ADAPTIVE ACTIVE SNORE NOISE CONTROL USING MICROPHONE AND LOUDSPEAKER ARRAYS

by

Nathan Walker

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Todd Moon, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Jonathan Phillips, Ph.D. Committee Member	David F. Feldon, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2026

Copyright © Nathan Walker 2026

All Rights Reserved

ABSTRACT

Adaptive Active Snore Noise Control Using Microphone and Loudspeaker Arrays

by

Nathan Walker, Master of Science

Utah State University, 2026

Major Professor: Todd Moon, Ph.D.
Department: Electrical and Computer Engineering

Snoring is a common source of sleep disturbance that negatively impacts partner sleep quality and relationship well-being. Existing mitigation approaches, including continuous positive airway pressure (CPAP) devices, nasal strips, and earplugs, either require wearable hardware or provide limited attenuation of the low-frequency acoustic energy characteristic of snoring.

This thesis proposes a non-wearable acoustic noise mitigation system using near-field microphone and loudspeaker array processing to actively reduce perceived snoring at a listener's ear. The system employs adaptive beamforming and active noise control techniques to generate a spatially localized anti-noise field directed toward the sleeping partner while minimizing disturbance to the surrounding environment. Near-field array modeling enables improved low-frequency cancellation performance compared to conventional passive methods.

Experimental results demonstrate localized snore attenuation and improved suppression at low frequencies, indicating the feasibility of targeted acoustic cancellation for sleep environments without requiring wearable devices.

(85 pages)

PUBLIC ABSTRACT

Adaptive Active Snore Noise Control Using Microphone and Loudspeaker Arrays

Nathan Walker

Snoring can wreck a good night's sleep, not just for the snorer but for their partner, and over time it can strain relationships. Common fixes like CPAP machines, nasal strips, and earplugs either have to be worn or don't do much against the deep, low-pitched rumble that makes snoring so hard to ignore.

This research explores a different approach, a device that doesn't need to be worn at all. It uses a small set of microphones to listen to the snoring and a set of speakers to play back carefully timed anti-snore noise, which is sound designed to cancel out the snore. The system focuses this cancellation on the area around the sleeping partner's ear, so the snoring is quieter where it matters while the rest of the room is left largely undisturbed.

In testing, the system reduced snoring noise at the target location, with the biggest improvements at low frequencies where traditional methods struggle. These results suggest that targeted, wearable-free noise cancellation could offer a practical way to help couples sleep better.

To my wife Grace, thank you for supporting me through this.

ACKNOWLEDGMENTS

Thank you God the Almighty for giving me an opportunity to learn and grow in your creation. Through you is my life given to me, let me honor you by following your will Lord. Thank you as well to all the people that have guided me on my journey. Thank you Dr. Gunther for pointing me to Dr. Moon's direction to make this thesis possible. Thank you Dr. Moon for all of your help and support.

Nathan C. Walker

CONTENTS

Page

ABSTRACT . . . iii

PUBLIC ABSTRACT . . . iv

ACKNOWLEDGMENTS . . . vi

LIST OF TABLES . . . ix

1 INTRODUCTION . . . 1

2 Literature Review . . . 3
- 2.1 Near-field Array Processing . . . 3
- 2.2 Active Noise Control Systems . . . 4
- 2.3 Multi-Channel and Spatial Active Noise Control . . . 4
- 2.4 Adaptive Filtering Algorithms for ANC . . . 5
- 2.5 Snore Cancellation . . . 6

3 Research and Design Methods . . . 8
- 3.1 Array Processing Using Sinc Interpolation . . . 8
- 3.2 Mechanical Layout . . . 10
- 3.3 Additional Electronics . . . 12
- 3.4 Adaptive filtering . . . 13
- 3.5 Experimental Setup . . . 19

4 Experimental Results . . . 22
- 4.1 Array Processing and Plots . . . 22
- 4.2 Runtime Performance . . . 27
- 4.3 Programmable Filter Array . . . 28
- 4.4 Simple ANC with Adaptive Filters . . . 32
- 4.5 Broadband ANC . . . 35
- 4.6 Cross-Correlation Analysis of System Delay . . . 39
- 4.7 Spatial ANC with LAGS . . . 42

5 Conclusions and Future Work . . . 46
- 5.1 Summary of Research . . . 46
- 5.2 Contributions . . . 47
- 5.3 Discussion . . . 48
  - 5.3.1 Real-Time Array Processing . . . 48
  - 5.3.2 Adjustable Experimental Platform . . . 49
  - 5.3.3 Programmable Filter Bank . . . 49
  - 5.3.4 Adaptive Filtering . . . 50
- 5.4 Future Work . . . 51

5.4.1 Hardware . . . 51
5.4.2 Adaptive Algorithms . . . 52
5.4.3 Full Array Evaluation . . . 52
5.5 Final Remarks . . . 53

REFERENCES . . . 54

APPENDICES . . . 56
A System Operation and Implementation . . . 57
A.1 Setting Up the ADC and DAC in a Linux Environment . . . 57
A.2 How to Use the Code . . . 58
A.3 Supporting Functions . . . 59

LIST OF TABLES

Table Page

4.1 Maximum and minimum RMS amplitude and their difference for the six focal-point locations. 25

4.2 Timing performance of the real-time signal processing system for different processing configurations. 28

4.3 Measured frequency-domain characteristics of the programmable filter array for the six programmed cutoff frequencies. 31

4.4 Mean-square error values calculated over selected two-second time windows. 44

CHAPTER 1

INTRODUCTION

Individual sensors in multi-sensor systems are often impacted by measurement noise and limited aperture, resulting in reduced spatial resolution. Array processing techniques address this limitation by coherently combining signals collected from multiple sensors to achieve improved effective gain and spatial selectivity. Coherent combination is enabled by exploiting propagation delays between sensors caused by differences in source-to-sensor distance.

Traditional array processing commonly relies on the far-field approximation. As an acoustic or electromagnetic signal propagates over sufficiently large distances, the wavefront can be approximated as planar across the array aperture. Under this assumption, conventional array processing methods estimate the direction of arrival (DOA) of the source but cannot directly determine source location. Consequently, multiple spatially separated arrays are typically required to perform localization.

Near-field array processing relaxes the planar wave assumption by exploiting the curvature of spherical wave fronts. Prior research has demonstrated that near-field processing enables simultaneous estimation of both direction and range, allowing a single array to localize a source [1]. This capability is valid when the source lies within the Fresnel region, which can be approximated by $r \approx \frac{2L^2}{\lambda}$, where L represents the largest dimension of the array and λ represents the operating wavelength [2] [3].

This research proposes the use of microphone and loudspeaker arrays to implement an active noise control (ANC) system capable of spatially locating and cancelling unwanted acoustic sources. One motivating application is the mitigation of sleep disturbances caused by snoring. Snoring affects approximately 25% of men and 15% of women and has been shown to negatively impact sleep quality and interpersonal relationships. Existing mitigation techniques such as CPAP therapy and surgical intervention focus on modifying the

physiology of the individual producing the snore [1]. In contrast, this research aims to reduce perceived acoustic disturbance without requiring behavioral or physiological modification of either individual.

Active noise control systems require adaptive filtering techniques to model acoustic transfer paths and minimize residual error between a reference signal and an error microphone measurement. Given a desired signal $d(n)$ and a reference signal containing delayed and noisy observations $x(n)$, adaptive filters iteratively estimate a transfer function that produces and output $y(n)$ that approximates the desired signal. The error signal is defined as $e(n) = d(n) - y(n)$; the adaptive filter coefficients are adjusted to minimize the error signal.

Several adaptive filtering algorithms have been extensively studied for ANC applications. The Least Mean Squares (LMS) algorithm offers low computational complexity but typically exhibits slower convergence. Recursive Least Squares (RLS) algorithms provide faster convergence and improved tracking performance at the expense of increased computational complexity. This research proposes investigating the Fast Array Recursive Least Squares (FARLS) algorithm as a potential compromise between computational efficiency and convergence performance for array-based ANC systems.

While active noise control and near-field array processing have both been extensively studied, limited work has investigated the integration of near-field localization techniques with computationally efficient adaptive filtering methods for real-time spatially selective acoustic cancellation. Existing ANC systems often assume fixed or poorly localized noise sources, limiting their effectiveness in dynamic environments [12].

The goal of this research is to develop and evaluate a framework that combines near-field array processing with FARLS-based adaptive filtering to enable real-time localization and suppression of acoustic interference in multi-sensor ANC systems.

CHAPTER 2

Literature Review

2.1 Near-field Array Processing

Near-field array processing exploits the physical characteristics of wave propagation. As a signal radiates from a source, it initially exhibits a spherical wavefront. At sufficiently large distances, the wavefront can be approximated as planar across the receiving array. The transition between these propagation regimes is commonly described using the Fraunhofer distance, $r \approx \frac{2L^2}{\lambda}$, where L is the maximum array aperture and λ is the signal's wavelength. The region between the reactive near-field and the Fraunhofer far-field is referred to as the Fresnel, or radiating near-field, region where wavefront curvature remains significant [2].

In far-field array processing, planar wavefronts produce approximately linear phase differences across sensors in the array. These phase differences depend primarily on the direction of arrival (DOA) of the incoming signal. As a result, far-field processing techniques typically estimate source direction, while range information is generally not directly observable. Localization of the source therefore often requires multiple spatially separated arrays or additional geometric constraints.

In contrast, near-field processing exploits spherical wavefront propagation, where phase differences between sensors depend on the absolute distance between each sensor and the source. The resulting steering vector becomes range-dependent, enabling simultaneous estimation of both source direction and range. Consequently, near-field processing allows a single array to perform source localization [3] [4].

Near-field array processing has traditionally been applied to higher-frequency signals, where shorter wavelengths reduce the aperture required to observe wavefront curvature. Many established applications therefore exist in ultrasonic imaging, underwater acoustics, and acoustic holography [2] [5]. This frequency dependence has limited investigation of near-

field localization techniques in lower-frequency acoustic environments, motivating further research in this area.

2.2 Active Noise Control Systems

Active noise control (ANC) systems operate by generating an anti-noise signal that destructively interferes with unwanted acoustic disturbances. A reference microphone measures a noisy signal $x(n)$, which is processed by an adaptive filter to produce a cancelling signal $y(n)$. When reproduced through a loudspeaker, the cancelling signal is designed to have equal magnitude and opposite phase relative to the primary noise at a target location, resulting in destructive interference.

ANC systems are typically characterized by primary and secondary acoustic paths. The primary path describes the propagation of noise from the source to the error microphone, producing the primary disturbance signal $d(n)$. The secondary path represents the propagation of the cancelling signal from the loudspeaker through the acoustic environment to the error microphone. Because the secondary path modifies the amplitude and phase of the cancelling signal, it must be accounted for during adaptive filter design to ensure stable and effective noise cancellation [6].

In spatial ANC systems, the residual error signal is measured at specific spatial locations using one or more error microphones [7]. These microphones provide feedback to the adaptive filter, allowing filter coefficients to be updated in order to minimize acoustic pressure at targeted listening regions. The spatial distribution of microphones and loudspeakers therefore plays a critical role in determining the effectiveness and coverage of the noise control region.

2.3 Multi-Channel and Spatial Active Noise Control

While single-channel active noise control (ANC) can reduce noise at a single point, real-world environments usually require suppression over a larger area. Multi-channel or spatial ANC systems tackle this by using multiple microphones and speakers to shape the sound field across a region or zone.

These systems coordinate multiple cancelling sources to generate interference patterns that cancel unwanted noise while preserving desired signals. Beamforming can be combined with adaptive filtering to improve spatial selectivity, allowing the system to focus cancellation on specific directions or locations.

Multi-channel ANC comes with added challenges. More sensors and speakers increase computational complexity, require modeling multiple secondary paths, and can introduce stability issues due to cross-channel interactions and changing acoustic conditions. Fully populated arrays are expensive and difficult to manage, which motivates more efficient approaches.

Incorporating spatial information, such as source localization or near-field array processing, offers a promising way to focus cancellation without needing full multi-channel systems. Even though this work focuses on a single cancelling source, the techniques for spatial localization and adaptive filtering are directly relevant to multi-channel ANC and represent a step toward more advanced spatial noise control [8] [9].

2.4 Adaptive Filtering Algorithms for ANC

In the world of active noise control, several adaptive filtering algorithms are commonly used to estimate the transfer function between reference signals and error microphones. Three widely employed algorithms are:

- Least-Mean Squares (LMS)
- Recursive Least Squares (RLS)
- Fast Array Recursive Least Squares (FARLS)

The LMS algorithm begins with an initial estimate of the filter coefficients and updates them iteratively as new data is received. Updates are computed using a gradient descent approach to minimize the error signal energy. This makes LMS computationally simple and robust, but it typically converges more slowly than other algorithms, especially for longer filters or highly correlated inputs.

RLS algorithms exploit the structure of the input data to achieve faster convergence. By incorporating all past input data into a rank-1 update of the filter coefficients, RLS effectively "remembers" the history of the signal and can converge in approximately the length of the filter. However, this comes at the cost of significantly higher computational complexity compared to LMS, especially for multi-channel systems [10].

FARLS is a variation of RLS designed to reduce computational demands while preserving fast convergence. It achieves this by transforming the standard RLS computations using Givens rotations along a hyperbolic curve to perform the rank-1 updates more efficiently. This approach reduces the number of operations required per iteration, making it well-suited for multi-sensor and spatial ANC applications where computational resources are limited.

FARLS is chosen for this work because it offers a combination of fast convergence and reduced computational complexity compared to standard RLS algorithms [11]. By efficiently updating filter coefficients using Givens rotations, FARLS can track time-varying signals more quickly while requiring fewer operations per iteration. This makes it particularly suitable for applications like spatial ANC, where multiple sensors and adaptive filters increase computational demands, and rapid adaptation is critical to effective noise cancellation.

2.5 Snore Cancellation

Snore cancellation has been studied for several years, with early approaches typically relying on one or two microphones to measure the snore signal. While these systems could effectively reduce noise at the microphone location, cancellation at the listener's ear was often limited due to poor spatial coverage [8] [9] [12] [13] [14] [15] [16] [17]. Traditional implementations often used two large speakers mounted on a test headboard to generate the cancelling signal. Although functional, these setups were constrained by limited localization of the snore source, reducing the effectiveness of noise suppression in realistic listening positions.

Most existing snore cancellation systems use the Filtered-X Least Mean Squares algorithm, commonly abbreviated as FXLMS. FXLMS attempts to account for both the primary

acoustic path from the snore source to the error microphone and the secondary path from the cancelling speaker to the microphone. This approach has been shown to work well in simulations and real-time experiments. However, the highly dynamic and non-stationary nature of snore signals motivates the exploration of alternative adaptive filtering methods.

In this work, the Fast Array Recursive Least Squares (FARLS) algorithm is proposed as an alternative. FARLS combines the fast convergence properties of the RLS algorithm with reduced computational complexity, making it well-suited for real-time applications [11]. While it introduces some numerical sensitivity, FARLS is expected to provide improved adaptation to time-varying snore signals and better performance in multi-sensor configurations compared to traditional FXLMS implementations.

