***Faculty of Information Engineering and Technology Department of Communication Engineering***

***Signals and Systems Theory (COMM401)***

***Lab Project (PIANO)***

***Audio Processing***

***Milestone 1 Report***

**The group:**

* **Aly Raafat AbdelFattah Aly, id: 52-1008, Tutorial number: P-06**
* **Teodore Maged, id:52-3837, Tutorial number: P-06**

The imports we used:

1. Numpy (easily manipulate arrays)
2. Matplotlib (plot the graph of the song)
3. Sounddevice (play the song)
4. Math (to use ceil method)
5. scipy.fftpack (to convert time domain signals to frequency domain signals )

Variables and brief description about them:

* t (This the continuous time interval of 3 seconds through which the song will be played)
* Third octave notes: C3, D3, E3, F3, G3, A3, B3
* Fourth octave notes: C4, D4, E4, F4, G4, A4, B4
* Song (a 2D array below is my song where each subarray contains a note from the 3rd octave at index 0 and a not from the 4th octave at index 1)
* Start (array containing the start of each interval)
* end (array containing the end of each interval)
* j (a counter which we will use to loop on the start and end arrays)
* x (a 1D array that takes the shape of array t and is filled with zeros initially and this the final array where we will add the final notes and the values in this array will be plotted on the y-axis)
* start\_of\_interval (it multiplies (sin(2𝜋\* note[0] \*𝑡) + sin(2𝜋\*note[1] \*𝑡))\*𝑢(𝑡−start[j]) by the start of an interval at index j of the start array)
* end\_of\_interval (interval it multiplies (sin(2𝜋\* note[0] \*𝑡) + sin(2𝜋\*note[1] \*𝑡))\*𝑢(𝑡−end[j]) by the end of an interval at the same index j of the end array)
* N (It is number of samples and is equal to the song duration(3) x 1024)
* f (It is the frequency axis range from 0 to 512 and its size is int(N/2))
* x\_f\_data
* x\_f (this is the frequency domain signal of the time domain signal x)
* f1 and f2 (They are two random frequencies which are used to generate the noise signal)
* n (it is the noise signal and 𝑛(𝑡)=sin(2\*𝑓1\*𝜋\*𝑡) + sin(2𝑓2\*𝜋\*𝑡))
* xn (it is equal to add the noise signal n to time domain signal x)
* xn\_f\_data
* xn\_f (this is the conversion of the contaminated time domain signal xn to frequency domain signal xn\_f)
* max\_in\_x\_f (stores the maximum note in x\_f)
* max\_j (stores the index of a peak in xn\_f)
* j (index incremented to keep track of the location of elements in xn\_f)
* max\_k (stores the previous peak of index max\_j in xn\_f)
* f1\_temp (stores one peak of xn\_f at max\_j index and round it up to next integer and subtracts by 1)
* f2\_temp (stores second peak of xn\_f at max\_k index and round it up to next integer and subtracts by 1)
* 𝑥\_𝑓𝑖𝑙𝑡𝑒𝑟𝑒𝑑(𝑡)=𝑥𝑛(𝑡)−[sin(2\*𝑓1\_temp\*𝜋\*𝑡)+sin(2\*𝑓2\_temp\*𝜋\*𝑡)]
* x\_filtered\_data
* x\_filtered\_f (this is the conversion of the time domain signal x\_filtered to frequency domain signal x\_filtered\_f)

Brief Description of Milestone 1:

The for loop is used to loop on the 2D array called song where each sub array is called note and note[0] is a note from the 3rd octave and note[1] is a note from the 4th octave. The array called start of interval is used to multiply (sin(2𝜋\*note[0]\*𝑡)+sin(2𝜋\*note[1]\*𝑡))\*𝑢(𝑡−start[j]) by the start of an interval at index j of the start array and the array called end of interval is used to multiply (sin(2𝜋\*note[0]\*𝑡)+sin(2𝜋\*note[1]\*𝑡))\*𝑢(𝑡−end[j]) by the end of an interval at the same index j of the end array. Then we subtract the 2 arrays to get the time (interval) through which the note was played and add it to array x so the equation is (sin(2𝜋\*note[0]\*𝑡)+sin(2𝜋\*note[1]\*𝑡))\*(𝑢(𝑡−start[j])-𝑢(𝑡−end[j])).Then, we plot the graph of time (x-axis) against x (y-axis) using plt.plot(t,x) and play the song for 3 seconds and displays its sound using sd.play(x, 3∗1024).

Chart

Description automatically generated

We plotted six plots divided into two figures(3 rows and 2 columns).

Brief Description of Milestone 2:

We set the number of samples 𝑁 to the song duration (3)x1024. Then, set the frequency axis range from 0 to 512 and its size is int(N/2)). Then, we convert the time signal 𝑥(𝑡) to the frequency signal x(𝑓) and we generate the noise signal where the two random frequencies are selected (f1,f2). Then we add the generated noise 𝑛(𝑡)(= sin(2𝑓1\*𝜋\*𝑡)+sin(2\*𝑓2\*𝜋\*𝑡)) to the signal 𝑥(t). Afterwards, we convert the noise contaminated signal 𝑥𝑛(𝑡) to the signal x𝑛(f) in frequency domain. Then, we find the two random noise frequencies by looping over the xn\_f to find the frequency indices that corresponds to the peaks (f1\_temp,f2\_temp) of the signal xn(𝑓) which is higher than the maximum peak of the original signal x(𝑓) without noise and we round it to the next integer then subtract it by 1. Lastly, we filter the noise by subtracting two tones with the two found frequencies (f1\_temp,f2,temp) (𝑥𝑛(𝑡)−[sin(2\*𝑓1\_temp\*𝜋\*𝑡) + sin(2𝑓2\_temp\*𝜋\*𝑡)]). Then, we plot all the necessary plots and play the sound of the filtered noise.

First Figure:

* The first plot at (row 1 column 1) contains the time domain signal without noise (x)
* The second plot at (row 2 column 1) contains the time domain signal with noise (xn)
* The third plot at (row 3 column 1) contains the time domain signal after noise cancellation (x\_filtered).

Second Figure:

* The first plot(row 1 column 2) contains the frequency domain signal without noise (x\_f).
* The second plot(row 2 column 2) contains the frequency domain signal with noise (xn\_f).
* The third plot(row 3 column 2) contains the frequency domain signal after noise cancellation (x\_filtered\_f).

Diagram

Description automatically generated