ABSTRACT

*Most images have noise which creates blur and degrade the quality of image. Image analysis can be obtained by using various filtering techniques. Generally, filtering can be done in frequency domain where we use Fourier transform. For this project, we have analyzed smoothening functions like using Butterworth low pass filter, ideal low pass filter and Gaussian low pass filter.*

Contents

[**NOISE REMOVAL FROM 2-D SIGNAL** 4](#_Toc91460161)

[**1.** **Introduction** 4](#_Toc91460162)

[1.1) Smoothening: 5](#_Toc91460163)

[1.2) Fourier Transform 5](#_Toc91460164)

[**2.** **Frequency Domain Filtering** 5](#_Toc91460165)

[**3.** **Low Pass Filtering** 6](#_Toc91460166)

[*3.1)* *Rectangular Low Pass filter:* 6](#_Toc91460167)

[*3.2)* *Butterworth Low Pass filter:* 6](#_Toc91460168)

[*3.3)* *Gaussian Low Pass filter:* 6](#_Toc91460169)

[**4.** **Implementation** 7](#_Toc91460170)

[*4.1)* *Image loading and pre-processing:* 8](#_Toc91460171)

[*4.2)* *Fourier Transform:* 9](#_Toc91460172)

[*4.3)* *Rectangular Filter Design:* 10](#_Toc91460173)

[*4.4)* *Butterworth Filter Design:* 11](#_Toc91460174)

[*4.5)* *Gaussian Low Pass Filter Design:* 12](#_Toc91460175)

[*4.6)* *Inverse Fourier Transform:* 13](#_Toc91460176)

[*4.7)* *Outputs:* 14](#_Toc91460177)

[**5.** **Results** 15](#_Toc91460178)

[*5.1)* *Rectangular Filter:* 16](#_Toc91460179)

[5.2) *Gaussian Filter:* 17](#_Toc91460180)

[5.3) *Butterworth Filter:* 18](#_Toc91460181)

[5.4) *Summary:* 18](#_Toc91460182)

[**6.** **Salt and Pepper Noise Removal:** 19](#_Toc91460183)

[6.1) *Addition of Salt and Pepper Noise:* 20](#_Toc91460184)

[6.2) *Removal of Salt and Pepper Noise:* 21](#_Toc91460185)

[**7.** **Conclusion:** 24](#_Toc91460186)

**NOISE REMOVAL FROM 2-D SIGNAL**

# **Introduction**

*Removal of unwanted disturbances from our signal is called noise removal. Filtering techniques are used to remove noise from our signal. Filtering can be done in either time domain or frequency domain. Our implementation of filters is done in frequency domain.*

*The signal we were given was a 2-D signal, meaning we were given an* ***image*** *with noise in it. As mentioned earlier, image being a 2D signal, we can represent an image in the form (x, y) in spatial domain where (x, y) are spatial coordinates.*

*We use the techniques of 1D signal manipulation to manipulate an image. One such technique is to transform the image into frequency domain by taking the Fourier transform to remove any unwanted disturbance known as noise. All frequency filters can also be implemented in the spatial domain. If there is a simple kernel for the desired filter effect. Furthermore, it is computationally less costly to perform the filtering in the spatial domain. Usually noise attacks the higher frequency, hence it can be removed by using a low pass filter.*

Figure Noise Removal in Frequency domain

## Smoothening:

*In smoothening what we do is we actually perform low pass filtering. Low pass filtering means allowing the low frequency components to pass and restrict high frequency components.*

*To apply a low pass filter in frequency domain, we have to transform our signal to the frequency domain. This can be done using the Fourier transform.*

## Fourier Transform

*Fourier transform named after the French mathematician Jean Baptiste Joseph Fourier is one of the most important transforms that is used in Image Processing. Fourier transform is understood with the help of this real world example:  
'White light when passes through prism get divided into its spectrum VIBGYOR (Violet, Indigo, Blue, Green, Yellow, Orange, and Red) this phenomenon of splitting white light into its constituents colors is called Fourier transform when an infinite domain gets converted into a finite domain and when we place an inverted prism in front of prism we get back again the white light on the screen it is called Inverse Fourier Transform.*

# **Frequency Domain Filtering**

In frequency domain filtering we have the image as F (u, v). This can be achieved by performing Discrete Fourier transform f(x, y) can be converted into F (u, v) by using the formula:

*Where (x, y) are spatial domain coordinates, (u, v) are frequency domain coordinates and M \* N is image matrix size for calculation M=N.*

