International Journal of Research and Review
Volume 11; Issue: 11; November 2024
Website: www.ijrrjournal.com
Research Paper *E-ISSN: 2349-9788; P-ISSN: 2454-2237*

Design and Simulation of Adaptive Filters in Real-Time Environment

Joshua Okoekhian[1], Mathurine Guiawa[2], Ikenna Onyegbadue[3]

[1,2,3]Electrical/Electronic Engineering Department
[1,2,3]College of Engineering, Igbinedion University Okada, Edo State, Nigeria

Corresponding Author: Joshua Okoekhian

DOI: https://doi.org/10.52403/ijrr.20241118

ABSTRACT

The results of the study showed that adaptive filters were designed and simulated to enhance signal processing capabilities in real-time environments. The study focused on the LMS and NLMS algorithms for echo and noise cancellation in audio signals, exploring the impact of different mu values (0.01, 0.05, 0.1, and 0.5) on algorithm performance. A comprehensive literature review underscored the importance of effective adaptive filtering techniques in real-time audio processing and highlighted the trade-off between convergence speed and stability. The results indicated that a mu value between 0.05 and 0.1 optimally balances convergence speed and stability for both LMS and NLMS algorithms. The NLMS algorithm consistently outperformed LMS due to its superior adaptation to signal power variations and better stability at higher mu values. Recommendations include using the NLMS algorithm for most applications, careful tuning of the mu value, and further research into adaptive and hybrid methods. These findings aimed to enhance echo and noise cancellation, resulting in clearer and more intelligible audio signals in real-world applications.

Keywords: Adaptive Filters, LMS Algorithm, NLMS Algorithm, Real-Time Signal Processing

INTRODUCTION

In today's rapidly evolving digital world, the demand for efficient signal processing techniques in real-time environments has surged across various domains such as telecommunications, audio processing, biomedical engineering, and more (Avalos et al., 2011; Tan & Jiang, 2018; Sahaana, 2022). Adaptive filtering has emerged as a crucial tool in addressing the challenges posed by varying environments, non-stationary signals, and noise interference. Adaptive filters dynamically adjust their parameters based on the input data, allowing them to adapt to changing conditions and enhance the desired signal while attenuating unwanted components (Aboy et al., 2005)

The motivation behind this project stems from the need to further explore and understand the capabilities of adaptive filters in real-time signal processing scenarios. Despite significant advancements in adaptive filtering techniques, there remains a continuous quest for more efficient algorithms and methodologies to address specific application requirements (Ahirwal et al., 2021; Comminiello et al., 2022). By delving into the design and simulation of adaptive filters, this study seeks to contribute to the on-going efforts aimed at improving signal processing capabilities in real-time environments.

Several studies have highlighted the importance of adaptive filtering in various applications. For instance, in telecommunications, adaptive filters are

employed for echo cancellation, channel equalization, and interference suppression (Haykin, 2017; Diniz, 2019). Similarly, in biomedical signal processing, adaptive filters play a crucial role in de-noising electrocardiogram (ECG) signals and removing artefacts (Moody et al., 2001; Rupp et al., 2016; Qin et al., 2024). The versatility of adaptive filters makes them indispensable tools in diverse fields, driving the need for continued research and development.

This research aimed to design and simulate adaptive filters to enhance signal processing capabilities in real-time environments.

LITERATURE REVIEW

Adaptive filters are a class of signal processing filters that automatically adjust their parameters to optimize their performance in response to changing input signals or operating conditions. Unlike traditional fixed filters, which have predetermined coefficients, adaptive filters continuously update their coefficients based on the input data and a specified optimization criterion. This adaptability enables adaptive filters to effectively mitigate noise, suppress interference, and enhance the desired signals in dynamic and non-stationary environments (Cowan, 1996; Apolinário & Netto, 2008 ;).

At the core of adaptive filtering is the concept of error minimization through iterative adjustment of filter coefficients. The adaptation process typically involves the comparison of the filter output with a reference signal or desired response, known as the error signal. By minimizing this error signal over time, adaptive filters converge towards optimal coefficient values that best approximate the desired signal while minimizing distortion and interference (Tan & Jiang, 2018; Diniz, 2019).

Various adaptive filter algorithms exist, each offering distinct advantages and trade-offs depending on the application requirements and computational resources available. Some commonly used adaptive filter algorithms include:

1. Least Mean Squares (LMS) Algorithm: The LMS algorithm is a widely employed adaptive filtering algorithm due to its simplicity and computational efficiency. It updates filter coefficients in the direction of the gradient of the mean square error, making it suitable for applications with large datasets and real-time processing requirements (Hayes, 1996; Haykin & Widrow, 2003; Liu, et al., 2011; Diniz, 2019).
2. Recursive Least Squares (RLS) Algorithm: The RLS algorithm recursively computes the filter coefficients by minimizing the sum of squared errors over a finite data window. RLS offers faster convergence rates and improved tracking performance compared to LMS but may be computationally more demanding, particularly for large filter lengths (Albu et al., 2001; Ifeachor & Jervis, 2002; Xu et al., 2002; Van Vaerenbergh et al., 2012).
3. Normalized LMS Algorithm: The normalized LMS algorithm enhances the stability and convergence properties of the LMS algorithm by normalizing the step size based on the input signal power. This adaptation mechanism enables the algorithm to maintain stable operation across varying signal power levels (Bershad & Feintuch, 1986; Hayes, 1996; Haykin & Widrow, 2003; Liu, et al., 2011; Diniz, 2019).
4. Kalman Filter: The Kalman filter is an optimal recursive algorithm used for estimation in dynamic systems subject to Gaussian noise. It combines information from measurements and a dynamic model to estimate the state of a system while minimizing the mean square error (Zarchan & Musoff, 2000; Stepanov, 2011; Fauzi & Batool, 2019; Lora-Millan et al., 2021; Kalita & Lyakhov, 2022).
5. Adaptive Noise Cancelling (ANC) Algorithms: ANC algorithms aim to suppress unwanted noise or interference by adaptively adjusting a filter to match

the characteristics of the noise. These algorithms are particularly useful in applications such as echo cancellation, active noise control, and interference suppression.

Adaptive filters offer a powerful and versatile approach to signal processing in dynamic and noisy environments. By continuously adjusting their parameters based on input data, adaptive filters can adapt to changing conditions and effectively enhance signal quality in real-time applications.

Various Adaptive Filter Algorithms

Least Mean Squares (LMS) Algorithm

The Least Mean Squares (LMS) algorithm is one of the most widely used adaptive filter algorithms due to its simplicity and computational efficiency (Haykin, 2017). In the LMS algorithm, filter coefficients are iteratively adjusted in the direction of the negative gradient of the mean square error with respect to the filter weights. Mathematically, the weight update equation for the LMS algorithm can be expressed as:

$$\boldsymbol{h}(n+1) = \boldsymbol{h}(n) + \mu e(n)\boldsymbol{x}(n)$$

Where:

$\boldsymbol{h}(n)$ is the vector of filter coefficients at iteration n

μ is the adaptation step size (also known as the learning rate)

$e(n)$ is the error signal at iteration n, $\boldsymbol{x}(n)$ is the input signal vector at iteration n.

The LMS algorithm is particularly suitable for applications with large datasets and real-time processing requirements, as it updates filter coefficients incrementally based on individual data samples.

Normalized LMS Algorithm

The Normalized Least Mean Squares (LMS) algorithm is a variation of the traditional LMS algorithm that improves its stability and convergence characteristics, particularly in scenarios where the input signal power varies significantly (Haykin, 2017). In the Normalized LMS algorithm, the adaptation step size is adjusted based on the power of the input signal, ensuring that the algorithm remains stable even when the input signals amplitude changes. The weight update equation for the Normalized LMS algorithm can be expressed as:

$$\boldsymbol{h}(n+1) = \boldsymbol{h}(n) + \frac{\mu}{\|\boldsymbol{x}(n)\|^2 + \epsilon} e(n)\boldsymbol{x}(n)$$

Where:

$\boldsymbol{h}(n)$ is the vector of filter coefficients at iteration n,

μ is the adaptation step size (learning rate),

$e(n)$ is the error signal at iteration n,

$\boldsymbol{x}(n)$ is the input signal vector at iteration n

$\|\boldsymbol{x}(n)\|^2$ is the squared magnitude of the input signal vector,

ϵ is a small positive constant (typically used to avoid division by zero).

The Normalized LMS algorithm ensures that the algorithm's behaviour is consistent across different signal power levels. This improves the algorithm's stability and convergence properties, making it suitable for a wide range of real-time signal processing applications. The Normalized LMS algorithm is particularly useful in scenarios where the input signal amplitude varies significantly, such as in adaptive equalization and echo cancellation applications.

Recursive Least Squares (RLS) Algorithm

The Recursive Least Squares (RLS) algorithm is another widely used adaptive filter algorithm that offers faster convergence rates and improved tracking performance compared to the LMS algorithm (Haykin, 2017). In the RLS algorithm, filter coefficients are recursively updated using an exponentially weighted least squares estimation approach. The weight update equation for the RLS algorithm can be expressed as:

$$\boldsymbol{h}(n+1) = \boldsymbol{h}(n) + \frac{1}{\lambda}\boldsymbol{P}(n)\boldsymbol{x}(n)e(n)$$

Where:

λ is the forgetting factor $0 < \lambda \leq 1$,

$\boldsymbol{P}(n)$ is the inverse of the autocorrelation matrix at iteration n.

The RLS algorithm is particularly effective in scenarios where the input signal statistics

change over time or when a high tracking speed is required. However, it may be computationally more demanding, especially for large filter lengths, due to the need to invert the autocorrelation matrix.

These adaptive filter algorithms, along with others such as the Normalized LMS algorithm and Kalman filter algorithms, offer a range of options for adaptively filtering signals in real-time environments, each with its own advantages and limitations.

Kalman Filter

The Kalman filter is an optimal recursive algorithm used for estimation in dynamic systems subject to Gaussian noise (Haykin, 2017). It combines information from measurements and a dynamic model to estimate the state of a system while minimizing the mean square error. The Kalman filter operates in two main stages: prediction and update. The prediction equation is given by:

$$\hat{\mathbf{x}}(n|n-1) = \boldsymbol{F}(n)\hat{\mathbf{x}}(n-1|n-1) + \boldsymbol{B}(n)\mathbf{u}(n)$$

Where:

$\hat{\mathbf{x}}(n|n-1)$ is the predicted state estimate at time n,

$\boldsymbol{F}(n)$ is the state transition matrix at time n,

$\hat{\mathbf{x}}(n-1|n-1)$ is the previous state estimate,

$\boldsymbol{B}(n)$ is the control input matrix at time n,

$\mathbf{u}(n)$ is the control input vector at time n.

