SIT789 – Robotics, Computer Vision and Speech Processing

Credit Task 7.2: Speech enhancement

Objectives

The objectives of this lab include:

- Applying low-pass, high-pass and band-pass filters to remove noise from speech signals
- Enhancing speech quality using spectral subtraction
- Practising scipy.signal package

Tasks

1. Filtering

In this section, we will practise low-pass, high-pass, and band-pass filters. You first need to download the dataset in the Resources supplied in OnTrack. This dataset includes clean speech clips (in CleanSignals), noisy speech clips (in NoisySignals) and noise clips that contain only noise (in Noise). The dataset also include different types of noise, e.g., background noise from train stations, babble noise. The clean and noisy audio clips are from the NOIZEUS dataset. The noise clips are from the MS-SNSD.

In this lab, we will use scipy.signal. We update scipy with the newest version by doing:

```
pip install --upgrade scipy
```

Depending on the permission you set on your computer, you may need to provide user option in your pip command, e.g.,

```
pip install --user --upgrade scipy
```

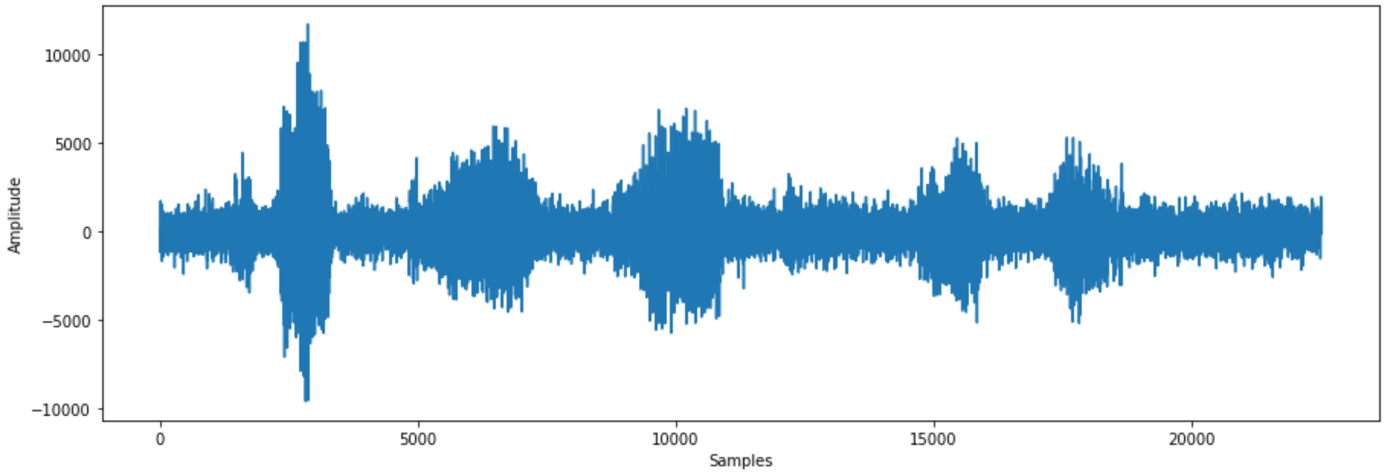
To get the update affected, you need to restart your computer.
Take the audio file sp01_station_sn5.wav in NoisySignal/Station as an example. You then load the audio file as follows.

```
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import IPython.display as ipd

from pydub import AudioSegment
from pydub.utils import mediainfo
```

```python
noisy_speech = AudioSegment.from_wav('NoisySignal/Station/sp01_station_sn5.wav')
noisy_s = noisy_speech.get_array_of_samples()   # samples x(t)
noisy_f = noisy_speech.frame_rate   # sampling rate f
```

