Reza Jahani	Digital Signal Processing	Due Date: 17/10/1400
810198377	CA3 Report	Teacher: Dr Badiei

▪ Section I: Voice Signal Processing

(+) For the first part we consdier α=0.8 as already told.
To provide an echo with 0.15s delay we'd better choose β=$0.15fs$ where f_S = 48KHz.

$$y[n] = x[n] + 0.8x[n - 7200]$$

To gain the output of this system we convolve x[n] which is the voice signal we already accuired with h[n] which is:

h[n]=$\delta[n] + 0.8\delta[n - 7200]$

(+) After conduction of convolution and adding random noise to signal, the audio file is saved in file noisy_echoed_voice.wav

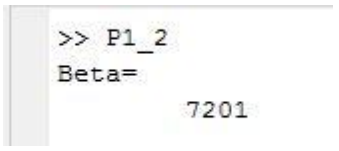
- File for this part is **P1_0.m**

Question 1)

(+)

- The desired function is written in **necho.m**

This function was used with the audio file "noisy_echoed_voice.wav" which we saved in the end of previous question and the results can be observed in figure 1.1
Also the audiofile "noisy_voice.wav" is saved as desired. (This action is done by the function necho())



```
>> P1_2
Beta=
        7201
```

Figure 1.1

In next (+) we discuiss the logic behind the function necho() and we will compare the Beta we accuired using the function and the Beta we used to add echo to the original voice.

- Code for this part is **P1_1.m**

(+) In this part we want to first compare the Beta obtained from the function and the Beta we used for adding echo to the voice. As we can see in figure 1.1 the Beta we accuired is 7201 while the Beta we used to add echo the voice was 7200.
The reason behind this difference is presence of random noise which changes everytime we run the code. If there was no noise we were able to reach the exact amount of 7200, although in this case in presence of noise the nature of signal would change a little and the cepstrum would not be the cepstrum with those expected peaks.

Now let's take a look at the algorithm we used to remove echo. We already knew that for adding echo to the signal, we are using a system with system function presented in equation 1.1

$$H(z) = 1 + \alpha z^{-\beta} \qquad (eq\ 1.1)$$

Consider we have obtained β using the logic presented at the beginning of the instructions of project. By this, I mean the relation β has with peaks in cepstrum.

Assuming we know what β is:

$$z(t) = y(t) + n(t)$$

where n(t) is the noise and y(t) is the signal with echo, output of system presented in equation 1.1.

$$H(z) = \frac{Y(z)}{X(z)}$$

$$Y(z) = H(z)X(z)$$

Now we are going to use a filter which is presented in equation 1.2

$$H_i(z) = \frac{1}{1 + \alpha z^{-\beta}} \qquad (eq\ 1.2)$$

Now we pass z(t) through the system presented by equation 1.2, naming the output w(t)

$$W(z) = Z(z)H_i(z)$$

$$W(z) = \big(Y(z) + N(z)\big)H_i(z)$$

$$W(z) = Y(z)H_i(z) + N'(z)$$

$$knowing\ H_i(z) = \frac{1}{H(z)}$$

$$W(z) = X(z) + N'(z)$$

$$applying\ Inverse\ Z - Trans.$$

$$w(t) = x(t) + n'(t)$$

(Equation 1.3)

Based on equation 1.3, we proved that the output of a system presented by eq 1.2 will be the original voice added to a noise term which we plan to remove in next question.

(+) In this part we aim to plot time-domain & frequency-domain of the noisy echoed voice and the noisy voice.