*Similarly F (u, v) can be converted into f(x, y) by using the formula:*



# **Low Pass Filtering**

*We implemented three types of low pass filters in this project:*

## *Rectangular Low Pass filter:*

*This is the simplest of the three filters. This filter cuts off all high frequency components of the image which are greater than a specified distance D0.*

*The equation of this filter is:*

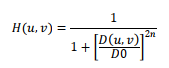


*Where D0 = cut off frequency given as input by user and D (u, v) = distance from the point (u, v) to the origin.*

## *Butterworth Low Pass filter:*

*To eliminate ringing effect, we have to remove the sharp cut offs in frequency domain. So, Butterworth filter are used they remove the ringing effect but up to a certain limit.*

*The equation of this filter is:*



*Where given as input by user and of filter and*

## *Gaussian Low Pass filter:*

*The disadvantages of Butterworth Low Pass Filter are removed in Gaussian filter i.e., no ringing effect at all. Here we do not consider the order of filter but we use ' σ ' i.e., standard deviation.*

*The equation of this filter is:*

# **Implementation**



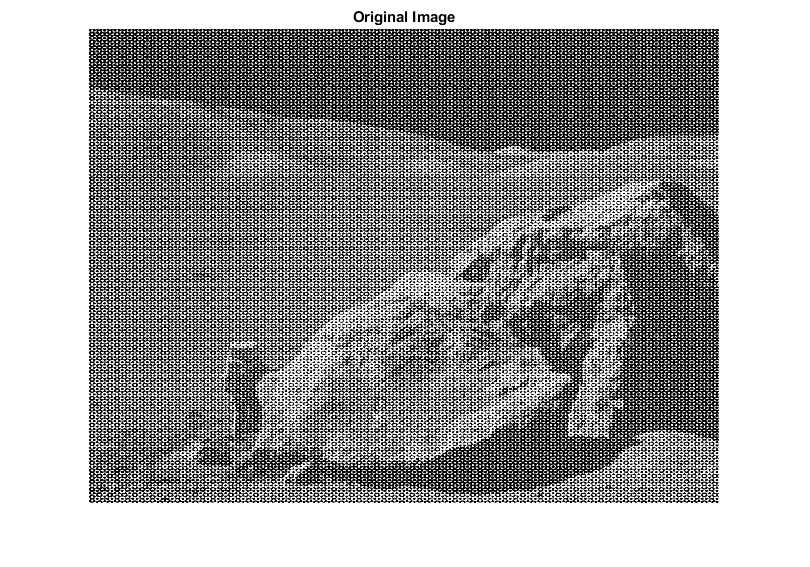
1. *Load the Image*
2. *Convert the pixel values to double data type.*
3. *Calculate the size of the image’s matrix (M x N)*
4. *Compute the Fourier Transform of the image using the Fourier’s formula.*
5. *Design the filters.*
6. *Multiply the FFT of the image with the designed filter.*
7. *Take the inverse Fourier transform of the filtered image.*
8. *Display the image.*

## *Image loading and pre-processing:*

img\_input=imread('E:\Studies\7th Semester\DSP lab\Project\Noisy\_img.tif'); %read image

imshow(img\_input); title("Original Image")

input = im2double(img\_input);



## *Fourier Transform:*

[M, N] = size(input); % Calculating Rows and Col

Wm = zeros(M, M);

Wn = zeros(N, N);

for x = 1:M-1

for y = 1:N-1

input(x, y) = input(x, y) \* (-1)^(x + y);

end

end

for k = 0:M-1

for n = 0:M-1

Wm(k+1, n+1) = exp(-1i \* pi \* 2 \* k \* n/ M);