The update equation is given by:

$$\hat{\mathbf{x}}(n|n) = \hat{\mathbf{x}}(n|n-1) + \boldsymbol{K}(n)[\boldsymbol{z}(n) - \boldsymbol{H}(n)\hat{\mathbf{x}}(n|n-1)]$$

$$\boldsymbol{P}(n|n) = [\mathbf{I} - \mathbf{K}(n)\boldsymbol{H}(n)]\boldsymbol{P}(n|n-1)$$

Where:

$\hat{\mathbf{x}}(n|n)$ is the updated state estimate at time n,

$\boldsymbol{K}(n)$ is the Kalman gain matrix at time n,

$\boldsymbol{z}(n)$ is the measurement vector at time n,

$\boldsymbol{H}(n)$ is the measurement matrix at time n,

$\boldsymbol{P}(n|n)$ is the updated error covariance matrix at time n,

I is the identity matrix.

The Kalman filter is widely used in various applications, including navigation, control systems, and tracking, where accurate estimation of system states is crucial.

Applications of Adaptive Filters

Adaptive filters find diverse applications across various fields due to their ability to dynamically adjust to changing signal conditions and effectively enhance signal processing tasks. Some common applications of adaptive filters include:

1. **Echo Cancellation:** In telecommunications and audio processing systems, echo cancellation is crucial for removing echoes caused by signal reflections. Adaptive filters are particularly effective as they can dynamically estimate and subtract the echo component from the received signal. This results in significantly improved voice quality and intelligibility during communication, making conversations clearer and more pleasant for users (Messini & Djendi, 2019).
2. **Active Noise Control (ANC):** Adaptive filters are extensively used in ANC systems to reduce or eliminate unwanted noise from audio signals. These filters adjust their coefficients adaptively based on a reference signal containing the noise component. By doing so, ANC systems can effectively cancel out noise, creating a quieter and more comfortable environment. This technology is widely applied in various settings, including headphones, car cabins, and industrial machinery, enhancing user experience and operational efficiency (Lu et al., 2021).
3. **Biomedical Signal Processing:** In biomedical engineering, adaptive filters play a pivotal role in de-noising and artifact removal from physiological signals such as electrocardiograms (ECG), electroencephalograms (EEG), and electromyograms (EMG). By adaptively filtering out noise and interference, these filters enhance the

accuracy of diagnostic information extracted from biomedical signals, thereby improving patient monitoring and diagnosis. This leads to better healthcare outcomes and more precise medical interventions (Bruce, 2001; Devasahayam, 2019).

4. **Channel Equalization:** In digital communication systems, adaptive filters are crucial for channel equalization, compensating for channel distortions and inter-symbol interference (ISI). By continuously adjusting the filter coefficients based on the received signal and channel characteristics, adaptive equalizers effectively mitigate the effects of channel distortion. This capability ensures reliable communication in various applications, including wireless communication systems, digital modems, and broadband communication networks (Huq et al., 2009).
5. **System Identification:** Adaptive filters are instrumental in system identification tasks, where the objective is to estimate the impulse response or transfer function of a dynamic system. By applying adaptive filtering techniques to input-output data from the system, these filters can accurately model and identify the system dynamics. This capability is essential for system analysis, control, and optimization across various engineering applications, providing valuable insights into system behavior and facilitating more effective system management (Ljung, 2015).

MATERIALS AND METHOD

The development of a real-time simulation environment for adaptive filtering research requires careful consideration of software and hardware requirements to ensure optimal performance and functionality. The following outlines the key software and hardware components needed for setting up the simulation environment:

Software Requirements:

1. Simulation Software: Utilize simulation software such as MATLAB/Simulink, a specialized simulation tools tailored for real-time signal processing.
2. Programming Environments: Install programming environments and development tools for algorithm implementation, simulation scripting, and data analysis. MATLAB, a specialized DSP development environment.
3. Signal Processing Libraries: Incorporate signal processing libraries and toolboxes for implementing adaptive filtering algorithms, signal generation, noise modelling, and system simulation. Libraries such as DSP System Toolbox in MATLAB provide comprehensive support for signal processing tasks.
4. Simulation Models: Develop simulation models and algorithms for adaptive filtering, system modelling, signal generation, and noise addition. Use simulation software or programming environments to implement and validate the models.
5. Visualization Tools: Integrate visualization tools and plotting libraries for real-time monitoring, data visualization, and performance analysis. Tools as MATLAB plots, a specialized visualization software facilitate data visualization and interpretation.

Hardware Requirements:

1. Computing Platform: Choose a computing platform capable of running simulation software and performing real-time signal processing tasks. Options include desktop computers, workstations, embedded systems, or specialized hardware platforms with sufficient computational resources.
2. Processor and Memory: Selecting a processor with adequate computational power and memory capacity to handle real-time signal processing tasks. Multi-core processors, high-speed processors,

and sufficient RAM are essential for efficient simulation execution.

Simulation Setup and Configuration

Once the software and hardware requirements are identified, the next step is to set up and configure the simulation environment for conducting real-time adaptive filtering experiments. The following steps outline the simulation setup and configuration process:

1. **Environment Configuration:** Install and configure the simulation software, programming environments, and required libraries on the computing platform. Ensure compatibility and proper integration of software components for seamless operation.
2. **Algorithm Implementation:** Implement adaptive filtering algorithms, signal processing routines, and system models using the chosen programming environment. Develop modular and efficient code for algorithm execution, data processing, and visualization.
3. **Simulation Setup:** Define simulation parameters, such as sampling rate, signal duration, noise characteristics, and system dynamics. Configure simulation models, algorithm parameters, and input/output interfaces according to the research objectives and experimental design.
4. **Real-Time Execution:** Execute the simulation in real-time mode to ensure timely processing of signals and interactions with external devices. Monitor system performance, CPU usage, and memory usage to optimize algorithm execution and prevent runtime errors.

Implementation of Adaptive Filter Algorithms

The implementation of adaptive filter algorithms involves translating the mathematical formulations of the algorithms into executable code for simulation and real-time processing. This process requires careful consideration of algorithmic details, programming techniques, and optimization strategies to ensure efficient and accurate execution. The following steps outline the implementation process for adaptive filter algorithms:

1. **Algorithm Selection:** Choose the appropriate adaptive filter algorithms based on the research objectives, application requirements, and performance considerations. Common adaptive filter algorithms include the Least Mean Squares (LMS), and Normalized LMS,
2. **Algorithm Formulation:** Formulate the selected adaptive filter algorithms in terms of mathematical equations and update rules. Understand the underlying principles, convergence properties, and parameter dependencies of the algorithms to guide the implementation process effectively.
3. **Programming Environment:** Select a suitable programming environment and language for algorithm implementation. MATLAB, a specialized DSP development environment would suffice.
4. **Code Development:** Develop modular and efficient code to implement the adaptive filter algorithms. Break down the algorithm into manageable components, such as initialization, coefficient updates, error calculation, and convergence monitoring, to facilitate code organization and readability.
5. **Parameter Initialization:** Initialize the algorithm parameters, such as filter coefficients, step size (learning rate), and other control parameters. Proper initialization ensures stability, convergence, and optimal performance of the adaptive filter algorithms.
6. **Signal Processing Operations:** Implement signal processing operations, such as signal acquisition, pre-processing, filtering, and post-processing, as needed for the specific application. Incorporate signal generation, noise modelling, and data

manipulation routines to simulate realistic signal scenarios and experimental conditions.

7. **Simulation and Testing:** Validate the implemented adaptive filter algorithms through simulation and testing. Use synthetic data generated from simulation models or real-world data collected from experiments to assess algorithm performance, convergence behaviour, and robustness under various conditions.

Performance Metrics and Evaluation Criteria

Performance metrics and evaluation criteria play a crucial role in assessing the effectiveness, efficiency, and robustness of adaptive filter algorithms in signal processing applications. These metrics provide quantitative measures of algorithm performance and enable comparison between different algorithms, parameter settings, and system configurations. The selection of appropriate performance metrics depends on the specific research objectives, application requirements, and desired outcomes. Some common performance metrics and evaluation criteria for adaptive filter algorithms include:

1. **Mean Square Error (MSE):** MSE is a measure of the average squared difference between the estimated output of the adaptive filter and the desired output. It quantifies the accuracy of the filter in approximating the desired signal and is commonly used as a primary performance metric in adaptive filtering.
2. **Convergence Speed:** Convergence speed refers to the rate at which the adaptive filter algorithm converges to a stable solution. It measures how quickly the filter adapts to changes in the input signals and reaches a steady-state condition. Faster convergence speed is desirable for real-time applications with dynamic signal conditions.
3. **Computational Complexity:** Computational complexity assesses the computational resources required to execute the adaptive filter algorithm. It includes measures such as the number of multiplications, additions, memory accesses, and processing time needed per iteration. Lower computational complexity indicates more efficient algorithm implementation.
4. **Simulation and Experimental Validation:** Simulation and experimental validation assess the performance of the adaptive filter algorithm using synthetic data generated from simulation models or real-world data collected from experiments. Validation ensures that the algorithm meets the desired performance criteria and accurately captures the underlying signal characteristics and system behaviour.

Design Equations

Least Mean Squares (LMS) algorithm

The Least Mean Squares (LMS) algorithm is a widely used adaptive filter algorithm in the field of signal processing and machine learning. It is often employed for tasks like noise cancellation, system identification, and equalization. The derivation of the LMS algorithm involves minimizing the mean squared error between the desired response and the estimated response of a system. Here's a derivation of the LMS algorithm from first principles:

Considering a discrete-time linear system described by the following equation:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) + v(n)$$

Where:

$y(n)$ is the output of the system at time n;

$x(n)$ is the input to the system at time n;

$h(k)$ are the unknown system coefficients to be estimated?

$v(n)$ is the additive noise.

The goal is to estimate the unknown system coefficients $h(k)$ using the Least Mean Squares (LMS) algorithm.

The error at time n is defined as:

$$e(n) = d(n) - y(n)$$

Where:

$d(n)$ is the desired response or the target output.

The mean squared error (MSE) is defined as:

$$\boldsymbol{J} = \frac{1}{2}\mathbb{E}[e^2(n)]$$

Where

$\mathbb{E}[\cdot]$ denotes the expectation operator.

To minimize the mean squared error, we need to update the filter coefficients $h(k)$ in such a way that the error is minimized. The LMS algorithm uses a stochastic gradient descent approach to update the filter coefficients.