CHAPTER 3

Research and Design Methods

This chapter describes the design and implementation of the active snore cancellation system. First, the fractional delay interpolation method used for microphone array alignment is presented. Next, the mechanical platform and supporting electronics are described. Finally, the adaptive filtering algorithms used for active noise cancellation are introduced, culminating in the Fast Array Recursive Least Squares (FARLS) algorithm employed throughout this work.

3.1 Array Processing Using Sinc Interpolation

To enable accurate time alignment across the microphone array, a fractional delay filtering approach based on sinc interpolation was selected. Sinc interpolation is well-suited for broadband signals, as it provides exact reconstruction for bandlimited signals according to the Shannon-Whittaker sampling theorem. Since snore signals exhibit broadband characteristics, this approach preserves signal fidelity during delay compensation.

A continuous-time delayed signal can be expressed using the Shannon-Whittaker interpolation formula as

$$s(t-\tau) = \sum_{l=-\infty}^{\infty} s(lT)\,\mathrm{sinc}\left(2B(t-\tau-lT)\right), \tag{3.1}$$

where T is the sampling period, B is the signal bandwidth, and τ is the desired delay.

In practice, the infinite summation is not realizable. Therefore, the interpolation is approximated by truncating the sinc function to a finite number of terms:

$$s(t-\tau) = \sum_{l=-P}^{P} s(lT)\,\mathrm{sinc}\left(2B(t-\tau-lT)\right), \tag{3.2}$$

where P determines the length of the interpolation filter.

Sampling the delayed signal at discrete time instances $t = nT$ yields

$$s(nT - \tau) = \sum_{l=-P}^{P} s(lT)\,\mathrm{sinc}\left(2B(nT - lT - \tau)\right). \tag{3.3}$$

This expression can be interpreted as a finite impulse response (FIR) filter that implements both integer and fractional delay. The integer (bulk) delay accounts for whole sample shifts, while the fractional delay is achieved through sinc-based interpolation.

A real-time implementation of this approach was developed in C. For each sensor in the array, the required delay was decomposed into an integer delay component and a fractional delay component. The integer delay was implemented using a buffer offset, while the fractional delay was realized using the truncated sinc interpolation filter.

To compute the required delay at each microphone, the propagation delay from the acoustic source to each sensor was first estimated. Let the maximum delay across all microphones be denoted as $\tau_{\max}$. The relative delay applied to the i-th microphone is then computed as

$$\tau_i = \tau_{\max} - \tau_{\mathrm{mic},i}, \tag{3.4}$$

where $\tau_{\mathrm{mic},i}$ is the propagation delay from the source to the i-th microphone. This relative delay ensures that signals from all microphones are temporally aligned, effectively compensating for differences in propagation time. As a result, the array achieves constructive combination of the received signals, producing the desired array gain.

The interpolation length P determines the trade-off between computational complexity and interpolation accuracy. In this work, a value of $P = 10$ was selected, resulting in a filter length of $2P + 1$. This value was chosen empirically, as it provided sufficient interpolation accuracy while maintaining computational efficiency for real-time implementation.

This approach enables precise time alignment across the microphone array, which is critical for accurate near-field localization and subsequent adaptive noise control processing.

3.2 Mechanical Layout

The microphone array developed during previous work provided only a rudimentary test platform with limited ability to adjust microphone spacing and sensor height. To improve experimental flexibility, a new adjustable test structure was designed and constructed.

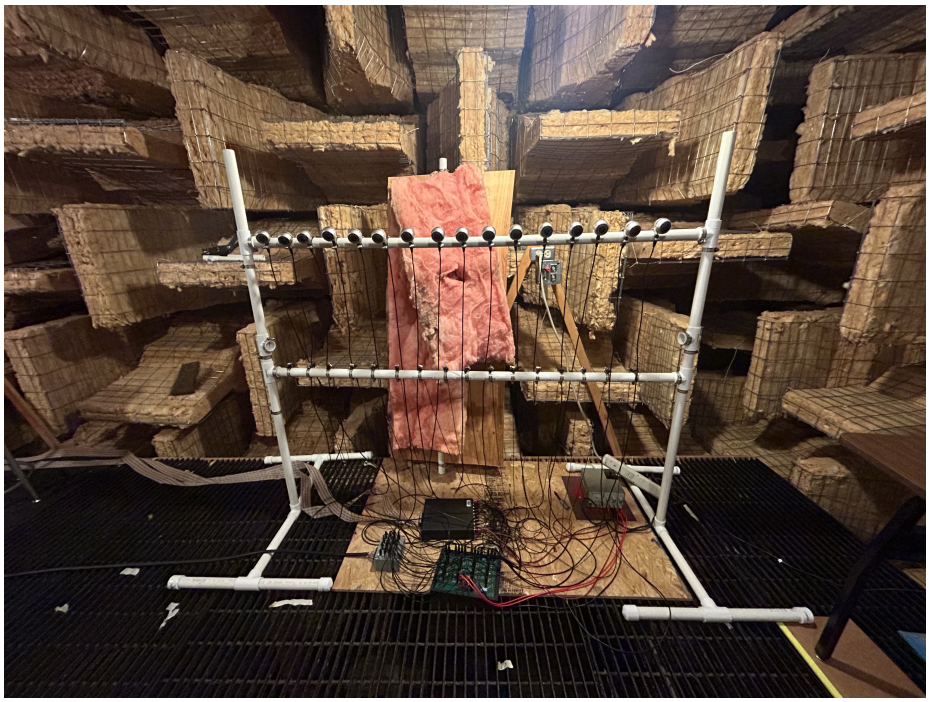


Fig. 3.1: Adjustable mechanical structure constructed for the microphone and speaker array, providing configurable sensor spacing and height.

The support structure was built using PVC pipe and standard PVC fittings to provide a lightweight, low-cost, and easily reconfigurable platform. The design allows the height of the microphone array, the spacing between individual microphones, and the orientation of the array to be adjusted. These adjustable parameters enable the array geometry to be modified for different experimental configurations and localization scenarios.

To facilitate complete system testing, a loudspeaker array was incorporated into the same support structure. The loudspeaker array was designed independently of the microphone array, allowing its height, orientation, and position to be adjusted without affecting the microphone geometry. This arrangement provides a repeatable experimental setup for

evaluating signal acquisition, beamforming, and adaptive noise control algorithms.

To reduce acoustic crosstalk between the loudspeaker array and the microphone array during testing, a removable sound baffle was constructed. The baffle consisted of a PVC frame with a plywood panel covered in acoustic insulation. The design allowed the baffle to be repositioned and adjusted in both height and orientation, providing flexibility for different experimental configurations while reducing direct acoustic coupling between the arrays.



Fig. 3.2: Custom interface enclosure used to provide connections between the microphone array and the linux machine.

A custom interface enclosure was also designed to connect the loudspeaker array to the digital-to-analog converter (DAC) installed in the host computer. The enclosure accepts the ribbon cable output from the DAC and routes each channel to standard 3.5 mm audio connectors. The enclosure was fabricated from a plastic box using a manual milling machine to create the required openings for the connectors. Internal wiring was completed by soldering each ribbon cable conductor to its corresponding 3.5 mm audio jack.

Initial testing indicated that the DAC output contained significant high-frequency components associated with the stepped output waveform, producing audible distortion in the

loudspeakers. To reduce these artifacts, an analog low-pass filter bank was designed and incorporated between the DAC outputs and the speakers. Unlike the interface enclosure, the filter enclosure was manufactured using additive manufacturing (3D printing). The enclosure was designed in Onshape using the PCB model, allowing a custom enclosure to be produced that accurately supported the assembled circuit board while simplifying fabrication.

The completed mechanical and electrical test platform provides a configurable environment for evaluating microphone array processing, loudspeaker placement, and future adaptive noise control experiments.

3.3 Additional Electronics

As described in the previous section, a custom interface enclosure was constructed to connect the ribbon cable output of the digital-to-analog converter (DAC) to standard 3.5 mm audio connectors. This required soldering each conductor of the ribbon cable connector to its corresponding 3.5 mm audio jack, providing a reliable electrical interface between the DAC and the loudspeaker array.

The DAC reproduces digital signals using a zero-order hold, resulting in a stepped output waveform. Although the desired audio signal lies within the audible frequency range, the stepped waveform also contains unwanted high-frequency spectral components. Because the loudspeakers are capable of reproducing frequencies across much of the audible spectrum, these high-frequency components introduced audible distortion during testing.

To reduce these artifacts, a programmable analog low-pass filter bank was designed and constructed. Since multiple sampling frequencies were anticipated during experimentation, a fixed cutoff frequency was not suitable. Instead, a programmable filter architecture was selected, allowing the cutoff frequency to be adjusted as needed for different sampling rates while maintaining adequate suppression of the high-frequency DAC images.

The filter bank was implemented using the MAX270CPP+ programmable analog filter integrated circuit. A custom printed circuit board (PCB) was designed and manufactured to interface with the device. The PCB accepts standard 3.5 mm audio inputs from the DAC

interface enclosure and provides filtered 3.5 mm outputs for connection to the loudspeaker array. The programmable filter integrated circuits were configured through software, allowing the filter characteristics to be modified without redesigning the hardware.

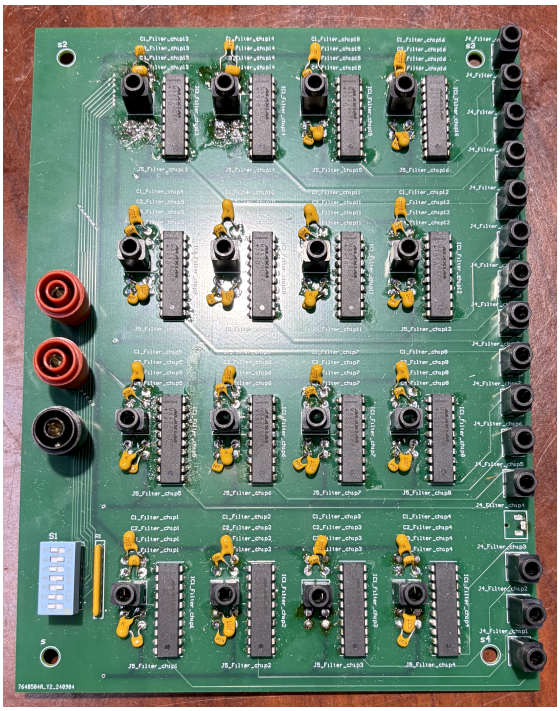


Fig. 3.3: Custom DAC and filter-bank printed circuit board used for analog filtering.

Each programmable filter integrated circuit contains two second-order low-pass filter sections. To achieve a steeper roll-off and improved attenuation of unwanted high-frequency components, two second-order sections were cascaded to form a fourth-order low-pass filter. Experimental testing demonstrated that the cascaded configuration provided substantially better suppression of the DAC-induced high-frequency artifacts than a single second-order filter.

3.4 Adaptive filtering

Active Noise Control (ANC) seeks to reduce unwanted sound by generating an acoustic signal that destructively interferes with the original noise. Since sound propagates as

a pressure wave, a second wave of equal amplitude and opposite phase can be produced to reduce the sound pressure level at a desired location. Unlike passive noise control techniques, which rely on absorbing or blocking sound, ANC actively estimates the appropriate cancellation signal based on the measured acoustic environment.

In a feedforward ANC system, a reference microphone measures the incoming noise before it reaches the location where cancellation is desired. As the sound propagates through the environment, reflections, diffraction, and other acoustic effects alter the waveform before it reaches the error microphone. Because these environmental effects are generally unknown and may change over time, an adaptive filter is used to continuously estimate the acoustic path and generate the appropriate cancellation signal.

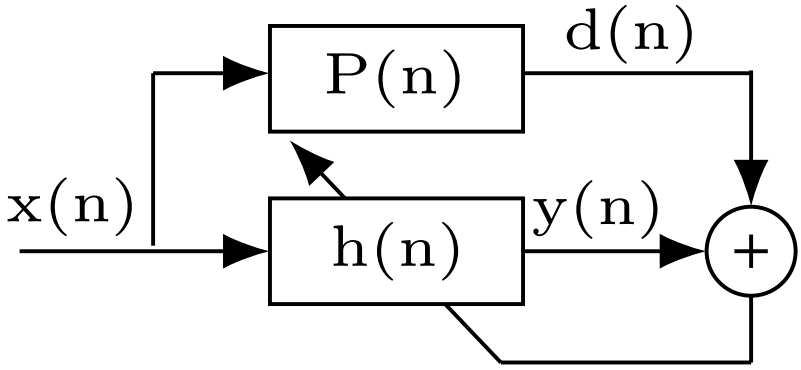


Fig. 3.4: Block diagram of the adaptive filtering process used for active noise cancellation.

Figure 3.4 illustrates the standard adaptive filtering model. The reference signal, $x[n]$, represents the measured disturbance before it propagates through the acoustic environment. The desired signal, $d[n]$, represents the disturbance measured at the error microphone after propagation through the unknown acoustic path. The adaptive filter, represented by the coefficient vector $\mathbf{h}$, operates on the reference signal to produce the cancellation signal.

For an adaptive filter with M coefficients, the reference signal is represented as an input vector containing the current and previous reference samples,

$$\mathbf{x} = \begin{bmatrix} x[n] & x[n-1] & \cdots & x[n-M+1] \end{bmatrix}^T, \tag{3.5}$$

and the adaptive filter coefficients are represented by the coefficient vector

$$\mathbf{h} = \begin{bmatrix} h_0[n] & h_1[n] & \cdots & h_{M-1}[n] \end{bmatrix}^T. \tag{3.6}$$

The filter output is calculated as the inner product of the input and coefficient vectors,

$$y[n] = \mathbf{x}^T\mathbf{h}. \tag{3.7}$$

The objective of the adaptive filter is to continually adjust the filter coefficients so that the output $y[n]$ approaches the desired signal $d[n]$. The resulting error is therefore defined as

$$e[n] = d[n] - y[n]. \tag{3.8}$$

The adaptive algorithm attempts to minimize the mean squared error between the desired signal and the filter output. The corresponding cost function is

$$J[n] = E\left[e^2[n]\right] = E\left[\left(d[n] - \mathbf{x}^T\mathbf{h}\right)^2\right]. \tag{3.9}$$

Expanding the squared term gives

$$J[n] = E\left[d^2[n] - 2d[n]\mathbf{x}^T\mathbf{h} + \left(\mathbf{x}^T\mathbf{h}\right)^2\right]. \tag{3.10}$$

To determine the filter coefficients that minimize the cost function, the gradient of $J[n]$ with respect to the filter coefficient vector is computed as

$$\nabla_{\mathbf{h}} J = -2E\left[\mathbf{x}d[n]\right] + 2E\left[\mathbf{x}\mathbf{x}^T\right]\mathbf{h}. \tag{3.11}$$

The optimal solution requires knowledge of the statistical expectations of the input signal, which are generally unavailable during real-time operation. The Least Mean Squares (LMS) algorithm addresses this problem by replacing the expected gradient with its instantaneous estimate,

$$\nabla_{\mathbf{h}} J \approx -2e[n]\mathbf{x}. \tag{3.12}$$

Although this approximation does not produce the exact gradient at every iteration, the average behavior of the algorithm follows the direction of steepest descent. This allows the filter coefficients to converge toward the minimum of the mean squared error surface through an iterative steepest-descent process.

