CPE 367 – Experiment 4a – v3
Delay Effects: Positional Audio and Echo – Dr F DePiero (48 Points)

Overview and Motivation

The signal processing in this experiment involves digital filters with particular delay characteristics. Often when creating digital filters, the magnitude characteristics of the frequency response are the most critical. (For example, a high pass or low pass filter). In this experiment we explore the delay instead. This experiment will also provide a first introduction to IIR filters.

3-D positional audio is an emerging technology. In the near-term media creators will be able to create a script for short sound clips in a fashion similar to actors' scripts and action choreography. The positional audio data defines when and where the sound clip should occur (within a home or theater, for example). The positional effect is accomplished by controlling the amount of delay for a given sound clip, as it is added into the signal for each speaker of a surround sound system. We will implement the positional audio by creating a stereo output signal, using an input mono signal. You will probably get best results by listening with headphones. In addition to positional audio, we will also implement echo effects.

An echo can be created using an FIR difference equation

$$y[n] = b_! \quad x[n] \quad + \quad b_" \quad x[n-d] \qquad \text{(Eq. 1)}$$

This operation mixes together a delayed sample with the current sample to yield a single echo. Meaning that a sound such as a handclap would be repeated exactly once. Alternatively, we can use an IIR approach

$$y[n] = b_! \quad x[n] \quad + \quad a_" \quad y[n-d] \qquad \text{(Eq. 2)}$$

In this approach the sound of a handclap would repeat theoretically forever. In practice, once the signal's absolute value drops below the smallest representable (integer) value, it will round to zero. For the 16 bit WAV files the lower limit is 1 (out of 2^16).

Learning Objectives

- Find the equivalent difference equation for an IIR system
- Implement both FIR and IIR filters in Python
- Determine the impulse response for a simple IIR system

Prerequisite Learning Objectives

- Implement a digital filter in Python
- Python programming with WAV file I/O and some experience with MatLab

Procedures, Questions and Deliverables

1) Positional Audio

A simple 1D version of positional audio can be implemented by creating a stereo signal. Equations 3a and 3b define outputs for the right and left channels of a stereo WAV file.

$y_{\#}[n] = x[n - d_{\#}]$ (Eq. 3a) $y_{\$}[n] = x[n - d_{\$}]$ (Eq. 3b)

To create the illusion of a position for the sound source, delays d_R and d_L are chosen based on the speed of sound propagation. If a sound reaches each ear simultaneously then the source appears to be directly ahead of the listener. If the sound reaches the left ear first then the sound appears to be to the left, and similarly for the right. For a sound source that seems to be fully to the left of the listener, then the relative time difference should be approximately equal to the amount of time for sound to propagate through a distance equal to the width of a person's head. Assume the distance between the ears is ~8 inches and that the speed of sound is 343 m/s. Work with a sample rate of 16 kHz.

- **1a) How long does it take sound to propagate 8 inches? (in samples) (2)**

Write a Python program that uses the joy.wav file for input and generates a stereo WAV file that implements delays, as described by Equations 3a & 3b. Your program should gradually shift the apparent position from fully one side to the other in approximately 4 seconds. (This is twice as long as the example provided). Use your FIFO class.

