[1]

Paper Lab Session 1: Simulation of Acoustic Echoes

Aloma Cuadrado Alvarez, Iker Pirla Casals, EETAC

***Abstract*—Throughout this session we will simulate and characterize various acoustic effects using MATLAB and some basic digital signal processing tools. In short, we will simulate the acoustic reverberation in a room.**

I. INTRODUCTION

In order to carry out the laboratory session we will follow the schema represented below in *Figure 1*.

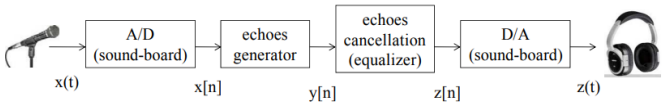


Figure 1.- General scheme and block diagram

As can be seen, this process consists of different points that will be analyzed progressively as the proposed questions are answered.
The process starts by converting a signal from analogue to digital, and then simulates the echo effect (the effect of the acoustic reverberation in a room). Later we will reverse this process to eliminate the echo effect, and finally we will convert the digital signal back to an analogue signal.

II. BACKGROUND STUDY

In order to be able to implement these blocks, it is first necessary to know how certain functions work. To do this, with the help of MATLAB we will study the most suitable ones for this exercise. These include the 'sound' and 'soundsc' functions as well as the 'audioplayer' and 'audiorecorder' functions.
The first of these functions ("sound") is responsible for collecting the sound signal and converting it into a matrix that will be sent to the loudspeaker with a sampling frequency of 44100 Hz.

In relation to the second function ("soundsc") scales the values of audio signal y to fit in the range from –1 to 1, and then sends the data to the speaker at the default sample rate of 44100 Hz. By first scaling the data, "soundsc" plays the audio as loudly as possible without clipping. The mean of the dynamic range of the data is set to zero. These two functions share the same purpose which is basically to play audio.
Regarding the remaining two functions, one is responsible for recording audio data from an input device such as a microphone for processing in MATLAB ('audiorecorder'), while the other is responsible for playing audio data ('audioplayer'). The audioplayer object contains properties that enable additional flexibility during playback. For example, you can pause, resume, or define callbacks using the audioplayer object functions.

Echoes generation
Firstly, we will perform a simple reverberation, using a signal whose configuration is a simple echo. The input to the function shall be a vector of signal samples (x) at frequency (fs).
The signal output will be created using the following equation:

$$y[n] = x[n] + \alpha \cdot x[n - N] \qquad (1)$$

α = Amplitude of the single echo
N= Delay of the echo in samples

This signal delay is due to the direct relationship between sampling frequency and seconds.

To visualize what we are going to develop, we have made a programming diagram of this simple echo (figure 2). For this we need to know this relationship:

$$N = \Delta(delay) \cdot fs$$

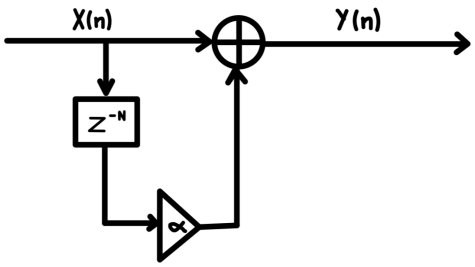


Figure 2.- Programming diagram simple echo

Analyzing the programming diagram we can see that we are dealing with a FIR system as well as a stable system since the poles are less than 1. The corresponding impulse response of the system is :

$$h[n] = \delta[n] + \alpha \cdot \delta[n - N] \quad H(z) = 1 + \alpha \cdot z^{-N} \quad (2)$$

We will continue to analyze complex reverberation.This configuration corresponds to a more complex reverberation

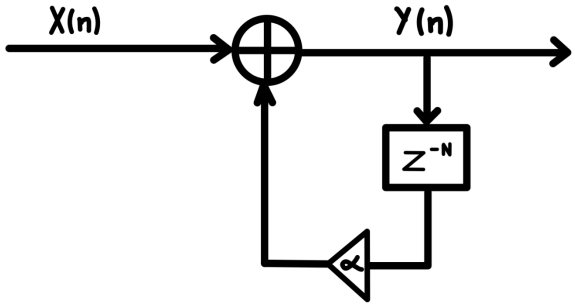
[1]

producing multiple echoes. In this case, the signal with the echoes is created using the same equation as in the simple reverberation:

$$y[n] = x[n] + \alpha \cdot y[n - N] \quad (3)$$

α = Amplitude of the single echo
N= Delay of the echo in samples

This system will present the following configuration:



Next we will extract the impulse response of the system. First we will perform the Z-transformation and from there we will discuss what kind of system we are dealing with.

$$Y[z] = X[z] + \alpha \cdot Y[z] * z^{-N}$$

$$Y[z] * (1 - \alpha \cdot z) = X[z]$$

$$H[z] = \frac{1}{(1-\alpha \cdot z^{-N})} \quad (4)$$

$$h[n] = \sum_{k=0}^{\infty} \alpha^{k} \cdot \delta[n - kN] \quad (5)$$

It can be seen at a glance that for the system to be stable the value of $\alpha < 1$ to stay within the circumference of radius 1. Therefore with this condition we can conclude that it is an IIR system.

