Lab Project

By SHM

Signal Processing

(Monsoon 2023)

PART 1 – ECHO CREATION

In this section, we will work on the task of creating echo effects for a given audio file.

OBJECTIVE/AIM

To create an echo effect for a given audio file.

INPUT

An input audio file with no echo effect.

PROBLEM SOLVING APPROACH

In this question, we are generating echo using *filters*.

An "echo" refers to a delayed and attenuated version of a sound or signal that is heard after the original sound. It is a time-delayed replication of the original signal, where the delayed version is often quieter than the original due to the signal being reflected off surfaces or travelling a longer path.

For example, the combination of the direct sound represented by discrete signal y[n] and a single echo appearing D samples later (which is related to delay in seconds) can be generated by the equation of the form (called a difference equation)

$$x[n] = y[n] + \alpha * y[n - D], |\alpha| < 1$$

where x[n] is the resulting signal and α models attenuation of the direct sound.

There are two approaches to this problem depending on the number of echoes we require.

Multiple close-spaced echoes can be created digitally using the difference equation (using FIR- here we create finite number of echoes in the given input audio signal):

$$x[n] = \sum_{k=0}^{N-1} \alpha^k y[n - kD]$$

which generates multiple echoes spaced D samples apart with exponentially decaying amplitudes.

Another method is given by:

$$x[n] = \alpha y[n] + y[n - D] + \alpha x[n - D]$$

which simulates a higher echo density using IIR Filter – recursive approach between current output sample x[n] and current input y[n] and past input samples x[n-D] where D is the delay.

Implementation:

1. We have created four functions:
   Function 1 (loadAudioFile.m): This function is used to load the input audio file.
   Function 2 (echoCreation.m): This function is used to create echoes in an audio file.
   Here we are giving parameters a, b for the filter and the applying the *filter function* using the recursive approach:

   y = filter(b,a,x) filters the input data x using a rational transfer function defined by the numerator and denominator coefficients b and a.

   The input-output description of the `filter` operation on a vector in the Z-transform domain is a rational transfer function. A rational transfer function is of the form

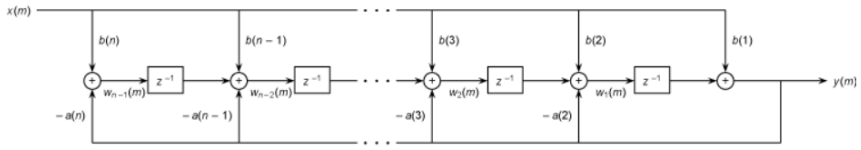
   $$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(n_b + 1)z^{-n_b}}{1 + a(2)z^{-1} + \ldots + a(n_a + 1)z^{-n_a}} X(z),$$

   which handles both FIR and IIR filters [1]. n_a is the feedback filter order, and n_b is the feedforward filter order. Due to normalization, assume $a(1) = 1$.

   You also can express the rational transfer function as the difference equation

   $$a(1)y(n) = b(1)x(n) + b(2)x(n-1) + \ldots + b(n_b + 1)x(n - n_b) - a(2)y(n-1) - \ldots - a(n_a + 1)y(n - n_a).$$

   Furthermore, you can represent the rational transfer function using its direct-form II transposed implementation, as in the following diagram. Here, $n_a = n_b = n\text{-}1$.

   The operation of `filter` at sample m is given by the time-domain difference equations

   $$\begin{aligned} y(m) &= b(1)x(m) + w_1(m-1) \\ w_1(m) &= b(2)x(m) + w_2(m-1) - a(2)y(m) \\ &\vdots \quad = \quad \vdots \qquad \vdots \\ w_{n-2}(m) &= b(n-1)x(m) + w_{n-1}(m-1) - a(n-1)y(m) \\ w_{n-1}(m) &= b(n)x(m) - a(n)y(m). \end{aligned}$$

