **Figure 4-17**. Here we add 7 different values together: the offset compensation value calculated during the first iteration, the index values for each sweep (final values are −250 and −15 for Sweeps 1 and 2 respectively), the sample index corrections for each sweep, the constant offset value, and the current playback iteration multiplied by the buffer size.

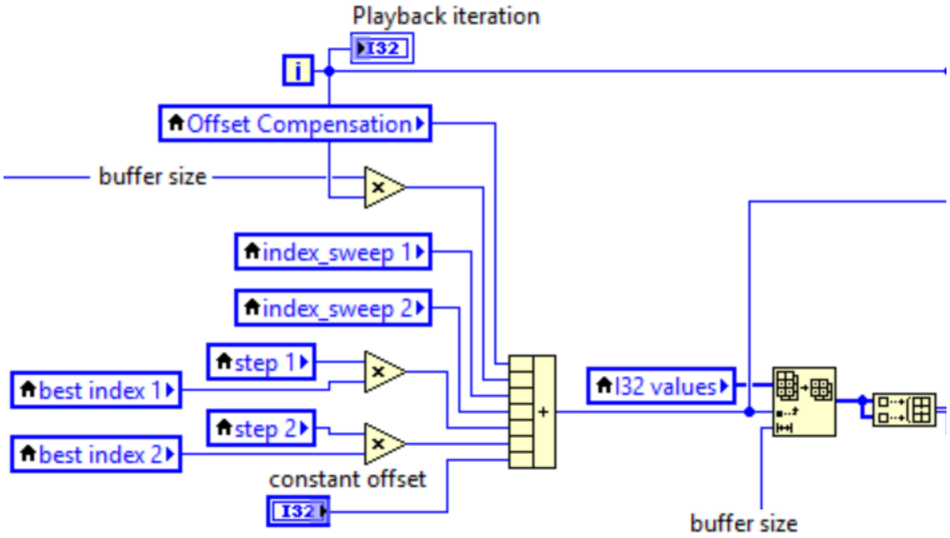


Figure 4-25: Accumulative offset correction within first loop in **Figure 4-17**.

CHAPTER 5: SYSTEM VERIFICATION/TESTING

In this chapter, we will show and discuss the experimental test results for the amplitude matching calibration step, the dual-sweep synchronization algorithm, and the complete system performance.

5.1 Amplitude Matching Calibration

5.1.1 Overview and Setup

To avoid the need for amplitude synchronization, we perform an amplitude calibration step, which matches the amplitude of the microphone signal to that of the loudspeaker signal. This will ensure that there is a 1:1 ratio between the measured audio and the cancellation audio and will allow magnitude variations in the frequency domain to accurately reflect the amplitude variation required for cancellation in the time domain. In this project, the final amplitude value is controlled by four different level-controls: 1) the volume setting on the PC, 2) the volume knob on the loudspeaker, 3) the gain-control knob on the microphone preamplifier, and 4) the microphone gain setting within Windows' sound control panel.

To perform the calibration step, we set 3 of the 4 level-controls previously listed to constant settings – We choose the volume knob on the loudspeaker as the only variable. Next we play a 1 kHz sine wave signal through the loudspeaker and record the output audio with the microphone. We compare the RMS of the recorded waveform to that of the original waveform, and manually adjust the volume knob on the loudspeaker until the RMS values match. We can sweep the input level of the sine wave from -30 dB to 0 dB to ensure the amplitude matches over a wide dynamic range.

We create a data collection VI to sweep the input level from −30 dB to 0 dB in steps of +3 dB and store the input and output RMS values in an array. This is shown in **Figure 5-1** below.

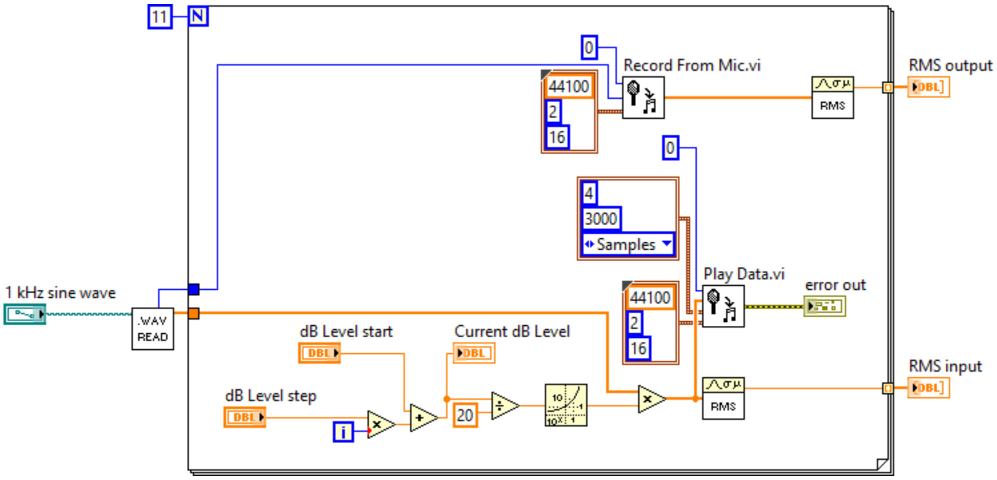


Figure 5-1: Amplitude matching calibration block diagram.

5.1.2 Test Results

Table 5-1 shows the final amplitude matching calibration test results. For this test, the PC volume level was set to 16 (out of 100) and both microphone gain levels were set to their maximum.

Table 5-1: Amplitude matching calibration results with 1 kHz reference signal.

Input Level (dB)	Input RMS	Output RMS	RMS percent error (%)
-30	0.0224	0.0224	0.0295
-27	0.0316	0.0315	0.4015
-24	0.0446	0.0443	0.7475
-21	0.0630	0.0625	0.8638
-18	0.0890	0.0881	0.9795
-15	0.1257	0.1242	1.2327
-12	0.1776	0.1749	1.5089
-9	0.2509	0.2467	1.6748
-6	0.3544	0.3483	1.7272
-3	0.5006	0.3982	20.4596
0	0.7071	0.3985	43.6386

We can see that the input and output RMS levels are very similar to each other for input levels between −30 dB and −6 dB. However, as the input level approaches 0 dB, the output RMS no longer increases linearly. This is shown visually in **Figure 5-2** and **Figure 5-3**. The most likely reason why the output RMS values level off as the input approaches 0 dB is that this particular soundcard or codec has built-in compression or limiting as the input level approaches peak gain.

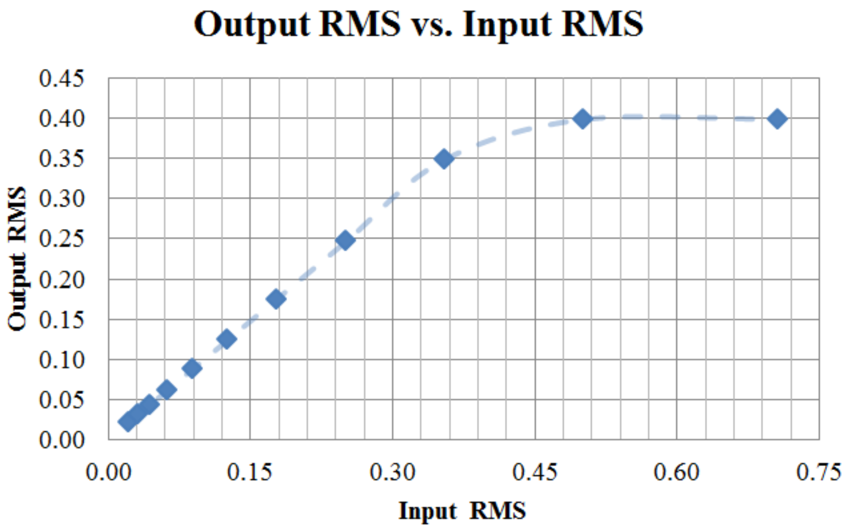


Figure 5-2: Output RMS vs. Input RMS for amplitude matching at 1 kHz.

The trend line representing the linear region between −30 dB and −6 dB has a slope of 0.9813, which is a 1.87% error from the desired value of 1.0.

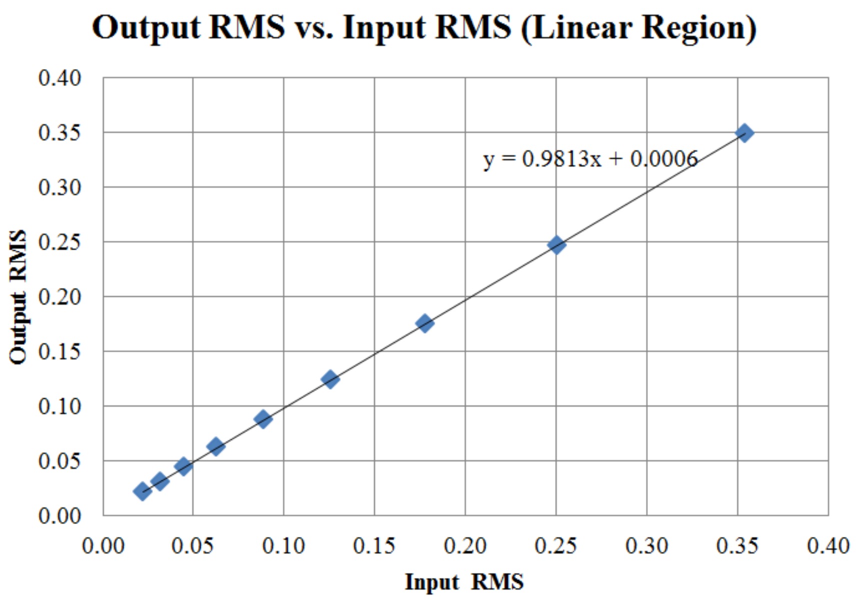


Figure 5-3: Linear region of **Figure 5-2**.

Figure 5-4 below shows the RMS percent error as the input level is swept from −30 dB to 0 dB. The error is less than 2% until the input level reaches −6 dB, where it begins to increase due to the peak compression.

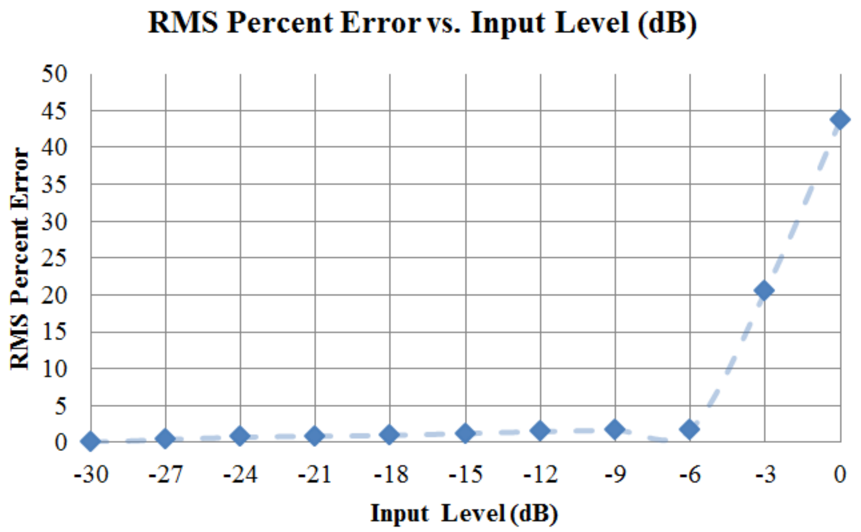


Figure 5-4: RMS percent error vs. input level (dB).

In this test, we developed a reliable way to calibrate the amplitude of the speaker output to match that of the microphone measurement. We discovered that the soundcard or codec introduces compression as the input level approaches 0 dB. This mismatch at high amplitude levels will negatively affect the overall system's ability to attenuate loud noise, but it should not have an impact on its performance for amplitude levels below −6 dB. Soundcards or codecs that do not introduce compression near 0 dB will not create this problem.

5.2 Final synchronization (Dual-Sweep) Testing and Analysis

5.2.1 Overview and Setup

To test the reliability of the dual-sweep synchronization method from Section 4.8, we design a test suite in LabVIEW consisting of a modified version of the complete system which bypasses the environmental filter transfer function block. Bypassing this block simulates a perfect loudspeaker frequency response and a perfect environment.

Next, we play a WAVE file with Windows Media Player at 100% volume before running the LabVIEW program. The cancellation loudspeaker is connected to the computer's audio-out jack and the measurement microphone is placed near the woofer (as opposed to the tweeter) of the loudspeaker. The woofer outputs frequencies at the lower end of the audible range, which are more likely to be attenuated. Here, the cancellation occurs within the soundcard – the loudspeaker is used to play the canceled audio so that the microphone can detect the optimal cancellation region for each sample index sweep. **Figure 5-5** shows the physical setup configuration for this test.

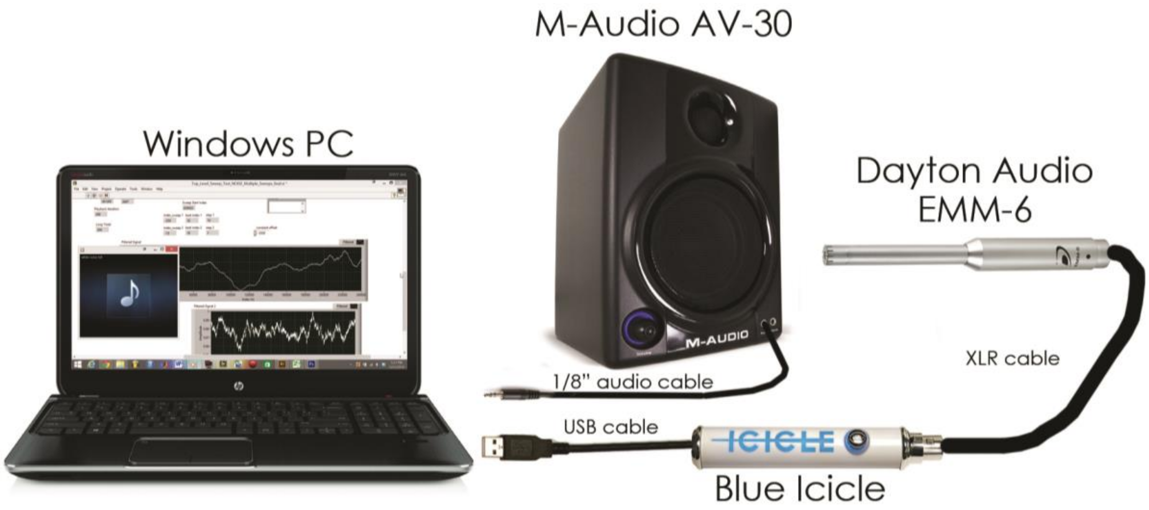


Figure 5-5: Test setup for dual-sweep synchronization.

