# Project Report

**EENG 341 Signals and Systems**

**Term Project - Fall 2024**

**Team Members:** Imran Kadir and Pavan Kanakkassery

## Objective

The objective of this project is to create, analyze, and visualize a music signal using signal processing techniques. Specifically, our group used the "Happy Birthday" song. It was composed using sinusoidal waveforms to represent musical notes, then analyzed in the time and frequency domains, and a spectrogram was generated. Extra credit tasks, including the addition and filtering of noise, were optionally explored.

## Procedure

1. **Music Composition:**
   1. Each musical note (C, D, E, etc.) was represented as a sine wave with its corresponding frequency.
   2. Short notes were defined with a duration of 0.4 seconds and longer notes were set to 0.8 seconds.
   3. Pauses were implemented using a zero vector to add silence between notes.
   4. The "Happy Birthday" song was constructed by combining notes in the correct sequence.
2. **Analysis:**
   1. Time-domain visualization was plotted to observe the waveform of the composed music.
   2. Frequency-domain analysis was performed using FFT to identify the frequency components.
   3. A spectrogram was generated to visualize how the frequency content of the signal varied over time.
3. **Output:**
   1. The song was saved as a .wav file and played back to verify correctness.
   2. Graphical representations of time-domain, frequency-domain, and spectrogram plots were created.

## Time-Domain Analysis

The time-domain graph (Figure 1) shows the amplitude of the song's audio signal throughout its total duration, with the x-axis representing Time (s) and the y-axis representing Amplitude. Since the notes (C, D, E, etc.) are generated using a sine wave where f is the frequency, the sine wave oscillates between -1 and 1. This can be seen in the peaks and troughs of the graph. The frequency of the oscillations corresponds to the note being played at that time. For example, a note C, completes 261.63 cycles per second, leading to a higher density of oscillations in a short time. The notes last 0.4 seconds as per the code, and the oscillations can be observed for that duration before a pause. The pauses, representing silence in the audio, appear as flat segments at zero amplitude. The signal was down-sampled for better visualization, as it was originally represented as solid bars from -1 to 1 when viewing the graph fully, rather than as viewable waveforms. The down-sampled output still accurately represents the original time-domain signal of the song.

## Frequency-Domain Analysis

The frequency-domain graph (Figure 2) shows the magnitude of each of the frequency components in the audio signal, with the x-axis representing Frequency (Hz) and the y-axis representing Magnitude. This graph is generated using the Fourier Transform, which breaks down the time-domain signal into its frequency components. The peaks in the graph correspond to the frequencies of the musical notes (e.g., 261.63 Hz for C, 293.66 Hz for D) and their harmonics. The magnitude represents the strength of each frequency in the signal, so the peaks indicate the most dominant frequencies corresponding to the notes played in the song.

In the graph, the most dominant frequency is 261.63 Hz, which corresponds to the C note. The pauses between the notes result in less overall energy in the frequency domain during those segments, which is reflected by the low magnitude values. This graph accurately represents the song by highlighting the dominant frequencies of each note, confirming the accuracy of the generated music and providing insight into the harmonic content of the signal.

## Spectrogram Analysis

The spectrogram (Figure 3) provides a detailed time-frequency representation of the signal, with the x-axis representing Time (s) and the y-axis representing Frequency (kHz). This visualization shows how the frequency content of the signal evolves over time. The alternating bands of activity correspond to the different musical notes, while the inactivity represents pauses or silence.

The color in the spectrogram indicates the amplitude of the frequency components, with bright yellow lines representing high energy and showing the most dominant frequencies of the notes. Any harmonics are visible as additional lines. The darker blue areas represent low energy or silence. By examining the bright yellow lines, you can see which note is being played at each section of the song, along with its frequency. The pauses between notes are depicted as dark blue segments.

Each note lasts for 0.4 seconds, as set by the code, and this duration can be visually identified in the spectrogram. This graph accurately represents the song, showing the transition from one note to the next with pauses in between, providing a clear and comprehensive view of the signal's time-frequency characteristics.

## Designing a Filter to Remove Noise

When attempting to remove the noise added to the music signal, our team researched different types of filters. During this process, we came across the Butterworth filter, which is widely used in signal processing for its smooth frequency response.