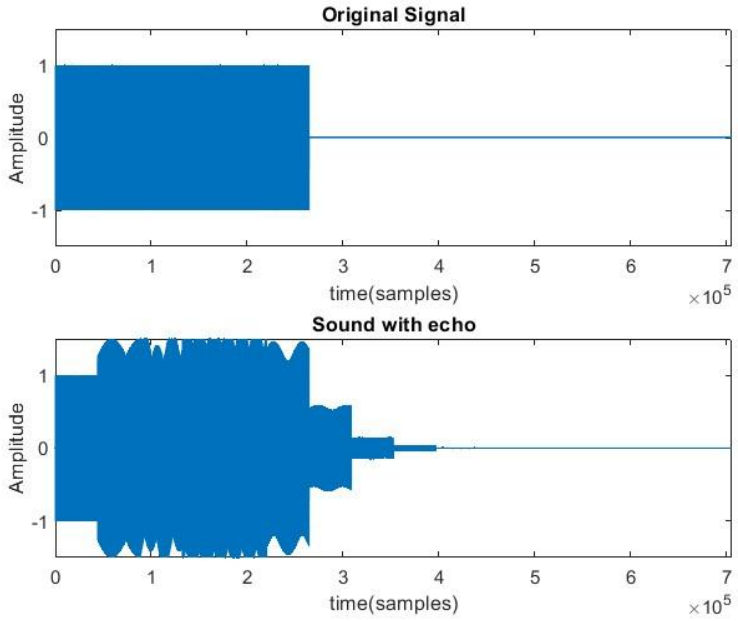
   We are also plotting the Original signal and the Output signal (Sound with Echo).
   Function 3 (checkAudioFile.m): This function is used to check if the given audio file is “mean” or “stereo”. If the audio file is stereo, we will convert it into mono by taking mean along its dimensions else it will stay the same.
   Function 4 (saveOutputAudio.m): This function is used to save the output audio file with echo effect.

2. **Main File (Q1.m):** This is the main file where we are giving the input audio file to be evaluated.

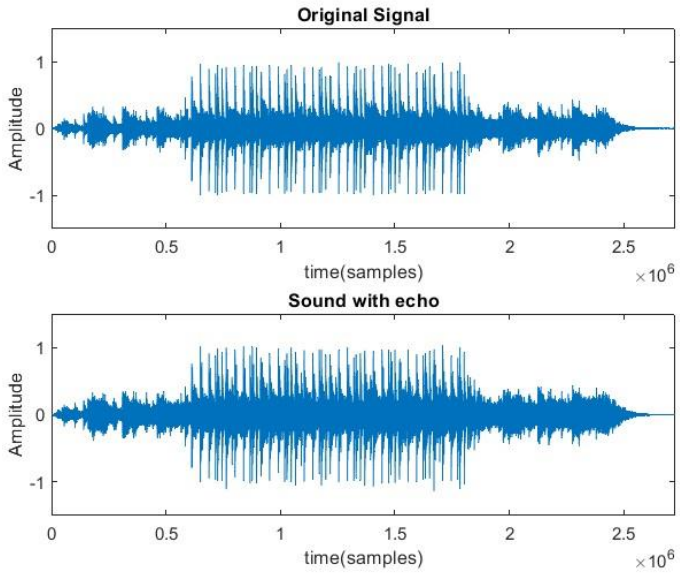
We are loading it using “loadAudioFile” function.
Then we are checking if the given input audio file is “mono” or “stereo” and converting the stereo to mono using “checkAudioFile” function.
We are giving the attenuation parameter (alpha) = 0.25, and delay (D) = 2*Fs i.e., D = 2*(Sampling Frequency)
Then using “echoCreation” function we are creating echo in the given input file.
Lastly, we are saving the output file using “saveOutputAudio” function.

RESULTS

1) Q1.wav



2) Q1_hard.wav



CONCLUSION

Therefore, we can use the concept of filters (FIR and IIR) to create echoes in the input audio file.

PART 2 – ECHO CANCELLATION

In this section, we will remove the echo component from the given input audio file.

OBJECTIVE/AIM

To implement an echo cancellation algorithm to remove the echo components from the signal, leaving only the original, clean audio.

INPUT

An audio signal which contains both the original audio and the echo with non-uniform delays.

