Lab 5: Microphone input, Circular Buffer, & Blocking

DSP Lab (EE 4163 / EL 6183)

Last Edit: Saturday 8th October, 2016 08:15

1 Microphone input

The demo program `filter_play_mic.py` takes audio input from the microphone, filters it using a bandpass filter, and plays the output audio signal to the speakers.

When reading the input signal from the microphone it is recommended that headphones be used to avoid feedback problems (sound passing from the speaker back into the microphone).

```python
# filter_play_mic.py
# Record from micrphone, filter signal, play output to speaker

import pyaudio
import struct
import math
# import wave

from myfunctions import clip16

WIDTH = 2           # Number of bytes per sample
CHANNELS = 1        # mono
RATE = 16000        # Sampling rate (frames/second)
DURATION = 10        # duration of processing (seconds)

N = DURATION*RATE   # N : Number of samples to process


# Difference equation coefficients
b0 =  0.008442692929081
b2 = -0.016885385858161
b4 =  0.008442692929081

# a0 =  1.000000000000000
a1 = -3.580673542760982
a2 =  4.942669993770672
a3 = -3.114402101627517
a4 =  0.757546944478829

# Initialization
x1 = 0.0
x2 = 0.0
x3 = 0.0
x4 = 0.0
y1 = 0.0
y2 = 0.0
y3 = 0.0
y4 = 0.0

p = pyaudio.PyAudio()
```

```python

# Open audio stream
stream = p.open(format = p.get_format_from_width(WIDTH),
                channels = CHANNELS,
                rate = RATE,
                input = True,
                output = True)

print("**** Start ****")

for n in range(0, N):

    # Get one frame from audio input (microphone)
    input_string = stream.read(1)

    # Convert binary string to tuple of numbers
    input_tuple = struct.unpack('h', input_string)

    # Convert one-element tuple to number
    input_value = input_tuple[0]

    # Set input to difference equation
    x0 = input_value

    # Difference equation
    y0 = b0*x0 + b2*x2 + b4*x4 - a1*y1 - a2*y2 - a3*y3 - a4*y4

    # Delays
    x4 = x3
    x3 = x2
    x2 = x1
    x1 = x0
    y4 = y3
    y3 = y2
    y2 = y1
    y1 = y0

    # Compute output value
    output_value = clip16(y0)    # Number

    # output_value = clip16(x0)   # Bypass filter (listen to input directly)

    # Convert output value to binary string
    output_string = struct.pack('h', output_value)

    # Write binary string to audio stream
    stream.write(output_string)

print("**** Done ****")

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

1.1 Exercises

1. Modify the demo program so it plays a stereo signal. In one channel, play the input audio signal. In the other channel, play the output of the bandpass filter. The two signals should play simultaneously.

2. Modify the demo program by specifying the optional PyAudio stream parameter `frames_per_buffer`. SUBMIT Simply use the following code segment. Run the program with FPB set to each of the values: 1024,

256, 64. Comment on your observation — what is the effect of this parameter?

```
# Open audio stream
stream = p.open(format = p.get_format_from_width(WIDTH),
                channels = CHANNELS,
                rate = RATE,
                input = True,
                output = True,
                frames_per_buffer = FPB)
```

2 Basic delay effects

Reading: First sections of Chapter 2 of text book.

Several demo programs, shown in class, are available of the course web page. The following demo program implements the basic delay (with no feedback). An important point is the use of a circular buffer in the implementation. The delay and gain parameters are set within the program.