end

end

for k = 0:N-1

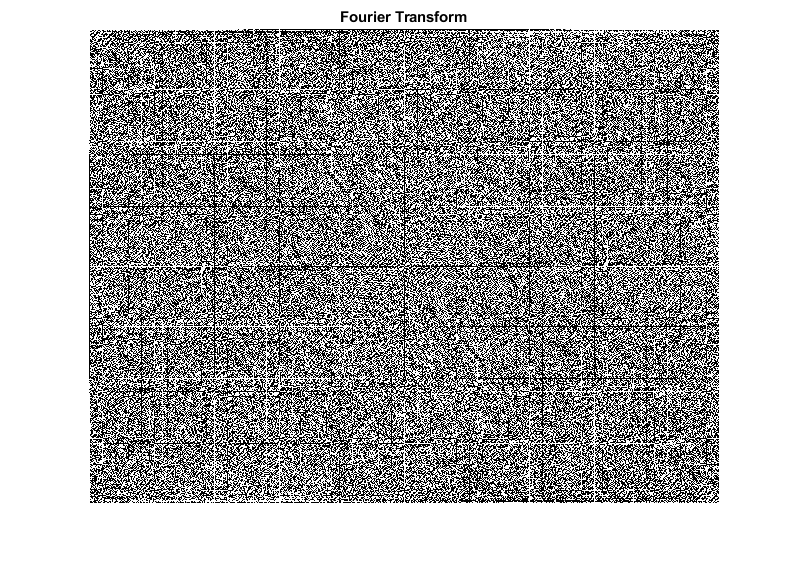
for n = 0:N-1

Wn(k+1, n+1) = exp(-1i \* pi \* 2 \* k \* n / N);

end

end

Fft = Wm \* input \* Wn;



## *Rectangular Filter Design:*

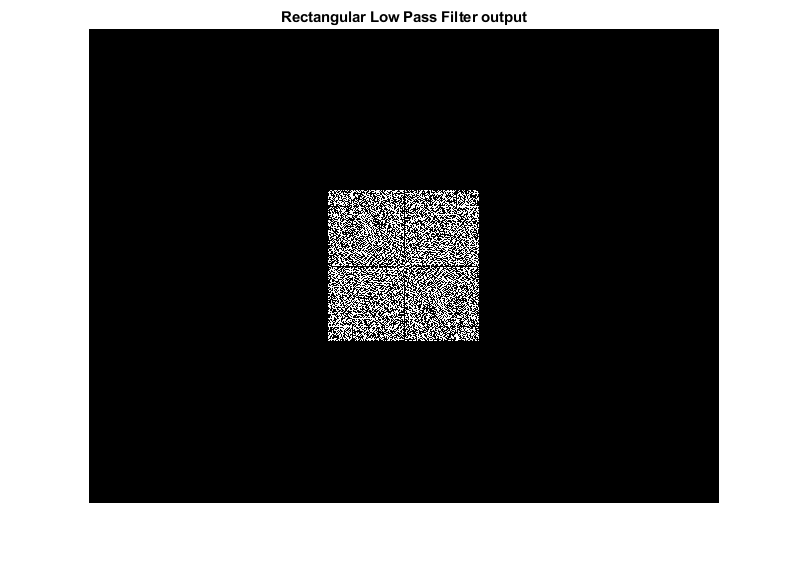
D=75; % filter size parameter

Low(1:M,1:N)=0;

Low(0.5\*M-D:0.5\*M+D,0.5\*N-D:0.5\*N+D)=1;

Filter\_out\_rectangular = Fft .\* Low;

imshow(real(Filter\_out\_rectangular)); title('Rectangular Low Pass Filter output')



## *Butterworth Filter Design:*

order = 64; %filter order

% Assign Cut-off Frequency

D0 = 75; % one can change this value accordingly

% Designing filter

u = 0:(M-1);

v = 0:(N-1);

idx = find(u > M/2);

u(idx) = u(idx) - M;

idy = find(v > N/2);

v(idy) = v(idy) - N;

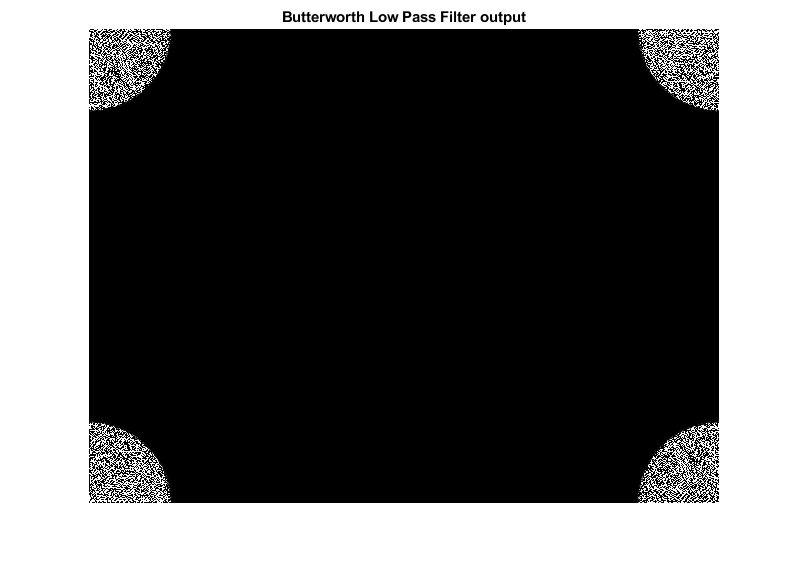
[V, U] = meshgrid(v, u);