### Objective

To design a filter that effectively removes high-frequency noise while preserving the main components of the music signal.

### Research Process

Initially, our understanding of filtering was limited. Through online resources, including MATLAB documentation and academic articles, we learned about the properties of the Butterworth filter:

* It is an infinite impulse response (IIR) filter.
* It provides a maximally flat frequency response in the passband, which is ideal for retaining the signal's original characteristics without distortion.

We chose the Butterworth filter over other types, such as Chebyshev and Elliptic filters, due to its smooth frequency response and minimal phase distortion, which are crucial for preserving the quality of the music signal.

**Signal-to-Noise Ratio**: The signal-to-noise ratio (SNR) measures the amount of signal power compared to the noise power. A higher SNR indicates a cleaner signal with less noise. To successfully add white Gaussian noise, we set the desired SNR value to 10 dB for the noisy signal. This means the noise power is adjusted so that the signal power is 10 times greater than the noise power. With an SNR of 10 dB, the signal remains recognizable but with noticeable noise.

### Methodology

* We designed a 5th-order Butterworth low-pass filter using MATLAB's butter function.
* The cutoff frequency was set to 1000 Hz based on the spectral analysis of the original and noisy signals.
* The filter was applied to the noisy signal using the filter function.

### Results

* Time-Domain Analysis for Noisy Signal:   
  The time-domain graph (Figure 4) of the noisy signal reveals the same underlying waveform as the original signal. However, it is evident that the noisy waveform has additional random fluctuations, indicating the presence of white Gaussian noise. The noisy signal oscillates between -1 and 1, similar to the original, but the added random noise makes the waveform appear less distinct and more erratic. These fluctuations obscure the clarity of the original signal, making it harder to discern the individual notes. Despite this, the overall structure of the signal remains recognizable, allowing for effective noise reduction through filtering.
* Frequency-Domain Analysis for Noisy Signal:

The frequency-domain graph (Figure 5) of the noisy signal still displays the same fundamental peaks, indicating the most dominant frequencies corresponding to the notes played in the original song. However, there is a noticeable increase in noise in the baseline of the signal. This increase is due to the addition of white Gaussian noise, which has a uniform distribution across all frequencies, thereby adding noise throughout the entire frequency spectrum.

The peaks in the frequency-domain plot represent the primary frequencies of the musical notes, such as 261.63 Hz for the C note. Despite the presence of these peaks, the baseline noise is elevated, which can obscure the clarity of the signal. This added noise is evident as a continuous, elevated level across the frequency spectrum, making it harder to distinguish the individual notes.

By analyzing the frequency-domain plot, we can see that the white Gaussian noise introduces high-frequency components that were not present in the original signal. This analysis confirms the need for an effective filtering technique to remove these unwanted high-frequency components while preserving the essential frequencies of the music signal.

* Time-Domain Analysis for Filtered Signal:   
  The time domain of the filtered signal (Figure 6) is smoothed out indicating that some higher frequency noise was effectively removed while still retaining the overall signal shape. The amplitude is still in between the ranges of -1 and 1 similar to the original signal.
* *Frequency-Domain Analysis for filtered Signal:*

The frequency domain of the filtered signal (Figure 7) shows the same fundamental peaks as the original signal. However, with the inclusion of the Butterworth filter, it successfully attenuates frequencies above the cutoff frequency (1000 Hz) and reduces the noise while preserving the more important lower frequencies that are used. The result of using the Butterworth filter is a cleaner signal with less noise.

### Implementation Details

During the implementation, selecting the appropriate filter order and cutoff frequency was crucial. We experimented with different values to achieve optimal noise reduction without distorting the original signal.

Comparison with Other Filters

Compared to other filters, the Butterworth filter provided a good balance between noise reduction and signal preservation. Other filters, such as Chebyshev, introduced more ripple in the passband, which was undesirable for our application.

### Future Improvements

Future work could involve experimenting with different filter types or orders to further optimize noise reduction. Additionally, adaptive filtering techniques could be explored to dynamically adjust the filter parameters based on the signal characteristics.