Using this instantaneous gradient, the filter coefficients are updated according to

$$\mathbf{h}[n+1] = \mathbf{h}[n] + \mu e[n]\mathbf{x}[n], \tag{3.13}$$

where μ is the adaptation step size. The value of μ determines the convergence characteristics of the algorithm. Larger values of μ increase the adaptation rate but may lead to instability or increased steady-state error, while smaller values improve stability at the expense of slower convergence. Consequently, the step size must be selected to balance convergence speed and stability for the desired application.

Initially, the LMS algorithm was used during testing due to its computational efficiency and simplicity while the real-time computational constraints of the system were still being determined. The reduced computational complexity of LMS allowed early testing of acoustic noise cancellation performance and evaluation of the overall system architecture.

Although LMS is computationally efficient, its convergence rate depends on the eigenvalue spread of the input correlation matrix. In highly correlated signals, such as speech and snoring, LMS may require many iterations before converging. Since active noise control systems must adapt quickly to changing acoustic environments, a faster adaptive algorithm is desirable.

Unlike LMS, Recursive Least Squares (RLS) minimizes an exponentially weighted least-squares cost function rather than relying on a stochastic gradient approximation. Consequently, RLS typically converges significantly faster than LMS but requires substantially more computation because a matrix inverse, or an equivalent recursive update, must be maintained.

The Fast Array Recursive Least Squares (FARLS) algorithm exploits the recursive structure of the RLS solution to reduce the computational complexity associated with conventional matrix operations. Rather than explicitly computing the full inverse correlation matrix at each iteration, FARLS recursively updates intermediate quantities, resulting in convergence characteristics similar to RLS while reducing the computational cost.

In the FARLS formulation used in this work, the input vector is denoted by $\mathbf{u}[n]$ and the adaptive filter coefficient vector is denoted by $\mathbf{w}[n]$. The input vector contains the current and previous samples used by the adaptive filter and can be represented as

$$\mathbf{u} = \begin{bmatrix} x[n] & x[n-1] & \cdots & x[n-M+1] \end{bmatrix}^T. \tag{3.14}$$

The FARLS filter output is therefore calculated from the input and coefficient vectors as

$$y[n] = \mathbf{w}^T \mathbf{u}. \tag{3.15}$$

The FARLS algorithm recursively updates the coefficient vector as new samples are received. The general procedure used in this work is summarized in Algorithm 1.

Algorithm 1 FARLS Adaptive Filtering Procedure

Require: Input sample $x(n)$, desired signal $d(n)$

Require: Filter length M, forgetting factor λ

Initialize filter coefficients $\mathbf{w}[0]$ and FARLS recursive array A

for $n = 1, 2, \ldots, N$ **do**

Update the input vector $\mathbf{u}$ and recursive quantities in A

Calculate and apply the circular rotation parameters

Calculate and apply the hyperbolic rotation parameters

Update the recursive quantities using λ

Calculate the filter output:

$$y[n] = \mathbf{w}^T \mathbf{u}$$

Calculate the prediction error:

$$e[n] = d[n] - y[n]$$

Update the filter coefficients:

$$w_j[n] = w_j[n-1] + g_j[n]e[n], \qquad j = 1, \ldots, M$$

end for

return $\mathbf{w}[n]$

The FARLS procedure begins by initializing the filter coefficients and the recursive array used to store the intermediate quantities required by the algorithm. At each iteration, a new input sample is incorporated into the input vector, which contains the current and previous samples used by the adaptive filter. The recursive quantities in the array are then updated using the new input information and the state retained from the previous iteration.

The recursive update is performed using circular and hyperbolic rotations. These rotations transform the intermediate quantities in the FARLS array while avoiding the

explicit formation and inversion of the input correlation matrix required by conventional RLS. The forgetting factor is also incorporated during these recursive updates, allowing older samples to have progressively less influence on the least-squares solution. This enables the algorithm to continuously adapt as new samples are received.

After the recursive quantities have been updated, the current filter output is calculated from the input vector and filter coefficients, and the prediction error is determined as the difference between the desired signal and the filter output. The prediction error and the calculated gain are then used to update the filter coefficients. This coefficient update allows the filter to respond to the most recent error while retaining information from previous iterations through the recursive state variables. This process is repeated for each incoming sample, allowing the FARLS filter to continuously adapt to changes in the input signal.

3.5 Experimental Setup

The experimental platform consisted of a workstation running the Ubuntu Linux operating system. All signal processing and adaptive filtering algorithms were implemented in the C programming language and compiled using the GNU Compiler Collection (GCC). The workstation was equipped with an Intel processor, an NVIDIA graphics processing unit (GPU), and sufficient system memory to satisfy the real-time computational requirements of the active noise control algorithms.

Audio acquisition and playback were performed using the PCIe66-16AI32SSC analog-to-digital converter (ADC) and the 66-16AO16 digital-to-analog converter (DAC), respectively. The system operated at a sampling frequency of 8 kHz with a 16-bit sample resolution. The adaptive filtering algorithms processed the acquired audio in real time, continuously generating the cancellation signals transmitted to the loudspeaker array.



Fig. 3.5: Linux-based workstation used for real-time signal processing and adaptive noise cancellation.

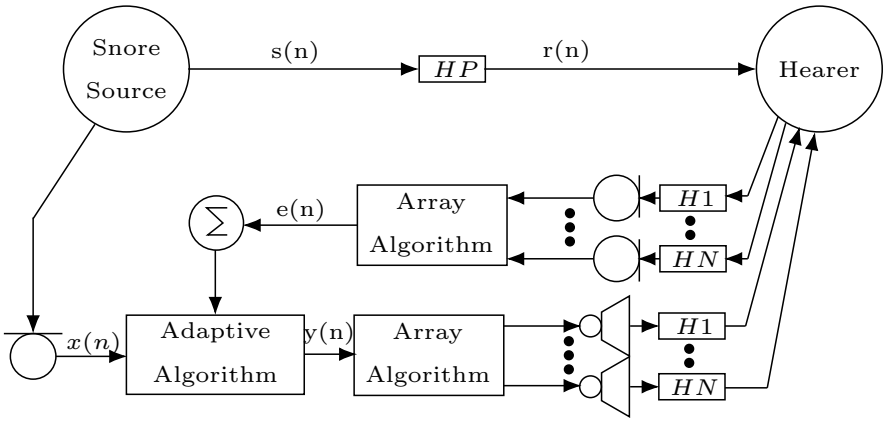


Fig. 3.6: Block diagram illustrating the signal flow through the active noise cancellation system.

Figure 3.6 illustrates the overall signal flow of the experimental platform. Audio signals acquired by the microphone array were digitized by the ADC and transferred to the Linux workstation for real-time processing. After the adaptive filtering algorithm computed the

cancellation signal, the processed audio was transmitted through the DAC, passed through the analog low-pass filter bank described in the previous section, and reproduced by the loudspeaker array.

The ADC and DAC cards interfaced with the experimental hardware through ribbon cable connections. Each ribbon cable was connected to a custom interface enclosure that converted the ribbon cable outputs into standard 3.5 mm audio connectors. The microphone array was connected to the ADC interface enclosure, while the loudspeaker array was connected to the DAC interface enclosure through the programmable analog low-pass filter bank.

The experimental hardware consisted of a 16-element microphone array and a 16-element loudspeaker array mounted on the adjustable support structure described in Section 3.2. The microphone spacing was adjustable, with experiments performed using a spacing of approximately 0.08575 meters. The speaker array was similarly adjustable, allowing different acoustic geometries to be evaluated while maintaining repeatable positioning throughout the experiments.

All experiments were conducted inside an anechoic chamber to minimize reflections and external acoustic interference. This environment allowed the performance of the microphone array, beamforming algorithms, and adaptive noise control system to be evaluated under controlled acoustic conditions.

To evaluate system performance, known audio signals were used as the disturbance source. Test signals were generated using an external loudspeaker positioned at predetermined locations relative to the microphone array. These controlled acoustic sources provided repeatable input signals for evaluating the fractional delay interpolation, beamforming, and adaptive filtering algorithms. Experimental results obtained using this setup are presented and discussed in Chapter 4.

CHAPTER 4

Experimental Results

This chapter presents the experimental results obtained from the implemented active noise cancellation system. The experiments evaluate the spatial response of the microphone array, the computational performance of the real-time processing system, the operation of the programmable filter array, and the performance of adaptive ANC using LMS and FARLS algorithms. The results of the adaptive ANC experiments are then used to investigate the effects of system delay through cross-correlation analysis. Finally, a controlled-delay experiment is used to demonstrate spatial acoustic cancellation using a known signal delay and polarity.

4.1 Array Processing and Plots

The purpose of this experiment was to determine whether the microphone-array processing produced the expected spatial response when an acoustic source was moved relative to a specified focal point. For each test, a focal point was selected in front of the microphone array and held fixed while an acoustic source was physically moved toward and away from that location. If the array-processing algorithm correctly focuses on the selected location, the processed signal should have a larger RMS amplitude when the source is near the focal point and a smaller RMS amplitude as the source moves away from it. The variation in RMS amplitude therefore provides an experimental indication of the spatial selectivity of the microphone array.

For each focal-point location, the array-processing algorithm was operated continuously for 15 seconds while the processed array output was recorded. The acoustic source was initially positioned near the selected focal point and was then moved around the focal region during the recording. The source position was not measured or recorded as a function of time. Therefore, the exact source position corresponding to each point in the recorded signal

cannot be determined. Instead, the experiment was used to observe whether movement of the acoustic source produced the expected changes in the RMS amplitude of the processed array output.

The experiment was performed using six focal-point locations. The horizontal coordinate of the focal point was varied between -0.5, 0, and 0.5 m, while the distance from the microphone array was set to either 1.0 or 1.5 m. Each focal-point location was tested using both a sinusoidal source and a broadband white-noise source. The two source types were used to determine whether the observed spatial response was present for both narrowband and broadband acoustic signals.

The recorded processed-array signals were analyzed by calculating the RMS amplitude in decibels as a function of time. The RMS amplitude represents the overall signal level of the processed microphone-array output over each analysis interval. Under the expected spatial response, the RMS amplitude should increase when the acoustic source moves toward the selected focal point and decrease when the source moves away from it. Because the source position was not recorded during the experiment, the plots cannot be used to determine the exact spatial coordinates of the source at each point in time. Instead, they show how the processed array output changed as the source was moved around the focal region.

The RMS amplitude response for the sinusoidal source at the six focal-point locations is shown in Figure 4.1. The corresponding response for the broadband white-noise source is shown in Figure 4.2.

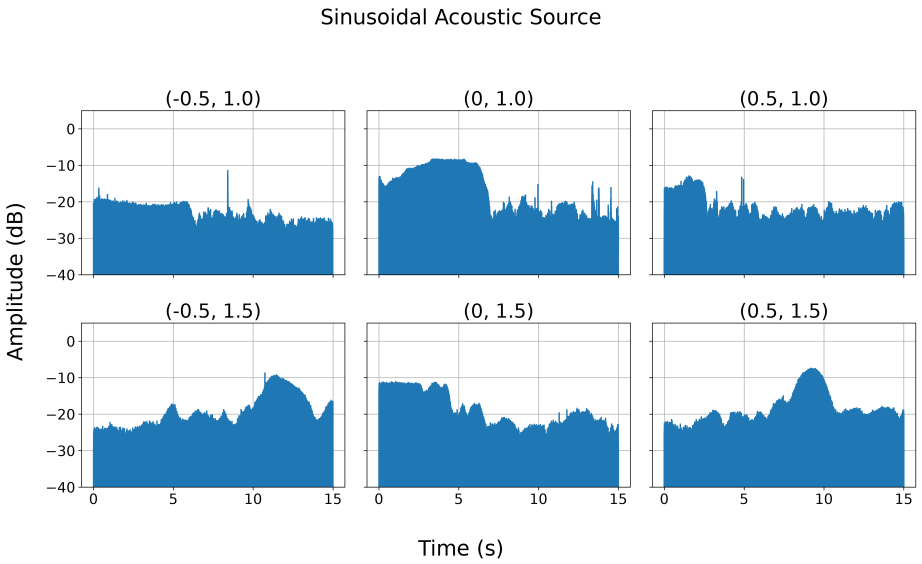


Fig. 4.1: Processed signal level of the microphone array for a sinusoidal acoustic source at the six tested focal-point locations.

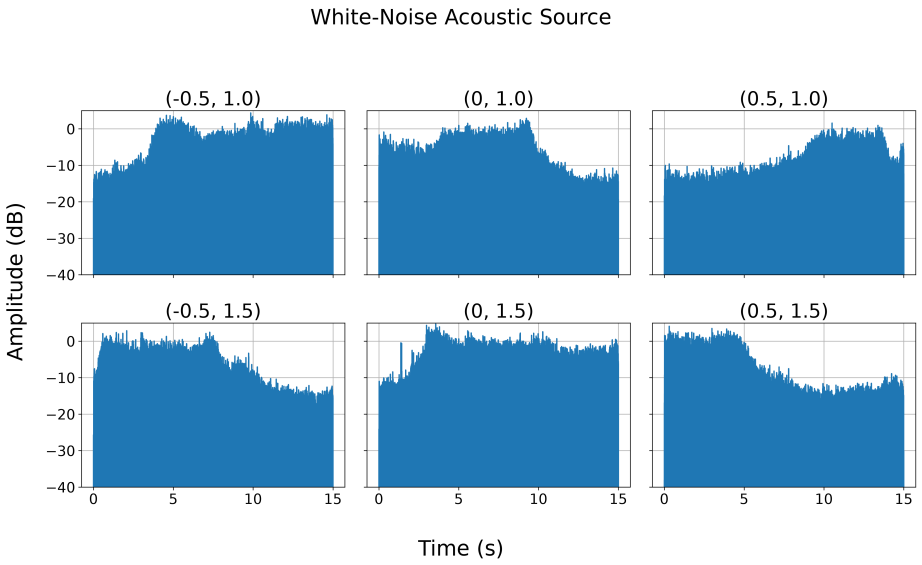


Fig. 4.2: Processed signal level of the microphone array for a white-noise acoustic source at the six tested focal-point locations.

As shown in the figures, the processed RMS amplitude varies over time as the acoustic source is moved around the focal region. In general, the RMS amplitude increases when

the source approaches the selected focal region and decreases as the source moves away. This behavior agrees with the expected response of the array-processing algorithm. When the source is located near the selected focal point, the propagation delays applied to the microphone signals allow the received signals to combine more constructively, producing a larger processed output. When the source is farther from the focal point, the signals combine less constructively, resulting in a lower processed RMS amplitude.

To provide a numerical comparison between the different focal-point locations, the maximum and minimum measured RMS amplitudes were extracted from each recording. The difference between these values was also calculated. This maximum-to-minimum difference provides a measure of how much the processed RMS amplitude changed during the source movement. A larger difference indicates a more pronounced change in the array response, while a smaller difference indicates a less pronounced change.

Table 4.1: Maximum and minimum RMS amplitude and their difference for the six focal-point locations.

