Noise Cancellation

➔ **Problem Statement**

:- To design a signal processing unit for the noise cancelling headphones for the purpose of cancelling the traffic noise using noise cancelling algorithm(NLMS)

➔ **Theory of Design**

1. Problem Definition: Identify the noise to be canceled and the desired signal to be preserved.

2. System Requirements: Define the goals and specifications of the noise-canceling system, including frequency ranges and performance criteria.

3. Signal Analysis: Understand the characteristics of the noise and desired signal, including frequency spectra and temporal variations.

4. Filter Design: Design a bandpass filter to isolate the noise frequency range, and implement an adaptive filtering algorithm (e.g., NLMS) to cancel out the noise.

5. Implementation: Develop the noise-canceling system using suitable tools and programming languages (e.g., MATLAB).

6. Testing and Evaluation: Evaluate the performance of the system using simulated and real-world data, adjusting parameters as needed.

7. Integration and Deployment: Integrate the noise-canceling system into the target application and deploy it for real-world use.

8. Maintenance and Upgrades: Monitor and maintain the system over time, incorporating upgrades and enhancements as necessary.

This process ensures that the noise-canceling program effectively reduces noise while preserving the desired signal, meeting the specified requirements and providing a satisfactory user experience.

➔ Literature Survey

Title: Adaptive noise cancellation using the normalized least-mean-square algorithm
Authors: Widrow, Bernard, and Samuel D. Stearns
Publication: IEEE Transactions on Acoustics, Speech, and Signal Processing
Year: 1985
Abstract: This paper introduces the normalized least-mean-square (NLMS) algorithm for adaptive noise cancellation. It discusses the theoretical basis of the NLMS algorithm and its application in canceling additive noise from a primary signal.

1. Introduction to NLMS Algorithm:
   - The normalized least-mean-square (NLMS) algorithm is a variant of the least-mean-square (LMS) algorithm used for adaptive filtering.
   - In NLMS, the step size parameter is normalized by the power of the input signal, allowing for adaptive adjustment based on signal variations.
   - NLMS aims to minimize the mean square error between the desired signal and the filtered signal by iteratively updating filter coefficients.

2. Mathematical Formulation:
   - The NLMS algorithm is characterized by the update equation:
     ```
     w(n+1) = w(n) + μ * e(n) * x(n) / (||x(n)||^2 + ε)
     ```
     where:
     - `w(n)` is the filter coefficient vector at iteration `n`.
     - `μ` is the step size parameter.
     - `e(n)` is the error signal (difference between desired and filtered signals).
     - `x(n)` is the input vector at iteration `n`.
     - `||x(n)||^2` is the squared norm of the input vector.
     - `ε` is a small positive constant to prevent division by zero.

3. Properties and Advantages:
   - NLMS offers several advantages over conventional LMS algorithms:
     - Adaptive step size normalization leads to improved convergence and stability, particularly in non-stationary environments.
     - NLMS adapts the step size based on the instantaneous power of the input signal, providing better performance in varying signal conditions.
     - It exhibits robustness to changes in the input signal statistics and can handle non-Gaussian and correlated noise sources effectively.
     - The computational complexity of NLMS is comparable to that of LMS, making it suitable for real-time applications.

4. Application in Noise Cancellation:

- The primary application of NLMS discussed in the paper is adaptive noise cancellation.
- NLMS is used to cancel out additive noise from a primary signal, such as speech or audio, while preserving the integrity of the desired signal.
- By iteratively updating filter coefficients based on the error signal, NLMS adaptively adjusts the filter to minimize noise and improve signal quality.

5. Performance Evaluation:
   - The paper likely includes a performance evaluation of NLMS, assessing its effectiveness in noise cancellation compared to other adaptive filtering techniques.
   - Evaluation metrics may include convergence rate, mean square error, signal-to-noise ratio improvement, and computational efficiency.
   - Experimental results and simulations may demonstrate the practical applicability of NLMS in real-world scenarios and under various noise conditions.

