Signal Processing Project Report

Adithi Samudrala 2022102065
Vaishnavi Shivkumar 2022102070
Saktidhar PV 2022112005

December 6, 2023

Abstract

This project explores signal processing through three main tasks: echo creation, echo cancellation, and sound detection/classification. The investigation involves the synthesis of artificial echoes, development of methods to cancel unwanted echoes, and the application of advanced signal processing for sound detection and classification using MATLAB. Practical implementations and insights are presented to address audio challenges. The report provides a concise overview of echo creation, the second task, and the third task involving signal processing techniques and algorithms implemented in MATLAB.

1 Introduction

In the field of audio signal processing, addressing challenges related to echo, cancellation, and sound detection is crucial for ensuring high-quality communication and audio playback. Artificial echoes aid in the development and testing of echo cancellation algorithms. Echo cancellation is essential for mitigating unwanted reflections, enhancing the clarity of transmitted or recorded content. Additionally, sound detection and classification are fundamental tasks with applications ranging from security surveillance to voice-activated systems. Robust algorithms are required to distinguish between different sound sources and accurately classify them, contributing to the efficiency and reliability of audio processing systems. The report delves into the methodologies employed for each task, detailing signal processing techniques and algorithms implemented in MATLAB. The objective is to provide a comprehensive understanding of the methodologies used, challenges encountered, and the results obtained in the pursuit of effective echo creation, cancellation, and sound detection and classification. The findings are anticipated to contribute to the advancement of signal processing techniques applicable to audio systems and telecommunications.

2 Part 1 - Echo Creation

2.1 Objective/Aim

This section aims to explore audio signal processing, focusing on creating an echo effect for a given audio file. The objective is to utilize signal processing techniques to generate an audible, natural-sounding echo, simulating common acoustic environments.

2.2 Input

The input for this task is an audio file provided in a .WAV format, serving as the basis for creating the echo effect.

2.3 Problem Solving Approach

2.3.1 Reading the Audio File:

The `audioread` function is used to read the audio file specified by the input parameter `filename`. The audio signal is stored in the variable `y`, and the sampling frequency is stored in `Fs`.

2.3.2 Preparing for Echoes:

The variable `col` represents the number of channels in the audio file (mono or stereo). A time lag of 1 second (`timelag`) is defined, and the corresponding sample offset (`delta`) is calculated based on the sampling frequency. The original audio signal (`orig`) is extended by appending zeros to simulate a delay of 3 seconds.

2.3.3 Creating Echoes:

Three echoes (`echo1`, `echo2`, and `echo3`) are generated by introducing delays with decreasing magnitudes. Each echo is created by appending zeros before and after the original signal, and the amplitude is reduced by multiplying with scaling factors (0.5 in the first echo, 0.5×0.5 in the second echo, and so on).

2.3.4 Combining Signals:

The original signal and the three echoes are combined to create the final signal (`total`). This simulates the effect of multiple echoes occurring at different time intervals after the original sound.

MATLAB Code Implementation:

```
function [total, Fs] = ques1b(filename)
[y, Fs] = audioread(filename);
col = size(y, 2);
timelag = 1;
delta = round(Fs * timelag);
orig = [y; zeros(3 * delta, col)];
echo1 = [zeros(delta, col); y;
        zeros(2 * delta, col)] * 0.5;

echo2 = [zeros(2 * delta, col); y;
        zeros(delta, col)] * 0.5 * 0.5;

echo3 = [zeros(3 * delta, col); y]
        * 0.5 * 0.5 * 0.5;

total = orig + echo1 + echo2 + echo3;
end
```

2.4 Alternative approach

A feedback system is a system where outputs are fed back into the inputs to generate echos. In a feedback echo system, the output signal is not only influenced by the original input but also by the previous output. The basic idea is to feed a portion of the output signal back into the system, creating a recursive process that generates echoes over time. The feedback approach is often more efficient in creating a sustained echo effect and is easier to implement compared to the feedforward method.

In this example, the `delayed_signal` is fed back into the system with each iteration, creating a recursive process that generates echoes. The transfer function of such a system can be represented as:

$$H(z) = \frac{1}{1 - \alpha^k z^{-\Delta}}$$

where:

- α^k is the decay factor, with $0 < \alpha < 1$,
- Δ is the delay in samples.

This transfer function represents the relationship between the current output and the previous outputs, introducing a feedback loop that contributes to the creation of echoes.

2.5 Results

After applying the signal processing techniques, the result is an altered audio file containing the desired echo effect. Adjustable parameters, including delay time, provide flexibility to fine-tune the characteristics of the echo. The resulting audio will be played to demonstrate the achieved effect.