The update rule for the LMS algorithm is given by:

$$\Delta h(k) = \mu e(n)x(n-k)$$

Where:

μ is the step size or the learning rate.

Using the update rule, we can update the filter coefficients as follows:

$$h(k+1) = h(k) + \Delta h(k)$$

Substituting the expression for $\Delta h(k)$, we get:

$$h(k+1) = h(k) + \mu e(n)x(n-k)$$

To derive the LMS algorithm, we need to substitute the expression for the error (n) into the update rule. This gives:

$$h(k+1) = h(k) + \mu[d(n) - y(n)]x(n-k)$$

$$h(k+1) = h(k) + \mu\left[d(n) - \sum_{i=0}^{M-1} h(i)x(n-i)\right]x(n-k)$$

This is the update rule for the LMS algorithm. It updates the filter coefficients iteratively based on the error between the desired response and the estimated response of the system. The LMS algorithm can be implemented in real-time and is computationally efficient, making it suitable for online learning applications. It converges to the optimal solution in the mean square sense under certain conditions on the input signals and the step size.

Normalized Least Mean Squares (NLMS) algorithm

NLMS is a variation of the LMS algorithm that normalizes the step size by the square of the input signal. This normalization ensures that the step size adapts to the variations in the input signal, leading to better convergence properties, especially when dealing with signals of varying magnitudes.

Starting from the same discrete-time linear system equation:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) + v(n)$$

Where:

$y(n)$ is the output of the system at time n;

$x(n)$ is the input to the system at time n;

$h(k)$ are the unknown system coefficients to be estimated;

$v(n)$ is the additive noise.

The error at time n is defined as:

$$e(n) = d(n) - y(n)$$

Where:

$d(n)$ is the desired response or the target output.

The NLMS algorithm updates the filter coefficients as follows:

$$h(k+1) = h(k) + \frac{\mu}{\|x(n)\|^2}e(n)x(n-k)$$

Where:

μ is the step size or the learning rate;

$\|x(n)\|^2$ is the squared norm of the input signal at time n.

The squared norm of the input signal is given by:

$$\|x(n)\|^2 = \sum_{i=0}^{M-1} |x(n-i)|^2$$

Substituting this expression into the NLMS update rule:

$$h(k+1) = h(k) + \frac{\mu}{\|x(n)\|^2}e(n)x(n-k)$$

$$h(k+1) = h(k) + \frac{\mu}{\sum_{i=0}^{M-1}|x(n-i)|^2} e(n)x(n-k)$$

This is the update rule for the NLMS algorithm. It updates the filter coefficients based on the error between the desired response and the estimated response of the system, normalized by the squared norm of the input signal.

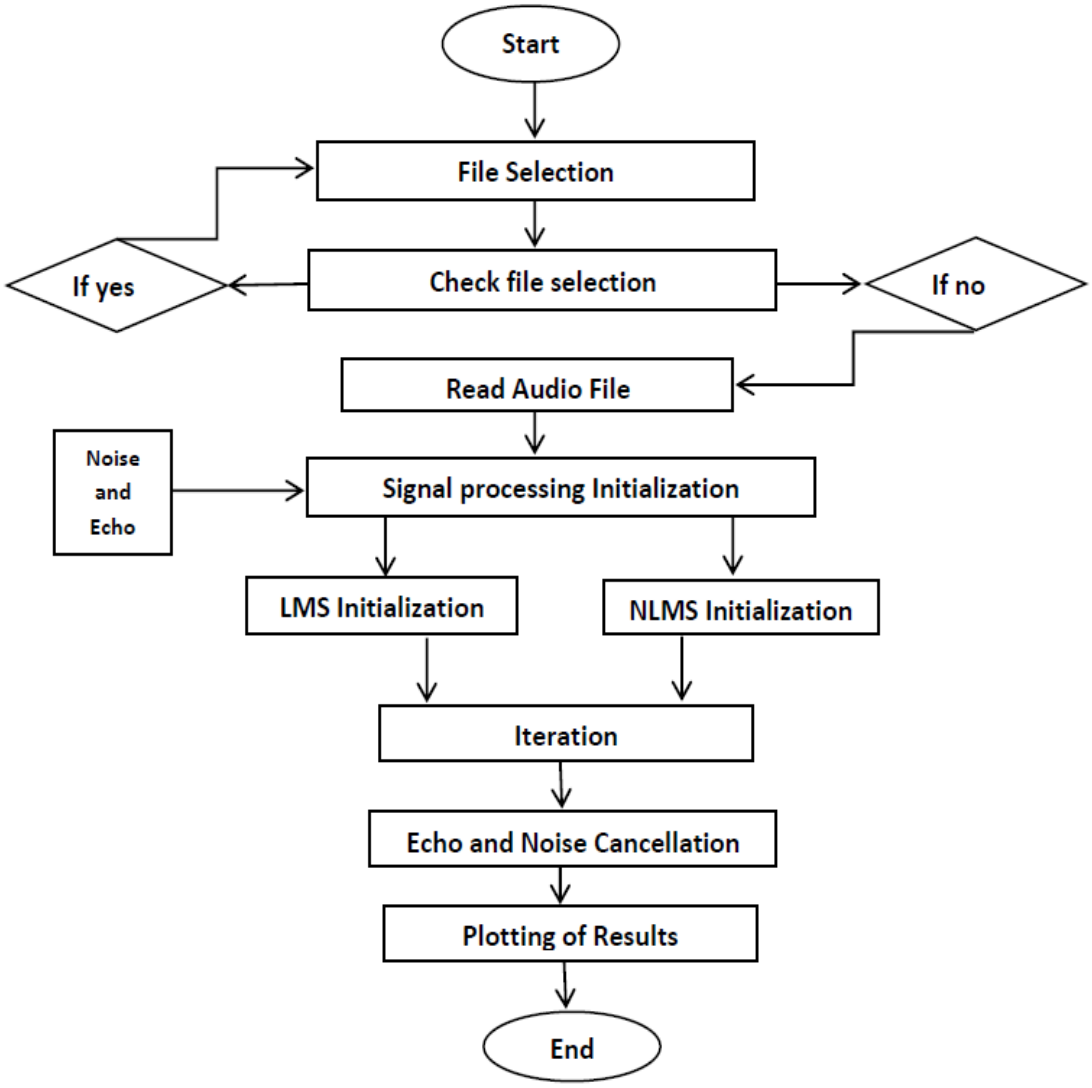

Figure 1: Simulation Flowchart

RESULTS AND DISCUSSION

Simulation input parameters

The following input parameters for the simulation were used in MATLAB, and were presented in table 1 along with a detailed explanation of each parameter:

Table 1: Simulation input parameters value description and explanation

Parameter	Value/Description	Explanation
audioFilePath	Fullfile (path, file)	The full path to the selected audio file.
Fs	Sample rate of the audio file (e.g., 44100 Hz)	Sampling frequency of the audio signal, which determines how often the audio signal is sampled.
N	Length of the audio signal d	Total number of samples in the audio signal.

T	(0:N-1) / Fs	Time vector corresponding to each sample in the audio signal.
X	d + randn(N, 1) * 0.5	Audio signal with added Gaussian noise (standard deviation of 0.5).
echo_delay	round(0.1 * Fs)	Delay for the echo effect, set to 0.1 seconds.
echo_gain	0.5	Gain of the echo signal, set to 0.5.
echo_signal	zeros(echo_delay, 1); x(1:end - echo_delay) * echo_gain	Simulated echo signal with specified delay and gain.
Noise	randn(N, 1) * 0.2	Noise added to the mixed signal (standard deviation of 0.2).
mixed_signal	x + echo_signal + noise	Combined signal consisting of the original signal, echo, and noise.
mu_values	[0.01, 0.05, 0.1, 0.5]	Array of step sizes for the LMS and NLMS algorithms.
time_start	0.55	Start time for plotting the signals.
time_end	0.57	End time for plotting the signals.
time_indices	find(t >= time_start & t <= time_end)	Indices of the time vector corresponding to the plotting range.

Detailed Explanation of Each Parameter:

1. Audio File Path: This is the complete path to the audio file chosen by the user. It combines the path and filename to access the file for reading.
2. Fs: The sample rate of the audio file, given in Hertz (Hz). It represents how frequently samples of the audio signal are taken per second. Common values are 44100 Hz for CD-quality audio.
3. N: The number of samples in the audio signal, calculated by finding the length of the audio data array d. It gives the total number of data points in the signal.
4. t: A time vector created by dividing the sample indices by the sample rate Fs. This vector represents the time (in seconds) corresponding to each sample.
5. x: The original audio signal d with added noise. The noise is generated from a Gaussian distribution with a standard deviation of 0.5, which simulates real-world noise in the audio signal.
6. echo_delay: The number of samples that represents the delay for the echo effect, set to 0.1 seconds. This creates a time shift in the audio signal to simulate an echo.
7. echo gain: The amplification factor applied to the echo signal. In this case, it is set to 0.5, meaning the echo signal is half as strong as the original signal.
8. echo signal: The generated echo signal, which is created by shifting the original signal by the echo_delay and scaling it by the echo_gain.
9. noise: Additional noise added to the signal to simulate real-world conditions where signals are often mixed with background noise.
10. mixed_signal: The final signal that combines the original signal, the echo effect, and the additional noise.
11. mu_values: The learning rates used in the LMS (Least Mean Squares) and NLMS (Normalized Least Mean Squares) algorithms. These values control how quickly the adaptive filters adjust their weights during the learning process.
12. time_start: The beginning of the time interval used for plotting. It specifies where the plotting of signals should start in seconds.
13. time_end: The end of the time interval for plotting. It defines where the plotting of signals should end in seconds.
14. time_indices: The indices of the time vector t that fall within the specified time range [time_start, time_end]. These indices are used to extract and plot the relevant portion of the signals.

These parameters and their values define how the audio signal is processed, simulated, and analyzed in the provided code.

