Report

Milestone 1 Signals

DR: Ahmed ElMahdy

TA: Randa ELKhosht

Tutorial Number: 7

Team Member:	**ID:**
Mark Mahrous ElGendy	52-23533
Sama Samy Salah	52-20681

Steps:

1: We imported the libraries that we need in this milestone which are numpy , matplotlib.pyplot and sounddevice.

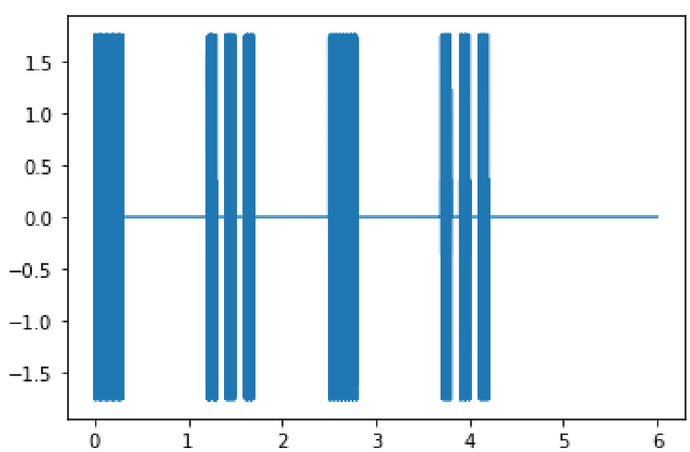
2: We set the total song playing time to be 6 seconds starting from zero for 2*12*1024 samples.

3: We assigned the frequencies of the tones for both third octave with the left hand in a numpy array called F_array ("F" upper case) and fourth octave with the right hand in a numpy array called f_array ("f" lower case) so we can loop on both arrays to get the frequencies and using them to generate the tones.

4: We assigned the starting time values of the tones in a numpy array called t_array ("t" lower case) and the certain periods of time of the tones in a numpy array called T_array("T" upper case) so we can loop on both arrays to get the starting time and period time and using them to specify the time of each tones.

5: Since we have 8 frequencies with each hand we created a variable (N=8) and counter (c=0) and accumulator variable (x=0) so we can loop on the 4 arrays we have using while loop to get the frequencies one by one from F_array and f_array and corresponding starting time from t_array and corresponding periodic time from T_array to create the first tone and accumulate it in the accumulator variable "x" and same for the second and third tones till we loop on the whole arrays.

6: We have now the generated song in variable "x" so we can finally plot it in the time domain to form the following figure using plt.plot(t,x) and play the song to hear it using sd.play(x,3*1024).



7: We set the number of samples to be equals the duration of the song * 1024 (6*1024) samples and the frequency axis range to be an array ➔ f = np. linspace(0 , 512 , int(N/2)).

8: We converted the song without noise (x) from the time domain to the frequency domain through ➔x_f = fft(x) & x_f = 2/N * np.abs(x_f [0:np.int(N/2)]).

9: We used the function np.random.randint(0 , 512 , 2) to generate two random integer frequencies to represent the noise add to the song ➔ $fn1$, fn2 = np. random. randint(0, 512, 2) & the noise generated using the function ➔ n = np.sin(2*np.pi*fn1*t)+np.sin(2*np.pi*fn2*t).

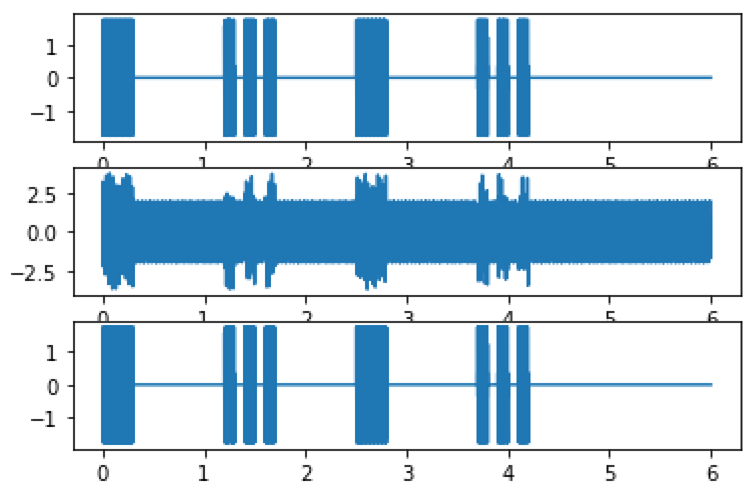
10: We added the noise to the song ➔ (xn = n + x), then we converted the (xn) to the frequency domain ➔ xn_f = fft(xn) & xn_f = 2/N * np.abs(xn_f [0:np.int(N/2)]).

11: We used ➔(z = np.where(xn_f>math.ceil(np.max(x)))) to create z which contains array of indecies of the two random frequencies and their types then we get the indecies through ➔(index1 = z[0][0] & index2 = z[0][1]) , then we got the frequencies through ➔(found1 = int(f[index1]) & found2 = int(f[index2])).

12: We filtered the song from noise through ➔(xFiltered = xn - (np.sin(2*np.pi*found1*t)+np.sin(2*np.pi*found2*t))), then we played it to get the original song without noise through ➔(sd.play(xFiltered, 3*1024)).

13: We converted the filtered song to frequency domain through ➔(xFiltered_f = fft(xFiltered) && xFiltered_f = 2/N * np.abs(xFiltered_f [0:np.int(N/2)])) to graph it.

14: We figured the the graphs in the time domain alone using plt.figure() & plt.subplot(3, 1, _) to get the following three figures.



15: Then we figured the graphs in the frequency domain alone using plt.figure() & plt.subplot(3, 1, _) to get the following three figures.