Tecnológico de Monterrey **Campus Ciudad de México** **School of Engineering and Sciences** **Department of Mechatronics**	**TE2019 Digital Signal Processing Laboratory** Student's 1 name and ID: Alfredo Zhu Chen - A01651980 Student's 2 name and ID: Luis Arturo Dan Fong - A016506720 Student's 3 name and ID: Miguel Alejandro Cruz Banda - A01656527 Lab # 4 The convolution sum Date: 12/03/2020

1 INTRODUCTION

Convolution is a mathematical operation that combines two signals to produce a third signal. In the field of digital signals and its processing it is very important, since it allows obtaining the output signal of a system from the input signal and the impulse response. In other words, we can predict the output, knowing the input and the impulse response. This is shown graphically in the figure 1.1[1].

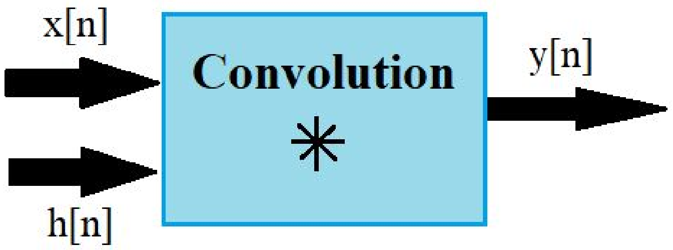


Figure 1.1: Graphical representation of the convolution sum.

Convolution and related operations are found in many engineering and mathematical applications[2].

- In statistics, is a weighted moving average.
- In probability theory, the probability distribution of the sum of two independent random variables is the convolution of each of their probability distributions.
- In optics, many types of "spots" are described with convolutions. A shadow is the convolution of the shape of the light source that creates the shadow and the object whose shadow is being cast. An out of focus photo is the convolution of the correct image with the blurred circle formed by the iris diaphragm.
- In acoustics, an echo is the convolution of the original sound with a function that represents the varied objects that reflect it.
- In electrical engineering and electronics, the output of a linear system (stationary or time-invariant) is the convolution of the input with the response of the system to an impulse.
- In physics, wherever there is a linear system with a "superposition principle", a convolution operation appears.

So many other applications exist for this operation, but the ones that will be studied in this laboratory section is in sound and image.

OBJECTIVES

1. To implement the convolution sum algorithm in MATLAB
2. To implement basic audio and image filters based on the convolution sum
3. To identify the usefulness of the convolution sum

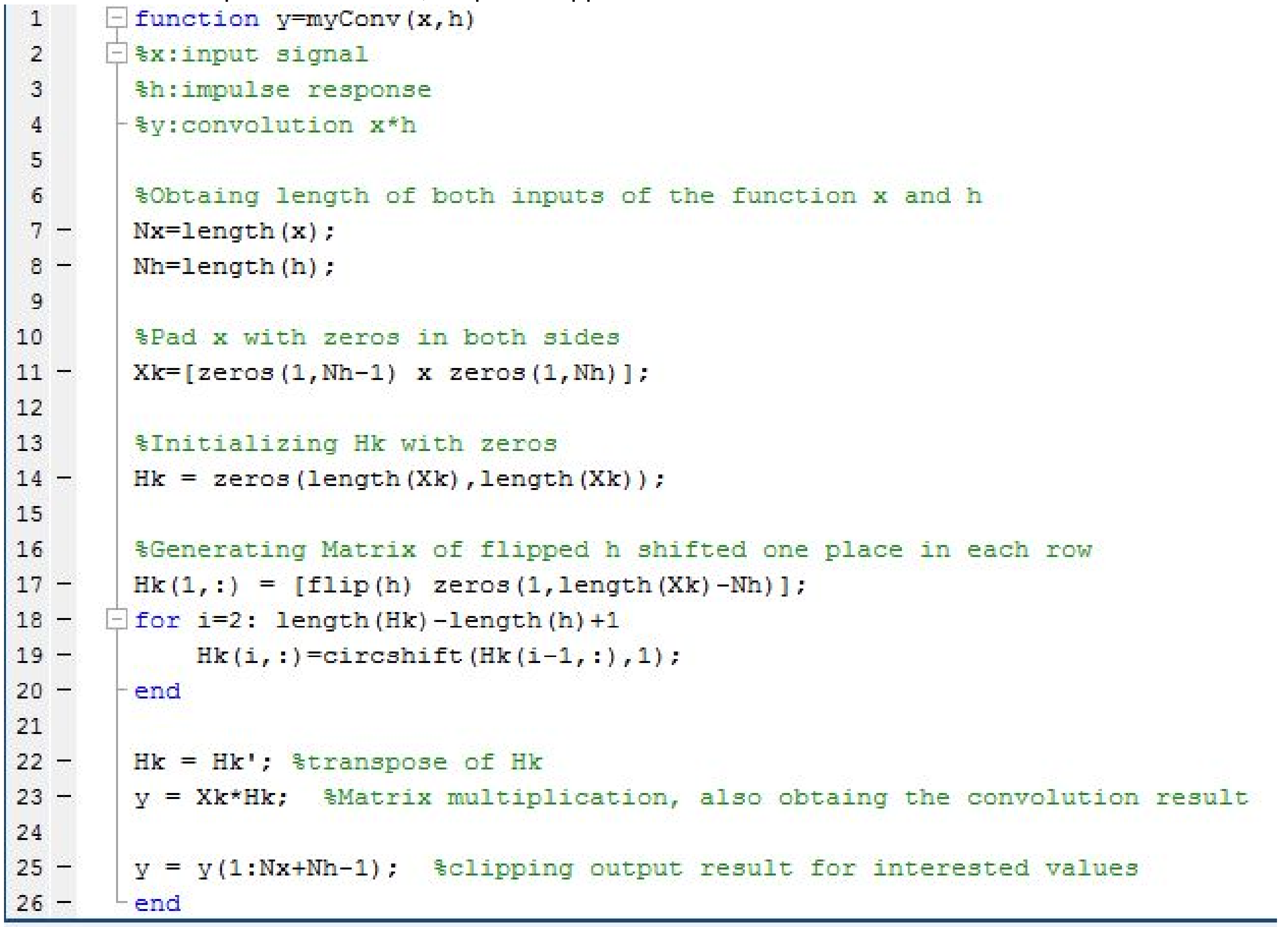
2 MATERIALS & METHODS

The materials used in this laboratory were:

- PC or laptop with at least 4GB RAM and and 10MB of space
- MATLAB R2019a or newer versions plus the Audio with the Signal Processing Toolboxes
- Personal headphones
- Test images

2.1 Coding the convolution sum algorithm from scratch

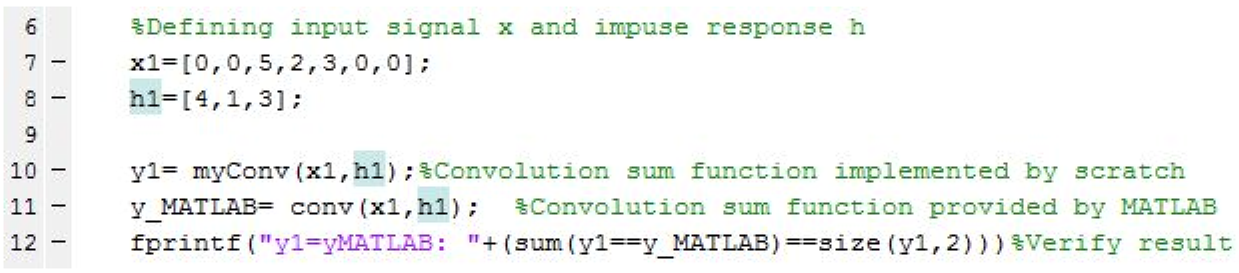
Considering the given arguments *x* and *h*, a matlab file defined and named as *myConv(x,h)* was created for it to be used for this section of the laboratory practice, with input signal *x* and impulse response *h* as arguments. the function *length(x)* was used to determine the number of values of a vector. A new *Xk* vector and a *Hk* matrix were defined as in line 11 and 14 respectively(figure 2.1.1). Then, Hk was filled and shifted one place with *h* in each row posible. Finally the variable "*y"* was defined as the product of the multiplication of *Xk* and the transpose value of *Hk,* output is clipped to maintain the needed values.