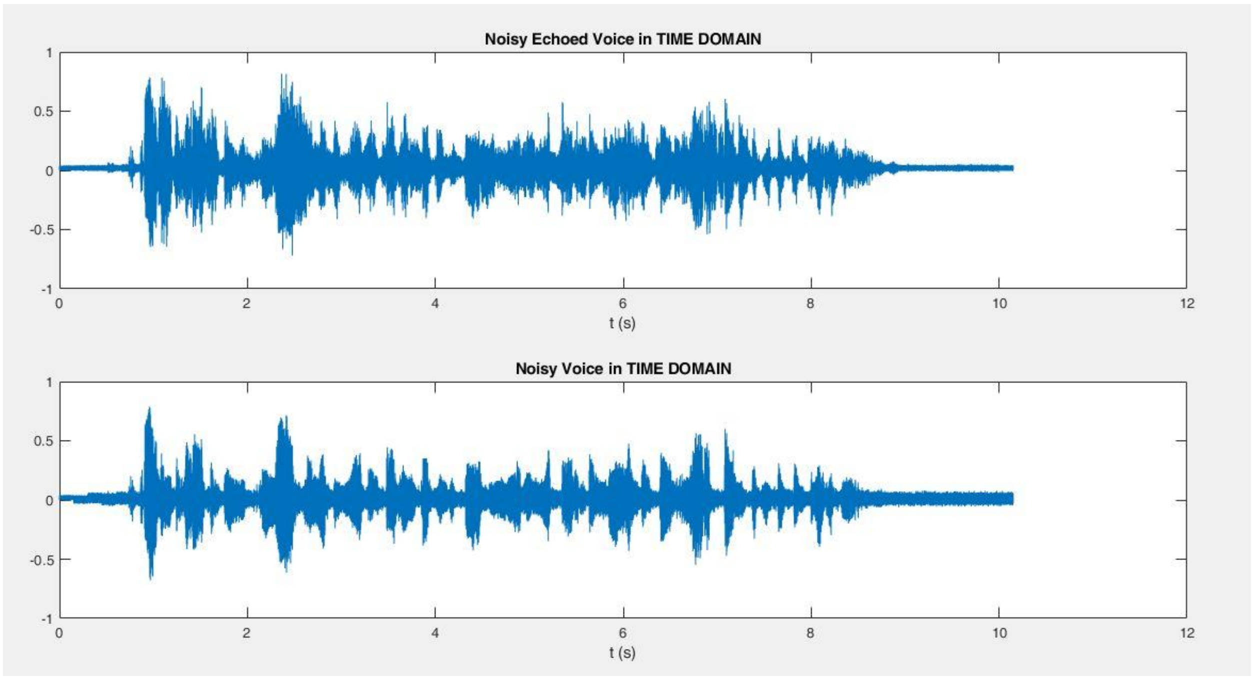


Figure 1.2: Noisy Echoed Voice and Noisy Voice in time domain

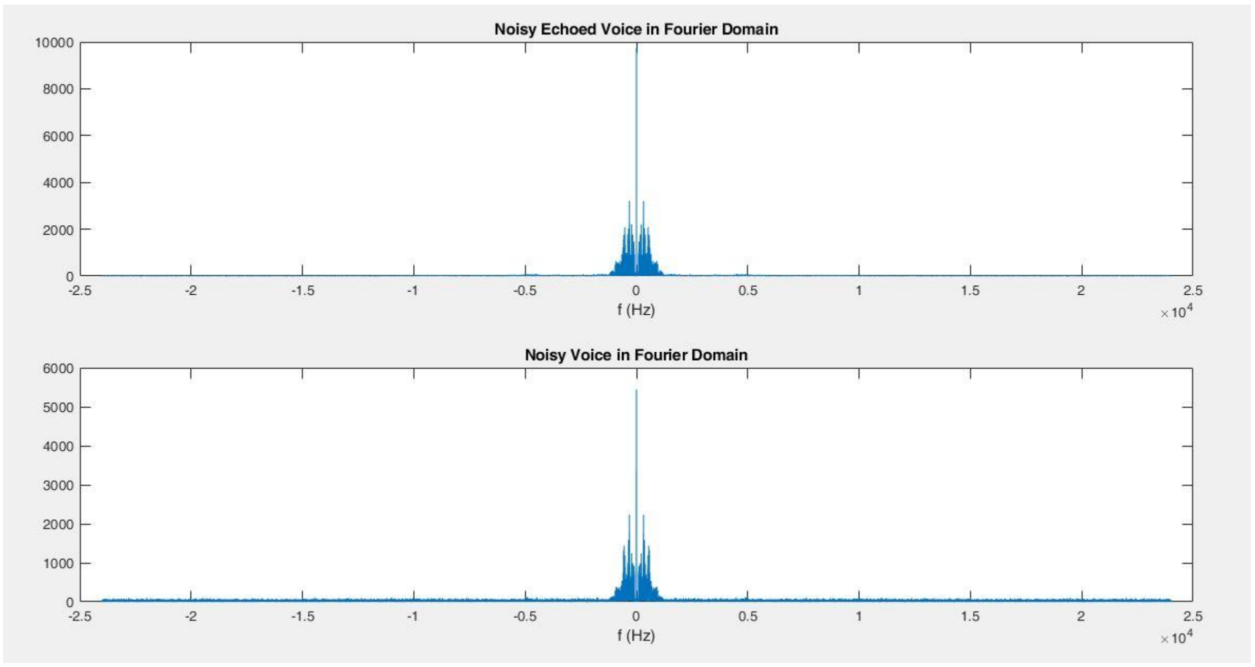


Figure 1.3: Noisy Echoed Voice and Noisy Voice in frequency domain

As we expected there is a little difference between the diagrams and the reason is obviously the echo. For example in time domain, the moment the echo starts signal's amplitude gets different from the voice without echo and this fact can be observed easily in figure 1.2
Mentioned in previous paragraph this difference causes a little difference in fourier transfrom too. The difference which can be observed in figure 1.3 that reflects this fact that signal's fourier transfrom are different a little. As we know

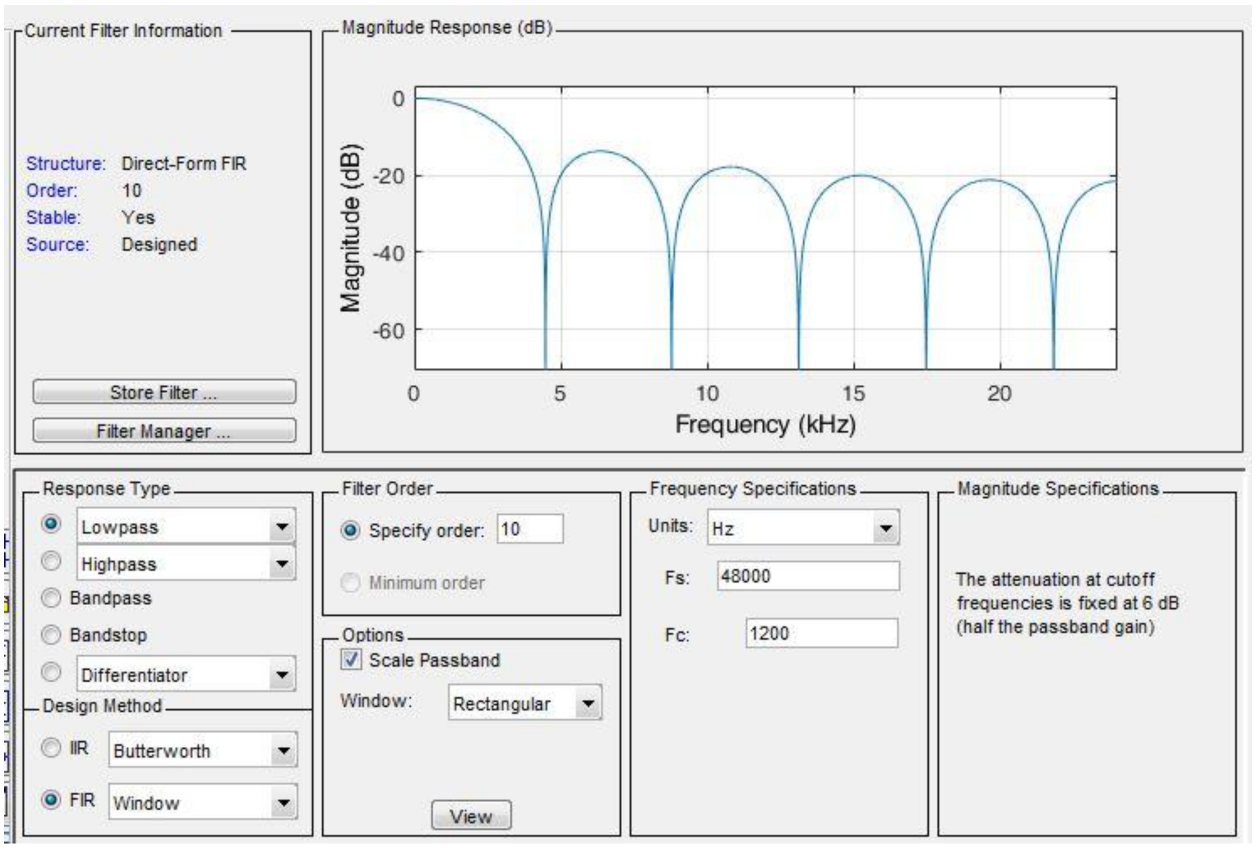
$$y[n] = x[n] + \alpha x[n - \beta]$$
$$Y(e^{j\omega}) = (1 + e^{-j\beta\omega})X(e^{j\omega})$$

- Code for this part is **P1_2.m**

Question 2)

(+)
The voiced speech has a fundamental frequency which for adult male varies from 85Hz to 155Hz while this frequency for an adult female is in range 165Hz to 255Hz.The consonants are found in frequencies above 500 Hz. Accurately speaking in range of 2KHz-4KHz.
Bandstop edge for human voice can be cosidered 3800 Hz.
Bandstop attenuation can be considered more than 80 dB.
Passband ripple can be considered 3 dB.
All these parameters for human voice are accuired with research.

(+)
Comparing the diagram we already accuired, we can conclude that the main part of the frequency band of the signal is in range (0-1200) Hz.
So the only thing we can do is using a LPF to get rid of the out of band noise and decrease power of noise which is outside the frequency band.
We have to set the cut off frequency at 1200 Hz and steps of designing the filter can be observed in figure 1.4



Current Filter Information

Structure:	Direct-Form FIR
Order:	10
Stable:	Yes
Source:	Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Window]

Filter Order: [✓ Specify order: 10] Minimum order

Options: [✓ Scale Passband]
Window: Rectangular

Frequency Specifications
Units: Hz
Fs: 48000
Fc: 1200

Magnitude Specifications
The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

Figure 1.4: Desiging the LPF Filter using rectangular window

As already mentioned the cut off frequency of the filter which should be employed is supposed to be 1200 Hz. This parameter is set at 1200 Hz as required. Rectangular window is employed.

Cleared voice is saved in array ***y*** which by using the function sound() we can notice a huge difference which is because of removing the noise outside the frequency band of the signal

In next question, input voice and the cleared voice are plotted both in time and frequency domain.

(+) In figure 1.5 and figure 1.6, noisy signal and cleared signal can be observed in time domain and frequency domain respectively.

