pyrecspecwaterfallsampling

January 25, 2017

1 This program shows the characteristics of the audio recorded via a waterfall(going up) spectrogram where the horizontal shows the frequencies for the coreesponding block of the recording and the vertical axis is the time.

1.0.1 Input:

As the program runs the recording starts and wait for thr user inputs from the keyboard. Below are the key inputs and their effects: ### 's' - turn on/off the sampling ### 'f' - turn on/off the filtering (to remove aliasing) ### 'q' - quit the recording and hence the program(program will not end by clicking close on the window, so instead press 'q')

1.0.2 Output:

Real time audio spectrogram for the processed blocks of the recording moving upward(like an upside down waterfall).

Import relevant modules.

```
In [1]: """
        Using Pyaudio, record sound from the audio device and plot a waterfall spectrum display,
        aliasing. It includes a low pass filter for aliasing reduction.
        Filtering and sampling can be toggled on and off using the key 'f' for filter on/off, an
        differences. Use key 'q' to quit.
        Usage example: python pyrecspecwaterfallsampling.py
        Gerald Schuller, August 2015
        """
        %matplotlib inline
        import pyaudio
        import struct
        import numpy as np
        import scipy.signal
        import cv2
```

Define the variables

```
In [2]: CHUNK = 2048 #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
        N=8.0;    #sampling rate
```

Initialize the sound card and print out audio input and output properties

```
In [3]: p = pyaudio.PyAudio()

        a = p.get_device_count()
        print("device count=",a)

        for i in range(0, a):
            print("i = ",i)
            b = p.get_device_info_by_index(i)['maxInputChannels']
            print(b)
            b = p.get_device_info_by_index(i)['defaultSampleRate']
            print(b)

        stream = p.open(format=p.get_format_from_width(WIDTH),
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        output=True,
                        #input_device_index=3,
                        frames_per_buffer=CHUNK)
```

```
('device count=', 16L)
('i = ', 0)
2
44100.0
('i = ', 1)
2
44100.0
('i = ', 2)
0
44100.0
('i = ', 3)
0
44100.0
('i = ', 4)
0
44100.0
('i = ', 5)
2
44100.0
('i = ', 6)
```

```
2
44100.0
('i = ', 7)
0
44100.0
('i = ', 8)
0
44100.0
('i = ', 9)
0
44100.0
('i = ', 10)
0
44100.0
('i = ', 11)
0
44100.0
('i = ', 12)
2
44100.0
('i = ', 13)
0
44100.0
('i = ', 14)
2
44100.0
('i = ', 15)
0
44100.0
```

Designing an IIR filter

```
In [4]: #Our Lowpass Filter:
        [b,a]=scipy.signal.iirfilter(4, 1900.0/16000,rp=60,btype='lowpass')
        #Memory for the filter:
        zd=np.zeros(5-1)
        zu=np.zeros(5-1)
```

```
In [5]: print("Program to demonstrate audio aliasing and anti-aliasing filtering")
        print("Downsampling factor: N=8")
        print("Toggle filter before and after sampling on/off: press key 'f'")
        print("Toggle sampling on/off: press key 's'")
        print("To quit press key 'q'")
```

```
Program to demonstrate audio aliasing and anti-aliasing filtering
Downsampling factor: N=8
Toggle filter before and after sampling on/off: press key 'f'
Toggle sampling on/off: press key 's'
To quit press key 'q'
```

