Image Processing - Exercise 2

Shir Rashkovits, shirashko, 209144013

Introduction

The primary aim of this exercise was to get experienced with transitioning between a signal's standard representation and its frequency-based, utilizing the FFT and STFT methods. Through this approach, I learned to decompose a signal into its individual frequency components in a script, a technique particularly useful for noise reduction tasks, and for many other tasks as well. The primary tool throughout this process was the STFT method. Using this simplifies the task of breaking down signals into their frequency components, in relevant time segments, thereby making it more straightforward to identify and eliminate noise in the necessary time windows.

During my analysis, I encountered two distinct types of noise:
- In the first audio, there was a consistent and prominent noise present throughout the entire audio which was concentrated in one frequency. This frequency had much bigger magnitude relative to all other frequencies in all windows of the audio spectrogram. This enabled me to avoid different approaches in different time points in the audio, and to use FFT to apply the denoised mask filter.
- The second audio presented a more complex challenge with a range of low frequencies which created the noise, presented only in part of the audio, and showed amplitude variation in this time range as well. These differences influenced my noise removal strategy. The consistent noise in the first audio was relatively easy to identify and remove. However, the second audio required a more nuanced approach due to the changing nature of its noise. Here, I had to examine each segment of the audio using STFT, which was slightly more complex. I needed to be cautious to ensure that I was removing only the noise parts without affecting the actual audio content. For that I needed to identify the time range and frequency range that created the noise.

Algorithm

For the first audio: (Q1)
Algorithm:
1. Load the audio from the path
2. Decompose the audio into its frequency constituents using the FFT algorithm.
3. Calculate the magnitude of each frequency Fourier coefficient of the audio.

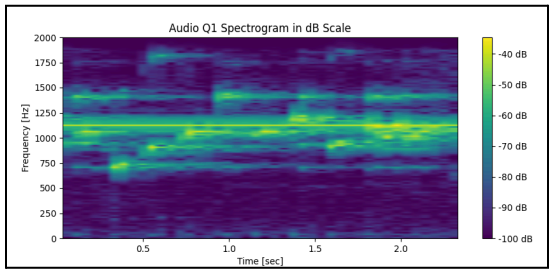
4. Find the frequency with the biggest magnitude and its conjugate (argmax) (The positive and negative frequency: u* = argmax u (F(u)) and N-u*)
5. Apply Mask in the form of zero the Fourier coefficient of the result from step 4 and not change the rest (binary mask).
6. Calculate the IFFT of the filtered result from step 5
7. Return the denoised audio received in step 6

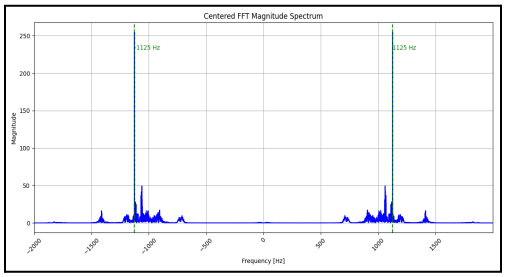
Implementation:
For loading the audio I have used the Scipy library. For calculating FFT, magnitude (np.abs for complex numbers calculate it),and argmax of the frequencies amplitudes was implemented using numpy. For applying the filter I have assigned in the relevant Fourier coefficients (1125, N-1125) numpy array (which was received from applying FFT on the audio) zero, which practically applied the denoised filter. At the end I applied IFFT (took only the real part of the result since the real signal didn't include a complex part) on the denoised result and returned the result. For centering FFT results and plotting in exploration I have used scipy and matplotlib, and for saving the audio to listen if the result is clean I have used soundfile library.

Hyper parameters, thresholds, or other choices used in your algorithm:
When exploring the signal I have used STFT with default parameters of the scipy.signal.spectrogram which are with window size of 256 and an overlapping window of ½ of the window size. Thanks to specifying the sample rate with sr argument based on scipy library implementation I got in the audio spectrogram all the frequency range expected (sr=4000 -> frequency range up to 2000).

Challenges:
I had a hard time understanding how we get frequencies in the range of 2000 in the spectrogram if in every window we have less than 2000 samples, due to the Nyquist law. Eventually I understood that the sampling rate provided as an argument made the scipy algorithm adjust accordingly. Also, I was a bit afraid that removing the suspicious frequency altogether would cause problems, if the real signal also contained the frequency in some way. but because it was only one frequency and I didn't have the real signal to compare to, and the human ear is not that sensitive, it sounded good enough.

Exploration which leaded to the choice of this algorithm:





When exploring the signal, I first listened to the audio and noticed that a constant high-frequency noise is present in the signal from the beginning to the end. Upon plotting a spectrogram of the audio, I observed a frequency with a much larger amplitude than all other frequencies, consistent across all windows. This was unnatural compared to the rest of the frequencies in the spectrogram. Given the suspicious frequency's larger amplitude in all windows, I decided to work with the Fast Fourier Transform (FFT), which does not require high time resolution. This decision was based on the expectation that the consistent amplitude of the noise across time would be evident even without segmenting the audio into short time frames. Indeed, the FFT analysis revealed one frequency (1125 Hz, along with its mirror at -1125 Hz when centering) significantly higher than the rest. I attempted to remove this frequency to determine if it was the source of the noise, and after doing so, the audio was significantly cleaner.

