****

**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

**GHATKOPAR (W), MUMBAI - 400 086**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2020 - 2021**

**M.Sc.( I.T.) SEM I**

**Image and Vision Processing**

**Name: SHREYANS UPADHYAY**

**Roll No.: 18**



**CERTIFICATE**

This is to certify that Mr. **Upadhyay Shreyans Indresh** with Roll No. **18** has successfully completed the necessary course of experiments in the subject of **Image and Vision Processing** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -I.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Internal Examiner External Examiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Head of Department College Seal

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Practical No.** | **Practical** | **Page** |
| 1. | Implement Basic Intensity functions   1. Image Inverse 2. Log Transformation 3. Power-law Transformation | 4 |
| 2. | Piecewise Transformation   1. Contrast Syretching 2. Thresholding 3. Bit-Plane Slicing | 11 |
| 3. | Implement Histogram Equalization | 22 |
| 4. | Image-filtering in Spatial Domain   1. Low-Pass Filter/Smoothing Filter   (Average, weighted-Average, Median and Gaussian)   1. High-Pass filter/Sharpening Filter   (Laplacian Filter, Sobel, Robert and Prewit filter to detect edge) | 25 |
| 5. | Analyze image in frequency Domain   1. Low Pass/Smoothing Filter 2. High Pass/Sharpening Filter |  |
| 6. | Color Image Processing   1. Pseudocoloring 2. Separating the RGB channels 3. Color Slicing | 34 |
| 7. | Image Compression Techniques and Watermarking   1. Implement Huffman Coding 2. Add a watermark to the Image | 42 |
| 8. | Basic Morphological Operations   1. Boundary Extraction 2. Thinning and Thickening 3. Hole filling and Skeletons | 46 |

**Practical No. 1**

**Implement basic Intensity transformation functions**

1. **Image Inverse**

The negative or inverse of an image with intensity levels in the range [0, L-1] is obtained by using the negative transformation, which is given by the expression,

S = L – 1 – r

Where L – 1 (Maximum pixel value)

r (Pixel of an image)

imread function is used to read an image as a matrix from the file FILENAME or from the online resource URL.

rgb2gray function is used to transform an image or colormap from red-green-blue (RGB) color space to agrayscale intensity image.

im2double is a function which is used to convert image to double precision.

imshow is a function used to display the image

**Code**

x = imread('tiger.jpeg'); #Reading the image

x = rgb2gray(x); #Converting RGB image to gray-level

x = im2double(x); #Converting image into double data type

[row col] = size(x); #taking image size into matrix form

for i = 1:row #reading row value from starting to end and storing in (i) variable

for j = 1:col #reading row value from starting to end and storing in (j) variable

N(i,j)=1-x(i,j); #subtracting input matrix values from 255 and storing in new Matrix(N)

endfor

endfor

figure

imshow(x);

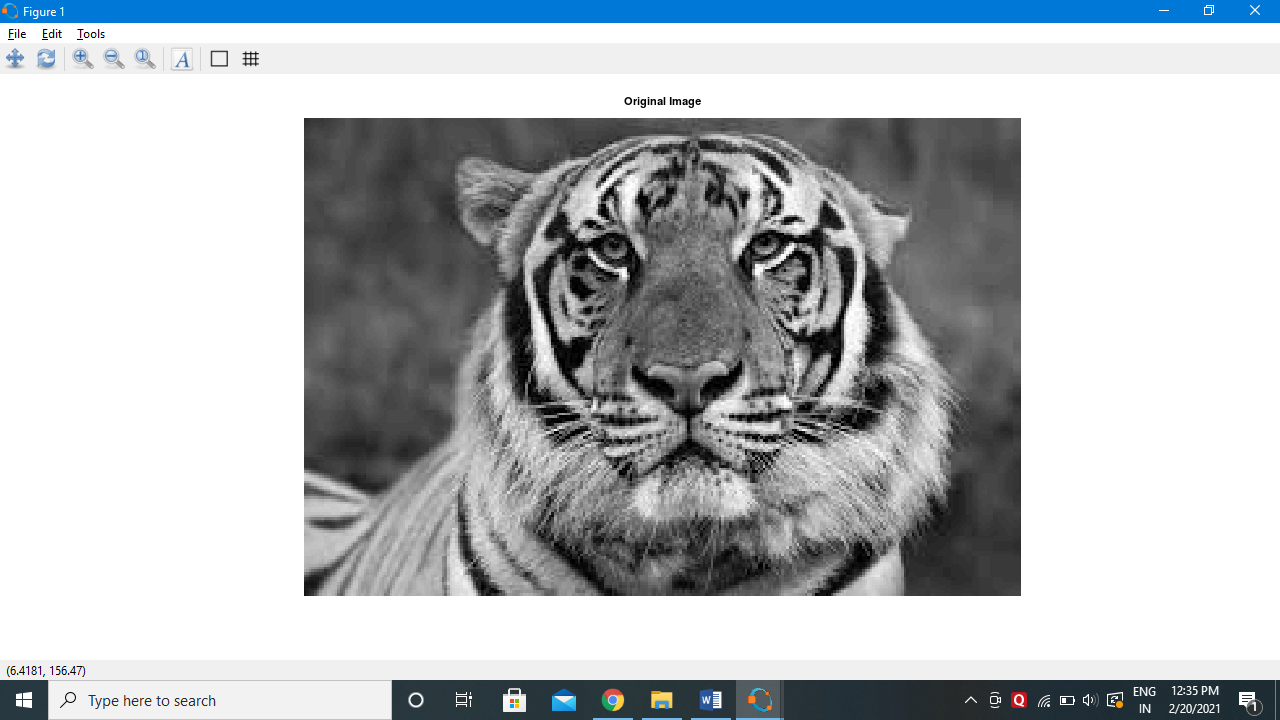
title('Original Image');