5.2.2 Test Results

The first test uses a white noise source as the unwanted audio. Out of 20 trials, the dual-sweep synchronization method was successful in all but 1 of them (95% success rate). The only failure was due to the cancellation "sweet spot" not falling within the range of the first sweep. Possible reasons for this include: 1) variable processing speeds within Windows, 2) large deviation between the cross-correlation result and the expected value, and 3) additional unwanted noise interfering with the initial audio measurement.

Recall from Section 4.8.2 that the sweep may require a constant index offset. This particular test required a constant offset of -3500 samples. The following two figures show the recorded microphone signals (averaged and compared to the original sweep signal) for both Sweep 1 and Sweep 2. Here the cancellation region falls within the range of Sweep 1, and the system is able to correctly locate the optimal cancellation sample index.

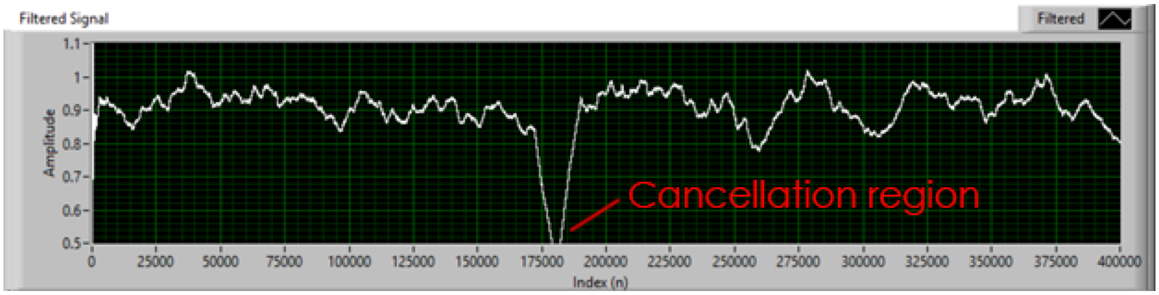


Figure 5-6: Sweep 1 test (white noise source, ideal environment/devices) – #1.

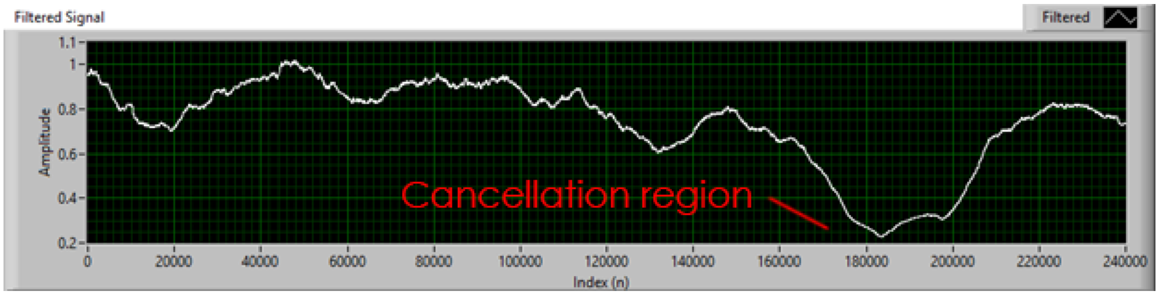


Figure 5-7: Sweep 2 test (white noise source, ideal environment/devices) – #1.

In the following example, the microphone recording for Sweep 2 begins slightly before the index sweep begins, which means that the microphone records a short period of cancellation at the beginning of the measurement. Since we do not want this false positive to be identified as the best cancellation index, we disregard the first 24000 samples and use the remaining samples to determine the best index (shown in **Figure 5-9**).

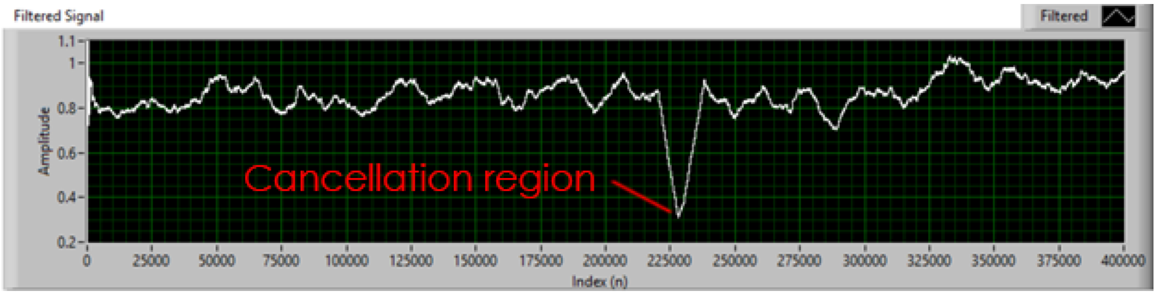


Figure 5-8: Sweep 1 test (white noise source, ideal environment/devices) – #2.

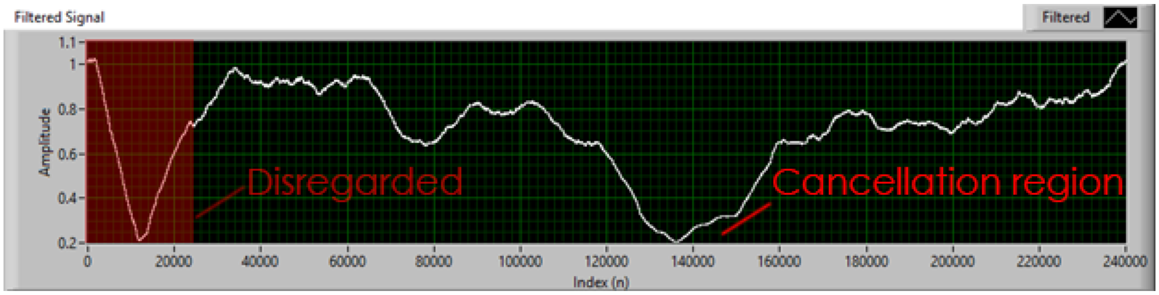


Figure 5-9: Sweep 2 test (white noise source, ideal environment/devices) – #2.

From **Figure 5-7** and **Figure 5-9**, we see that the unwanted white noise appears to be reduced by approximately 80%, or 14 dB. However, recall that this waveform is normalized, averaged, and compared to the normalized and averaged input waveform, so it doesn't reflect the actual attenuation. To get a better idea of the attenuation this produces, we must compare the frequency responses of the measured noise and the measured cancellation waveforms. **Figure 5-10** shows the attenuation in the frequency domain.

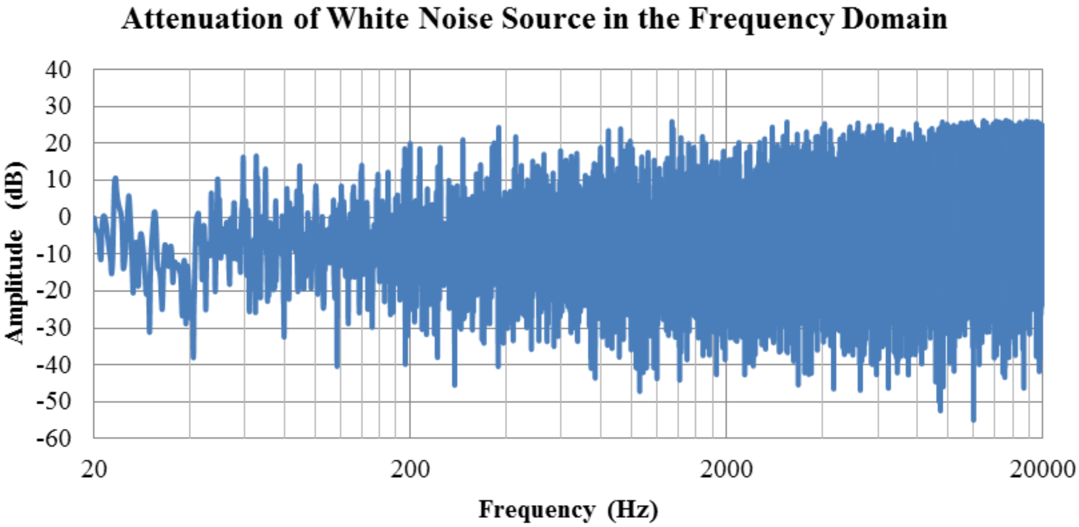


Figure 5-10: Attenuation of white noise source in the frequency domain.

From **Figure 5-10**, it is still difficult to understand how well the system attenuates noise. The best metric to understand the overall noise cancellation performance is to compare the total power of the noise signal with the cancellation signal. To do this, we simply square and every element in both waveforms, sum the array elements and divide the result from the cancellation waveform with that of the noise waveform. In the example above, the total signal power is attenuated by 9.4 dB. For this case, the total signal power attenuation ranges from 8-15 dB.

For the next test, we replace the white noise sound source with a low-pass-filtered version of an actual song and run an additional 20 trials. Theoretically, we should expect the system to produce better cancellation because perfect synchronization is not required to significantly attenuate low-frequency sound. However, the filtered signal has much less frequency content than the unfiltered WAVE file version and this could reduce the reliability of the cross-correlation result – *i.e.* the system may no longer consistently and accurately locate the starting index of the recorded excerpt within the unfiltered WAVE file. We will most likely require different sweep parameters and a different constant sample index offset value to compensate for this. After many tests, we find that the constant sample index which yields the most consistent results is -4750 samples. This result was found to deviate by as much as ±1000 samples. To accommodate for this wide sample offset deviation, we could increase the step size of Sweep 1, or increase the start-to-stop iteration range. The main drawback of increasing the sweep range is a longer sweep time. For this test, however, we will leave the sweep parameters as they were and only adjust the constant offset value.

After 20 trials, we find that the dual-sweep algorithm located the best cancellation index 14 times (70%). 6 tests failed because the desired sample index was outside of the range of the sweep. As we hypothesized, the following graphs show a wider cancellation "sweet spot." We also see that we can achieve better attenuation than in the previous test.

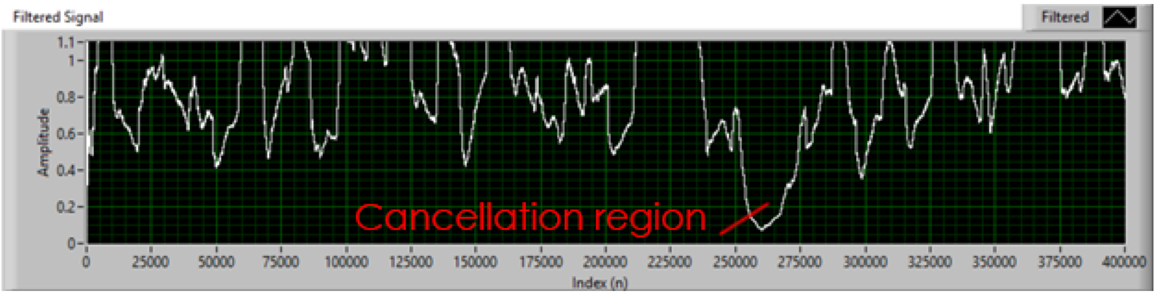


Figure 5-11: Sweep 1 test (low-pass song source, ideal environment/devices) – #1.

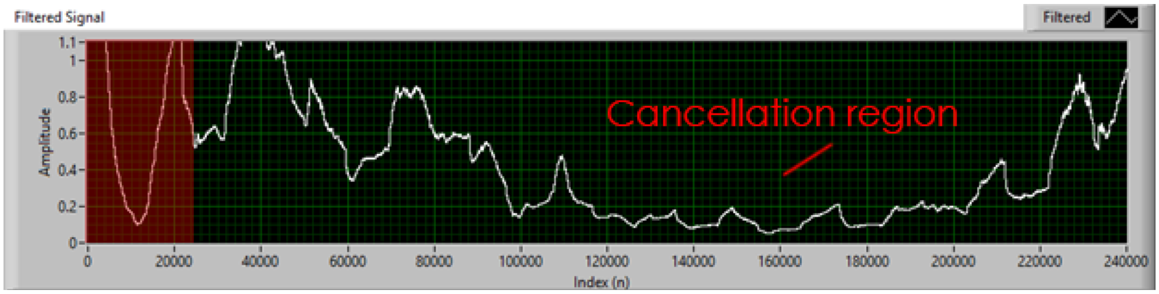


Figure 5-12: Sweep 2 test (low-pass song source, ideal environment/devices) – #1.

The following two figures show another successful sweep in which the normalized and averaged unwanted noise shows a reduction of 96%, or 28 dB. Again this metric does not accurately reflect the actual noise cancellation performance given this particular input signal; it simply serves as an estimate for the magnitude of the attenuation as well as proof that cancellation is present.