D\_ = sqrt(U.^2 + V.^2);

H = 1./(1 + (D\_./D0).^(2\*order));

Filter\_out\_buterworth = H.\*FT\_img;

imshow(real(Filter\_out\_buterworth)); title('Rectangular Low Pass Filter output')



## *Gaussian Low Pass Filter Design:*

R=75; % filter size parameter

X=0:N-1;

Y=0:M-1;

[X, Y]=meshgrid(X,Y);

Cx=0.5\*N;

Cy=0.5\*M;

Lo=exp(-((X-Cx).^2+(Y-Cy).^2)./(2\*R).^2);

mesh(X,Y,Lo)

axis([ 0 N 0 M 0 1])

h=gca;

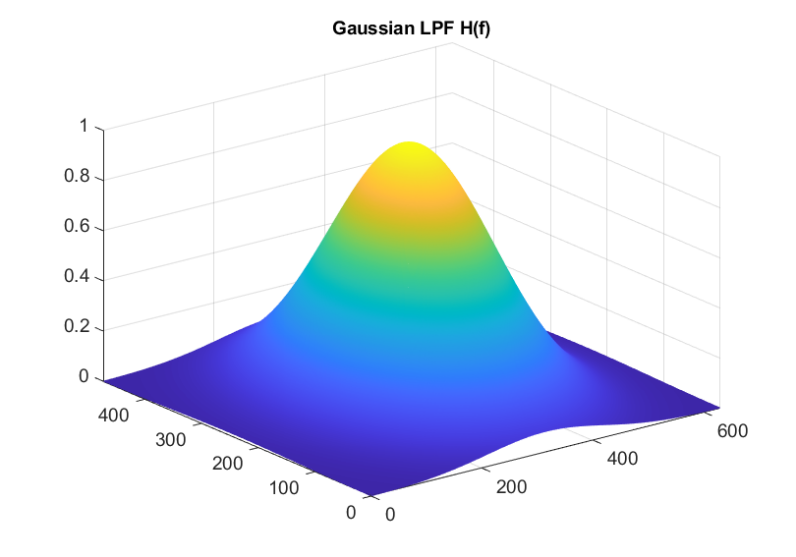
get(h,'FontSize') ;

set(h,'FontSize',14);

title('Gaussian LPF H(f)','fontsize',14);

Filter\_out\_gaussian = Lo .\* Fft;

imshow(real(Filter\_out\_gaussian)); title('Guassian Low Pass Filter output')





## *Inverse Fourier Transform:*

for k = 0:M-1

for n = 0:M-1

Wm(k+1, n+1) = exp(1i \* 2 \* pi \* k \* n/ M);

end

end

for k = 0:N-1

for n = 0:N-1

Wn(k+1, n+1) = exp(1i \* 2 \* pi \* k \* n / N);

end

end

## *Outputs:*

%%% Guassian output %%%

f\_guassian = Wm \* Filter\_out\_gaussian \* Wn/N/M;

for x = 1:M-1

for y = 1:N-1

f\_guassian(x, y) = f\_guassian(x, y) \* (-1)^(x + y);

end

end

output = im2double(abs(f\_guassian));

imshow(output); title('Output After Guassian filtering Inverse FFt')

%%% Rectangular output %%%

f\_rectangular = Wm \* Filter\_out\_rectangular \* Wn/N/M;

for x = 1:M-1

for y = 1:N-1

f\_rectangular(x, y) = f\_rectangular(x, y) \* (-1)^(x + y);

end

end

output = im2double(abs(f\_rectangular));

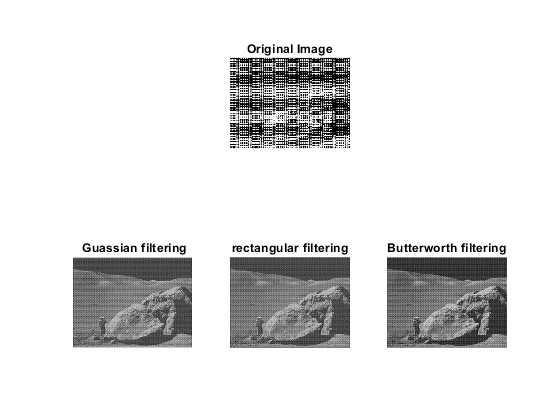
imshow(output); title('Output of rectangular filtering After Inverse FFt')

%%% Butterworth output %%%

output\_image = real(ifft2(double(Filter\_out\_buterworth)));

imshow(output\_image, []); title('Output of Butterworth filtering After Inverse FFt')

# **Results**



The above figure shows the results of all three filters. All three filters are removing the noise from the image as expected.