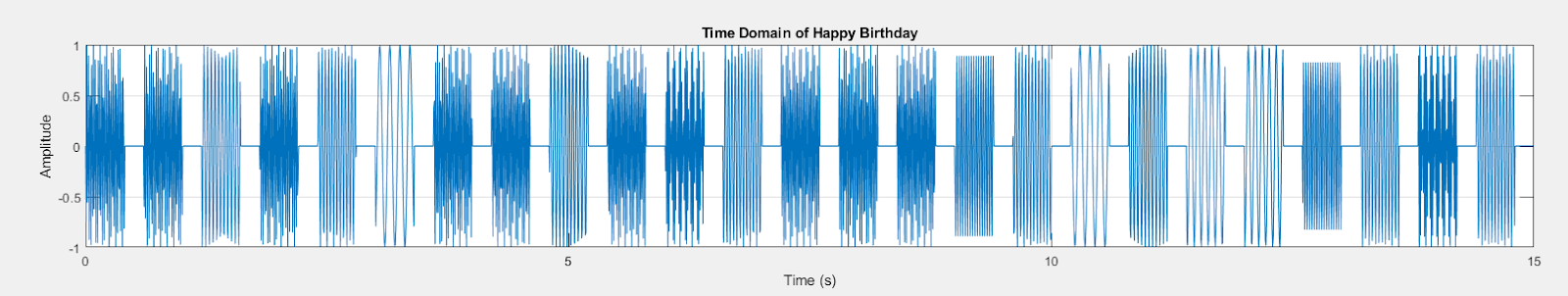
## Team Member Contributions

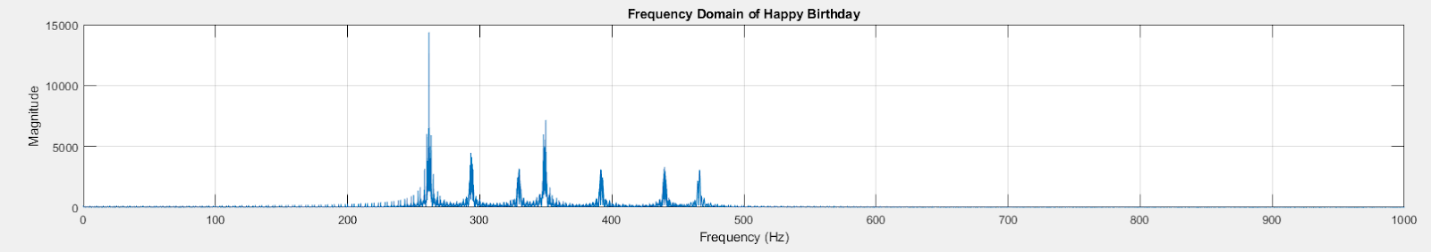
* **Imran:** Code implementation for generating music and visualization.
* **Pavan:** Assisted with analyzing results and documentation.

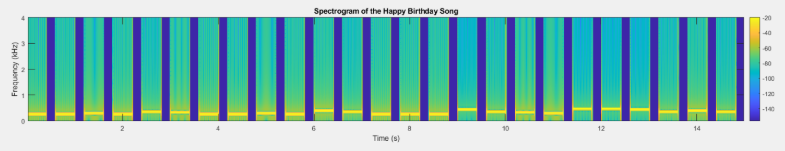
## Conclusion

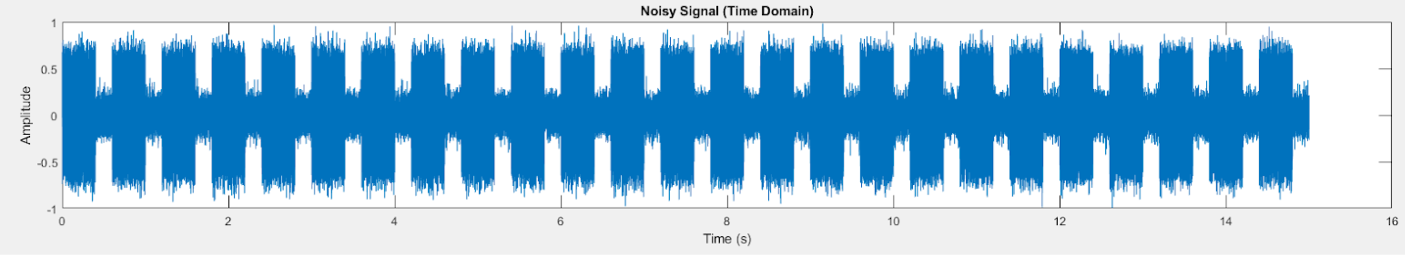
This project successfully demonstrated the process of music signal generation and analysis using MATLAB. The "Happy Birthday" song was composed, visualized, and analyzed in both time and frequency domains, with a detailed spectrogram analysis. The project highlights the practical application of signal-processing techniques in audio engineering.