PROBLEM SOLVING APPROACH

Here we are using two approaches to solve this problem:

1) **Using Autocorrelation**
   Our aim is to remove echo from the given input signal without the use of any desired signal. We try to compute sample locations where echo is added to our input signal through autocorrelation. In a signal containing echoes, peaks in the autocorrelation function occur at points corresponding to the time delays of the echoes.
   Autocorrelation is used to analyze the similarity of a signal with a delayed version of itself and echoes are essentially delayed and attenuated replicas of the original signal.
   We try to compute the peaks in our autocorrelation function by finding maximas in our signal until the maximas are sorted in descending order. The samples' locations correspond to these peaks give us the points where echo is added to the original signal.
   Once we have identified the sample positions where echoes are introduced in our signal, our next step involves determining the attenuation factor. This is achieved by comparing the amplitude of the input signal at these delay points with amplitude of the initial sample points where echoes are assumed to be absent.
   The attenuation factor quantifies the extent to which the echoes have been weakened or diminished compared to the original signal. This crucial

information allows us to understand how much each echo needs to be adjusted to restore the fidelity of the original signal.
Having acquired the delay and attenuation factors, the final phase involves designing a filter. This filter takes parameters as inputs and operates to approximate the original signal by compensating for the delays and adjusting the amplitudes based on the attenuation factors.
Consider the input signal to be of the form:

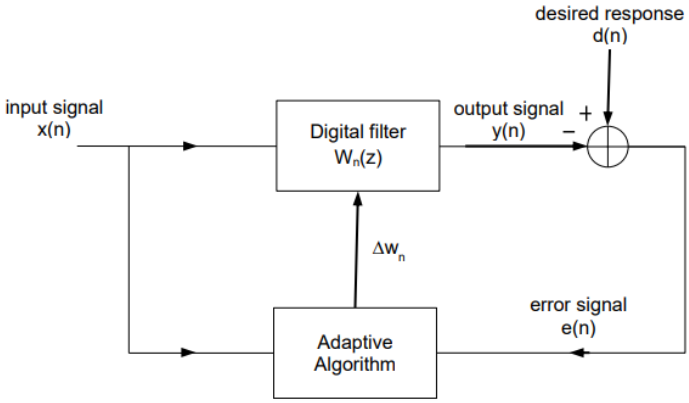
$y[n] = x[n] + \alpha_1 x[n-d_1] + \alpha_2 x[n-d_2] + \alpha_3 x[n-d_3] + \ldots\ldots\ldots + \alpha_m x[n-d_m]$

$Y/X = (1 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \alpha_3 z^{-3} + \ldots\ldots\ldots + \alpha_1 z^{-1})$

This gives us the transfer function of our filter.

2) Using NLMS – This involves the implementation of an *adaptive filter algorithm*. If the signal and noise characteristics are unknown or change continuously over time, the need for an adaptive filter arises.

Adaptive filters do not have constant filter parameters, they have the capability to continuously adjust their coefficients to their operating environment. A desired signal d(n) has to be estimated from a noise-corrupted observation x(n) = d(n) + v1(n), corrupted by additive broadband noise v1(n). Usually, the method uses a primary input containing the corrupted signal and a reference input containing noise correlated in some unknown way with the primary noise. Adaptive filter consists of two distinct parts: a digital filter with adjustable coefficients Wn(z) and an adaptive algorithm which is used to adjust or modify the coefficients of the filter.



The filtering process involves the computation of the filter output y(n) in response to the filter input signal x(n). The filter output is compared with the desired response d(n), generating an error estimation e(n). The feedback error signal e(n) is then used by the adaptive algorithm to modify the adjustable coefficients of the filter wn, generally called weight, in order to minimize the error according to some optimization criterion.
Algorithm used (NLMS – Normalized Least Mean-Square Method)

1. **Initialization:**

- Initialize the filter coefficients w to zeros or small random values.
- Set the step size parameter β, which determines the convergence rate.

2. **Iterative Update:**
   - For each sample i in the input signal:
     - Prepare the input vector U_i by stacking the current input sample with the previous $N-1$ samples, where N is the filter order.
     - Compute the filter output $y_i = w^T \cdot U_i$.
3. **Compute the Error:**
   - Calculate the error E_i by subtracting the filtered output y_i from the desired signal d_i: $E_i = d_i - y_i$.
4. **Update Weights:**
   - Update the filter weights w using the NLMS update equation: $w_{i+1} = w_i + (\|Ui\|^2 / \beta) \cdot E_i \cdot U_i^*$ where β is the step size, $||\cdot||$ denotes the norm, and $*$ represents the conjugate transpose.
5. **Repeat:**
   - Repeat steps 2-4 for each sample in the input signal.