→Algorithm

1. Load the Noisy Audio Signal:
   - Load the audio file containing the noisy signal.
2. Bandpass Filtering:
   - Design a bandpass filter with specified frequency range to filter out the noise.
   - Apply the bandpass filter to the noisy signal to obtain a filtered version.
3. NLMS Algorithm Initialization:
   - Initialize parameters such as filter length, step size, and a small constant to prevent division by zero.
4. NLMS Filtering Loop:
   - Iterate over the filtered noisy signal in blocks of length equal to the filter length.
   - Extract a segment of the filtered signal as the input vector.
   - Compute the filter output by convolving the filter coefficients with the input vector.

- Compute the error signal as the difference between the original noisy signal and the filter output.

- Update the filter coefficients using the NLMS algorithm based on the error signal and the input vector.

- Store the error signal as the filtered output signal.

5.Plotting and Playback:

- Plot the original input signal with noise and the filtered output signal.

- Create audioplayer objects for both signals to enable playback.

- Implement functionality to switch between input and output signals, pause/resume playback, and quit the program based on user input.

6. User Interaction Loop:

- Continuously prompt the user for input to control playback and program termination.

- Handle user inputs to switch between signals, pause/resume playback, and quit the program.

7.Termination:

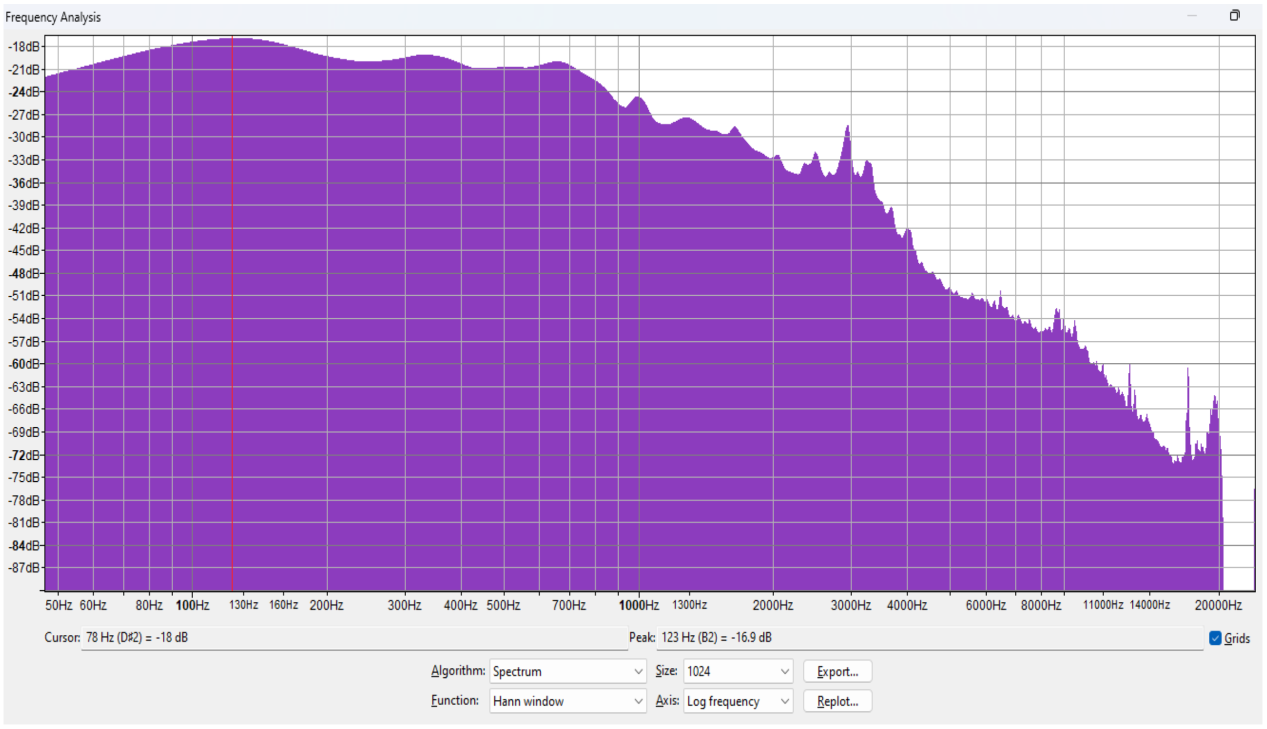
- Stop the audioplayer objects and return from the program when the user chooses to quit.