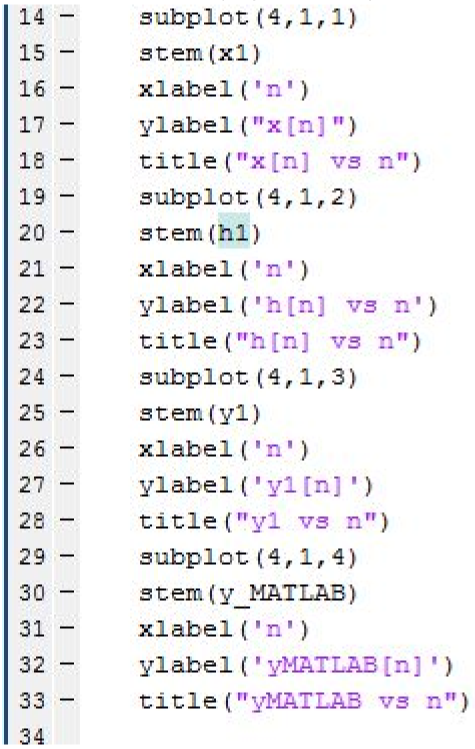
```
1   function y=myConv(x,h)
2   %x:input signal
3   %h:impulse response
4   %y:convolution x*h
5
6   %Obtaing length of both inputs of the function x and h
7   Nx=length(x);
8   Nh=length(h);
9
10  %Pad x with zeros in both sides
11  Xk=[zeros(1,Nh-1) x zeros(1,Nh)];
12
13  %Initializing Hk with zeros
14  Hk = zeros(length(Xk),length(Xk));
15
16  %Generating Matrix of flipped h shifted one place in each row
17  Hk(1,:) = [flip(h) zeros(1,length(Xk)-Nh)];
18  for i=2: length(Hk)-length(h)+1
19      Hk(i,:)=circshift(Hk(i-1,:),1);
20  end
21
22  Hk = Hk'; %transpose of Hk
23  y = Xk*Hk;  %Matrix multiplication, also obtaing the convolution result
24
25  y = y(1:Nx+Nh-1);  %clipping output result for interested values
26  end
```

Figure 2.1.1 Creating the *myConv(x,h)* function

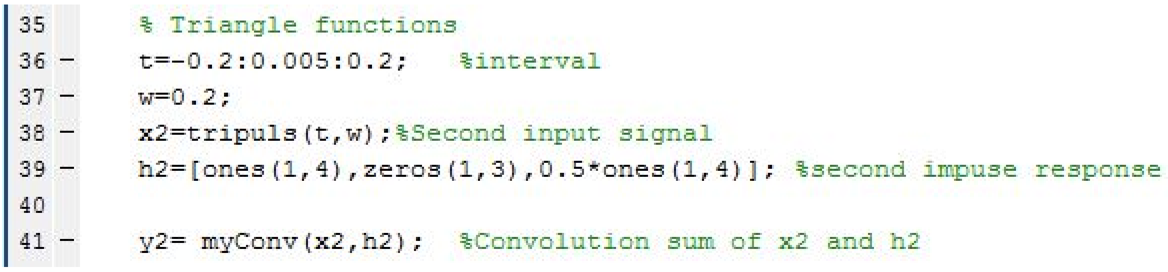
In figure 2.1.2, considering the given values of *x1* and *h1,* a comparison was required between the values obtained from our implemented *myConv(x,h)* function (Figure 2.1.1) and the *conv(u,v)* function from Matlab. By using *fprintf(A)* the result of the comparison between the sum of correct values and size of *y1* (from *myConv(x,h)* was printed in order to know if our implementation was correct.


```
6       %Defining input signal x and impuse response h
7       x1=[0,0,5,2,3,0,0];
8       h1=[4,1,3];
9
10      y1= myConv(x1,h1);%Convolution sum function implemented by scratch
11      y_MATLAB= conv(x1,h1);  %Convolution sum function provided by MATLAB
12      fprintf("y1=yMATLAB: "+(sum(y1==y_MATLAB)==size(y1,2)))%Verify result
```
Figure 2.1.2 Comparison between *myConv* and *Conv*

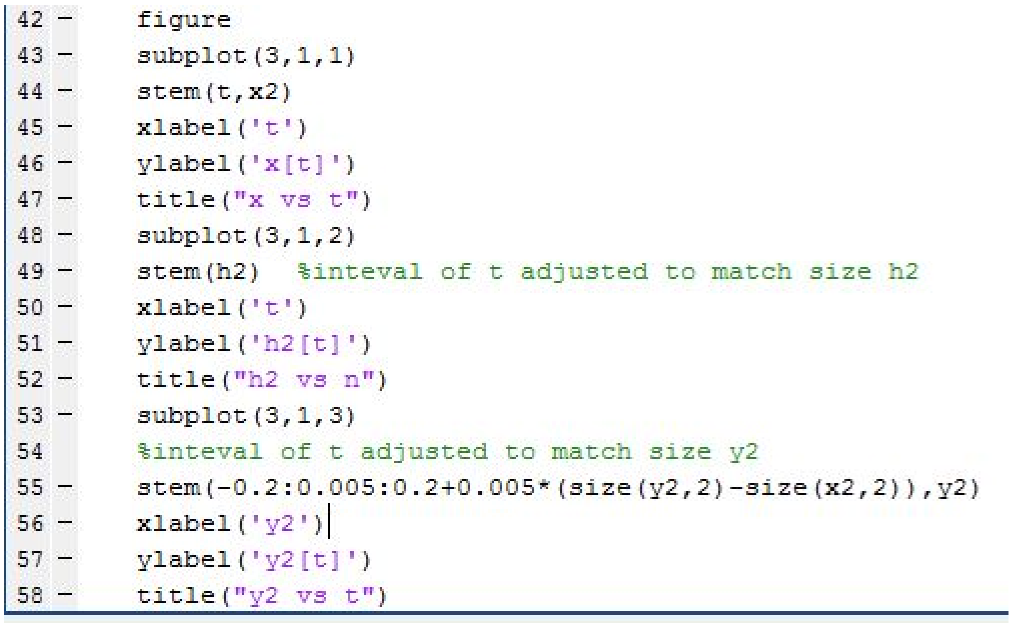
All sequences(*x,h, y1 and y_Matlab*) were plotted using the function *plot(X,Y)* and stacked with function *subplot(m,n,p)* for a better graphical representation(Figure 2.1.3).


```
14      subplot(4,1,1)
15      stem(x1)
16      xlabel('n')
17      ylabel("x[n]")
18      title("x[n] vs n")
19      subplot(4,1,2)
20      stem(h1)
21      xlabel('n')
22      ylabel('h[n] vs n')
23      title("h[n] vs n")
24      subplot(4,1,3)
25      stem(y1)
26      xlabel('n')
27      ylabel('y1[n]')
28      title("y1 vs n")
29      subplot(4,1,4)
30      stem(y_MATLAB)
31      xlabel('n')
32      ylabel('yMATLAB[n]')
33      title("yMATLAB vs n")
34
```
Figure 2.1.3 Plotting all sequences to verify *myConv* function

Another input signal *x2,* a sampled aperiodic triangle, was created with function *tripuls(t,w)* in the time interval *t* with the width of *w*(figure 2.1.4). A second impulse response signal was defined in line 39. After knowing that our convolution sum implementation was correct, it is used to perform convolution sum of *x2* with *h2.*


```
35      % Triangle functions
36      t=-0.2:0.005:0.2;   %interval
37      w=0.2;
38      x2=tripuls(t,w);%Second input signal
39      h2=[ones(1,4),zeros(1,3),0.5*ones(1,4)]; %second impuse response
40
41      y2= myConv(x2,h2);  %Convolution sum of x2 and h2
```
Figure 2.1.4.Second convolution sum

In the next figure 2.1.5, all sequences were plotted as before, but there were some changes which needed to be done to match the size. The size of the output signal *y2* was also not matching the size of *t*, so the same interval was extended with the same step size to finally match the size.

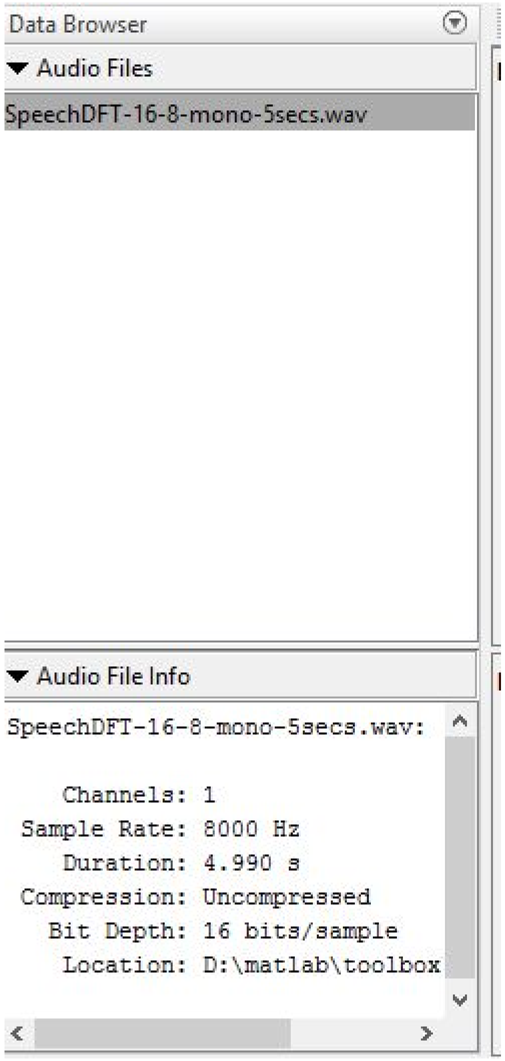


```
figure
subplot(3,1,1)
stem(t,x2)
xlabel('t')
ylabel('x[t]')
title("x vs t")
subplot(3,1,2)
stem(h2)  %inteval of t adjusted to match size h2
xlabel('t')
ylabel('h2[t]')
title("h2 vs n")
subplot(3,1,3)
%inteval of t adjusted to match size y2
stem(-0.2:0.005:0.2+0.005*(size(y2,2)-size(x2,2)),y2)
xlabel('y2')
ylabel('y2[t]')
title("y2 vs t")
```

Figure 2.1.5. Plotting all sequences of second convolution.

