pyrecplayMDCT

May 3, 2017

1 Python Example: *pyrecplayMDCT*

1.1 Importing the relevant modules:

```
In [1]: import pyaudio
        import struct
        import numpy as np
        import cv2
        import scipy.fftpack as spfft
        %matplotlib inline
```

1.2 Defining the variables:

```
In [2]: N=1024 #Number of subbands and block size
        CHUNK = N #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
```

2 Defining the delay matrix:

```
In [3]: def Dmatrix(samples):
        #implementation of the delay matrix D(z)
           #Delay elements:
           out=np.zeros(N)
           out[0:(N/2)] = Dmatrix.z
           Dmatrix.z = samples[0:(N/2)]
           out[N/2:N] = samples[N/2:N]
           return out
```

2.1 The inverse D(z) matrix:

```
In [4]: def Dmatrixinv(samples):
        #implementation of the delay matrix D(z)
           #Delay elements:
           out=np.zeros(N)
           out[(N/2):N]=Dmatrixinv.z
```

```
        Dmatrixinv.z=samples[(N/2):N]
        out[0:N/2]=samples[0:N/2]
        return out
```

```
In [5]: Dmatrixinv.z=np.zeros(N/2)
```

2.2 The F Matrix:

```
In [6]: fcoeff = np.sin(np.pi / (2 * N) * (np.arange(0, 2 * N) + 0.5))
        Fmatrix = np.zeros((N, N))
        Fmatrix[0 : N / 2, 0 : N / 2] = np.fliplr(np.diag(fcoeff[0 : N / 2]))
        Fmatrix[N / 2 : N, 0 : N / 2] = np.diag(fcoeff[N / 2 : N])
        Fmatrix[0 : N / 2, N / 2 : N] = np.diag(fcoeff[N: (N + N / 2)])
        Fmatrix[N / 2 : N, N / 2 : N] = -np.fliplr(np.diag(fcoeff[(N + N / 2) : (2 * N)]))
```

2.3 The inverse F matrix:

```
In [7]: Finv=np.linalg.inv(Fmatrix)
```

2.4 The DCT4 transform:

```
In [8]: def DCT4(samples):
            #use a DCT3 to implement a DCT4:
            samplesup = np.zeros(2 * N)
            #upsample signal:
            samplesup[1 :: 2]=samples
            y=spfft.dct(samplesup, type = 3) / 2
            return y[0 : N]
```

2.5 The complete MDCT, Analysis:

```
In [9]: def MDCT(samples):
            y=np.dot(samples,Fmatrix)
            y=Dmatrix(y)
            y=DCT4(y)
            return y
```

2.6 The inverse MDCT, synthesis:

```
In [10]: def MDCTinv(y):
             #inverse DCT4 is identical to DCT4:
             x=DCT4(y)*2/N
             #inverse D(z) matrix
             x=Dmatrixinv(x)
             #inverse F matrix
             x=np.dot(x,Finv)
             return x
```

2.7 Inilitialise sound card

```
In [11]: p = pyaudio.PyAudio()

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

```
('device count=', 11L)
```

2.8 Prints out all the audio inputs and the input sampling rate

```
In [12]: 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)
```

```
('i = ', 0)
2
44100.0
('i = ', 1)
0
44100.0
('i = ', 2)
128
48000.0
('i = ', 3)
0
44100.0
('i = ', 4)
0
44100.0
('i = ', 5)
0
44100.0
('i = ', 6)
0
44100.0
('i = ', 7)
32
44100.0
('i = ', 8)
0
48000.0
('i = ', 9)
32
44100.0
```

```
('i = ', 10)
16
44100.0
```

```
In [13]: 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)
```

```
In [ ]: print("* recording")

        #Size of waterfall diagramm:
        #max CHUNK/2 cols:
        rows=500
        cols=CHUNK
        fftlen=cols
        frame=0.0*np.ones((rows,cols,3))


        while(True):

            #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);

            #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))+
            #This is the FFT of the input:
            #y=np.fft.fft(samples[0:fftlen])
            #This is the analysis MDCT of the input:
            y=MDCT(samples[0:fftlen])
            #yfilt is the processed subbands, processing goes here:
            yfilt=y
            #yfilt=np.zeros(N)
            #yfilt[20:100]=y[20:100]
```

```
    #Waterfall color mapping:
    R=0.25*np.log((np.abs(yfilt/np.sqrt(fftlen))+1))/np.log(10.0)
    #Red frame:
    frame[rows-1,:,2]=R
    #Green frame:
    frame[rows-1,:,1]=np.abs(1-2*R)
    #Blue frame:
    frame[rows-1,:,0]=1.0-R
    #frame[rows-1,:,0]=frame[rows-1,:,1]**3
    # Display the resulting frame
    cv2.imshow('frame',frame)

    #Inverse FFT:
    #xrek=np.real(np.fft.ifft(yfilt))
    #Inverse/synthesis MDCT:
    xrek=MDCTinv(yfilt);
    #converting from short integers to a stream of bytes in "data":
    #data=struct.pack('h' * len(samples), *samples);
    data=struct.pack('h' * len(xrek), *xrek);
    #Writing data back to audio output stream:
    stream.write(data, CHUNK)

    #Keep window open until key 'q' is pressed:
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# When everything done, release the capture

cv2.destroyAllWindows()

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