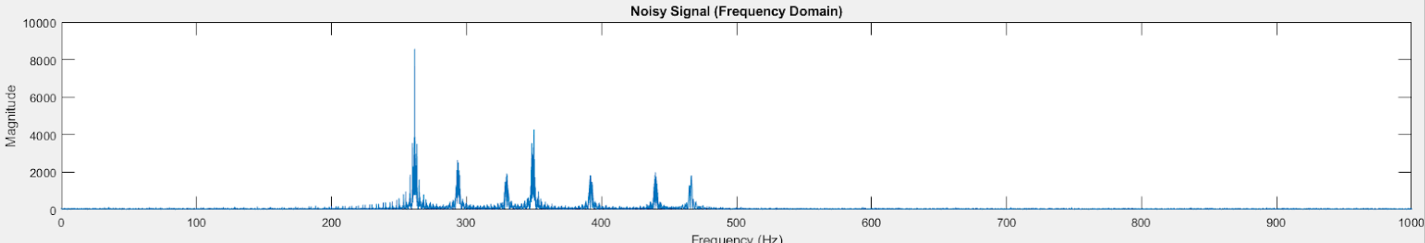
## Figures

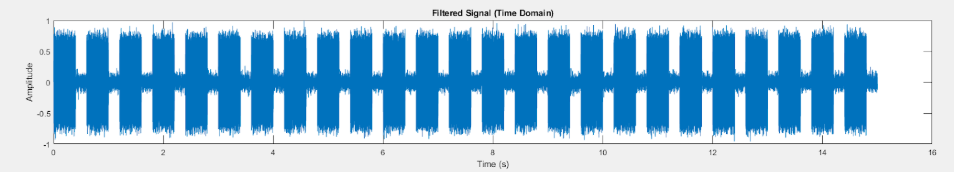
**Figure 1:** Time-Domain Representation of the "Happy Birthday" Song

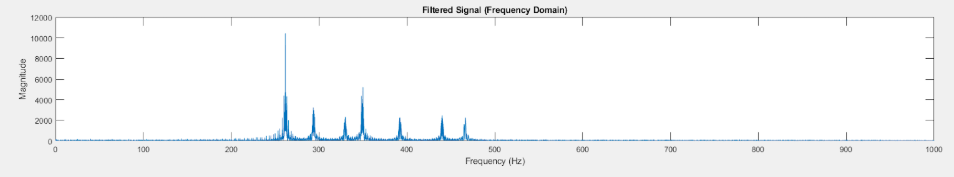
**Figure 2:** Frequency-Domain Representation of the "Happy Birthday" Song

**Figure 3:** Spectrogram of the "Happy Birthday" Song

**Figure 4:** Time-Domain Representation of the Noisy "Happy Birthday" Song

**Figure 5:** Frequency-Domain Representation of the Noisy "Happy Birthday" Song

**Figure 6:** Time-Domain Representation of the Filtered "Happy Birthday" Song

**Figure 7:** Frequency-Domain Representation of the Noisy "Happy Birthday" Song

## Code Appendix

The full code for the project is included below. Comments in the code provide explanations of each module.  
  
