EEN G-320 Controls Systems

Term Project 1

Members: Christian Calvo (1297165), and Erik Schlosser (1277551)

Due Date: 12/3/24

Git Repo: https://github.com/Tubliy/Proj1-Control-Systems.git

Objective:

The objective of this project is to design an active noise cancellation system. Using an IDE of our choice to produce 30 seconds of music, and merging it with a noise. The noise will occur at a designated time, something will be able to detect it, and the feedback control then cancels it out. Finally, a clean version will be produced and we'll be able to analyze it with a set of graphs we achieved from the combination of the audios.

Design Parameters:

- Input: There's a total of two inputs, the original song, and a noise.
- Output: cleanSong.wav after the song was successfully filtered.
- Noise/Disturbance: A short alarm noise played at a certain point in the song.
- Feedback: A loop that reads through the file constantly until the noise is cancelled.

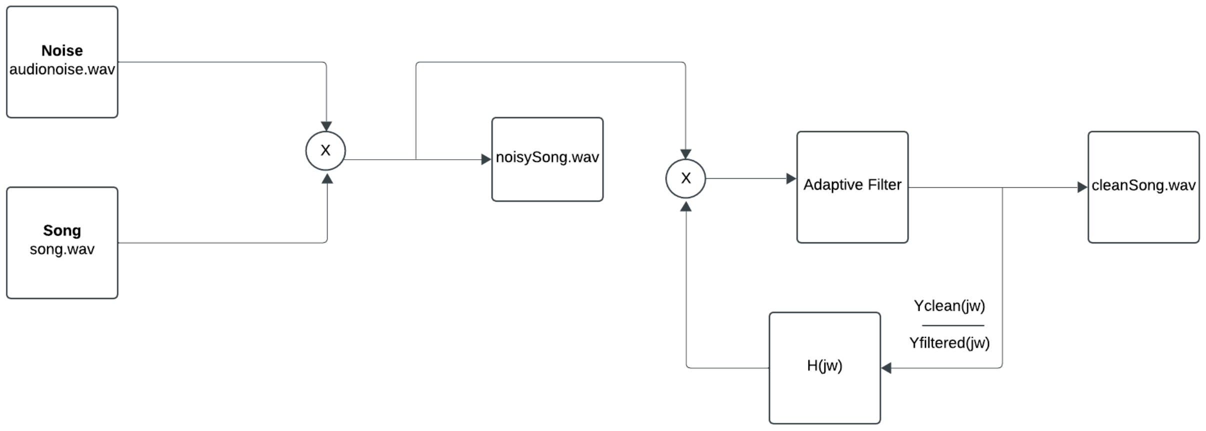


Figure 1A: Block Diagram of Noise Cancellation System.

Procedure:

DarthVaderThemeSong.m (Music Track code)

This file was used to create the music track, our sampling rate was $FS = 8000\ hz$. Which is an easier rate to work with and almost default when doing similar things. When defining the frequencies of the notes, this website helped translate a certain note to its respective frequency. The $struc(...)$; was used to define the data types, for example in the first block that uses it

```
freq = struct( ...
    'A', 440, ...          % A4
    'A_low', 220, ...      % A3
    'F', 349.23, ...       % F4
```

```
        'C_high', 523.25, ... % C5
        'E_high', 659.25, ... % E5
        'D_high', 587.33, ... % D5
        'B', 493.88, ...     % B4
        'G', 392 ...         % G4
    );
```

This gives A a frequency of 440, which when the notes are created it's called to by using freq.A. The notes were then created with the equation of a sine wave $y(t) = sin(2\pi \times freq.ofNote \times (0{:}\frac{1}{FS}{:}\,duration.ofNote))$. The duration is added into a time vector where it starts at t = 0 and goes up in steps of $\frac{1}{8000}$ to the duration of the given note. The output was stored in variables that defined the actual notes $Aquarter = sin(2\pi \times freq.A \times (0{:}\frac{1}{FS}{:}\,duration.quarter))$. After creating the notes, they are stored into a melody array that when called plays the song. However, the melody didn't match the music track that was expected, so a bass array was added for harmony. After, storing it in the song variable, the *audiowrite* function was used to export it as a .wav file.