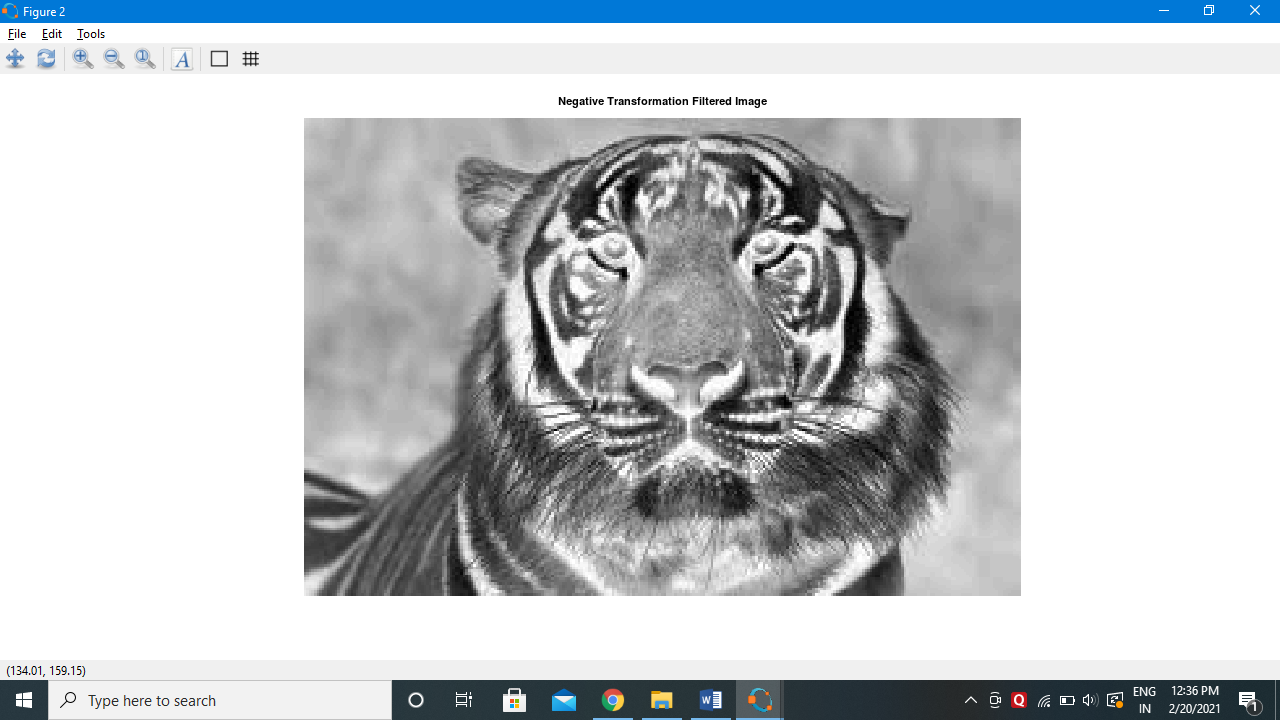
figure

imshow(N);

title('Negative Transformation Filtered Image');

**Output**

****

****

**Practical No. 1**

1. **Log Transformation**

The log transformation maps a narrow range of low intensity values in the input into a wider range of output levels. We use the transformation if this type to extend the values of dark pixel in an image while compress the higher-level values.

The general form of the log transformation is:

s = c log (r + 1)

Where c is a constant, and r ≥ 0.

**Code**

x = imread('tiger.jpeg'); #Reading the image

x = rgb2gray(x); #Converting RGB image to gray-level image

x = im2double(x); #Converting image into double data type

[row col] = size(x); #taking image size into matrix form

c=2; #here we are taking constant value into c variable

for i = 1:row #reading row value from starting to end and storing in (i) variable

for j = 1:col #reading row value from starting to end and storing in (j) variable

N(i,j)=c\*log(1+x(i,j)); #Here we are doing log calculation and storing the value into N

endfor

endfor

figure

imshow(x);

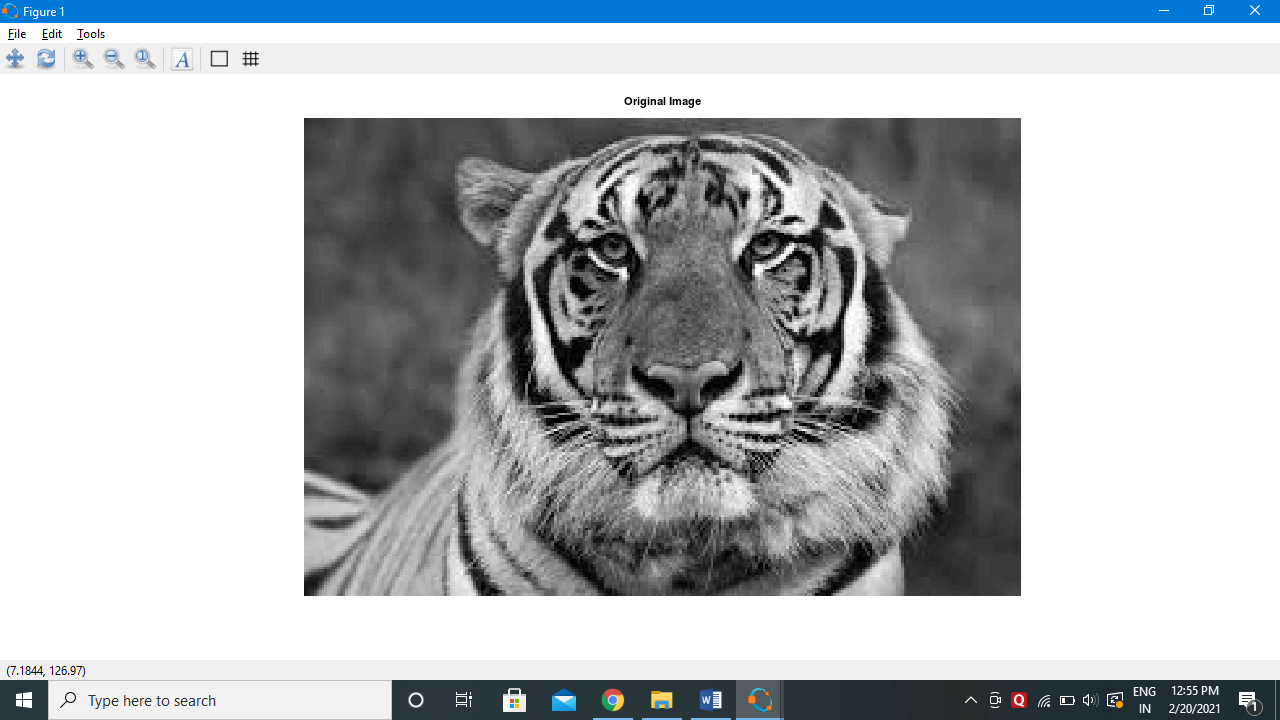
title('Original Image');