Recording audio and displaying waterfall for it.

```
In [6]: print("* recording")
        #Size of waterfall diagramm:
        #max CHUNK/2 rows:
        rows=500
        cols=512
        fftlen=cols*2
        frame=0.0*np.ones((rows,cols,3));
        frametxt=frame.copy()

        filteron=False
        downsampleon=False
        ctr=0
        cv2.putText(frame,"Audio Spectrogram", (20,50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,128,12
        cv2.putText(frame,"Toggle sampling on/off: key s", (20,100), cv2.FONT_HERSHEY_SIMPLEX, 1
        cv2.putText(frame,"(downsampling followed by upsampling)", (20,150), cv2.FONT_HERSHEY_SI
        cv2.putText(frame,"Toggle LP Filter: key f", (20,200), cv2.FONT_HERSHEY_SIMPLEX, 1, (255
        cv2.putText(frame,"(LP filter before and after sampling)", (20,250), cv2.FONT_HERSHEY_SI
        cv2.putText(frame,"Quit: key q", (20,300), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,128,128))
        cv2.putText(frame,"Sampling Freq.="+str(RATE)+"Hz", (20,350), cv2.FONT_HERSHEY_SIMPLEX,
        cv2.putText(frame,"Downsample Rate="+str(N), (20,400), cv2.FONT_HERSHEY_SIMPLEX, 1, (255

        while(True):
            ctr=ctr+1
            #Reading from audio input stream into data with block length "CHUNK":
            data = stream.read(CHUNK)
            #Convert from stream of bytes to a list of short integers (2 bytes here) in "samples
            #shorts = (struct.unpack( "128h", data ))
            shorts = (struct.unpack( 'h' * CHUNK, data ));
            samples=np.array(list(shorts),dtype=float);

            #start block-wise signal processing:
            #Low pass filter *before downsampling*:
            if filteron==True:
               [samples,zd]=scipy.signal.lfilter(b, a, samples, zi=zd)

            #Compute a block/an array of a unit pulse train corresponding a downsampling rate of

            #s=np.modf(np.arange(0,CHUNK)/N)[0]==0.0
            #make unit pulse train with modulus function "%":
            s=(np.arange(0,CHUNK)%N)==0
            #The sampling:
            #multiply the signal with the unit pulse train:
```

```
       if downsampleon == True:
          samples=samples*s;

       #Lowpass filtering *after upsampling*:
       #filter function:
       if filteron==True:
          [samples,zu]=scipy.signal.lfilter(b, a, samples, zi=zu)

       #end signal processing

       #play out samples:
       samples=np.clip(samples, -32000,32000)
       #converting from short integers to a stream of bytes in "data":
       data=struct.pack('h' * len(samples), *samples);
       #Writing data back to audio output stream:
       stream.write(data, CHUNK)

       if (ctr%4 ==0):
          #shift "frame" 1 up:
          frame[0:(rows-1),:]=frame[1:rows,:];
          #compute magnitude of 1D FFT of sound
          #with suitable normalization for the display:
          #frame=np.abs(np.ffqt.fft2(frame[:,:,1]/255.0))/512.0
          #write magnitude spectrum in lowes row of "frame":
          R=0.25*np.log((np.abs(np.fft.fft(samples[0:fftlen])[0:(fftlen/2)]/np.sqrt(fftlen)
          #Color mapping:
          #Red:
          frame[rows-1,:,2]=R
          #Green:
          frame[rows-1,:,1]=np.abs(1-2*R)
          #Blue:
          frame[rows-1,:,0]=1.0-R
          #frame[rows-1,:,0]=frame[rows-1,:,1]**3
          # Display the resulting frame

          cv2.imshow("Audio Spectrogram, filter: f, sampling: s, quit:q",frame+frametxt)

       #Keep window open until key 'q' is pressed:
       key=cv2.waitKey(1) & 0xFF;
       if key == ord('f'):
          filteron= not filteron
          cv2.putText(frame,"filter="+str(filteron), (20,498), cv2.FONT_HERSHEY_SIMPLEX, 0.
       if key == ord('s'):
          downsampleon= not downsampleon
          cv2.putText(frame,"sampling="+str(downsampleon), (20,498), cv2.FONT_HERSHEY_SIMPL
       if key == ord('q'):
           break
   # When everything done, release the capture
```

```
        cv2.destroyAllWindows()

        stream.stop_stream()
        stream.close()
        p.terminate()
```

```
* recording


c:\python27\lib\site-packages\ipykernel\__main__.py:56: DeprecationWarning: integer argument exp
```