pyrecplay_samplingblock

January 25, 2017

1 This program downsamples the audio recorded through the selected microphone and then plays its downsampled version back.

1.0.1 Input:

As the program starts, it starts recording and ends after 8 seconds. The input is then processed blockwise and downsamples it by the factor 'N' by multiplying an impulse train repeated after every 'N'th samples in a block(sets rest of the samples to zero).

1.0.2 Output:

Output is the real time downsampled recording. The output is now downsampled by the factor 'N'.

Import the relevant modules.

```
In [1]: """
        PyAudio Example: Make a sampling between input and output (i.e., record a
        few samples, multiply them with a unit pulse train, and play them back immediately).
        Using block-wise processing instead of a for loop
        Gerald Schuller, Octtober 2014
        """

        import pyaudio
        import struct
        #import math
        #import array
        import numpy as np
        #import scipy
```

Define the variables.

```
In [2]: CHUNK = 5000 #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
        RECORD_SECONDS = 8
```

Initialize the sound card.

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

        stream = p.open(format=p.get_format_from_width(WIDTH),
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        output=True,
                        #input_device_index=10,
                        frames_per_buffer=CHUNK)
```

Starts recording and plays back the downsampled version of the recording.(Blockwise Downsampling)

```
In [4]: print("* recording")

        #Loop for the blocks:
        for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
            #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=list(shorts);

            #start block-wise signal processing:

            #Compute a block/an array of a unit pulse train corresponding a downsampling rate of
            N=8.0;
            #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
            #multiply the signal with the unit pulse train:
            samples=samples*s;

            #end signal processing

            #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)

        print("* done")

        stream.stop_stream()
        stream.close()

        p.terminate()
```

```
* recording
* done
```