Transfer function:

$$H(z) = 1 + \alpha_1 z^{-k_1} + \alpha_2 z^{-k_2} + \alpha_3 z^{-k_3} \qquad (1)$$

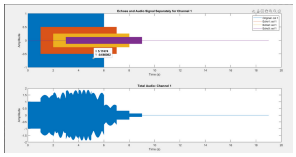

Figure 1: easy

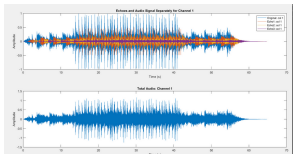

Figure 2: hard: channel 1

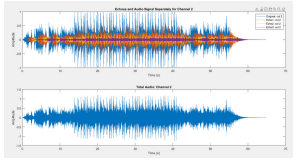

Figure 3: hard: channel 2

2.6 Conclusion

This section serves as an exploration into audio signal processing, specifically focusing on creating an echo effect. By providing a detailed problem-solving approach and presenting the expected results, the aim is to gain insights into the practical application of signal processing techniques for enhancing audio quality. The flexibility of adjustable parameters allows for simulating different acoustic environments, contributing to a comprehensive understanding of auditory experiences.

2.6.1 Limitations

`1.Limited Echo Characteristics`
The code applies a simple echo model with fixed

attenuation levels (0.5, 0.5 × 0.5, etc.). This may not capture the complexity of real-world echoes that vary in amplitude and decay over time.

`2.Potential Clipping`
If the original audio file has loud segments, the addition of multiple echoes may lead to clipping issues, where the signal exceeds the allowed amplitude range. This can result in distortion and reduce audio quality.

2.6.2 Optimization

`1.Dynamic Attenuation`
Implement a more sophisticated attenuation model that considers a dynamic decay over time. This can better emulate the natural behavior of echoes, where the reflected sound gradually diminishes.

`2.Handling Clipping`
Implement techniques to handle potential clipping issues, such as normalization or dynamic range compression. This ensures that the resulting audio maintains a suitable amplitude range without distortion.

3 Part 2 - Echo Cancellation

Echo is essentially the repetition of a sound caused by the reflection of sound waves, significantly degrading the quality of voice communication. In telecommunication systems, it manifests as the reverberation of a speaker's voice in the telephone receiver, often with a noticeable delay. This phenomenon not only disrupts the flow of conversation, it also leads to a loss of information, due to the overlap of the signals.

Echo cancellation technology is crucial in mitigating these challenges. It involves sophisticated algorithms that identify and eliminate the unwanted reflected sound.

3.1 Objective/Aim:

In a given file, we are given a mixed signal. We aim to remove the echo in this signal and retrieve the signal without echo.

3.2 Input:

Our input is an audio file with non-uniform delayed echoes in the format of a .WAV file.

3.3 Problem Solving approach:

We tried three problem-solving approaches to solve this problem.

Before delving into the specific approaches, it is essential to understand the underlying signal processing concepts. Echo cancellation in audio signals involves identifying and removing the delayed and attenuated copies of the original signal (echoes). This process typically involves autocorrelation to identify the delay and amplitude of the echo and filtering techniques to isolate and remove these echoes.

Autocorrelation is a powerful tool for echo removal because it helps in identifying the time delay and amplitude of the echo relative to the original signal. In essence, autocorrelation measures the similarity of a signal with a delayed version of itself over varying time intervals. This property is particularly useful in echo detection, as echoes are essentially delayed and often attenuated replicas of the original signal. By analyzing the autocorrelation function of a signal, one can pinpoint the time lags at which the echoes occur, indicated by significant peaks in the autocorrelation plot. The autocorrelation function $R(\tau)$ for a discrete signal $x[n]$ is defined as:

$$R(\tau) = \sum_{n} x[n] \cdot x[n+\tau] \tag{2}$$

where τ is the time lag. Once these time lags are identified, it becomes feasible to estimate the delay and amplitude of the echo. This information is crucial for constructing an inverse or cancellation signal that, when added to the original signal, effectively neutralizes the echo, thereby enhancing the clarity and quality of the audio. Autocorrelation's ability to work with single-channel input also makes it a versatile choice for echo removal in various applications.

3.3.1 Approach 1: Using Peak Calculation to Remove Echoes

In this approach, the primary objective is to identify and remove echoes from an audio signal. The process involves several key steps:

- **Cross-Correlation:** Utilizing the `xcorr` function, the method computes the cross-correlation of the mixed signal with itself. This step is crucial for identifying the time lag at which echoes occur.
- **Peak Detection:** The `findpeaks` function is employed to detect peaks in the cross-correlation output. These peaks represent potential echo points in the signal.

- **Echo Characterization:** The delay and amplitude of the echo are determined by analyzing the locations and heights of these peaks.
- **Echo Signal Construction:** An echo signal is synthesized using the calculated delay and amplitude, representing the unwanted echo in the original signal.
- **Echo Removal:** The synthesized echo signal is subtracted from the original mixed signal, resulting in an echo-reduced audio signal.