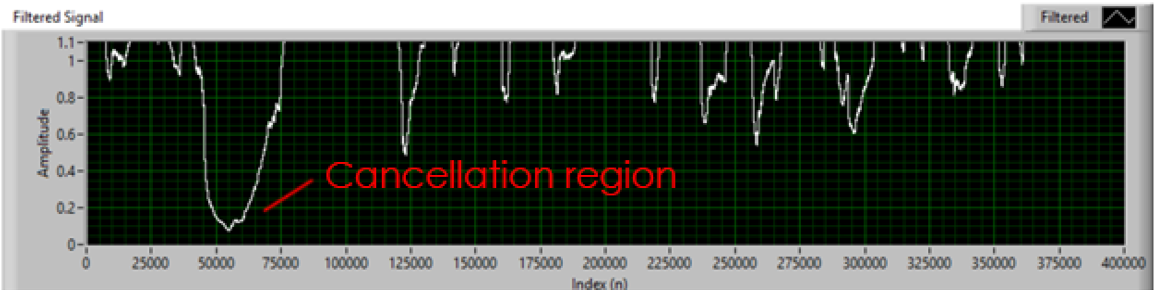


Figure 5-13: Sweep 1 test (low-pass song source, ideal environment/devices) – #2.

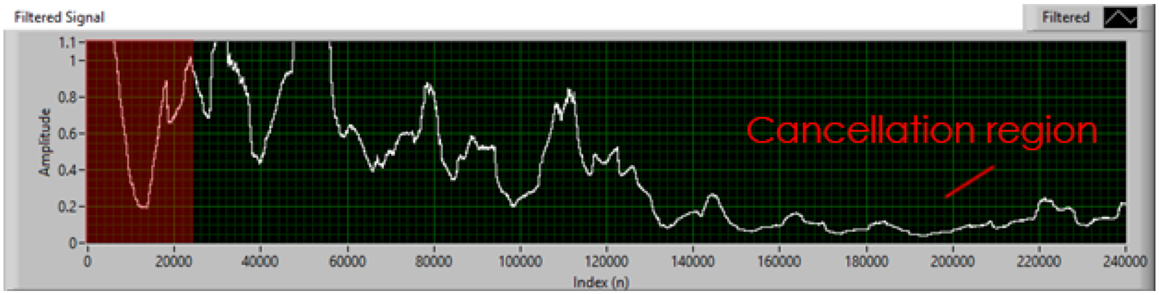


Figure 5-14: Sweep 2 test (low-pass song source, ideal environment/devices) – #2.

Figure 5-15 shows the noise attenuation in the frequency domain. We see that when the unwanted sound signal is low-pass in nature, the cancellation performance is much better than when the signal has a lot of high-frequency content. If we compare the input noise power with the cancellation noise power, we find that this system attenuates the noise power by 64 dB. Section 2.2 of this report shows a similar result.

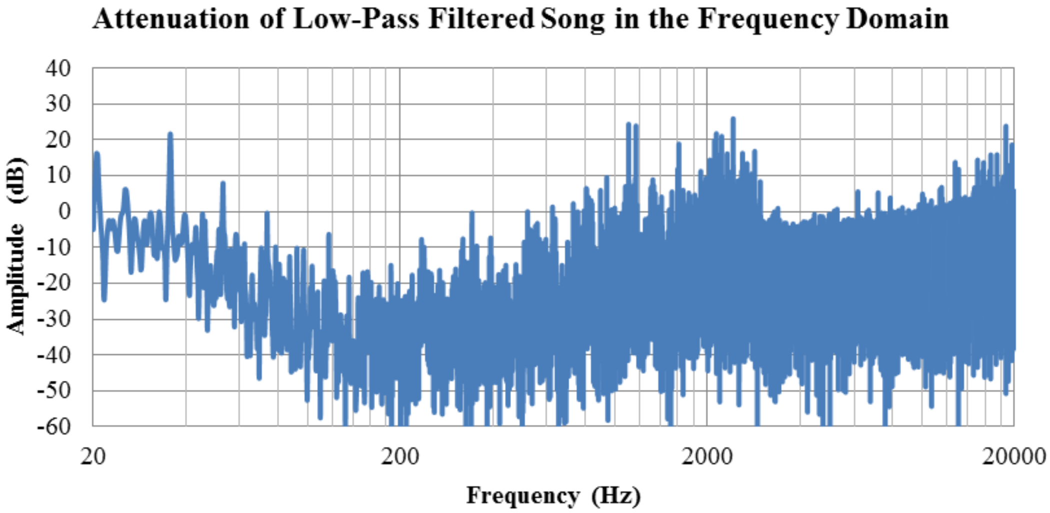


Figure 5-15: Attenuation of low-pass filtered song in the frequency domain.

From these tests, we see typical attenuation between 50-70 dB for low-pass filtered songs. Keep in mind that these tests simulate a perfect environment where the noise and cancellation audio occur within the soundcard.

5.3 Complete System Testing

Lastly, we test the performance of the entire system for both a white noise source and a low-pass filtered song. We also perform the test with and without loudspeaker equalization. The test-setup is similar to the one shown in **Figure 5-5** except that now instead of playing the noise within Windows, we have an external sound source.

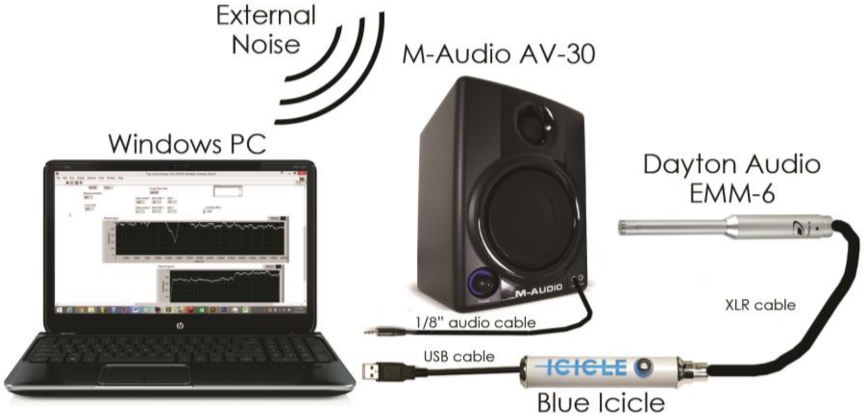


Figure 5-16: Complete system test-setup with external noise source.

5.3.1 White Noise Source

For this test, we use a white noise signal for the unwanted noise source. The following two figures show the measured environmental filter frequency response before and after windowing the impulse response. We observe that windowing has a greater impact on the low frequency response than the high frequency response.

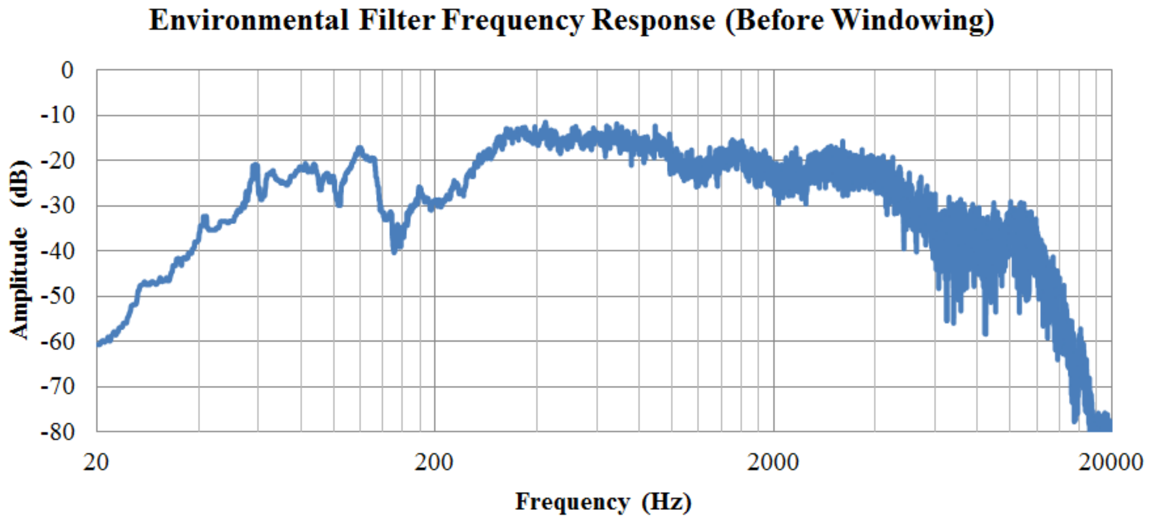


Figure 5-17: Measured environmental filter frequency response (before windowing).

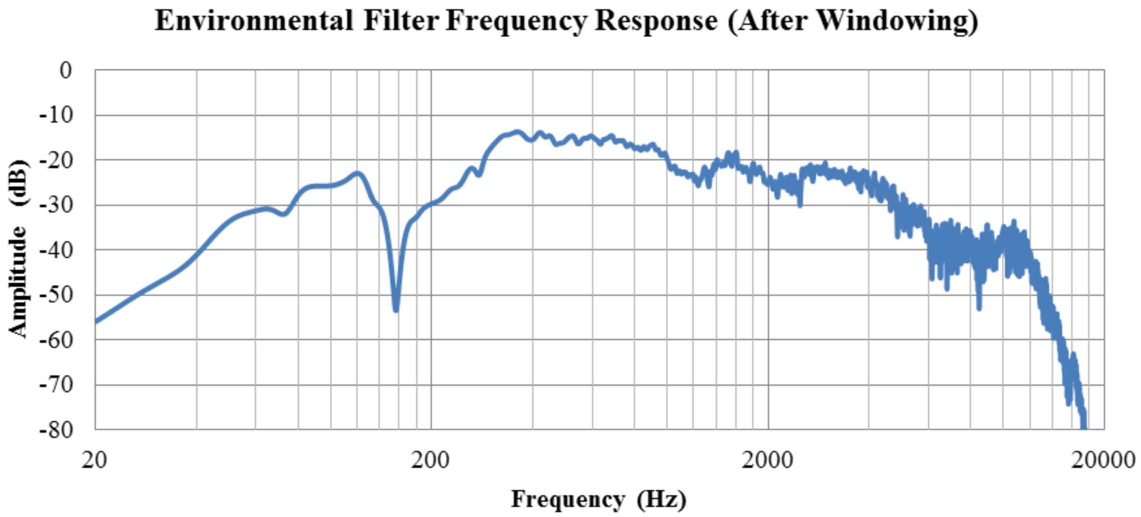


Figure 5-18: Measured environmental filter frequency response (after windowing).

The following two figures show results similar to those produced in the dual-sweep test section. We see that the first sweep correctly locates the general cancellation region, and the second sweep finds the precise location. This example shows the best noise reduction achieved over 20 trials.

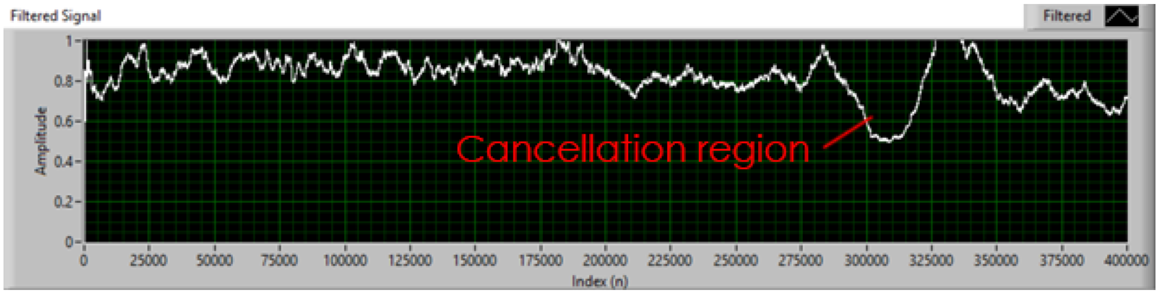


Figure 5-19: Full system test (white noise, Sweep 1).

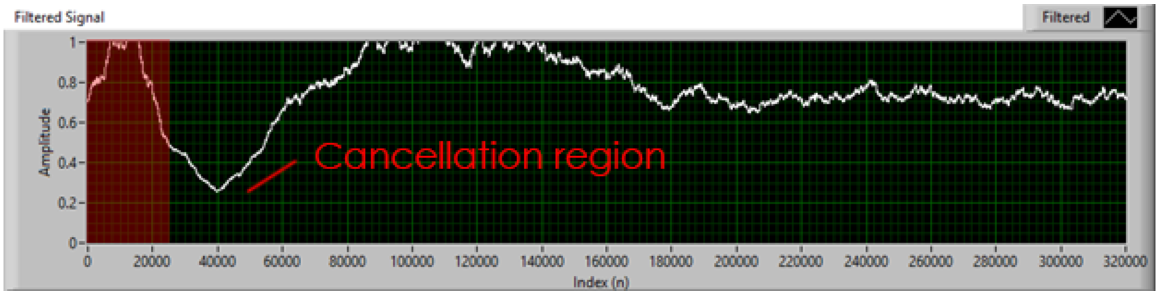


Figure 5-20: Complete system test (white noise, Sweep 2).

Here, the noise power is attenuated by 14 dB. **Figure 5-21** shows the attenuation of the white noise in the frequency domain. Out of 20 trials, this was the best case noise attenuation. The average attenuation is approximately 10 dB.

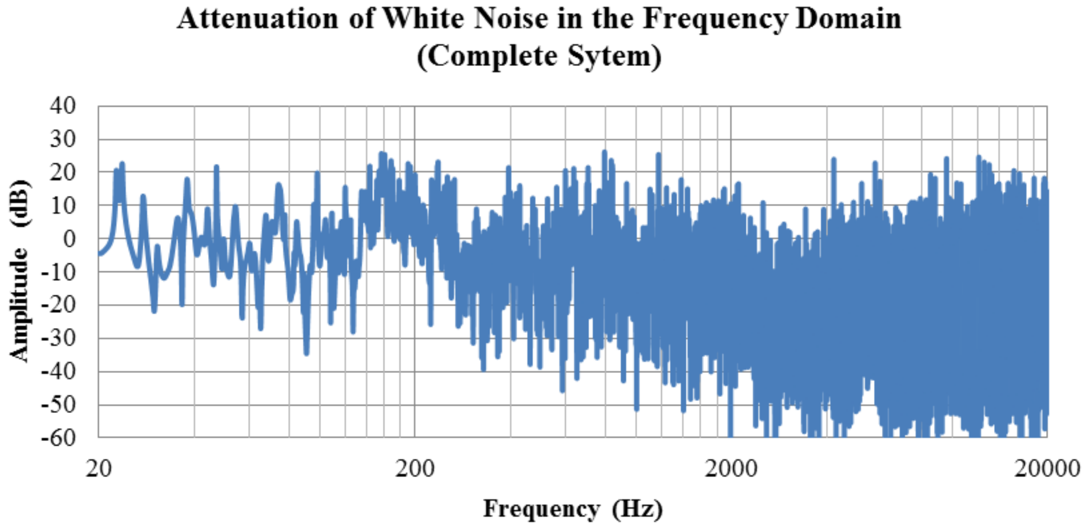


Figure 5-21: Attenuation of white noise source in the frequency domain (complete system).