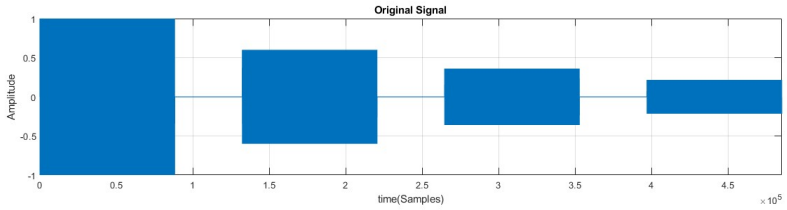
Key Concepts:

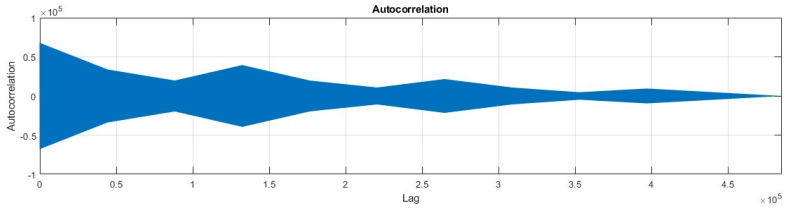
- **Filter Coefficients (w):** The filter coefficients are adjusted iteratively to minimize the error.
- **Step Size (β):** Determines the step size of each weight update. A smaller β leads to slower convergence but might prevent overshooting.
- **Filter Order (N):** Represents the number of past samples used to predict the current sample.
- **Input Vector (U_i):** Stacks the current sample with the previous $N-1$ samples, forming the input vector to the adaptive filter.
- **Error (E_i):** The difference between the desired signal and the filter output.