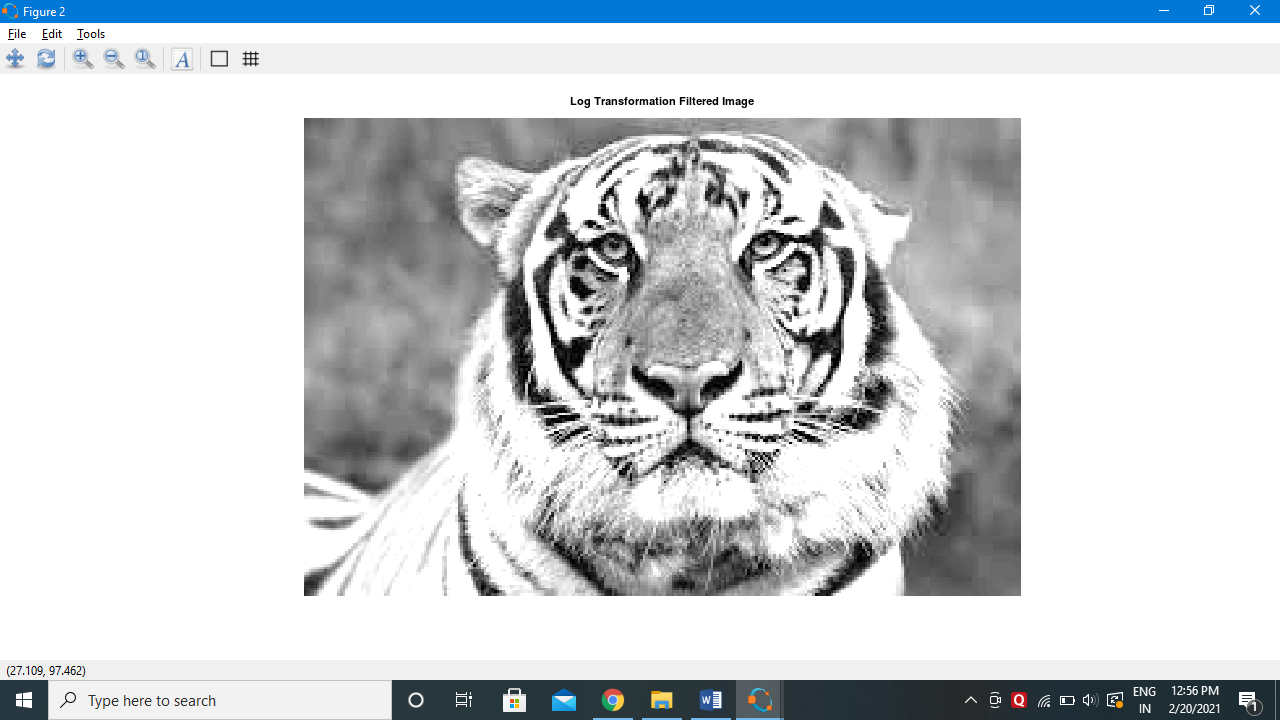
figure

imshow(N);

title('Log Transformation Filtered Image');

**Output**

****

****

**Practical No. 1**

1. **Power Law Transformation**

Power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels.

The nth power and nth root curves shown in below figure can be given by the expression as s = c r γ, This transformation function is also called as gamma correction. For various values of γ different levels of enhancements can be obtained. It is used to correct power law response phenomena. The different display monitors display images at different intensities and clarity. That means, every monitor has built-in gamma correction in it with certain gamma ranges and so a good monitor automatically corrects all the images displayed on it for the best contrast to give user the best experience. The gamma variation changes ratio of red green & blue along with intensity in color images. The difference between the log-transformation function and the power-law functions is that using the power-law function a family of possible transformation curves can be obtained just by varying the λ. This process is also called a gamma correction.

The Power Low Transformations can be given by the expression:

 s = c \* r ^ γ where, s is the output pixels value r is the input pixel value c and γ are the real numbers

**Code**

x=imread('irbd.jpeg'); #reading the image

x = rgb2gray(x); #Converting RGB image to gray-level image

x = im2double(x); #Converting image into double data type

[row col] = size(x); #taking image size into matrix form

gamma=2; #here we are taking constant value into “gamma” variable

c=1; #here we are taking constant value into “c” variable

for i = 1:row #reading row value from starting to end and storing in (i) variable

for j = 1:col #reading row value from starting to end and storing in (j) variable

N(i,j)=c\*(x(i,j)^gamma); #Here we are doing power-law calculation and storing there values into N matrix