Source	x (m)	y (m)	Max RMS (dB)	Min RMS (dB)	Max-Min (dB)
Sinusoidal	-0.5	1.0	-24.912	-32.379	7.467
Sinusoidal	0.0	1.0	-12.230	-32.084	19.855
Sinusoidal	0.5	1.0	-17.646	-31.759	14.112
Sinusoidal	-0.5	1.5	-13.420	-31.947	18.527
Sinusoidal	0.0	1.5	-12.690	-32.088	16.398
Sinusoidal	0.5	1.5	-11.326	-31.314	19.988
White noise	-0.5	1.0	-6.007	-21.646	15.639
White noise	0.0	1.0	-6.369	-22.831	16.462
White noise	0.5	1.0	-9.214	-22.686	13.472
White noise	-0.5	1.5	-7.795	-23.923	16.127
White noise	0.0	1.5	-5.661	-21.036	15.375
White noise	0.5	1.5	-6.367	-23.090	16.723

For the sinusoidal source, the difference between the maximum and minimum RMS amplitudes ranged from 7.467 dB to 19.988 dB. Five of the six focal-point locations produced changes greater than 14 dB. The smallest change occurred for the focal point at $(-0.5, 1.0)$ m, where the RMS amplitude varied by 7.467 dB. This smaller variation indicates that the spatial response was less pronounced during this particular measurement. The remaining focal-point locations exhibited substantially larger changes in RMS amplitude as the source was moved.

For the broadband white-noise source, the maximum-to-minimum RMS amplitude differences ranged from 13.472 dB to 16.723 dB. All six focal-point locations therefore exhibited a change greater than 13 dB. The relatively consistent variation across the six locations indicates that the array produced a measurable spatial response for broadband signals as well as for the sinusoidal source. This result is particularly relevant to active snore cancellation because snoring contains broadband acoustic energy rather than consisting of a single narrowband tone.

The sinusoidal measurement at the $(-0.5, 1.0)$ m focal point exhibited a noticeably smaller RMS amplitude variation than the other sinusoidal measurements. This may have resulted from differences in the source positioning or movement during that particular trial. However, because the source position was not measured throughout the recording, the cause of this difference cannot be determined quantitatively.

The similar response observed for both sinusoidal and broadband sources provides evidence that the spatial selectivity of the array-processing algorithm is not limited to a single-frequency signal. Additionally, measurable RMS amplitude changes were observed at both 1.0 m and 1.5 m from the array, indicating that the array-processing method produced a spatially varying response over both tested distances.

It is important to note that this experiment does not provide a complete two-dimensional characterization of the array response. Because the source position was not recorded simultaneously with the acoustic measurement, an exact relationship between source position and RMS amplitude cannot be established for each point in the recording. Instead, the

experiment demonstrates that moving an acoustic source around a selected focal region produces a measurable change in the RMS amplitude of the processed microphone-array output, consistent with the expected spatial selectivity of the array-processing algorithm.

4.2 Runtime Performance

This experiment aims to quantitatively evaluate whether the system is capable of processing data at the rate required for real-time active noise cancellation (ANC). The experiment evaluates the computational performance of the system as processing complexity is progressively increased, from basic data acquisition to the complete array processing and adaptive filtering pipeline.

This evaluation is important because the processing time must remain below the sampling period for the system to operate in real time. If the processing time exceeds the available time between samples, the algorithm will fall behind the incoming data and will be unable to perform ANC correctly.

To conduct this experiment, the Linux-based host machine containing the ADC and DAC cards was used. Because the purpose of this experiment was to evaluate computational performance rather than acoustic performance, the microphone and speaker arrays were not physically connected to the system. However, the software executed the same processing operations as if the full 16-microphone and 16-speaker arrays were connected. The sampling frequency was set to 8 kHz, which was the predetermined sampling frequency for the system.

For each configuration, 80,000 samples were processed while the computational complexity of the algorithm was progressively increased. The tested configurations included sample acquisition alone, processing of a single array, simultaneous microphone and speaker array processing, and finally the addition of the LMS and FARLS adaptive algorithms. The microphone and speaker arrays were configured with 16 channels each. The sinc interpolation filters used for array processing were 61 taps in length, and the adaptive filters contained 400 coefficients. For the LMS algorithm, the adaptation step size was set to $\mu = 0.001$.

Table 4.2: Timing performance of the real-time signal processing system for different processing configurations.

Processing Configuration	Median (μs)	95th Percentile (μs)
Sample acquisition only	57.208	58.941
Single array processing	61.416	67.417
Microphone + speaker array processing	65.374	72.907
Both arrays + LMS	68.879	78.879
Both arrays + FARLS	81.173	95.600

Table 4.2 shows the median and 95th-percentile processing times for each configuration. At a sampling frequency of 8 kHz, the available processing time per sample is 125 μs. The most computationally intensive configuration, which included both array processing and the FARLS adaptive algorithm, had a median processing time of 81.173 μs and a 95th-percentile processing time of 95.600 μs. Both values are below the 125 μs sampling period.

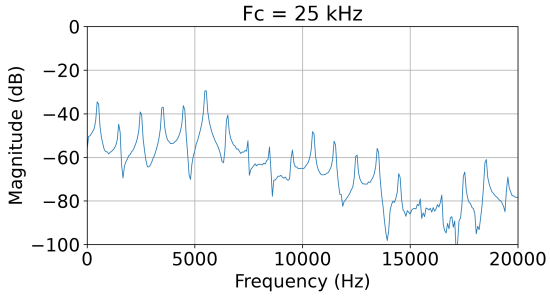
These results indicate that the processing pipeline generally completes within the available time interval for the tested configurations. The increase in processing time as additional processing stages are enabled demonstrates the computational cost associated with each stage while also showing that the complete tested processing pipeline remains below the sampling-period threshold under typical operating conditions.

4.3 Programmable Filter Array

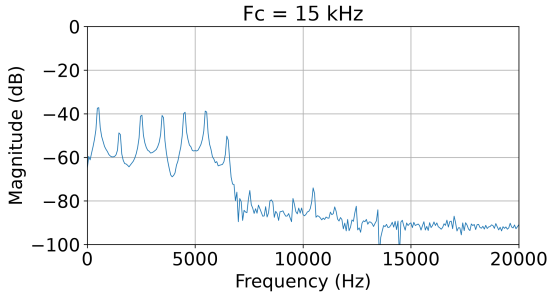
The purpose of this experiment is to demonstrate that the programmable filter array can attenuate high-frequency components of the DAC output while maintaining the modularity afforded by the programmable IC. This capability will allow future work to operate the ANC system with reduced interference from frequencies outside the designated operating spectrum, thereby reducing the burden placed on the adaptive algorithms.

The DAC reconstructs signals using a zero-order hold, which can introduce high-frequency spectral images alongside the desired audio content. The programmable filter array, implemented using the MAX270CPP+ IC and configured as a cascaded fourth-order low-pass filter (as described in Section 3.3), was evaluated to determine its effect on the frequency content of the DAC output. By adjusting the programmed cutoff frequency, the frequency response of the filter can be changed to attenuate frequency components above the desired operating bandwidth.

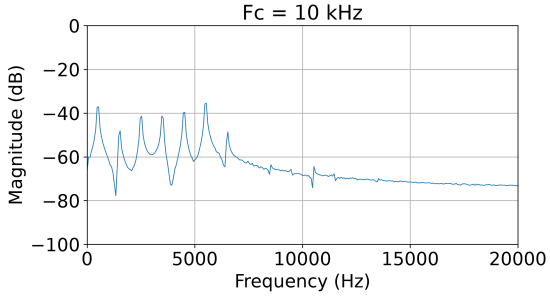
The experiment was conducted using six programmed cutoff frequencies: 25 kHz, 15 kHz, 10 kHz, 5 kHz, 3 kHz, and 2 kHz. A single tone noise was used for the experiment with the frequency of 500 Hz. The 25 kHz setting represents the highest cutoff frequency available from the programmable filter, while the remaining cutoff frequencies were selected as progressively lower values to evaluate the effect of reducing the filter bandwidth. For each cutoff frequency, the DAC output was recorded and analyzed in both the time and frequency domains. The resulting waveforms and FFTs were used to observe how the programmed cutoff frequency affected the output signal and the attenuation of higher-frequency components.



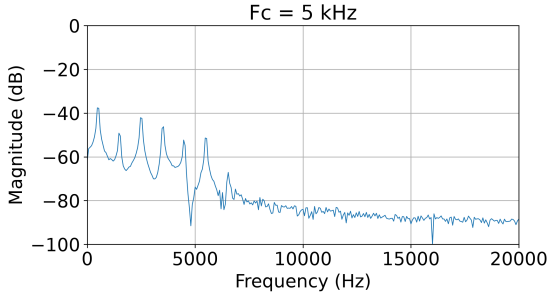
(a) $F_c = 25$ kHz



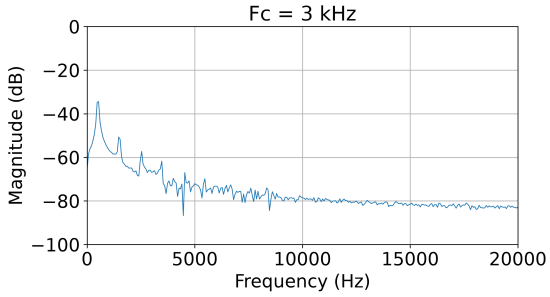
(b) $F_c = 15$ kHz



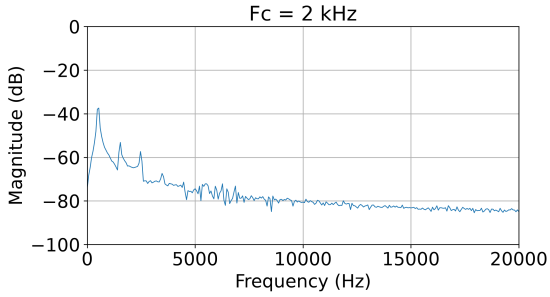
(c) $F_c = 10$ kHz



(d) $F_c = 5$ kHz



(e) $F_c = 3$ kHz



(f) $F_c = 2$ kHz

Fig. 4.3: Frequency-domain response of the programmable filter array for six programmed cutoff frequencies. Each plot shows the measured FFT magnitude in decibels of the 500 Hz sinusoidal input after filtering.

Table 4.3: Measured frequency-domain characteristics of the programmable filter array for the six programmed cutoff frequencies.

F_c (kHz)	500 Hz Magnitude (dB)	Largest Component Above 500 Hz (Hz)	Magnitude (dB)
25	-34.561	5533.33	-29.504
15	-37.472	5466.67	-38.785
10	-37.301	5533.33	-35.415
5	-37.611	2466.67	-42.107
3	-34.742	666.67	-47.770
2	-37.932	666.67	-50.432

The results show that changing the programmed cutoff frequency has a noticeable effect on the frequency content of the DAC output. The measured magnitudes of the 500 Hz input and the largest frequency component above 500 Hz for each cutoff frequency are summarized in Table 4.3. At the higher cutoff frequencies, such as 25 kHz and 15 kHz, more of the higher-frequency content remains visible in the FFT. For example, the largest component above 500 Hz has a magnitude of -29.504 dB at a cutoff frequency of 25 kHz, compared with -50.432 dB at a cutoff frequency of 2 kHz. The frequency of the largest component above 500 Hz also shifts from approximately 5.5 kHz at the higher cutoff frequencies to approximately 667 Hz at the 3 kHz and 2 kHz settings. These results demonstrate the increasing attenuation of higher-frequency components as the programmed cutoff frequency is decreased. The time-domain plots also show that the main signal remains present as the cutoff frequency is changed, although the lower cutoff frequencies produce a greater change in the output waveform. Overall, the results show the expected behavior of a low-pass filter as the programmed cutoff frequency is decreased.

The results from the six cutoff frequencies also demonstrate the programmability of the filter array. The cutoff frequency can be changed through the configuration of the

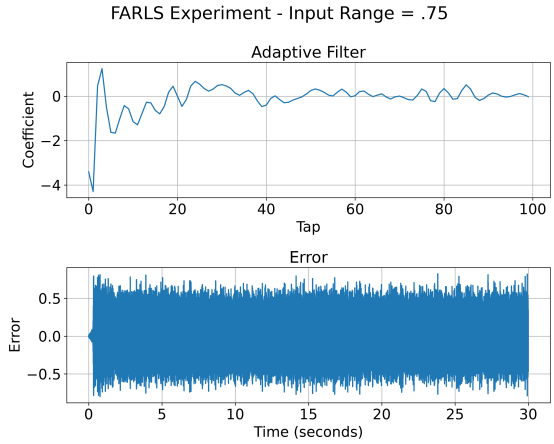
MAX270CPP+ without requiring a different hardware filter for each frequency range. This provides flexibility for the ANC system because the filter can be configured for the desired operating bandwidth depending on the signal being processed. For example, a higher cutoff frequency can be used when more of the signal bandwidth is required, while a lower cutoff frequency can be used when greater attenuation of higher frequency components is desired. This allows the same filter hardware to be used for different operating conditions and provides a practical way of controlling the frequency content presented to the ANC system.

4.4 Simple ANC with Adaptive Filters

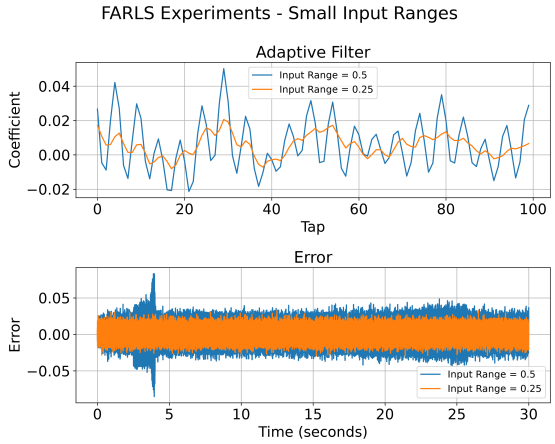
The purpose of this experiment was to evaluate the ability of the hardware to perform adaptive noise cancellation using a simplified setup. This experiment provides a starting point for evaluating the ANC system before incorporating the full microphone array and spatial processing. By beginning with a simplified configuration, the operation of the adaptive algorithms and the acoustic paths could be evaluated independently of the additional complexity introduced by spatial array processing.

This experiment was conducted using two microphones and two speakers. One microphone acted as the input to the adaptive algorithm, while the second microphone measured the error signal used to update the adaptive filter. One speaker acted as the source speaker and produced the noise signal, while the second speaker produced the cancelling signal generated by the adaptive filter. The input microphone was placed close to the source speaker to minimize the acoustic path between the source and the reference microphone. The error microphone was positioned approximately 0.75 meters from the source speaker, while the cancelling speaker was positioned approximately 0.6 meters from the error microphone.

For this experiment, a sinusoidal waveform was played through the source speaker, resulting in a single-tone input signal. Because the acoustic path was primarily composed of propagation through the anechoic chamber, the resulting adaptive filter was expected to approximate the response required to account for the propagation delay and phase of the acoustic path.



(a) FARLS experiment with an input range of 0.75.



(b) FARLS experiments with input ranges of 0.5 and 0.25.

Fig. 4.4: FARLS adaptive noise cancellation results for the tested input ranges.