For the second audio: (Q2)
Algorithm:
1. Load the audio from the given path
2. Apply STFT to the audio signal (which decomposes the audio into its frequency components at various timestamps)
3. Identify the time windows corresponding to 1.5 to 4 seconds of the audio duration. Within these windows, selectively zero out the Fourier coefficients in the 550-650 Hz frequency range. **work on positive frequency and mirror on the corresponding negative
4. Apply ISTFT to reconstruct the audio signal from its modified spectral representation
5. Return the audio result from step 4.

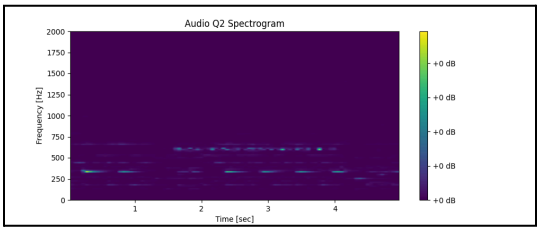
Implementation:
Loading the audio and calculating the STFT using the scipy library. Using numpy I have calculated the time indices and the noise band, and afterward I have assigned zero in the desired windows to the desired frequency range. At the end, using scipy I applied ISTFT and returned the result.

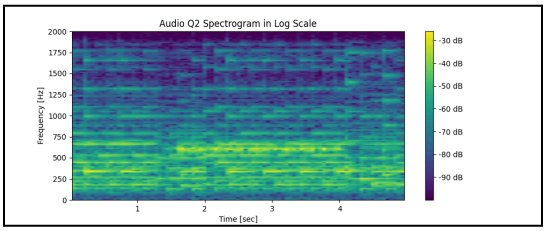
Hyper parameters, thresholds, or other choices used in your algorithm:
I have used for the STFT window size 250, which is divider of the number of q2 audio samples (2000), and is closest to the default window size of scipy (I don’t know in details there exact implementation so I have preferred to stay close to their default at first, and since it worked well, I sticked with it). It is important to choose a window size dividing in the number of samples because if not, the scipy library will create another window for the extra samples and then when applying the ISTFT we will have more samples then the original audio, which is not something we want (we want the same audio length). The rest of the DTFT parameters were taken to be the library default one

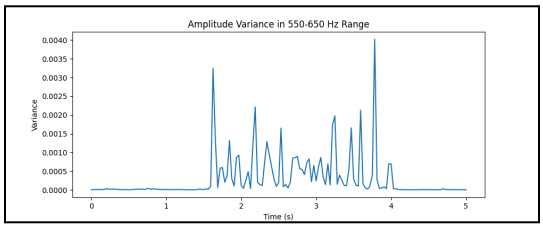
(for example, overlap is half of the window size).
The choice of the frequency and time ranges chosen to apply the filter on were due to the spectrogram display and they might not be 100% accurate but they gave good enough resolution to the human ear (in my opinion). More elaborate calculations could be conducted to get a more precise range but for the purpose of this exercise I thought it was good enough. I also attached graphs which show the mean and variance of the frequencies in 550-650 through the audio to try to justify the time range choice (1.5-4).

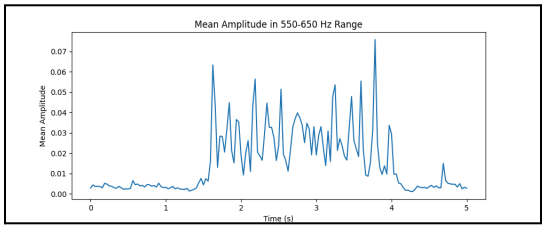
Challenges:
I was a bit worried that zeroing out frequencies from some windows to others would be very noticeable, since it isn't a smooth transition. However, because I used overlapping windows and since this range of frequencies wasn't very prominent at other times in the audio, the denoised audio sounded pretty good after applying the filter.

Exploration which leaded to the choice of this algorithm:
When hearing the audio, it sounded great until some point in time and from some point in time. Therefore I suspected the noise is present only in some range of time. At this point I have displayed a spectrogram of the audio which it's displayed below. It was easy to see (especially in the linear scale spectrogram) that there is a range of frequencies around 600 Hz which is present only in a range which seems consistent with the time range the noise of the audio sounds like.









Conclusion

In this exercise I have learned how to utilize the frequency domain in order to analyze audio. My conclusion from the denoised assignment after experimenting with more than one audio, is that we might need to apply different approaches on different audios because noise can vary in its appearance (frequency range, amplitude variations etc) and time. I have understood the power of spectrogram and frequency domain in helping to explore and understand components of an audio signal.