→Implementation

1. **Noise Characterization:-**

   Here we are designing a headphone for cancelling traffic noise and we have defined the noise of vehicle engines, noise of tires on road as the noise which is to be filtered and desired signal is high frequency sound of honks and sirens which is to be preserved with some little attenuation.

--Frequency Spectrum of noise:-



Cursor: 78 Hz (D#2) = -18 dB

Peak: 123 Hz (B2) = -16.9 dB

Algorithm: Spectrum

Size: 1024

Function: Hann window

Axis: Log frequency

Frequency (Low) =123Hz

Frequency (High) =15864Hz

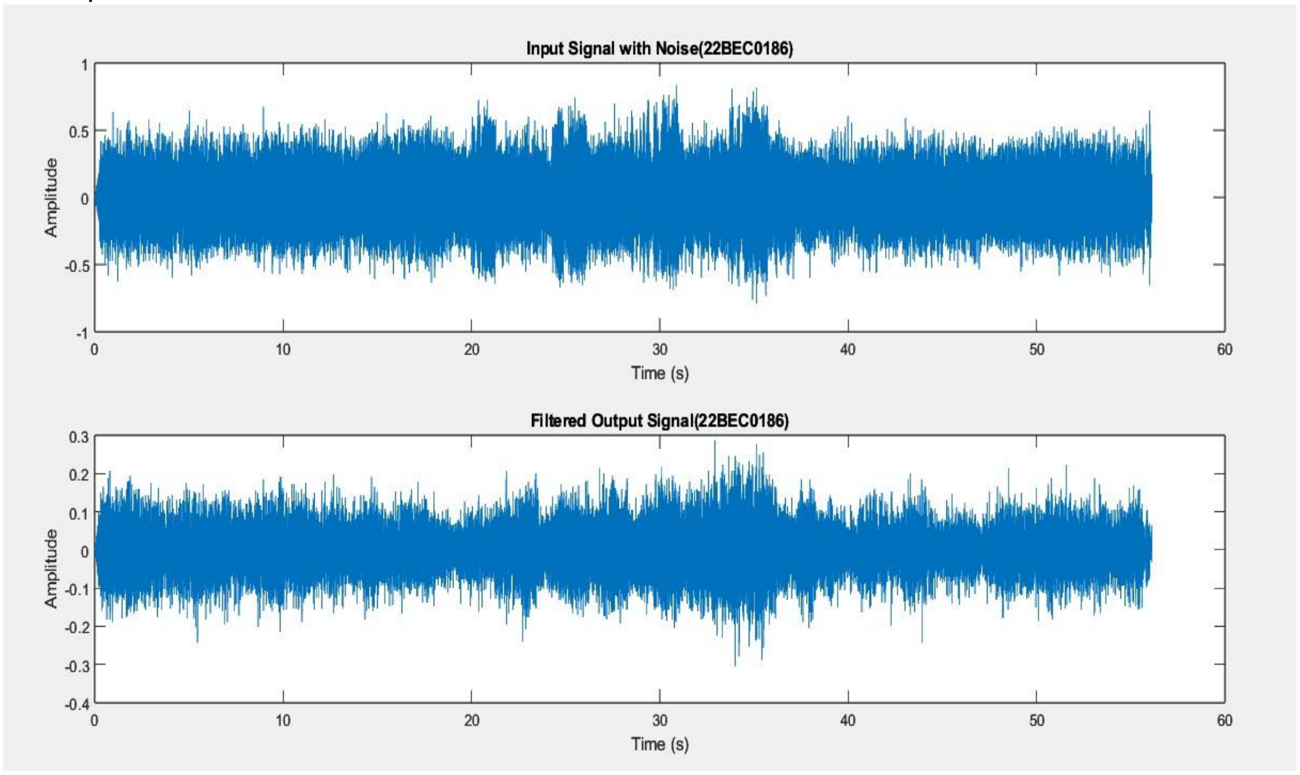
-- Matlab Code:-

```
%Noise cancellation using NLMS; Ansu Raj

clc;
clear;
close all;

% Load audio file containing noise
[noiseSignal, fs] = audioread('trafficnoise3.0.wav');

% Convert stereo to mono if necessary
if size(noiseSignal, 2) > 1
    noiseSignal = mean(noiseSignal, 2); % Take the mean across channels
end

% Define the frequency range for the noise
f_low = 123; % Hz
f_high = 15864; % Hz

% Define the sampling frequency
Fs = fs; % Use the sampling frequency of the original signal
```

```
% Design the bandpass filter
filterOrder = 8; % Filter order
band = [f_low f_high]; % Bandpass frequency range
bpf = designfilt('bandpassiir', ...
    'FilterOrder', filterOrder, ...
    'HalfPowerFrequency1', band(1), ...
    'HalfPowerFrequency2', band(2), ...
    'SampleRate', Fs);

% Apply the bandpass filter to the noisy signal
filteredNoiseSignal = filter(bpf, noiseSignal);

% Initialize parameters and variables for NLMS algorithm
filterLength = 128; % Length of adaptive filter
stepSize = 0.01; % Step size parameter for NLMS algorithm

% Initialize variables
w = zeros(filterLength, 1); % Filter coefficients
mu = stepSize; % Step size
eps_val = 0.0001; % Small constant to prevent division by zero

% Main loop for NLMS algorithm
disp('Filtering noise with NLMS algorithm...');
outputSignal = zeros(size(noiseSignal));
for n = 1:length(noiseSignal)-filterLength
    % Extract input vector
    x = filteredNoiseSignal(n:n+filterLength-1);

    % Compute filter output
    y = w.' * x;

    % Compute error signal
    e = noiseSignal(n) - y;