Lets look into each one by one:

## *Rectangular Filter:*



*The rectangular low pass filter is removing the noise from the image but the foreground of the image has become dull meaning that edges in the front of the image have become very dull. We can also see that this filter also reduces the contrast of the whole image.*

## *Gaussian Filter:*



*The gaussian filter is removing the noise as expected but is making the whole image smooth. Depending on the application, we can say that by using this filter, the information in the image is also being smoothened which may lead to loss of information. Like in the above figure, all the edges have become dull meaning we can not clearly see the information in the picture.*

## *Butterworth Filter:*



*The Butterworth filter is also successful in removing the noise from the image. It does not however smoothen the whole image to the extent that information is lost neither does it reduce the contrast of the image. But the image is still very sharp when compared to the gaussian filter*

## *Summary:*

*After implementation of all these three filters, I would say that the best results were given by the Butterworth filter followed by the Gaussian filter. If we increased the parameters of the gaussian filter, we can reduce the overall smoothness as shown:*

# **Salt and Pepper Noise Removal:**

*Salt-and-pepper noise is a form of noise sometimes seen on images. It is also known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels. To remove the Salt and Pepper noise, we use a median filter which is a non-linear filter. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighboring pixels. The pattern of neighbors is called the "window", which slides, pixel by pixel, over the entire image*

*The image we are using for this is:*

img = imread('E:\Studies\7th Semester\DSP lab\Project\Hennessey-Ford-F-150-Raptor-1b.jpg');

img\_to\_gray = rgb2gray(img);

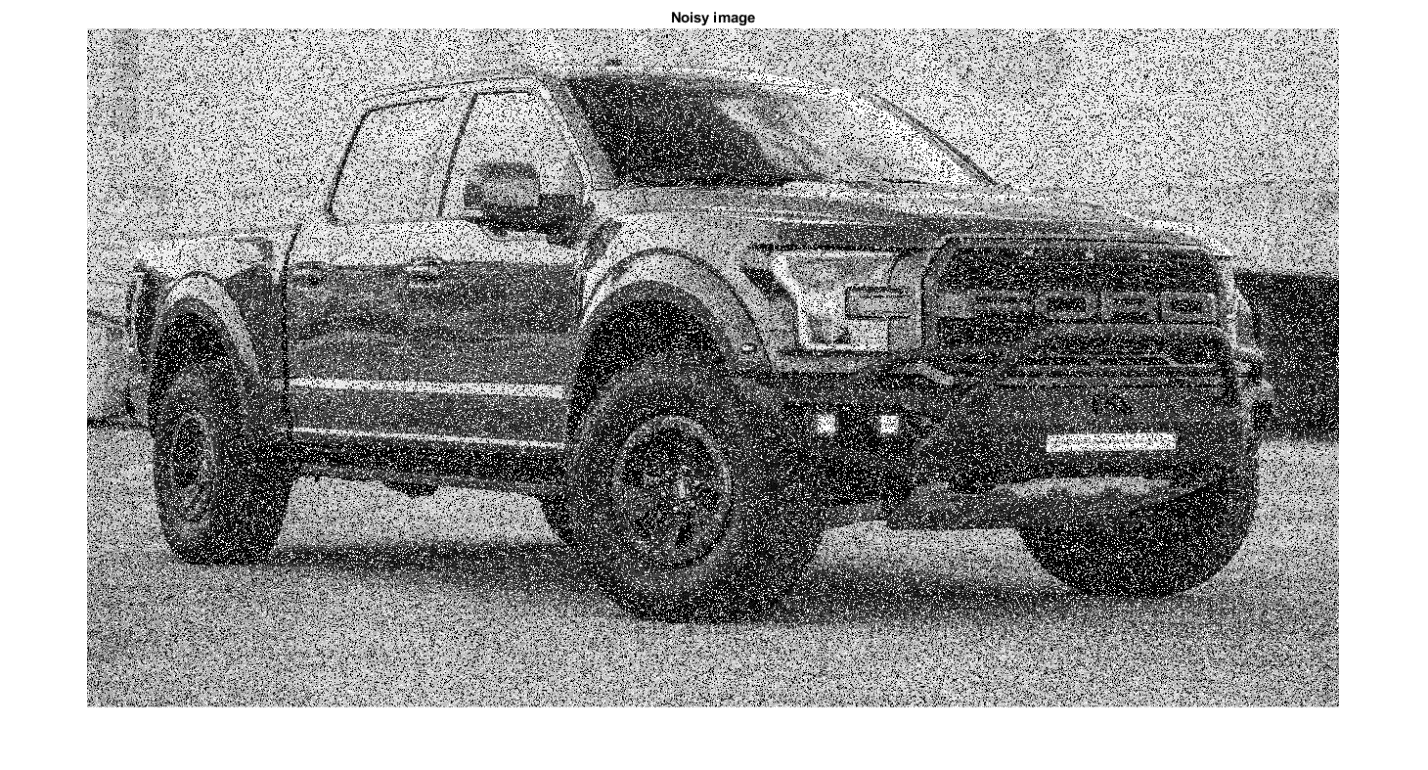
imshow(img);title('Original Image')



