SIT789 – Robotics, Computer Vision and Speech Processing

Pass Task 7.1: Introduction to Speech Processing

Objectives

This lab will get you through basic operations in speech processing and signal processing in Python. Specifically, the objectives of this lab include:

- Audio signal IO
- Calculating spectrum and spectrogram of speech sounds
- Speech processing using librosa and pydub

Tasks

1. Audio signal IO

From week 7 onwards, we will be using librosa and pydub for speech signal IO and speech signal processing. These libraries can be installed from cmd/terminal using the following commands

```
pip install librosa
```

```
pip install pydub
```

We will be mainly using pydub for speech signal IO and librosa for speech signal processing. Documentation for librosa can be found at https://librosa.org/doc/latest/index.html.

In this section, we will learn how to read/write audio signals from files. You need to download the audio file arctic_a0005.wav supplied in OnTrack. This file is from the CMU ARCTIC database.
You then load the audio signal from arctic_a0005.wav as follows.

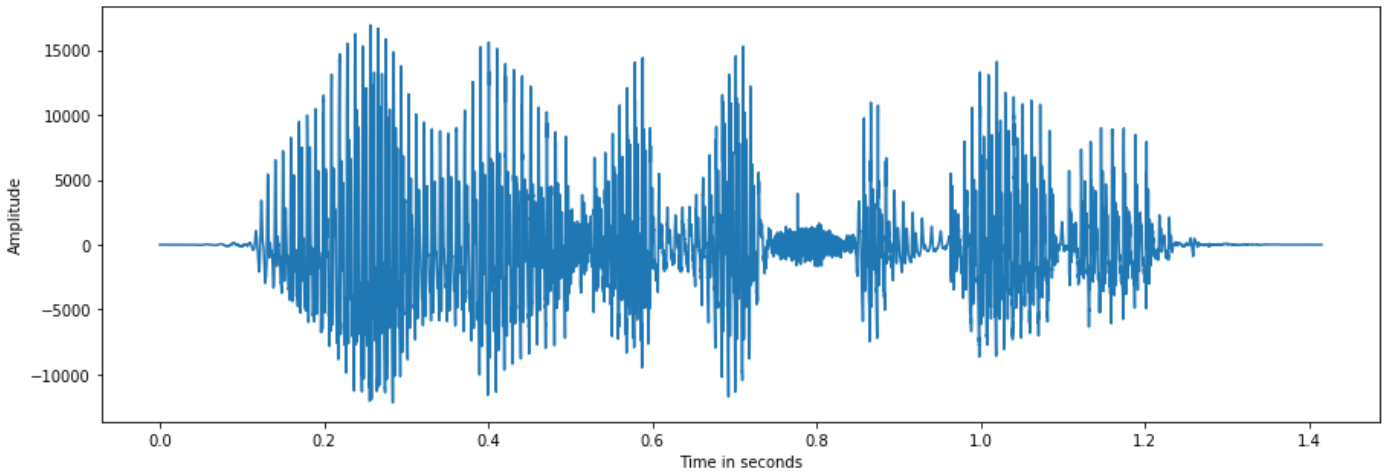
```
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import IPython.display as ipd

from pydub import AudioSegment
from pydub.utils import mediainfo

speech = AudioSegment.from_wav('arctic_a0005.wav') # Read audio data from file
x = speech.get_array_of_samples()  # samples x(t)
x_sr = speech.frame_rate  # sampling rate f - see slide 24 in week 7 lecture slides

print('Sampling rate: ', x_sr)
print('Number of samples: ', len(x))
```

To understand the number of samples in x, you can get the duration of arctic_a0005.wav (in seconds) and multiply this duration by the sampling rate as follows.

```
duration = librosa.get_duration(path = 'arctic_a0005.wav')
n_samples = duration * x_sr
print('duration: ', duration)
print('n_samples: ', n_samples)
```

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

```
x_range = np.linspace(0, duration, len(x))

plt.figure(figsize = (15, 5))
plt.plot(x_range, x)
plt.xlabel('Time in seconds')
plt.ylabel('Amplitude')
```