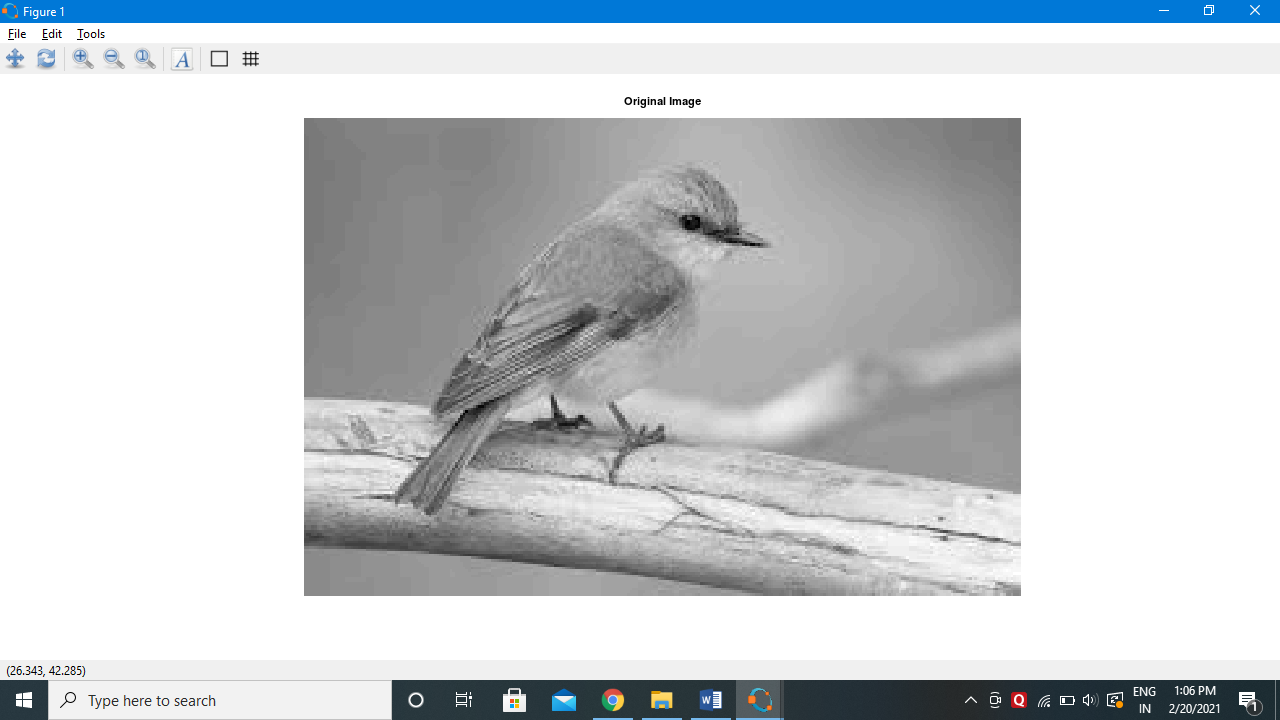
endfor

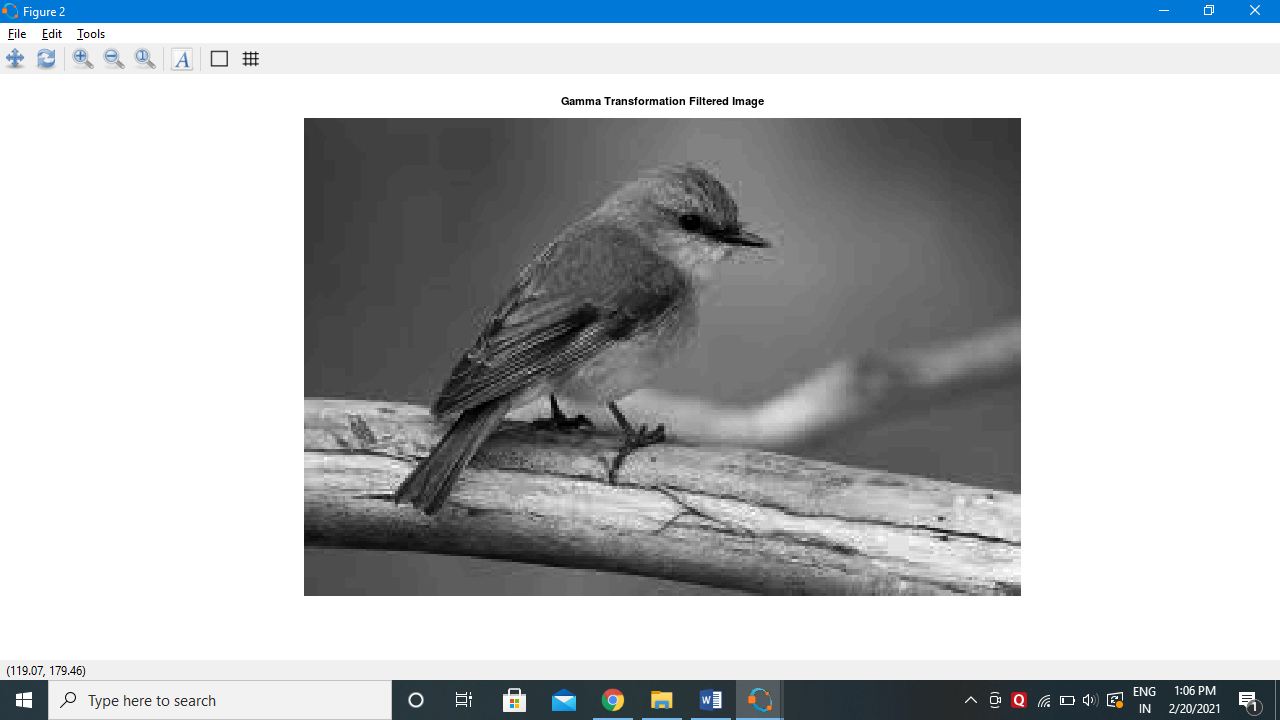
endfor

figure, imshow(x); title('Original Image');

figure, imshow(N); title('Gamma Transformation Filtered Image');

**Output**





**Practical No. 2**

**Piecewise Transformation**

1. **Contrast Stretching**

Contrast can be simply explained as the difference between maximum and minimum pixel intensity in an image.

Consider an image matrix A=[100,100,100,100,100; 100,100,100,100,100; 100,100,100,100,100; 100,100,100,100,100; 100,100,100,100,100;]

The maximum value in this matrix is 100 and the minimum value is also 100.

Contrast = maximum pixel intensity(subtracted by) minimum pixel intensity

100-100=0, so there is no contrast

Code

pkg load image;

clear all;

r = imread("fields.jpg");

#r=rgb2gray(r);

r = im2double(r);

[m n] = size(r); % Getting the dimensions of the image.

#here we are taking 4 input from user

r1=input("Enter R1: ");

r2=input("Enter R2: ");

s1=input("Enter S1: ");

s2=input("Enter S2: ");

#Calculation of contrast stretching

a = s1/r1;

b = (s2-s1)/(r2-r1);

c = (255-s2)/(255-r2);

for i=1:m

for j=1:n

if r(i,j) < r1

s(i,j) = a\*r(i,j);

elseif r(i,j) < r2

s(i,j) = b\*(r(i,j)-r1)+s1;

else

s(i,j) = c\*(r(i,j)-r2)+s2;

endif

endfor

endfor

#Displaying the Original and Contrast Images

figure(3);

subplot(1,2,1)

imshow(r);

title("Original Image");

subplot(1,2,2)

imhist(r);

title('Histogtram Of Original Image');

figure(4);

subplot(1,2,1)

imshow(s);

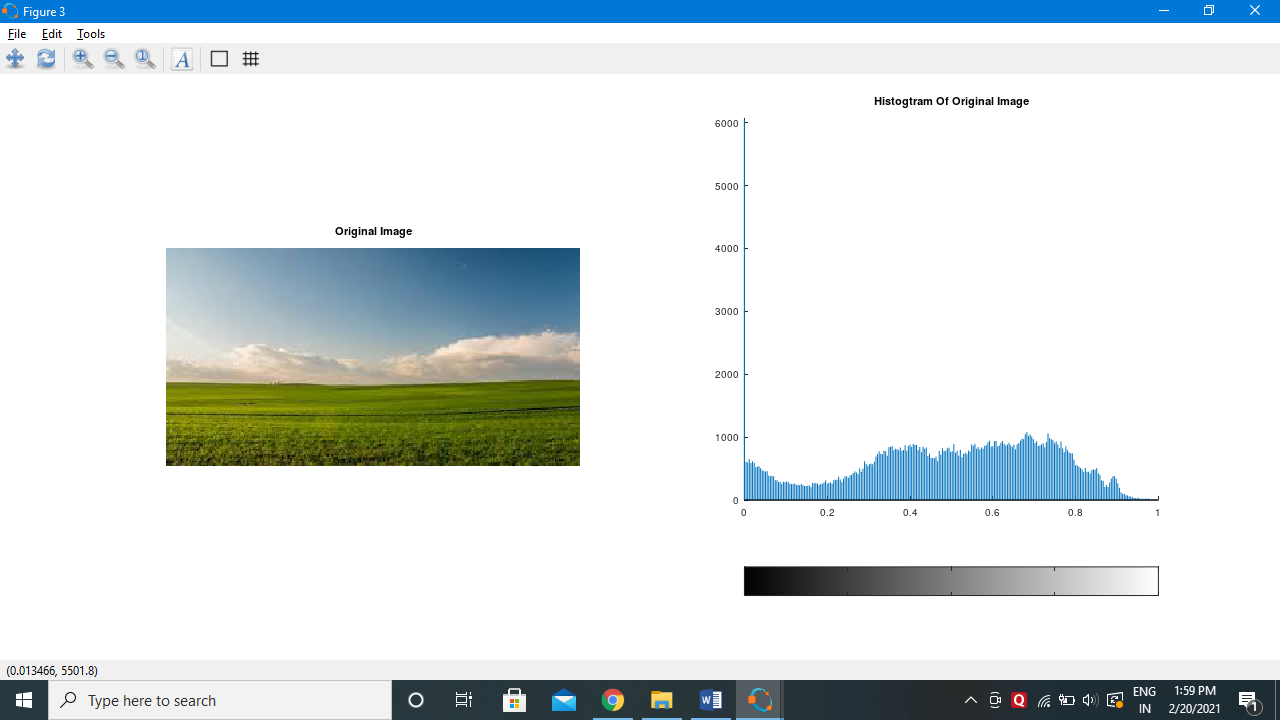
title("Contrast Streched Image");