5.3.2 Low-Pass Filtered Song Source

Here, we repeat the previous test for a low-pass filtered song. As demonstrated in Section 5.2, we expect the system to provide better noise attenuation because active attenuation is better at canceling low-frequency noise. We can achieve the best noise-cancellation performance if we anticipate low-pass filtered noise and lower the band-pass filter's high cutoff frequency from 12 kHz to 2 kHz. This helps because any high-frequency noise the microphone picks up will not be factored into the environmental filter transfer function and thus the cancellation waveform will not include the song's high-frequency content. Put another way, if we know that the unwanted noise is low-pass filtered and does not contain high-frequency content, then we can take advantage of this by lowering the high frequency cutoff of the transfer function so that the system does not attempt to cancel the song's low-frequency content with any high-frequency content.

Figure 5-22 shows the measured environmental filter transfer function (after windowing) with the high cutoff frequency set to 12 kHz (left) and 2 kHz (right). Both show strong frequency content at the low end of the audible range below 2 kHz. However, the plot on the left exhibits significant frequency content above 2 kHz.

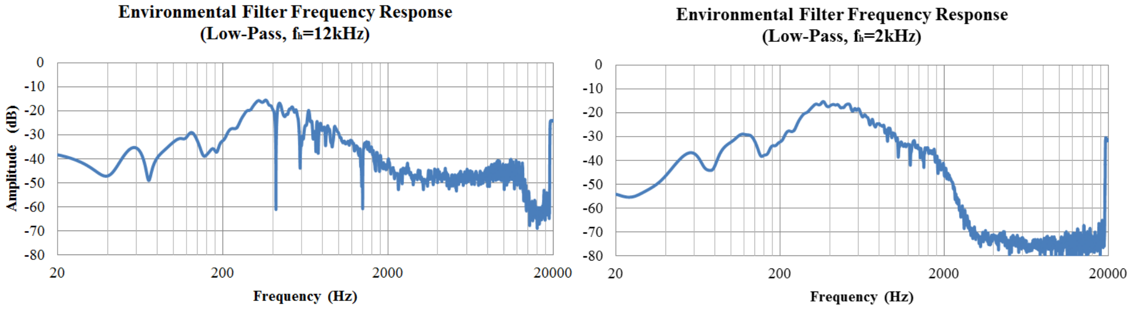


Figure 5-22: Environmental filter frequency response for low-pass filtered song with f_h = 12 kHz (left) and 2 kHz (right) after windowing.

The plot on the right is a better representation of the actual unwanted song's frequency response and thus will provide better cancellation performance.

The following two figures show the results of the dual-sweep synchronization step. We observe that Sweep 1 locates the cancellation region, and Sweep 2 finds the optimal sample index. Here, we see that the noise appears to be attenuated by approximately 90%, or 20 dB.

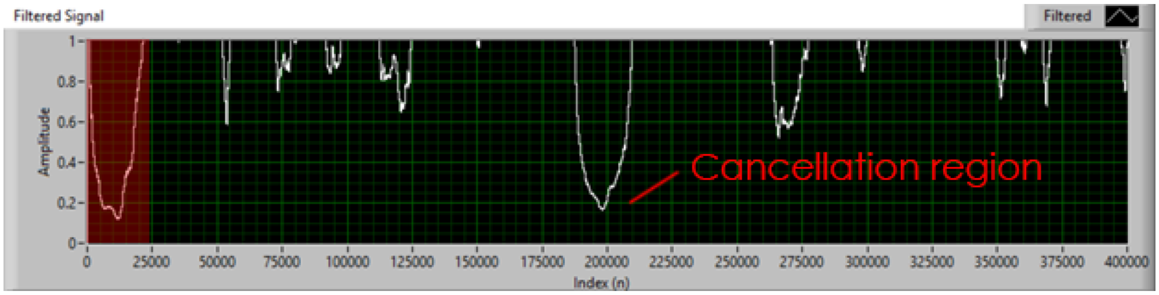


Figure 5-23: Complete system test (low-pass filtered song, Sweep 1).

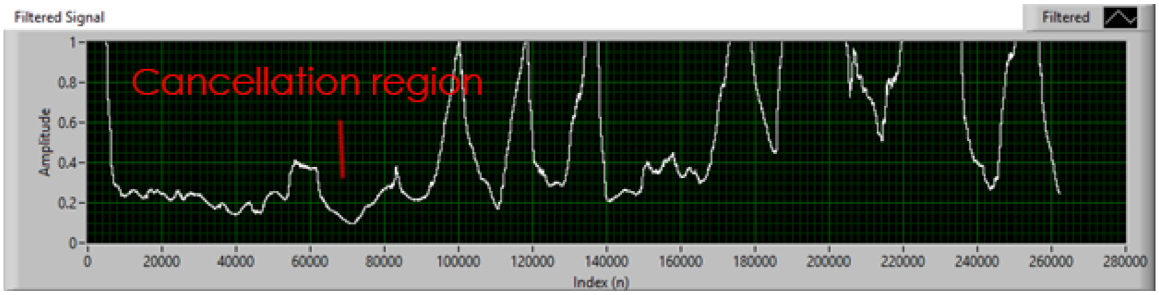


Figure 5-24: Complete system test (low-pass filtered song, Sweep 2).

This system is very sensitive to additional noise: For example, the existence of 60 Hz noise will cause the environmental filter transfer function to amplify the song's 60 Hz frequency content, which is not desirable. Also, there is a possibility that random impulses of noise may occur during the cancellation region, which could prevent the system from correctly locating the best possible cancellation index.

It also appears that the overall system performance depends on the available resources within Windows. For example, if there are too many tasks running in the background that require a significant amount of system memory and processor demands, then the processing within LabVIEW will take longer. Luckily, the system is designed to handle these slight fluctuations in processing performance. However, tests show that varying the system's resources affects the constant sample index offset required for time-synchronization. A dedicated processor would not have this problem. The dual-sweep synchronization step also requires that the system completes the post-Sweep-1 processing before the start of Sweep 2, since Sweep 2 uses the result from Sweep 1 to determine its sweep location.

CHAPTER 6: CONCLUSIONS

6.1 Summary

In this report, we introduced a novel frequency domain method for the active acoustic cancellation of known audio sources and demonstrated a system prototype in LabVIEW running on a Windows PC using Dayton Audio's EMM-6 measurement microphone and M-Audio's AV-30 loudspeakers. We showed that in a perfect environment with a perfect measurement microphone and loudspeaker, *i.e.* flat frequency responses with high signal-to-noise ratios, we can achieve roughly 300 dB noise attenuation, where the error associated with the 32-bit floating point computations is the only thing preventing perfect cancellation.

We also showed that representing the environmental filter frequency response by a low-order linear filter changes the typical noise attenuation to 50-70 dB. In this case, small errors in the environmental filter recreation have a greater impact on the system's ability to attenuate noise.

Finally, for the complete system implementation, we achieved typical wideband (white noise) attenuation between 8-15 dB with low-frequency attenuation in the range of 10-20 dB. For the most part, these attenuation values exceed the goal of 10 dB and are within the range of the typical performance of active noise-canceling headphones.

6.2 Limitations of Current Design

6.2.1 Hardware

The most significant limitation on the current design is the quality of the microphone and loudspeaker. Although both the EMM-6 and AV-30 offer great performance for their respective prices, a cancellation system utilizing this frequency domain method requires perfect measurement and playback devices. If the microphone cannot detect certain frequencies, then the computed environmental filter transfer function will filter those frequencies out. Similarly, if the loudspeaker cannot reproduce certain frequencies, then it will not be able to attenuate noise at those frequencies. The performance of this cancellation system is heavily debilitated by the worst-performing device.

6.2.2 Mono vs. Stereo

Another limitation of this implementation is that it only takes into account 1 channel (mono). Songs typically have left and right channels that together produce a stereo sound field. Therefore, any unwanted song in the environment will contain a combination of phase-shifted and attenuated audio from both the left and right channels. In this general case, the location of the microphone within the stereo field becomes important. Furthermore, this project does not take into account the multipath effects of an audio waveform reflecting off of objects in the environment.

6.2.3 Microsoft Windows OS

Another major limitation of this design is the non-real-time processing within Windows. The nondeterministic timing within Windows makes audio synchronization a difficult task. This project makes use of microsecond timers to track the amount of time elapsed between key processes; however, even with these corrections, the offset required for perfect synchronization still deviates by a few thousand samples, depending on system resources, *i.e.* available memory and processor bandwidth. One obvious solution would be to switch to a real-time operating system. However, one important goal of this project is to make use of software that most people already have, and most people do not use real-time operating systems.

6.3 Future Improvements

6.3.1 Microphone and Loudspeaker Array

One possible future improvement is to implement the cancellation system with an array of microphones and speakers. This array of devices could potentially expand the zone of silence, or the region in space where significant cancellation occurs. [16] shows a robust MIMO control framework for the compensation of loudspeaker-room responses. It is shown that increasing the number of support loudspeakers results in better loudspeaker-room equalization. It was previously mentioned that the most important loudspeaker requirement is that it have a flat frequency response.

6.3.2 Loudspeaker Equalization with Inverse Filtering

A possible alternative of supplementary loudspeaker equalization technique is to use inverse filtering to "undo" the frequency response of the loudspeaker [17] [18]. While it is possible to attenuate frequencies that are overrepresented in the loudspeaker frequency response, it is not feasible to boost frequencies that the loudspeaker cannot reproduce. Refer to Appendix B, for an investigation on loudspeaker equalization using Wiener deconvolution.

6.3.3 Improved Initial Synchronization

This project relies heavily on the ability of the cross-correlation procedure to accurately locate the recorded excerpt within the original WAVE file. A large error in this step will propagate throughout the rest of the signal chain: First, the frequency content of the mismatched excerpts will likely be different, and thus the calculated environmental filter transfer function will not accurately represent the actual environmental filter response. Additionally, miscalculating the exact location of the excerpt will lead to timing synchronization errors which must be corrected by the dual-sweep algorithm.

Another improvement that must be made on the initial synchronization procedure is to account for the repetitive structure of songs. A poorly made song may have repetitive elements that are exact copies throughout the song. This will cause multiple peaks in the cross-correlation function, which could cause the system to find the wrong starting index location. Unfortunately, there is no easy solution to this problem. To account for this potential repetition, a future implementation of this design must store all possible starting index locations based on a thresholded magnitude of the cross-correlation result. The cancellation system will work as long as the repeated structure of the song remains identical to the found location. However the cancellation will no longer work when the song changes, and at this point, the system must jump to the other possible locations.

6.4 Concluding Remarks

This thesis provides strong evidence that the acoustic cancellation of known audio sources in the frequency domain is a viable method for canceling unwanted songs in an environment. The 8-15 dB broadband attenuation and 10-20 dB low-frequency attenuation compares favorably with the typical performance of existing active noise cancellation systems. A future implementation of this system can only improve upon these results.

REFERENCES

[1] W. Passchier-Vermeer and W. Passchier, "Noise exposure and public health," *Environmental Health Perspectives,* no. 108, pp. 123-131, 2010.

[2] O. Jones, "Challenges Await In Noise-Cancelling Headset Design - Electronic Design," 13 June 2013. [Online]. Available: http://electronicdesign.com/analog/challenges-await-noise-cancelling-headset-design. [Accessed 28 April 2014].

[3] Wolfson Microelectronics, *"Ambient Noise Cancellation for Headphones and Handsets," Application Note,* Aug. 2010, Rev 1.1.

[4] B. Rafaely, "Active Noise Reducing Headset," in *International Congress and Exhibition on Noise Control Engineering*, The Hague, The Netherlands, 2001.

[5] V. Sethia, "Noise Cancellation in Headphones," Electronic Systems Group, EE Dept, IIT Bombay, Nov. 2002.

[6] J. C. Doyle, B. A. Francis and A. R. Tannenbaum, Feedback Control Theory, Maxwell MacMillan International, 1992.

[7] Y. Xiao and J. Wang, "A New Feedforward Hybrid Active Noise Control System," *IEEE Signal Processing Letters,* vol. 18, no. 10, pp. 591-594, 2011.

[8] G. Sessler and J. West, "Self-biased condenser microphone with high capacitance," *The Journal of the Acoustical Society of America,* vol. 34, no. 11, p. 1787, 1962.

[9] A. Ohyagi, D Adachi, H Kodama and Y Yasuno, "Electret Condenser Microphones for Measurement Instruments," *2008 13th International Symposium on Electrets,* p. C135, 2008.

[10] Dayton Audio, *"EMM-6 Measurement Microphone," EMM-6 Datasheet.*

[11] Blue Microphones, *"Blue Icicle Datasheet," Datasheet,* 2008.

[12] R. W. a. S. E. Rafael Gonzalez, Digital Image Processing Using Matlab., Prentice Hall, 2003.

[13] M-Audio, *"M-Audio AV-30 User Guide," Datasheet,* 2010.

[14] A. R. Collins, "Zoom FFT," [Online]. Available: http://www.arc.id.au/ZoomFFT.html. [Accessed 15 May 2014].

[15] C. Zeitnitz, *WaveIO Version 0.70: A Soundcard interface for LabView (Help File),* Nov. 2010.

[16] L.-J. Brännmark, A. Bahne and A. Ahlén, "Compensation of Loudspeaker–Room Responses in a Robust MIMO Control Framework," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 6, pp. 1201-1216, 6 June 2013.