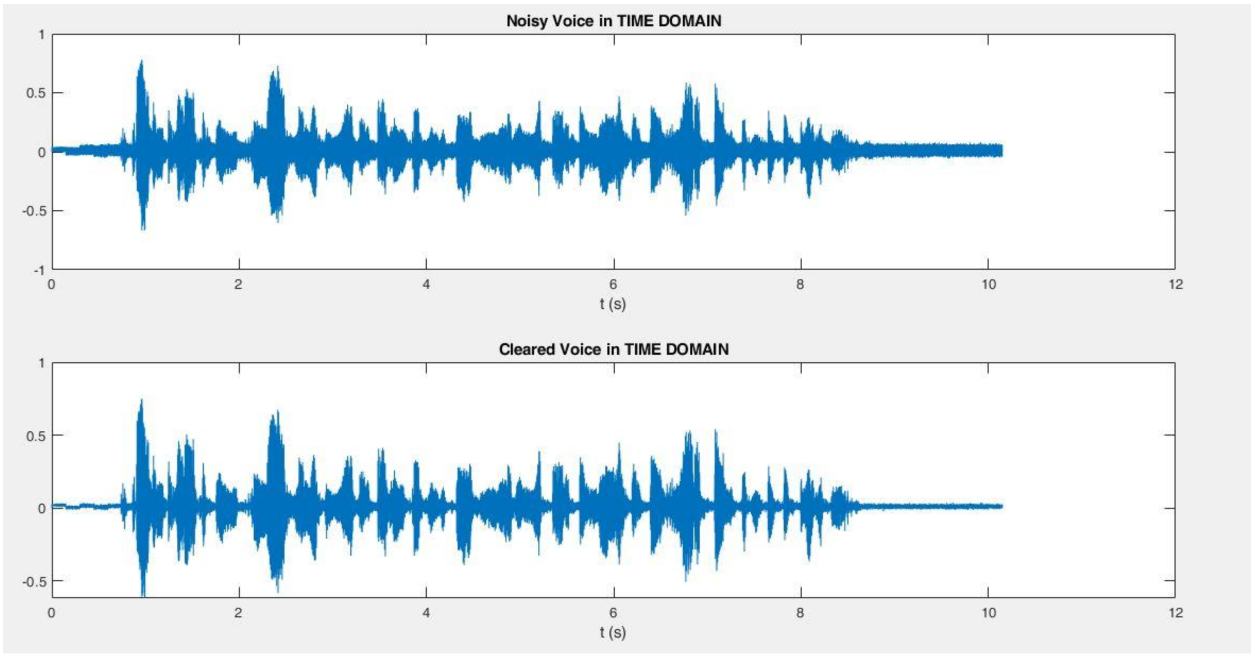


Figure 1.5: Time domain

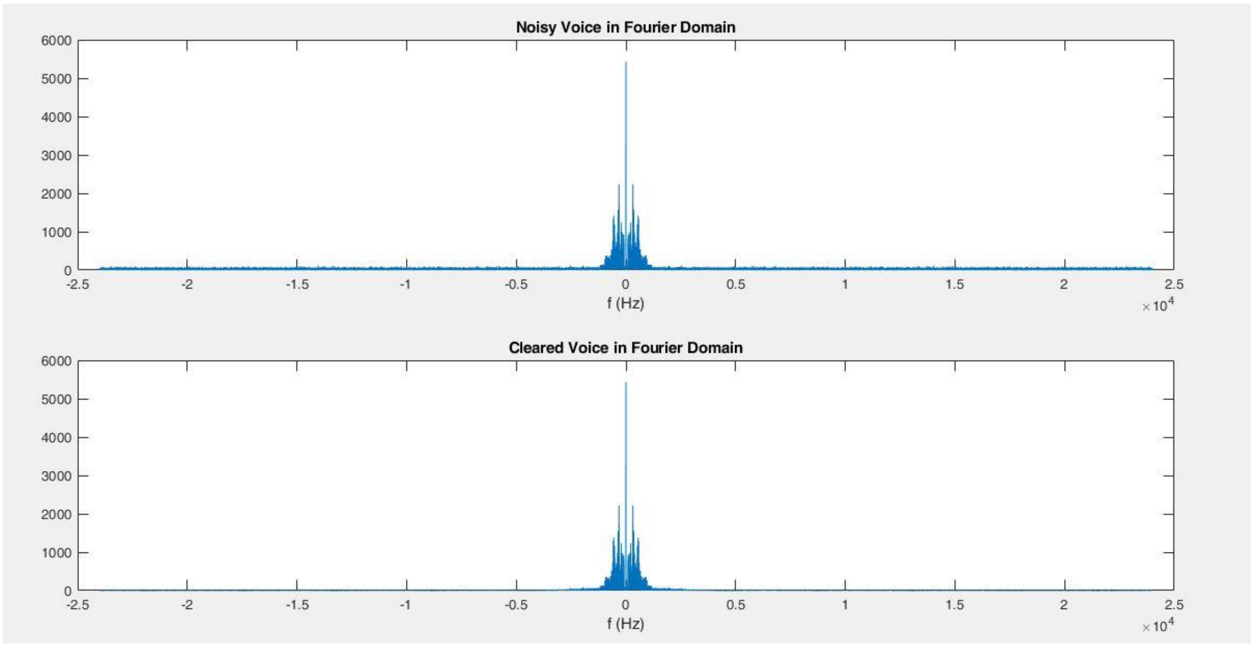


Figure 1.6: Frequency Domain

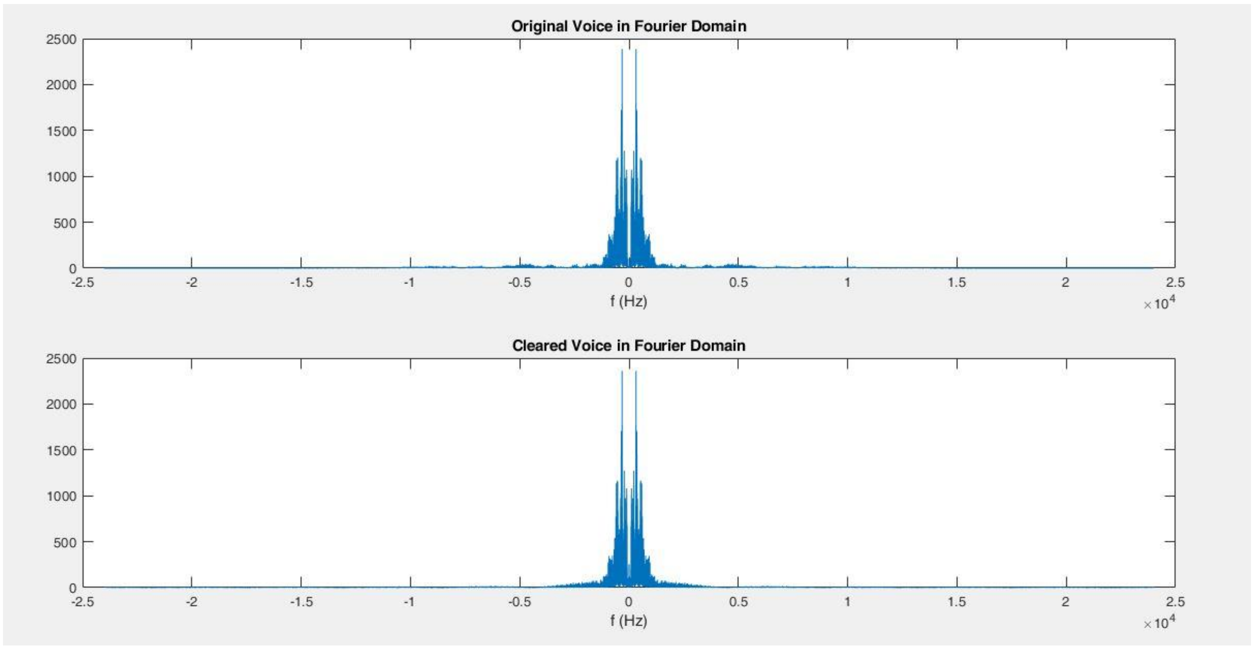
As it can be observed a little difference can be noticed between cleared voice and noisy voice in both frequency and time domain.
Obviously the reason behind this fact is that after clearance of noise the output signal gets closer the original voice. It can be totally noticed that power of noise has decreased.
Also this fact should be mentioned that by removing the noise, we can only remove noise outside the frequency band of the signal and still a little noise will be left inside the band of the signal but SNR will be high enough that noise won't be detected that much.
For further comparisons cleared voice and original voice will be plotted in time and frequency domain in next part.