Results
Explanation of Each Plot:

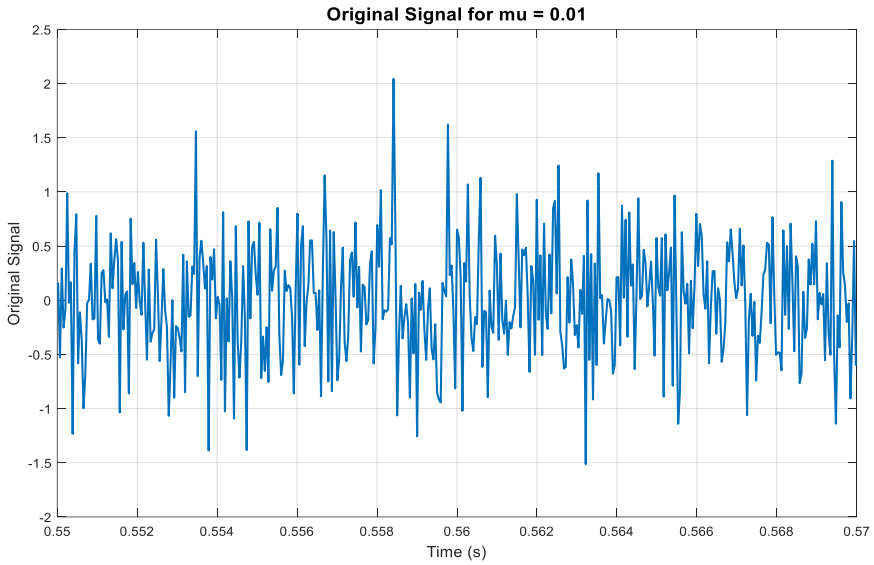

Figure 2: Original signal plot

1. **Original Signal:** This plot shows the audio signal with Gaussian noise added, representing the raw, unprocessed audio before any echo or noise cancellation techniques are applied. By visualizing the original signal in its noisy state, this plot provides a reference point to assess how the noise affects the signal's waveform. The Observation of this plot is crucial for understanding the baseline characteristics of the signal and for evaluating the effectiveness of any subsequent signal processing.

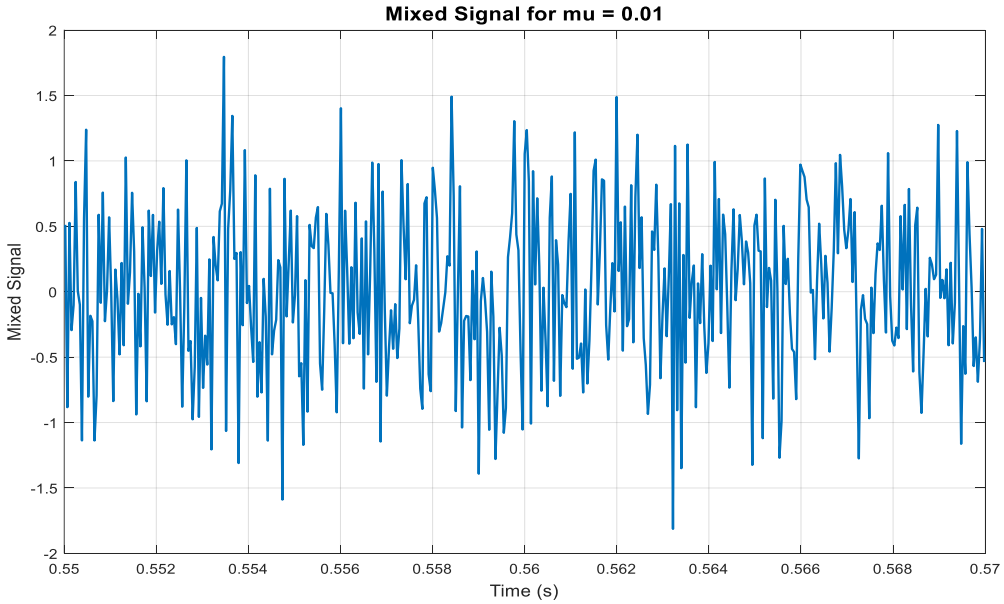

Figure 3: Mixed signal plot

2. **Mixed Signal:** This plot is the combination of the original audio signal with both the simulated echo and additional noise, resulting in a complex composite signal. The visual representation of this mixed signal

illustrates how the echo introduces time-delayed repetitions of the original signal, and how the added noise creates random variations. This visualization helps in understanding the compounded effect of these disturbances on the signal, revealing the challenges faced in isolating and processing the original audio content amidst these artifacts.

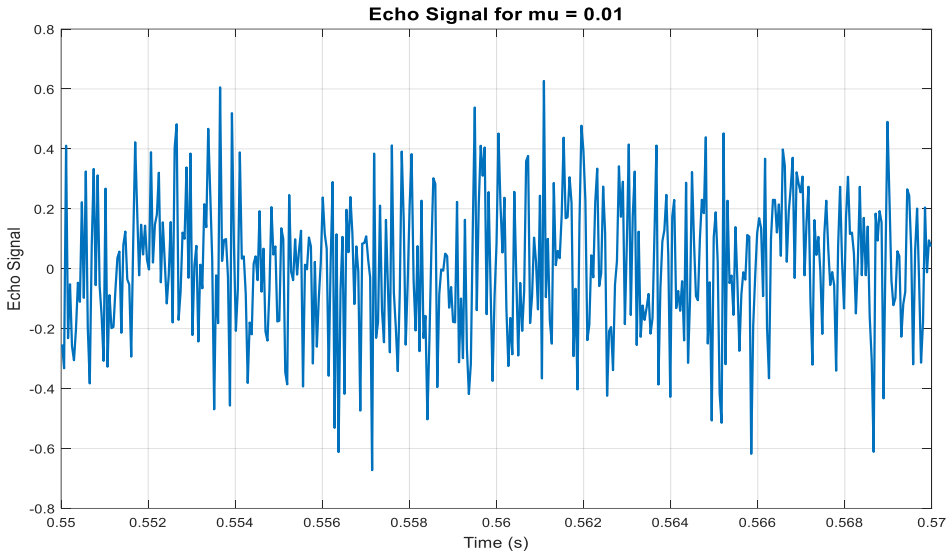

Figure 4: Echo signal plot

3. **Echo Signal:** The echo signal plot isolates the simulated echo component, which is created by delaying the original signal and applying a gain. This plot allows for an examination of how the echo appears as an added, delayed version of the original signal. The focus on the echo is to help analyze its impact on the overall signal, including how the delay and gain parameters affect the echo's prominence and interaction with the original signal.

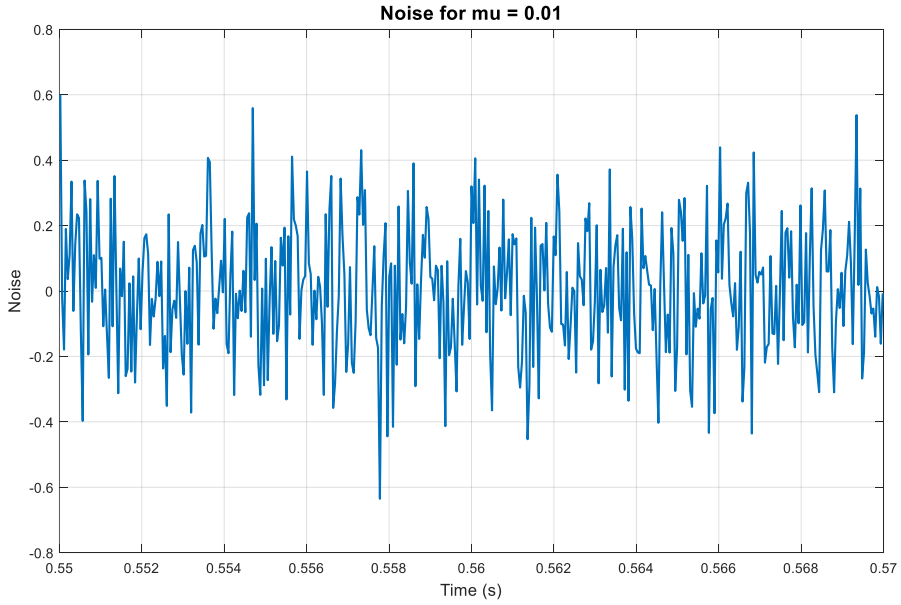

Figure 5: Noise signal plot

4. **Noise:** This plot depicts the additional noise component separately, showing its random and unpredictable nature. This visual presentation of the noise component helps to assess its amplitude and distribution, which also helps in understanding its contribution to the overall mixed signal. The randomness of the noise can obscure the original signal, and this plot provides insight into how the noise level and characteristics influence the clarity and quality of the audio.

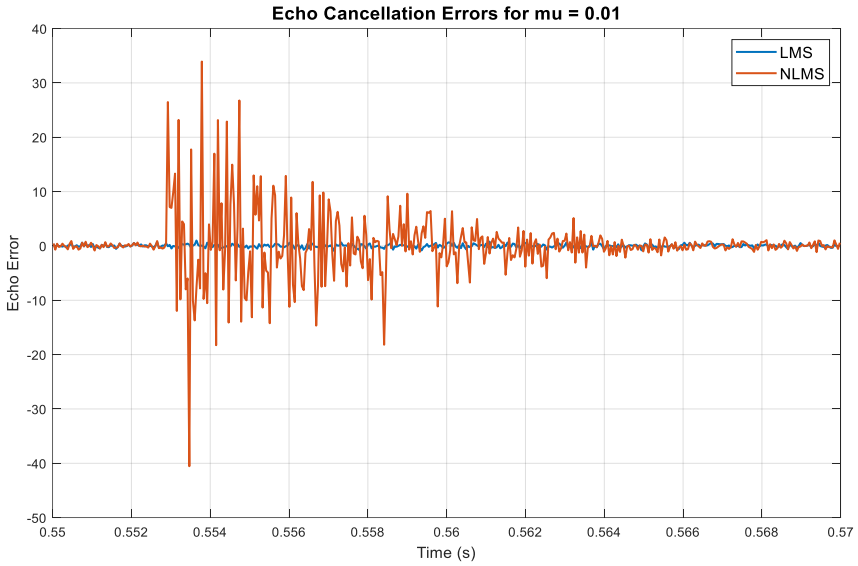

Figure 6: Echo cancellation Error plot

5. **Echo Cancellation Errors:** This plot illustrates the error in echo cancellation performance for both LMS (Least Mean Squares) and NLMS (Normalized Least Mean Squares) algorithms. It shows the difference between the mixed signal and the signal reconstructed after applying each algorithm. Smaller error values indicate that the algorithm has more effectively removed the echo, while larger errors suggest that the algorithm's performance is less effective. Comparing these errors helps in evaluating and contrasting the efficacy of the LMS and NLMS algorithms in echo cancellation.