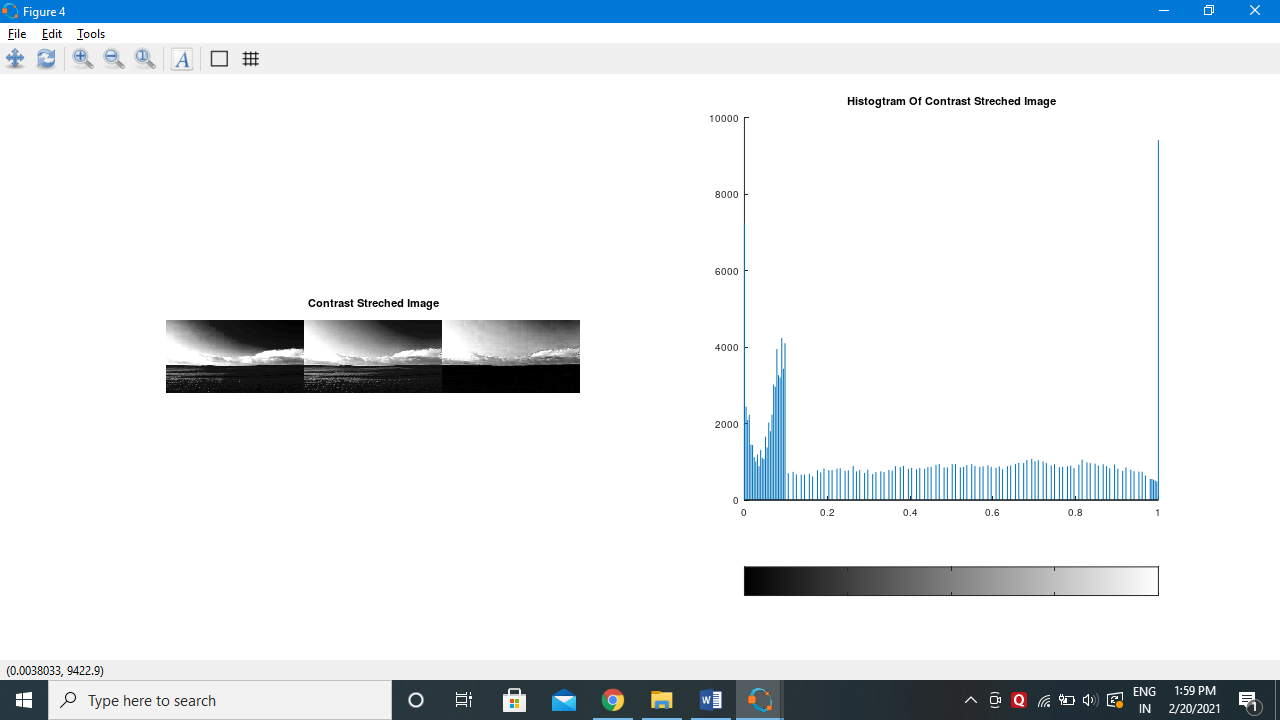
subplot(1,2,2)

imhist(s);

title('Histogtram Of Contrast Streched Image');

**Output**

****

****

1. **Thresholding**

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast.

Code

pkg load image;

clear all;

r=imread("field.jpeg");

#r=rgb2gray(r);

#r=im2double(r);

imhist(r);

thr=150;

[m n]=size(r);

s=zeros(m,n);

for i=1:m

for j=1:n

if(r(i,j))>thr

s(i,j)=1;

else

s(i,j)=0;

endif

endfor

endfor

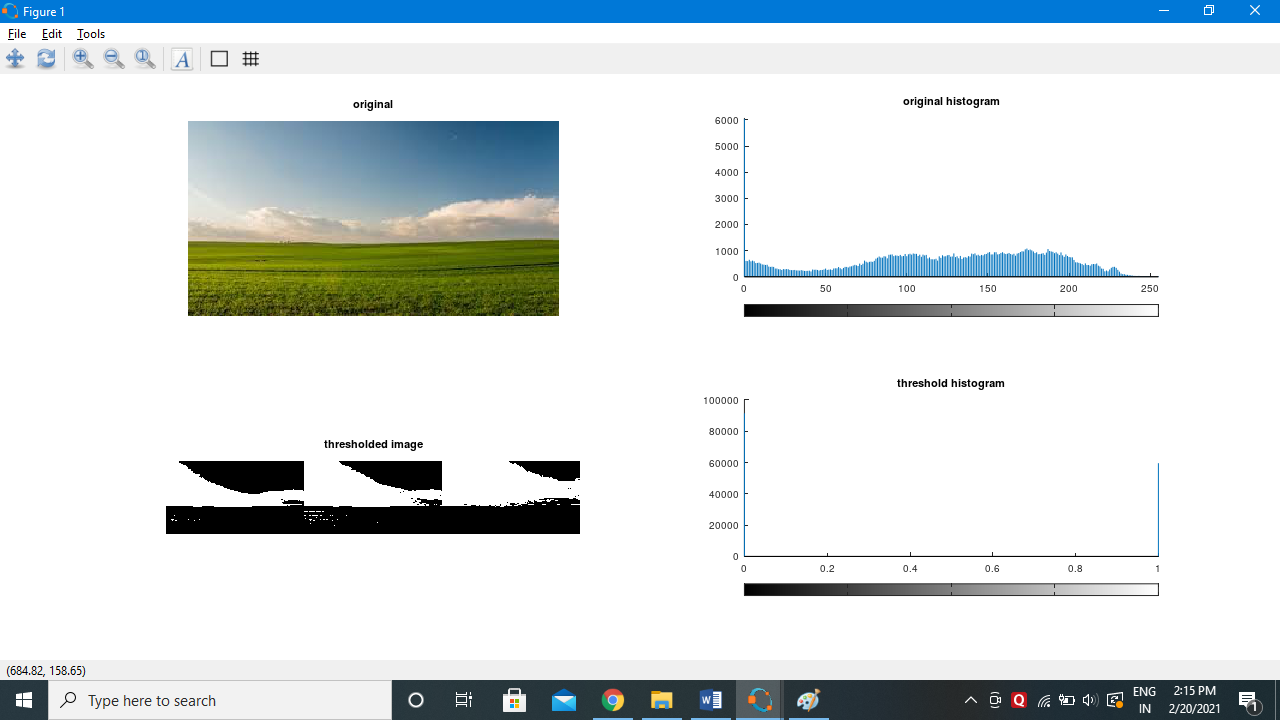
subplot(2,2,1); imshow(r); title("original");

subplot(2,2,2); imhist(r); title("original histogram");

subplot(2,2,3); imshow(s); title("thresholded image");

subplot(2,2,4); imhist(s); title("threshold histogram");