2.2 Convolution and sound

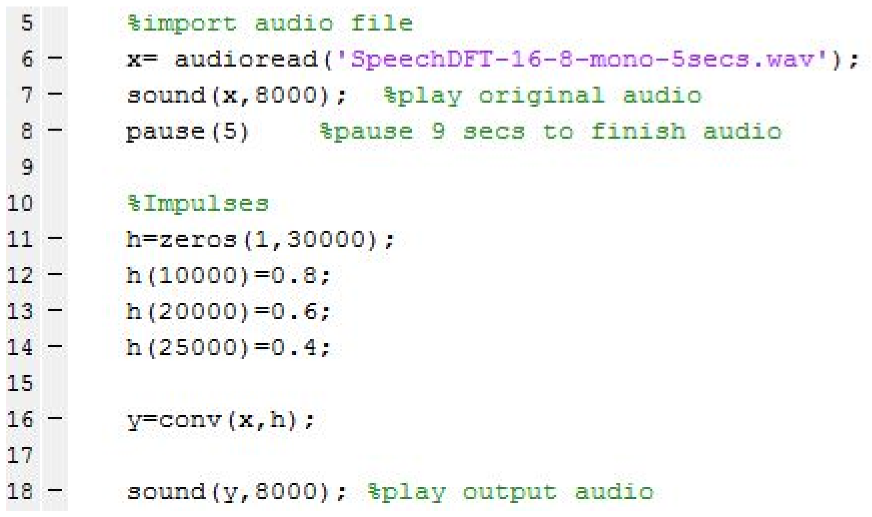
In this experiment, the audio file from matlab named "SpeechDFT-16-8-mono-5secs.wav" was used. The sample rate and other related information of the audio can be found in figure 2.1.1, which was obtained by opening the audio file from the MATLAB installed folder "matlab/toolbox/audio/samples" in MATLAB's APP "Audio Labeler".



```
SpeechDFT-16-8-mono-5secs.wav:

    Channels: 1
 Sample Rate: 8000 Hz
    Duration: 4.990 s
 Compression: Uncompressed
   Bit Depth: 16 bits/sample
    Location: D:\matlab\toolbox
```

Figure 2.2.1.Obtaining Sample Rate and Duration of audio file.

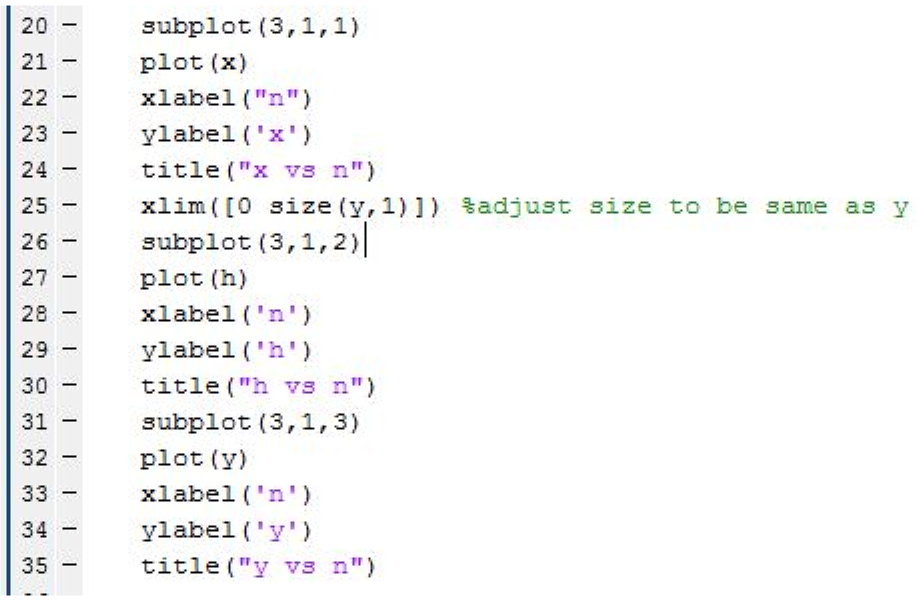
As it can be seen in figure 2.2.2, the audio file was imported using function *audioread(filename)* and played using function *sound(y,Fs)* with the sample rate as one of the arguments. A pause of 5 seconds with MATLAB function *pause(s)* was performed in order to seperate it with the sound after convolution sum operation. A vector of zeros *h* was created with a length of 30000. Different values of amplitude were assigned to the indexes 10000,20000 and 25000 as impulse responses. The convolution sum was performed with MATLAB function *conv(u,v)* and the effect of the sound can be appreciated by using *sound(y,Fs)* again.



```
%import audio file
x= audioread('SpeechDFT-16-8-mono-5secs.wav');
sound(x,8000);  %play original audio
pause(5)    %pause 9 secs to finish audio

%Impulses
h=zeros(1,30000);
h(10000)=0.8;
h(20000)=0.6;
h(25000)=0.4;

y=conv(x,h);

sound(y,8000); %play output audio
```

Figure 2.2.2 Convolution operation of audio signal

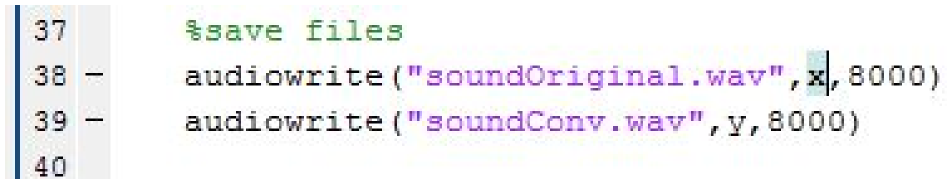
The input audio *x*, impulse response *h* and output audio *y* were plotted as shown in the next figure 2.2.3. The horizontal axis of *x* is adjusted to be the same as *y*, this is important for later analysis purposes in order to know the effects of convolution sum has in the audio.



```
subplot(3,1,1)
plot(x)
xlabel("n")
ylabel('x')
title("x vs n")
xlim([0 size(y,1)]) %adjust size to be same as y
subplot(3,1,2)
plot(h)
xlabel('n')
ylabel('h')
title("h vs n")
subplot(3,1,3)
plot(y)
xlabel('n')
ylabel('y')
title("y vs n")
```

Figure 2.2.3 Plotting input audio *x*, impulse response *h* and output audio *y*.

The next figure 2.2.4 shows how to save the audio as a *wav* format in the same folder of this script. The sample rate and the name of the files were specified in the arguments of function *audiowrite(filename, y,Fs).*

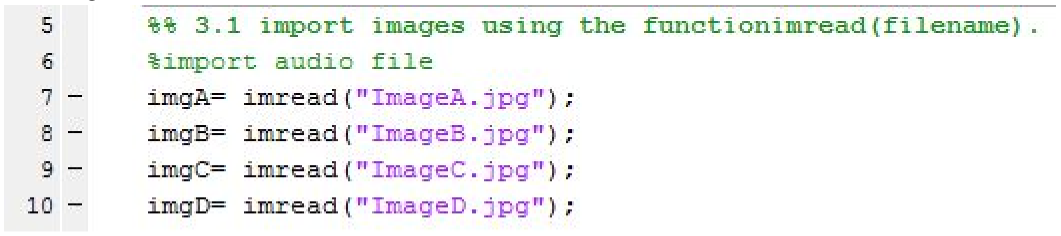


```
%save files
audiowrite("soundOriginal.wav",x,8000)
audiowrite("soundConv.wav",y,8000)
```

Figure 2.2.4 Saving input audio *x* and output audio *y*

2.3 Convolution and image

From figure 2.3.1, to perform this part of the lab session, it is needed to import the images “A” and “B”, given by the instructor, using the MATLAB function *imread(filename).* It is important to know that the images should be in the same folder as the working directory, otherwise the directory has to be specified with the name of the image.