Mathematically, the echo removal process can be understood as follows. Let $x[n]$ be the original signal and $e[n]$ be the echo signal. The mixed signal $m[n]$ can be represented as:

$$m[n] = x[n] + \alpha \cdot e[n - \tau] \tag{3}$$

where α is the amplitude of the echo and τ is the time delay. The goal is to reconstruct $e[n]$ and subtract it from $m[n]$ to retrieve the original signal $x[n]$. This is achieved by:

$$x[n] = m[n] - \alpha \cdot e[n - \tau] \tag{4}$$

MATLAB Code Implementation:[H]

```
function non_echo_signal =
approach1_function(mixed_signal,fs)
[y, x] = xcorr(mixed_signal, 'coeff');
y = y(x>=0);
x = x(x>=0);
figure(1);
stem(x/fs, y);

[pks, lcs] = findpeaks(y, x);
[peaks, locs] = findpeaks(pks, lcs);
echo_delay = locs(2) - locs(1);
echo_amplitude = peaks(2) / peaks(1);
echo_signal = [zeros(echo_delay, 1); ...
    mixed_signal(1:end-echo_delay)];
non_echo_signal = mixed_signal - ...
    echo_amplitude * echo_signal;
figure(2);
subplot(2,1,1)
plot(mixed_signal, 'r');
subplot(2,1,2)
 plot(non_echo_signal, 'g');

end

clc, clearvars, close all
filename1 = 'C:\Users\shivk\Documents\ ...
Matlab\sp_project\input\';
filename2 = 'q2_easy.wav';
filename = [filename1 filename2];
[mixed_signal, fs] = audioread(filename);

non_echo_signal = ...
approach1_function(mixed_signal,fs);

soundsc(non_echo_signal,fs);
```

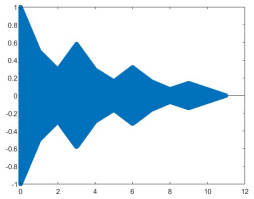

Figure 4: Autocorrelation

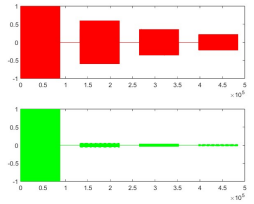

Figure 5: Mixed and Echoless signal

However, this method may not be effective for complex signals with multiple frequency components. The simplistic approach of peak detection in autocorrelation may fail to accurately identify all echoes, especially when they vary in frequency and time. Therefore, a more sophisticated method is required for complex audio signals.

3.3.2 Approach 2: Using Peak Calculations to Remove Echoes After Frequency Splitting

This approach is similar to the previous one. The primary difference between this approach and the previous one is that in this approach we create multiple bandpass filters. These bandpass filters are designed to isolate specific frequency bands corresponding to the notes in the C major scale. By splitting the signal into these bands, the echo cancellation process can be applied more precisely to each frequency range. The steps involved are:

- **Bandpass Filtering:** The mixed signal $m[n]$ is passed through a series of bandpass filters, each centered around the frequency of a note in the C major scale.

- **Echo Cancellation per Band:** The echo cancellation algorithm, as described in Approach 1, is applied independently to each frequency band. This allows for targeted removal of echoes that are specific to certain frequencies.
- **Signal Reconstruction:** After processing each band, the signals are recombined to form the complete audio signal, now with reduced echoes across all frequencies.

Mathematically, the bandpass filtering can be represented as: where $m_f[n]$ is the filtered signal in a specific frequency band, and f_{low} and f_{high} are the lower and upper bounds of the bandpass filter. The echo cancellation in each band follows the same principle as in Approach 1, but is applied to $m_f[n]$ instead of the full spectrum signal $m[n]$.