We can see that in cleared voice in the ending void which nothing is said the amplitude of the signal has decreased while in the noisy voice the amplitude there is more.
Also we can see that fourier transform of the signal is no longer non-zero outside the band.

- Code for this part is in file **P1_3.m**
- The filter designed using fdatool is in file **Filter.m**

Comparison between accuired vioce and original voice:

As stated before, for further and accurate analysis and comparison accuired output voice will be compared with the original voice recorded in the beginning of the project.



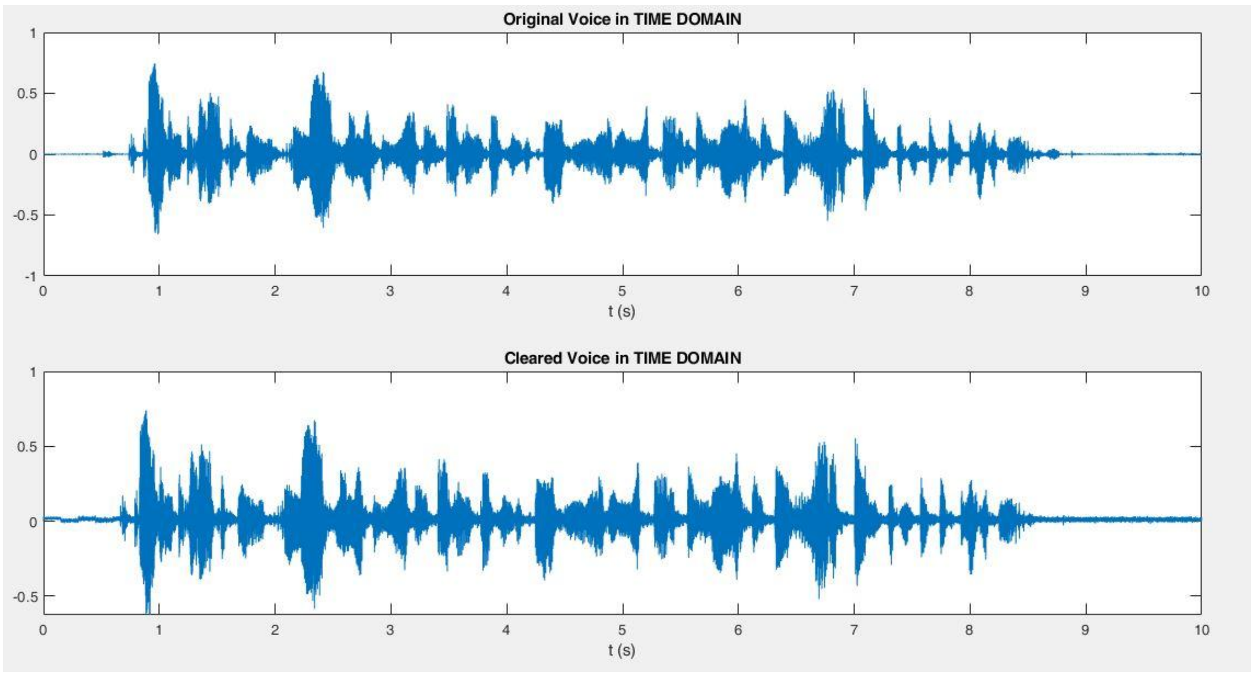


Figure 1.7: Time & Frequency Domain Presentation of original and cleared voice

As presented in figure 1.7 we can easily notice that the output is so close the original voice since their fourier transform is so similar and also in time domain, ignoring that little noise left on the signal due to the fact that it can not be removed, signals are so similar.

Objective completed and we were able to accuire the voice without echo and removing the noise as much as it was possible.

- Code for this part is **P1_4.m**

▪ Section II: Image Processing

(+) In this part we aim to:

- Implement different kernels on the image file.
- Present the output to each kernel.
- Explain how each kernel works.

The code which has been written for this part asks the user to choose the kernel respectively as the instruction of the project. e.g 1 stands for sharpen and 8 stands for identity kernel.
The original image is brought in figure 2.1 to be able to compare the original image with the outputs of each kernel.

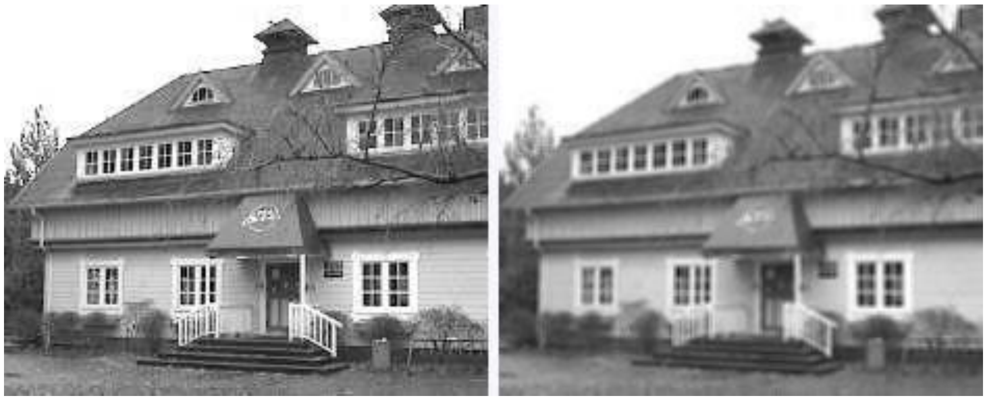


Figure 2.1: House Image

In next pages we employ the kernels presented in the instructions and by entering the number refering to each kernel the code written produces the output which can be observed in figures 2.2 to 2.9
In table below keys to each filter is instructed.

Filter	**Key**	**Filter**	**Key**
sharpen	1	Avg moving	5
blur	2	Line H	6
outline	3	Line V	7
gauss	4	identity	8

After entering each key once the code has run, following results will be presented.

Sharpen Kernel)

- This filter increases the difference in adjacent pixel values making the image look more vivid.