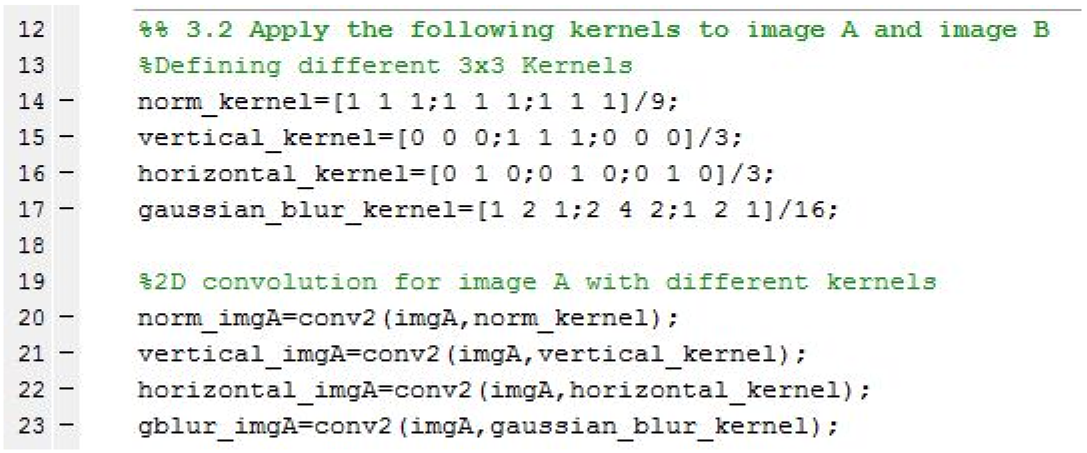
```
5      %% 3.1 import images using the functionimread(filename).
6      %import audio file
7  -   imgA= imread("ImageA.jpg");
8  -   imgB= imread("ImageB.jpg");
9  -   imgC= imread("ImageC.jpg");
10 -   imgD= imread("ImageD.jpg");
```

Figure 2.3.1 Importing all images

The previous images are imported in order to apply 2D convolution with four kernels specified in the instructions with the 2D convolution function *conv2(img, kernel)*. The kernels are shown in the figure 2.3.2. The creation of the kernels could be done by writing the lines of code from 14 to 17 and the convolutions are performed with the lines from 20 to 24 in figure 2.3.3. Similarly, the same kernels are applied to image “B”(see appendix for more details).

$$\text{a)}\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \text{c)}\ \frac{1}{3}\begin{bmatrix}0 & 1 & 0\\ 0 & 1 & 0\\ 0 & 1 & 0\end{bmatrix} \qquad \text{b)}\ \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \qquad \text{d)}\ \frac{1}{3}\begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 1\\ 0 & 0 & 0\end{bmatrix}$$

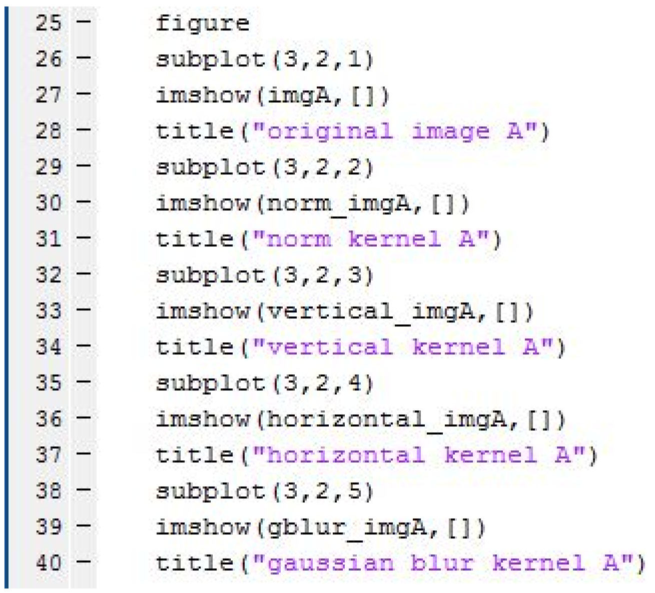
Figure 2.3.2 Kernels for images A and B.



```
12     %% 3.2 Apply the following kernels to image A and image B
13     %Defining different 3x3 Kernels
14 -   norm_kernel=[1 1 1;1 1 1;1 1 1]/9;
15 -   vertical_kernel=[0 0 0;1 1 1;0 0 0]/3;
16 -   horizontal_kernel=[0 1 0;0 1 0;0 1 0]/3;
17 -   gaussian_blur_kernel=[1 2 1;2 4 2;1 2 1]/16;
18
19     %2D convolution for image A with different kernels
20 -   norm_imgA=conv2(imgA,norm_kernel);
21 -   vertical_imgA=conv2(imgA,vertical_kernel);
22 -   horizontal_imgA=conv2(imgA,horizontal_kernel);
23 -   gblur_imgA=conv2(imgA,gaussian_blur_kernel);
```

Figure 2.3.3 Defining kernels and applying 2D convolution

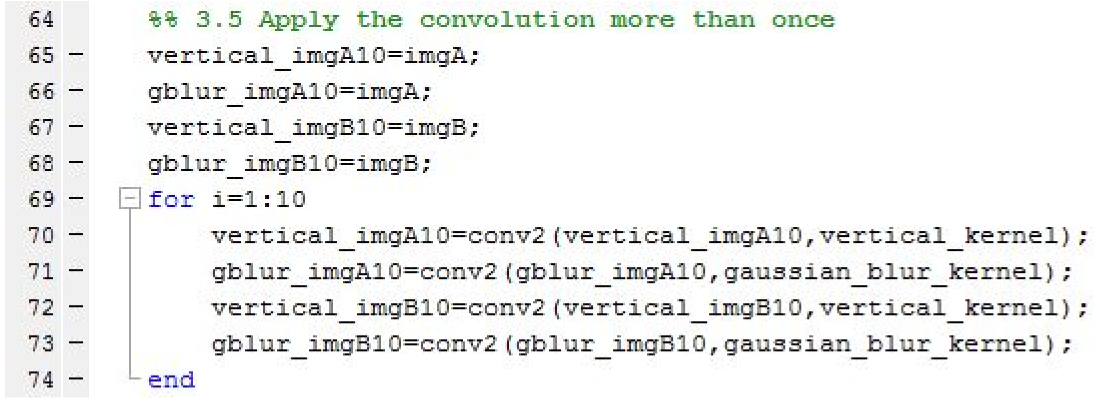
After the convolution operations, the resulting images were shown using the MATLAB function *imshow(img,[low high]*) as it can be seen in the next figure 2.3.4. The second parameter was specified as “[]” when calling the function, since it is needed to indicate that the minimum value is black and the maximum value is white for a grayscale image, otherwise it will take the default display range which could affect the result.



```
figure
subplot(3,2,1)
imshow(imgA,[])
title("original image A")
subplot(3,2,2)
imshow(norm_imgA,[])
title("norm kernel A")
subplot(3,2,3)
imshow(vertical_imgA,[])
title("vertical kernel A")
subplot(3,2,4)
imshow(horizontal_imgA,[])
title("horizontal kernel A")
subplot(3,2,5)
imshow(gblur_imgA,[])
title("gaussian blur kernel A")
```

Figure 2.3.4 Plotting all resulting images after convolution

Once the previous process is done, it is needed to select two kernels from the previous step and apply the 2D convolution function more than once for each kernel to the original image(figure 2.3.5). In this case, a *for* loop is used to iterate to achieve 10 times 2D convolution. The same procedure is taken for image “B” and both results will be shown later in section 3.3.



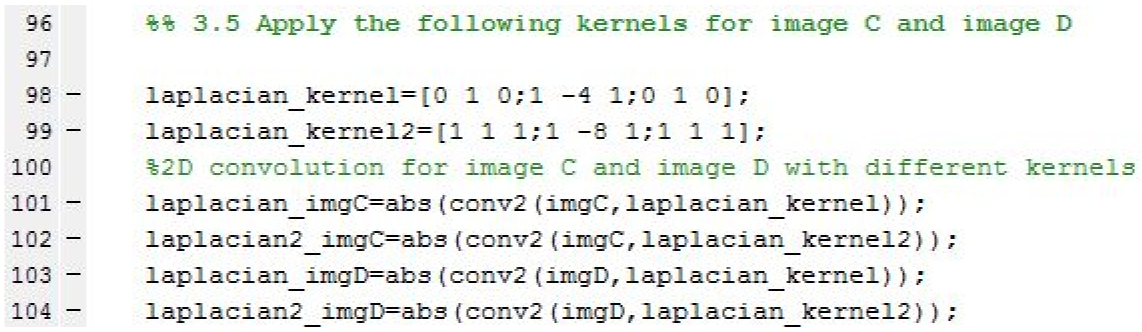
```
%% 3.5 Apply the convolution more than once
vertical_imgA10=imgA;
gblur_imgA10=imgA;
vertical_imgB10=imgB;
gblur_imgB10=imgB;
for i=1:10
    vertical_imgA10=conv2(vertical_imgA10,vertical_kernel);
    gblur_imgA10=conv2(gblur_imgA10,gaussian_blur_kernel);
    vertical_imgB10=conv2(vertical_imgB10,vertical_kernel);
    gblur_imgB10=conv2(gblur_imgB10,gaussian_blur_kernel);
end
```

Figure 2.3.5 Applying 2D convolution more than once

The images “C” and “D” are imported with the same function *inmread(filename)* as before in figure 2.3.1. These were applied with other kernels specified in the lab briefing and shown in the figure 2.3.6 with the 2D convolution function. In figure 2.3.7, the kernels are defined and applied to the images. Because of the possible negative values of the convolution, the magnitudes of the resulting images were obtained by MATLAB function *abs(x)*, this adjusts the image to be in the correct range of values.
The results can be found later in section 3.3, it has to be analyzed what impact the kernels had on the images, which kernel had the best performance applied to the images in the same section, explain what is the effect of applying the same kernel more than once and the difference between the first kernels and the second ones.

$$e)\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad f)\ \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 2.3.6 Kernels for images C and D.