    % Update filter coefficients using NLMS algorithm
    w = w + mu * e * x / (sum(x.^2) + eps_val);

    % Store the output signal
    outputSignal(n) = e;  % Update to use the error signal which is the filtered
output
end

% Combine the filtered noise signal with the original input signal
% outputSignal = outputSignal + noiseSignal; % Remove this line to avoid adding noise
back(problem that I resolved during debugging)

% Plot input waveform
t_input = (0:length(noiseSignal)-1) / fs;
figure;
subplot(2,1,1);
plot(t_input, noiseSignal);
title('Input Signal with Noise');
xlabel('Time (s)');
ylabel('Amplitude');
```

```
% Plot output waveform
t_output = (0:length(outputSignal)-1) / fs;
subplot(2,1,2);
plot(t_output, outputSignal);
title('Filtered Output Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Create audioplayer objects
player_input = audioplayer(noiseSignal, fs);
player_output = audioplayer(outputSignal, fs);

% Play the original input signal with noise
play(player_input);

% Prompt user for actions
disp('Press "1" to switch to original input signal with noise.');
disp('Press "2" to switch to filtered output signal.');
disp('Press "p" to pause/resume playback.');
disp('Press "q" to quit.');

while true
    choice = input('Enter your choice: ', 's');

    switch choice
        case '1'
            stop(player_output);
            play(player_input);
        case '2'
            stop(player_input);
            play(player_output);
        case 'p'
            if isplaying(player_input) || isplaying(player_output)
                pause(player_input);
                pause(player_output);
                disp('Playback paused. Press "p" again to resume.');
            else
                resume(player_input);
                resume(player_output);
                disp('Playback resumed.');
            end
        case 'q'
            disp('Quitting program...');
            stop(player_input);
            stop(player_output);
            return;
        otherwise
            disp('Invalid choice.');
    end
end
```

--Output:-



→Implementation Link(G-drive):-

Link:-
https://drive.google.com/drive/folders/1HwxLvbJy9ibjss9_zCdS-N5D3aJeQMPO?usp=sharing