The output of this kernel can be observed in figure 2.2

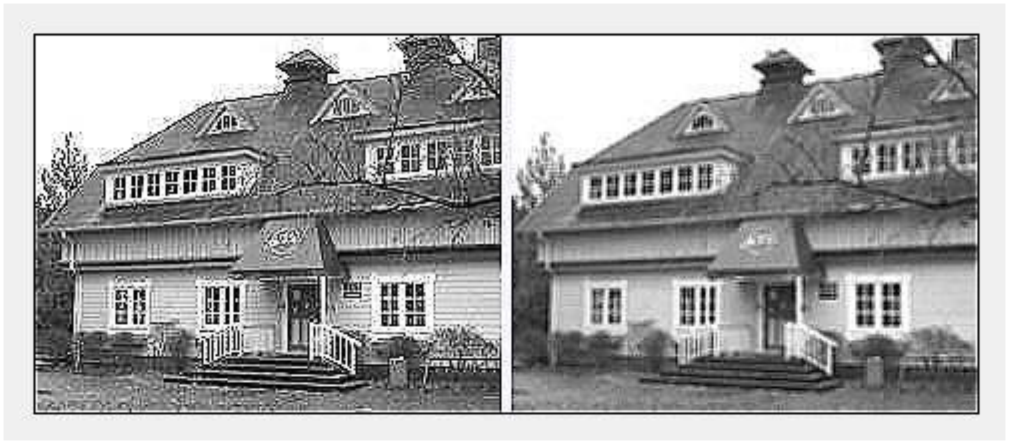


Figure 2.2: Output of Sharpen Kernel

Blur Kernel)

- By bluring an image, the color alteration and transition from one side of an edge to another is made rather smooth than sudden.

The output of Blur Kernel is presented in figure 2.3

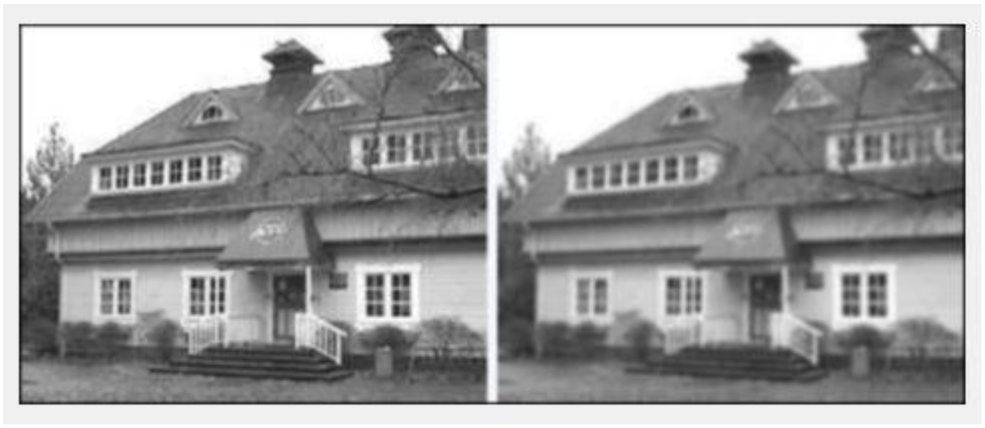


Figure 2.3: Output of Blur Kernel

Outline Kernel)

- ❖ This kernel is employed to highlight large differences in pixel values. A pixel which is next to a pixel with the same intensity turns black while a pixel next to a pixel with strong differnece turns white.

Output of this kernel can also be observed in figure 2.4

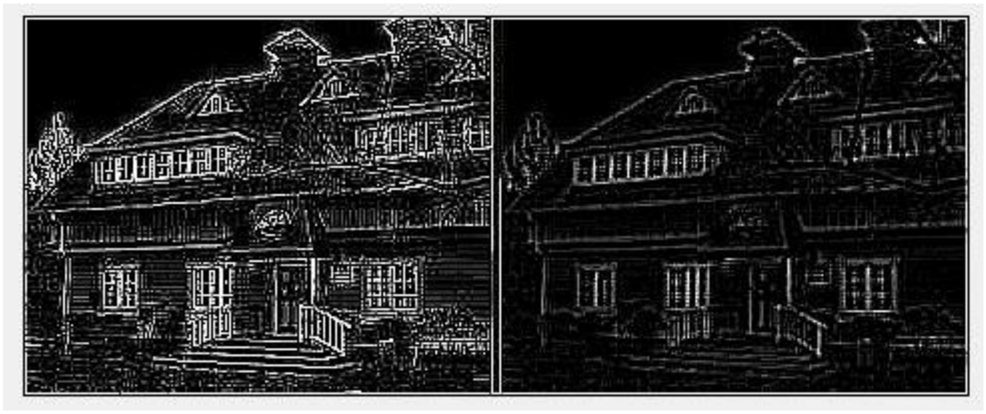


Figure 2.4: Outline Kernel

Summing up so far, we can observe that these filters have already created the output we expected based on their functionalities.

Gauss Kernel)

- ❖ This kernel outputs a weighted average of each pixel neighborhood. The effect is more like the blur kernel.

The output to this kernel is shown below in figure 2.5

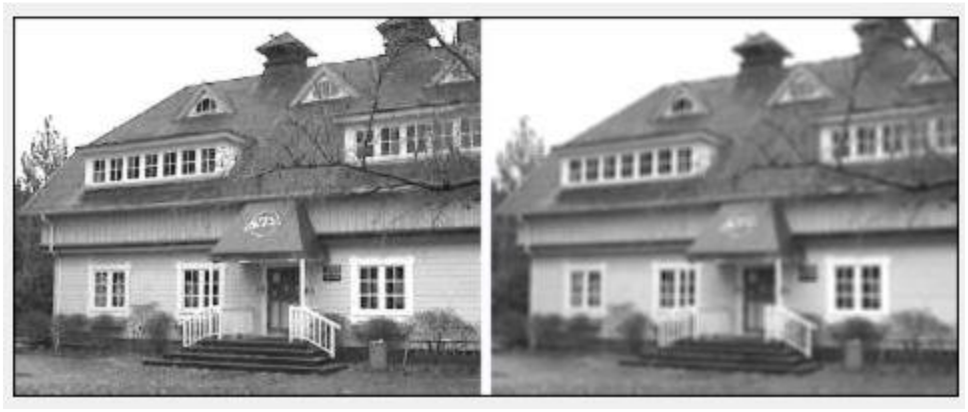


Figure 2.5: Gauss Kernel

Avg Moving Kernel)

- This filter is used to smooth the image by reducing the amount of intensity variation between neighbour pixels. It works by moving through the pixels and replacing them with the average value of the neighbouring pixels and the pixel itself.

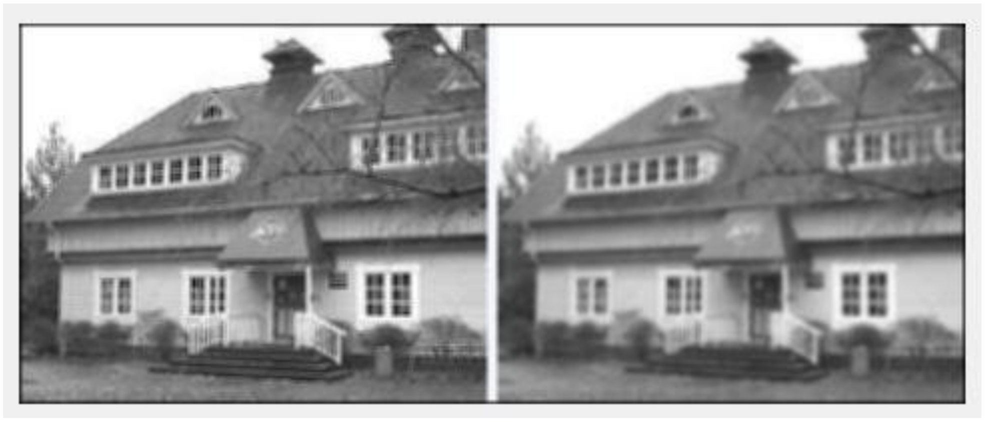


Figure 2.6: Moving Average Kernel

Line H)

- This kernel detects the horizental lines where there is a large difference on both sides.

