Signals and Systems Course Project
Real-Time Voice Converter

Alireza Moradi - Sara Kodeiri

July 29, 2020

1 Overview

The goal was to implement a system that changes the pitch of an input sound from device's mic with minimum delay(milliseconds) and play it through speakers. This project is implemented in python and also in swift for iOS devices.This is the document for the python implementation. Below are the packages used in this project.

- `pyaudio` for streaming input sound from mic.
- `librosa` for sound manipulation and analysis.
- `scipy` for resampling.
- `time` for general purposes.
- `numpy` for converting mic's input byte string to numpy array.

[ ]:
```
import pyaudio
import librosa
import scipy
import time
import numpy as np
```

`pitch_shift` is the core of the project. Basically we stretch or compress the input signal and then resample it to be the same length as the original signal. we determine the stretch or compress amount of the signal by $r = 2^{\frac{-n}{12}}$. Computations are done in time-frequency domain so we take the Short-Time Fourier Transform(STFT) of the signal and use `phase vocoder` algorithm to stretch or compress the signal and finally take the iSTFT of the signal to change it back to time domain. At last the result is resampled back to the original rate to have the same length as the original signal and to make sure they are the same length we fix the length of the final signal to be the same as the original one.

[ ]:
```
def pitch_shift(signal, n):
    r = 2.0 ** (-float(n) / 12)
    stft = librosa.core.stft(signal)
    stft_stretch = librosa.core.phase_vocoder(stft, r)
    len_stretch = int(round(len(signal) / r))
    signal_stretch = librosa.core.istft(stft_stretch, dtype=signal.dtype,␣
 ↪length=len_stretch)
    n_samples = len(signal)
    signal_resample = scipy.signal.resample(signal_stretch, n_samples, axis=-1)
    signal_resample = librosa.util.fix_length(signal_resample, len(signal))
    return signal_resample
```

Here is a callback function for the pyaudio stream which does the process in the input sound (pitch shifting in this case) asynch and returns the processed signal to be played through the output. All the processes are real-time.

```
[ ]: def callback(in_data, frame_count, time_info, flag):
         ch = np.fromstring(in_data, dtype=np.float32)
         new_signal = pitch_shift(ch, shift_amount)
         return new_signal, pyaudio.paContinue
```

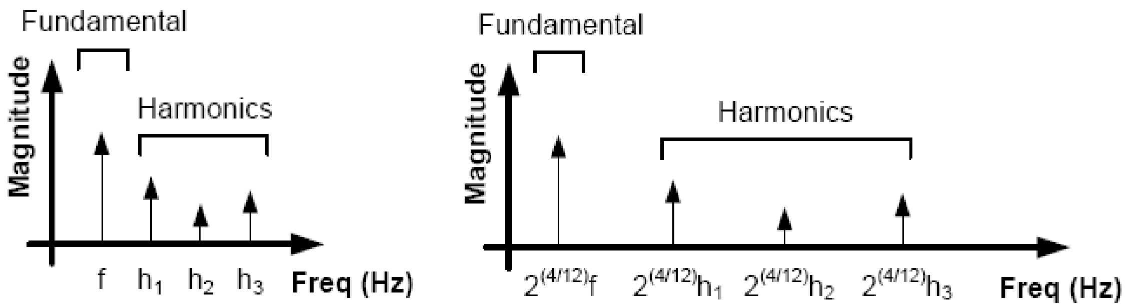
The main function contains configuration for pyaudio stream with 44.1KHz sampling rate of a mono audio and 12288 buffer size for processing. `shift_amount` is the amount that we want to shift the pitch of the sound which is usually between -10 and 10.

```
[ ]: if __name__ == '__main__':
         print("Preparing...")

         RATE = 44100
         CHUNK = int(4096 * 3)
         CHANNELS = 1
         shift_amount = 5

         p = pyaudio.PyAudio()
         stream = p.open(format=pyaudio.paFloat32,
                         channels=CHANNELS,
                         rate=RATE,
                         output=True,
                         input=True,
                         frames_per_buffer=CHUNK,
                         stream_callback=callback)

         stream.start_stream()
         t = time.time()
         while time.time() - t < 3 * stream.get_input_latency():
             pass
         print("Voice Converter Activated...")
         try:
             while stream.is_active():
                 pass
         except KeyboardInterrupt:
             print("Keyboard interrupt detected.")
             print("exiting...")

         stream.stop_stream()
         stream.close()
```

```
    p.terminate()
    pass
```