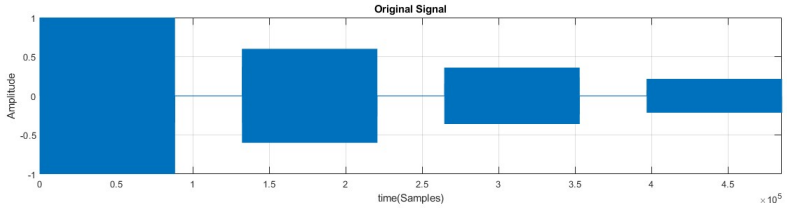
RESULTS

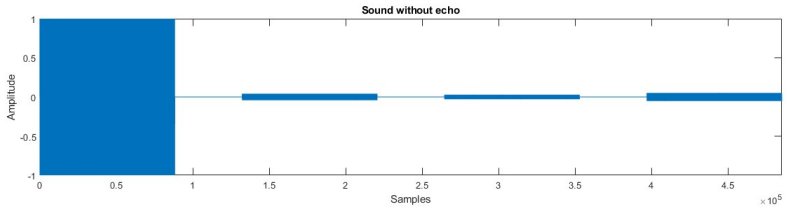
1) AUTO-CORRELATION

Q2_easy.wav

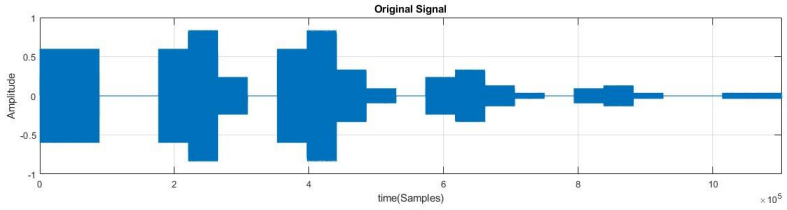


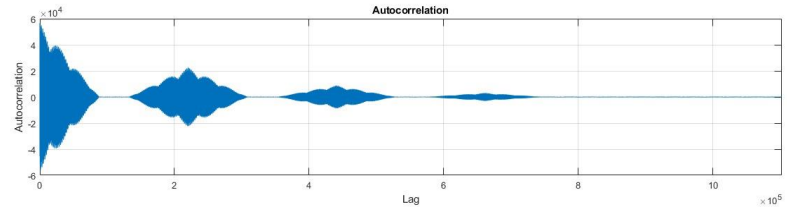


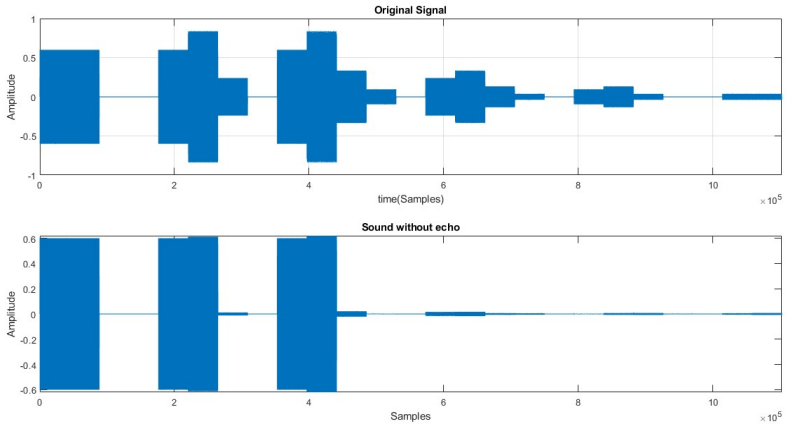




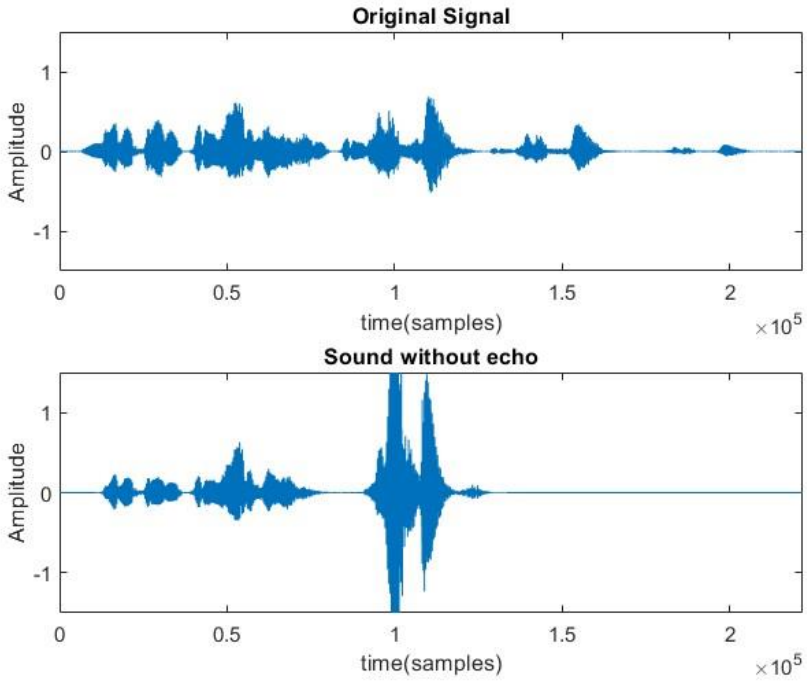
Q2_not_so_easy.wav







2) NLMS



CONCLUSION

Using the approach of either auto-correlation or NLMS, we are finally able to get an echo-free output signal.

PART 3 – WHAT IS THIS NOISE?

In this section, we will work on the task of classifying background noise in music recordings without removing the noise.

OBJECTIVE/AIM

To accurately identify and categorize the type of noise present in the recording, distinguishing source of origin.

The types of Noises can be:

- Fan
- Pressure cooker
- Water Pump
- Traffic

INPUT

Input Music Recording: A music recording in a specific audio format (e.g., WAV or MP3), which contains both the original music and background noise from indoor and outdoor sources.

PROBLEM SOLVING APPROACH

In this question, we are using the concept of *correlation* of two signals.