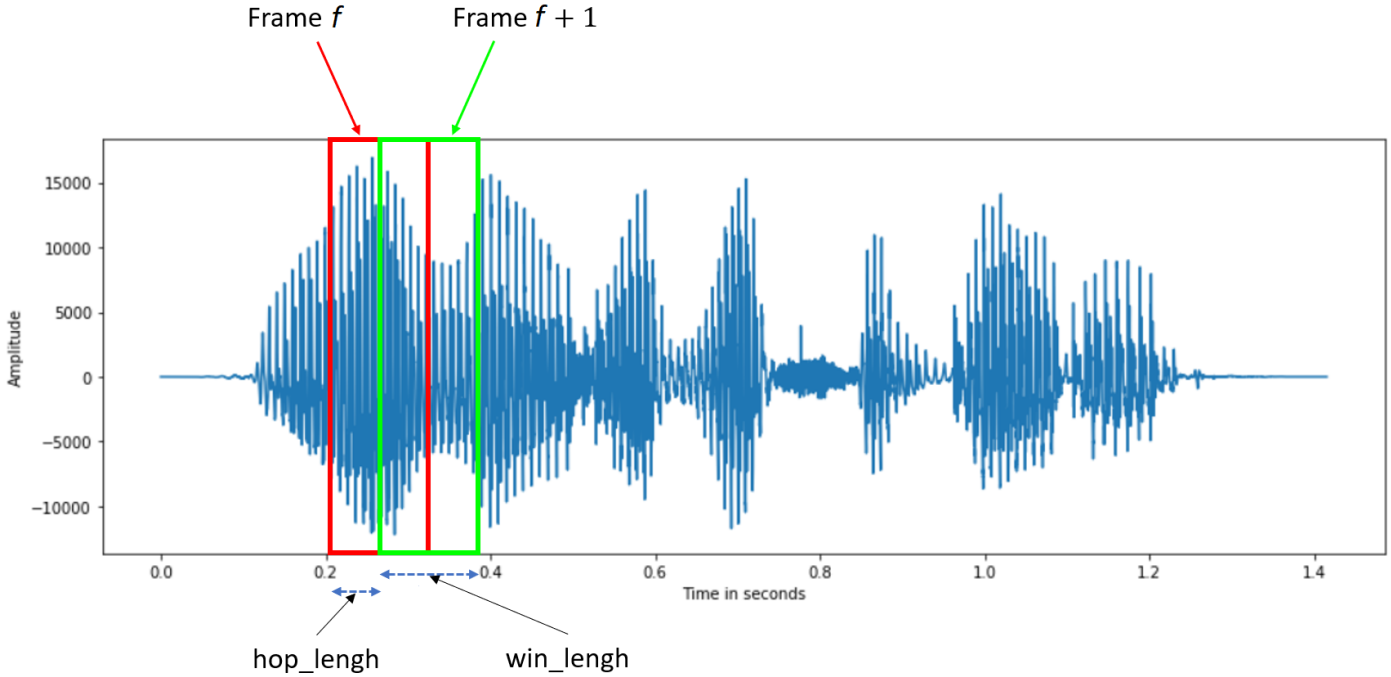
Now, suppose that we want to divide the signal x into two equal parts, named x1 and x2, i.e., each part has 11320 samples. We then store those parts into .wav files. To save audio data to .wav files, you can pydub as follows.

```
mid_point = int(len(x) / 2)
x1 = x[0:mid_point]
x2 = x[mid_point:len(x)]

x1_audio = AudioSegment(
                data = x1, #raw data
                sample_width = 2, #2 bytes = 16 bit samples
                frame_rate = x_sr, #frame rate
                channels = 1) #channels = 1 for mono and 2 for stereo

x2_audio = AudioSegment(
                data = x2, #raw data
                sample_width = 2, #2 bytes = 16 bit samples
                frame_rate = x_sr, #frame rate
                channels = 1) #channels = 1 for mono and 2 for stereo

x1_audio.export('arctic_a0005_1.wav', format = 'wav')
x2_audio.export('arctic_a0005_2.wav', format = 'wav')
```

You now can check the audio clips saved in arctic_a0005_1.wav and arctic_a0005_2.wav.
You task is to visualise x1 and x2 in time domain.