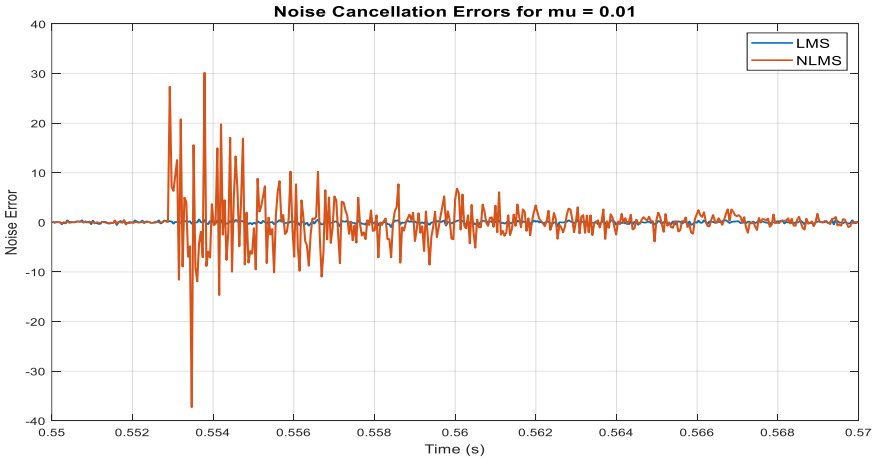

Figure 7: Noise cancellation Error plot

6. **Noise Cancellation Errors:** This plot displays the errors associated with noise cancellation for both LMS and NLMS algorithms. It represents the discrepancy between the noisy mixed signal and the signal obtained after applying each algorithm. Smaller error values signify that the algorithm has successfully reduced the noise, improving the signal's clarity. This plot is essential for comparing how effectively each algorithm handles noise reduction and for determining which approach provides better noise cancellation.

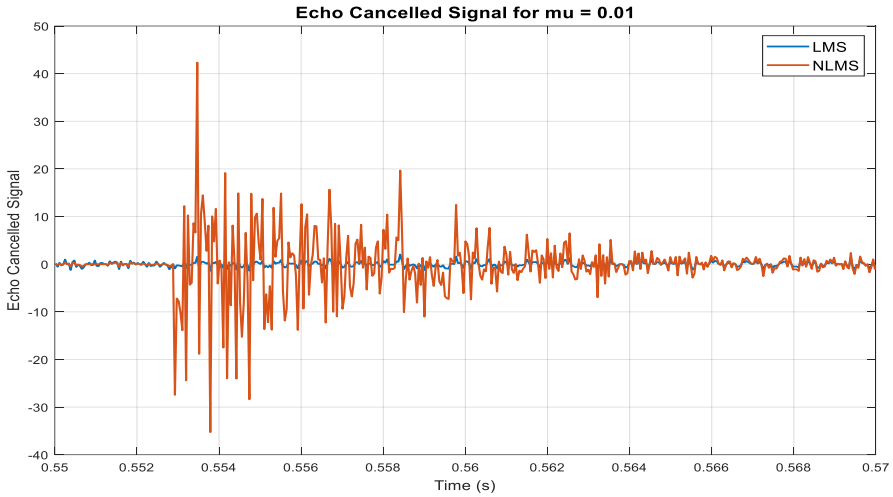

Figure 8: Echo cancelled signal plot

7. **Echo Cancelled Signal:** This plot showcases the signal after applying echo cancellation algorithms, both LMS and NLMS. It visually represents the success of each algorithm in removing the echo component from the mixed signal. The comparison of these plots helps in the assessment of the quality of echo removal and determines how closely the processed signal resembles the original, uncorrupted signal. This evaluation is crucial for understanding the effectiveness of the echo cancellation techniques used.

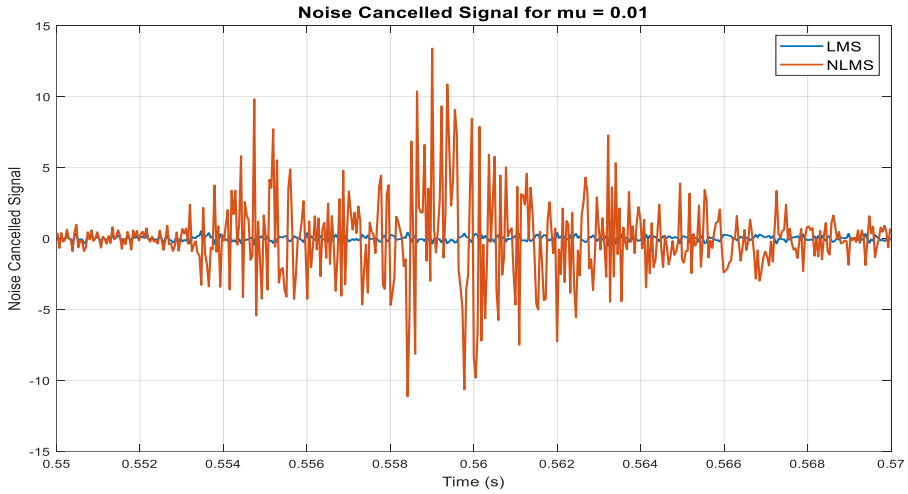

Figure 9: Noise cancelled signal plot

8. **Noise Cancelled Signal:** This plot displays the result of noise cancellation for both LMS and NLMS algorithms, showing the signal after the noise

component has been processed. It illustrates how well each algorithm has succeeded in mitigating noise, improving the signal's overall quality. The close observation of these plots helps in the evaluation of the performance of each algorithm in reducing noise and enhancing the signal's clarity, providing insight into which method offers better noise reduction.

Detailed Result Analysis

The details of the observations for each set of mu values are described below:

Mu = 0.01:

This is the detailed description of each plot for **Mu = 0.01**:

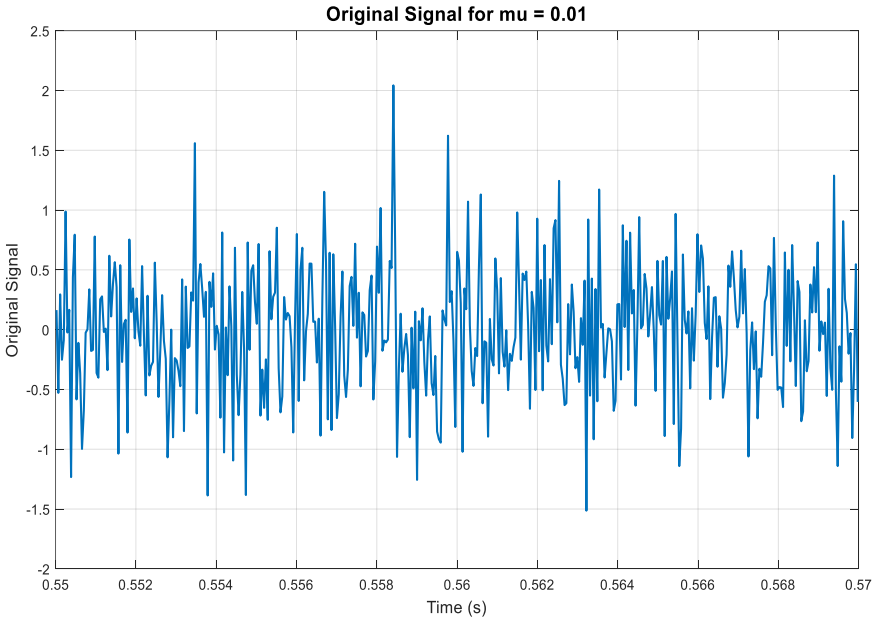

Figure 10: Plot of Original signal for mu = 0.01

1. **Original Signal:** For **Mu = 0.01**, the original signal appears relatively clean with only minor noise added. This plot provides a clear view of the signal's basic characteristics before any echo or noise cancellation is applied, highlighting how the low level of added noise affects the signal.

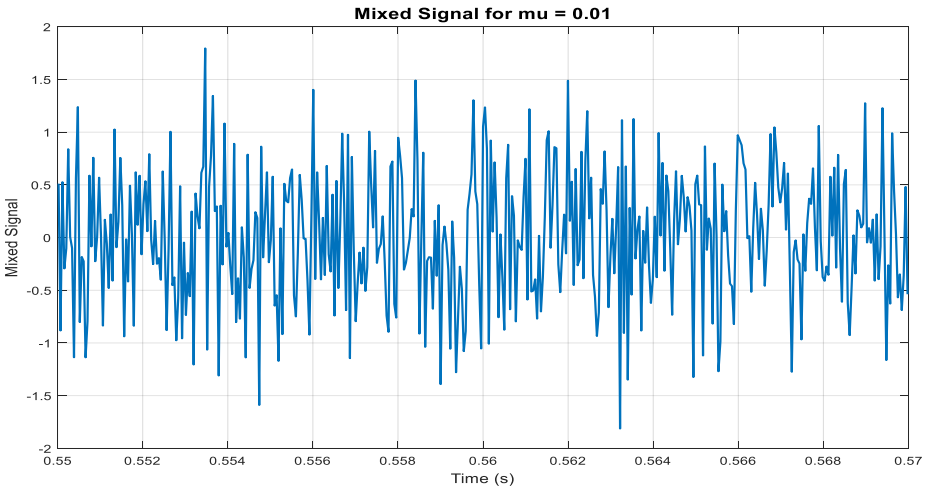

Figure 11: Plot of mixed signal for mu = 0.01

2. **Mixed Signal:** The mixed signal plot reveals increased complexity due to the combined effects of added echo and noise. At this low learning rate, the echo and noise contribute significantly to the signal's distortion, making it more challenging to distinguish the original audio content.

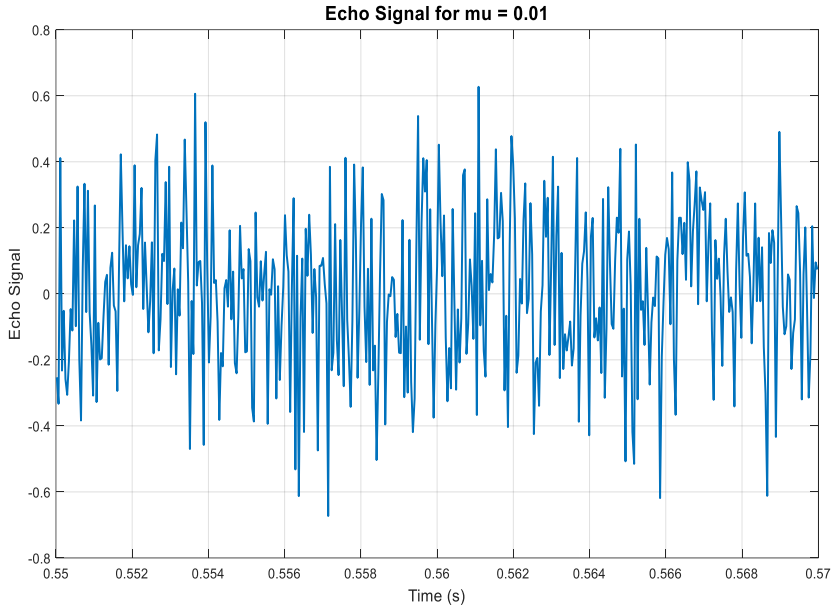

Figure 12: Plot of echo signal for mu = 0.01

3. **Echo Signal:** This plot clearly displays the echo as a delayed version of the original signal. The delay introduced in the echo effect is evident, showing how the signal is repeated with a time shift. This visualization helps in understanding how the echo component overlaps with the original signal.