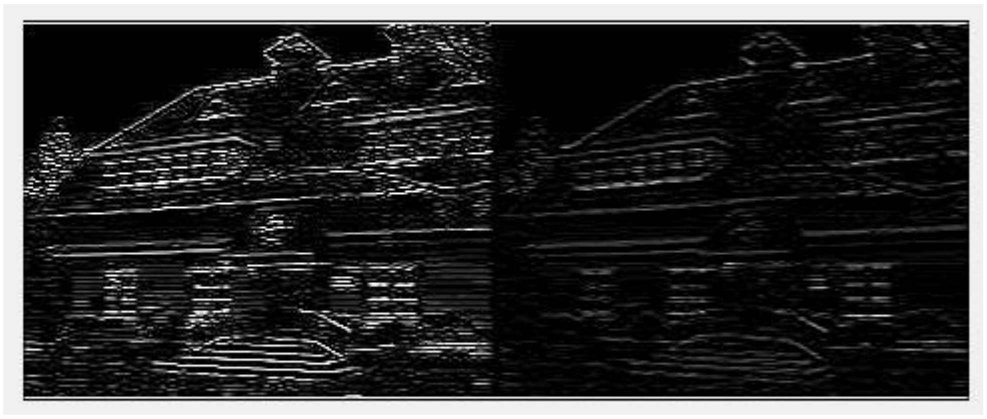


Figure 2.7: Line H kernel

Line V)

- ❖ This kernel detects the horizental lines where there is a large difference on both sides.

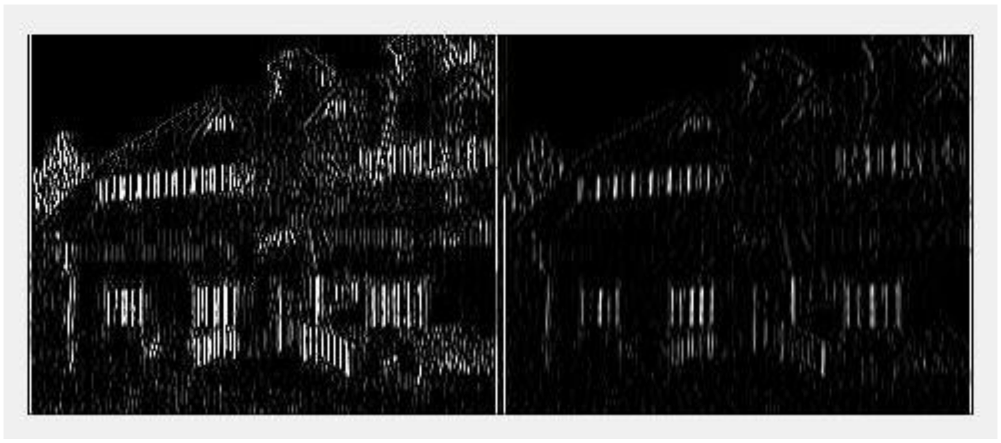


Figure 2.8: Line V Kernel

Identity)

- ❖ This kernel returns the input as the output with no change.

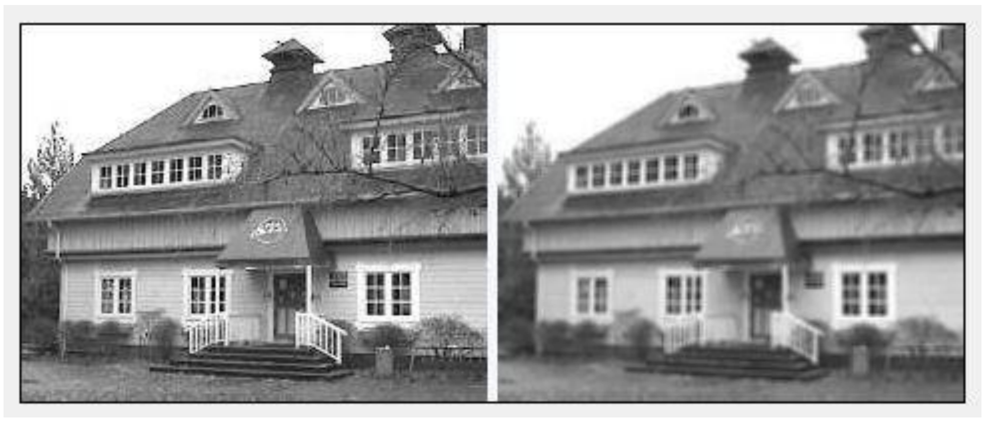


Figure 2.9: Identity Kernel

- Code for this part is in file **P2_1.m**

(+) Now in this part, we first use imresize() function to resize the image with scale of 1/5. Then by employing the function again with scale of 5 and method of 'nearest', we reconstruct the image. After this step we will increase the quality of the picture using **guassian and moving avg kernels.**
In each part the result that we can observe will be presented.