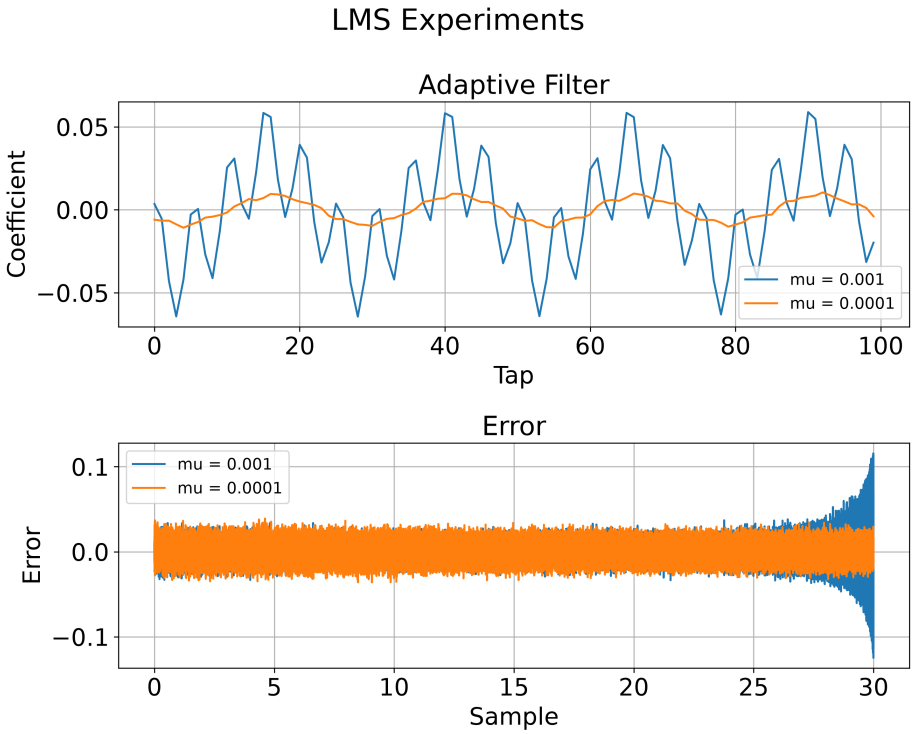


Fig. 4.5: LMS adaptive noise cancellation results for the tested step sizes.

Overall, the results show that the error signal does not converge despite testing a variety of μ values and input ranges. In the LMS experiments, even when the experiment was run for 40 seconds, the system eventually became unstable. Other combinations of algorithm parameters were also tested, but the results shown in Figures 4.4 and 4.5 represent the general behavior observed throughout the experiments. In addition to the error signal failing to converge, the resulting filter coefficients did not correspond to the expected response of the acoustic path through the anechoic chamber. For a single-tone input, the filter response would be expected to represent the required propagation delay and phase relationship between the reference and error signals. The resulting filters did not exhibit the expected behavior.

The FARLS results demonstrate similar difficulties with convergence. As shown in Figure 4.4a, the larger input range causes the amplitude of the adaptive filter output to increase substantially without reaching a stable solution. Reducing the input range did not result in convergence, although the lower input ranges changed the behavior of the algorithm. Figure 4.4b shows an example of the algorithm responding to a change in the error signal at approximately four seconds. The filter coefficients change rapidly in an attempt to reduce the error, but the resulting change does not produce a stable solution. The algorithm then adjusts the coefficients again and temporarily reduces the error. This behavior demonstrates that FARLS is responding to changes in the measured error, but it is not able to consistently identify the required acoustic response and converge to a stable solution.

The LMS results also show a lack of stable convergence. At lower μ values, the filter coefficients change slowly and the algorithm is unable to reduce the error sufficiently within the duration of the experiment. Increasing μ causes the algorithm to adapt more quickly, but the larger step size eventually causes the filter to become unstable. This behavior is consistent with the sensitivity of the LMS algorithm to the step size, where an excessively large value can cause successive coefficient updates to overshoot the desired solution and result in divergence.

Although neither algorithm successfully achieved the desired ANC behavior, the FARLS results appear somewhat more promising than the LMS results. FARLS is able to respond rapidly to changes in the error and, in some cases, temporarily reduce the error following a change in the filter coefficients. However, neither algorithm was able to converge to a stable steady-state error or produce a filter response corresponding to the expected acoustic path. Despite testing different algorithm parameters, input ranges, and physical configurations, a stable solution was not obtained. Further testing using a broadband excitation is therefore necessary to determine whether the lack of convergence is related to the single-tone excitation or to another characteristic of the system.

4.5 Broadband ANC

The purpose of this experiment was to extend the previous ANC experiment from a single-tone input to broadband noise. Snoring is a broadband acoustic signal, and therefore the ANC system must be capable of operating with broadband inputs for the system to be applicable to active snore cancellation. For a properly converging adaptive filter, the error signal should generally decrease as the filter adapts to the acoustic environment and determines an appropriate cancellation response. This experiment was therefore used to determine whether the adaptive filters could converge when operating with a broadband input.

To conduct the experiment, a white-noise signal was generated by the computer and output through the primary speaker. A microphone positioned near the primary speaker provided the reference signal to the adaptive filter. The adaptive filter used this reference signal to calculate the cancellation signal, which was output through the secondary speaker. An error microphone was positioned within the acoustic field to measure the resulting signal after the primary and secondary signals interacted. The error signal was then fed back into the adaptive filter to update the filter coefficients. The experiment was operated for 30 seconds, and the resulting error signal was recorded for analysis.

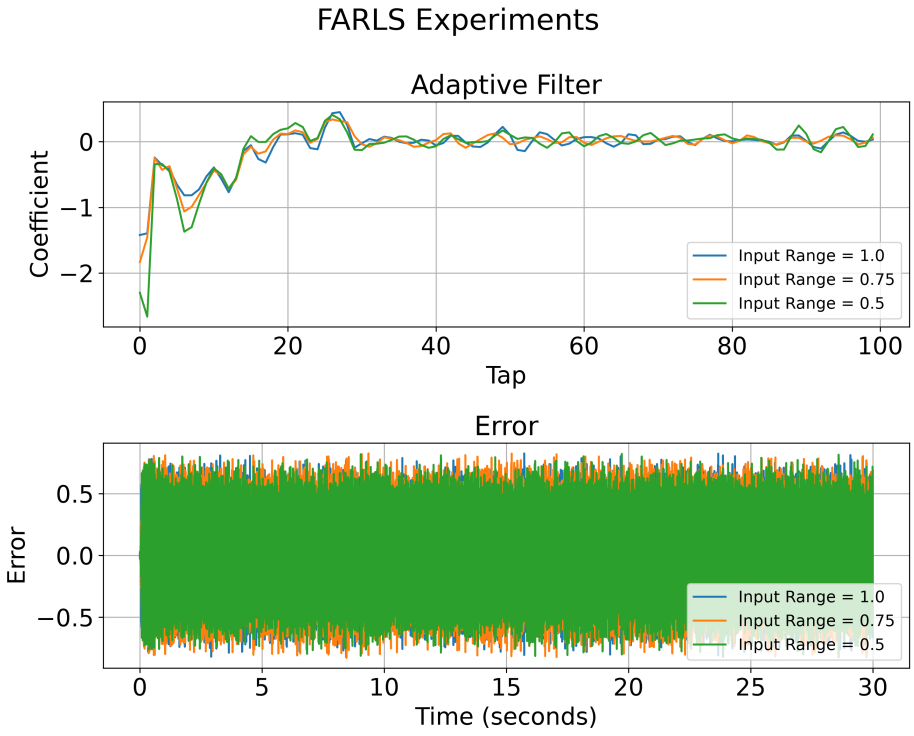


Fig. 4.6: Broadband ANC results using the FARLS adaptive filter for the tested input ranges.

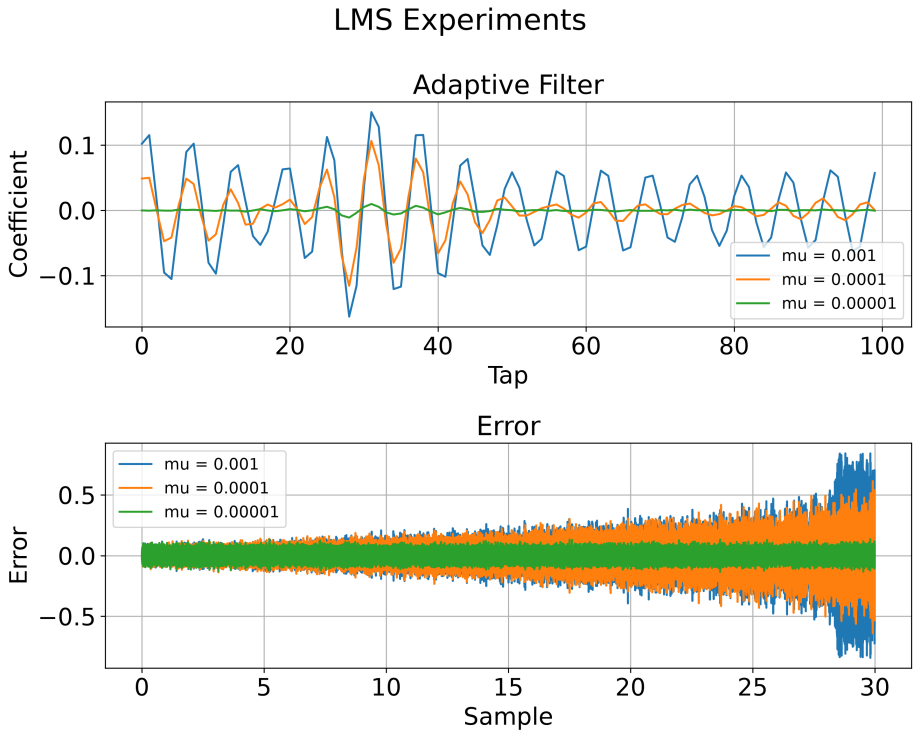


Fig. 4.7: Broadband ANC results using the LMS adaptive filter for the tested step sizes.

The results did not show the expected reduction in the error signal associated with convergence. Instead, the error remained relatively consistent throughout the experiments, with neither adaptive filter producing a sustained reduction toward a steady-state value. This behavior was observed across the different input ranges and adaptive filter parameters that were tested. The results therefore indicate that changing the input level or adaptive filter parameters alone was not sufficient to achieve convergence with the broadband input.

The FARLS and LMS algorithms exhibited similar difficulties when operating with the broadband signal, although their adaptation behavior differed. The FARLS results show that the filter coefficients continued to change as the algorithm attempted to respond to the measured error, but these changes did not result in a stable reduction in the error signal. Similarly, the LMS algorithm responded to changes in its step size, with smaller values producing slower adaptation and larger values producing more rapid changes but a greater tendency toward instability. The inability of both algorithms to achieve stable convergence

suggests that the observed behavior is not limited to a particular adaptive algorithm.

Several possible causes of the lack of convergence were investigated during the debugging process. The first step was to verify the hardware and signal paths. The signal path from the computer to both speakers was traced to verify that the source and cancellation signals were being transmitted correctly. The microphone signal paths were also examined to verify that the reference and error signals were being correctly received by the computer. After the hardware signal paths were verified, the investigation was focused on the software and adaptive filter parameters.

For the LMS implementation, multiple values of the step size μ were tested to determine whether the adaptation rate was preventing convergence. The initial conditions of the adaptive filters were also varied. The filters were initialized using an impulse response with a delay corresponding to the expected acoustic propagation time, white-noise values, and zero-valued coefficients. Additionally, the filter length was varied by testing lengths both longer and shorter than the expected acoustic response. None of these changes produced a significant improvement in the convergence behavior.

The results from these tests suggest that the adaptive filter parameters and initial conditions were unlikely to be the primary cause of the observed behavior. Another possible cause is the timing relationship between the reference signal, the calculated cancellation signal, and the resulting acoustic signal. In particular, a delay within the computer and audio system could cause the cancellation signal to arrive at the error microphone later than required. If the delay is sufficiently large, the adaptive filter may be unable to generate the cancellation signal at the appropriate time relative to the noise being cancelled. This would prevent the adaptive filter from accurately modeling the acoustic path, even if the filter parameters were otherwise appropriate.

The possibility of an unaccounted system delay is investigated in the following experiment using cross-correlation. By measuring the relative timing between the generated signal and the microphone signals, the experiment can determine whether the computer-based signal path introduces additional delay beyond the expected acoustic propagation

time. This provides a means of testing whether system timing may be contributing to the lack of convergence observed in the broadband ANC experiments.

4.6 Cross-Correlation Analysis of System Delay

The purpose of this experiment was to investigate the source of the behavior observed in the previous broadband ANC experiment. In particular, the experiment was used to investigate the delay between the signal generated inside the computer and the signal observed at the microphones. A delay between the generated signal and the signal received by the microphones can affect the ability of the adaptive filter to correctly model the acoustic path and produce a cancelling signal at the appropriate time.

Cross-correlation was used to determine the relative delay between the recorded signals. The peak of the cross-correlation indicates the lag at which the two signals have the greatest similarity. For the two microphone signals, this lag represents the relative propagation delay between the two microphone locations. The cross-correlation between the generated signal and a microphone signal includes both the acoustic propagation time and any additional delay introduced by the computer and audio system.

For this experiment, a sinusoidal signal was generated inside the computer and output through the audio system to a speaker. Two microphones were used to record the resulting acoustic signal. One microphone was positioned near the speaker, while the second microphone was positioned farther away from the speaker. The signals from both microphones were recorded along with the generated signal so that the relative delays between the signals could be determined.

Two cross-correlations were used to analyze the system delay. The first was calculated between the two microphone signals. This measurement was used to determine the relative delay caused by the difference in acoustic propagation distance between the two microphones. The second cross-correlation was calculated between the generated signal and the farther microphone signal. This measurement represents the combined delay from generating the signal, passing through the computer and audio system, and propagating through the acoustic path to the microphone.

Comparing the delay measured between the two microphones with the delay between the generated signal and the far microphone provides a way to identify the additional delay present in the computer-based signal path. The difference between these measurements can be compared with the expected acoustic propagation delay to determine whether the system contains a delay that was not accounted for in the adaptive filter.

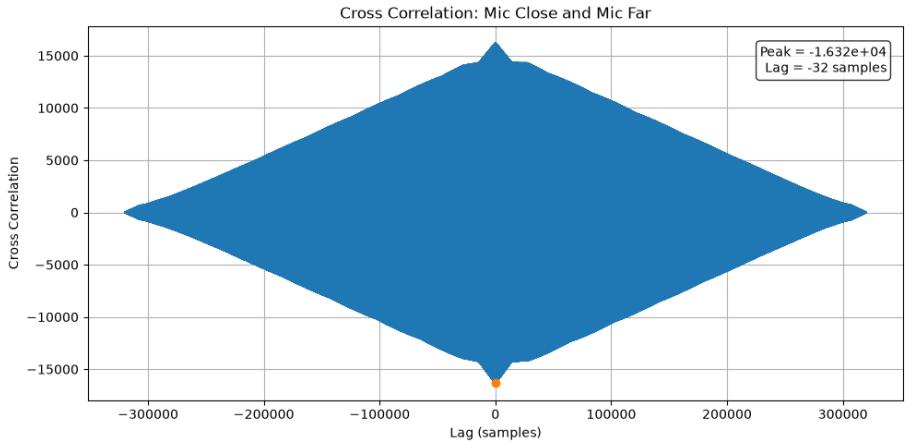


Fig. 4.8: Cross-correlation between the near and far microphone signals for a sinusoidal acoustic source.