Echoes cancellation (equalizer)
In this process we simply need to reverse the previous process.

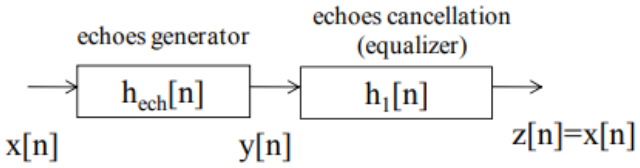


Figure 4

To reverse the process we simply need the multiplication of the two to be unitary, therefore:

$$H_{ech}(Z) \cdot H_1(Z) = 1 \quad (6)$$

So the condition will be satisfied when:

$$z[n] = y[n] - \alpha \cdot z[n - N] \quad (7)$$

$$H_1(Z) = \frac{1}{1 + \alpha \cdot z^{-N}} \quad (7.1)$$

$$h_1[n] = \sum_{k=0}^{\infty} (-\alpha)^{k} \cdot \delta[n - kN] \quad (7.2)$$

We check that with the (2) and (4) equations we get the result in (6).

$$(1 + \alpha \cdot z^{-N}) \cdot \frac{1}{1 + \alpha \cdot z^{-N}} = 1$$

Once we know how the simple echo equalizer works we have to make a complex one with multiple echoes, that way we will get a fully equalized sound. This is the equation that the equalizer satisfies:

$$z[n] = y[n] - \alpha \cdot y[n - N] \quad (8)$$

Its corresponding impulsional response for h1[n] is:

$$Heq(Z) = 1 - \alpha \cdot Z^{-N}$$

$$heq[n] = \delta[n] - \alpha \cdot \delta[n - N] \quad (9)$$

Finally we check that with the impulsional response (7) and (9) satisfy the condition (6)

$$h[n] * heq[n] = \delta[n]$$

$$heq[n] * h_1[n] = \delta[n] - \alpha \cdot \delta[n - N] * \sum_{k=0}^{\infty} (-\alpha)^{k} \cdot \delta[n - kN]$$
$$= \delta[n]$$

III. LABORATORY SESSION

Once we know the different aspects, characteristics and details of the study we are going to carry out, we are ready to start the MATLAB program and check everything we have studied previously.

First of all, we have to create a new project in MATLAB, as well as establish the most important variables that we will use throughout the study. These are:

fs	alpha	delay	t
44100 Hz	0,8	1 (s)	5 (s)

Once we have everything ready to start, we will create the first function that will be in charge of converting the signal from the analog domain to the digital domain using the microphone of a headset connected to the PC.
The code of the function is the following one:

```
function [x] = Analog_to_digital(t,fs)

    obj=audiorecorder(fs,16,1);
    recordblocking(obj,t);
    x= getaudiodata(obj);
end
```

If we use the function (plot(x)) we can visually see the signal generated by the recorded sound.

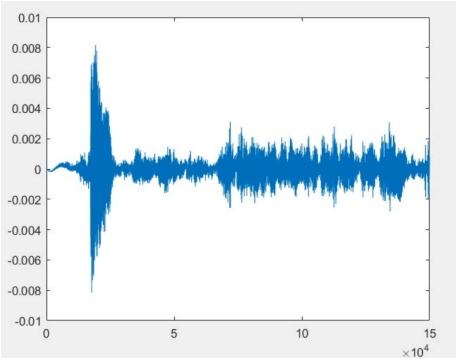


Figure 5- Sound used

On the other hand we have to create a function that will do the opposite of the previously described function, we will switch from digital to analogue. Its code will be:

```
function [x] = Digital_to_Analog(vec,fs)
sound(vec,fs);
end
```

The next step will be to create a signal together with a single echo, for this we will have to adapt the function (1) to MATLAB, taking into account several assumptions, such as the values in which the signal is greater than N. The function is as follows:

```
function [y] = simple_echo_generation(x1,fs,alpha,delay)

    N= delay*fs;
    Longitud = length(x1);
    y=zeros(Longitud,1);
    for n=1:Longitud
        if n>N
            y(n)=x1(n) + alpha*x1(n-N);
        else
            y(n)=x1(n);
        end

    end


end
```

If we plot the function we can see this echo.