Correlation: It is a mathematical operation that measures the similarity between two signals. There are two types of correlations:

- Cross Correlation:
  Cross correlation measures the similarity between two different signals as a function of the time lag between them. It is often used in DSP for comparing signals or detecting the presence of one signal within the other.
  Mathematically, cross-correlation function between two signals can be given as:

  The cross correlation between two complex signals $x_1(t)$ and $x_2(t)$ is given by,

  $$R_{12}(\tau) = \int_{-\infty}^{\infty} x_1(t)\, x_2^*(t-\tau)\, dt = \int_{-\infty}^{\infty} x_1(t+\tau)\, x_2^*(t)\, dt$$

- Auto Correlation:
  Auto correlation measures the similarity between a signal and a delayed version of itself. It helps in identifying periodicity or repeating patterns within a signal.

Mathematically, auto correlation for a signal is given by:

The autocorrelation of an energy or aperiodic signal $x(t)$ is defined as –

$$R_{11}(\tau) = R(\tau) = \int_{-\infty}^{\infty} x(t)x^*(t-\tau)\ dt$$

Implementation:

3. We have created two functions:
   Function 1 (loadAudioFile.m): This function is used to load the input audio file.
   Function 2 (checkAudioFile.m): This function is used to check if the given audio file is "mean" or "stereo". If the audio file is stereo, we will convert it into mono by taking mean along its dimensions else it will stay the same.
   Function 3 (classifyNoise.m): This function is used to classify noise present in the given audio files using the concept of correlation.
   - *Extracting signal* – Here we will first extract a part of signal from all of the given audio files where the noise is maximum i.e., where the SNR (signal to noise ratio) is minimum. [In the case of given audio files, the time interval is from 30s to 40s]. We are then creating pairwise difference signals from each other to get six different combinations.
   - *Cross-Correlation* – Here we are performing correlation of all the extracted signals with the audio file that will be provided to find the presence of a specific noise using *xcorr* function.
   - *r = xcorr(x,y) returns the cross-correlation of two discrete-time sequences. Cross-correlation measures the similarity between a vector x and shifted (lagged) copies of a vector y as a function of the lag. If x and y have different lengths, the function appends zeros to the end of the shorter vector, so it has the same length as the other.*

     The cross correlation between two complex signals $x_1(t)$ and $x_2(t)$ is given by,

$$\mathrm{R}_{12}(\tau) = \int_{-\infty}^{\infty} \mathrm{x}_1(\mathrm{t})\,\mathrm{x}_2^*(\mathrm{t}-\tau)\,\mathrm{dt} = \int_{-\infty}^{\infty} \mathrm{x}_1(\mathrm{t}+\tau)\,\mathrm{x}_2^*(\mathrm{t})\,\mathrm{dt}$$

     If $x_1(t)$ and $x_2(t)$ are real signals, then,

$$\mathrm{R}_{12}(\tau) = \int_{-\infty}^{\infty} \mathrm{x}_1(\mathrm{t})\,\mathrm{x}_2(\mathrm{t}-\tau)\,\mathrm{dt} = \int_{-\infty}^{\infty} \mathrm{x}_1(\mathrm{t}+\tau)\,\mathrm{x}_2(\mathrm{t})\,\mathrm{dt}$$

The resemblance between the convolution and the correlation can be proved analytically as follows –

The convolution of two signals $x_1(t)$ and $x_2(-t)$ is given by,

$$x_1(t) * x_2(-t) = \int_{-\infty}^{\infty} x_1(\tau)\, x_2(\tau - t)\, d\tau \quad \cdots (1)$$

By replacing the variable τ by another variable p in the integral of Eqn. (1), we get,

$$x_1(t) * x_2(-t) = \int_{-\infty}^{\infty} x_1(p)\, x_2(p - t)\, dp \quad \cdots (2)$$

Now, replacing the variable t by τ in Eqn. (2), we have,

$$x_1(\tau) * x_2(-\tau) = \int_{-\infty}^{\infty} x_1(p)\, x_2(p - \tau)\, dp = R_{12}(\tau)$$