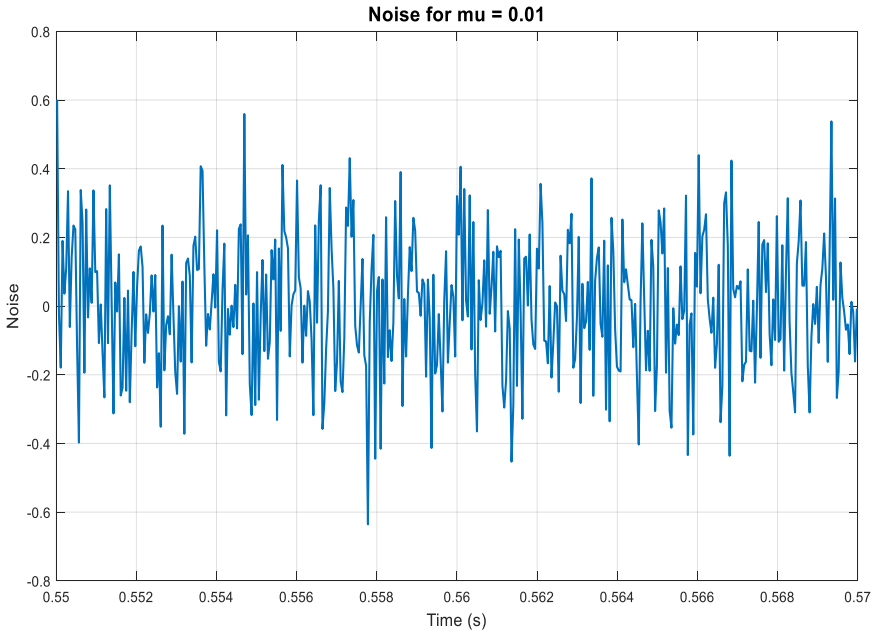

Figure 13: Plot of noise for mu = 0.01

4. **Noise:** The noise plot shows the random noise component separately. It appears as an unpredictable and irregular signal that adds variability to the audio. This randomness in the noise illustrates its impact on the overall mixed signal.

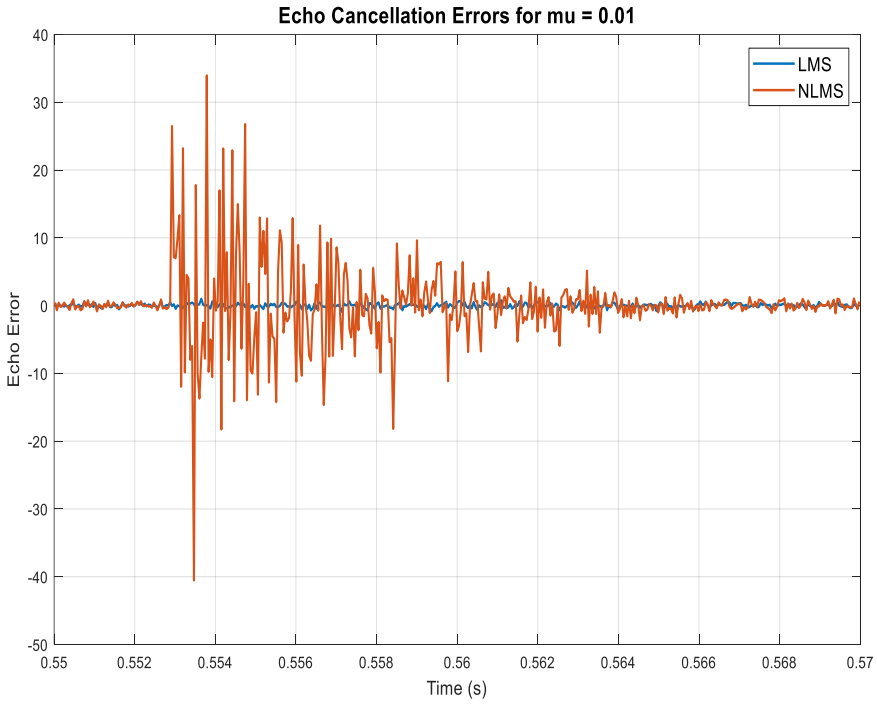

Figure 14: plot of echo cancellation errors for mu = 0.01

5. **Echo Cancellation Errors (LMS and NLMS):** For **Mu = 0.01**, the echo cancellation errors decrease gradually over time for both LMS and NLMS algorithms, indicating a slow adaptation process. The NLMS algorithm generally exhibits a slightly faster decrease in errors due to its normalization process, which adjusts more dynamically to the signal's variations.

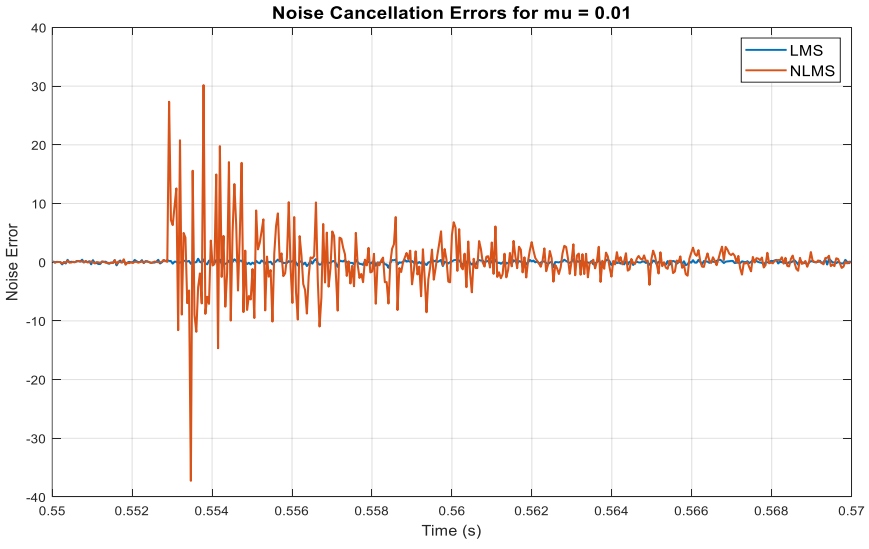

Figure 15: plot of noise cancellation errors for mu = 0.01

6. **Noise Cancellation Errors (LMS and NLMS):** Similar to the echo cancellation errors, the noise cancellation errors also decrease slowly. This gradual reduction shows the slow convergence of the algorithms in adapting to the noise. Both LMS and NLMS algorithms demonstrate this gradual improvement, with NLMS typically performing a bit better due to its normalization approach.

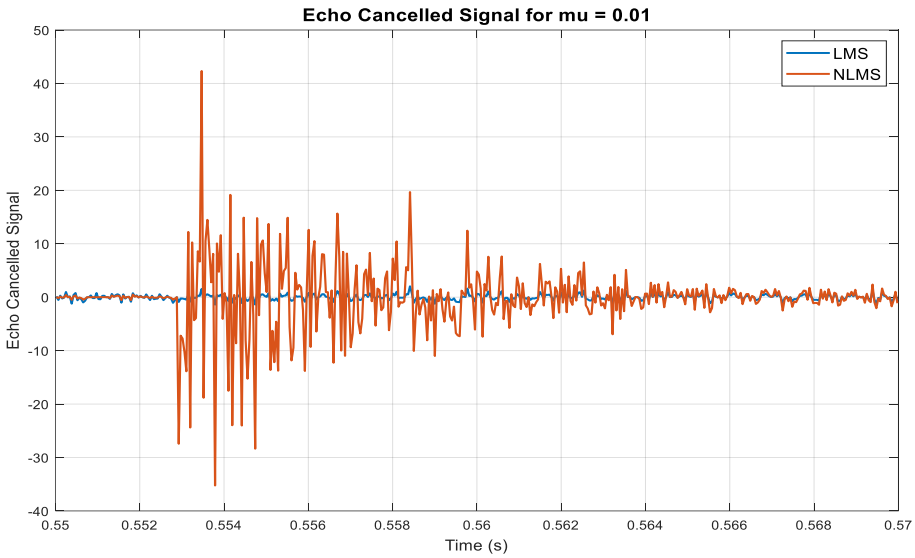

Figure 16: plot of echo cancelled signal for mu = 0.01

7. **Echo Cancelled Signal (LMS and NLMS):** The plot of the echo-cancelled signal reveals that the echo is only partially removed by the algorithms, leaving some remnants visible. The low learning rate (Mu = 0.01) results in slow convergence, meaning that the algorithms are still in the process of adjusting to fully cancel out the echo.

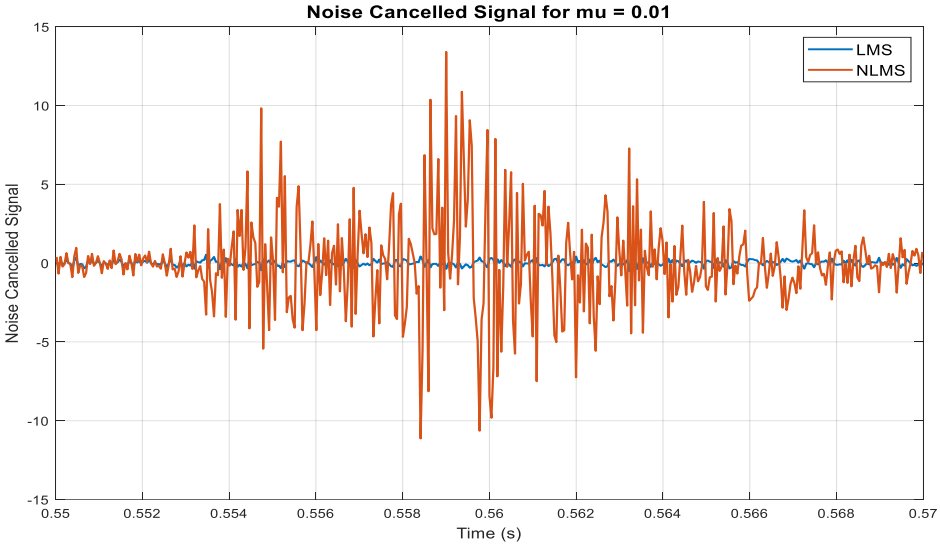

Figure 17: Plot of noise cancelled signal for mu = 0.01

8. **Noise Cancelled Signal (LMS and NLMS):** Similarly, the noise-cancelled signal plot shows partial noise reduction. The remnants of noise indicate that the

low learning rate has led to slow adaptation and incomplete removal of noise. Both LMS and NLMS show partial success, with NLMS generally performing better due to its normalization feature.