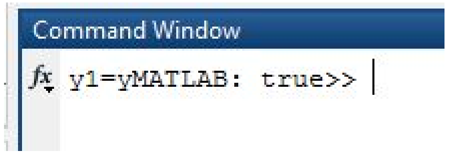
```
%% 3.5 Apply the following kernels for image C and image D

laplacian_kernel=[0 1 0;1 -4 1;0 1 0];
laplacian_kernel2=[1 1 1;1 -8 1;1 1 1];
%2D convolution for image C and image D with different kernels
laplacian_imgC=abs(conv2(imgC,laplacian_kernel));
laplacian2_imgC=abs(conv2(imgC,laplacian_kernel2));
laplacian_imgD=abs(conv2(imgD,laplacian_kernel));
laplacian2_imgD=abs(conv2(imgD,laplacian_kernel2));
```

Figure 2.3.7 Applying 2D convolution for image “C” and “D“

3 RESULTS AND DISCUSSION

3.1 Coding the convolution sum algorithm from scratch

After comparing the implemented function *myConv(x,h)* from scratch and the MATLAB function *conv(u,v)*, it can be seen from the figure 3.3.1 that the two functions obtained the same result with the given signals and that the implementation is verified. The same interpretation can be taken from the figure 3.1.2, which shows the input signal *x,* the impulse response *h* and both outputs *y* obtained from the two methods. The two plots for the output also verifies that the implementation *myConv(x,h)* is successful.



```
y1=yMATLAB: true>>
```

Figure 3.1.1 Verifying result of *myConv(x,h)* implementation in Command Window

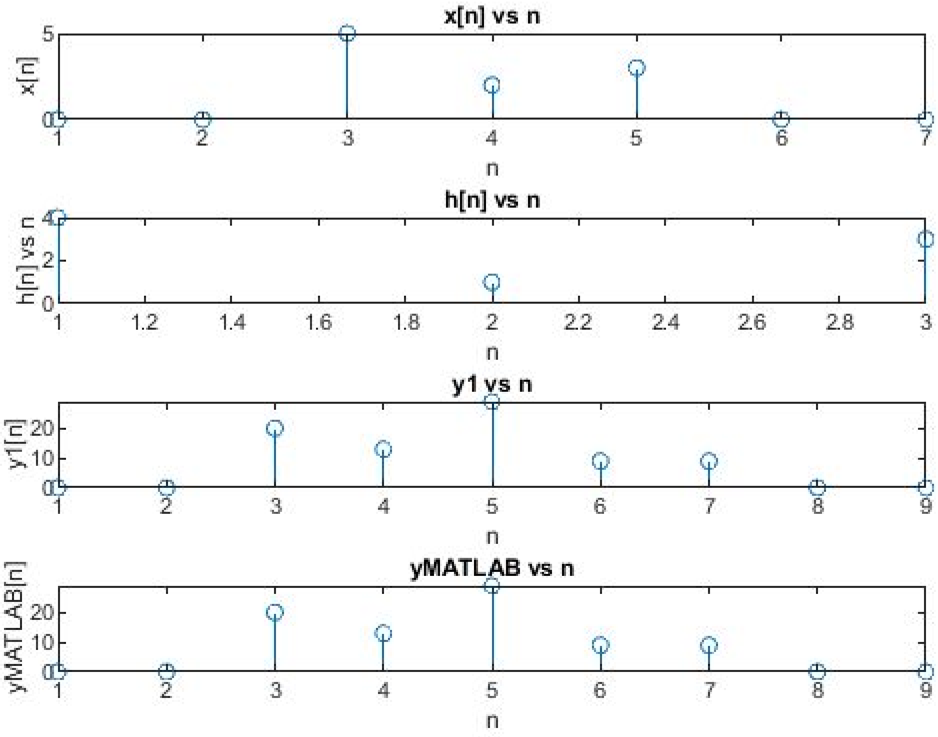


Figure 3.1.2 Verifying *myConv(x,h)* implementation graphically

From the next figure 3.1.3, it can be appreciated the convolution of the input triangular function and the impulse response. It can be observed that the output signal is smoother than the input signal and it also achieves higher amplitude values. The impulse function has values of 1 at the beginning, 0 at the middle and 0.5 at the end. By having this difference, the output signal is not symmetric as the input signal and it seems slightly weighted on the right side.

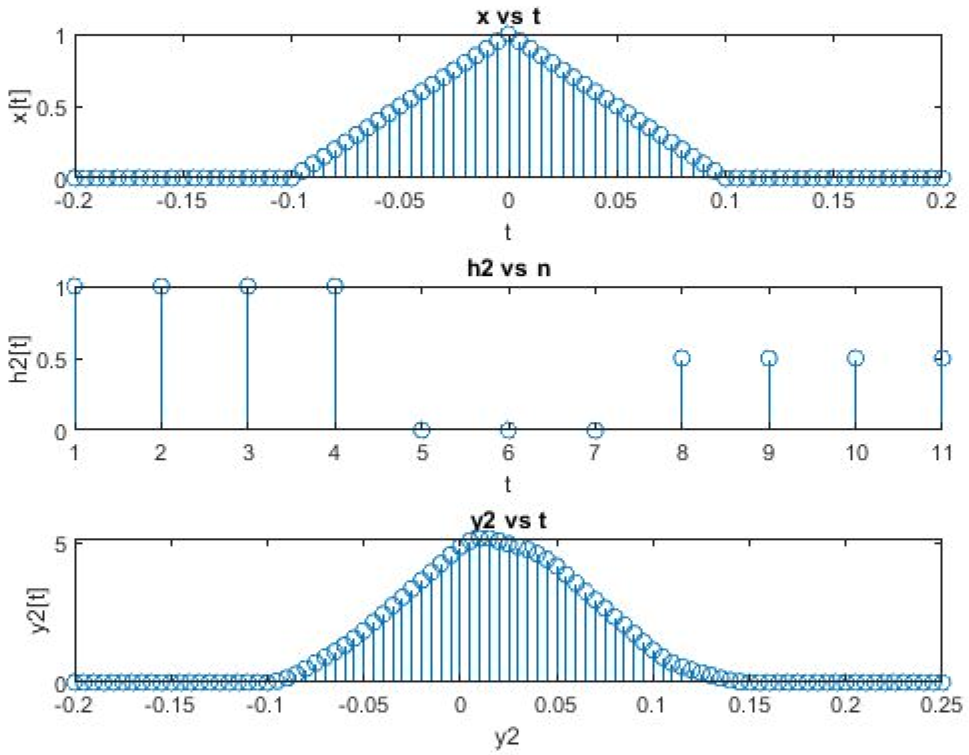


Figure 3.1.3 Applying 2D convolution for image “C” and “D“

3.2 Convolution and sound

The next figure 3.2.1 shows the input audio *x*, impulse response *h* and the result of the convolution *y*. After hearing and comparing the input and output audio, it can be appreciated that an effect of echo was achieved by doing convolution. Each impulse different from 0 in *h* was performing a reproduction of the original audio signal in a different defined moment, so the structure *h* defines at which moment and at what amplitude is producing the echo. From plots of *x* and *y*, the output audio has larger duration than the input signal and starts a few moments later, this is because of the convolution sum and the echo effect remaining in the audio, input signal *x* ends with almost 40000 samples while the output ends with almost 70000 samples.