Original picture is brought below in figure 2.10

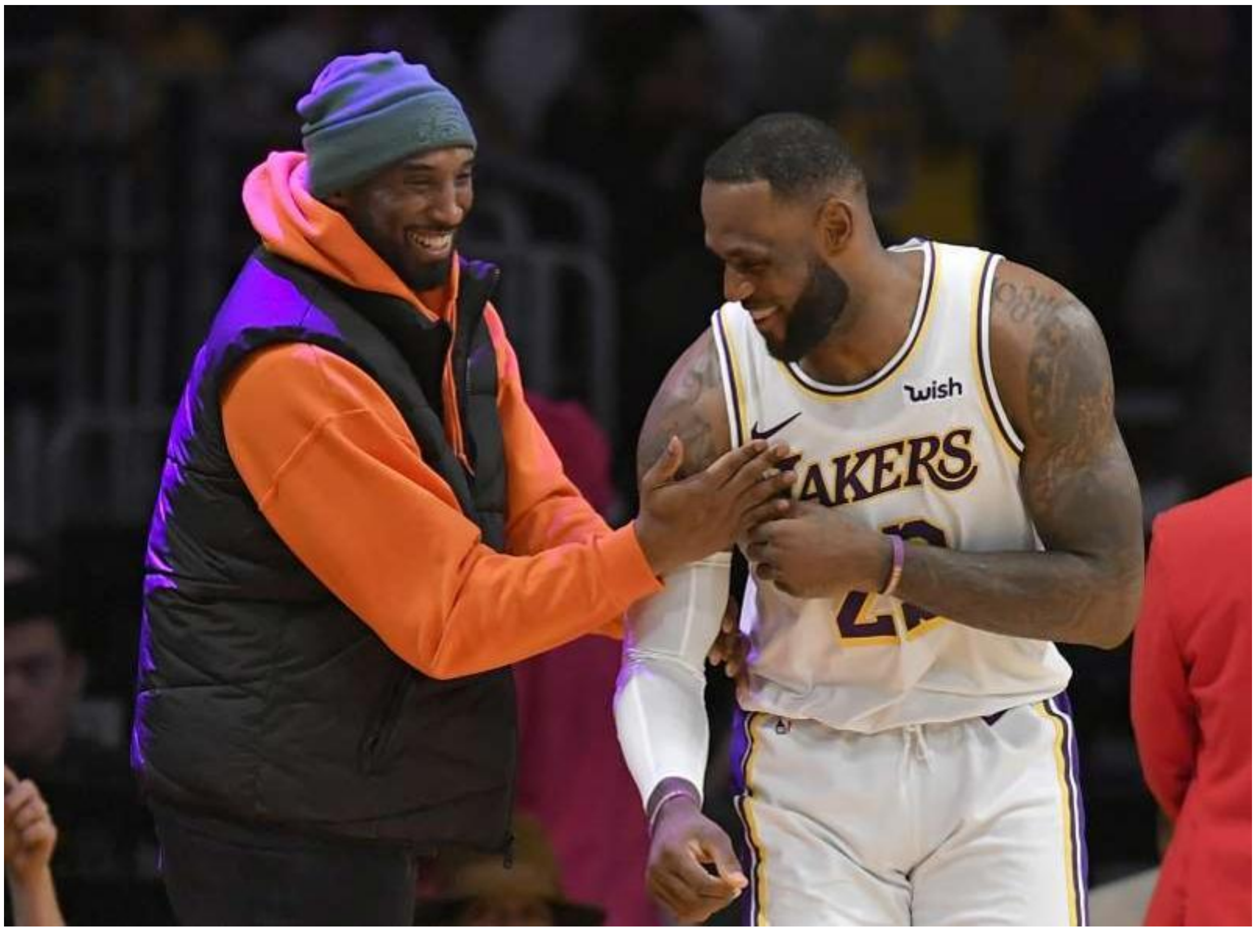


Figure 2.10: The original picture

First output which is the resized image with scale of 1/5 is preseted in figure 2.11

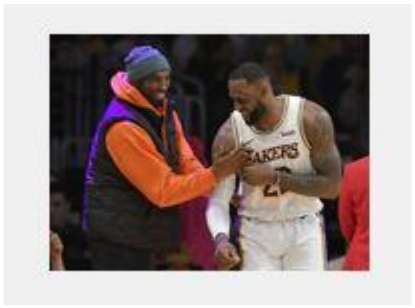


Figure 2.11: Resized Image with scale of 1/5

As we expected the size of the image is scaled with factor of 1/5.

Now in figure 2.12 the reconstructed image is presented.

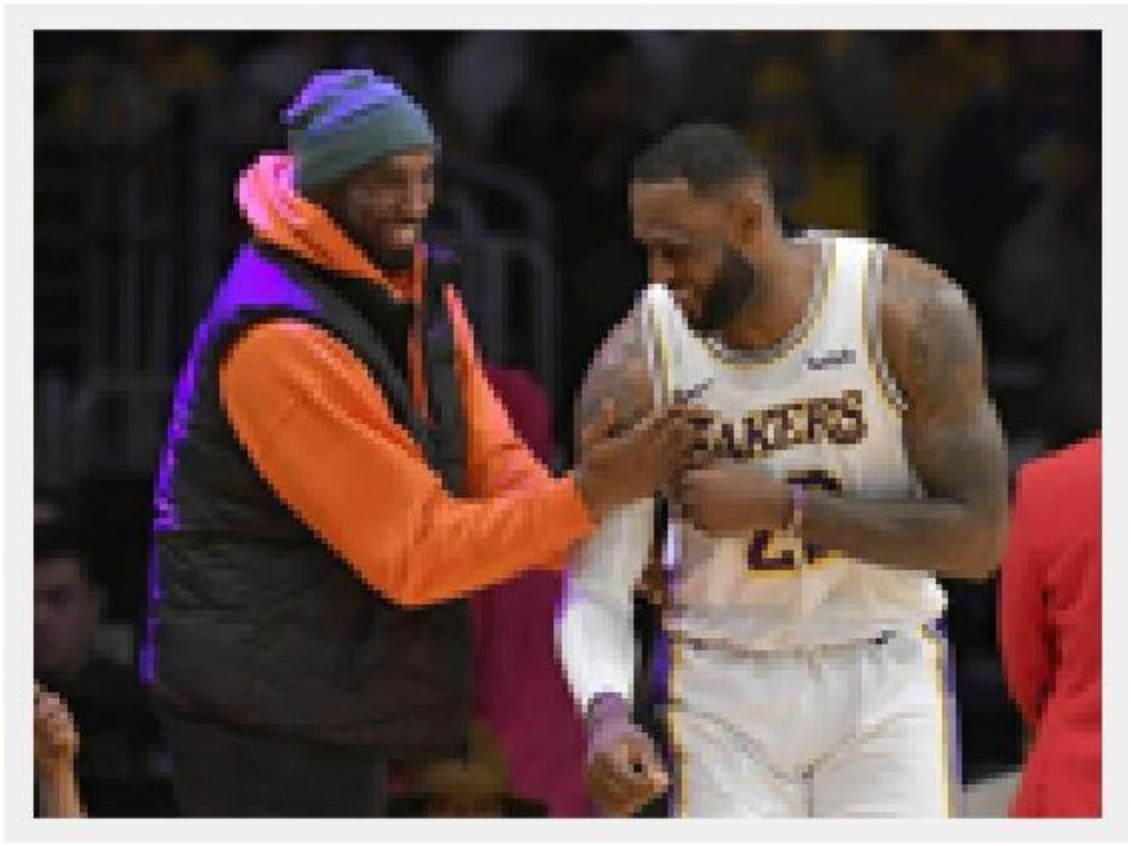


Figure 2.12: Reconstructed Image with factor of 5

As it can be observed the reconstructed image is not the same as the original picture. The reason behind this is that after we scaled the size of the picture with factor of 1/5, some data has been thrown away and lost to reduce the size of the picture and overally the number of pixels presenting the picture has decreased. After rescaling the image with factor of 5, we no longer have access to those lost pixels and data which we threw away in first step. In conclusion as we can see, number of the pixels are reduced and the quality of the picture is not like the first original picture we had in the beginning.

As instructed, we employed guassian & avg moving kernels to increase the quality of the picture and further result can be observed in figures 2.13 and 2.14