2. Fourier Transform

In this section, we will apply the Fourier transform on audio signals to compute the spectrograms of the signals. In practice, we do not calculate the Fourier transform of a signal on its entire time domain but only on short temporal segments. These temporal segments are called frames and detected using windowing techniques. Two consecutive frames can have some overlapping as shown in the following figure.



To calculate the Fourier transfrom of x, you need to determine the range of frequencies of interest and call the method librosa.stft as follows.

```
#range of frequencies of interest for speech signal.
#It can be any positive value, but should be a power of 2
freq_range = 1024

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

#number of samples between two consecutive frames, by default it is set to win_length/4
hop_length = int(win_length / 2)

#windowing technique
window = 'hann'

X = librosa.stft(np.float32(x),
                 n_fft = freq_range,
                 window = window,
                 hop_length = hop_length,
                 win_length = win_length)
```

librosa.stft returns an array, e.g., X. This array includes freq_range/2+1 rows corresponding to the frequencies in [0, freq_range/2]. **Note**: Since both the input and output of the Fourier transform can be sequences of complex values, the Fourier transform produces the spectra for both the real and imaginary components with the same frequency range. In our case, since x is a sequence of real values, the spectra for the real and imaginary components will be symmetric and thus only one spectrum is generated. This explains the rows in X.

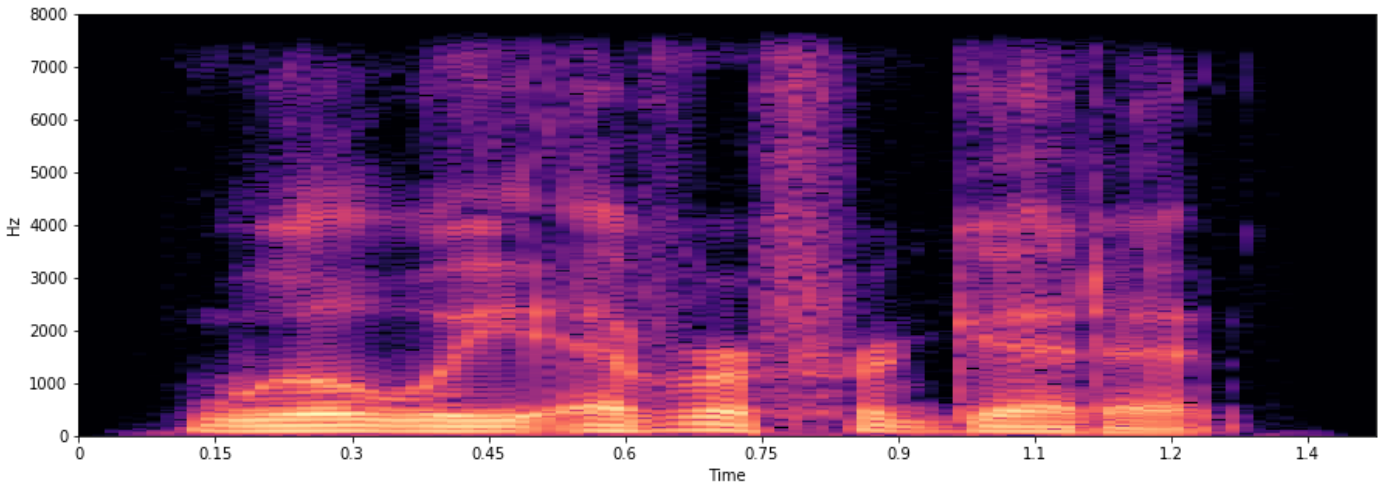
If x include n_samples samples, the array X will have $\left\lceil \frac{n\_samples}{hop\_length} \right\rceil$ columns where $\lceil r \rceil$ is the smallest integer value that is equal to or greater than r.

You can check the dimension of X by calling:

```
print(X.shape)
```

X actually contains the spectrogram of x. To visualise X, we can perfrom:

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

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



Your task is to create the spectrograms for x1 and x2.