## *Addition of Salt and Pepper Noise:*

img\_noisy = imnoise(img\_to\_gray,'salt & pepper',0.3);

imshow(img\_noisy); title('Noisy image')

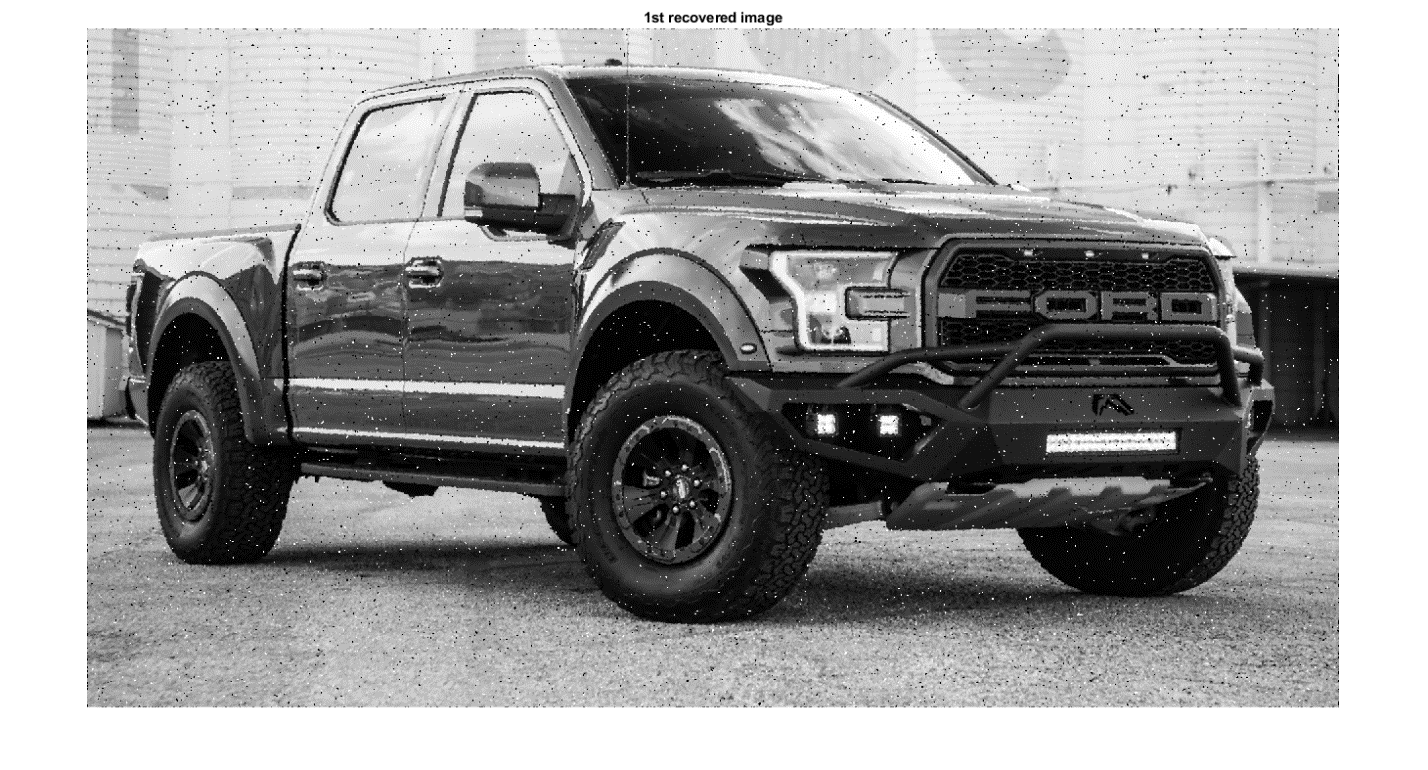


*In the above picture, we added salt and pepper noise at an intensity of 30%*

## *Removal of Salt and Pepper Noise:*

img\_recovery = medfilt2(img\_noisy);