This detailed breakdown illustrates how the low learning rate impacts the performance of echo and noise cancellation algorithms, highlighting the challenges and partial successes observed in the simulation.

Mu = 0.05:

This is the detailed description of each plot for **Mu = 0.05**:

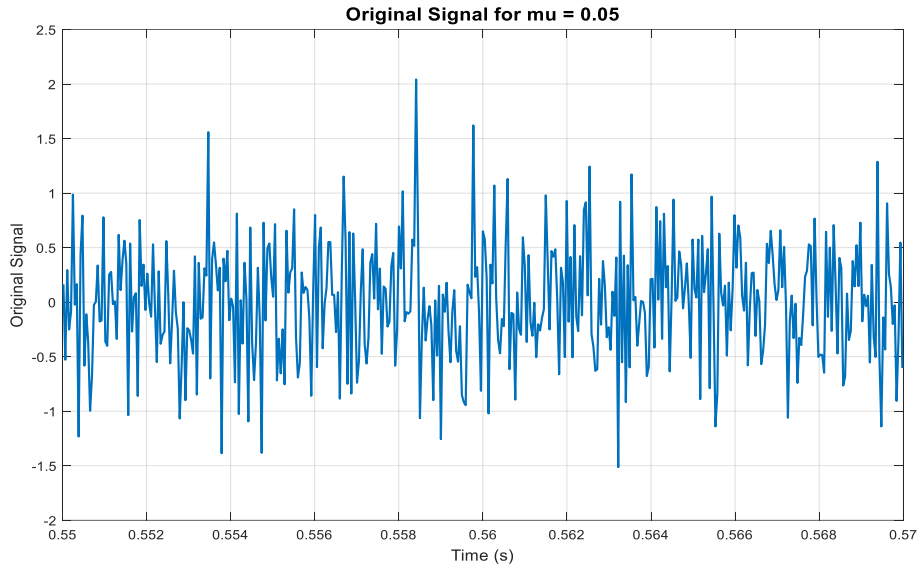

Figure 18: Plot of original signal for mu = 0.05

1. **Original Signal:** The plot of the original signal for **Mu = 0.05** is consistent with the plots from previous learning rates. It displays the audio signal with minor added noise, providing a baseline reference for comparing the effects of different learning rates on signal processing.

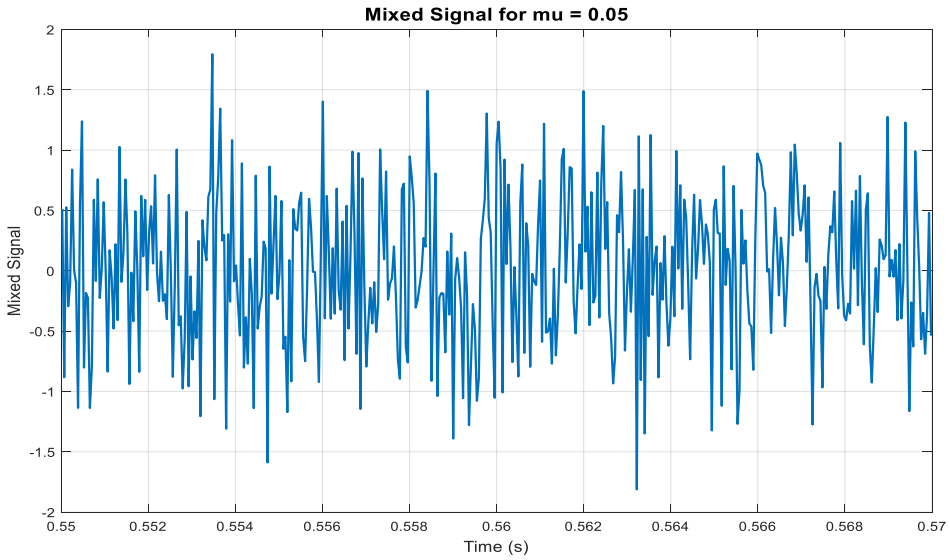

Figure 19: Plot of mixed signal for mu = 0.05

2. **Mixed Signal:** The mixed signal plot remains the same as in previous scenarios. It illustrates the combined effect of the original signal, echo, and added noise, setting the stage for evaluating the performance of the cancellation algorithms.

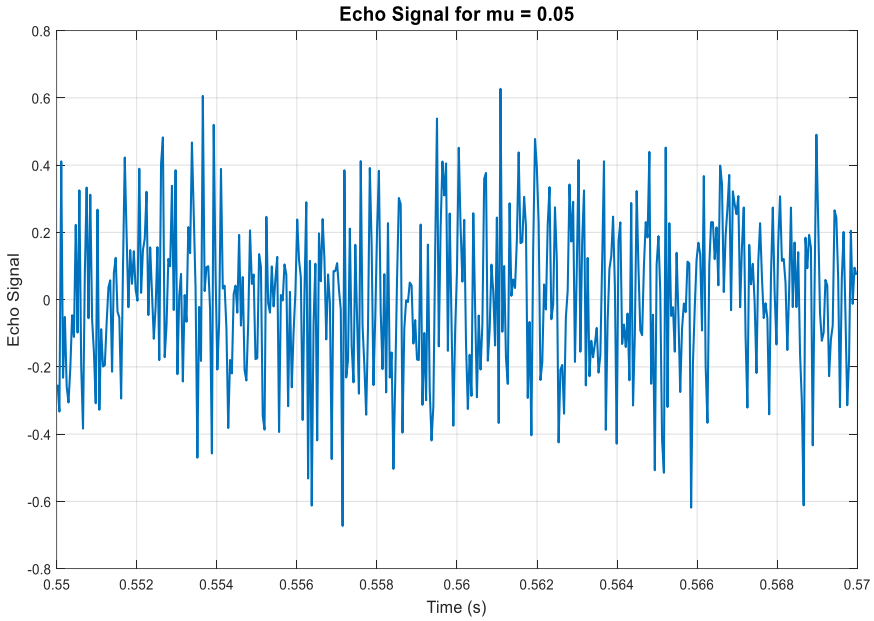

Figure 20: Plot of echo signal for mu = 0.05

3. **Echo Signal:** The echo signal plot is identical to the previous plots, showing the delayed version of the original signal. This plot helps in understanding the impact of the echo on the overall signal.

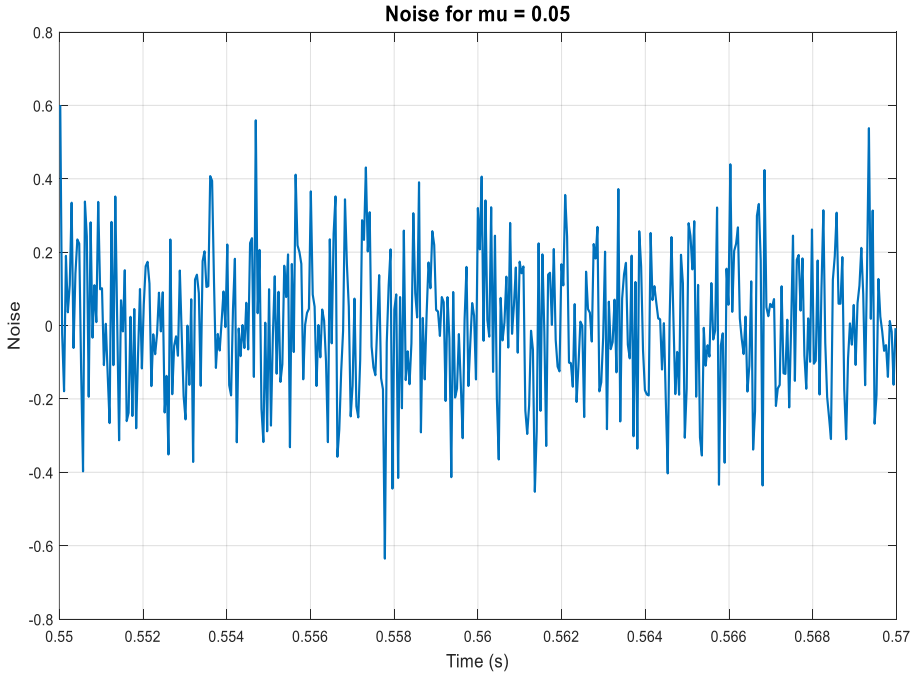

Figure 21: Plot of noise for mu = 0.05

4. **Noise:** The noise plot continues to represent the added random noise component. It shows how the noise affects the signal, maintaining consistency with the previous descriptions.

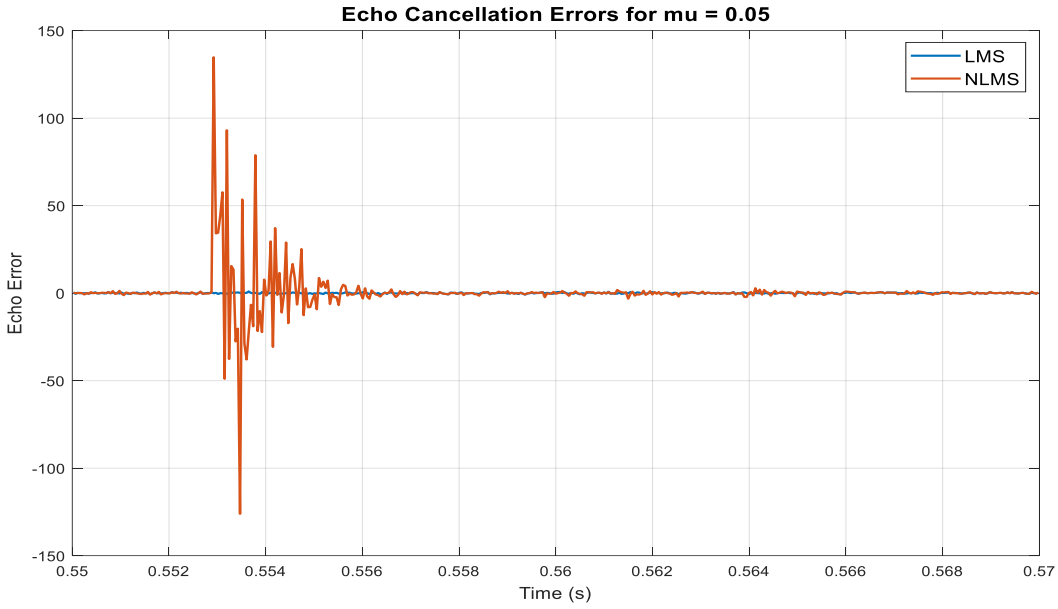

Figure 22: Plot of echo cancellation errors for mu = 0.05

5. **Echo Cancellation Errors (LMS and NLMS):** For **Mu = 0.05**, the echo cancellation errors decrease faster than they did with **Mu = 0.01**, indicating that both LMS and NLMS algorithms adapt more quickly to the echo. The NLMS algorithm demonstrates a quicker convergence compared to LMS, thanks to its normalization mechanism which adjusts more dynamically to changes in the signal.