**Output**

****

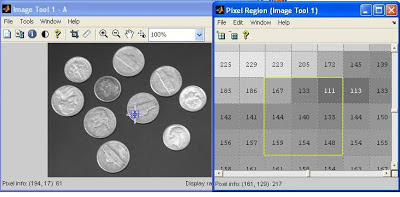
1. **Image Reconstruction using n-bit Planes (Bit-Plane Slicing)**

Bit-Plane Slicing

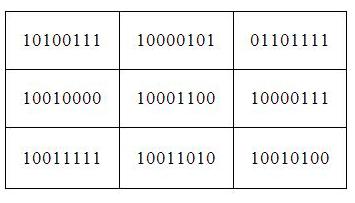
 Digitally, an image is represented in terms of pixels.

These pixels can be expressed further in terms of bits.

Consider the image ‘coins.png’ and the pixel representation of the image.

[](https://1.bp.blogspot.com/-Errw8_KJu3o/UJejlDf5vAI/AAAAAAAAAus/0qk0rXtHqKk/s1600/bit_plane_slicing_1.JPG)

Consider the pixels that are bounded within the yellow line. The binary formats for those values are (8-bit representation)

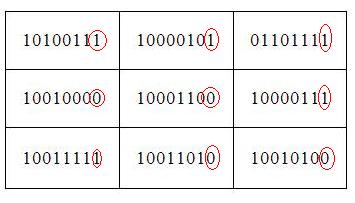
[](https://1.bp.blogspot.com/-3QQ8iTVPL2U/UJejrwXL6xI/AAAAAAAAAvU/s5MSiCXixz8/s1600/bit_representation.JPG)

The binary format for the pixel value 167 is 10100111

Similarly, for 144 it is 10010000

This 8-bit image is composed of eight 1-bit planes.

Plane 1 contains the lowest order bit of all the pixels in the image.

[](https://4.bp.blogspot.com/-kGSyLtTblqo/UJejtiU9xCI/AAAAAAAAAvk/0neXMgDSKWE/s1600/bit_representation_low_order.JPG)

And plane 8 contains the highest order bit of all the pixels in the image.



Let’s see how we can do this using MATLAB

A=[167 133 111

      144 140 135

      159 154 148]

B=bitget(A,1);  %Lowest order bit of all pixels

‘bitget’ is a MATLAB function used to fetch  a bit from the specified position from all the pixels.

B=[1 1 1

      0 0 1

      1 0 0]

B=bitget(A,8);%Highest order bit of all pixels

B=[1 1 0

      1 1 1

      1 1 1]

Code

A=imread('dollar.jpg');

g=rgb2gray(A);

B=zeros(size(g));

#Getting the bit at specified position#

g1 = bitget(g,1);

g2 = bitget(g,2);

g3 = bitget(g,3);

g4 = bitget(g,4);

g5 = bitget(g,5);

g6 = bitget(g,6);

g7 = bitget(g,7);

g8 = bitget(g,8);

figure,

subplot(2,2,1)

imshow(logical(g1));

title('Bit 1');

subplot(2,2,2)

imshow(logical(g2));

title("Bit 2");

subplot(2,2,3)

imshow(logical(g3));

title('Bit 3');

subplot(2,2,4)

imshow(logical(g4));

title('Bit 4');

figure,

subplot(2,2,1)

imshow(logical(g5));

title('Bit 5');

subplot(2,2,2)

imshow(logical(g6));

title("Bit 6");

subplot(2,2,3)

imshow(logical(g7));

title('Bit 7');

subplot(2,2,4)

imshow(logical(g8));

title('Bit 8');

#B=bitset(B,4,bitget(A,4));

B=bitset(B,5,g5);

B=bitset(B,6,g6);

B=bitset(B,7,g7);

B=bitset(B,8,g8);

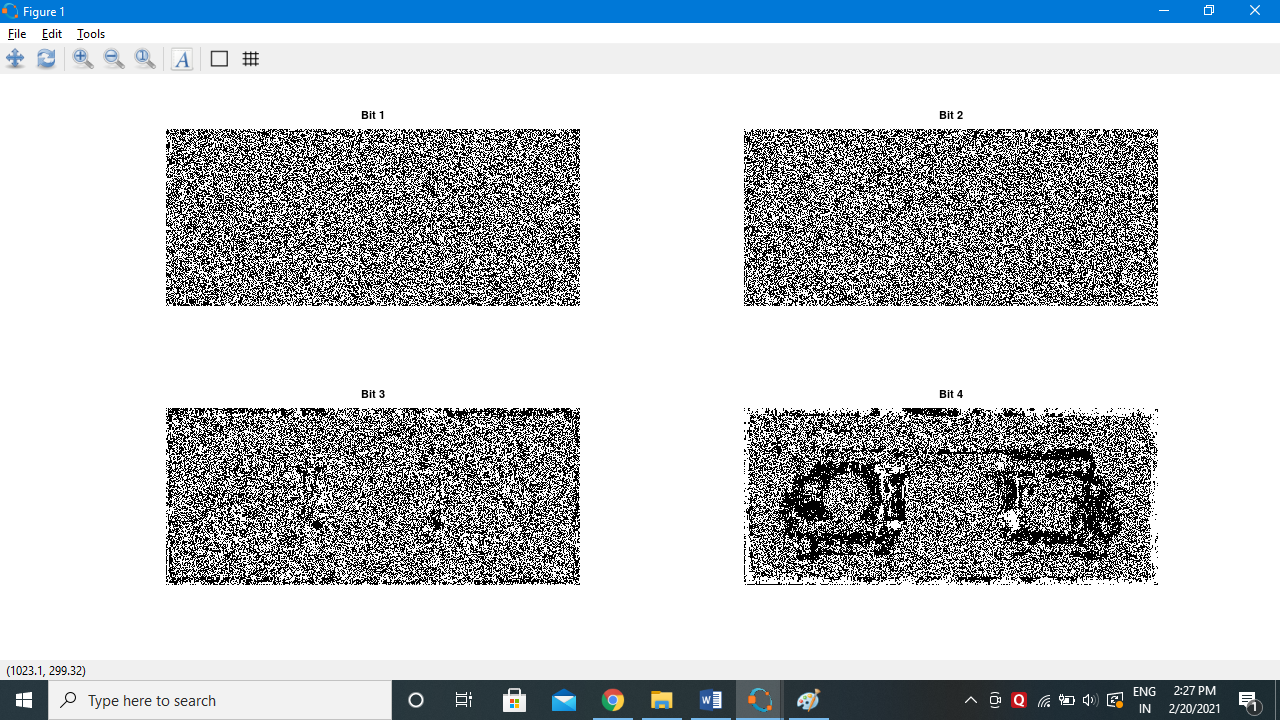
B=uint8(B);