audioMerge.m (Merging with noise disturbance)

The file mentioned was used to merge the noise and the music track together. Matching sampling rates if they differed, as well as adjusting the size of the noise to match the music track. Even if it was longer or shorter than the track, it was either trimmed or zeros were added as silent noise. It also converted the noise to mono, meaning it gave it a one column vector for audio data. But later turned it back to stereo if the music track was in that format.

```
noisySong = song;
noisySong(FS_Index:FS_Index + length(noise) - 1, :) = ...
   noisySong(FS_Index:FS_Index + length(noise) - 1, :) + noise;
```

$FSIndex$ is the starting index of where the noise will be added to in the output noisySong.wav. $FSIndex + length(noise) - 1$ this is the ending index of the section, it has -1 to ensure that the noise will stay in between the bounds. The + noise just adds the noise in as an element. The track is then exported as noisySong.wav.

audioCancel.m (Noise cancellation of disturbance)

This file was used to cancel out the noise in noisySong.wav using a feedback loop that keeps running until the noise is eliminated. This is shown in the code below:

```
% Set a threshold for residual noise
threshold = 0.001;
% Initial residual to enter the loop
residual = inf;
% Start the feedback loop
while residual > threshold
```

```
    % Perform FFT
    Y_clean = fft(y_clean);
    Y_filtered = fft(y_filtered);
    % Calculate Transfer Function
    H = Y_clean ./ Y_filtered;
    % Apply Noise Cancellation
    Y_filtered= H .* fft(y_noisy);
    % Inverse FFT
    y_filtered = ifft(Y_filtered);
```

Until the remaining noise or error is below the threshold, the loop keeps canceling out noise. represents the frequency domain filter or transfer function. It is computed as the ratio of the spectrums of the filtered signal (Y_filtered) and the clean signal (Y_clean). The goal of this operation is to figure out how to adjust the noisy signal so that it approaches the clean signal. y_noisy The time-domain noisy signal.The noisy signal is converted to the frequency domain using fft(y_noisy). To lessen the noise and obtain an updated filtered signal in the frequency domain, multiply H by the noisy signal in the frequency domain (H.* fft(y_noisy)). After that, the time domain is restored. Finally, the residue is calculated and the cleanSong.wav is exported.

timeandfreqPlot.m (Plotting the time and frequency domain)

The code below was used to plot the needed graphs, and below what it does is explained.

```
% Create a new figure
figure;
% Plot for 'song.wav'
[y, Fs] = audioread('song.wav');
duration = length(y) / Fs;
t = linspace(0, duration, length(y));
Y = fft(y);
freq_resolution = Fs / length(Y);
f = linspace(0, Fs - freq_resolution, length(Y));
% Time domain of song.wav
subplot(4, 3, 1);
plot(t, y);
title('song.wav - Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
% Frequency domain of song.wav
subplot(4, 3, 2);
plot(f, abs(Y));
title('song.wav - Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0, Fs/2]);
% Spectrogram of song.wav
subplot(4, 3, 3);
spectrogram(y, 256, [], [], Fs, 'yaxis');
title('song.wav - Spectrogram');
colorbar;
```

Here, Fs is the file sampling rate and y is the audio data. The length of y divided by the sampling rate yields the duration. Time "t" begins at 0 and advances in increments of the computed duration until it reaches the length of y. Thus, y vs. t is the time domain plot.