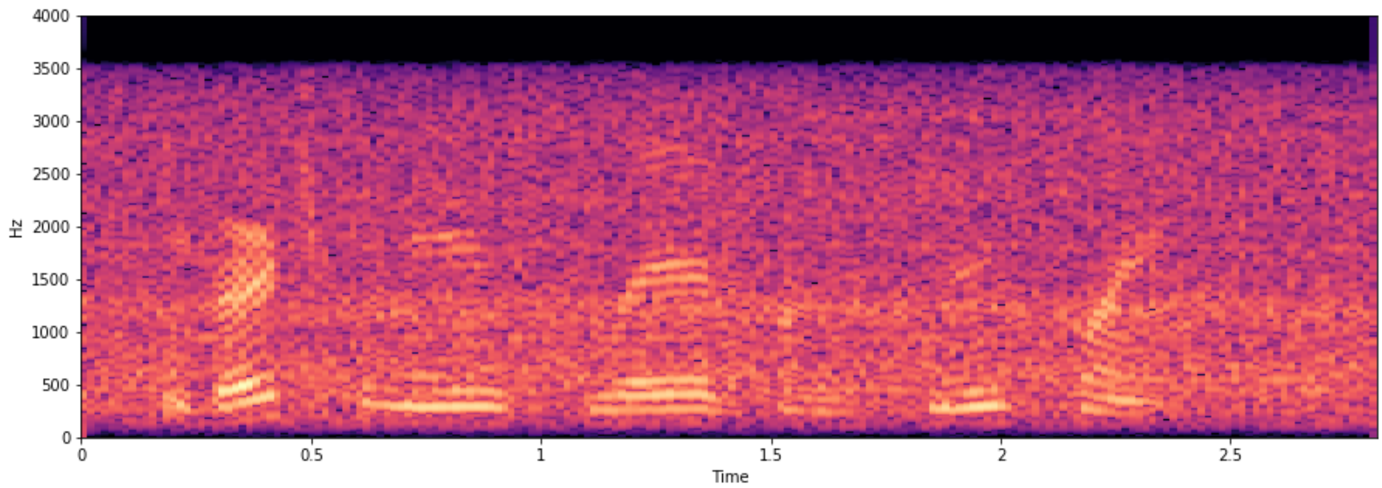
We can visualise the signal noisy_s in time domain using the following code

```python
plt.figure(figsize = (15, 5))
plt.plot(noisy_s)
plt.xlabel('Samples')
plt.ylabel('Amplitude')
```



Now, we examine the signal in noisy_s in frequency domain by plotting its spectrogram using the Fourier transform.

```python
#range of frequencies of interest for speech signal.
#It can be any positive value, but should be a power of 2
freq_range = 2048

#window size: the number of samples per frame. Each frame is of 30ms = 0.03 sec
win_length = int(noisy_f * 0.03)

#number of samples between two consecutive frames
hop_length = int(win_length / 2)

#windowing technique
window = 'hann'

noisy_S = librosa.stft(np.float32(noisy_s),
                       n_fft = freq_range,
                       window = window,
                       hop_length = hop_length,
                       win_length = win_length)

plt.figure(figsize = (15, 5))

#convert the amplitude to decibels, just for illustration purpose
noisy_Sdb = librosa.amplitude_to_db(abs(noisy_S))
librosa.display.specshow(noisy_Sdb, #spectrogram
                        sr = noisy_f, #sampling rate
                        x_axis = 'time', #label for horizontal axis
                        y_axis = 'linear', #presentation scale
                        hop_length = hop_length) #hop_length
```



Suppose that we want to remove frequencies higher than 1000Hz from noisy_s. We define a cut-off frequency cutoff_freq = 1000 and construct a Butterworth (low-pass) filter with this cut-off frequency as follows.

```
from scipy import signal

#order
order = 10

#sampling frequency
sampling_freq = noisy_f

#cut-off frequency. This can be an array if band-pass filter is used
#this must be within 0 and cutoff_freq/2
cutoff_freq = 1000

#filter type, e.g., 'lowpass', 'highpass', 'bandpass', 'bandstop'
filter_type = 'lowpass'

#filter
h = signal.butter(N = order,
                  fs = sampling_freq,
                  Wn = cutoff_freq,
                  btype = filter_type,
                  analog = False,
                  output = 'sos')
```