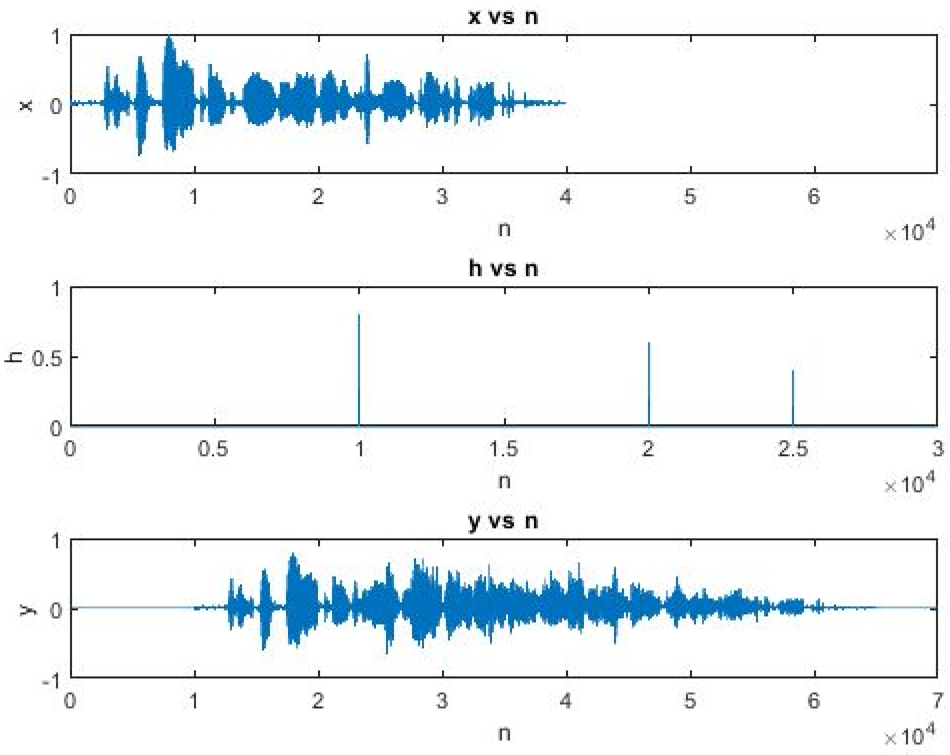


Figure 3.2.1 Convolution for audio signal

3.3 Convolution and image

As it is shown in the figures 3.3.1 and 3.3.2, the original image is compared with the resulting 2D convolutions of itself. The original image has "salt and pepper" noise on it, so it is important to see how different kernels alter this kind of noise. The effects of the 2D convolution from different kernels are resumed in the next table.

Table 1. Observations of the images applying convolution.

Kernel	**Observations on the image "A" and image "B" after 2D convolution(figure 3.3.1 and 3.3.2)**
Kernel "a": Box Blur	The image is blurred distributedly since the kernel is taking the mean value of the a center pixel with the 8 surrounded pixels. The "salt and pepper" on image "A" are slightly removed because they are averaged with their closest pixels values. This happens simillary on image "B", the original image was very noisy, but after the 2D convolution it became smoother.
Kernel "b": Gaussian Blur	By comparing with the Box Blur kernel, the Gaussian Blur kernel makes similar effects for an image, it blurs the image. The main difference is that the center takes the largest value and the smallest values toward the edges and corners. So

	in image “A”, it can be seen that the image is smoother and blurrer, the details were removed and the noisy points are not that evident as in the original image. For image “B”, the smoothing effect is achieved and the color is more likely to be the same in contrast with the Box Blur kernel.
Kernel “c”: Vertical Mean	After applying this kernel to image “A”, evident horizontal lines with noisy points remain because this kernel is taking column averaging. In image “B”, the effect was not as evident as in image “A”, we think it is due to the darkness present in the image. Although, when the image is zoomed, the horizontal lines can hardly be seen, just like the image “A”. Also it can be seen a little improvement in the quality of the image, it has less noise than the original.
Kernel “d”: Horizontal Mean	Once the kernel has been applied to image “A”, the vertical noisy lines can still be observed, obtaining the opposite direction of the lines when the kernel “c” is applied. Kernel “d” is taking row averaging, so this explains the vertical lines. It is seen the same effect in the image “B” just as the past kernel, is not visible enough due to the darkness in the picture, but the noise reduces a little.

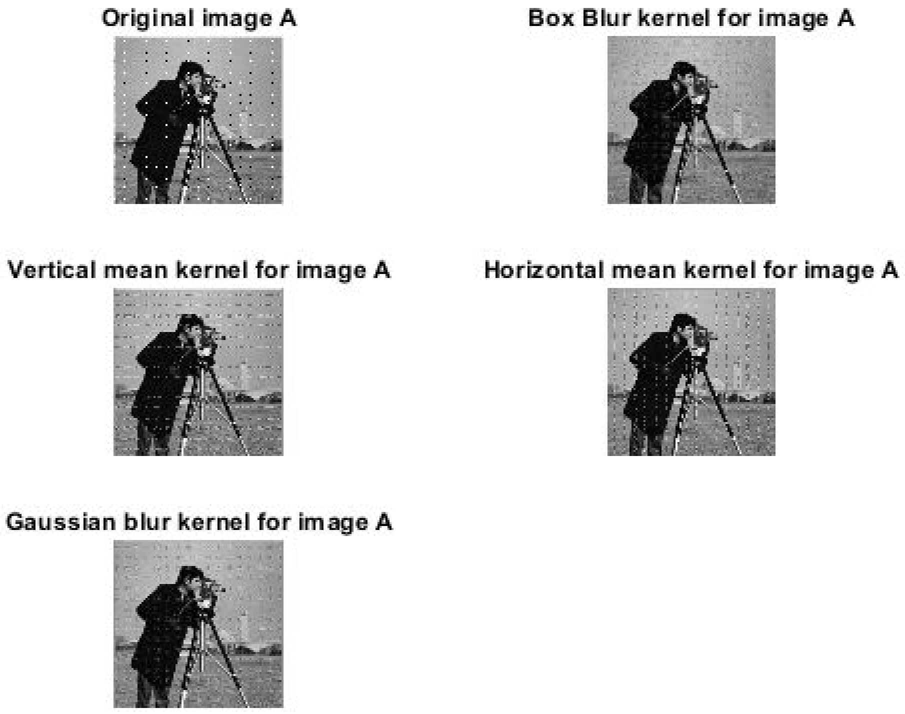

Original image A
Box Blur kernel for image A
Vertical mean kernel for image A
Horizontal mean kernel for image A
Gaussian blur kernel for image A

Figure 3.3.1 Applying kernels to image ”A”

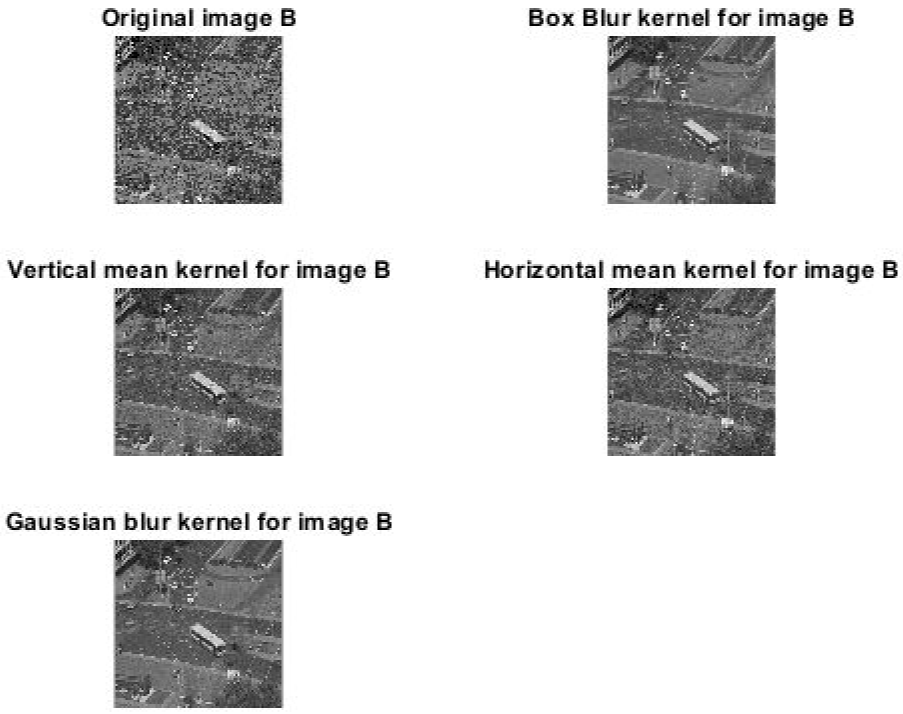

Original image B
Box Blur kernel for image B
Vertical mean kernel for image B
Horizontal mean kernel for image B
Gaussian blur kernel for image B

Figure 3.3.2 Applying kernels to image ”B”

Based on the las two figures, for the image “A”, it can be seen that the kernel in which the image seems more smoother in the “Box blur” or “a” and the “Gaussian blur” or “b”, and it has the same result for the image “B”. This is because the pixels in the images are represented as integers, and after blurring each pixel of the resulting image, has a value equal to the average of the pixels surrounding the pixel analized, including that pixel.