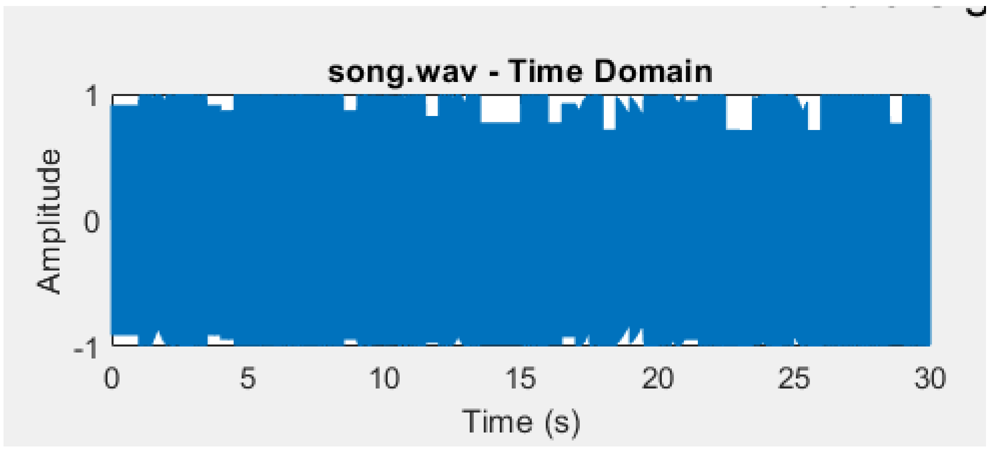


Figure 2A: Time domain plot.

A function's fourier transform can be found using the fft method. The trans form can be stored in Y thanks to the line of code Y = fft(y);. The identical code from the "duration" field is used for frequency resolution. Beginning at zero, frequency "f" progresses in increments of the determined freq_resolution until it reaches the length of Y(jw). Thus, Y(jw) vs. f is the time frequency domain plot.

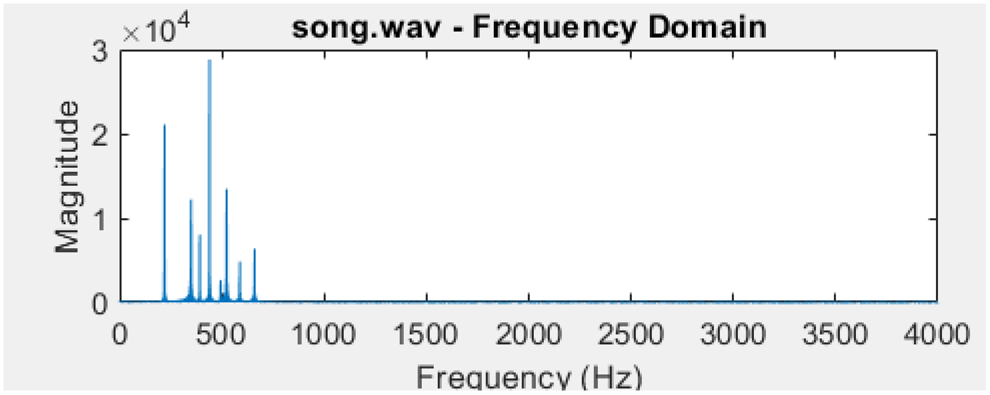


Figure 2B: Frequency domain plot.

Lastly, x represents time and y represents frequency in the spectrogram plot. The graph's hue indicates a signal's strength or amplitude at a given moment in time. Higher amplitudes are indicated by darker colors, and vice versa. Nothing significant is included in the plot's code; the []s are standard Matlab requirements, and the window size is 256.

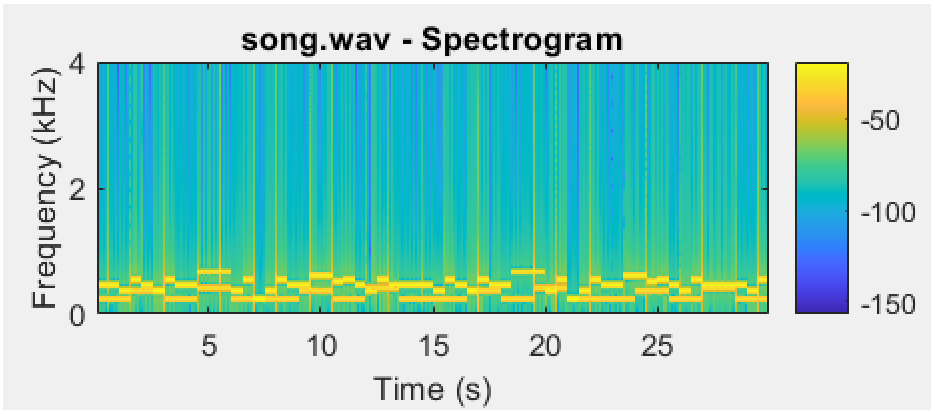


Figure 2C: Spectrogram.