2 Pitch

The pitch of a sound corresponds to the set of frequencies the sound is made of. Sounds like musical instruments and human voice are harmonic and all of them contain a fundemental frequency which corresponds to where the first rise in sound's energy occurs in the frequency domain and this happens again in multiples of fundemental frequency as the picture below shows.



So if the harmonics are not multiples of the fundemental the sound will be distorted and will sound bad to the ear. So if we want to change the pitch of a sound these harmonics should be preserved and simply shifting the frequency of the sound is not the answer because it will ruin the harmonics. So basically in order to change the pitch we have to do 2 things:

- change the signal duration without changing the pitch.
- Once this is done, we will perform resampling to return to the initial duration with a shift in the pitch.

P.S. The scaling factor is defined as the factor used to stretch or compress the spectrum in order to adjust the frequencies such that the pitch is shifted.($r = 2^{\frac{-n}{12}}$ in our code)

We will split out sound to many small chunks in a way that they have overlapping. These chunks will be spaced differently to stretch or compress the signal but this causes a problem. Because the chunks are spaced differently there will be discontinuities in the resulting signal and thats where `phase vocoder` is used to fix this issue.

3 Phase Vocoder

Phase Vocoder consists of 3 stages:

- Analysis
- Processing
- Synthesis

3.1 Analysis

The core of `phase vocoder` is Short-Time Fourier Transform(STFT) which is basically the normal Fast Fourier Transform combined with windowing. windowing means taking a small chunk of a large signal(looking at the signal in through small window of time). Windowing affects the alters the spectrum of a signal so the window we will use should minimize this effect. Considering trade-offs a `Hann` window of size N is used.(as the journal suggests) We the take the FFT of a chunk of signal with N samples:

$$X_i[k] = \sum_{0}^{N-1} x[n + i * hop_a]w[n]e^{-j(\frac{2k\pi n}{N})} \quad 0 \leq k < N$$

Where $x[n]$ is the original signal, $w[n]$ is the `Hann` window, $X_i[k]$ is the spectrum of chunk i. Also chunks have overlapping in order to increase the resolution of the spectrum. Finally hop_a refers to the number of samples between two consecutive chunks which is $\frac{N}{4}$ for 75% overlapping.

3.2 Processing

After FFT we have N frequency bins from 0 to $\frac{(N-1)}{N}f_s$ with $\frac{f_s}{N}$ jumps where f_s is the sample rate. If a signal lies between two frequency bins, Its energy will be divided between its neighbor bins. The phase information between two consecutive chunks is important, i.e. the phase shift (The phase difference between the two chunks) can be used to figure out the true frequency of a bin. Because phase information is provided by the FFT the frequencies are between $-\pi$ and π so to determine the true frequency we use the following equation:

$$(w_{true}[k])_i = w_{bin}[k] + (w_{FFT}[k])_i$$

Where k and i are bin and chunk indices respectively. Now the new phase of each bin is calculated by multiplying the true frequency with the time interval of the synthesis stage:

$$(\phi_s)_i[k] = (\phi_s)_{i-1}[k] + \Delta t_s * (w_{true}[k])_i$$

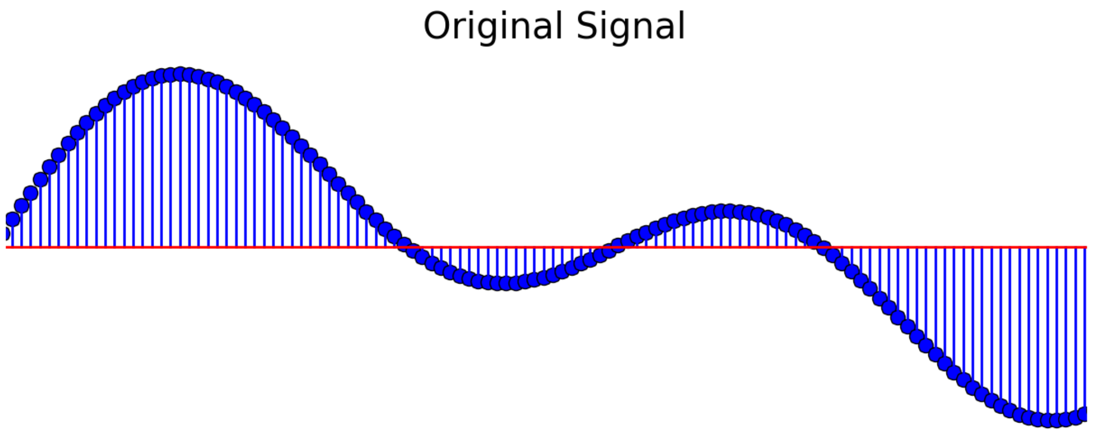
3.3 Synthesis

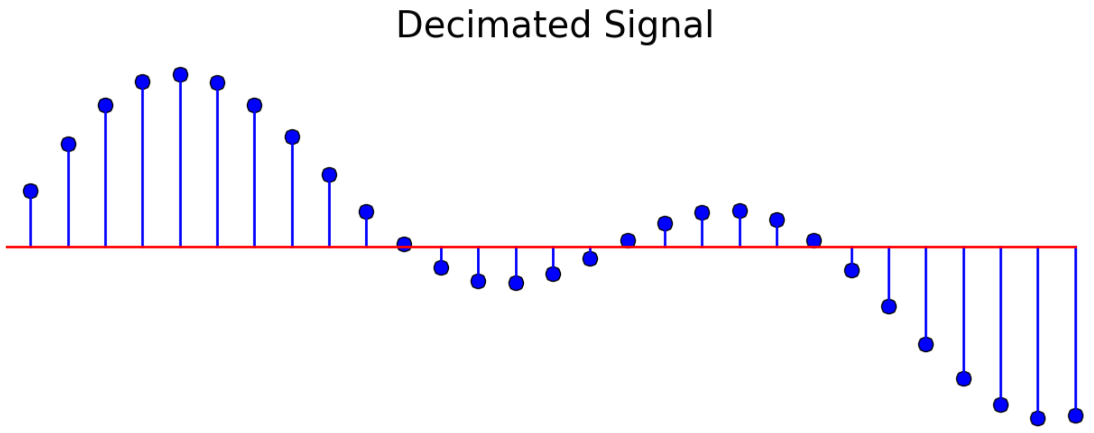
So now that the phase of the current chunk of the signal is adjusted we take the iFFT of each of the frame spectrums and apply a `Hann` window again to smooth the resulting signal $q_i[n]$. Finaly each frame is *overlap and added* (OLA) as shown below:

$$y[n] = \sum_{0}^{L-1} q_i[n - i * hop_s](u[n - i * hop_s] - u[n - i * hop_s - N])$$

4 Resampling

Now that we have stretched or compressed the signal without changing the pitch we have to resample the resulting signal to change its duration back to original and thus shift the pitch.





5 Summary

What we did here is summarized in the following steps:

- compute the STFT of the signal, which is the FFT of a short, overlapping and smoothly windowed block of samples.
- perform some tasks and manipulations.
- and perform an inverse STFT by taking the iFFT on each chunk and adding the resulting waveform chunks, also called overlap and add (OLA).