- **1b) Documentation: Paste the sections of your Python code that implements the difference equation and the gradual shift (6)**

```
def Differential_manchester(inp):
    inp1=list(inp)
    li,lock,pre=[],False,"
    for i in range(len(inp1)):
        if inp1[i]==0 and not lock:
            li.append(-1)
            li.append(-1)
            li.append(1)
            lock=True
            pre='S'
        elif inp1[i]==1 and not lock :
            li.append(1)
            li.append(1)
            li.append(-1)
            lock=True
            pre='Z'
        else:
            if inp1[i]==0:
                if pre=='S':
                    li.append(-1);li.append(1)
                else:
                    li.append(1);li.append(-1)
            else:
                if pre=='Z':
                    pre='S'
                    li.append(-1);li.append(1)
                else:
                    pre='Z'
                    li.append(1);li.append(-1)

    return li


def AMI(inp):
    inp1=list(inp)
    inp1.insert(0,0)
    lock=False
    for i in range(len(inp1)):
        if inp1[i]==1 and not lock:
            lock=True
            continue
        elif lock and inp1[i]==1:
            inp1[i]=-1
            lock=False
    return inp1



def plot(li):
    plt.subplot(7,1,1)
    plt.ylabel("Unipolar-NRZ")
    plt.plot(unipolar(li),color='red',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,2)
    plt.ylabel("P-NRZ-L")
    plt.plot(polar_nrz_l(li),color='blue',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,3)
    plt.ylabel("P-NRZ-I")
    plt.plot(polar_nrz_i(li),color='green',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,4)
    plt.ylabel("Polar-RZ")
    plt.plot(polar_rz(li),color='red',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,5)
    plt.ylabel("B_Man")
    plt.plot(Biphase_manchester(li),color='violet',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,6)
    plt.ylabel("Dif_Man")
    plt.plot(Differential_manchester(li),color='red',drawstyle='steps-pre',marker='>')
    plt.subplot(7,1,7)
    plt.ylabel("A-M-I")
    plt.plot(AMI(li),color='blue',drawstyle='steps-pre',marker='>')
    plt.show()


if __name__=='__main__':
    print("Enter the size of Encoded Data : ")
    size=int(input())
    li=[]
    print('Enter the binary bits sequnce of length ',size,' bits : \n')
    for i in range(size):
        li.append(int(input()))
    plot(li)
```

- **1c) Documentation: Upload your output stereo WAV file (6)**

The file **cpe367_delay.py** is an example of a main Python program that you can modify. It includes the cpe367_way.py class for WAV file I/O. It also includes a do-nothing version of a my_fifo class, which you can replace with your own working version. The main program begins with a short test section to check the operation of your fifo. Feel free to comment this out once you are confident that the fifo is working properly. The main program may be run at the command line. It does file I/O using hardcoded file names (modify as needed). A couple highlights of the main program follow. Open a stereo WAV file using

# configure the output WAV file

num_ch = 2

samples_8_16_bits = 16

sample_rate_hz = 16000

wav_out.set_wav_out_configuration(num_ch, samples _8_16_bits,sample_rate_hz)

Write two integers to the WAV file using

# output a stereo pair of samples to a WAV file

wav_out.write_wav_stereo(yout_left,yout_right) ***2) FIR Echo***

Initially let the delay, d, be 1 sample in the FIR-style echo in Eq. 1. Use a 50%/50% mix between the input signal and the echo, in other words $b_0 = b_d = 0.5$.

- **2a) What is the impulse response, h[n], for this system with d = 1? (2)**

Implement the FIR-style echo defined in Eq. 1, in Python. Use a 0.125 Second delay and a 50%/50% mix between the input signal and the echo. Generate a mono signal and use the joy.wav signal for the input. Use your FIFO class.

- **2b) How long does the echo theoretically persist? (in samples) (2)**

  Echo chambers (ECs) are enclosed epistemic circles where like-minded people communicate and reinforce pre-existing beliefs. It remains unclear if cognitive errors are necessarily required for ECs to emerge, and then how ECs are able to persist in networks with available contrary information. We show that ECs can theoretically emerge amongst error-free Bayesian agents, and that larger networks encourage rather than ameliorate EC growth. This suggests that the network structure itself contributes to echo chamber formation. While cognitive and social biases might exacerbate EC emergence, they are not necessary conditions. In line with this, we test stylized interventions to reduce EC formation, finding that system-wide truthful 'educational' broadcasts ameliorate the effect, but do not remove it entirely. Such interventions are shown to be more effective on agents newer to the network. Critically, this work serves as a formal argument for the responsibility of system architects in mitigating EC formation and retention.
- **2c) Documentation: Paste the sections of your Python code that implement the difference equation using your FIFO code (6)**

```
 import queue
q1 = queue.Queue()
#Addingitems to the queue
q1.put(11)
q1.put(5)
q1.put(4)
q1.put(21)
q1.put(3)
q1.put(10)

#using bubble sort on the queue
n =  q1.qsize()
for i in range(n):
   x = q1.get() # the element is removed
   for j in range(n-1):
      y = q1.get() # the element is removed
      if x > y :
         q1.put(y)   #the smaller one is put at the start of the queue
      else:
         q1.put(x)  # the smaller one is put at the start of the queue
         x = y     # the greater one is replaced with x and compared again with nextelement
```

```
    q1.put(x)

while (q1.empty() == False):
print(q1.queue[0], end = " ")
        q1.get()
```