$$[H]m_f[n] = \text{Bandpass}(m[n], f_{\text{low}}, f_{\text{high}}) \quad (5)$$

MATLAB Code Implementation:

```
function non_echo_signal =
filter_it(mixed_signal, fs)
    [y, x] = xcorr(mixed_signal, 'coeff');
    y = y(x>=0);
    x = x(x>=0);

    [pks, lcs] = findpeaks(y, x);
    [peaks, locs] = findpeaks(pks, lcs);

    if length(locs) < 2
        disp('Not enough peaks for echo ...
        detection. Returning ...
        original signal.');
        non_echo_signal = mixed_signal;
        return;
    end

    echo_delay = locs(2) - locs(1);
    echo_amplitude = peaks(2) / peaks(1);
    echo_signal = [zeros(echo_delay, 1); ..
    mixed_signal(1:end-echo_delay)];
    non_echo_signal = mixed_signal - ...
    echo_amplitude * echo_signal;
    threshold = 0.*max(non_echo_signal);
    for i = 1:length(non_echo_signal)
        if(non_echo_signal(i)<threshold)
            non_echo_signal(i) = 0;
        end
    end
end

function combined_output = ...
approach2_function(mixedSignal,fs)
notes = {'C4', 'D4', 'E4', 'F4', ...
'G4', 'A4', 'B4'};
frequencies = [261.63, 293.66, 329.63, ...
349.23, 392.00, 440.00, 493.88];
bandWidth = 20;
figure;
subplot(length(notes) + 2, 1, 1);
plot((0:length(mixedSignal)-1)/fs, mixedSignal);
title('Mixed Audio Signal');
combined_output = zeros(size(mixedSignal));
for i = 1:length(notes)
    filteredSignal = bandpass(mixedSignal, ...
    [frequencies(i) - bandWidth/2, ...
    frequencies(i) + bandWidth/2], fs);
    processed_signal = filter_it(filteredSignal, fs);
    combined_output = combined_output + processed_signal;
    subplot(length(notes) + 2, 1, i + 1);
    plot((0:length(processed_signal)-1) ...
    /fs, processed_signal);
    title(['Processed Signal for ' notes{i}]);
end
subplot(length(notes) + 2, 1, length(notes) + 2);
plot((0:length(combined_output)-1)/fs, combined_output);
title('Combined Output Signal');

figure;
plot((0:length(combined_output)-1)/fs, combined_output);
title('Combined Output Signal');
end

clc; clearvars; close all;
filename1 = 'C:\Users\shivk\Documents\ ...
Matlab\sp_project\input\';
filename2 = 'q2_not_so_easy.wav';
string1 = [filename1 filename2];
[mixedSignal, fs] = audioread(string1);
combined_output = approach2_function(mixedSignal,fs);
soundsc(mixedSignal,fs);
```

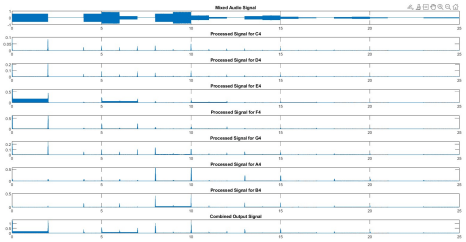

Figure 6: Frequency split

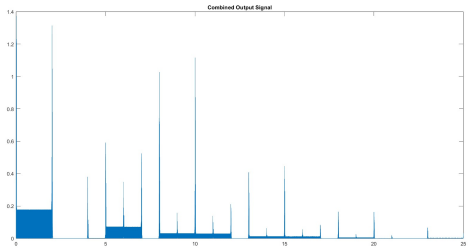

Figure 7: Final Output

However, this method faces challenges due to the complex nature of audio signals. When an audio signal is split into different frequency bands, the echoes in each band might not align perfectly with the original signal. This misalignment, especially if the echo has a frequency-dependent delay or alteration, can make it difficult to accurately identify and cancel the echo in each band. Moreover, the process of filtering and recombining the signal bands can introduce phase shifts or artefacts, potentially leading to a less coherent or altered final output. These factors can compromise the effectiveness of echo cancellation, particularly in complex or dynamically varying audio environments.

3.3.3 Approach 3: Alternative Methods of Peak Calculation: Autocorrelation Echo Peaks Using an Iterative Pivot Point

This approach employs autocorrelation to identify echo peaks in an audio signal, using an innovative method that involves iteratively adjusting a pivot point. The pivot point represents the peak of the autocorrelation function, around which a dynamic threshold line is rotated to identify significant peaks. The algorithm can be visualized as a line rotating from a pivot, with the pivots iteratively changing based on the identified peaks.

Algorithm Description:

1. **Autocorrelation:** The autocorrelation of the mixed signal is computed using `xcorr`. Although the details of autocorrelation are not the focus, it is essential for identifying time lags where echoes might occur.

2. **Normalization and Plotting:** The lag values are normalized, and the autocorrelation function is plotted.

3. **Iterative Pivot Adjustment:** Starting from the maximum value of the autocorrelation function, the algorithm iteratively adjusts the pivot point. A line, representing a dynamic threshold, rotates from this pivot.

4. **Significant Peak Identification:** Peaks that surpass the threshold line are marked as significant. The pivot point is then updated to the newly identified peak, and the process repeats.

5. **Local Maxima Detection:** Local maxima of the significant values are found, indicating potential echo points.