CODE FOR THE SONG

clear; % Remove all variables from the workspace

% Sampling frequency

FS = 8000;

% Define the notes with durations > Both

C = sin(2\*pi\*261.63\*(0:1/FS:0.4)); % C note (short)

D = sin(2\*pi\*293.66\*(0:1/FS:0.4)); % D note (short)

E = sin(2\*pi\*329.63\*(0:1/FS:0.4)); % E note (short)

F = sin(2\*pi\*349.23\*(0:1/FS:0.4)); % F note (short)

G = sin(2\*pi\*392.00\*(0:1/FS:0.4)); % G note (short)

A = sin(2\*pi\*440.00\*(0:1/FS:0.4)); % A note (short)

B = sin(2\*pi\*493.88\*(0:1/FS:0.4)); % B note (short)

Bb = sin(2\*pi\*466.16\*(0:1/FS:0.4)); % Bb note (short)

% Define longer duration notes for some parts of the song > Both

C\_long = sin(2\*pi\*261.63\*(0:1/FS:0.8)); % C note (longer)

D\_long = sin(2\*pi\*293.66\*(0:1/FS:0.8)); % D note (longer)

E\_long = sin(2\*pi\*329.63\*(0:1/FS:0.8)); % E note (longer)

F\_long = sin(2\*pi\*349.23\*(0:1/FS:0.8)); % F note (longer)

G\_long = sin(2\*pi\*392.00\*(0:1/FS:0.8)); % G note (longer)

A\_long = sin(2\*pi\*440.00\*(0:1/FS:0.8)); % A note (longer)

B\_long = sin(2\*pi\*493.88\*(0:1/FS:0.8)); % B note (longer)

Bb\_long = sin(2\*pi\*466.16\*(0:1/FS:0.8)); % Bb note (longer)

% Silence (pause between notes) > Pavan

pause = zeros(1, FS\*0.2); % 0.2 seconds of silence

% Assemble the song using the sequence of notes > Both

line1 = [C, pause, C, pause, D, pause, C, pause, F, pause, E, pause]; % Happy birthday to you

line2 = [C, pause, C, pause, D, pause, C, pause, G, pause, F, pause]; % Happy birthday to you

line3 = [C, pause, C, pause, C, pause, A, pause, F, pause, E, pause, D, pause]; % Happy birthday dear [Name]

line4 = [Bb, pause, Bb, pause, A, pause, F, pause, G, pause, F, pause]; % Happy birthday to you

% Combine all lines to form the complete song > Both

song = [line1, line2, line3, line4];

% Save the song as a .wav file > Both

audiowrite('Happy\_Birthday.wav', song, FS);

% Play the song > Both

[y, FS] = audioread('Happy\_Birthday.wav');

sound(y, FS);

disp('Playing Happy Birthday Song');

disp('Song audio saved as "Happy\_Birthday.wav".');

% Generate the time vector for the signal > Imran

t = (0:length(y)-1) / FS;

% Create a figure with three subplots

figure;

% Downsample the signal for better visualization > Imran

downsample\_factor = 50; % Reduce data points by a factor of 50 > Imran

y\_downsampled = y(1:downsample\_factor:end); % Take every 50th point > Imran

t\_downsampled = t(1:downsample\_factor:end); % Adjust the time vector > Imran

% 1. Time Domain Plot > Imran

subplot(3, 1, 2);

plot(t\_downsampled, y\_downsampled); % Plot the downsampled data

xlim([0 15]);

xlabel('Time (s)');

ylabel('Amplitude');

title('Time Domain of Happy Birthday');

grid on;

% 2. Frequency Domain Plot > Imran

N = length(y); % Number of samples

f = (0:N-1) \* (FS / N); % Frequency vector

fft\_y = abs(fft(y)); % Compute magnitude of FFT

subplot(3, 1, 3);

plot(f(1:floor(N/2)), fft\_y(1:floor(N/2))); % Only plot positive frequencies

xlim([0 1000]); % Limit x-axis to 0-1000 Hz

xlabel('Frequency (Hz)');

ylabel('Magnitude');

title('Frequency Domain of Happy Birthday');

grid on;

% 3. Spectrogram Plot > Both

subplot(3, 1, 1);

spectrogram(song, 256, 250, 256, FS, 'yaxis'); % Window length, overlap, FFT size, sampling frequency

title('Spectrogram of Happy Birthday');

colorbar;