To better interpret the Fourier transform, we perform a simulation as follows. Suppose that we have two signals s1 and s2 defined as,

$$s_1(t)=A_1\sin(2\pi f_1 t)$$
$$s_2(t)=A_2\sin(2\pi f_2 t)$$

where $A_1=1$, $f_1=50$, $A_2=0.5$, $f_2=80$.

We now define,

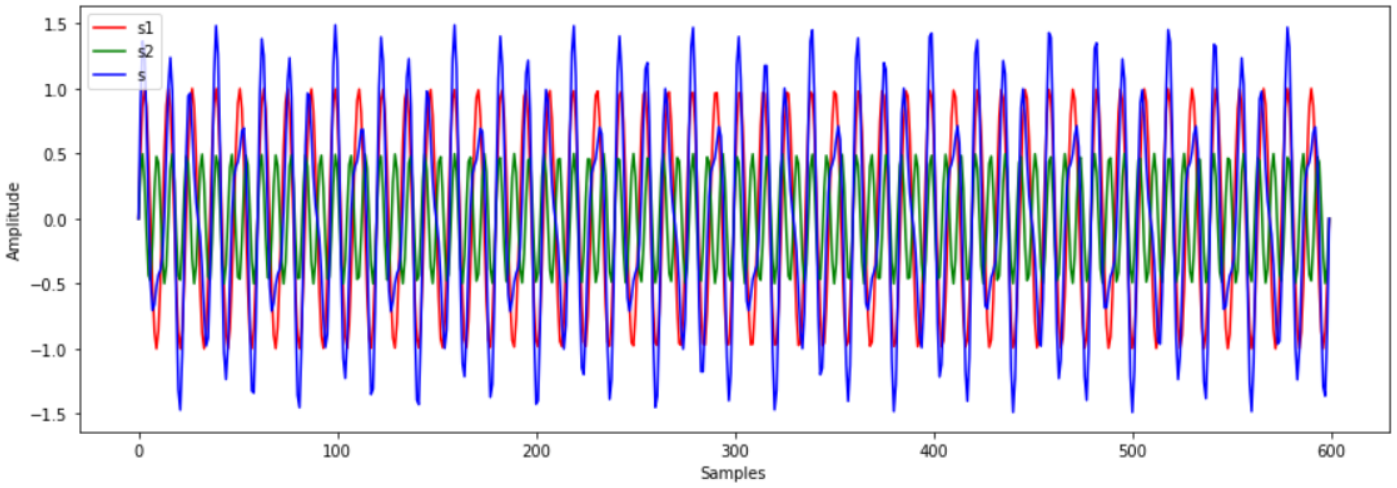
$$s(t)=s_1(t)+s_2(t)$$

We can initialise s_1, s_2, and s with 600 samples as follows.

```
#number of samples
N = 600

#sample spacing
T = 1.0 / N

t = np.linspace(0.0, N*T, N)
s1 = np.sin(50.0 * 2.0 * np.pi * t)
s2 = 0.5 * np.sin(80.0 * 2.0 * np.pi * t)
s = s1 + s2
```

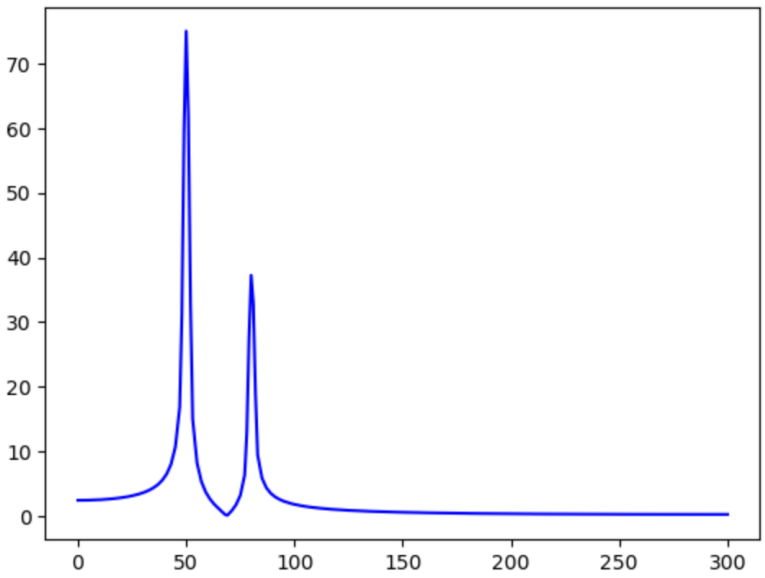
We can visualise s_1, s_2, and s as,

```
plt.figure(figsize = (15, 5))
plt.plot(s1, label = 's1', color = 'r')
plt.plot(s2, label = 's2', color = 'g')
plt.plot(s, label = 's', color = 'b')
plt.xlabel('Samples')
plt.ylabel('Amplitude')
plt.legend(loc = "upper left")
```