The mathematical representation of the threshold line rotating from the pivot point is given by:

$$y_{\text{line}}[i] = \text{pivot} - q \times \text{pivot} \tag{6}$$

where q is a scaling factor for the threshold line, and pivot is the current peak value in the autocorrelation function. **MATLAB Code Implementation:**

```
function approach3_function(mixed_signal,fs)
q = 0.002;
[y, x] = xcorr(mixed_signal, 'coeff');
y = abs(y(x >= 0));
x = x(x >= 0);
x_n = x / max(x);

figure(1);
stem(x_n, y);
hold on
plot(x_n, y(1) + q * y, 'LineWidth', 1.5)
hold on
plot(x_n, mixed_signal)
hold off
legend("Autocorrelation", "Line");
title('Autocorrelation of the Signal');
xlabel('Normalized Lag');
ylabel('Autocorrelation Coefficient');

significant_points = [];
pivot = max(y);
significant = zeros(1, length(y));
significant(1) = max(y);
pivot_index = 1;
y_line = zeros(1, length(y));
j = 1;

while j <= length(x)-1
    for i = pivot_index+1:length(x)
        cute = 0;
        y_line(i) = pivot - q * pivot;
        if y_line(i) < 0
            break;
        end

        if y(i) >= y_line(i)
```

```
            if y(i) >= y(i-1) ...
            && y(i)>=y(i+1)
                disp([y(i-1), y(i), y(i+1)]
                significant(i) = y(i);
                pivot = y(i);
                pivot_index = i;
                cute = 1;
                break;
            end
        end
    end

    j = j + 1;

    if cute==0
        q = q*1.5;
    end
    if y_line(length(x)) < y(length(x))
        break
    end
    if pivot <= 0 || pivot_index >= ...
    length(x)
        break;
    end
end
% Find local maxima in significant values
[localMaxValues, localMaxIndices] = ...
findpeaks(significant, ...
"MinPeakDistance", 0.5*fs);

% Update the plot to show the local maxima
figure(3);
plot(x, y, 'b');
hold on;
plot(x(localMaxIndices), ...
localMaxValues, 'mo');
hold off;
title('Local Maxima in Significant ...
Autocorrelation Points');
xlabel('Lag');
ylabel('Autocorrelation Coefficient');
legend('Autocorrelation', 'Local Maxima');
end

clc, clearvars, close all;
filename1 = 'C:\Users\shivk\Documents ...
\Matlab\sp_project\input\';
filename2 = 'q2_not_so_easy.wav';
filename = [filename1 filename2];
[mixed_signal, fs] = audioread(filename);

approach3_function(mixed_signal,fs);
```

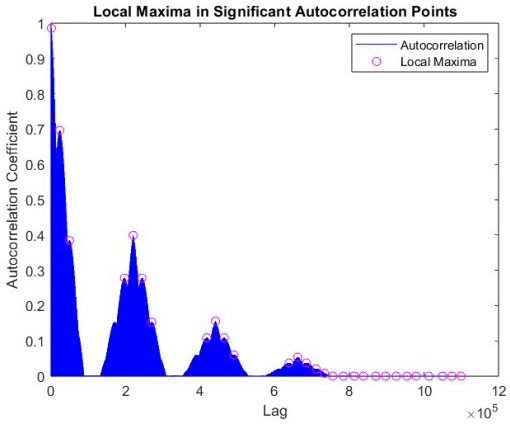

Figure 8: Autocorrelation

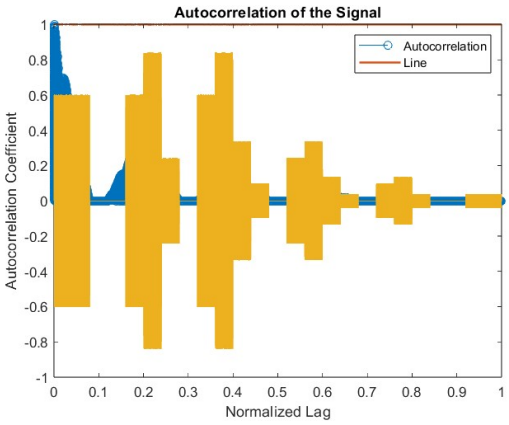

Figure 9: Signal and Autocorrelation

Limitations of the Approach: While innovative, this approach has limitations. The method relies heavily on the accurate identification of peaks in the autocorrelation function, which can be challenging in complex audio signals with multiple overlapping echoes or varying echo characteristics. The iterative nature of adjusting the pivot point and the dynamic threshold line may not always converge to the true echo peaks, especially in signals with high variability or noise. Additionally, the method's sensitivity to the scaling factor q and the choice of initial pivot can lead to inconsistent results, making it less reliable for diverse audio environments.

3.3.4 Approach 4: Alternative Methods of Peak Calculation: Autocorrelation Echo Peaks Using an Iterative Pivot Point at last index

This modified approach builds upon the previous method of using autocorrelation for echo detec-

tion, with a key alteration in the pivot point adjustment mechanism. The pivot now starts from the right-most point and moves iteratively to identify significant peaks.