3) IIR Echo

Initially let the delay, d, be 1 sample for the IIR-style echo in Eq. 2. Use a 50%/50% mix between the input signal and the echo, in other words $b_0 = a_d = 0.5$

- **3a) What are the first 4 values in the sequence of the impulse response? (2)**
  In signal processing, the impulse response, or impulse response function (IRF), of a dynamic system is its output when presented with a brief input signal, called an impulse. More generally, an impulse response is the reaction of any dynamic system in response to some external change.
  Infinite impulse response (IIR) is a property applying to many linear time-invariant systems. Common examples of linear time-invariant systems are most electronic and digital filters. Systems with this property are known as IIR systems or IIR filters, and are distinguished by having an impulse response which does not become exactly zero past a certain point, but continues indefinitely. This is in contrast to a finite impulse response (FIR) in which the impulse response h(t) does become exactly zero at times $t > T$ for some finite T, thus being of finite duration.
  Finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters
  The impulse response (that is, the output in response to a Kronecker delta input) of an Nth-order discrete-time FIR filter lasts exactly N + 1 samples (from first nonzero element through last nonzero element) before it then settles to zero.
- **3b) What is the impulse response, h[n], for this system with d = 1? (4)**
  In the context as outlined above it should be mentioned that the Laplace transformation of the impulse respose h(t) of a system gives the transfer function H(s).
  Or vice versa: The inverse Laplace transformation of H(s) is identical to the impulse response h(t).
  Hence, h(t) and H(s) are connected to each other via the Laplace transform.

Implement the IIR-style echo defined in Eq. 2, in Python. Use a 0.125 Second delay and a 50%/50% mix between the input signal and the echo. Generate a mono signal and use the joy.wav signal as the input. Use your FIFO class.

- **3c) How long does the echo theoretically persist? (in samples) (2)**
  Simulations were conducted using a broad range of parameter values, and we show results for a representative sample: three search parameter values: $\alpha = 0.05, 0.1, 1$ (recall $\alpha \in [0,1]$), and five pruning parameter values: fixed values $\beta = 0.1, 0.5, 1, 2$ for

‘confirmatory agents’, and values randomly assigned each time increment, β is ignored by ‘stochastic’ agents. In addition, two social network structures (model-R denotes random connections, which acts as a structural null hypothesis, and model-SF denotes a scale-free structure24, which is closer in structure to real social networks). For each parameter combination, 100 simulations were run (Methods).

In a hypothetical case where all agents converge from their random initial beliefs to the objective truth (μtrue = 0.5), we would observe a progressive extinction of uncertainty for each agent over time (σown→0). To test for this, at the aggregate level, we run sufficiently long simulation periods that belief distributions (frequency of μown) reach a stable state (typically < 50 iterations), i.e. all agents reach subjective certainty for their μown. While we do indeed observe agents’ beliefs tending towards the objective mean (Fig. 1), suggesting agents are updating their views in line with objective truth, some agents, crucially, do not end the simulation believing in the objective mean (that is, μown≠0.5 for some agents). Further, in situations where individual agents can access larger numbers of other beliefs (e.g. α = 1), and even with relatively weak confirmatory pruning (high ‘open-mindedness’, e.g. β = 2), they are likely to find support for their prior beliefs regardless of how extreme these may be, leading them to become fixed. Larger networks therefore increase the numbers of agents who fail to reach the objective mean. This is made possible by the formation of ECs.

- **3d) Documentation: Paste the sections of your Python code that implement the difference equation using the FIFO code (6)**

```
// Adds elements {0, 1, 2, 3, 4} to queue
for (int i = 0; i < 5; i++)
   q.push(i);

// Display contents of the queue.
cout << "Elements of queue-";

print_queue(q);

// To remove the head of queue.
// In this the oldest element '0' will be removed
int removedele = q.front();
q.pop();
cout << "removed element-" << removedele << endl;

print_queue(q);
```

```
    // To view the head of queue
    int head = q.front();
    cout << "head of queue-" << head << endl;

    // Rest all methods of collection interface,
    // Like size and contains can be used with this
    // implementation.
    int size = q.size();
    cout << "Size of queue-" << size;

    return 0;
}
```