```
# play_delay.py
# Reads a specified wave file (mono) and plays it with a delay.
# This implementation uses a circular buffer.

import pyaudio
import wave
import struct
import math
from myfunctions import clip16

wavfile = 'author.wav'
print('Play the wave file %s.' % wavfile)

# Open the wave file
wf = wave.open( wavfile, 'rb')

# Read the wave file properties
SIGNAL_LEN  = wf.getnframes()   # Signal length
CHANNELS = wf.getnchannels()    # Number of channels
WIDTH = wf.getsampwidth()       # Number of bytes per sample
RATE = wf.getframerate()        # Sampling rate (frames/second)

print('The file has %d channel(s).'          % CHANNELS)
print('The frame rate is %d frames/second.'  % RATE)
print('The file has %d frames.'              % SIGNAL_LEN)
print('There are %d bytes per sample.'       % WIDTH)

# Set parameters of delay system
Gdp = 1.0           # direct-path gain
Gff = 0.8           # feed-forward gain
delay_sec = 0.05    # 50 milliseconds
# delay_sec = 0.02
delay_samples = int( math.floor( RATE * delay_sec ) )

print('The delay of {0:.3f} seconds is {1:d} samples.'.format(delay_sec, delay_samples))

# Create a buffer to store past values. Initialize to zero.
BUFFER_LEN = delay_samples   # set the length of buffer
buffer = [ 0 for i in range(BUFFER_LEN) ]

```

```
# Open an output audio stream
p = pyaudio.PyAudio()
stream = p.open(format      = pyaudio.paInt16,
                channels    = 1,
                rate        = RATE,
                input       = False,
                output      = True )


# Get first frame (sample)
input_string = wf.readframes(1)

k = 0       # buffer index (circular index)

print ('* Start...')

while input_string != '':

    # Convert string to number
    input_value = struct.unpack('h', input_string)[0]

    # Compute output value
    output_value = Gdp * input_value + Gff * buffer[k]
    output_value = clip16(output_value)

    # Update buffer
    buffer[k] = input_value
    k = k + 1
    if k >= BUFFER_LEN:
        k = 0

    # Convert output value to binary string
    output_string = struct.pack('h', output_value)

    # Write output value to audio stream
    stream.write(output_string)

    # Get next frame (sample)
    input_string = wf.readframes(1)

print('* Done')

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

2.1 Exercises

1. What is the difference equation and transfer function of the digital filter system implemented in the program above (with gain parameters G_{dp}, G_{ff}, and delay of N samples)? What are the poles and zeros? Use Matlab to plot the pole-zero diagram of the filter. Are there any parameter settings that make the system unstable? Explain.

2. Experiment with different delay and gain parameters. How do short delays (e.g., less than 50 milliseconds) and long delays (e.g., longer than 0.2 seconds) sound different?

3. Modify the demo program so it produces a stereo output, with a different delay in right and left channels.

4. Note that the provided demo program truncates the output audio before it is finished (the end of the delayed signal is truncated). It is not noticeable for short delays or for wav files ending with a sufficiently long period of silence, but in other cases it may be noticeable. Modify the demo program so that the output signal is not truncated at the end (i.e., so that the trailing end of the final echo is played.)

5. Modify the demo program so the input audio is from the microphone.

6. Modify the demo program so the output signal is saved to a wave file.

7. Modify the demo program so the input audio is from the microphone, and the output signal is saved to a wave file. Create a wave file of applying the filter to yourself saying your name; and submit your wave file as part of your work. SUBMIT

8. The demo programs provide two implementations of the circular buffer: one program uses a minimal length buffer and one buffer index, the other program uses a longer buffer and two buffer indices. They give identical output signals, even though the circular buffer is implemented differently. Explain.

9. Modify the simple vibrato demo program `play_vibrato_simple.py` so it plays a stereo output signal. Use different vibrato parameters in the left and right channels. SUBMIT

10. Modify the simple vibrato demo program `play_vibrato_simple.py` so the audio input is from the microphone (not a wave file).

3 Blocking

The demo program `playAM_blocking_fix.py` applies amplitude modulation (AM) to a signal obtained from a wave file. This moves the signal to higher frequencies and changes the way the signal sounds.

3.1 Exercises

1. Modify the demo program so the audio input is read from the microphone instead of a wave file.

2. Modify the demo program so it produces a stereo output signal, where a different modulation frequency is used for left and right channels. Listen to the output using headphones.

3. (Combination of 1. and 2.) Modify the demo program so it produces a stereo output signal, where a different modulation frequency is used for left and right channels. The output stereo signal should be saved to a wave file. Listen to the output using headphones. Submit your wave file as part of your work. SUBMIT