Algorithm Adjustments:

1. **Pivot Initialization:** The pivot is initialized to the right-most point of the autocorrelation function, representing the latest potential echo.
2. **Iterative Backward Movement:** The algorithm iteratively moves backward from the end of the autocorrelation function, adjusting the pivot point to identify significant peaks.
3. **Dynamic Threshold Line:** A threshold line, with a slight negative gradient, is used to identify significant peaks. The line's gradient is adjusted dynamically based on the identified peaks.
4. **Significant Peak Detection:** Peaks that surpass the threshold line are marked as significant, and the pivot point is updated accordingly.

The mathematical representation of the threshold line is given by:

$$y_{\text{line}}[i] = \text{pivot} + q \times x_n[i] \tag{7}$$

where q is the scaling factor for the threshold line, and $x_n[i]$ is the normalized lag.

MATLAB Code Implementation:

```
function ...
approach4_function(mixed_signal,fs)
q = -0.0001;
[y, x] = xcorr(mixed_signal, 'coeff');
y = abs(y(x >= 0));
x = x(x >= 0);
x_n = x / max(x);

figure(1);
stem(x_n, y);
hold on
plot(x_n, y(1) + q * x_n, ...
'LineWidth', 1.5)
hold off
legend("Autocorrelation", "Line");
title('Autocorrelation of the Signal');
xlabel('Normalized Lag');
ylabel('Autocorrelation Coefficient');

significant_points = [];
pivot = max(y);
pivot_index = length(x);
significant = zeros(1, length(y));
significant(1) = y(1);
j = 0;
while j <= length(x)-1
    for i = length(x):-1:pivot_index
        cute = 0;
        y_line(i) = pivot + q * (x_n(i));
        if y_line(i) < 0
            break;
        end

        if y(i) >= y_line(i)
            if i > 1 && y(i) >= y(i-1) && y(i) >= y(i+1)
                disp("in2")
                significant(i) = y(i);
                pivot = y(i);
                pivot_index = i;
                cute = 1;
                break;
            end
        end
    end

    j = j + 1;

    if cute == 0
        q = q*1.01;
    end
    if y_line(1) < y(1)
        break
    end
    if pivot <= 0 || pivot_index <= 1
        break;
    end
end

figure(2);
plot(x, y, 'b');
hold on;
plot(x(significant > 0), ...
significant(significant > 0), 'r*');
hold off;
title('Significant Points in Autocorrelation');
xlabel('Lag');
ylabel('Autocorrelation Coefficient');
legend('Autocorrelation', 'Significant Points');
end
% Initialize pivot to the right-most point
clc, clearvars, close all;
filename1 = 'C:\Users\shivk\ ...
Documents\Matlab\sp_project\input\';
filename2 = 'q2_easy.wav';
filename = [filename1 filename2];
[mixed_signal, fs] = audioread(filename);

approach4_function(mixed_signal,fs);
```

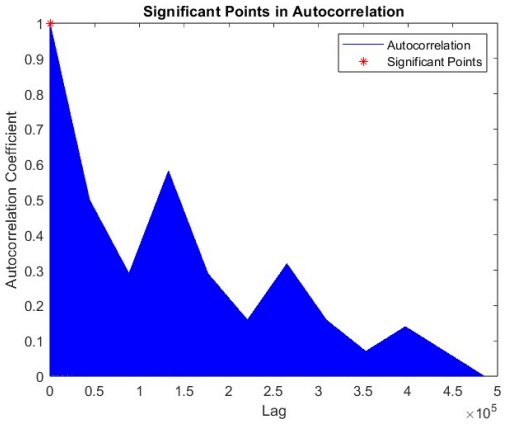

Figure 10: Autocorrelation

Advantages and Limitations: Compared to the previous approach, this method offers a more systematic way of scanning the autocorrelation function from the latest potential echo backward. This can potentially lead to more accurate identification of later echoes in the signal. However, similar to the previous approach, this method still faces challenges in complex audio signals. The accuracy of peak detection is highly dependent on the gradient q and the initial pivot position. In signals with multiple overlapping echoes or varying echo characteristics, the method may not consistently identify all significant echoes. Additionally, the backward scanning approach may miss earlier echoes, especially in signals with high variability or noise.

3.4 Further Optimizations:

In general, if we had a deep learning model that could stochastically, and exponentially reduce the threshold, we could have easily captured the peaks of this mixed signal's autocorrelation. We try our level best to do that with an algorithm, but we cannot change the initial slope and the rate of change in slope in this method effectively. Additionally we can look at more complex solutions to solve this problem like stft, wavelet transform etc. We can also use adaptive filters.

4 Part 3 - Noise Classification

4.1 Objective/Aim

In this section, you will work on classifying background noise in music recordings without removing the noise. Your objective is to accurately identify and categorise the type of noise present in the recording, distinguishing the source of origin.