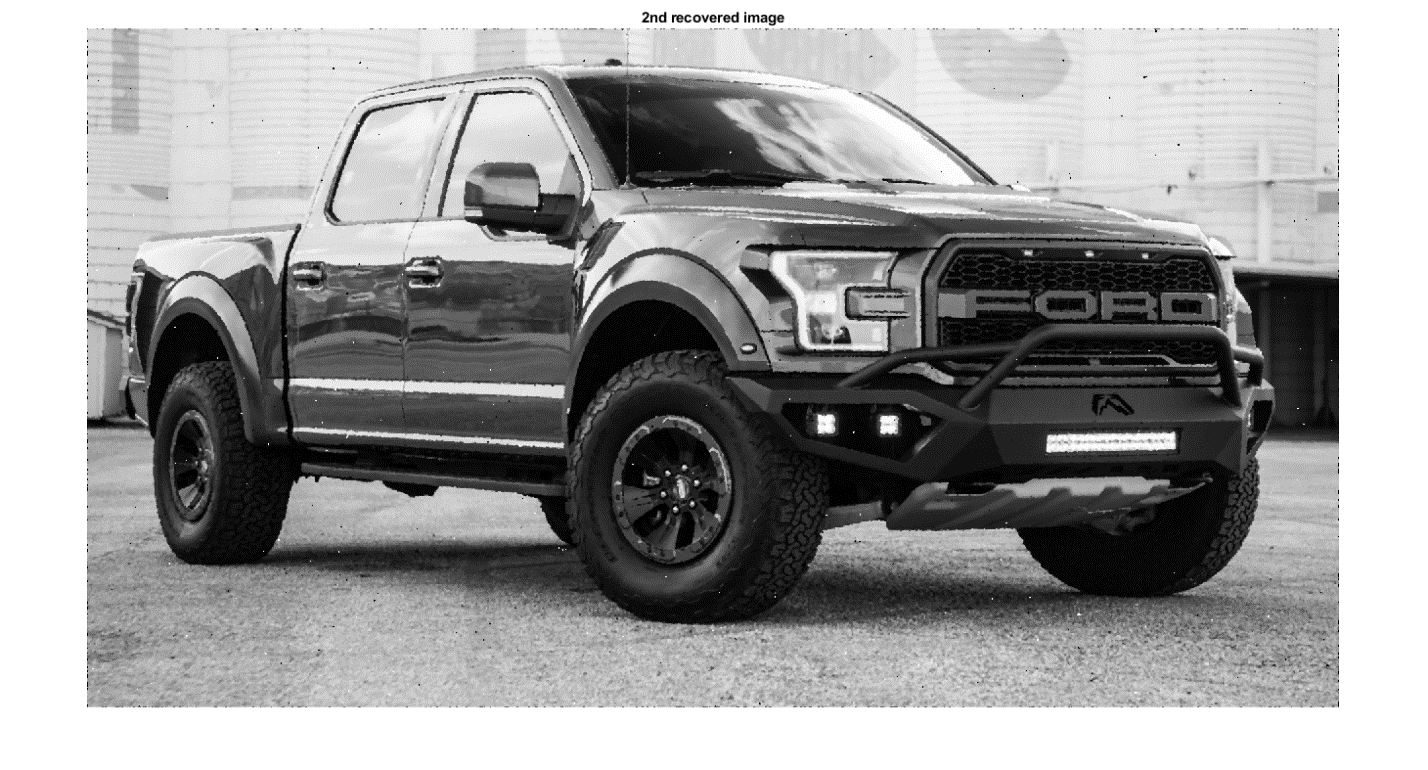
imshow(img\_recovery); title('1st recovered image')



*Applying the median filter once does not remove all of the noise added. We’ll pass this picture through another median filter*

img\_recovery\_2 = medfilt2(img\_recovery);

