


Branch: master ▾

Audio-Spectrum-Analyzer-in-Python / audio_spectrum_pt2_spectrum_analyzer.ipynb

Find file

Copy path

 markjay4k show the spectrum d51ef7a on Sep 11, 2017

1 contributor

174 lines (173 sloc) | 4.64 KB

RawBlameHistory



```
In [40]: """
Notebook for streaming data from a microphone in realtime

audio is captured using pyaudio
then converted from binary data to ints using struct
then displayed using matplotlib

scipy.fftpack computes the FFT

if you don't have pyaudio, then run

>>> pip install pyaudio

note: with 2048 samples per chunk, I'm getting 20FPS
when also running the spectrum, its about 15FPS
"""

import pyaudio
import os
import struct
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft
import time
from tkinter import TclError

# to display in separate Tk window
%matplotlib tk

# constants
CHUNK = 1024 * 2          # samples per frame
FORMAT = pyaudio.paInt16  # audio format (bytes per sample?)
CHANNELS = 1              # single channel for microphone
RATE = 44100              # samples per second

In [54]: # create matplotlib figure and axes
fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 7))

# pyaudio class instance
p = pyaudio.PyAudio()

# stream object to get data from microphone
stream = p.open(
    format=FORMAT,
    channels=CHANNELS,
    rate=RATE,
    input=True,
    output=True,
    frames_per_buffer=CHUNK
)

# variable for plotting
x = np.arange(0, 2 * CHUNK, 2)          # samples (waveform)
xf = np.linspace(0, RATE, CHUNK)        # frequencies (spectrum)

# create a Line object with random data
line, = ax1.plot(x, np.random.rand(CHUNK), '-', lw=2)

# create semilogx line for spectrum
line_fft, = ax2.semilogx(xf, np.random.rand(CHUNK), '-', lw=2)

# format waveform axes
ax1.set_title('AUDIO WAVEFORM')
ax1.set_xlabel('samples')
ax1.set_ylabel('volume')
ax1.set_ylim(0, 255)
ax1.set_xlim(0, 2 * CHUNK)
plt.setp(ax1, xticks=[0, CHUNK, 2 * CHUNK], yticks=[0, 128, 255])

# format spectrum axes
ax2.set_xlim(20, RATE / 2)

print('stream started')

# for measuring frame rate
frame_count = 0
start_time = time.time()

while True:

    # binary data
    data = stream.read(CHUNK)

    # convert data to integers, make np array, then offset it by 127
    data_int = struct.unpack(str(2 * CHUNK) + 'B', data)

    # create np array and offset by 128
    data_np = np.array(data_int, dtype='b')[::2] + 128

    line.set_ydata(data_np)

    # compute FFT and update line
    yf = fft(data_int)
    line_fft.set_ydata(np.abs(yf[0:CHUNK]) / (128 * CHUNK))

    # update figure canvas
    try:
        fig.canvas.draw()
        fig.canvas.flush_events()
        frame_count += 1

    except TclError:

        # calculate average frame rate
        frame_rate = frame_count / (time.time() - start_time)

        print('stream stopped')
        print('average frame rate = {:.0f} FPS'.format(frame_rate))
        break

stream started
stream stopped
average frame rate = 15 FPS

In [ ]: 
```