Faculty of Information Engineering and Technology
Department of Communication Engineering
Prof. Dr. Ahmed El-Mahdy

Lab Project
Audio Processing

Project Description

This project is classified into **4** parts:

Signal Generation Procedure

Part 1: The Pianist

In this part, you will generate your own signal by combining pairs of piano notes. Each note corresponds to a specific frequency, and you will generate a signal for each note based on the frequencies of the 3rd and 4th octaves. You will assume that both hands are playing at the same time, meaning you'll have pairs of notes (one from each octave) being played together.

The goal is to create a continuous sound by adding these note signals together. Each note is represented by a sine wave, and the signal for each note is active for a defined period of time. The total signal you generate will be a sum of all these individual note signals.

You can select the number of pairs of notes, N, and the specific frequencies F_i and f_i from the 3rd and 4th octaves, respectively. Each pair of notes will have a start time t_i and a duration T_i. The final signal is a sum of these individual note signals, each active during its respective time period.

Part 2: Song Generation

Once you've generated the individual note signals, you will combine them to form a complete song. The song is a superposition of the sine waves corresponding to the selected notes, and the combined signal will represent the full melody.

You will use the formula provided to generate this combined signal. The resulting signal will be a smooth combination of all the individual note signals played at the same time, forming a musical piece that you created. The length of the song and the notes you choose will determine the final sound produced by the signal.

This process will allow you to create a simple musical composition using the frequencies of piano notes from the 3rd and 4th octaves.

Noise Addition and Cancellation

Part 3: Add the Noise

In this part, you will simulate the presence of noise by adding two sinusoidal components to your original signal. These components represent unwanted noise at specific frequencies. Since you don't know exactly where the noise is added in the time domain, you generate it randomly. The noisy signal is simply the sum of the original signal and the noise.

The key point here is that you are introducing disturbances into the signal, but at this stage, you don't know which part of the signal contains the noise.

Part 4: Remove the Noise

To remove the noise, you will first convert the noisy signal into the frequency domain. By analyzing the frequency spectrum, you can identify peaks that correspond to the added noise. These peaks indicate the presence of noise at specific frequencies.

Your code should scan through the frequency spectrum and look for these high peaks. Once you identify the noisy frequencies, you can filter them out by zeroing out the corresponding frequency components. This leaves you with a cleaner signal, without the noise, in the frequency domain.

Finally, you will convert the filtered signal back to the time domain and listen to the result, which should now be the original signal with the noise removed.

This process allows you to remove noise without knowing exactly where it was added, relying on frequency domain analysis to identify and filter out the noise.

Project Guidelines

- Groups of **Three** students maximum (from the same Lab group, **NO** cross-labs allowed)
- **Copied reports or code means zero for both versions!**
- You should submit a hard copy of:
  - **Code:** Your Python script/s
  - **Report:**
    - **Time domain plots**: signal without noise, with noise, and after cancellation
    - **Frequency domain plots**: signal without noise, with noise, and after cancellation
    - Total: **6 Figures**

Background

Single tone generation:

Each pressed piano key/note number i, corresponds to a single tone with frequency f_i generated for a certain period of time T_i starting from time t_i over a defined time range t.

$$x_i(t) = \sin(2\pi f_i t) \cdot [u(t - t_i) - u(t - t_i - T_i)]$$

Example: Assume that we started pressing the middle C key ($f_1 = 261.63$ Hz), at time $t_1 = 0$ for a duration of $T_1 = 1.5$ seconds, the resultant signal is:

$$x_1(t) = \sin(2 \cdot 261.63 \cdot \pi t) \cdot [u(t) - u(t - 1.5)]$$

Parts 1 and 2: Song Generation

Signal/Song Generation

Generate your own signal using N pairs of notes chosen from the 3rd and 4th piano octaves for F_i and f_i, respectively. Assume you are playing with both hands at the same time.

$$x(t) = \sum_{i=1}^{N} [\sin(2\pi F_i t) + \sin(2\pi f_i t)] \cdot [u(t - t_i) - u(t - t_i - T_i)]$$

Piano Octaves Reference Table

Note	3rd Octave (Hz)	4th Octave (Hz)
C	130.81	261.63
D	146.83	293.66
E	164.81	329.63
F	174.61	349.23
G	196.00	392.00
A	220.00	440.00
B	246.94	493.88

Table 1: Piano note frequencies for 3rd and 4th octaves

Procedure

1. Import libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import sounddevice as sd
```

2. Set song duration and time vector:

```
t = np.linspace(0, 3, 3 * 44100 )
```

3. Set number of note pairs N, and define F_i, f_i, t_i, T_i values.
4. Define the signal $x(t)$ using the formula above.
5. Plot using `plt.plot(t, x)`
6. Play the song using:

```
sd.play(x, 3 * 44100 )
```

7. The factor of "3" accounts for the 3-second duration of the signal. Consequently, the total number of samples for the 3-second signal becomes $3 \times 44100 = 132300$ samples.

Parts 3 and 4: Noise Addition and Cancellation

Noise Cancellation Procedure

Noise Generation

Assume noise is generated as:

$$n(t) = \sin(2\pi f_{n1} t) + \sin(2\pi f_{n2} t)$$

Then the noisy signal becomes:

$$x_n(t) = x(t) + n(t)$$

Frequency Conversion

1. Add:

```
from scipy.fftpack import fft
```

2. Set sample count and frequency axis:

```
N = 3 * 44100
f = np.linspace(0, 512, int(N/2))
```

3. Compute FFT:

```
x_f = fft(x)
x_f = 2/N * np.abs(x_f[0:int(N/2)])
```

Noise Cancellation

1. Randomly generate:

```
fn1, fn2 = np.random.randint(0, 512, 2)
```

2. Add noise and plot noisy time-domain signal.
3. Compute frequency domain of noisy signal.
4. Identify peaks corresponding to noise.
5. Subtract sinusoidal components of f_{n1}, f_{n2} from $x_n(t)$.
6. Play filtered signal:

```
sd.play(x_filtered, 3 * 44100 )
```