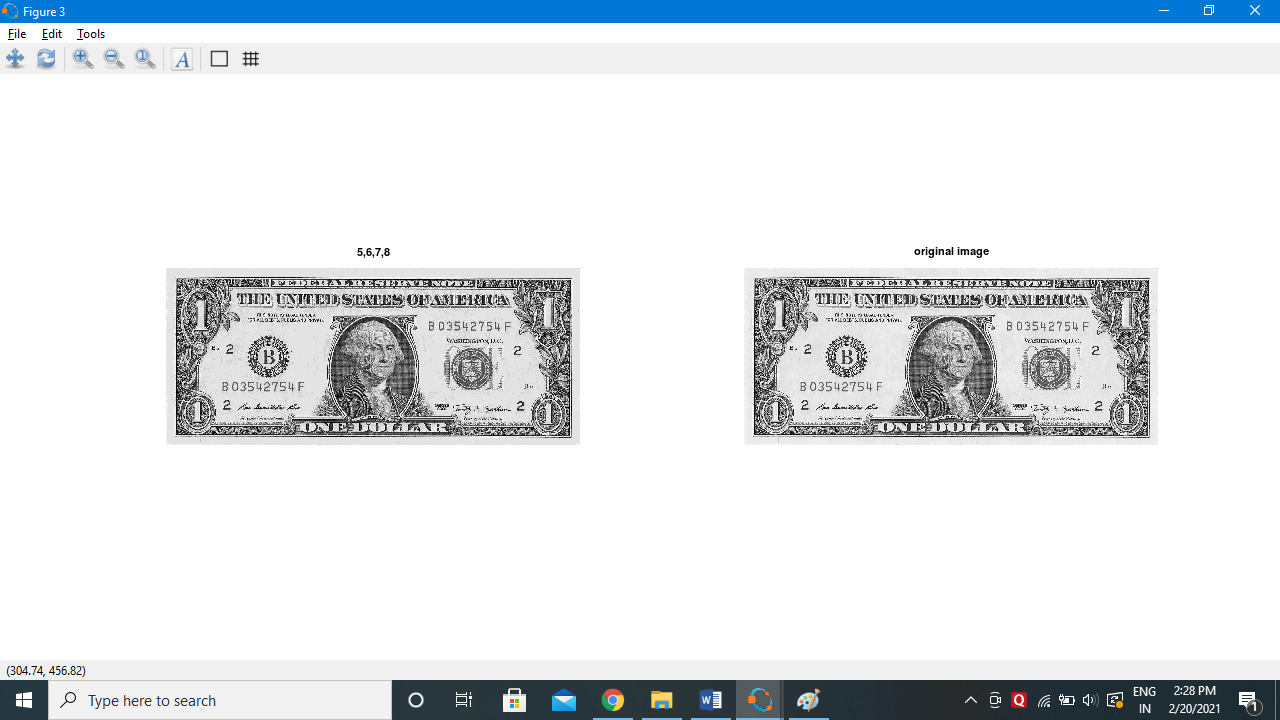
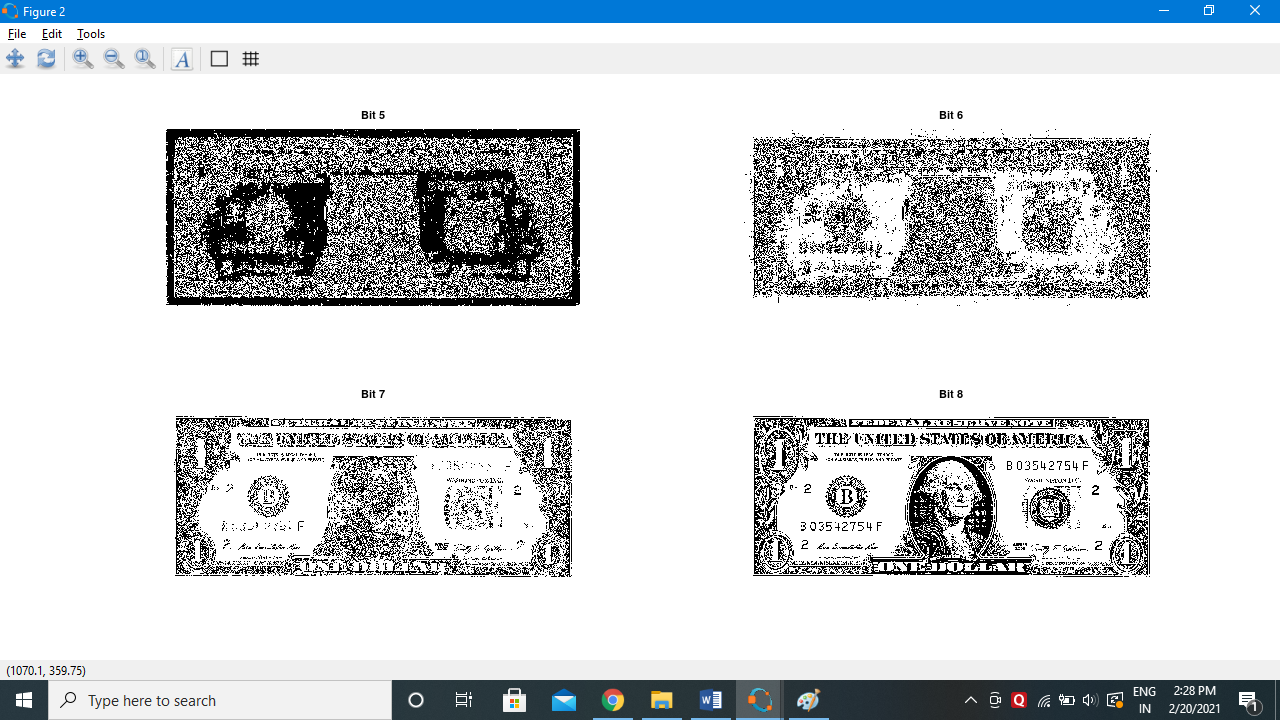
figure,

subplot(1,2,1),imshow(B); title("5,6,7,8")

subplot(1,2,2),imshow(g); title("original image");

**Output**

****

****

**Practical No 3**

**Implement Histogram Equalization**

The following steps are performed to obtain histogram equalization

1. Find the frequency of each pixel value

Consider a matrix A=[1,4,2;5,1,3;1,2,4] with number of bins=5

The pixel value 1 occurs 3 times. Similarly the pixel value 2 occurs 2 times and so on.