Since $s(t)$ is formed by two sinusoidal signals $s_1(t)$ and $s_2(t)$ of different frequencies, the Fourier transform of s is expected to have two peaks corresponding to the frequencies of s_1 and s_2, respectively. We visualise the Fourier transform of s (denoted as S) below.

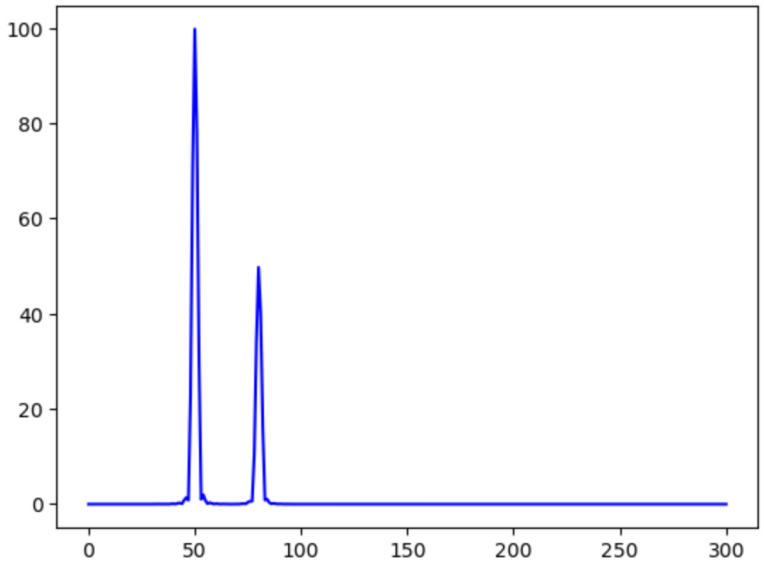
```
S = librosa.stft(s, n_fft = N, window = 'hann', hop_length = N, win_length = N)
S_0 = S[:, 0]
mag_S_0 = np.abs(S_0)
plt.plot(mag_S_0, color = 'b')
```



Theoretically, the Fourier transform of a signal is defined in infinite domain. However, in practice, one can apply the Fourier transform on a finite and truncated signal. To calculate the Fourier transform of truncated signals, librosa.stft uses windowing techniques. However, windowing techniques are sensitive to the borders of the truncated signals.

In the above code, we set win_length=N and hop_length=N, i.e., the window fully covers the truncated signal. To visualise the sensitivity mentioned above, we run an experiment in which a smaller window is used, i.e., win_length is set to a smaller value. In addition, instead of using the Fourier transform of a border frame, e.g., S[:,0], we can take the Fourier transform of an intermediate frame, e.g., S[:,1].

```
#we define a window length m with fewer samples
m = 400

S = librosa.stft(s, n_fft = N, window = 'hann', hop_length = int(m/2), win_length = m)

#we take S_1, which is an intermediate frame.
S_1 = S[:, 1]

mag_S_1 = np.abs(S_1)
plt.plot(mag_S_1, color = 'b')
```



As shown, the Fourier transform S_1 better shows the frequencies f_1=50 and f_2=80.

Different windowing techniques make different border effects. Your task is to investigate these effects by experimenting the parameter window in librosa.stft to calculate S_1 with options including: 'boxcar' (i.e., rectangular window), 'hann' and 'hamming', and discuss your observations.

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) to OnTrack.