DISCUSSION

To perform a detailed comparative analysis of the results based on the plots generated by the provided code for different mu values (0.01, 0.05, 0.1, and 0.5), there is need to examine how each value affects the performance of the LMS and NLMS algorithms in terms of echo and noise cancellation.

The findings revealed that the original signal was contaminated with both noise and echo to create a mixed signal, effectively simulating real-world scenarios where audio signals often suffer from such interferences.

Examining the effect of different mu values, it was found that for mu = 0.01, both LMS and NLMS algorithms exhibited a slow convergence rate, with high initial error signals for both echo and noise cancellation that decreased gradually. The echo and noise cancelled signals showed some improvement, but residual noise and echo were still present. With mu = 0.05, the convergence rate improved compared to mu = 0.01, and error signals decreased more rapidly, indicating more effective cancellation. The resulting echo and noise cancelled signals were cleaner, with less residual noise and echo. At mu = 0.1, both LMS and NLMS algorithms demonstrated fast convergence, with error signals dropping significantly, showing efficient cancellation. The echo and noise cancelled signals were notably cleaner, with minimal residual interference. For mu = 0.5, the convergence was very fast, but the risk of instability increased, especially for the LMS algorithm. However, the NLMS algorithm handled the high mu value better, maintaining stability and effectively cancelling echo and noise. The resulting

echo and noise cancelled signals were very clean, but the LMS algorithm showed potential instability artifacts.

Comparing the performance of the algorithms, it was observed that the NLMS algorithm consistently outperformed the LMS algorithm across all mu values. The normalization step in NLMS allowed it to adapt more effectively to variations in signal power, leading to better stability and faster convergence. At higher mu values, NLMS maintained stability better than LMS, resulting in cleaner cancellation.

CONCLUSION

Based on the simulation results, the following conclusions can be drawn:

1. **Optimal Mu Value:** A mu value between 0.05 and 0.1 strikes a good balance between convergence speed and stability for both LMS and NLMS algorithms. Higher mu values (e.g., 0.5) can lead to faster convergence but risk instability, particularly for the LMS algorithm.
2. **Algorithm Preference:** The NLMS algorithm is generally preferred over the LMS algorithm due to its ability to handle signal power variations more effectively and maintain stability at higher mu values. NLMS provided more consistent and cleaner cancellation of both echo and noise across different mu values.
3. **Practical Implications:** For real-world applications where both echo and noise need to be cancelled from audio signals, the NLMS algorithm with a mu value around 0.05 to 0.1 is recommended. Proper tuning of mu is crucial to achieve the desired balance between convergence speed and stability, ensuring effective cancellation without introducing instability.

RECOMMENDATIONS

Recommendations for enhancing echo and noise cancellation in audio processing applications are as follows:

1. **Algorithm Selection:** Utilize the NLMS algorithm for most practical scenarios involving echo and noise cancellation due to its superior performance and stability.
2. **Mu Value Tuning:** Adjust the mu value carefully based on the specific characteristics of the audio signal and the level of interference. Generally, a mu value in the range of 0.05 to 0.1 is effective.
3. **Further Research:** Explore adaptive methods for dynamically adjusting the mu value in real-time to optimize the performance of LMS and NLMS algorithms under varying signal conditions. Investigate hybrid approaches that combine the strengths of both LMS and NLMS algorithms for improved robustness and performance.
4. **Application to Real-World Signals:** Evaluate the algorithms on a broader range of real-world audio signals with different types and levels of interference to validate their effectiveness and refine the parameter tuning process.

These recommendations if followed will help in the effectiveness of echo and noise cancellation in audio processing applications, resulting in clearer and more intelligible audio signals.

Declaration by Author

Acknowledge: The author extends his heartfelt appreciation to the Administration of the National Institute of the National Institute of Construction Technology and Management (NICTM), Uromi and the Tertiary Education Trust Fund (TetFund).

Source of Funding: Tertiary Education Trust Fund (TetFund)

Conflict of Interest: Nil

REFERENCES

1. Aboy, M., Márquez, O. W., McNames, J., Hornero, R., Trong, T., & Goldstein, B. (2005). Adaptive modeling and spectral estimation of nonstationary biomedical signals based on Kalman filtering. IEEE transactions on biomedical engineering, 52(8), 1485-1489.

2. Ahirwal, M. K., Kumar, A., & Singh, G. K. (2021). Fundamentals of Adaptive Filters. In Computational Intelligence and Biomedical Signal Processing: An Interdisciplinary, Easy and Practical Approach (pp. 21-50). Cham: Springer International Publishing.
3. Almagbile, A., Wang, J., & Ding, W. (2010). Evaluating the performances of adaptive Kalman filter methods in GPS/INS integration. Journal of Global Positioning Systems, 9(1), 33-40.
4. Apolinário Jr, J. A., & Netto, S. L. (2008). Introduction to adaptive filters. In QRD-RLS Adaptive Filtering (pp. 1-27). Boston, MA: Springer US.
5. Avalos, J. G., Sanchez, J. C., & Velazquez, J. (2011). Applications of adaptive filtering. Adaptive Filtering Applications, 1, 3-20.
6. Benesty, J., Cohen, I., & Chen, J. (2017). Fundamentals of signal enhancement and array signal processing. John Wiley & Sons.
7. Bouaafia, S., Messaoud, S., Khemiri, R., & Sayadi, F. E. (2022, May). An FPGA-SoC based Hardware Acceleration of Convolutional Neural Networks. In 2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT) (pp. 537-542). IEEE.
8. Cao, W., & Zhang, Q. (Eds.). (2021). Adaptive Filtering - Recent Advances and Practical Implementation. IntechOpen. doi: 10.5772/intechopen.91562
9. Casebeer, J., Bryan, N. J., & Smaragdis, P. (2022). Meta-af: Meta-learning for adaptive filters. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 31, 355-370.
10. Comminiello, D., Nezamdoust, A., Scardapane, S., Scarpiniti, M., Hussain, A., & Uncini, A. (2022). A new class of efficient adaptive filters for online nonlinear modeling. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 53(3), 1384-1396.
11. de Campos, M. L., Werner, S., & Apolinário Jr, J. A. (2004). Constrained adaptive filters. In Adaptive Antenna Arrays: Trends and Applications (pp. 46-64). Berlin, Heidelberg: Springer Berlin Heidelberg.
12. Dewasthale, M. M., Kharadkar, R. D., & Bari, M. (2015, December). Comparative performance analysis and hardware implementation of adaptive filter algorithms for acoustic noise cancellation. In 2015 International Conference on Information Processing (ICIP) (pp. 124-129). IEEE.
13. Diniz, P. S. (2019). Adaptive Filtering: Algorithms and Practical Implementation. Springer Nature.
14. Dogancay, K. (2008, May). Performance benefits of resource-constrained adaptive filtering. In 2008 3rd International Symposium on Wireless Pervasive Computing (pp. 266-269). IEEE.
15. Esposito, D., De Caro, D., Di Meo, G., Napoli, E., & Strollo, A. G. (2019). Low-power hardware implementation of least-mean-square adaptive filters using approximate arithmetic. Circuits, Systems, and Signal Processing, 38(12), 5606-5622.
16. Fauzi, H., & Batool, U. (2019). A three-bar truss design using single-solution simulated Kalman filter optimizer. Mekatronika, 1(2), 98-102.
17. Freire, P. J., Srivallapanondh, S., Napoli, A., Prilepsky, J. E., & Turitsyn, S. K. (2022). Computational complexity evaluation of neural network applications in signal processing. arXiv preprint arXiv:2206.12191.
18. Fuhg, J. N., Fau, A., & Nackenhorst, U. (2021). State-of-the-art and comparative review of adaptive sampling methods for kriging. Archives of Computational Methods in Engineering, 28, 2689-2747.
19. Haykin, S. (2017). Adaptive systems for signal process. Advanced Signal Processing: Theory and Implementation for Sonar, Radar, and Non-Invasive Medical Diagnostic Systems, 25.
20. Hoang, H. S., & Baraille, R. (2017). A comparison study on performance of an adaptive filter with other estimation methods for state estimation in highdimensional system. Advances in Statistical Methodologies and their Application to Real Problems. Rijeka, Croatia: IntechOpen, 29-52.
21. Huq, K. M. S., Bergano, M., Gameiro, A., & Arefin, M. T. (2009). Channel Equalization in Digital Transmission. IJCSIS, 89.
22. Jiang, C., Zhang, S., Li, H., & Li, Z. (2021). Performance evaluation of the filters with adaptive factor and fading factor for GNSS/INS integrated systems. GPS Solutions, 25, 1-11.

23. Kalita, D., & Lyakhov, P. (2022). Moving Object Detection Based on a Combination of Kalman Filter and Median Filtering. Big Data and Cognitive Computing, 6(4), 142.
24. Katz, D. (2021). Real-time Audio Processing with Optimized Signal Path. ATZelectronics worldwide, 16(10), 42-45.
25. Liu, X., Wang, C., & Fan, X. (2023, August). A Real-Time Parallel Information Processing Method for Signal Sorting. In International Conference on Database and Expert Systems Applications (pp. 298-303). Cham: Springer Nature Switzerland.

How to cite this article: Joshua Okoekhian, Mathurine Guiawa, Ikenna Onyegbadue. Design and simulation of adaptive filters in real-time environment. *International Journal of Research and Review*. 2024; 11(11): 215-238. DOI: *https://doi.org/10.52403/ijrr.20241118*

******