[17] S. Hendry, *"Loudspeaker Equalisation to Compensate for Low Frequency Room Acoustics." Ph.D. Dissertation, School of Physics, Univ. of Edinburgh,* 2007.

[18] G. S. Norcross, A. G. Soulodre and C. M. Lavoie, "Subjective Investigations of Inverse Filtering," *Journal of the Audio Engineering Society*, vol. 52, no. 10, pp. 1003-1026, 2004.

[19] National Instruments, *LabVIEW 2012 Help,* 2012.

[20] G. Heinzel, A. Rüdiger and R. Schilling, "Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows," Max Planck Institute (MPI) für Gravitationsphysik / Laser Interferometry & Gravitational Wave Astronomy, 2002.

APPENDIX A. LABVIEW SUB-VIS

A.1 Microphone Data Acquisition

LabVIEW uses three built-in functions to record microphone audio. The following is a detailed description of the functionality and IO of each VI according to the LabVIEW 2012 help file [19].

A.1.1 Sound Input Configure VI

This VI configures a sound input device to acquire data and send the data to the buffer.

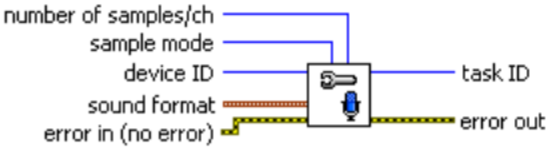


Figure A-1: Sound Input Configure.vi IO for microphone input.

number of samples/ch: specifies the number of samples per channel in the buffer. Use a large number of samples for continuous operations. Use a smaller number of samples if you want to use less memory.

sample mode: specifies whether the VI acquires samples just once (**Finite Samples**) or continuously (**Continuous Samples**). In **Finite Samples** mode, call Sound Input Read only until you have written the number of samples specified in **number of samples/ch**. In **Continuous Samples** mode, you can call Sound Input Read repeatedly as needed.

device ID: is the input or output device you access for a sound operation. In general, most users should select the default value of 0. The value ranges from 0 to *n*–1, where *n* is the number of input or output devices on the computer.

sound format: sets the acquisition rate, the number of channels, and the bits per sample of the sound operation. The values for each of these controls is dependent on your sound card.

sample rate (S/s) sets the sampling rate for the sound operation.

number of channels specifies the number of channels. This input can accept as many channels as the sound card supports. For most sound cards 1 is Mono and 2 is Stereo.

bits per sample specifies the quality of each sample in bits. Common resolutions are 16 bits and 8 bits. The default is 16 bits.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

task ID returns an identification number associated with the configuration on the specified device. You can pass **task ID** to other sound operation VIs.

error out contains error information. This output provides standard error out functionality.

A.1.2 Sound Input Read VI

This VI reads data from a sound input device.

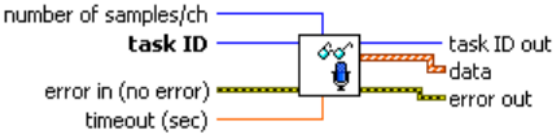


Figure A-2: Sound Input Read.vi IO for microphone input.

number of samples/ch specifies the number of samples per channel to read from the buffer.

task ID is the sound operation from the configured device you want to manipulate or input. You generate **task ID** with the Sound Input Configure VI.

error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

timeout (sec) specifies the time, in seconds, that the VI waits for the sound operation to complete. This VI returns an error if the time elapses. The default is 10. If you set timeout (sec) to -1, the VI waits indefinitely.

task ID out is the manipulated sound operation originally passed to the **task ID** parameter.

data reads any sound data from the internal buffers. For multi-channel sound data, **data** is an array of waveforms where each element of the array is a single channel.

> **t0** is the start time for the first sample read. LabVIEW approximates the initial time that the first sample was read because the sound file does not contain this data.
>
> **dt** is the sampling interval. It is the inverse of the sampling rate specified by the Sound Input Config VI.
>
> **Y** is the sound data. If the array data type is a floating-point numeric, **Y** ranges from -1.0 to 1.0.
>
> The specified data type determines the range of values for the sound data.

error out contains error information. This output provides standard error out functionality.

A.1.3 Sound Input Clear VI

This VI stops acquisition of data, clears the buffer, returns the task to the default state, and clears the resources associated with the task. The task becomes invalid.

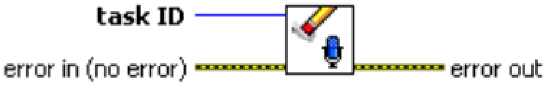


Figure A-3: Sound Input Clear.vi IO for microphone input.

task ID is the sound operation from the configured device you want to manipulate or input. You generate **task ID** with the Sound Input Configure VI.
error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
error out contains error information. This output provides standard error out functionality.

A.2 **Audio Playback Via WaveIO v0.70**

WaveIO is a soundcard interface for LabVIEW. While the default sound library *lvsound.dll* has some limitations in I/O channels, sampling rates, and resolutions, the WaveIO library overcomes these shortcomings by providing flexible access to the soundcard with all possible hardware supported features. The connection between the LabView VI and actual sound hardware is done via the standard Windows API. Therefore, all sound systems providing drivers for Windows should be accessible via the WaveIO interface. This applies to on-board sound, PCI sound cards as well as USB sound systems [15]. WaveIO also allows for a direct write to the output buffer with low latency.

The WaveIO package contains the following Soundcard Interface VIs in waveio.llb:

WaveIO_Open: opens a sound device for recording or playback. The device has to be started before actually reading/writing data to it. The open will return an error if the device is not existing, already open or the selected sound format is not supported by the device. A watchdog thread is started, if the specified timeout is greater than zero. The watchdog will close the device, if the no call to DLL for the specific device is recorded within the given time window. For each device (and I/O mode) a separate watchdog thread is started.

WaveIO_Start: starts the opened sound device.

WaveIO_Play: sends the provided data to the soundcard. An error will occur, if the size of the provided data does not match the buffer setting of the open command.

WaveIO_Record: waits and retrieves data from the soundcard.

WaveIO_Stop: stops the sound device. Required before closing the device.

WaveIO_Close: closes the sound device.

Figure A-4 shows the VI used in this project that formats the output audio (sample rate, number of channels, bits per sample, and buffer size) and configures WaveIO for audio playback. In the project signal chain, this block goes between the first cancellation block and the audio playback loop (before the overlap and save method as well). The sample rate is set to 44.1 kHz, the number of channels is set to 2, the bit size is set to 16 bits, and the buffer size is set to 4000 samples.

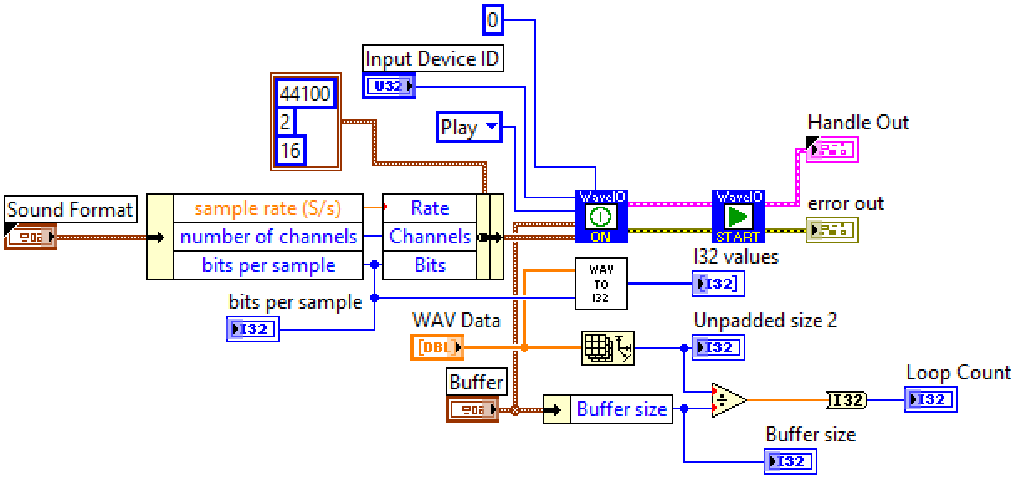


Figure A-4: Play Data Format & Setup.vi block diagram.

A.3 Built-in Signal Processing VIs

This project utilizes many built-in signal processing VIs. These VIs include:

- Cross-correlation
- FFT and Inverse FFT
- Median filter
- Butterworth filter
- Smoothing filter
- Windowing (Hanning Window)
- Interpolation

A.3.1 Cross-Correlation VI

This VI computes the cross correlation of the input sequences **X** and **Y**. The cross correlation *R*$_{xy}$(*t*) of the sequences *x*(*t*) and *y*(t) is defined by the following equation:

$$R_{xy}(t) = x(t)\otimes y(t) = \int_{-\infty}^{\infty} x^*(\tau)\cdot y(t+\tau)d\tau \qquad \textbf{Eq. A-1}$$

The discrete implementation is as follows:

$$x_j = 0, j < 0\ or\ j \geq N$$
$$y_j = 0, j < 0\ or\ j \geq M$$

$$h_j = \sum_{k-0}^{N-1} x_k^* \cdot y_{j+k}$$

$$\text{for j} = -(\text{N}-1), -(N-2), \ldots, -1, 0, 1, \ldots, (M-2), (M-1) \qquad \textbf{Eq. A-2}$$

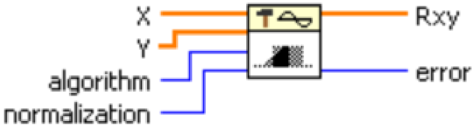


Figure A-5: CrossCorrelation.vi IO.

<table>
<tr><td><b>X</b> is the first input sequence.</td></tr>
<tr><td><b>Y</b> is the second input sequence.</td></tr>
<tr><td><b>algorithm</b> specifies the correlation method to use. When <b>algorithm</b> is <b>direct</b>, this VI computes the cross correlation using the direct method of linear correlation. When <b>algorithm</b> is <b>frequency domain</b>, this VI computes the cross correlation using an FFT-based technique. If <b>X</b> and <b>Y</b> are small, the <b>direct</b> method typically is faster. If <b>X</b> and <b>Y</b> are large, the <b>frequency domain</b> method typically is faster. Additionally, slight numerical differences can exist between the two methods.
<table>
<tr><td>0</td><td>direct</td></tr>
<tr><td>1</td><td>frequency domain (default)</td></tr>
</table></td></tr>
<tr><td><b>normalization</b> specifies the normalization method to use to compute the cross correlation between <b>X</b> and <b>Y</b>.
<table>
<tr><td>0</td><td>none (default)</td></tr>
<tr><td>1</td><td>unbiased</td></tr>
<tr><td>2</td><td>biased</td></tr>
</table></td></tr>
<tr><td><b>Rxy</b> is the cross correlation of <b>X</b> and <b>Y</b>.</td></tr>
<tr><td><b>error</b> returns any error or warning from the VI.</td></tr>
</table>

A.3.2 FFT VI

This VI computes the fast Fourier transform (FFT) of the input sequence **X**. For 1D signals, the FFT VI computes the discrete Fourier transform (DFT) of the input sequence with a fast Fourier transform algorithm. The 1D DFT is defined as:

$$Y_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N} \text{ for } n = 0, 1, 2, \ldots, \mathrm{N} - 1 \qquad \textbf{Eq. A-3}$$

where *x* is the input sequence, *N* is the number of elements of *x*, and *Y* is the transform result. The frequency resolution is:

$$\Delta f = \frac{f_s}{N} \qquad \textbf{Eq. A-4}$$

where f_s is the sampling frequency.

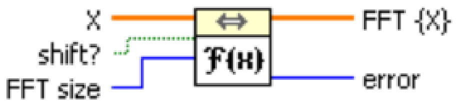


Figure A-6: FFT.vi IO.

X is a real vector.
shift? specifies whether the DC component is at the center of **FFT {X}**. The default is FALSE.
FFT size is the length of the FFT you want to perform. If **FFT size** is greater than the number of elements in **X**, this VI adds zeros to the end of **X** to match the size of **FFT size**. If **FFT size** is less than the number of elements in **X**, this VI uses only the first *n* elements in **X** to perform the FFT, where *n* is **FFT size**. If **FFT size** is less than or equal to 0, this VI uses the length of **X** as the **FFT size**.
FFT {X} is the FFT of **X**.
error returns any error or warning from the VI.

A.3.3 Inverse FFT VI

This VI computes the inverse discrete Fourier transform (IDFT) of the input sequence **FFT {X}**. For a 1D, *N*-sample, frequency domain sequence *Y*, the IDFT is defined as:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} Y_k e^{j2\pi kn/N} \text{ for } n = 0, 1, 2, \dots, N-1 \qquad \textbf{Eq. A-5}$$

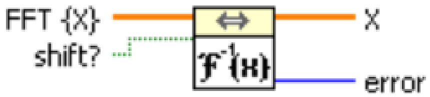


Figure A-7: Inverse FFT.vi IO.

FFT {X} is the complex valued input sequence.
shift? specifies whether the DC component is at the center of **FFT {X}**. The default is FALSE.
X is the inverse complex FFT of **FFT{X}**.
error returns any error or warning from the VI.

A.3.4 Median Filter VI

This VI applies a median filter of rank to the input sequence X, where rank is right rank if right rank is greater than zero, or left rank if right rank is less than zero. The Median Filter VI obtains the elements of **Filtered X** using the following equation:

$$y_i = \text{Median}(J_i) \quad \text{for } i = 0, 1, 2, \dots, n-1, \qquad \textbf{Eq. A-6}$$

where *Y* represents the output sequence **Filtered X**, *n* is the number of elements in the input sequence **X**, J_i is a subset of the input sequence **X** centered about the i^{th} element of **X**, and the indexed elements outside the range of **X** equal zero. The following equation describes J_i:

$$J_i = \{x_{i-rl}, x_{i-rl+1}, K, x_{i-1}, x_i, x_{i+1}, K, x_{i+rr-1}, x_{i+rr}\}, \qquad \textbf{Eq. A-7}$$

where *rl* is the filter **left rank**, and *rr* is the filter **right rank**. The value given to either **left rank** or **right rank** defines the order of the median filter.