The types of Noises can be:
1. Fan
2. Pressure cooker
3. Water Pump
4. Traffic

4.2 Input

The inputs for this task are 4 audio files provided in a .WAV format, each containing a mixture of a song and some background noise.

4.3 Problem-Solving Approach

4.3.1 Reading the Audio File:

The `audioread` function is used to read the audio file specified by the input parameter `filename`. The audio signal is stored in the variable `y`, and the sampling frequency is stored in `Fs`.

4.3.2 Converting into Frequency Domain:

The `FFT` function, inbuilt in MATLAB, is used to convert the given input signal into the frequency domain. This way, we can find out the frequencies present in the given signal.

`FFT Calculation:`

Let $x(t)$ be a signal, and the Fast Fourier Transform (FFT) is computed as follows:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt$$

In code, this is represented as:

$$Y(f) = \text{fft}(x)$$

This is done for all the signals given in the Sample Set.

`finding minimum of the FFTS:`

To find the common frequency components representing noise, the minimum value across multiple FFTs is calculated:

$$Y_{\text{final}}(f) = \min(Y_1(f), Y_2(f), Y_3(f), Y_4(f))$$

4.3.3 Extracting the noise:

Once we compare all the frequency responses of the Sample inputs, the frequencies common in all of them represent the original song without any noise. Once the original song is obtained, it is removed from the frequency responses, resulting in only the frequency responses of the noise. If we

apply the IFFT function, we get the noise signal in the time-domain.
Given the FFT of noise and music (fft_{nm}) and the FFT of only music (fft_{m}), the noise-only frequency component is obtained:

$$\text{noise\_only\_freq}(i) = \text{fft}_{\text{nm}}(i) - \text{fft}_{\text{m}}(i)$$

4.3.4 Detecting:

Once the sample for all the noises is obtained, each signal is compared in the frequency domain with the provided input signal. Whichever has the maximum similarity is the one present in the input signal.
Cross-correlation:

The cross-correlation between the input audio and each noise type is computed using:

$$\text{corr}(\text{signal}, \text{input\_audio}) =$$

$$\sum_k \text{signal}(k) \cdot \text{input\_audio}(k + \tau)$$

4.3.5 Noise Classification:

Counting Correlation Peaks:

The number of peaks exceeding a threshold is counted for each noise type.

$$\text{count\_corr1} = \sum_k |\text{corr1}(k)| > \text{threshold}$$

Selecting the Dominant Noise Type:

The noise type with the maximum count is selected as the dominant noise type.

$$[\sim, \text{max\_index}] = \max([\text{count\_corr1}, \text{count\_corr2},$$
$$\text{count\_corr3}, \text{count\_corr4}])$$

MATLAB Code Implementation:

```
function [fs, Y1, Y2, Y3, Y4] = ...
find_ffts(string1, string2, ...
string3, string4)
[data1, fs] = audioread(string1);
[data2, ~] = audioread(string2);
[data3, ~] = audioread(string3);
[data4, ~] =audioread(string4);

Y1 = fft(data1);
Y2 = fft(data2);
Y3 = fft(data3);
Y4 = fft(data4);

function y_final = ...
find_music(Y1, Y2, Y3, Y4)
y_final = zeros(1, length(Y1));
for i = 1:length(Y1)
    y_final(i) =
    min([Y1(i), Y2(i), Y3(i), Y4(i)]);
end
end

function noise_only = ...
find_noise_audio(fft_nm, fft_m)
%here fft_nm is fft of noise and music,...
and fft_m is fft of only music
n = length(fft_nm);
noise_only_freq = zeros(1, n);
for i = 1:n
    noise_only_freq(i) = ...
    fft_nm(i) - fft_m(i);
end
noise_only = ifft(noise_only_freq);

function input_audio = ...
loadmusicfile(input)
[input_audio, ~] = audioread(input);
end

clc, clearvars, close all

fan = 'music_ceiling-fan.wav';
traffic = 'music_city-traffic.wav';
cooker = 'music_pressure-cooker.wav';
pump = 'music_water-pump.wav';


[fs, fft_fan, fft_traffic, fft_cooker, fft_pump] = ...
find_ffts(fan, traffic, cooker, pump);

%plot_audio(fs,fft_fan, ...
fft_traffic, fft_cooker, fft_pump);

fft_music = find_music(fft_fan, ...
fft_traffic, fft_cooker, fft_pump);

fan_audio = find_noise_audio(fft_fan, fft_music);
traffic_audio = ...
find_noise_audio(fft_traffic, fft_music);
cooker_audio = ...
find_noise_audio(fft_cooker, fft_music);
pump_audio = find_noise_audio(fft_pump, fft_music);

audiowrite('fan_audio.wav', ...
fan_audio(30*fs:40*fs), fs);
disp("done")
audiowrite('traffic_audio.wav', ...
traffic_audio(30*fs:40*fs), fs);
disp("done")
audiowrite('cooker_audio.wav', ...
cooker_audio(30*fs:40*fs), fs);
```

```
disp("done")
audiowrite('pump_audio.wav', ...
pump_audio(30*fs:40*fs), fs);
disp("done")

function solve(input_audio)
s1 = 'cooker_audio.wav';
s2 = 'fan_audio.wav';
s3 = 'pump_audio.wav';
s4 = 'traffic_audio.wav';
[s_1, fs] = audioread(s1);
[s_2, ~] = audioread(s2);
[s_3, ~] = audioread(s3);
[s_4, ~] = audioread(s4);

soundsc(input_audio,fs);
corr1 = xcorr(s_1, input_audio);
corr2 = xcorr(s_2, input_audio);
corr3 = xcorr(s_3, input_audio);
corr4 = xcorr(s_4, input_audio);

threshold = 0.25*max([(corr1 + corr2 + corr3 + corr4)/4]);

% count_corr1 = sum(abs(corr1) > threshold);
% count_corr2 = sum(abs(corr2) > threshold);
% count_corr3 = sum(abs(corr3) > threshold);
% count_corr4 = sum(abs(corr4) > threshold);

count_corr1 = max(abs(corr1));
count_corr2 = max(abs(corr2));
count_corr3 = max(abs(corr3));
count_corr4 = max(abs(corr4));

count_correlations = ...
[count_corr1, count_corr2, count_corr3, count_corr

[~, max_index] = max(count_correlations);

if max_index == 1
    disp("The noise is of a cooker");
    soundsc(s_1, fs);
elseif max_index == 2
    disp("The noise is of a fan");
    soundsc(s_2, fs);
elseif max_index == 3
    disp("The noise is of a pump");
    soundsc(s_3, fs);
else
    disp("The noise is of traffic");
    soundsc(s_4, fs);
end
end
```