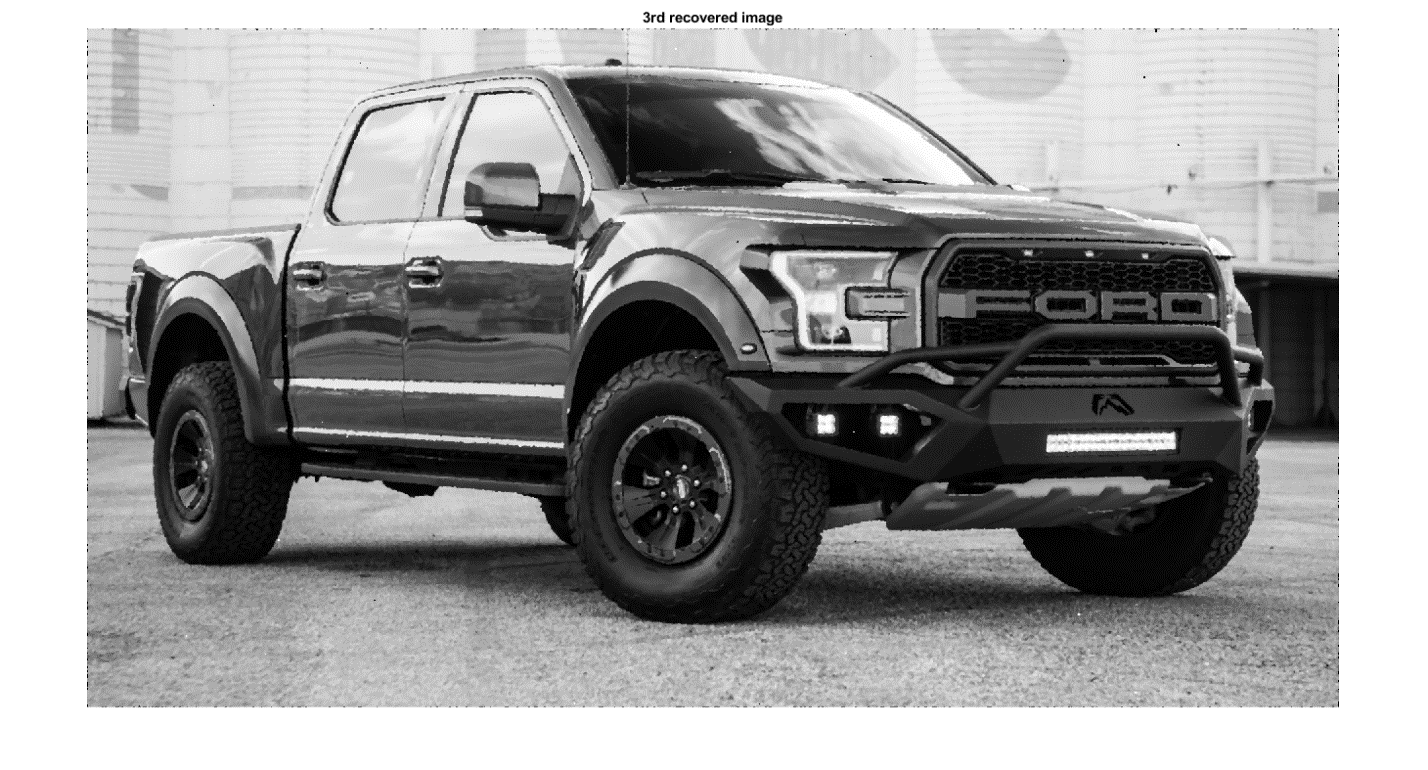
imshow(img\_recovery\_2); title('2nd recovered image')



*Applying the median filter twice also does not remove all of the noise added. Although majority of the noise has been removed but we can still see a little noise in the image. Hence, We’ll pass this picture through another median filter*

img\_recovery\_3 = medfilt2(img\_recovery\_2);

imshow(img\_recovery\_3); title('3rd recovered image')



*After the third iteration, we can see that all the noise has been removed from the picture. The output picture we get is the same as the one we input, just in gray scale because I couldn’t find any function in MATLAB which can convert a gray scale image back to RGB.*

*Above is a picture of Hennessy Ford F150 Raptor truck. Its one of the most amazing truck to ever roll out of a factory floor. The best thing about this truck is that it can withstand the BAJA 1000 rally in stock condition. But I guess this wasn’t enough, so, guys at Hennessy turned the already behemoth into something out of this world with 758 bhp @ 7,000 rpm (570 rear wheel hp). A 0-60 mph: 4.1 sec.*

# **Conclusion:**

*In this project, we looked into*

* *Digital Image processing.*
* *Implementation of FFT and IFFT without using the built-in functions provided by MATLAB.*
* *Design of various low pass filters.*
* *Effect of different filters on an image.*
* *Salt and Pepper noise*
* *Salt and Pepper noise Removal.*