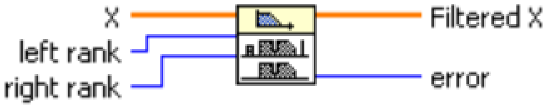


Figure A-8: Median Filter.vi IO.

X is the input signal to filter. The number of elements, *n*, in **X** must be greater than **right rank**. If the number of elements in **X** is less than or equal to **right rank**, the VI sets **Filtered X** to an empty array and returns an error.
left rank is the number of elements used to compute the median filter to the left side. **left rank** must be greater than or equal to 0. The default is 2.
right rank is the number of elements used to compute the median filter to the right side. If **right rank** is less than 0, the VI assumes **right rank** is equal to **left rank**. **right rank** must be less than **X**. The default is -1.
Filtered X is the output array of filtered samples. The size of this array is the same as the input array **X**.
error returns any error or warning from the VI.

A.3.5 Butterworth Filter VI

This VI Generates a digital Butterworth filter by calling the Butterworth Coefficients VI.

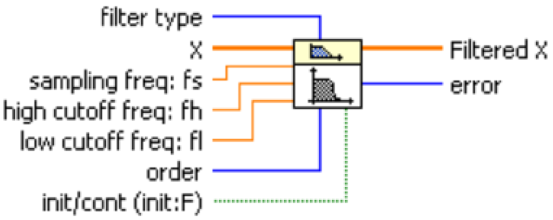


Figure A-9: Butterworth Filter.vi IO.

<table>
<tr><td><b>filter type</b> specifies the passband of the filter.
<table>
<tr><td>0</td><td>Lowpass</td></tr>
<tr><td>1</td><td>Highpass</td></tr>
<tr><td>2</td><td>Bandpass</td></tr>
<tr><td>3</td><td>Bandstop</td></tr>
</table></td></tr>
<tr><td><b>X</b> is the input signal to filter.</td></tr>
<tr><td><b>sampling freq: fs</b> is the frequency in Hz at which you want to sample <b>X</b> and must be greater than 0. The default is 1.0 Hz. If <b>sampling freq: fs</b> is less than or equal to 0, this VI sets <b>Filtered X</b> to an empty array and returns an error.</td></tr>
<tr><td><b>high cutoff freq: fh</b> is the high cutoff frequency in Hz. The default is 0.45 Hz. The VI ignores this parameter when <b>filter type</b> is 0 (Lowpass) or 1 (Highpass). When <b>filter type</b> is 2 (Bandpass) or 3 (Bandstop), <b>high cutoff freq: fh</b> must be greater than <b>low cutoff freq: fl</b> and observe the Nyquist criterion.</td></tr>
<tr><td><b>low cutoff freq: fl</b> is the low cutoff frequency in Hz and must observe the Nyquist criterion. The default is 0.125 Hz. If <b>low cutoff freq: fl</b> is less than or equal to 0 or greater than half the value of <b>sampling freq: fs</b>, the VI sets <b>Filtered X</b> to an empty array and returns an error. When <b>filter type</b> is 2 (Bandpass) or 3 (Bandstop), <b>low cutoff freq: fl</b> must be less than <b>high cutoff freq: fh</b>.</td></tr>
<tr><td><b>order</b> specifies the filter order and must be greater than 0. The default is 2. If <b>order</b> is less than or equal to 0, the VI sets <b>Filtered X</b> to an empty array and returns an error.</td></tr>
<tr><td><b>init/cont</b> controls the initialization of the internal states. The default is FALSE. The first time this VI runs or if <b>init/cont</b> is FALSE, LabVIEW initializes the internal states to 0. If <b>init/cont</b> is TRUE, LabVIEW initializes the internal states to the final states from the previous call to this instance of this VI. To process a large data sequence that consists of smaller blocks, set this input to FALSE for the first block and to TRUE for continuous filtering of all remaining blocks.</td></tr>
<tr><td><b>Filtered X</b> is the output array of filtered samples.</td></tr>
<tr><td><b>error</b> returns any error or warning from the VI.</td></tr>
</table>

A.3.6 Smoothing Filter VI

This VI generates a smoothing filter and yields forward-only (FIR) coefficients. This option is available only when you select **Smoothing** from the **Filtering Type** pull-down menu within the Filter Express VI. You can then choose to specify the smoothing filter as moving average or exponential. **Figure A-11** shows the smoothing filter configuration window within the Filter Express VI.

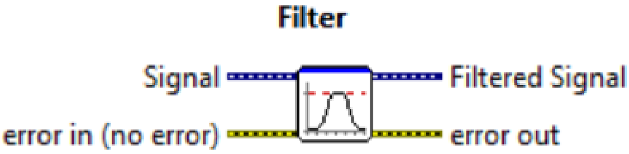
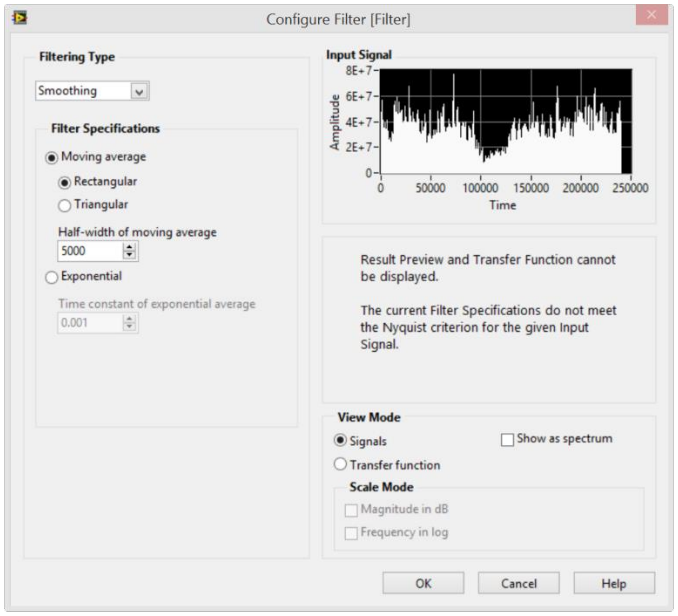


Figure A-10: Filter Express VI IO.



Filtering Type: Smoothing

Filter Specifications

● Moving average

● Rectangular

○ Triangular

Half-width of moving average: 5000

○ Exponential

Time constant of exponential average: 0.001

Result Preview and Transfer Function cannot be displayed.

The current Filter Specifications do not meet the Nyquist criterion for the given Input Signal.

View Mode

● Signals □ Show as spectrum

○ Transfer function

Scale Mode

□ Magnitude in dB

□ Frequency in log

Figure A-11: Smoothing filter configuration window within Filter Express VI.

A.3.7 Hanning Window VI

This VI applies a Hanning window to the input signal **X**. The Hanning window is defined by the following equation:

$$y_i = \frac{1}{2} x_i [1 - \cos\left(\frac{2\pi i}{N-1}\right)] \text{ for } i = 0, 1, 2, \dots, N-1 \qquad \textbf{Eq. A-8}$$

The following figure shows the Hanning window and its Fourier transform [20].

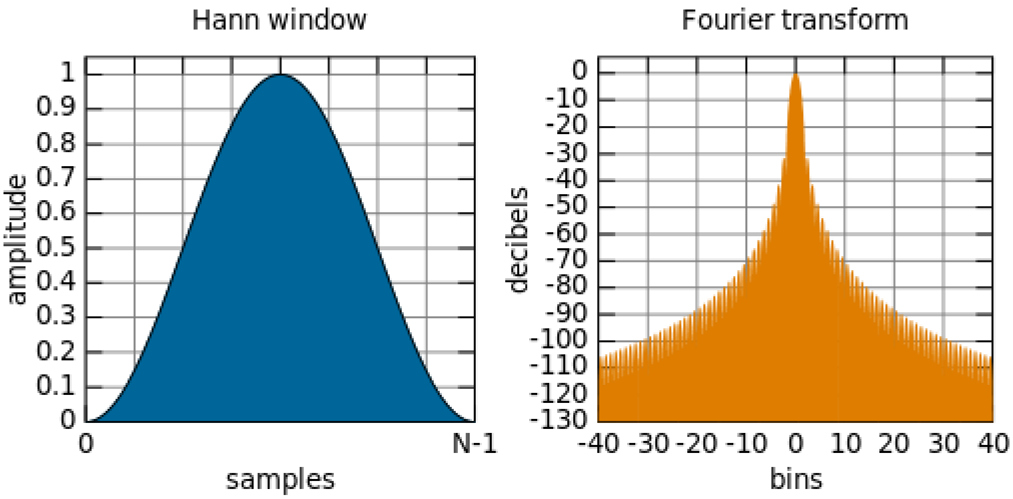


Figure A-12: Hann (Hanning) window and its Fourier transform [20].

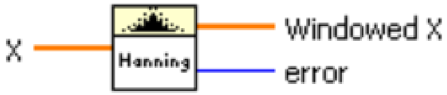


Figure A-13: Hanning window.vi IO.

X is a real vector.
Windowed X is the input signal with the window applied.
error returns any error or warning from the VI.

A.3.8 Interpolation VI

This VI linearly interpolates a decimal **y value** from an **array of numbers or points** using a **fractional index or x** value.

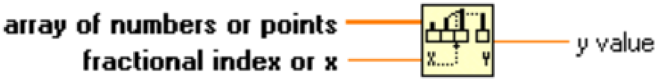


Figure A-14: Interpolate 1D Array.vi IO.

array of numbers or points can be an array of numbers or an array of points where each point is a cluster of x and y coordinates. If this input is an array of points, the function uses the first element in the cluster (x) to obtain a fractional index by linear interpolation. The function then uses this fractional index to compute the output **y value** from the second cluster element (y).

fractional index or x is the index or x-value at which the function should return a y-value. For example, if **array of numbers or points** contains two double-precision, floating-point numeric values, 5 and 7, and **fractional index or x** is set to 0.5, the function returns 6.0, which is halfway between the values at elements 0 and 1.

If **array of numbers or points** contains an array of data point sets, the function returns the linearly interpolated **y value** at the x-value corresponding to **fractional index or x**. For example, if the array contains two points, (3,7) and (5,9), and **fractional index or x** is set to 3.5, the function returns 7.5.

fractional index or x does not interpolate beyond the bounds of an array or data point set. For example, if the parameter is set lower than the first element or x-value in an array, the function returns the value of the first element or the y-value of the first data point. Similarly, if the parameter is set too high, the function returns the value in the final element or the final y-value. **fractional index or x** must be located directly on a point or between two points for the function to work correctly.

y value is the interpolated value of the element at the fractional index or the interpolated y-value of the fractional data point, in **array of numbers or points**.

A.4 Additional User-Created VIs

This project requires some miscellaneous VIs that are not completely discussed in Chapter 4 but are critical to the functionality of the system.

A.4.1 Bit Weight Converter VI

One such VI is the bit weight converter VI, which converts the WAVE values (double precision floating point) to 32-bit integer values and vice versa. This function is necessary because WaveIO requires 32-bit integers for audio playback, while all FFT computations are with double precision floating point values. **Figure A-15** shows the VI for the conversion from WAVE file to 32-bit integers according to the following equation.

$$\text{I32}_{\text{values}} = \text{round}\left(\text{floor}\left(\frac{\text{WAV}_{\text{values}}}{2\cdot\text{A}_{\text{max}}/2^{\text{n}}}\right)\right) \qquad \textbf{Eq. A-9}$$

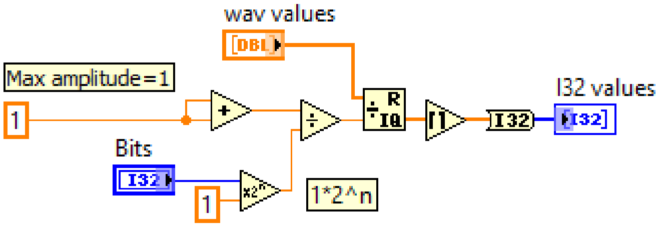


Figure A-15: Bit Weight Converter (WAVE to I32).vi block diagram.

A.4.2 Sweep Parameters VI(s)

The dual-sweep algorithm discussed in Section 4.8 uses nested VIs to compute important values required to produce a successful sweep based on the given sweep parameters. These computations are located in sub-VIs to minimize the visual clutter within the main VI.

Figure A-16 shows the block diagram of the VI which computes values such as the total size of the microphone recording required to capture the entire sweep. It also show the calculation for the equivalent step size in the microphone recording that corresponds to one step of the sweep.

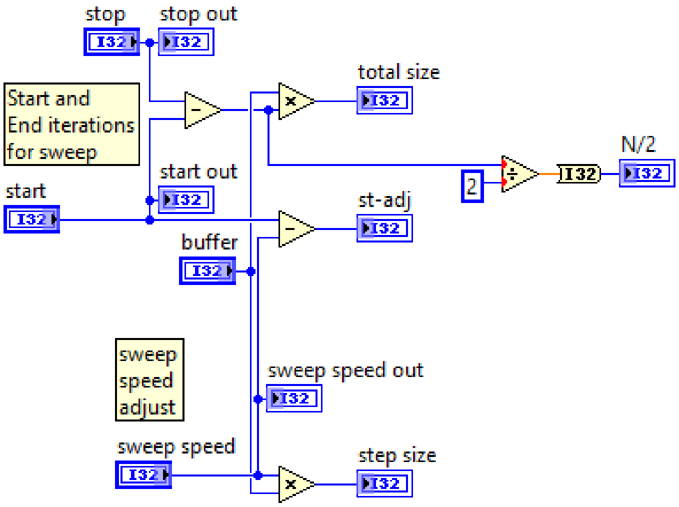


Figure A-16: Sweep parameter calculation block #1.