Member Contributions:

Christian Calvo - Wrote the objective,procedure, audioMerge.m code, README.txt, and Appendix.

Erik Schlosser - Wrote the analysis of the graphs, audioCancel.m code, and LavendarTown.M code.

Works Cited:

Musescore. “The Imperial March.” *Musescore.com*, Musescore, 24 May 2012, www.musescore.com/user/39593079/scores/15455794. Accessed 30 Nov. 2024.

"Note Frequencies." *Muted.io*, www.muted.io/note-frequencies/. Accessed 30 Nov. 2024.

"Spectrogram." *MathWorks*, www.mathworks.com/help/signal/ref/spectrogram.html.

Accessed 30 Nov. 2024.

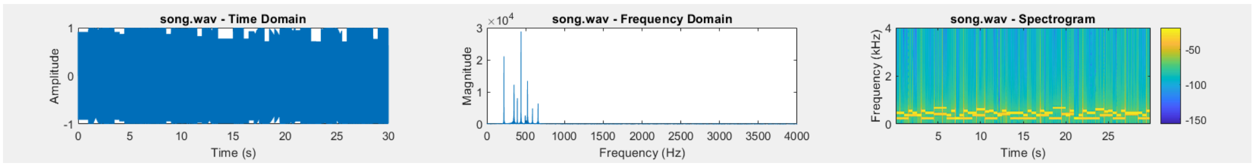
"Signal Processing Tutorial Launchpad." *MathWorks*, www.mathworks.com/academia/student_center/tutorials/signal-processing-tutorial-launch

pad.html. Accessed 30 Nov. 2024.

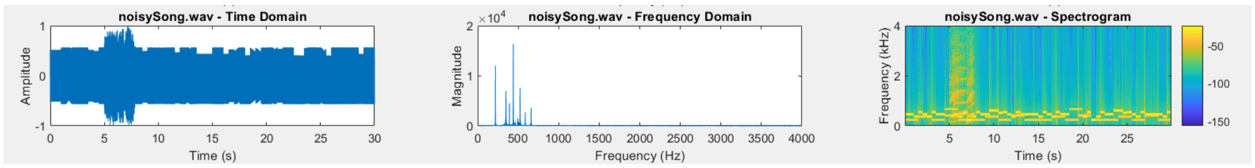
"MATLAB Tutorial Launchpad." *MathWorks*,

www.mathworks.com/academia/student_center/tutorials/mltutorial_launchpad.html?confirmation_page#. Accessed 30 Nov. 2024.

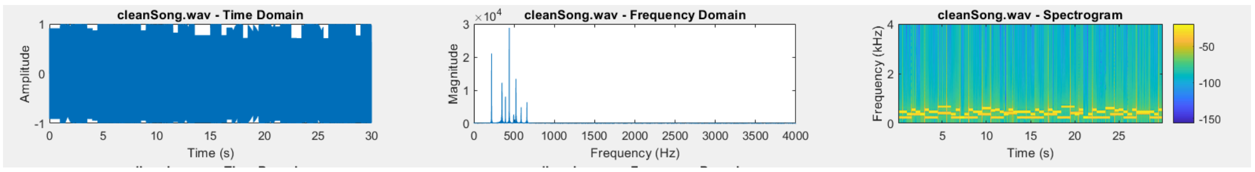
Analysis:



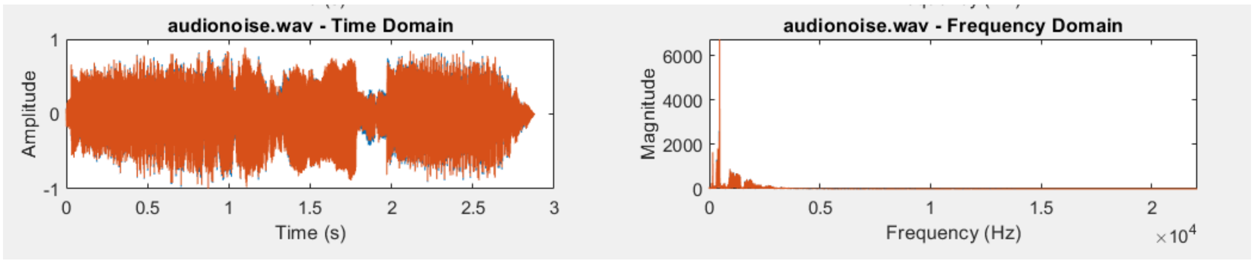
Time,Frequency, and Spectrogram of song.wav (**Figure 3A**)



Time,Frequency, and Spectrogram of noisySong.wav (**Figure 3B**)



Time,Frequency, and Spectrogram of cleanSong.wav (**Figure 3C**)



Time and Frequency domain of audionoise.wav (**Figure 3D**)

Figure 3A's time-domain graph displays a broad, continuous signal spanning 30 seconds; amplitude variations occur. This graph shows that the signal is clear and free of faults that could cause interruptions or other issues. In contrast, there is a very obvious flaw in Figure 3B between the music's 5- and 10-second marks. The noise we added to the song is what caused this irregularity. We used a feedback loop and noise reduction to eliminate this anomaly in Figure 3C. The outcome was flawless, and the loud song's noise was entirely muffled. This was caused by our while loop, which continued to run until the noise was entirely eliminated. It used the fourier transforms of both noisySong.wav and the regular song.wav, then found the transfer function which was $H(jw) \ = \ \frac{Yclean(jw)}{Yfiltered(jw)}$ used to compare and adaptively modify the noisy signal.

Figures 3A, 3B, and 3C's frequency-domain plots clearly compare the effects of noise on the signal and its mitigation. The original song's clear, harmonic substance is represented by the distinct, sharp peaks in Figure 3A at particular frequencies. Figure 3B, on the other hand, illustrates the impact of the extra noise, with wider, less distinct peaks and more energy dispersed over the spectrum. The original frequencies are obscured by this noise, which makes it more difficult to decipher the signal. Following noise cancellation, Figure 3C shows a notable decrease in these undesired elements. The fact that the peaks are sharper and more like those in Figure 3A shows that much of the noise has been successfully reduced without compromising the original harmonic's integrity.

With the notes at the beginning signifying the melody and the notes at the end signifying the harmony, each of the three spectrograms clearly delineates each musical segment. Spectrogram 3A is flawless and displays the music playing as it should. However, as with the time-domain representation, the noise is seen in Spectrogram 3B. The boundary between the noise and silence is distinct. The majority of the noise has been eliminated, according to Spectrogram 3C, while some residue is still present. There are noticeably more luminous regions close to the notes, which suggests filter artifacts.

Appendix:

Matlab Code 1: (DarthVaderThemeSong.m)

```
clear; % Remove all variables from the workspace
% Sampling frequency
FS = 8000; % Sampling frequency in Hz
% Frequencies for the Imperial March
freq = struct( ...
   'A', 440, ...         % A4
   'A_low', 220, ...     % A3
   'F', 349.23, ...      % F4
   'C_high', 523.25, ... % C5
   'E_high', 659.25, ... % E5
   'D_high', 587.33, ... % D5
   'B', 493.88, ...      % B4
   'G', 392 ...          % G4
);
% Note durations (in seconds)
duration = struct(...
'quarter',  0.5, ...
'half',  1.0,...
'eighth',  0.25,...
'dotted_quarter',  0.75...
);
Aquarter = sin(2*pi*freq.A*(0:1/FS:duration.quarter));
Fquarter = sin(2*pi*freq.F*(0:1/FS:duration.quarter));
C_highquarter = sin(2*pi*freq.C_high*(0:1/FS:duration.quarter));
Ahalf = sin(2*pi*freq.A*(0:1/FS:duration.half));
E_highquarter = sin(2*pi*freq.E_high*(0:1/FS:duration.quarter));
A_lowquarter = sin(2*pi*freq.A_low*(0:1/FS:duration.quarter));
D_highquarter = sin(2*pi*freq.D_high*(0:1/FS:duration.quarter));
Bquarter = sin(2*pi*freq.B*(0:1/FS:duration.quarter));
Gquarter = sin(2*pi*freq.G*(0:1/FS:duration.quarter));
melody = [
   Aquarter, Aquarter, Fquarter, C_highquarter,...
   Aquarter, Fquarter, C_highquarter, Ahalf,...
   E_highquarter, E_highquarter, E_highquarter,...
   Fquarter, C_highquarter, A_lowquarter,...
   Fquarter, C_highquarter, Ahalf,...
   D_highquarter, D_highquarter, Bquarter,...
   C_highquarter, Aquarter, Fquarter,...
   C_highquarter, Ahalf,...
];
% Bass line (optional for more depth)
bass = [

   A_lowquarter, A_lowquarter, A_lowquarter,...
   Fquarter,Fquarter, Fquarter,...
   A_lowquarter, A_lowquarter, A_lowquarter,...
   Gquarter, Gquarter, Gquarter,...
   A_lowquarter, A_lowquarter, A_lowquarter,...
];
% Combine melody and bass (bass is lower in volume for harmony)
bass = bass * 0.5; % Lower bass volume
song = [melody + [bass, zeros(1, length(melody) - length(bass))]];
% Normalize song to avoid clipping
song = song / max(abs(song));
% Write the final music to a .wav file
audiowrite('song.wav', song, FS);
% Playback
audio = audioplayer(song, FS);
play(audio);
```

Matlab Code 2: (audioMerge.m)

```
% Load audio files
[song, Fs_s] = audioread('song.wav');
[noise, Fs_n] = audioread('audionoise.wav');
% Convert noise to mono if necessary
if size(noise, 2) > 1
   noise = mean(noise, 2);
end
% Resample the disturbance signal to match the song's sampling rate
if Fs_n ~= Fs_s
   try
       % Use resample if Signal Processing Toolbox is available
       noise = resample(noise, Fs_s, Fs_n);
   catch
       % Alternative resampling using interp1
       disp("Signal Processing Toolbox not installed");
       t_original = (0:length(noise)-1) / Fs_n; % Original time vector
       t_target = (0:1/Fs_s:(length(noise)-1)/Fs_n)'; % Target time vector
       noise = interp1(t_original, noise, t_target, 'linear'); % Linear
interpolation
   end
end
% Specify the starting time for noise in seconds
startTimeofNoise = 10;
% Calculate the starting index for adding the disturbance
FS_Index = round(startTimeofNoise * Fs_s);
% Ensure the disturbance signal fits within the music signal length
if FS_Index + length(noise) > length(song)
   noise = noise(1:length(song) - FS_Index); % Trim noise
elseif FS_Index + length(noise) < length(song)
   noise = [noise; zeros(length(song) - (FS_Index + length(noise)), 1)]; % Pad
noise
end
% Match channels if music is stereo
```

```
if size(song, 2) > 1 && size(noise, 2) == 1
   noise = [noise, noise]; % Convert mono to stereo
end
% Add noise to the music
noisySong = song;
noisySong(FS_Index:FS_Index + length(noise) - 1, :) = ...
   noisySong(FS_Index:FS_Index + length(noise) - 1, :) + noise;
% Normalize the combined audio to prevent clipping
noisySong = noisySong / max(abs(noisySong), [], 'all');
% Save the combined audio as a new file
audiowrite('noisySong.wav', noisySong, Fs_m);
% Displaying Sampling rates for Procedure
disp('Sampling rate of Fs_s');
disp(Fs_s);
disp('Sampling rate of Fs_n')


disp(Fs_n);
disp('Audio processing completed and saved as noisySong.wav');
```