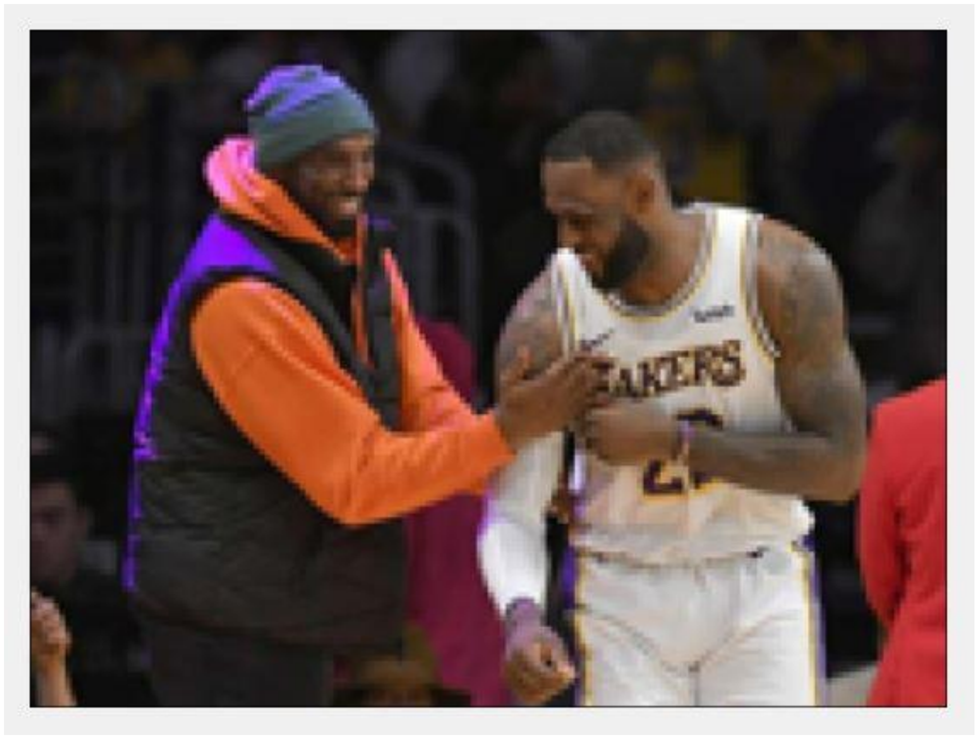


Figure 2.13: After using Avg Moving Kernel

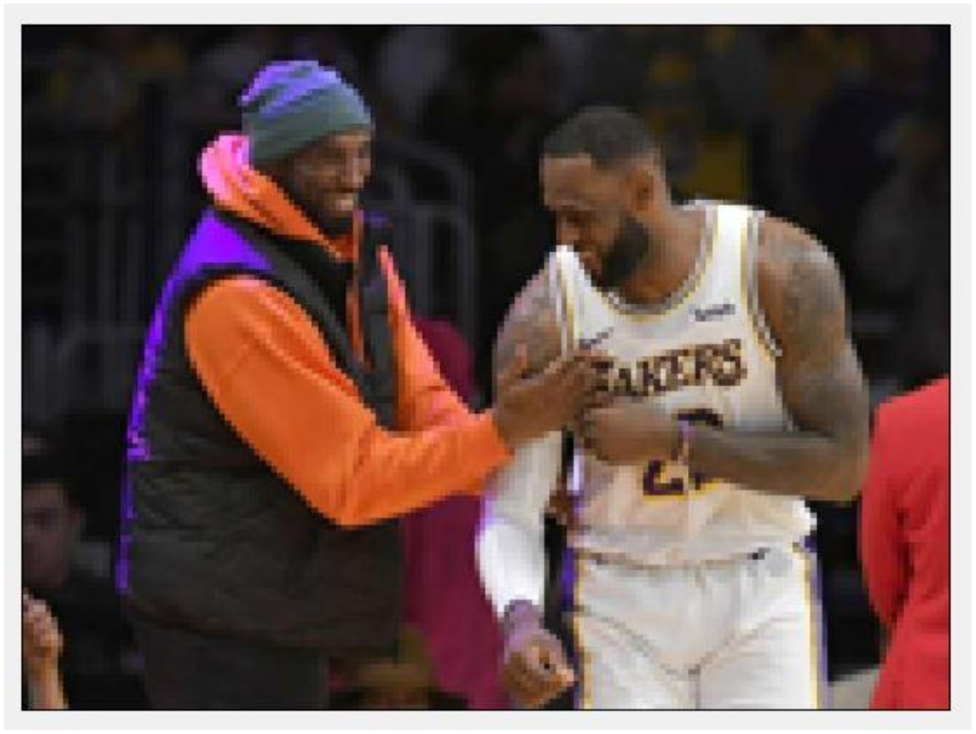


Figure 2.14: After using Guassian Kernel

- This part code is in file **P2_2.m**

(+) [Bonus]
In this part we aim to write a program to detect horizental and vertical edges of an object in a picture. The algorithm which is used for object localization is explained below:

First the image is loaded into a matrix. After converting the image to gray scale we use line V and line H kernels on the image to find the edges. Note that these kernels find the pixels which are having pixels around them that one side is very different from the other side. (line H finds these pixels horizentaly and line V finds these pixels vertically) and converts them to white pixel in the output.
After implementing these kernels on the image, we find those pixels which are white. In fact we have successfully found the edges. Now we find the location of these pixels and finaly after finding for how long we need to draw the rectangle and after computing the width and length of these lines, using rectangle() function, we will draw a red rectangle around the object. The result of the program can be observed in figure 2.15

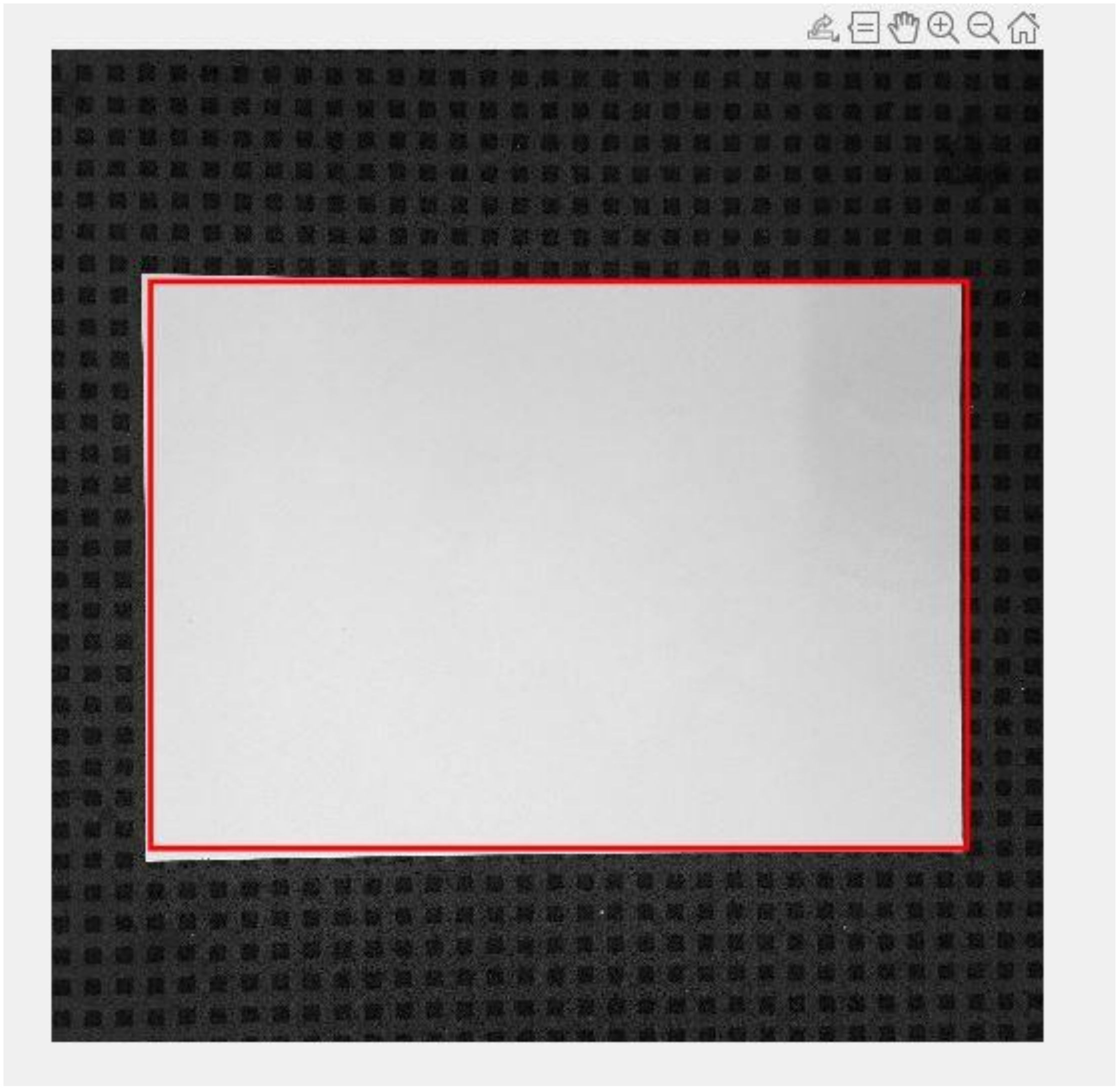


Figure 2.15: Object Localization