4.4 Results

After applying the signal processing techniques, we can extract and detect the kind of noise present in a given input signal. The noises that can be detected are:

1. Fan - 11
2. Pressure Cooker - 12
3. Water Pump - 13
4. Traffic - 14

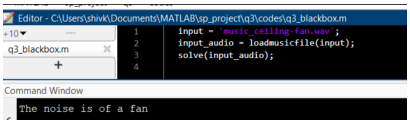

Figure 11:

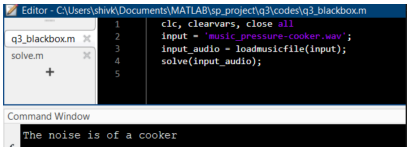

Figure 12:

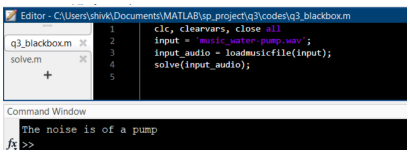

Figure 13:

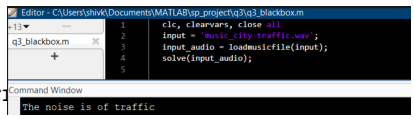

Figure 14:

4.5 Conclusion

4.5.1 Limitations

`1.Sensitivity to Threshold`
The threshold value (1000 in this case) is hard-coded and may not be universally applicable to different recordings. Setting an appropriate threshold may require experimentation or the use of adaptive thresholding methods.

`2.Assumption of Single Noise Type`
The approach assumes that the background noise is dominated by a single noise type. In real-world scenarios, multiple types of noise may coexist, and the method may not accurately identify all contributing sources.

`3.Fixed Time Window for Cross-Correlation`

The time window for cross-correlation is fixed (30*fs:40*fs), which may not be optimal for all recordings. Adaptive windowing or considering the entire signal may provide better results.

4.5.2 Optimization

`1.Adaptive Thresholding:`
Instead of a fixed threshold, we could use adaptive thresholding methods, such as statistical measures like mean and standard deviation, or machine learning-based approaches.

On an overall basis, this particular method helps us distinguish between different sounds present in a given signal, including noise. We can detect common noises present in signals and remove them to give a better and more clear signal using these techniques.

5 Bibliography

1. https://en.wikipedia.org/wiki/Echo
2. www.adaptivedigital.com/vqe-suite/echo-cancellation-explained/
3. https://www.youtube.com/watch?si=F4fShDz2IO0tfrZCv=BEBXj9A8jC8feature=youtu.be
4. https://in.mathworks.com/help/signal/ref/stft.html
5. https://in.mathworks.com/help/stats/corr.html
6. https://youtu.be/CSBDW9DpkHI?si=f2gMg6PsaouUG3Ot
7. Digital Signal Processing by Proakis
8. https://youtu.be/KMb9MtudjFs?si=j7z8A4TlAvmW85uT