Therefore, the relation between correlation and convolution of two signals is given by,

$$R_{12}(\tau) = x_1(t) * x_2(-t)\,|_{t=\tau}$$

Similarly,

$$R_{21}(\tau) = x_2(t) * x_1(-t)\,|_{t=\tau}$$

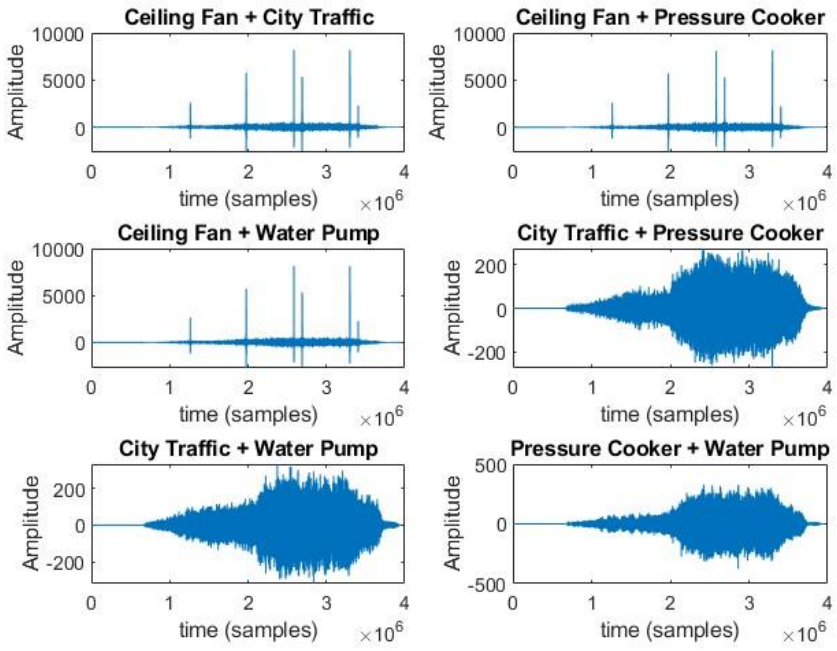
Hence, it proves that the correlation of signals $x_1(t)$ and $x_2(t)$ is equivalent to the convolution of signals $x_1(t)$ and $x_2(-t)$.

We are classifying the noise by observing the plots obtained.
Comparing the peak of all the plots, we can accurately identify and categorize the type of noise present.
For the plots, we are defining a time axis which has the same length as correlation result and finally displaying the maximum value of cross-correlation and the samples at which it occurs.
Out of the six signal outputs we will have a peak in three of them which indicates the presence of a certain noise in the input audio signal.

4. **Main File (Q3.m):** This is the main file where we give the input audio file to be evaluated.
   We are loading it using “loadAudioFile” function.
   Then we are checking if the given input audio file is “mono” or “stereo” and converting the stereo to mono using “checkAudioFile” function.
   Then using “classifyNoise” function we are classifying the noise present in the audio file into one of the categories (fan, traffic, water pump, pressure cooker).