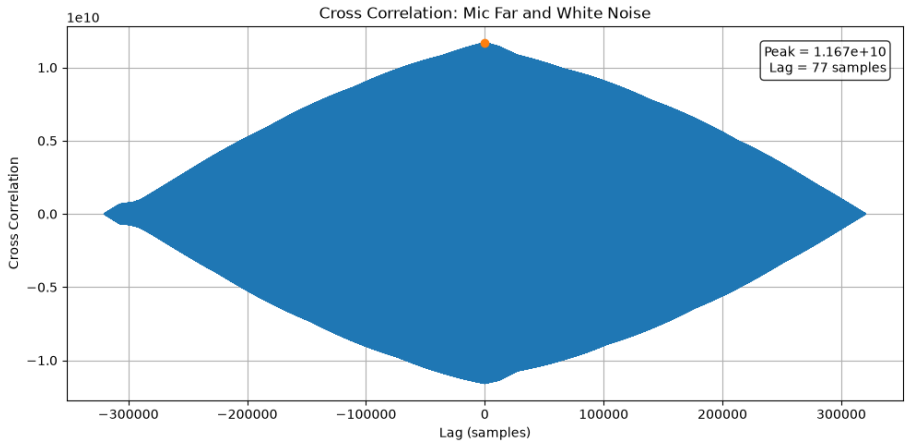


Fig. 4.9: Cross-correlation between the generated sinusoidal source signal and the far microphone signal.

Looking at Figure 4.8, the cross-correlation between the two microphones shows a peak at a lag of approximately 32 samples. This indicates that it takes approximately 32 samples for the acoustic signal to propagate from the location of one microphone to the other. Using the sampling frequency of 8000 Hz and the speed of sound of approximately 343 m/s, the distance corresponding to this delay can be calculated as

$$\frac{32 \text{ samples}}{8000 \text{ samples/s}} \times 343 \text{ m/s} = 1.372 \text{ m}.$$

This calculated distance corresponds closely with the distance between the two microphones in the experimental setup. This shows that the cross-correlation accurately measures the relative propagation delay between the two microphones.

In Figure 4.9, the cross-correlation between the generated signal and the far microphone shows a much larger delay. Since the near microphone was positioned very close to the speaker, the propagation time from the speaker to the far microphone should have been only slightly greater than the propagation time measured between the two microphones. However, the cross-correlation shows a peak at approximately 77 samples. This corresponds to a propagation distance of

$$\frac{77 \text{ samples}}{8000 \text{ samples/s}} \times 343 \text{ m/s} = 3.301 \text{ m}.$$

This distance is significantly greater than the physical distance between the speaker and the far microphone. The difference indicates that the measured delay includes additional latency from the computer-based signal path in addition to the acoustic propagation time. The results therefore indicate that there is an unaccounted-for delay between generating the signal in the computer and the signal being observed at the microphone.

The additional delay is significant for the ANC system because the adaptive filter must produce a cancelling signal that is correctly aligned in time with the acoustic signal being cancelled. If the delay introduced by the computer and audio system is not included in the model used by the adaptive filter, the cancelling signal may arrive at the error microphone

later than the portion of the noise signal it is intended to cancel. This temporal misalignment can prevent the adaptive filter from correctly reducing the error signal and may explain the lack of convergence observed during the previous broadband ANC experiment. Further investigation of the system delay and its inclusion in the adaptive filter could therefore improve the ability of the ANC system to converge.

4.7 Spatial ANC with LAGS

The purpose of this experiment is to demonstrate that spatial ANC is possible using the existing hardware setup and to provide a simple method for determining the delay required to produce acoustic cancellation. The previous ANC experiments used adaptive filters to determine the appropriate cancellation signal. However, the results of those experiments indicated that the delay within the system may prevent the adaptive algorithms from correctly adapting to the acoustic environment. This experiment therefore removes the adaptive portion of the system and instead uses a known delay to determine whether cancellation can be achieved when the timing of the cancellation signal is controlled.

The experiment uses an internally generated white-noise signal as the source signal. The signal is generated by the computer and output through a source speaker. Rather than using a reference microphone and adaptive filter to determine the cancellation signal, the same internally generated signal is used directly to produce the cancellation signal. A delayed copy of the source signal is generated and output through a second speaker. The second speaker therefore produces a cancellation signal that is related directly to the source signal but delayed by a known number of samples. The sign of the cancellation signal can also be changed to produce either an in-phase or out-of-phase signal at the cancelling speaker.

A single microphone is used as the error microphone to measure the combined acoustic response of the source and cancelling speakers. Using a single error microphone simplifies the experiment because the objective is not to adaptively determine the cancellation signal, but rather to determine whether a suitable delay exists that produces a minimum measured error. The experiment therefore provides a controlled test of whether the spatial arrange-

ment of the speakers and microphone can produce acoustic cancellation when the signal timing is known.

The delay applied to the cancellation signal is swept through a range of whole-sample delays. Each delay value is maintained for four seconds. During the first two seconds, the delayed signal is output with one polarity, and during the second two seconds, the polarity is reversed. This produces two measurements for each delay value and allows the effect of the relative phase between the source and cancelling signals to be observed. The delay is then changed and the process is repeated for the next delay value.

The mean-square (MS) value of the error signal is calculated separately for each two-second interval. The mean-square value is calculated according to

$$\mathrm{MS} = \frac{1}{N} \sum_{n=0}^{N-1} e[n]^2,$$

where $e[n]$ is the sampled error signal and N is the number of samples contained within the two-second interval. The calculated mean-square value provides a measure of the residual acoustic signal measured by the error microphone. A lower mean-square value therefore indicates that the source and cancelling signals are producing greater acoustic cancellation at the microphone.

The delay sweep provides a direct method of determining the delay that produces the greatest cancellation. If the cancellation speaker produces a signal with the appropriate delay and polarity relative to the source speaker, the measured mean-square error should decrease. Conversely, delays that do not correctly align the cancellation signal with the acoustic signal from the source speaker should produce a larger mean-square error. The minimum measured mean-square value can therefore be used to identify the delay that most closely matches the required acoustic propagation and system timing.

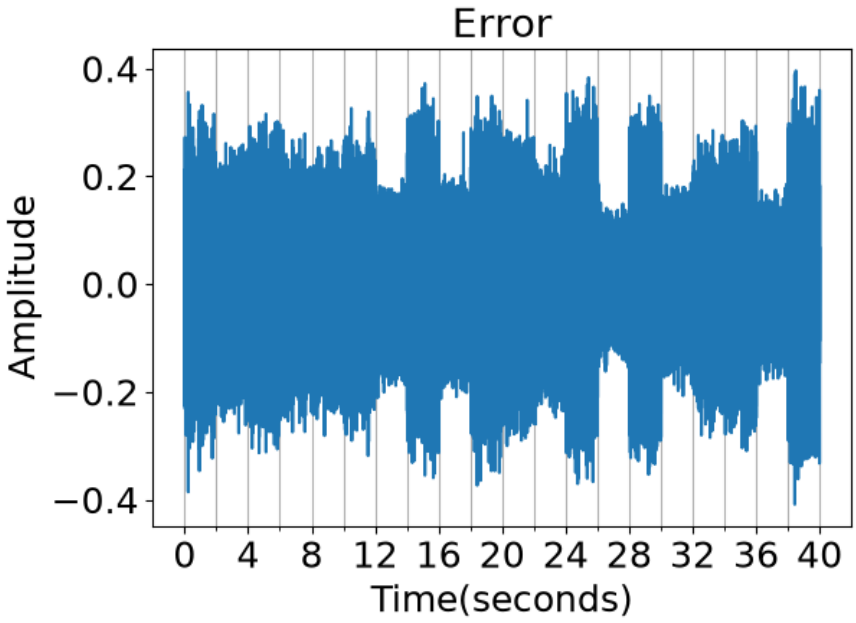


Fig. 4.10: Measured mean-square error as a function of the delay applied to the cancellation signal.

Table 4.4: Mean-square error values calculated over selected two-second time windows.

Time Window (s)	Mean-Square Error
12.0–14.0	0.006552
16.0–18.0	0.007804
22.0–24.0	0.009699
26.0–28.0	0.003814
30.0–32.0	0.006500
36.0–38.0	0.005162

The results in Figure 4.10 show that the measured mean-square error changes as the

delay applied to the cancellation signal is varied. As the delay approaches the delay required for the acoustic signals to align at the error microphone, the measured mean-square error decreases. The lowest mean-square value occurs between approximately 26 and 28 seconds in the recording, corresponding to a cancellation-signal delay of 14 samples with negative polarity. As shown in Table 4.4, the 26–28 s window produced a mean-square value of 0.003814, which was the lowest value among the selected time windows. This indicates that the cancellation signal produces the greatest reduction in the measured acoustic response when it is delayed by approximately 14 samples and inverted relative to the source signal.

The delay corresponding to the minimum error can be converted to a time delay using the sampling frequency of 8 kHz. The measured delay is therefore

$$t_d = \frac{14}{8000} = 0.00175 \text{ s} = 1.75 \text{ ms}.$$

This measured delay agrees with the estimated difference in acoustic propagation time between the source and cancelling speakers to the error microphone. The result indicates that the timing of the cancellation signal has a significant effect on the amount of acoustic cancellation produced at the error microphone.

The experiment does not produce complete cancellation because the delay is limited to integer numbers of samples. The actual acoustic propagation delay may fall between two sample periods, resulting in a fractional-sample delay that cannot be represented by the simple delay used in this experiment. Nevertheless, the experiment demonstrates that the hardware is capable of producing spatial acoustic cancellation when the relative timing and polarity of the source and cancellation signals are appropriately selected. A future implementation using fractional-delay filtering could provide finer control of the cancellation-signal timing and potentially produce greater cancellation.

CHAPTER 5

Conclusions and Future Work

5.1 Summary of Research

The objective of this research was to investigate the feasibility of using Active Noise Control (ANC) to reduce the perceived sound of snoring in a shared sleeping environment. Snoring is a common source of sleep disturbance that can negatively affect both the individual who snores and their sleeping partner. Existing approaches, such as earplugs, earbuds, or CPAP devices require the user to wear physical devices or address only specific medical causes of snoring. This work explored an alternative approach in which unwanted sound is reduced acoustically without requiring either individual to wear additional equipment.

The long-term vision of this research is the development of a practical active noise control system that could be integrated into a bedroom environment, such as within a headboard or bedside fixture. Such a system would provide localized noise reduction while remaining unobtrusive to both occupants.

To investigate this concept, a configurable experimental platform was designed and constructed. A 16-element microphone array and a 16-element loudspeaker array were developed to support near-field array processing experiments. Additional hardware, including custom interface enclosures and a programmable analog low-pass filter bank, was designed and fabricated to improve system integration and signal quality. The resulting platform provides a flexible environment for evaluating beamforming and adaptive noise control algorithms.

The accompanying software was expanded from previous work to support real-time operation under Linux. Fractional delay beamforming was implemented using a combination of integer delay compensation and sinc interpolation, enabling accurate time alignment across the microphone array. Adaptive filtering algorithms were then incorporated to gener-

ate acoustic cancellation signals. A Least Mean Squares (LMS) implementation was initially developed to establish a computationally efficient baseline, followed by implementation of the Fast Array Recursive Least Squares (FARLS) algorithm to improve convergence speed while maintaining real-time performance.

In general, this research established a complete hardware and software platform for investigating active snore cancellation. The developed system integrates microphone arrays, loudspeaker arrays, custom electronics, real-time signal processing, and adaptive filtering into a single experimental framework that provides a foundation for future research in acoustic noise control.

5.2 Contributions

This research resulted in several contributions toward the development of a real-time active snore cancellation platform.

The first contribution was the design and construction of a configurable experimental platform consisting of a 16-element microphone array and a 16-element loudspeaker array. Unlike the previous experimental setup, the new platform allows the height, orientation, and spacing of both arrays to be adjusted independently. This flexibility enables the system geometry to be modified for different beamforming and localization experiments while also providing a reusable platform for future research.

The second contribution was the implementation of real-time near-field array processing using integer delay compensation combined with sinc interpolation. The developed implementation provides accurate fractional delay estimation while remaining computationally efficient for real-time execution. This capability serves as the foundation for the beamforming and adaptive noise control algorithms used throughout this work.

The third contribution was the design and fabrication of supporting electronics, including custom interface enclosures and a programmable analog low-pass filter bank. These components improved signal quality between the digital-to-analog converter and the loudspeaker array while providing a practical interface between the experimental hardware and the workstation.

The fourth contribution was the implementation of adaptive noise control algorithms within the real-time software framework. A Least Mean Squares (LMS) implementation was developed to establish a computationally efficient baseline for testing the acoustic channel. Building upon previous offline implementations developed within the research group, the Fast Array Recursive Least Squares (FARLS) algorithm was adapted for real-time operation, providing a platform for evaluating improved convergence performance under practical operating conditions.

Together, these contributions establish a complete hardware and software platform capable of supporting future research in active snore cancellation and adaptive acoustic array processing.

5.3 Discussion

Throughout the development of the active noise control platform, several design decisions proved successful while others revealed limitations that provide opportunities for future improvement. The following discussion reflects on the major components developed during this research and the lessons learned throughout the design and implementation process.

5.3.1 Real-Time Array Processing

The real-time array processing implementation was one of the strongest aspects of this research. From the beginning, the software was designed to be modular and easily configurable so that future modifications could be made with minimal changes to the source code. Given the desired steering coordinates, the algorithm automatically computes the propagation delay for each sensor and generates the required integer and fractional delay filters. As a result, steering the array toward a new location only requires specifying a new set of coordinates rather than modifying the underlying implementation.

Dynamic memory allocation was also incorporated into the design, allowing the array processing algorithms to support different array geometries and interpolation filter lengths

without requiring substantial code modifications. This flexibility will simplify future research involving different microphone configurations or beamforming strategies.

The same sinc interpolation implementation was successfully adapted for the loudspeaker array. Rather than filtering incoming microphone signals, the interpolation filters were used to generate the appropriately delayed loudspeaker outputs, allowing the same processing framework to be used for both signal acquisition and signal reproduction.

5.3.2 Adjustable Experimental Platform

The adjustable microphone and loudspeaker platform also proved to be a significant improvement over the original experimental setup. Initially, the microphone array consisted of microphones taped to a 2×4 that was suspended from the ceiling using string. Although this arrangement allowed preliminary testing to be performed, modifying the array geometry or sensor height was cumbersome and time consuming.

Replacing the original structure with a PVC-based support frame produced a lightweight, adjustable support structure. The completed structure allows the height, orientation, and spacing of both the microphone and loudspeaker arrays to be adjusted independently. This flexibility significantly reduced setup time and enabled rapid evaluation of different array configurations throughout the research.

The addition of a removable acoustic baffle between the loudspeaker and microphone arrays also reduced direct acoustic coupling between the arrays, improving the quality of experimental measurements. One limitation of the PVC structure was its relatively low stiffness compared to the previous wooden support. Because the upper portion of the structure was heavier, slight movement of the array could occasionally occur. While this movement did not produce any noticeable degradation in experimental performance, a more rigid support structure could further improve mechanical stability in future implementations.