Figure A-17 shows the block diagram for the VI that performs the Boolean comparisons which determine the start of the sweep and the continuation of the sweep based on the sweep parameters.

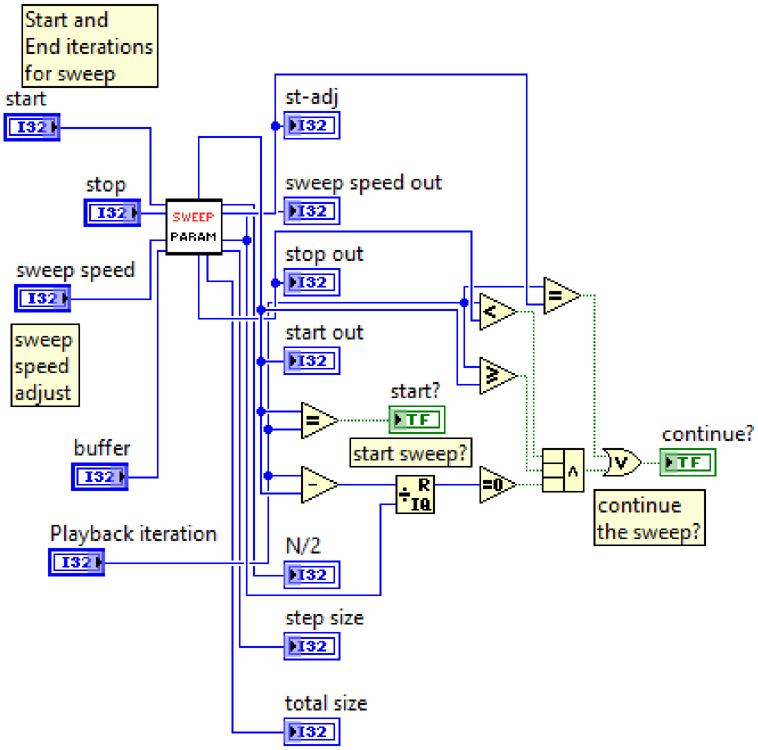


Figure A-17: Sweep parameter calculation block #2.

APPENDIX B. LOUDSPEAKER EQUALIZATION INVESTIGATION

B.1 Overview

In an attempt to equalize the loudspeaker frequency response, we investigate using an inverse filtering method known as Wiener deconvolution. The frequency response of a Wiener deconvolution filter is defined by **Eq. B-1** [12].

$$W(f) = \frac{H^{*}(f)}{|H(f)|^{2} + 1/SNR} \qquad \textbf{Eq. B-1}\ [12]$$

where $H(f)$ is the frequency response of the loudspeaker and SNR is its signal-to-noise ratio. We can simplify **Eq. B-1** by approximating $1/SNR$ with a constant. We can see that as the SNR goes to infinity, the Wiener deconvolution filter becomes an inverse filter.

B.1.1 LabVIEW Implementation

The first step is to obtain the loudspeaker frequency response $H(f)$. We do this by simultaneously playing and recording a logarithmic frequency sweep from 20 Hz to 20 kHz. We collect multiple measurements to obtain an average. **Figure B-1** shows the LabVIEW block diagram that collects data from 5 separate chirps.

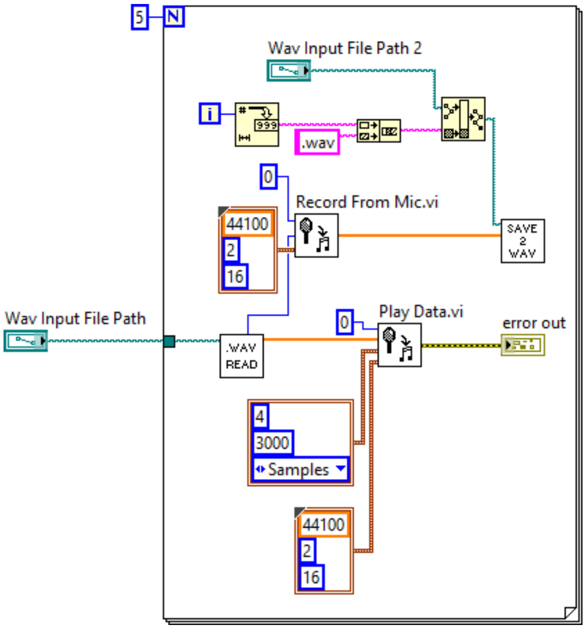


Figure B-1: Data collection block diagram for multiple 20 Hz to 20 kHz chirps.

Figure B-2 shows the block diagram that finds the average loudspeaker frequency response by comparing the FFT of the average of 5 microphone recordings with the FFT of the original chirp signal.

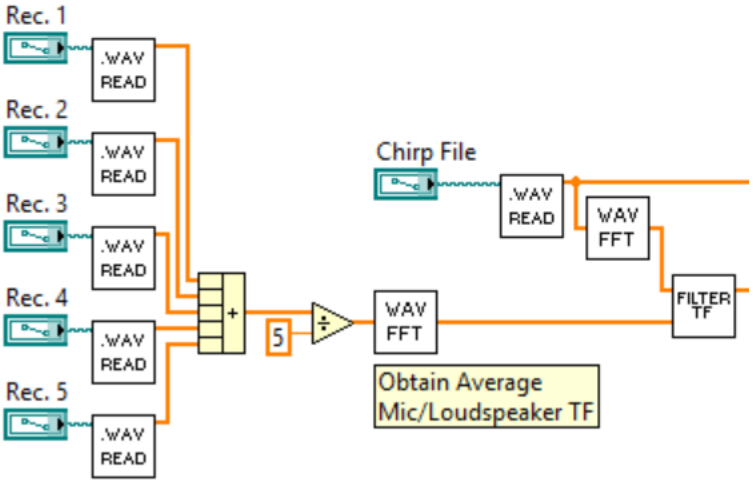


Figure B-2: Loudspeaker average transfer function block diagram.

Figure B-3 shows the resulting frequency response of the M-Audio AV-30 loudspeaker. **Figure B-4** shows the same plot with the amplitude in dB. Note that the amplitude values depend on the volume of the chirp signal as well as the gain of the microphone.

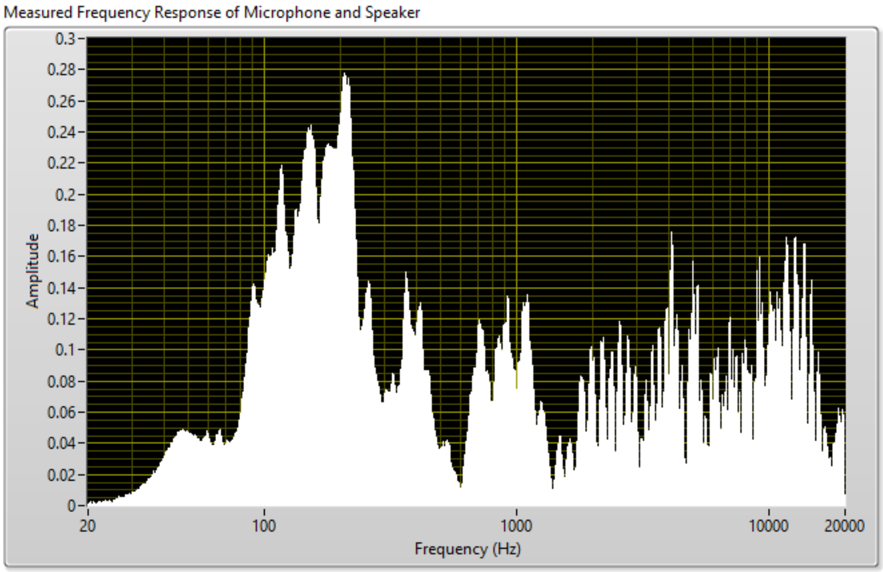


Figure B-3: Average measured frequency response of M-Audio AV-30 loudspeaker.

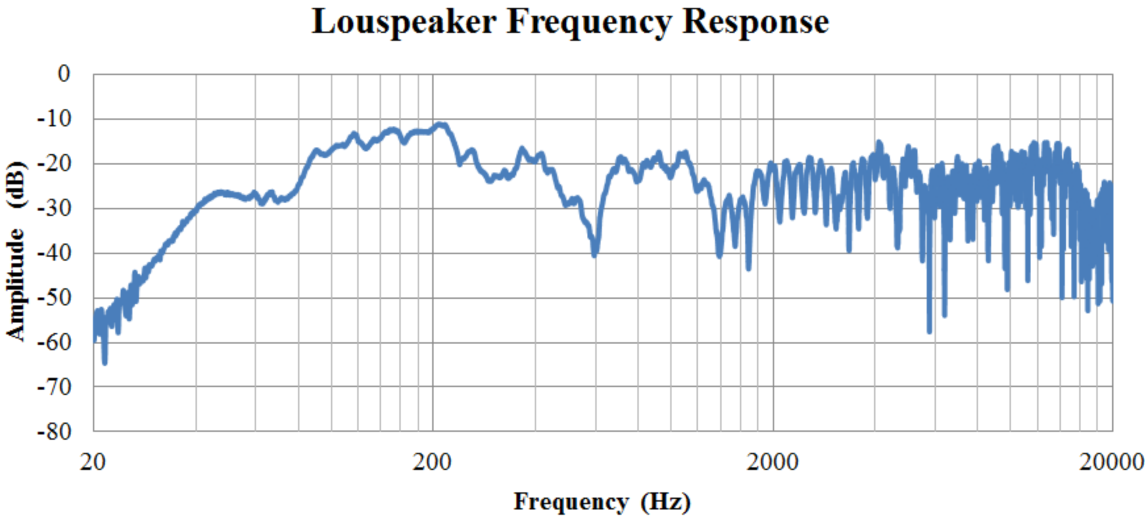


Figure B-4: Loudspeaker frequency response from **Figure B-3** in dB.

We must resize the transfer function to 262144 so that it matches with the size of the transfer function in Section 4.3. We use LabVIEW's built-in 1D interpolation VI and interpolate both the magnitude and phase components of the complex transfer function. **Figure B-5** shows the block diagram of this interpolation step.

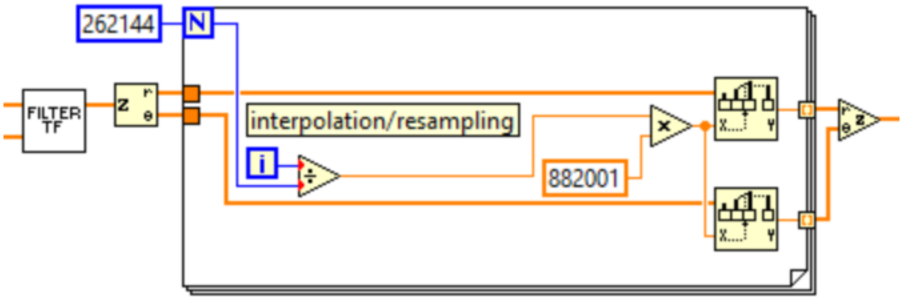


Figure B-5: Transfer function interpolation block diagram.

Next, we apply the Wiener deconvolution filter from **Eq. B-1** to the interpolated transfer function, shown in **Figure B-6** below. Here we choose a constant value of 0.002 for $1/SNR$. This is equivalent to a SNR value of approximately 54 dB.

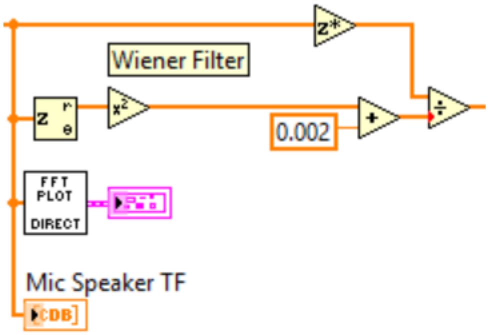


Figure B-6: Wiener deconvolution block diagram.

The last steps are to window (Hanning, size 5000) and filter (band-pass Butterworth, 50 Hz – 12 kHz, 5th order) the iFFT of the Wiener deconvolution filter values and to normalize the final frequency response, all shown in **Figure B-7**.

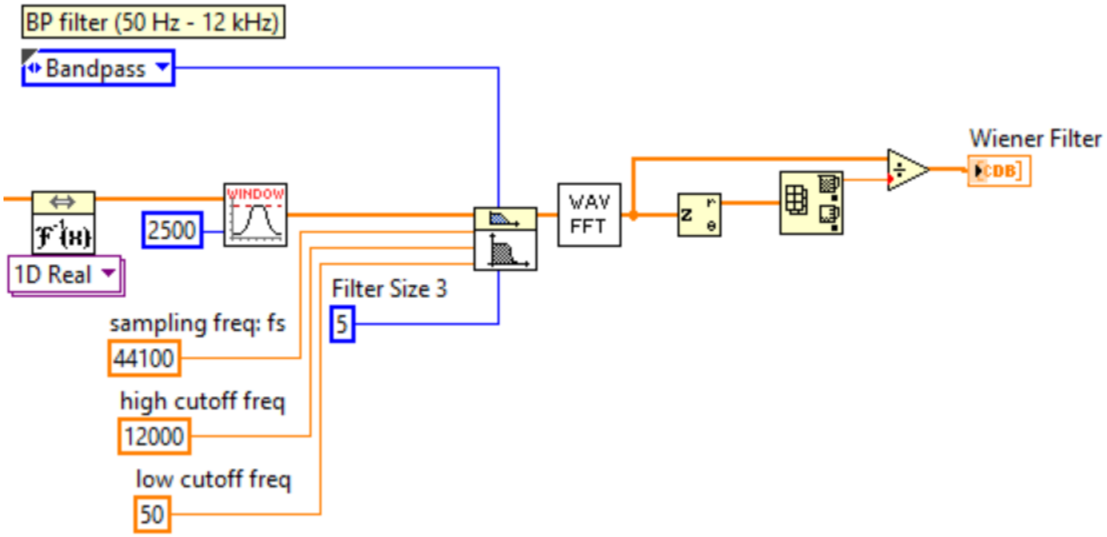


Figure B-7: Block diagarm of windowing, band-pass filtering, and normalization of Wiener deconvolution filter.