Matlab Code 3: (audioCancel.m)

```
% Load audio files
[y_clean, Fs_clean] = audioread('song.wav'); % Reference clean audio
[y_noisy, Fs_noisy] = audioread('noisySong.wav'); % Noisy audio to be cleaned
% Initialize processed audio as the noisy audio
y_filtered = y_noisy;
% Set a threshold for residual noise
threshold = 0.001;
% Initial residual to enter the loop
residual = inf;
% Start the feedback loop
while residual > threshold
   % Perform FFT
   Y_clean = fft(y_clean);
   Y_filtered = fft(y_filtered);
   % Calculate Transfer Function
   H = Y_clean ./ Y_filtered;
   % Apply Noise Cancellation
   Y_filtered= H .* fft(y_noisy);
   % Inverse FFT
   y_filtered = ifft(Y_filtered);
   % Calculate residual noise
   residual = norm(y_filtered - y_clean) / norm(y_clean); % Using norm as a
measure of difference
```

```
end
% Write the final processed audio to a new file
audiowrite("cleanSong.wav", y_filtered, Fs_noisy);
% Display a message indicating the completion of the process
disp('Processed audio file with feedback until clean saved as cleanSong.wav');
```

Matlab Code 4: (timeandfreqPlot.m)

```
% Create a new figure
figure;
% Plot for 'song.wav'
[y, Fs] = audioread('song.wav');
duration = length(y) / Fs;
t = linspace(0, duration, length(y));
Y = fft(y);
freq_resolution = Fs / length(Y);
f = linspace(0, Fs - freq_resolution, length(Y));
% Time domain of song.wav
subplot(4, 3, 1);
plot(t, y);
title('song.wav - Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
% Frequency domain of song.wav
subplot(4, 3, 2);
plot(f, abs(Y));
title('song.wav - Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0, Fs/2]);
% Spectrogram of song.wav
subplot(4, 3, 3);
spectrogram(y, 256, [], [], Fs, 'yaxis');
title('song.wav - Spectrogram');
colorbar;


% Plot for 'noisySong.wav'
[y, Fs] = audioread('noisySong.wav');
duration = length(y) / Fs;
t = linspace(0, duration, length(y));
Y = fft(y);
freq_resolution = Fs / length(Y);
f = linspace(0, Fs - freq_resolution, length(Y));
```

```
% Time domain of noisySong.wav
subplot(4, 3, 4);
plot(t, y);
title('noisySong.wav - Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
% Frequency domain of noisySong.wav
subplot(4, 3, 5);
plot(f, abs(Y));
title('noisySong.wav - Frequency Domain');
xlabel('Frequency (Hz)');


ylabel('Magnitude');
xlim([0, Fs/2]);
% Spectrogram of noisySong.wav
subplot(4, 3, 6);
spectrogram(y, 256, [], [], Fs, 'yaxis');
title('noisySong.wav - Spectrogram');
colorbar;
% Plot for 'cleanSong.wav'
[y, Fs] = audioread('cleanSong.wav');
duration = length(y) / Fs;
t = linspace(0, duration, length(y));
Y = fft(y);
freq_resolution = Fs / length(Y);
f = linspace(0, Fs - freq_resolution, length(Y));
% Time domain of cleanSong.wav
subplot(4, 3, 7);
plot(t, y);
title('cleanSong.wav - Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');
% Frequency domain of cleanSong.wav
subplot(4, 3, 8);
plot(f, abs(Y));
title('cleanSong.wav - Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0, Fs/2]);
% Spectrogram of cleanSong.wav
subplot(4, 3, 9);


spectrogram(y, 256, [], [], Fs, 'yaxis');
title('cleanSong.wav - Spectrogram');
colorbar;
% Plot for 'audiodisturb.wav'
[y, Fs] = audioread('audionoise.wav');
```

```
duration = length(y) / Fs;
t = linspace(0, duration, length(y));
Y = fft(y);
freq_resolution = Fs / length(Y);
f = linspace(0, Fs - freq_resolution, length(Y));
% Time domain of audionoise.wav
subplot(4, 3, 10);
plot(t, y);
title('audionoise.wav - Time Domain');
xlabel('Time (s)');
ylabel('Amplitude');


% Frequency domain of audionoise.wav
subplot(4, 3, 11);
plot(f, abs(Y));
title('audionoise.wav - Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0, Fs/2]);
% Add a global title for the entire figure
sgtitle('Audio Signal Analysis: Time, Frequency, and Spectrogram Domains');
```