5.3.3 Programmable Filter Bank

The programmable analog filter bank performed as intended throughout testing. The ability to rapidly adjust the cutoff frequency allowed the hardware to accommodate mul-

tiple sampling configurations without requiring circuit modifications. During experimental evaluation, the filter bank successfully reduced the high-frequency artifacts produced by the DAC while preserving the desired audio content.

Although the PCB functioned reliably, several design improvements became apparent after fabrication. The overall board footprint was larger than necessary because minimizing board size was not a primary design objective during the initial layout. A more compact design with shorter trace lengths would likely improve manufacturability and produce a cleaner layout.

Another issue encountered during testing involved several capacitors that occasionally failed, resulting in intermittent operation of the filter bank. An infrared thermometer was used to identify the failing components, allowing the defective capacitors to be replaced. Following replacement, the PCB operated as expected for the remainder of the experiments. Despite these minor issues, the programmable filter bank proved to be an effective component of the overall active noise control platform.

5.3.4 Adaptive Filtering

The implementation of the LMS and FARLS adaptive filtering algorithms demonstrated that both algorithms were capable of adapting their filter coefficients in response to the measured error signal; however, neither algorithm achieved stable convergence during the experiments. For narrowband signals, including sinusoidal inputs, both algorithms exhibited adaptation, but the resulting filter coefficients did not converge to the expected acoustic response or produce sustained acoustic cancellation. Similar behavior was observed when broadband signals were introduced, with neither algorithm producing a consistent reduction in the measured error signal. These results indicate that the lack of successful cancellation was not limited to a particular adaptive algorithm or signal bandwidth.

The observed lack of convergence led to an investigation of the timing characteristics of the system. Cross-correlation analysis identified approximately 45 samples, or 5.625 ms at the 8-kHz sampling rate, of additional delay in the computer-based signal path beyond the expected acoustic propagation delay. This additional latency can significantly affect

adaptive noise cancellation because the cancellation signal must be appropriately aligned with the acoustic signal at the error microphone. A subsequent controlled-delay experiment demonstrated that spatial acoustic cancellation could be achieved when the cancellation signal was manually delayed and inverted, supporting the conclusion that system timing is an important consideration for the adaptive ANC implementation.

This observation suggests that improvements in the hardware platform may provide greater performance gains than further algorithmic optimization alone. Future implementations using dedicated digital signal processors (DSPs), field-programmable gate arrays (FPGAs), or lower-latency acquisition hardware could significantly reduce system latency and improve broadband active noise cancellation performance.

Overall, the adaptive filtering algorithms successfully demonstrated the feasibility of real-time active noise control within the developed experimental platform. Although hardware limitations prevented optimal broadband performance, the software framework provides a solid foundation for continued research and future implementation of more advanced adaptive filtering techniques.

Overall, this research successfully integrated configurable array hardware, custom electronics, real-time signal processing, and adaptive filtering into a unified experimental platform. While several opportunities for improvement remain, the completed system establishes a strong foundation for future investigations into active snore cancellation and adaptive acoustic array processing.

5.4 Future Work

Although this research established a functional platform for investigating active snore cancellation, several opportunities remain to improve both the hardware and software. The following sections outline the most promising directions for future development.

5.4.1 Hardware

The most significant limitation encountered during this research was the latency introduced by the data acquisition and playback hardware. Experimental results indicated

that the delay between acquiring microphone data and reproducing the corresponding loudspeaker output limited the performance of the adaptive filtering algorithms, particularly for broadband signals.

Future work should first characterize the limitations of the existing hardware to determine the maximum performance achievable with the current platform. Understanding these limitations will provide valuable design requirements for the next generation of hardware.

To further improve system performance, lower-latency hardware should be investigated. Possible solutions include newer analog-to-digital and digital-to-analog converters, dedicated digital signal processors (DSPs), or field-programmable gate arrays (FPGAs). These platforms would provide greater control over the acquisition and playback process while eliminating many of the software overheads associated with commercial data acquisition hardware. In addition, a custom data acquisition PCB designed specifically for the requirements of this project could further reduce latency and improve system integration.

5.4.2 Adaptive Algorithms

Although both the LMS and FARLS algorithms were successfully implemented, the hardware limitations prevented a complete evaluation of their performance for broadband active noise control. Once improved hardware is available, additional investigation should be conducted to compare the convergence speed, computational complexity, and cancellation performance of multiple adaptive filtering algorithms.

Potential algorithms for future evaluation include Normalized LMS (NLMS), Filtered-x LMS (FxLMS), conventional Recursive Least Squares (RLS), and further refinements of the FARLS algorithm. A systematic comparison of these approaches would help determine the most appropriate adaptive algorithm for real-time active snore cancellation.

5.4.3 Full Array Evaluation

During the development process, many experiments were intentionally simplified to isolate individual subsystems and reduce sources of error. While this approach accelerated

development and debugging, it limited the evaluation of the complete active noise control system.

Future work should include comprehensive testing using the full microphone and loudspeaker arrays operating within the anechoic chamber. After validating the complete system under controlled laboratory conditions, experiments should be extended to acoustically realistic environments, such as a simulated bedroom. These experiments would provide insight into the effects of reflections, room acoustics, and changing source positions on overall cancellation performance and would represent an important step toward practical deployment of the technology.

5.5 Final Remarks

Overall, this research successfully demonstrated the design and implementation of a configurable real-time active noise control platform for investigating acoustic snore cancellation. The completed system integrates adjustable microphone and loudspeaker arrays, custom interface electronics, programmable analog filtering, real-time near-field array processing, and adaptive filtering algorithms into a unified experimental framework.

Although additional work is required before the system can be deployed in practical bedroom environments, this research establishes a strong foundation for future development. The modular hardware and software architecture allows future researchers to improve individual components without redesigning the entire system, making the platform well suited for continued investigation into adaptive acoustic array processing and active noise control.

Ultimately, this work demonstrates the feasibility of developing a flexible research platform capable of supporting future advancements in active snore cancellation. It is hoped that the hardware, software, and methodologies developed throughout this research will contribute to future systems that improve sleep quality through unobtrusive acoustic noise control.

REFERENCES

[1] T. K. Moon, M. Shekaramiz, and J. H. Gunther, "Source localization using a single array of sensors in nearfield configuration, and application to active noise cancellation," in *2024 Intermountain Engineering, Technology and Computing (IETC)*, 2024, pp. 64–68.

[2] P. Hou, C. Xia, and W. Liu, "Efficient design of nearfield focused beamformer with frequency invariant patterns for vertical line array," in *2013 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2013)*, 2013, pp. 1–5.

[3] R. Kennedy, T. Abhayapala, D. Ward, and R. Williamson, "Nearfield broadband frequency invariant beamforming," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 2, 1996, pp. 905–908 vol. 2.

[4] H. Chen, Y. Bao, and W. Ser, "Effects of sensor position errors on farfield/nearfield wideband beamformers for microphone arrays," *IEEE Sensors Journal*, vol. 15, no. 9, pp. 4812–4825, 2015.

[5] M. Peake, M. Karan, and D. Gray, "A hierarchical algorithm for nearfield acoustic imaging," in *Proceedings of the Tenth IEEE Workshop on Statistical Signal and Array Processing (Cat. No.00TH8496)*, 2000, pp. 712–716.

[6] T. K. Roy and M. Morshed, "Active noise control using filtered-xlms and feedback anc filter algorithms," in *2013 2nd International Conference on Advances in Electrical Engineering (ICAEE)*, 2013, pp. 7–12.

[7] N. Miyazaki and Y. Kajikawa, "Adaptive feedback anc system using virtual microphones," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 383–387.

[8] L. Liu, K. R. Pottim, and S. M. Kuo, "Ear field adaptive noise control for snoring: an real-time experimental approach," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 1, pp. 158–166, 2019.

[9] S. Chakravarthy and S. Kuo, "Application of active noise control for reducing snore," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 5, 2006, pp. V–V.

[10] H. Kaur and R. Talwar, "Performance comparison of adaptive filter algorithms for noise cancellation," in *2013 International Conference on Emerging Trends in Communication, Control, Signal Processing and Computing Applications (C2SPCA)*, 2013, pp. 1–5.

[11] F.-Y. Wu, H.-Z. Yang, and Z. Liu, "A fast array rls equalizer for phase coherent underwater acoustic communication," *IEEE Journal of Oceanic Engineering*, vol. 51, no. 1, pp. 676–689, 2026.

[12] S. M. Kuo and R. Gireddy, "Real-time experiment of snore active noise control," in *2007 IEEE International Conference on Control Applications*, 2007, pp. 1342–1346.

[13] S. Nobili, V. Bruschi, F. Bettarelli, and S. Cecchi, "A real time subband implementation of an active noise control system for snoring reduction," in *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2021, pp. 109–114.

[14] ——, "An efficient active noise control system with online secondary path estimation for snoring reduction," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 156–160.

[15] V. Bruschi, M. Cantarini, L. Serafini, S. Nobili, S. Cecchi, and S. Squartini, "An enhanced system for the detection and active cancellation of snoring signals," in *2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*, 2023, pp. 1–6.

[16] K.-C. Liao and C.-Y. Chang, "Application of digital signal processor to snore cancellation," in *Proceedings 2011 International Conference on System Science and Engineering*, 2011, pp. 218–222.

[17] L. Serafini, V. Bruschi, S. Nobili, E. Principi, S. Cecchi, and S. Squartini, "Joint detection and active cancellation of snoring signals in real-time," in *2023 4th International Symposium on the Internet of Sounds*, 2023, pp. 1–9.

APPENDICES

APPENDIX A

System Operation and Implementation

A.1 Setting Up the ADC and DAC in a Linux Environment

After the computer is powered off, restarted, or rebooted, several commands must be executed before the ADC and DAC APIs can be used. The first step is to obtain superuser privileges. For the `tmoon` account, this is accomplished using the following command:

```
sudo su
```

When prompted, enter the password associated with the `tmoon` account:

```
[sudo] password for tmoon: shoelace
```

After obtaining superuser privileges, the directories containing the ADC and DAC driver files must be accessed. The following commands initialize the ADC and DAC drivers, respectively.

For the ADC driver, navigate to the ADC driver directory and execute the following commands:

```
cd /usr/src/linux/drivers/16ai32ssc/driver
make clean
make all
./start
```

For the DAC driver, navigate to the DAC driver directory and execute the same commands:

```
cd /usr/src/linux/drivers/16ao16/driver
make clean
make all
./start
```

The commands above must be executed in both the ADC and DAC driver directories. Failure to initialize either driver may result in errors when attempting to use the ADC or DAC APIs.

After these steps have been completed for both drivers, the ADC and DAC APIs are initialized and ready for use.

A.2 How to Use the Code

From the home directory of the `tmoon` account, navigate to the directory containing the real-time array processing code using the following command:

```
cd /Documents/Nathan/code/real_time_array_processing
```

The following files are the primary source files used during the experiments:

```
main.c
main.h
Continuous_reading.c
makefile
pmatlib.h
pmatlib.c
```

The files `main.c`, `main.h`, and `Continuous_reading.c` contain the primary code used to configure and execute the experiments and are the main files that may need to be modified when changing the experimental configuration. The `pmatlib.h` and `pmatlib.c` files contain supporting functions used by the program and generally do not need to be modified.

The `makefile` should only need to be modified when an additional source file is added to the program. If a new source file is added, the appropriate source-file list in the `makefile` must be updated so that the new file is included during compilation.

To compile the program and run the selected test, execute the following commands:

```
make all
./continuous_reading
```

The `make all` command compiles the source code and generates the executable. The `./continuous_reading` command then executes the compiled program using the configuration specified in the source files.

The following sections describe the purpose of each of the primary files and explain which portions of the code should be modified when configuring the system for different experiments.

A.3 Supporting Functions

This appendix presents selected functions from the main source file used to implement the active snore cancellation system. The primary source file is `Continuous_reading.c`. The complete source file contains additional hardware configuration, data acquisition, and supporting functions that are not reproduced here. The functions included in this appendix were selected because they illustrate the principal computational steps of the proposed system.

The following function implements the LMS update step used for adaptive filtering. The filter coefficients are updated using the error signal, the step size, and the corresponding samples stored in the filter buffer.

```
void update_filter(double* filter, double* filter_buffer, double error, double
    step_size, int length){
  for(int i = 0; i < length; i++){
    filter[i] = filter[i] + step_size * error * filter_buffer[i];
  }
}
```

The following function implements the FIR filtering operation. The input buffer is shifted to make room for the newest sample, after which the filter output is calculated as the sum of the products between the filter coefficients and the corresponding buffer samples.

```
double FIR_filtering(double new_input, double* filter, double* filter_buffer, int
    filter_length){
   double sum = 0;
```

```
   for(int i = filter_length - 1; i > 0; i--){
      filter_buffer[i] = filter_buffer[i - 1];
      sum += filter[i] * filter_buffer[i];
   }
   filter_buffer[0] = new_input;
   sum += filter[0] * filter_buffer[0];

   return(sum);
}
```

The following functions implement the fractional-delay filters used for array processing. The function `build_filters()` calculates the propagation delay between the source position and each microphone position. The maximum propagation delay is then used as a reference to determine the relative delay for each sensor. This allows the individual sensor signals to be aligned to a common reference.

The relative delay is separated into an integer-sample delay and a fractional-sample delay. The integer portion is implemented using leading zeros, while the fractional portion is implemented using a truncated sinc filter. The resulting filter therefore combines the bulk delay and fractional delay into a single filter. The filter length is determined by the integer delay and the interpolation length, where P defines the half-length of the sinc filter.