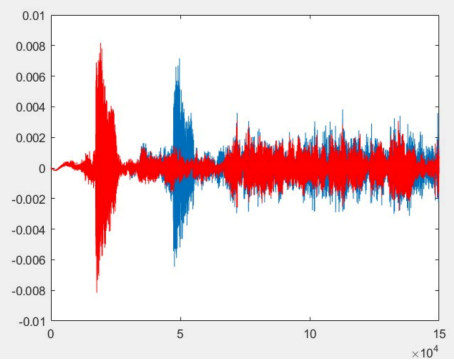


Figure 6-Sigle echo (in blue the echo)

Then we also run the sound for all the functions that we created, so we can listen to it.

Next we will create the multiple echo function which aims to generate a signal with multiple echoes. To do this we will have to adapt the function (3).

The code we will use will be:

```
function [y] = multiple_echo_generation(x1,fs,alpha,delay)


N= delay*fs;
Longitud = length(x1);
y=zeros(Longitud,1);
for n=1:Longitud
    if n>N
        y(n)=x1(n) + alpha*y(n-N);
    else
        y(n)=x1(n);
    end

end


end
```

If we plot the function we can see the original signal in yellow and the new echoes created in green.

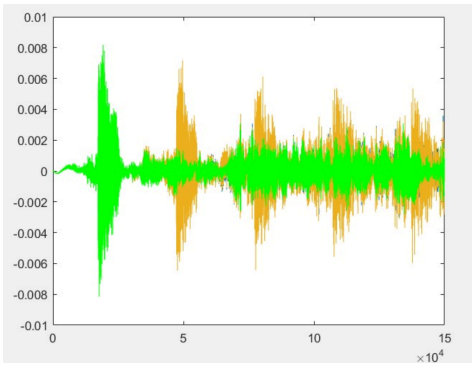


Figure 7- Multiple echo

In this graph we can see that each echo that appears loses amplitude compared to the original signal.

Once we have created both a single and a multiple echo it is time to re-equalise to get the first sound generated. To start with we will equalize the single echo signal (figure 6). To do

this we will have to adapt the function (7) to MATLAB. The function will be the following one:

```
function [y] = multiple_echo_generation(x1,fs,alpha,delay)

    N= delay*fs;
    Longitud = length(x1);
    y=zeros(Longitud,1);
    for n=1:Longitud
        if n>N
            y(n)=x1(n) + alpha*y(n-N);
        else
            y(n)=x1(n);
        end

    end


end
```

If we plot the function we can see the original signal

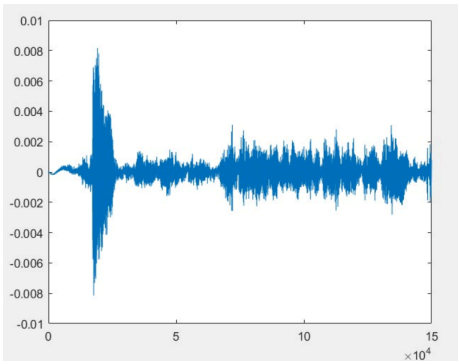


Figure 8- Signal after equalizing the single echo

And finally we will perform the same process for the multiple echo case. For this we will have to adapt the function (8). The code we will use will be the following:

```
function [z] =multiple_echo_equalization(y,fs,alpha,delay)
   N= delay*fs;
    Longitud = length(y);
    z=zeros(Longitud,1);
    for n=1:Longitud
        if n>N
            z(n)=y(n) - alpha*y(n-N);
        else
            z(n)=y(n);
        end

    end

end
```

If we plot the function we can see the original signal

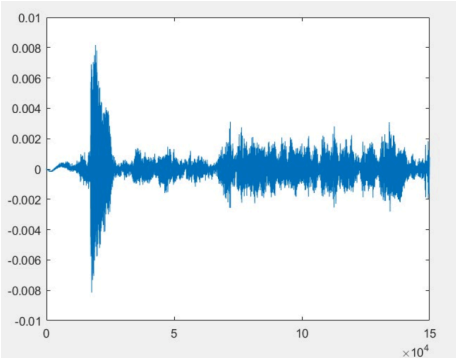


Figure 9-Signal after equalizing the multiple echo

As we can see our code is valid for multiple echo equalization so our lab session task is complete.

IV. CONCLUSIONS

Once we have done the practice we are able to understand better everything we have done in the theoretical classes that are related to this task. It has been quite useful to settle the acquired knowledge. To be able to modify an audio signal, creating new echoes has seemed to us a very pleasant and didactic practical example of the theory.

V. REFERENCES

[1] MATLAB

[2] Preparation of Papers for IEEE TRANSACTIONS and JOURNALS (March 2004)

[3] Introduction to MATLAB

[4] Simulation of Acoustic Echoes