Then, the process of applying the same kernel ten times, is shown for image “A” in figure 3.3.3 and for image “B” in figure 3.3.4. The process of applying “Gaussian blur” or “b” ten times demonstrates that the black and white dots present in the image are eliminated, the details of the image can be seen but the quality can be described as a little blurred when applying the “Gaussian blur” kernel. Even though the original images are very noisy, it can be obtained a much smoother image by applying this specific kernel(figure 3.3.4). This happens also on image “A”, the “salt and pepper” noise are removed and the resulting convolution image is smoother than applying it only once(figure 3.3.3).

By applying ten times the “Vertical mean” kernel or “c”, the image “A” is blurrer vertically, since it is only taking the average by the center column of the 3x3 kernel. Some horizontal lines can still be seen in the image, since those are not considered as important to smooth in this type of kernel(3.3.3). The effect of the convolution with the “Vertical mean”kernel is still not that visible for image “B”, since it does not have the same noise as image “A” and the remaining horizontal noise patterns are not marked as image “A”, but it is worth mentioning that the smoothing effect is notable.

After applying ten times the kernel, the edges of the image are turning black, this is because the 3x3 kernel applied to the borders or corners is not taking the same effect as the pixels close to the center. The main reason is that if the center of the kernel is centered on a pixel of the edge or the corner, there is some part of the kernel which is not overlapping the image, so it has to be taken into consideration for applying convolution multiple times. To solve this problem, a padding on the border can be done to preserve the information and avoid this problem.

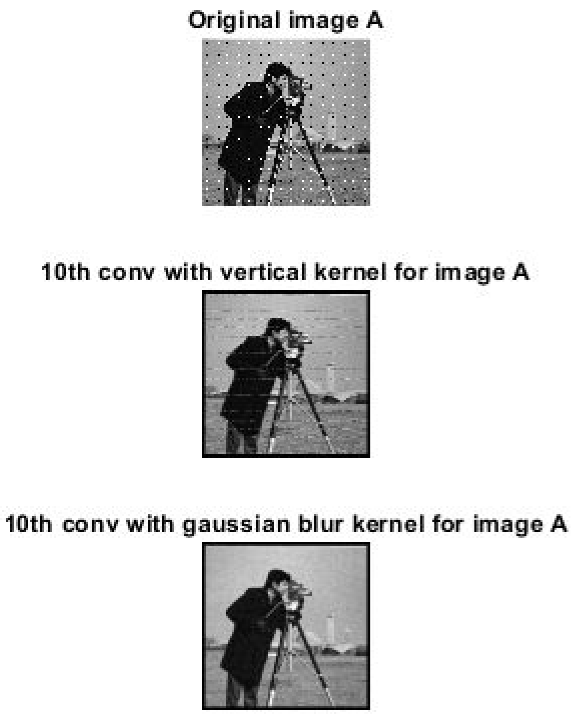

Original image A
10th conv with vertical kernel for image A
10th conv with gaussian blur kernel for image A

Figure 3.3.3 Applying 10 times the kernels selected to image ”A”.

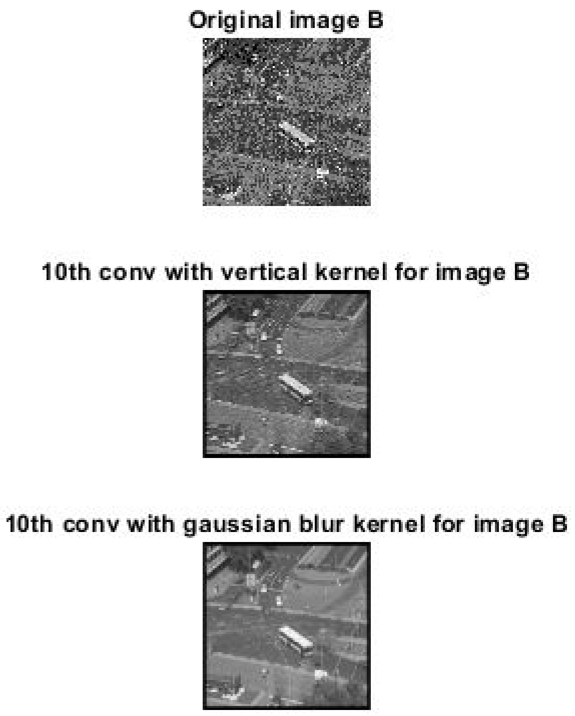

Original image B
10th conv with vertical kernel for image B
10th conv with gaussian blur kernel for image B

Figure 3.3.4 Applying 10 times the kernels selected to image ”B”.

The images with the convolution operations using different kernels applied to the images “C” and “D” are shown in the figure 3.3.5 and 3.3.6 respectively. This is caused by the convolution of kernel ‘e’ and ‘f’, known as “Laplacian kernels” or “Edge detection kernels”; in both cases the convolution results in having the second derivative of the image, producing an image in which the edges of the images are detected and emphasized, while the rest of the image turns black. Because these kernels are approximating the second derivative measurement on the image, they are very sensitive to noise. Therefore, it is often used to smooth the image before applying these two kinds of kernels. By using kernel “f”, the edges are better marked than by using kernel “e”. For instance in figure 3.3.6, there are still some noises above the house, but the image applied “laplacian2” or “f” kernel seems to have less noise and edges better highlighted than the image applied “laplacian” or “e” kernel.

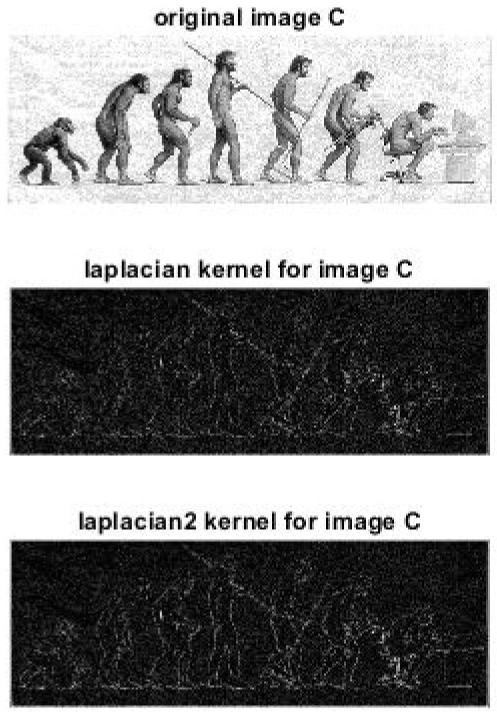


Figure 3.3.5 Image “C” with kernels ‘e’ and ‘f’.

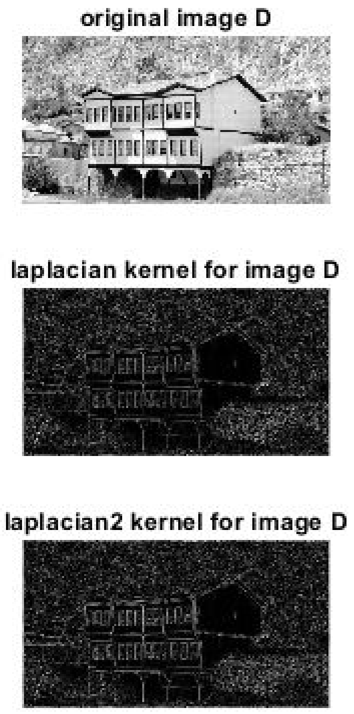


Figure 3.3.6 Image “D” with kernels ‘e’ and ‘f’.

4 CONCLUSION

In conclusion, the objectives were fulfilled, being that after testing our convolution and comparing it with the result obtained by the convolution command from the software, both were identical. The sound effect resulting from the convolution operation between a sound file and impulses in specific parts of the function was done and it is observed as an echo effect.

When applying 2D convolution on images, the effects were seen depending on the type of kernel used, it was analyzed what effects they had on the images, which one favored the most to improve the visibility of the image, it was understood how changing the values of the kernels affected the image and it was understood the process that happens when applied to the image.

On one hand, the use of convolution in audio is useful when making a reverb or echo to a signal, and the appreciation we had when applying it to the audio signal was as expected. On the other hand, the applications of the convolution operation are many, for example, the convolution of images with the edge detection kernels is widely used in machine vision and computer vision, particularly in the areas of feature detection and feature extraction. [3]

5 REFERENCES

[1] Stanley, W. D., Dougherty, G. R., Dougherty, R., & Saunders, H. (1988). Digital signal processing.

[2] Wikipedia. (January 25th of 2021). *Convolution*. Retrieved from Wikipedia: https://es.wikipedia.org/wiki/Convolución#:~:text=Convolución%20de%20un%20pulso%20cuadrado,de%20una%20variable%20ficticia%20τ.