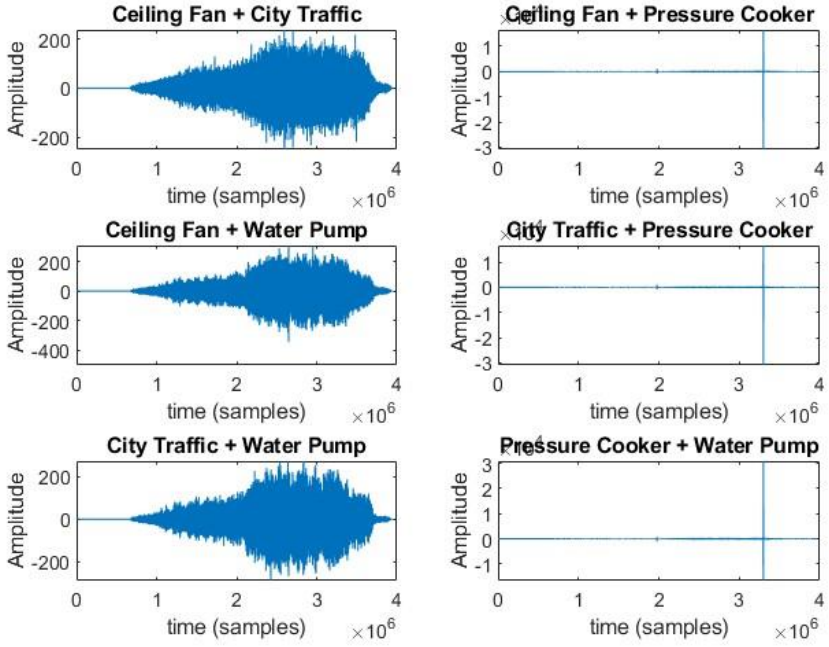
RESULTS

1) Checking for Ceiling Fan



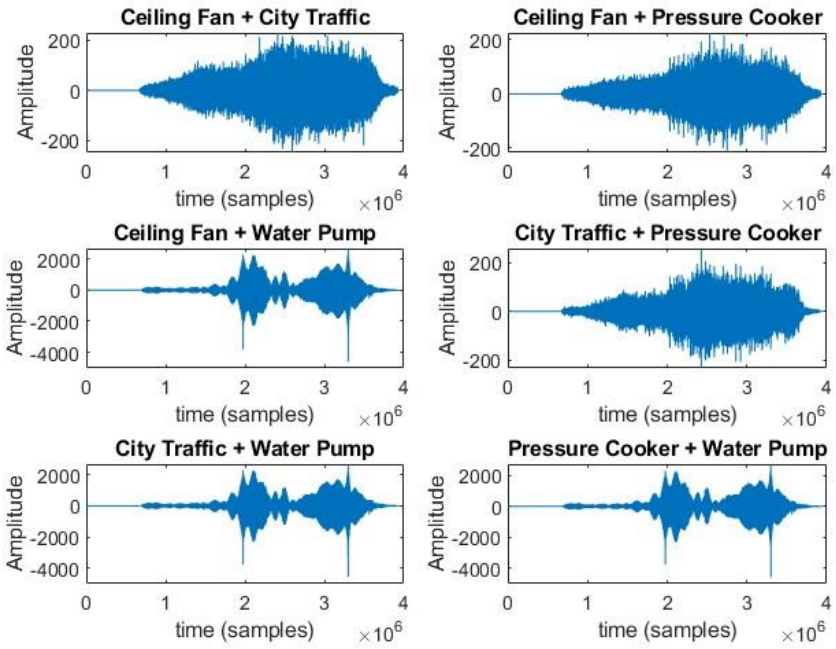

```
The detected noise type is: Ceiling Fan
```

2) Checking for Pressure Cooker



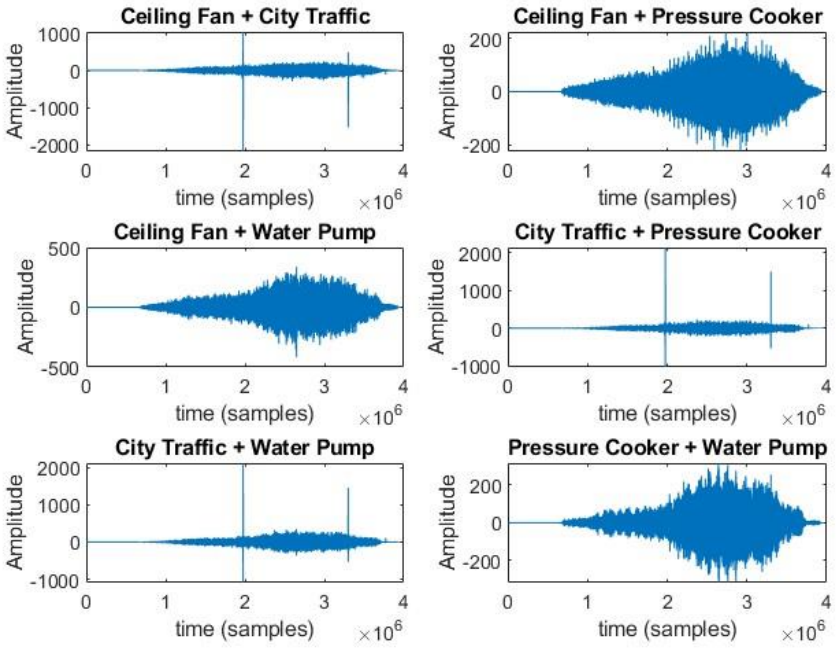

```
The detected noise type is: Pressure Cooker
```

3) Checking for Water Pump




```
The detected noise type is: Water Pump
```

4) Checking for Traffic Sounds




```
The detected noise type is: City Traffic
```

Therefore, we can use the concept of correlation to find the similarity between the two signals. By the use of cross-correlation, we are finally able to find the type of noise associated with each given input file.