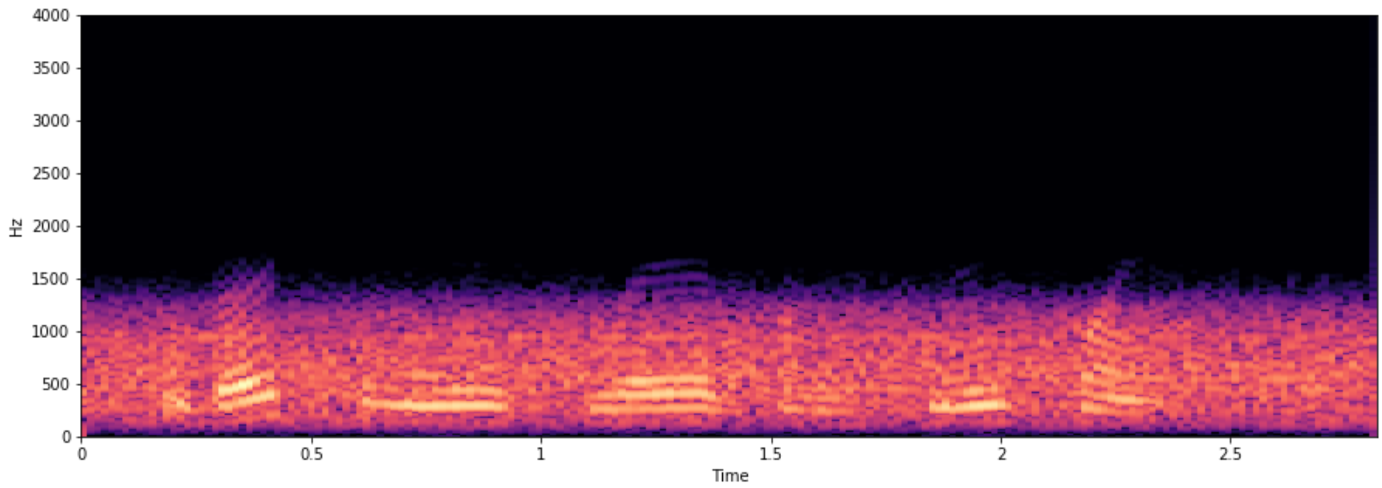
Now, we apply the filter h to noisy signal noisy_s as follows.

```
filtered_s = signal.sosfilt(h, noisy_s)
```

filtered_s is the filtering result of noisy_s. You will need to save filtered_s to file and auditorily evaluate its quality.

```
import array
import pydub
from pydub import AudioSegment

filtered_s_audio = pydub.AudioSegment(
         data = array.array(noisy_speech.array_type,
                            filtered_s.astype(np.int16)), #raw data
         sample_width = 2, #2 bytes = 16 bit samples
         frame_rate = noisy_f, #frame rate
         channels = 1) #channels = 1 for mono and 2 for stereo

filtered_s_audio.export('sp01_station_sn5_lowpass.wav', format = 'wav')
```

To visualise the effect of the filter h, we show the spectrogram of filtered_s (using the same manner as what we have done to show the spectrogram of noisy_s).



You should test this filter with the supplied speech data. You should also vary parameters used in the filter, e.g., order, cutoff_freq, and manually inspect the corresponding filtering results.

Your tasks here include:

1. Implement a high-pass filter with a cut-off frequency = 200Hz using Butterworth filter. You can use the same parameter settings used in the previous low-pass filter. **Hint:** In signal.butter, set Wn = 200, btype = 'highpass'.
2. Implement a band-pass filter with a pass band = [200Hz, 1000Hz] using Butterworth filter. You can use the same parameter settings used in the previous low-pass filter. **Hint:** In signal.butter, set Wn = [200, 1000], btype = 'bandpass'.