CODE FOR THE FILTER:

clear; close all; clc;

% Load the original audio file > Pavan

[inputFile, FS] = audioread('Happy\_Birthday.wav');

originalSignal = inputFile(:, 1); % Use one channel if stereo

% Normalize the signal > Pavan

originalSignal = originalSignal / max(abs(originalSignal));

% Downsample factor > Imran

downsampleFactor = 50;

% Downsample the time domain signal > Imran

timeVector = (0:length(originalSignal)-1)/FS; % Ensure valid time vector > Pavan

timeVectorDownsampled = timeVector(1:downsampleFactor:end); % Downsample time vector > Imran

originalSignalDownsampled = originalSignal(1:downsampleFactor:end); % Downsample signal > Imran

% Plot the original signal in the time and frequency domains > Both

figure;

subplot(3, 1, 1);

plot(timeVectorDownsampled, originalSignalDownsampled); % Downsampled time domain > Imran

title('Original Signal (Time Domain)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

freq = linspace(0, FS, length(originalSignal));

originalFFT = abs(fft(originalSignal));

plot(freq(1:floor(end/2)), originalFFT(1:floor(end/2))); % Show only positive frequencies

xlim([0 1000]);

title('Original Signal (Frequency Domain)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

disp('Playing the original signal...');

% Original song > Imran

sound(inputFile, FS);

pause(length(inputFile)/FS + 1);

% a. Add white Gaussian noise to the signal > Pavan

snrValue = 10; % Desired signal-to-noise ratio in dB

noisySignal = awgn(originalSignal, snrValue, 'measured'); % Add noise

% Calculate the noise > Imran

noise = noisySignal - originalSignal;

% Normalize noisy signal to avoid clipping when saving > Pavan

noisySignal = noisySignal / max(abs(noisySignal));

% Save the noisy signal as a .wav file > Pavan

audiowrite('noisy\_output.wav', noisySignal, FS);

% Plot the noisy signal in time and frequency domains > Both

figure;

subplot(3, 1, 1);

plot(timeVector, noisySignal);

xlim([0 15]);

title('Noisy Signal (Time Domain)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

noisyFFT = abs(fft(noisySignal));

plot(freq(1:floor(end/2)), noisyFFT(1:floor(end/2))); % Show only positive frequencies

xlim([0 1000]);

title('Noisy Signal (Frequency Domain)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

% Calculate the Signal-to-Noise Ratio (SNR) > Imran

disp('Playing the noisy signal...');

calculatedSNR = snr(originalSignal, noise);

disp(['Signal-to-Noise Ratio of Noisy Signal: ', num2str(calculatedSNR), ' dB']);

% Play the noisy signal > Both

sound(noisySignal, FS);

pause(length(noisySignal)/FS + 1); % Wait for playback to finish

% b. Design a filter to remove noise > Pavan

% Using a low-pass filter (Butterworth)

filterOrder = 5;

cutoffFrequency = 1000; % Adjust based on signal characteristics (Hz)

[b, a] = butter(filterOrder, cutoffFrequency / (FS / 2), 'low'); % Low-pass filter

filteredSignal = filter(b, a, noisySignal);

% Normalize filtered signal to avoid clipping when saving > Pavan

filteredSignal = filteredSignal / max(abs(filteredSignal));

% Save the filtered signal as a .wav file > Both

audiowrite('filtered\_output.wav', filteredSignal, FS);

% Plot the filtered signal in time and frequency domains > Both

figure;

subplot(3, 1, 1);

plot(timeVector, filteredSignal);

xlim([0 15]);

title('Filtered Signal (Time Domain)');

xlabel('Time (s)');

ylabel('Amplitude');

subplot(3, 1, 2);

filteredFFT = abs(fft(filteredSignal));

plot(freq(1:floor(end/2)), filteredFFT(1:floor(end/2))); % Show only positive frequencies

xlim([0 1000]);

title('Filtered Signal (Frequency Domain)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

% Play the filtered signal > Pavan

disp('Playing the filtered signal...');

sound(filteredSignal, FS);

disp('Filtered audio saved as "filtered\_output.wav".');

disp('Noisy audio saved as "noisy\_output.wav".');