The function `delaysinc()` computes the fractional-delay sinc coefficients. The parameter D defines the spacing between adjacent microphones.

```
void build_filters(double x_pos, double y_pos, double** filter_bank, u32*
    filter_length, double** filter_buffer, int num_sensors, float start_pos){
   double tau[num_sensors];
   double taup;
   double mic_pos = start_pos;
   double max = 0;

   for(int i = 0; i < num_sensors; i++){
```

```
      tau[i] = sqrt((mic_pos - x_pos)*(mic_pos - x_pos)+(-y_pos)*(-y_pos))/C;

      if (tau[i] > max){
         max = tau[i];
      }
      mic_pos += D;
   }

/* Build combined delay + fractional-delay filter */
   for (int i = 0; i < num_sensors; i++){
      taup = max - tau[i];

      double nTs = taup * Fs;
      int nTsint = floor(nTs);

      CALLOCVECTOR(filter_buffer[i], double,nTsint + 2 * P + 1);
      filter_length[i] = nTsint + 2 * P + 1;
      delaysinc(taup, filter_bank[i]);
   }
}

void delaysinc(double taup, double* filter){
   double nTs = taup * Fs;
   int nTsint = floor(nTs);
   double nTsfrac = nTs - nTsint;
   taup = nTsfrac * Ts;

   double tlist[2 * P + 1];

   if(nTsint != 0){
      for(int i = 0; i < nTsint; i++)
         filter[i] = 0;
```

```
   }

   for(int i = 0; i < 2 * P + 1; i++){
      tlist[i] = (((double)(i - P) / Fs) - taup)* B * 2 * M_PI;
   }

/*
 * Store the sinc filter immediately after the
 * integer-delay zeros.
 */
   for(int i = 0; i < 2 * P + 1; i++){
      if(tlist[i] != 0){
         filter[nTsint + i] = sin(tlist[i]) / tlist[i];
      }
      else{
        filter[nTsint + i] = 1.0;
      }
   }
}
```

The following functions configure the input and output devices used during the experiments. The output configuration function initializes the analog output hardware, selects the operating mode, and configures the sampling rate, output range, and data format. The input configuration function similarly configures the analog input hardware and selects the channels and input range used for data acquisition.

```
int setup_output(int* errs, int _io_mode){
   int qty = 0;
   s32 def_index = 0;
   int fd = 0;

   *errs += ao16_init_util(1);
   *errs += os_id_driver(ao16_open, ao16_read, ao16_close);
```

```
   *errs += ao16_count_boards(1, &qty);

   if ((*errs) || (qty <= 0))
      return(*errs);

   *errs = ao16_open_util(def_index, 0, -1, 1, &fd);
   *errs += ao16_config_ao(fd, -1, 1, RATE);
   *errs += ao16_tx_io_mode(fd, -1, 1, _io_mode, NULL);
   *errs += ao16_buffer_over_frame(fd, -1, 1, AO16_BUFFER_OVER_FRAME_CLR, NULL);
   *errs += ao16_buffer_over_data(fd, -1, 1, AO16_BUFFER_OVER_DATA_CLR, NULL);
   *errs += ao16_range(fd, -1, 1, AO16_RANGE_1_25, NULL);
   *errs += ao16_channel_sel(fd, -1, 1, 0xFFFF, NULL);
   *errs += ao16_fsamp_report_all(fd, -1, 1, NULL);
   *errs += ao16_data_format(fd, -1, 1, AO16_DATA_FORMAT_2S_COMP, NULL);
   return(fd);
}

int setup_input(int fd, int _io_mode){
   s32 chans = 0;
   int errs = 0;

   errs += ai32ssc_config_ai(fd, -1, 1, NRATE);
   errs += ai32ssc_rx_io_mode(fd, -1, 1, _io_mode, NULL);
   errs += _set_channels(fd, &chans);
   errs += _set_threshold(fd);
   errs += ai32ssc_data_format(fd, -1, 1, AI32SSC_DATA_FORMAT_2S_COMP, NULL);
   errs += ai32ssc_ain_range(fd, -1, 1, AI32SSC_AIN_RANGE_5V, NULL);
   return(errs);
}
```

The following functions implement the numerical operations used by the FARLS adaptive filtering algorithm. The matrix initialization establishes the initial state of the recursive

filter, while the mixed circular and hyperbolic rotations are used to update the factorized representation during each iteration.

```
void init_A(float A[][3]){
   int i,j;

   /* Initialize filter */
   for(i=0; i<M+2; i++){
      for(j=0; j<3; j++){
         A[i][j] = 0.0;
      }
   }

   A[0][0] = 1.01; /* Gamma */
   A[1][1] = etart*lamrt;            /* L initialization */
   A[M+1][2] = etart*lamrt*lamrtM;   /* L initialization */
}



void givens_mixed(float a, float b, float* rho, float* rootoneplusrhosquared){
   if(b == 0.0){
      *rho = 0.0;
      *rootoneplusrhosquared = 1.0;
   }
   else if(a == 0.0){
      fprintf(stderr,"Problem␣with␣givens␣rotation\n");
   }
   else{
      if(a < FLT_EPSILON*3.0){
         fprintf(stderr,"Dividing␣by␣small␣number␣%20.10f\n",a);
      }

      *rho = b/a;
```

```
      *rootoneplusrhosquared = sqrt(1.0+(*rho)*(*rho));
   }
}

void apply_circ_mixed_rot(float* a, float* b, float rho, float
    rootoneplusrhosquared){
   *a = ((*a) + rho*(*b))/rootoneplusrhosquared;
   *b = rootoneplusrhosquared*(*b) - rho*(*a);
}

int hyper_OD(float a, float b, float *c, float *s){
   int flag = 1;

   if(a-b == 0.0){
      fprintf(stderr,"Divide␣by␣zero␣%f,%f\n",a,b);
   }

   *c = sqrt((a+b)/(a-b))*0.5;
   *s = sqrt((a-b)/(a+b))*0.5;

   return flag;
}


void apply_hyper_OD(float* a, float* b, float c, float s){
   float x, y;

   x = (*a)-(*b);
   y = (*a)+(*b);

   x *= c;
   y *= s;
```

```
   (*a) = y+x;
   (*b) = y-x;
}
```

```
\appendixsection{Active Snore Cancellation Implementation}
\label{sec:functional_code}

This section presents selected portions of the functional code used to implement
    the active snore cancellation system. The primary function presented is \
    texttt{snore_cancellation()}, which integrates the major components of the
    system, including microphone-array processing, adaptive filtering, and speaker
    -array processing. The function demonstrates how these individual components
    are combined to perform real-time active noise cancellation.

The code can also be separated into individual components to implement or evaluate
     specific portions of the system independently. For example, the microphone-
    array processing and speaker-array processing can be isolated from the
    adaptive filtering algorithm for individual testing. Similarly, the adaptive
    filtering portion can be modified to evaluate different adaptive algorithms or
     parameter configurations.

The microphone-array processing applies the calculated propagation-delay filters
    to the microphone signals before the resulting array output is provided to the
     adaptive filtering algorithm. The adaptive filter then generates the
    cancellation signal \(y\), which is subsequently processed by the speaker-
    array filters before being transmitted to the output channels.

\begin{lstlisting}[language=C]
int snore_cancellation(int i_fd, int _io_mode){
```

```
int errs = 0;
s32 fd = 0;
int i = 0;
int seconds = 10;
int num_samples = Fs * seconds;

fd = setup_output(&errs, _io_mode);
errs += setup_input(i_fd, _io_mode);

float A[M+2][3] = {0}; /* Pre & post array */
float u[M+1] = {0};
float s, c; /* Circular rotation parameters */
float sh, ch; /* Hyperbolic rotation parameters */
float e_new=0.0;    /* A priori prediction error */
float e_old=0.0;
float g_old=0.0;
int j;

/* Filter parameters */
float lam = 1;
float lamrtinv;
float eta = 1.0;
lamrt = sqrt(lam);
etart = sqrt(eta);
lamrtM = pow(lamrt,(float)M);
lamrtinv = .999;
init_A(A);

double** filter_bank_m;
u32* filter_lengths_m;
double** filter_buffer_m;
```

```
CALLOCVECTOR(filter_bank_m, double*, NUM_MICS);
CALLOCVECTOR(filter_lengths_m, u32, NUM_MICS);
CALLOCVECTOR(filter_buffer_m, double*, NUM_MICS);

build_filters(0, .75, filter_bank_m, filter_lengths_m,
filter_buffer_m, NUM_MICS, MIC_START);

double** filter_bank_s;
u32* filter_lengths_s;
double** filter_buffer_s;

CALLOCVECTOR(filter_bank_s, double*, NUM_SPEAKERS);
CALLOCVECTOR(filter_lengths_s, u32, NUM_SPEAKERS);
CALLOCVECTOR(filter_buffer_s, double*, NUM_SPEAKERS);

build_filters(0.4572, 0.5588, filter_bank_s, filter_lengths_s,
filter_buffer_s, NUM_SPEAKERS, SPEAKER_START);

double* ad_filter;
double* ad_filter_buffer;

CALLOCVECTOR(ad_filter, double, NUM_TAPS);
CALLOCVECTOR(ad_filter_buffer, double, NUM_TAPS);

for(int i = 0; i < NUM_TAPS; i++){
   ad_filter[i] = ((rand()&0xF)/16.0 - .5)/10000;
   ad_filter_buffer[i] = 0;
}

u32* out;
CALLOCVECTOR(out, u32, 16);
```

```
double* error;
CALLOCVECTOR(error, double, num_samples);

float y = 0;

for(int k = 0; k < 16; k++)
   out[k] = 0;

int k = 0;
double temp = 0;

errs += ai32ssc_ain_buf_clear(i_fd, -1, 1, 1);

while(i < num_samples){
   errs += _read_data(i_fd);
   errs += ai32ssc_ain_buf_overflow(i_fd, -1, 0, -1, NULL);
   errs += ai32ssc_ain_buf_underflow(i_fd, -1, 0, -1, NULL);

   /* Microphone array processing */
   temp = 0;

  for(k = 0; k < NUM_MICS; k++){
     temp += FIR_filtering(
     (double)(_buffer[k] & 0x0000FFFF)/32767.0,
     filter_bank_m[k],
     filter_buffer_m[k],
     filter_lengths_m[k]);
  }

  /* Input a new sample */
  u[0] = (double)((s16)(_buffer[0]&0x0000FFFF)/32768.0);
```

```
/* Initialize for this iteration */
A[0][1] += u[0]*A[1][1];
A[0][2] += u[0]*A[1][2];
g_old = A[M+1][0];

/* Compute rotations and apply them to the array */
givens_mixed(A[0][0], A[0][1], &c, &s);
apply_circ_mixed_rot(&(A[0][0]),&(A[0][1]), c, s);
apply_circ_mixed_rot(&(A[M+1][0]),&(A[M+1][1]), c, s);

hyper_OD(A[0][0],A[0][2], &ch, &sh);
apply_hyper_OD(&(A[0][0]),&(A[0][2]), ch, sh);
apply_hyper_OD(&(A[M+1][0]),&(A[M+1][2]), ch, sh);

/* Complete last iteration */
A[M+1][1] *= lamrtinv;
A[M+1][2] *= lamrtinv;

/* Set the a priori error */
e_new = temp;

A[0][1] = 0.0;
A[0][2] = 0.0;

y = 0;

for(j=M; j>0; j--){
   /* Update adaptive filter coefficients */
   ad_filter[j-1] += g_old*e_old;

   /* Generate adaptive filter output */
   y += u[j-1]*ad_filter[j-1];
```

```
   /* Apply circular and hyperbolic rotations */
   g_old = A[j][0];

   apply_circ_mixed_rot(&(A[j][0]),&(A[j][1]), c, s);
   apply_hyper_OD(&(A[j][0]),&(A[j][2]), ch, sh);

   A[j][1] *= lamrtinv;
   A[j][2] *= lamrtinv;

   /* Shift the input data vector */
   u[j]=u[j-1];

   A[0][1] += u[j]*A[j+1][1];
   A[0][2] += u[j]*A[j+1][2];

   /* Shift Kalman gain */
   A[j+1][0] = A[j][0];
}

/* Calculate posterior error */
e_old = e_new/A[0][0];

A[1][0] = 0.0;

/* Save posterior filter error */
error[i] = e_old/A[0][0];

/* Speaker array processing */
for(k = 0; k < NUM_SPEAKERS; k++){
   out[k] = (u32)(32767 * FIR_filtering(
      y,
```

```
            filter_bank_s[k],
            filter_buffer_s[k],
            filter_lengths_s[k]));
      }

      /* Output signals to speaker array */
      ao16_write(fd, out, 4*16);

      if(errs > 0)
         break;

      i++;
   }
   return(errs);
}
```

The following function implements the lag experiment used to characterize the system delay. The code demonstrates the procedure used to evaluate different candidate delays by measuring the accumulated error energy over each test interval.

```
int lags(int i_fd, int _io_mode){
   int errs = 0;
   s32 fd = 0;
   int i = 0;
   int seconds = 100;
   int num_samples = seconds * Fs;

   int delay_start = 0;
   int delay_end = 50;
   int lag_index = delay_start;

   double scale_factor = .3;
   double y = 0;
```

```
fd += setup_output(&errs, _io_mode);
errs += setup_input(i_fd, _io_mode);

double* speaker_output;
CALLOCVECTOR(speaker_output, double, num_samples);

double* adaptive_output;
CALLOCVECTOR(adaptive_output, double, num_samples);

double* adaptive_input;
CALLOCVECTOR(adaptive_input, double, num_samples);

double* lag_buffer;
CALLOCVECTOR(lag_buffer, double, 200);

double* lag_filter;
CALLOCVECTOR(lag_filter, double, 200);

for(int j = 0; j < 200; j++){
   lag_filter[j] = 0;
   lag_buffer[j] = 0;
}

lag_filter[delay_start] = scale_factor;

u32* out;
CALLOCVECTOR(out, u32, 16);

for(int j = 0; j < NUM_SPEAKERS; j++)
   out[j] = 0;
```

```
double* error;
CALLOCVECTOR(error, double, num_samples);

int action = 0;
double cum_energy = 0;
double min_energy = 1E99;
double sign_min;
int lag_min = 0;

errs += ai32ssc_ain_buf_clear(i_fd, -1, 1, 1);

while(i < num_samples){
   errs += _read_data(i_fd);
   errs += ai32ssc_ain_buf_overflow(i_fd, -1, 0, -1, NULL);
   errs += ai32ssc_ain_buf_underflow(i_fd, -1, 0, -1, NULL);

   float random_val = ((float)rand()/RAND_MAX - 0.5);

   if(i > 1600 && (i % 16000) == 0 && lag_index < delay_end){
      if(cum_energy < min_energy){
         min_energy = cum_energy;
         lag_min = lag_index;
         sign_min = copysign(1.0, lag_filter[lag_index]);
         printf("found␣new␣min!␣min_energy␣=␣%f,␣lag_min␣=␣%d,␣sign_min␣=␣%f\n"
 , min_energy/Fs, lag_min, sign_min);
         printf("lags␣=␣%d,␣cum_energy␣=␣%f,␣sign␣is␣%.0f\n", lag_index,
 cum_energy/Fs, copysign(1.0, lag_filter[lag_index]));
      }

      cum_energy = 0;

      if(action == 0){
```

```
      lag_filter[lag_index] *= -1;
      action = 1;
   }
   else if(action == 1){
      lag_filter[lag_index++] = 0;
      lag_filter[lag_index] = scale_factor;
      action = 0;
   }
}

y = 0;

for(int k = 199; k > 0; k--){
   lag_buffer[k] = lag_buffer[k-1];
   y += lag_buffer[k] * lag_filter[k];
}

lag_buffer[0] = random_val;
y += lag_buffer[0] * lag_filter[0];

if(errs != 0)
   break;

error[i] = (double)((s16)(_buffer[0]&0x0000FFFF)/32768.0);
cum_energy += error[i]*error[i];

out[0] = (u32)(random_val*20000) & 0x0000FFFF;
out[15] = (u32)((y*20000)) & 0x0000FFFF;

speaker_output[i] = (double)random_val;
adaptive_output[i] = y;
```

```
        ao16_write(fd, out, 4*16);

        adaptive_input[i] = random_val;
        i++;
    }

    printf("min_lag = %d, min_energy = %f\n", lag_min, min_energy/Fs);

    FILE* file1;

    file1 = fopen("DATA PATH", "wb");
    fwrite(error, sizeof(double), num_samples, file1);
    fclose(file1);

    file1 = fopen("DATA PATH", "wb");
    fwrite(lag_filter, sizeof(double), 200, file1);
    fclose(file1);

    file1 = fopen("DATA PATH", "wb");
    fwrite(adaptive_output, sizeof(double), num_samples, file1);
    fclose(file1);

    file1 = fopen("DATA PATH", "wb");
    fwrite(adaptive_input, sizeof(double), num_samples, file1);
    fclose(file1);

    file1 = fopen("DATA PATH", "wb");
    fwrite(speaker_output, sizeof(double), num_samples, file1);
    fclose(file1);

    return(errs);
}
```