HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Technische Grundlagen
der angewandten Informatik

PyAudio - Python

J. Keppler

Konstanz, 14. Dezember 2015

Zusammenfassung (Abstract)

Thema:	PyAudio - Python	
Autoren:	J. Keppler	jkeppler@htwg-konstanz.de

PyAudio ist eine Wrapper-Klasse für Python, welche den Zugriff auf die plattform unabhängig Audio-I/O-Bibliothek *PortAudio* ermöglicht.

Ausführliche Informationen zu PyAudio findet man im Internet unter folgenden Links:

- http://people.csail.mit.edu/hubert/pyaudio/
- https://pypi.python.org/pypi/PyAudio

Inhaltsverzeichnis

1

Einleitung

1.1 Installation

Python verwendet das Konzept "*packages*ßum Strukturieren von Modulen. **PyAudio** ist in Python geschrieben und stellt eine Wrapper-Klasse für die Funktionen aus dem Modul **_portaudio** dar. PortAudio ist eine Open-Source Audio-I/O-Bibiothek, welche in C und C++ programmiert wurde und plattformunabhängig ist.

Das **_portaudio** - Modul wurde unter Windows 7 für Python 3.4 mit 64 Bit compiliert. Dazu wurde Visual Studio 2013 verwendet.

Die Installation wird in folgenden Schritten durchgeführt:

1. Das Microsoft Visual C++ 2013 Redistributable Package (x64) [vcredist_x64.exe] wird zunächst auf dem Computer ausgeführt.

2. Die Datei pyaudio.py wird in das Unterverzeichnis <python directory>\Lib kopiert.

3. Die Datei _portaudio.pyd wird in das Unterverzeichnis <python directory>\DLLs kopiert.

2

Funktionen

2.0.1 pyaudio.PyAudio - Funktion

Diese Funktion erzeugt eine Instanz von PyAudio.

3

Klassen

3.0.2 PyAudio - Klasse

open - Methode

Öffnet einen Audio-Stream.

3.0.3 Stream - Klasse

read - Methode

Liest die Daten aus dem Audio-Stream.

Anhang

A.1 Audio Sample

```
import pyaudio
import numpy
import matplotlib.pyplot as plt

FORMAT = pyaudio.paInt16
SAMPLEFREQ = 44100
FRAMESIZE = 1024
NOFFRAMES = 220
p = pyaudio.PyAudio()
print('running')

stream = p.open(format=FORMAT,channels=1,rate=SAMPLEFREQ,
                input=True,frames_per_buffer=FRAMESIZE)
data = stream.read(NOFFRAMES*FRAMESIZE)
decoded = numpy.fromstring(data, 'Int16');

stream.stop_stream()
stream.close()
p.terminate()
print('done')
plt.plot(decoded)
plt.show()
```

Listing 4.1: Audio sample as float number from pyaudio-stream

A.2 Mikrofon Aufnahme auswerten

```
import pyaudio
import struct
import math

INITIAL_TAP_THRESHOLD = 0.010
FORMAT = pyaudio.paInt16
SHORT_NORMALIZE = (1.0/32768.0)
CHANNELS = 2
RATE = 44100
INPUT_BLOCK_TIME = 0.05
INPUT_FRAMES_PER_BLOCK = int(RATE*INPUT_BLOCK_TIME)
# if we get this many noisy blocks in a row, increase the
   threshold
OVERSENSITIVE = 15.0/INPUT_BLOCK_TIME
# if we get this many quite blocks in a row, decrease the
   threshold
UNDERSENSITIVE = 120.0/INPUT_BLOCK_TIME
# if th enoise was longer than this many blocks, it's not a '
   tap'
MAX_TAP_BLOCKS = 0.15/INPUT_BLOCK_TIME

def get_rms( block ):
    # RMS amplitude is defined as the square root of the
    # mean over time of the square of the amplitude.
    # So we need to convert this string of bytes into
    # a string of 16-bit samples...

    # we will get one short out for each
    # two xhars in the string.
    count = len(block)/2
    format = "%dh"%(count)
    shorts = struct.unpack( format, block )
```

```
    # iterate over the block.
    sum_squares = 0.0
    for sample in shorts:
        # sample is a signed short in +/- 32728.
        # normalize it to 1.0
        n = sample * SHORT_NORMALIZE
        sum_squares += n * n

    return math.sqrt( sum_squares / count )

class TapTester(object):
    def __init__(self):
        self.pa = pyaudio.PyAudio()
        self.stream = self.open_mic_stream()
        self.tap_threshold = INITIAL_TAP_THRESHOLD
        self.noisycount = MAX_TAP_BLOCKS+1
        self.quietcount = 0
        self.errorcount = 0

    def stop(self):
        self.stream.close()

    def find_input_device(self):
        device_index = None
        for i in range( self.pa.get_device_count() ):
            devinfo = self.pa.get_device_info_by_index(i)
            print( "Device %d: %s"%(i,devinfo["name"]) )

            for keyword in ["mic","input"]:
                if keyword in devinfo["name"].lower():
                    print( "Found an input: device %d - %s"%(
                        i,devinfo["name"]) )
                    device_index = i
                    return device_index
```

```
        if device_index == None:
            print( "No prefeered input found; using deafult
               input device." )

        return device_index

    def open_mic_stream(self):
        device_index = self.find_input_device()

        stream = self.pa.open( format = FORMAT,
                               channels = CHANNELS,
                               rate = RATE,
                               input = True,
                               input_device_index =
                                  device_index,
                               frames_per_buffer =
                                  INPUT_FRAMES_PER_BLOCK )

        return stream

    def tapDetected(self):
        print( "Tap!" )

    def listen(self):
        try:
            block = self.stream.read(INPUT_FRAMES_PER_BLOCK)
        except IOError as e:
            # dammit.
            self.errorcount += 1
            print( "(%d) Error recording: %s"%(self.
               errorcount,e) )
            self.noisycount += 1
            return
```

```
        amplitude = get_rms( block )
        if amplitude > self.tap_threshold:
            # noisy block
            self.quietcount = 0
            self.noisycount += 1
            if self.noisycount > OVERSENSITIVE:
                # tunr down the sensitivity
                self.tap_threshold *= 1.1
            else:
                # quiet block.
                if 1 <= self.noisycount <= MAX_TAP_BLOCKS:
                    self.tapDetected()
                self.noisycount = 0
                self.quietcount += 1
                if self.quietcount > UNDERSENSITIVE:
                    # turn up the sensitivity
                    self.tap_threshold *= 0.9

if __name__ == "__main__":
    tt = TapTester()

    for i in range(1000):
        tt.listen()
```

Listing 4.2: Detect tap with pyaudio from live mic