p6_code

April 1, 2019

DTMF

1209 Hz 1336 Hz 1477 Hz 1633 Hz

697 Hz 1 2 3 A

770 Hz 4 5 6 B

852 Hz 7 8 9 C

941 Hz * 0 # D

In [ ]:
```
import math
import numpy as np
import time
import wave
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile
import pyaudio
import sys
import scipy.io.wavfile as wav
from numpy.lib import stride_tricks
```

In [ ]:
```
num_channels = 1
human_freq_rate = 44100
FORMAT = pyaudio.paFloat32
chunk_size = 1024
rec_tot_secs = 5
audio_output_file_name = "p6_173_toneCode.wav"
p = pyaudio.PyAudio()
```

In [ ]:
```
def gen_sin_wave(freq, len, rate):
    len = int(len * rate)
    factor = float(freq) * (math.pi * 2) / rate
    return np.sin(np.arange(len) * factor)
```

```
def add_sine_waves(f1, f2, len, rate):
    sine_wave1=gen_sin_wave(f1,len,rate)
    sine_wave2=gen_sin_wave(f2,len,rate)
    sum_sine_waves=sine_wave1+sine_wave2
    sa=np.divide(sum_sine_waves, 2.0)
    return sa

def play_music(st, freq=440, len=0.10, rate=44100):
    frames = []
    frames.append(gen_sin_wave(freq, len, rate))
    chunk = np.concatenate(frames) * 0.25
    st.write(chunk.astype(numpy.float32).tostring())

def play_code_music(st, phn_char, len=0.7, rate=44100):
    dailTone_freqs = { '8': (1330+6, 850+2), '2': (1330+6, 690+7), 'C': (1630+3, 850+2)
                  '4': (1200+9,760+10), '5': (1330+6, 760+10), '6': (1470+7, 760+10),
                  '7': (1200+9,850+2), '1': (1200+9,690+7), '9': (1470+7, 850+2), '3':
                  'B': (1630+3, 760+10), '0': (1330+6, 940+1), '#': (1470+7, 940+1), 'I
    dtmf_chars = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#', 'A', 'B'
    if type(phn_char) is not type(''):
        phn_char=str(phn_char)[0]
    phn_char = ''.join ([dd for dd in phn_char if dd in dtmf_chars])
    joined_chunks = []
    for digit in phn_char:
        digit=digit.upper()
        frames = []
        frames.append(add_sine_waves(dailTone_freqs[digit][0], dailTone_freqs[digit][1
        chunk = np.concatenate(frames) * 0.25
        joined_chunks.append(chunk)
        fade = 500 # 500ms
        fade_in = np.arange(0., 1., 1/fade)
        fade_out = np.arange(1., 0., -1/fade)
        chunk[:fade] = np.multiply(chunk[:fade], fade_in)
        chunk[-fade:] = np.multiply(chunk[-fade:], fade_out)
        time.sleep(0.1)

    X = np.array(joined_chunks, dtype='float32')
    st.write(X.astype(np.float32).tostring())



    for i in range(0, int(rec_tot_secs)):
        waveFile = wave.open(audio_output_file_name, 'wb')
        waveFile.setnchannels(num_channels)
        waveFile.setsampwidth(p.get_sample_size(FORMAT))
        waveFile.setframerate(human_freq_rate)
        waveFile.writeframes(X.astype(np.float32).tostring())
```

```
        waveFile.close()


def short_time_FT(sig, frameSize, overlapFac=0.5, window=np.hanning):
    win = window(frameSize)
    hop_sze = int(frameSize - np.floor(overlapFac * frameSize))
    taken_samps = np.append(np.zeros(int(np.floor(frameSize/2.0))), sig)
    columns = np.ceil( (len(taken_samps) - frameSize) / float(hop_sze)) + 1
    taken_samps = np.append(taken_samps, np.zeros(frameSize))
    frames = stride_tricks.as_strided(taken_samps, shape=(int(columns), frameSize), sti
    frames *= win

    return np.fft.rfft(frames)


def scale_freq_log(spec, sr=44100, factor=20.):
    timebins, freqbins = np.shape(spec)

    scale = np.linspace(0, 1, freqbins) ** factor
    scale *= (freqbins-1)/max(scale)
    scale = np.unique(np.round(scale))
    newspec = np.complex128(np.zeros([timebins, len(scale)]))
    for i in range(0, len(scale)):
        if i == len(scale)-1:
            newspec[:,i] = np.sum(spec[:,int(scale[i]):], axis=1)
        else:
            newspec[:,i] = np.sum(spec[:,int(scale[i]):int(scale[i+1])], axis=1)

    allfreqs = np.abs(np.fft.fftfreq(freqbins*2, 1./sr)[:freqbins+1])
    freqs = []
    for i in range(0, len(scale)):
        if i == len(scale)-1:
            freqs += [np.mean(allfreqs[int(scale[i]):])]
        else:
            freqs += [np.mean(allfreqs[int(scale[i]):int(scale[i+1])])]

    return newspec, freqs

def plot_spectrogram(wav_file_path, binsize=2**10, plotpath="p6_173_spectro.png", colo
    samplerate, taken_samps = wav.read(wav_file_path)

    s = short_time_FT(taken_samps, binsize)

    sshow, freq = scale_freq_log(s, factor=1.0, sr=samplerate)

    spec_Output = 20.*np.log10(np.abs(sshow)/10e-6)

    timebins, freqbins = np.shape(spec_Output)
```

```
        plt.figure(figsize=(15, 7.5))
        plt.imshow(np.transpose(spec_Output), origin="lower", aspect="auto", cmap=colormap
        plt.colorbar()

        plt.xlabel("time (s)")
        plt.ylabel("freq (hz)")
        plt.xlim([0, timebins-1])
        plt.ylim([0, freqbins])

        xlocs = np.float32(np.linspace(0, timebins-1, 5))
        plt.xticks(xlocs, ["%.02f" % l for l in ((xlocs*len(taken_samps)/timebins)+(0.5*bi
        ylocs = np.int16(np.round(np.linspace(0, freqbins-1, 10)))
        plt.yticks(ylocs, ["%.02f" % freq[i] for i in ylocs])

        if plotpath:
            plt.savefig(plotpath, bbox_inches="tight")
            plt.show()
        else:
            plt.show()

        plt.clf()

        return spec_Output

In [ ]: if __name__ == '__main__':
        st = p.open(format=pyaudio.paFloat32,
                        channels=1, rate=44100, output=1,frames_per_buffer=chunk_size)


        if len(sys.argv) != 2:
            # Enter the Number here to generate the Dial tone and spectrogram
            phn_char = "173"

        else:
            phn_char = sys.argv[1]
        play_code_music(st, phn_char)

        st.close()
        p.terminate()
        spec_Output = plot_spectrogram('p6_173_toneCode.wav')
```