Figure B-8 shows the resulting Wiener deconvolution filter for a large window size of M=80000. Notice how it mirrors the loudspeaker frequency response from **Figure B-3** – *i.e.* frequencies at which the magnitude of the loudspeaker frequency response is large show attenuation, while frequencies at which the magnitude of the loudspeaker frequency response is small show little to no attenuation. The filter is designed to only attenuate frequencies because amplification could introduce unwanted distortion. From this, we expect the amplitude of the equalized loudspeaker audio to be reduced.

When this large filter is applied to the chirp signal, the resulting audio exhibits harmonic distortion and it sounds very different from the original chirp.

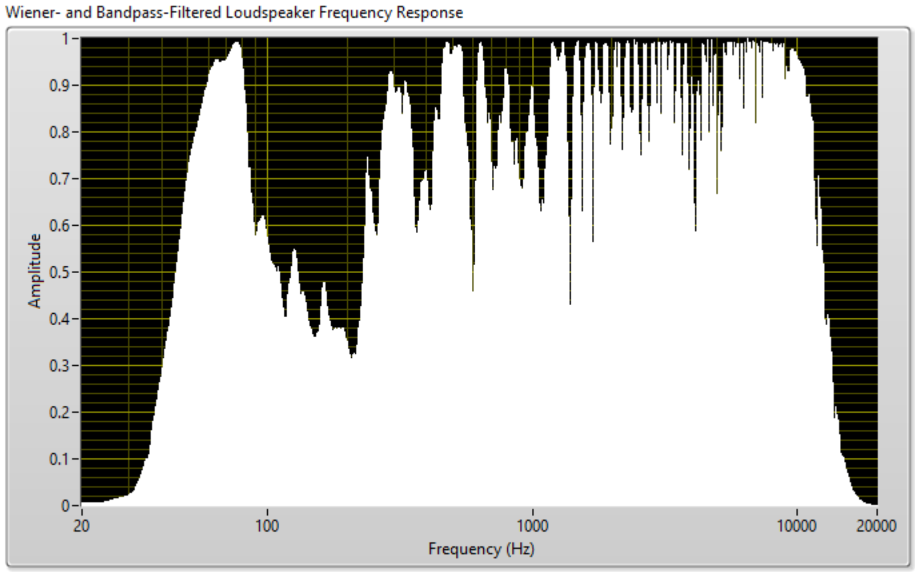


Figure B-8: Normalized Wiener deconvolution filter with window size M=80000.

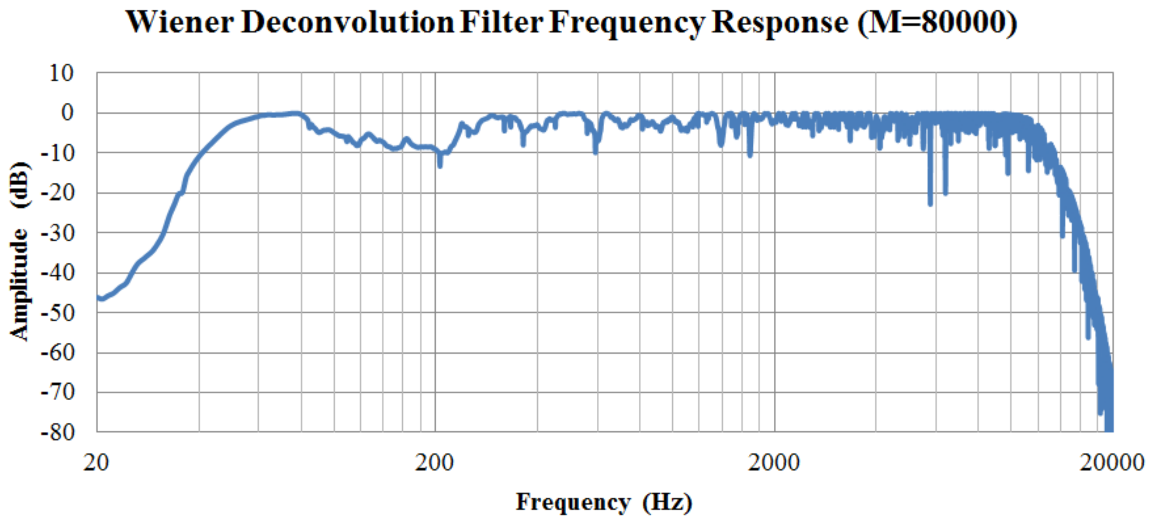


Figure B-9: Wiener deconvolution filter from **Figure B-8** in dB.

Figure B-10 shows the resulting Wiener deconvolution filter for a window size of M=5000. We observe that we lose significant resolution in the low-frequency range, characterized by the smooth low-frequency response. There is a trade-off between the window size (which must be small enough to limit the effects of circular convolution) and the low-frequency resolution of the Wiener deconvolution filter.

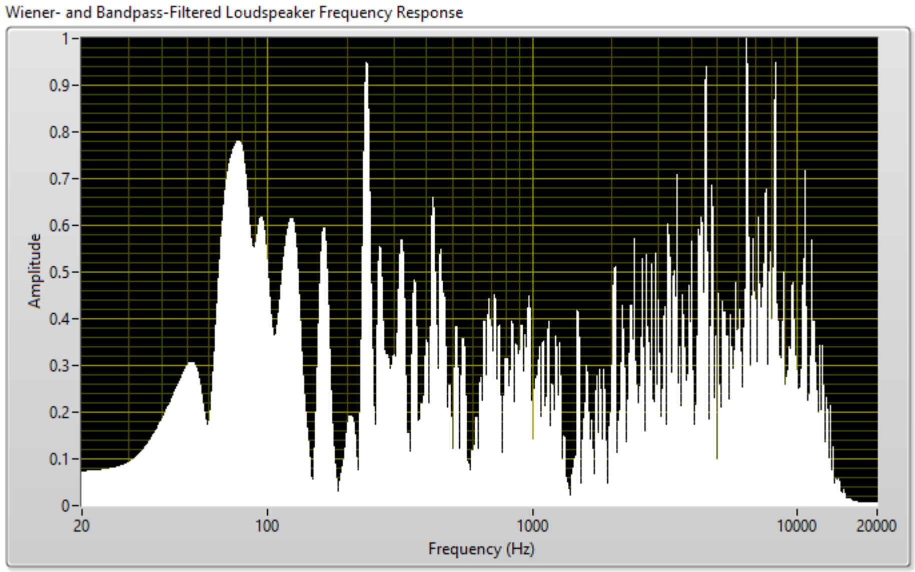


Figure B-10: Normalized Wiener deconvolution filter with window size M=5000.

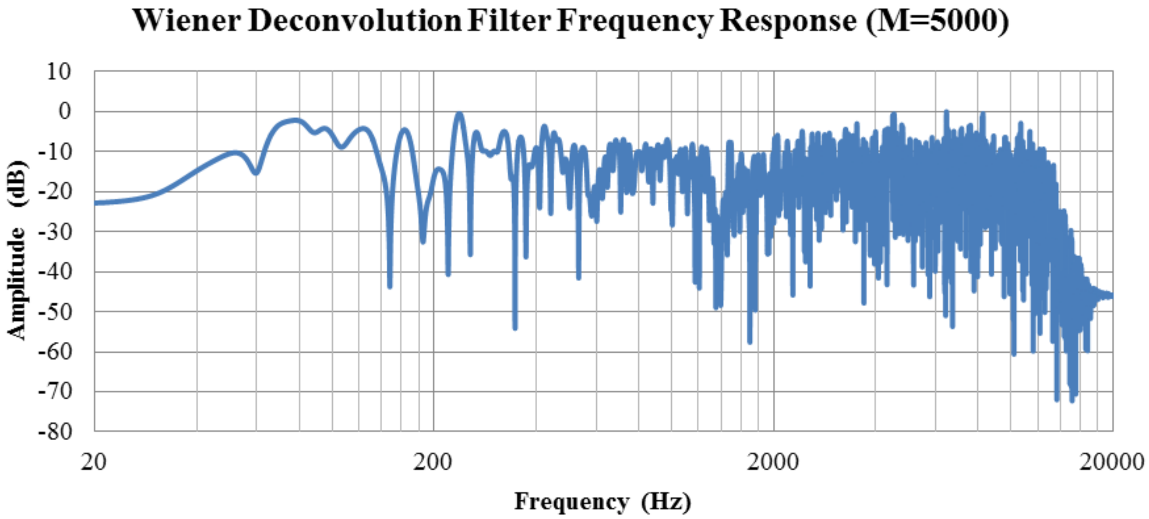


Figure B-11: Wiener deconvolution filter from **Figure B-10** in dB.

B.1.2 Test Results

We apply the Wiener deconvolution filter determined in the previous section to the frequency chirp signal, re-play and re-measure the filtered audio, and observe the resulting transfer function to determine if this deconvolution step improves the loudspeaker frequency response.

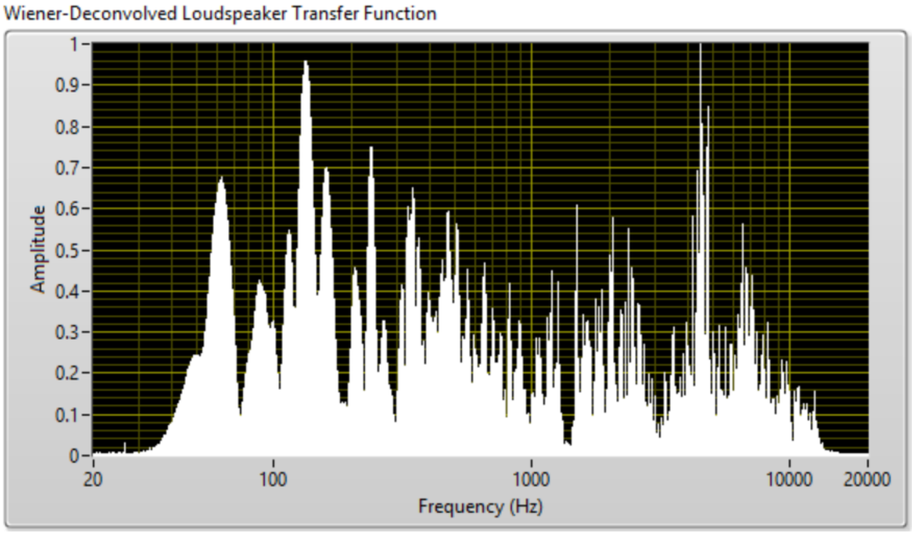


Figure B-12: Wiener-deconvolved loudspeaker transfer function (M=5000).

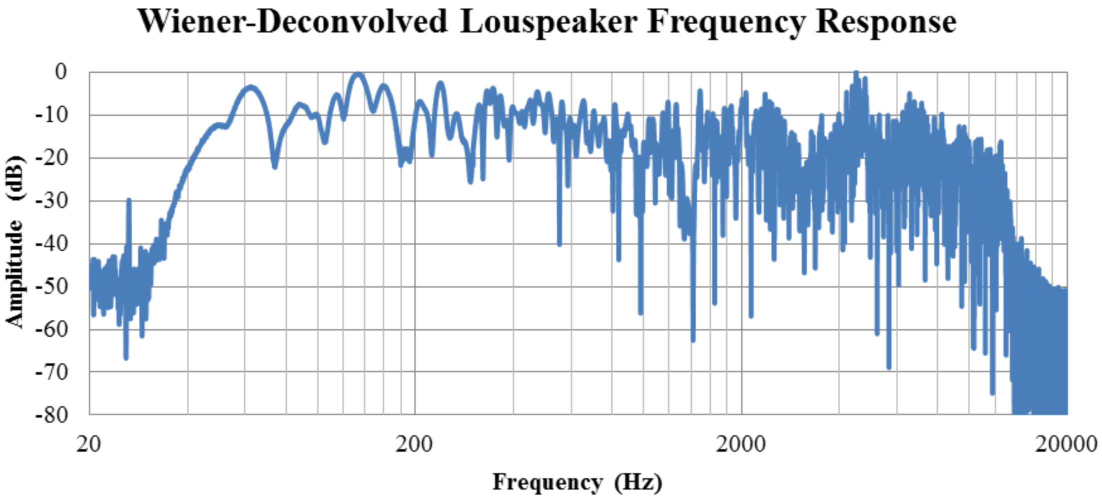


Figure B-13: Frequency response from **Figure B-12** in dB.

Figure B-14 shows a comparison between the loudspeaker frequency response before and after Wiener deconvolution. We observe marginal improvement in the low-frequency response as well as better performance in the frequency range between 400-600 Hz. However, there are many frequencies at which the magnitude response is very small. We expect the frequency nulls to remain uncompensated for as this filter does not perform any amplification. This result is desired because the loudspeaker will not be equalized by boosting frequencies which it cannot reproduce.

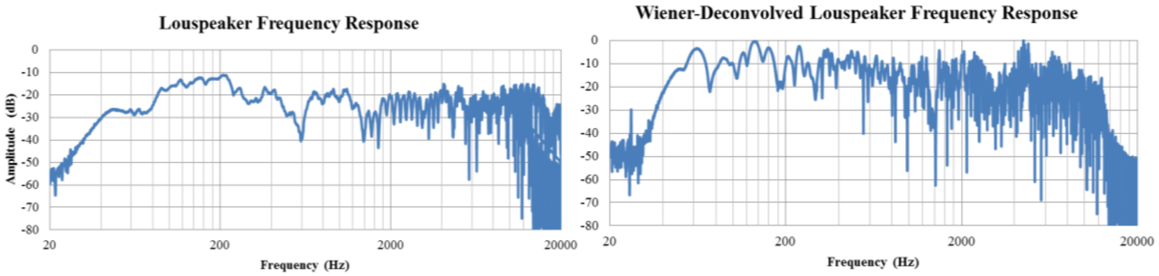


Figure B-14: Comparison between unfiltered (left) and filtered (right) loudspeaker frequency responses.