[3] Umbaugh, Scott E (2010). *Digital image processing and analysis : human and computer vision applications with CVIPtools (2nd ed.).* Boca Raton, FL: CRC Press. ISBN 978-1-4398-0205-2.

6 APPENDIX

Coding the convolution sum algorithm from scratch

a) myConv.m: convolution sum function implemented from scratch

```
function y=myConv(x,h)
%x:input signal
%h:impulse response
%y:convolution x*h

%Obtaing length of both inputs of the function x and h
Nx=length(x);
Nh=length(h);

%Pad x with zeros in both sides
Xk=[zeros(1,Nh-1) x zeros(1,Nh)];

%Initializing Hk with zeros
Hk = zeros(length(Xk),length(Xk));

%Generating Matrix of flipped h shifted one place in each row
Hk(1,:) = [flip(h) zeros(1,length(Xk)-Nh)];
for i=2: length(Hk)-length(h)+1
   Hk(i,:)=circshift(Hk(i-1,:),1);
end

Hk = Hk'; %transpose of Hk
y = Xk*Hk;  %Matrix multiplication, also obtaing the convolution result

y = y(1:Nx+Nh-1);  %clipping output result for interested values
end
```

b) Main program

```
%Clearing variables, command window and close all figures
clear all
close all
clc

%Defining input signal x and impuse response h
x1=[0,0,5,2,3,0,0];
h1=[4,1,3];

y1= myConv(x1,h1);%Convolution sum function implemented by scratch
y_MATLAB= conv(x1,h1);  %Convolution sum function provided by MATLAB
fprintf("y1=yMATLAB: "+(sum(y1==y_MATLAB)==size(y1,2)))%Verify result

subplot(4,1,1)
stem(x1)
xlabel('n')
ylabel("x[n]")
title("x[n] vs n")
subplot(4,1,2)
stem(h1)
```

```
xlabel('n')
ylabel('h[n] vs n')
title("h[n] vs n")
subplot(4,1,3)
stem(y1)
xlabel('n')
ylabel('y1[n]')
title("y1 vs n")
subplot(4,1,4)
stem(y_MATLAB)
xlabel('n')
ylabel('yMATLAB[n]')
title("yMATLAB vs n")

% Triangle functions
t=-0.2:0.005:0.2;   %interval
w=0.2;
x2=tripuls(t,w);%Second input signal
h2=[ones(1,4),zeros(1,3),0.5*ones(1,4)]; %second impuse response

y2= myConv(x2,h2);  %Convolution sum of x2 and h2
figure
subplot(3,1,1)
stem(t,x2)
xlabel('t')
ylabel('x[t]')
title("x vs t")
subplot(3,1,2)
stem(h2)  %inteval of t adjusted to match size h2
xlabel('t')
ylabel('h2[t]')
title("h2 vs n")
subplot(3,1,3)
%inteval of t adjusted to match size y2
stem(-0.2:0.005:0.2+0.005*(size(y2,2)-size(x2,2)),y2)
xlabel('y2')
ylabel('y2[t]')
title("y2 vs t")
```

2.3 Convolution and sound

```
%Clearing variables, command window and close all figures
clear all
close all
clc
%import audio file
x= audioread('SpeechDFT-16-8-mono-5secs.wav');
sound(x,8000);  %play original audio
pause(5)    %pause 9 secs to finish audio

%Impulses
h=zeros(1,30000);
h(10000)=0.8;
h(20000)=0.6;
h(25000)=0.4;
```

```
y=conv(x,h);

sound(y,8000); %play output audio

subplot(3,1,1)
plot(x)
xlabel("n")
ylabel('x')
title("x vs n")
xlim([0 size(y,1)]) %adjust size to be same as y
subplot(3,1,2)
plot(h)
xlabel('n')
ylabel('h')
title("h vs n")
subplot(3,1,3)
plot(y)
xlabel('n')
ylabel('y')
title("y vs n")

%save files
audiowrite("soundOriginal.wav",x,8000)
audiowrite("soundConv.wav",y,8000)
```
2.3 Convolution and image
```
%Clearing variables, command window and close all figures
clear all
close all
clc
%% 3.1 import images using the functionimread(filename).
%import audio file
imgA= imread("ImageA.jpg");
imgB= imread("ImageB.jpg");
imgC= imread("ImageC.jpg");
imgD= imread("ImageD.jpg");

%% 3.2 Apply the following kernels to image A and image B
%Defining different 3x3 Kernels
boxBlur_kernel=[1 1 1;1 1 1;1 1 1]/9;
vertical_kernel=[0 0 0;1 1 1;0 0 0]/3;
horizontal_kernel=[0 1 0;0 1 0;0 1 0]/3;
gaussian_blur_kernel=[1 2 1;2 4 2;1 2 1]/16;
%2D convolution for image A with different kernels
boxBlur_imgA=conv2(imgA,boxBlur_kernel);
vertical_imgA=conv2(imgA,vertical_kernel);
horizontal_imgA=conv2(imgA,horizontal_kernel);
gblur_imgA=conv2(imgA,gaussian_blur_kernel);
%showing results of convolution for image A
figure
subplot(3,2,1)
imshow(imgA,[])
title("Original image A")
subplot(3,2,2)
imshow(boxBlur_imgA,[])
title("Box Blur kernel for image A")
```

```
subplot(3,2,3)
imshow(vertical_imgA,[])
title("Vertical mean kernel for image A")
subplot(3,2,4)
imshow(horizontal_imgA,[])
title("Horizontal mean kernel for image A")
subplot(3,2,5)
imshow(gblur_imgA,[])
title("Gaussian blur kernel for image A")

%2D convolution for image B with different kernels
boxBlur_imgB=conv2(imgB,boxBlur_kernel);
vertical_imgB=conv2(imgB,vertical_kernel);
horizontal_imgB=conv2(imgB,horizontal_kernel);
gblur_imgB=conv2(imgB,gaussian_blur_kernel);
%showing results of convolution for image B
figure
subplot(3,2,1)
imshow(imgB,[])
title("Original image B")
subplot(3,2,2)
imshow(boxBlur_imgB,[])
title("Box Blur kernel B")
subplot(3,2,3)
imshow(vertical_imgB,[])
title("Vertical mean kernel B")
subplot(3,2,4)
imshow(horizontal_imgB,[])
title("Horizontal mean kernel B")
subplot(3,2,5)
imshow(gblur_imgB,[])
title("Gaussian blur kernel for image B")

%% 3.5 Apply the convolution more than once
vertical_imgA10=imgA;
gblur_imgA10=imgA;
vertical_imgB10=imgB;
gblur_imgB10=imgB;
for i=1:10
   vertical_imgA10=conv2(vertical_imgA10,vertical_kernel);
   gblur_imgA10=conv2(gblur_imgA10,gaussian_blur_kernel);
   vertical_imgB10=conv2(vertical_imgB10,vertical_kernel);
   gblur_imgB10=conv2(gblur_imgB10,gaussian_blur_kernel);
end
figure
subplot(3,1,1)
imshow(imgB,[])
title("Original image B")
subplot(3,1,2)
imshow(vertical_imgB10,[])
title("10th conv with vertical kernel for image B")
subplot(3,1,3)
imshow(gblur_imgB10,[])
title("10th conv with gaussian blur kernel for image B")
figure
```

```
subplot(3,1,1)
imshow(imgA,[])
title("Original image A")
subplot(3,1,2)
imshow(vertical_imgA10,[])
title("10th conv with vertical kernel for image A")
subplot(3,1,3)
imshow(gblur_imgA10,[])
title("10th conv with gaussian blur kernel for image A")

%% 3.5 Apply the following kernels for image C and image D

laplacian_kernel=[0 1 0;1 -4 1;0 1 0];
laplacian_kernel2=[1 1 1;1 -8 1;1 1 1];
%2D convolution for image C and image D with different kernels
laplacian_imgC=abs(conv2(imgC,laplacian_kernel));
laplacian2_imgC=abs(conv2(imgC,laplacian_kernel2));
laplacian_imgD=abs(conv2(imgD,laplacian_kernel));
laplacian2_imgD=abs(conv2(imgD,laplacian_kernel2));
%showing results of convolution for image C and image D
figure
subplot(3,1,1)
imshow(imgC,[])
title("original image C")
subplot(3,1,2)
imshow(laplacian_imgC,[])
title("laplacian kernel for image C")
subplot(3,1,3)
imshow(laplacian2_imgC,[])
title("laplacian2 kernel for image C")
figure
subplot(3,1,1)
imshow(imgD,[])
title("original image D")
subplot(3,1,2)
imshow(laplacian_imgD,[])
title("laplacian kernel for image D")
subplot(3,1,3)
imshow(laplacian2_imgD,[])
title("laplacian2 kernel for image D")
```