2. Noise attenuation

In this section, we will apply the spectral subtraction for noise attenuation. We will use the noisy speech data in the NoisySignals folder and the noise signals (of different noise types) in the Noise folder. The noisy speech data is organised into different sub-folders corresponding to their noise types, e.g., background noise from train stations, babble noise. Clean signals are also provided for evaluation of the noise attenuation algorithms.

Take the audio file sp01_station_sn5.wav in NoisySignal/Station as an example. Let y be the speech signal contained in sp01_station_sn5.wav, Y be the Fourier transforms of y, and mag_Y be the magnitude of Y (i.e., mag_Y = abs(Y)).

```
noisy_speech = AudioSegment.from_wav('NoisySignal/Station/sp01_station_sn5.wav')
y = noisy_speech.get_array_of_samples()  # samples x(t)
y_f = noisy_speech.frame_rate  # sampling rate f
win_length = int(y_f * 0.03) # Each frame is of 30ms = 0.03 sec
hop_length = int(win_length / 2) #number of samples between two consecutive frames

Y = librosa.stft(np.float32(y),
                 n_fft = 2048,
                 window = 'hann',
                 hop_length = hop_length,
                 win_length = win_length)
mag_Y = abs(Y)
```

Similarly, let d be the noise signal contained in Noise/Station/Station_1.wav. Let D be the Fourier transforms of d, and square_mag_D be the square of the magnitude of D. You are to implement d, D, and square_mag_D. **Note:** in this example, we are working on "station" noise. Therefore, Noise/Station/Station_1.wav is used. For other noise types, proper noise data should be used.

Note that square_mag_D is an array whose rows represent frequencies and columns represent temporal frames. Next, we calculate the means for rows in square_mag_D, i.e., averaged-time frequencies, (see Eq (17), slide 63 in week 7 lecture slides).

```
means_square_mag_D = np.mean(square_mag_D, axis = 1)
```

Your tasks include:

1. Implement d, D, and square_mag_D
2. Implement the Fourier transform H (see Eq (19), slide 64 in week 7 slides)
3. Estimate the Fourier transform S_hat (see Eq (20), slide 64 in week 7 slides)
4. Get the inverse of S_hat to retrieve s_hat, then save s_hat to file named sp01_station_sn5_spectralsubtraction.wav
5. Auditorily inspect the quality of sp01_station_sn5_spectralsubtraction.wav
6. Visualise the spectrogram of sp01_station_sn5_spectralsubtraction.wav, i.e., S_hat, and the spectrogram of the clean signal in CleanSignal/sp01.wav
7. Test the spectral subtraction algorithm on the supplied speech signals in NoisySignal, provide observations and draw conclusions. Note that, you should not expect outstanding performance in noise reduction by the spectral subtraction algorithm.

Hint:

1. H, and S_hat are 2D arrays and have the same shape with Y (i.e., same number of rows and same number of columns).
2. Use librosa.istft to calculate s_hat (the inverse DFT of S_hat) as follows.

   ```
   win_length = int(y_f * 0.03)
   hop_length = int(win_length / 2)
   s_hat = librosa.istft(S_hat,
                         win_length = win_length,
                         hop_length = hop_length,
                         length = len(y))
   ```

3. s_hat, computed using librosa.istft, may contain values out of the range of 16 bits. In order to store s_hat in a 16-bit wave file, you need to truncate the value of s_hat as follows.

   ```
   s_hat_truncated = np.int16(s_hat)
   for i, num in enumerate(s_hat_truncated):
       if num > 32767:
           s_hat_truncated[i] = 32767
       elif num < -32768:
           s_hat_truncated[i] = -32768
   ```

   Finally, you need to set parameter data in pydub.AudioSegment as:

   ```
   data = array.array('h', s_hat_truncated)
   ```

Submission instructions

1. Perform tasks required in Section 1 and 2.
2. Complete the supplied answer sheet with required results
3. Submit the answer sheet (.pdf) and code (.py) to OnTrack.