Lab 6: Circular Buffer 2, FFT, & Plotting

DSP Lab (EE 4163 / EL 6183)

Last Edit: Saturday 15th October, 2016 19:15

1 Blocking

The usual practice is to read and write blocks of samples to and from audio devices rather than one sample at a time. Demo programs are available on the course web page, including a program that reads the input signal from the microphone. 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).

The demo program `play_vibrato_simple.py` is a simple implementation of the vibrato effect. This implementation is poor because the time-varying fractional delay is implementing by rounding the delay to an integer. For better audio quality, interpolation is usually used instead, as in the demo program `play_vibrato_interpolation.py`

1.1 Exercises

1. The demo program `play_vibrato_interpolation.py` does not use blocking (it reads and writes a single frame at a time). Write a version of this program that reads, processes, and writes the audio signal in blocks.

2. Modify the demo program `play_vibrato_interpolation.py` so that it reads and writes the audio signal in blocks (as in previous exercise), and takes the input signal from the microphone instead of a wave file.

3. Modify the demo program `play_vibrato_interpolation.py` so that it reads and writes the audio signal in blocks (as in previous exercise), takes the input signal from the microphone instead of a wave file (as in previous exercise), and produces a stereo output audio signal. The left and right channels of the output signal should have different vibrato parameters (frequency and amplitude).

4. Write a Python program to implement the flanger effect. As described in Chapter 2 of the text book, the flanger effect is like the vibrato effect but it additionally has a direct path, as shown in the figure. The input signal should be read from a wave file.

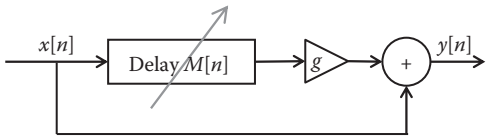

Figure 2.11
Block diagram of a basic flanger without feedback. The delay length $M[n]$ changes over time.

2 Plotting

This demo program `play_randomly_plots.py` plays notes randomly and plots the waveform on the screen. This requires the Python library `matplotlib`. See the tutorial at

`http://matplotlib.org/users/pyplot_tutorial.html`

2.1 Exercises

1. **Stereo.** Write a program based on the demo program `play_randomly.py` that produces stereo audio. The sounds in the left and right channels should start at independent times. To do that, create two independent input signals to go through two independent filters. The output of one filter goes the the left channel; the output of the other filter goes the right channel.

2. **Stereo with plotting.** Modify your program in the previous exercise so that it plots both channels of the signal — use a different color for left and right channels. The two waveforms in the plot maybe vertically offset from on another to improve legibility.

3. **Amplitude Modulation (AM).** The Python program `playAM_blocking_fix.py` applies amplitude modulation to a speech signal. Modify this program so that it displays in a figure window the frequency spectra of both the input and output signal in real-time. What is the relation between the two specta? Is the spectrum shifted by the expected amount?

3 Filtering

1. **Filtering (1).** Write a program that takes the input audio from a microphone and applies a bandpass filter. Use a bandpass filter with a pass-band from 500 Hz to 1000 Hz. For example, you can use the Butterworth filter obtain in Maltab with the command
   `[b, a] = butter(2, [500 1000]*2/Fs).`
   Play the input and output signals simultaneous on left and right channels of a stereo signal. The program should read and write the audio in blocks (not one frame at a time). Plot the input and output signals in real-time in the same figure window.

2. **Filtering (2).** Same as previous exercise, but plot the *frequency spectrum* (Fourier transform) of the input and output signals in the same figure window. Use the FFT to compute the spectrum. What is the relation between the two spectra?

3. **lfilter.** In some demos we have used the Matlab function `filter` which implements a difference equation. In Python, a similar function called `lfilter` is available in the SciPy library for scientific computing. (Here `lfilter` means *linear* filter.)

   `http://docs.scipy.org/doc/scipy/reference/signal.html`

   Modify your program from the previous exercise so that the filter is implemented using both the `lfilter` function and by explicit calculation as it is done for example in the demo program `filter_play_wav_mono.py`. Verify that both ways yield the same output signal by (1) plotting the time-domain waveform of the error between the two signals, (2) playing the output signals produced by each filter on the left and right channels of a stereo signal, respectively (listen to output using headphones).

   To avoid transient artifacts at the start of each block, you should specify the initial delay values `zi` in the `lfilter` function based on the final delay values `zf` from the previous block.

4. We have used the Matlab function `butter` to obtain the coefficients of a digital Butterworth filter. In Python, there is also a function `butter` in the SciPy library `scipy.signal`. Verify that the Python functions yields the same coefficients as the Matlab function.

4 Instantaneous Frequency

The instantaneous frequency of the signal $x(t) = \sin(\phi(t))$ is given by the derivative $f_{\text{inst}}(t) = \frac{1}{2\pi}\phi'(t)$. Find a signal such that the instantaneous frequency increases from 200 Hz to 500 Hz during a 5 second interval. Implement the generation of this signal in real-time using Python. Play the sound on the speaker as the signal is generated. Plot the time-domain waveform and frequency spectrum of the signal as it is played.

5 Circular Buffer with recursion

The demo programs `play_feedbackdelay1_test01.py` and `play_feedbackdelay1.py` implement a digital filter using a circular buffer with recursion.

Determine the difference equation for the filter implemented by each of these demo programs.