1. Find the probability of each frequency.

The probability of pixel value 1’s occurrence=frequency(1) no of pixels i.e. 3/9

1. Find the cumulative histogram of each pixel.

The cumulative histogram of 1=3

Cumulative histogram of 2= cumulative histogram of 1+frequency of 2=5

Cumulative histogram of 3=cumulative histogram of 2+frequency of 3=5+1=6.

1. Find the cumulative distribution probability of each pixel.

cdf(1)=cumulative histogram(1)/no. of pixels=3/9.

1. Calculate the final value of each pixel by multiplying cdf with no. of bins;

Cdf(1)=(3/9)\*(5)=1.6667. Round off the value.

1. Now replace the final values: [**2**,4,3; 5,**2**,3; **2**,3,4]

The final value for the bin is 2. It is placed in the place of 1 in the matrix.

Code

clear all;

close all;

pkg load image;

a=imread('field.jpeg');

#a=rgb2gray(a);

#a=a(1:10,1:10)

r=size(a,1);

c=size(a,2);

ah=uint8(zeros(r,c));

n=r\*c;

f=zeros(256,1);

pdf=zeros(256,1);

cdf=zeros(256,1);

cumm=zeros(256,1);

out=zeros(256,1);

for i=1:r

for j=1:c

values=a(i,j);

f(values+1)=f(values+1)+1;

pdf(values+1)=f(values+1)/n;

endfor

endfor

sum=0; L=255; size(pdf);

for i=1:size(pdf)

sum=sum+f(i);

cum(i)=sum;

cdf(i)=cum(i)/n;

out(i)=round(cdf(i)\*L);

endfor

for i=1:r

for j=1:c

ah(i,j)=out(a(i,j)+1);

endfor

endfor

figure,

subplot(2,2,1), imshow(a); title('original image');

subplot(2,2,2), imhist(a); title('original hist');

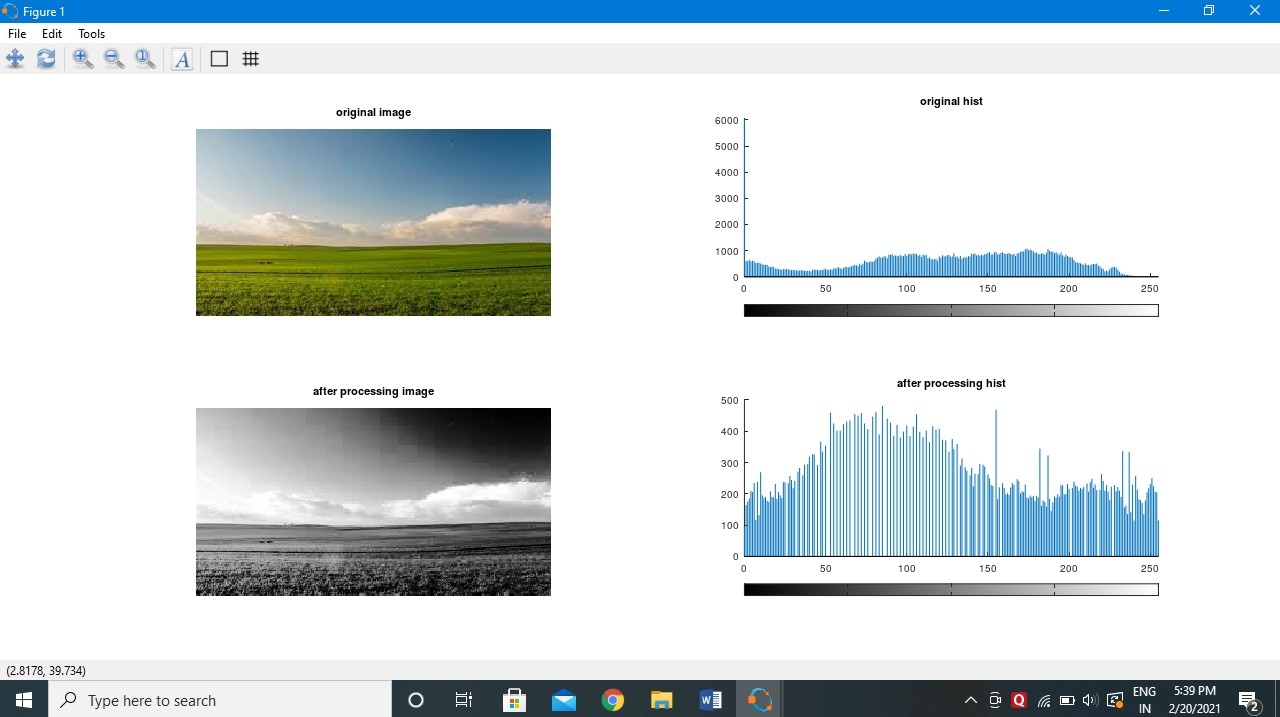
#he=histeq(a);

subplot(2,2,3), imshow(ah); title('after processing image');

subplot(2,2,4), imhist(ah); title('after processing hist');

#imhist(he);

**Output**

****

**Practical No. 4**

**Image Filtering in Spatial Domain**

1. **Low Pass Average Filter**

Code

close all;

pkg load image;

a=imread('hawk.jpeg');

Nim=imnoise(im,'salt & pepper');

w=(1/16)\*[1,2,1;2,4,1;1,2,1];

[ma,na]=size(Nim);

[mb,nb]=size(w);

c=zeros(ma+mb-1,na+nb-1);

size\_c=size(c);

for i=1:mb

for j=1:nb

r1=i

r2=r1+ma-1

c1=j

c2=c1+na-1

c(r1:r2,c1:c2)=c(r1:r2,c1:c2)+w(i,j)\*(Nim);

end

end

r1=floor(mb/2)+1;

r2=r1+ma-1;

c1=floor(nb/2)+1;

c2=c1+na-1;

c=c(r1:r2,c1:c2);

figure

subplot(1,2,1);

imshow(Nim);

title("Noisy Image (Salt & Pepper)");

subplot(1,2,2);

imshow(uint8(c));

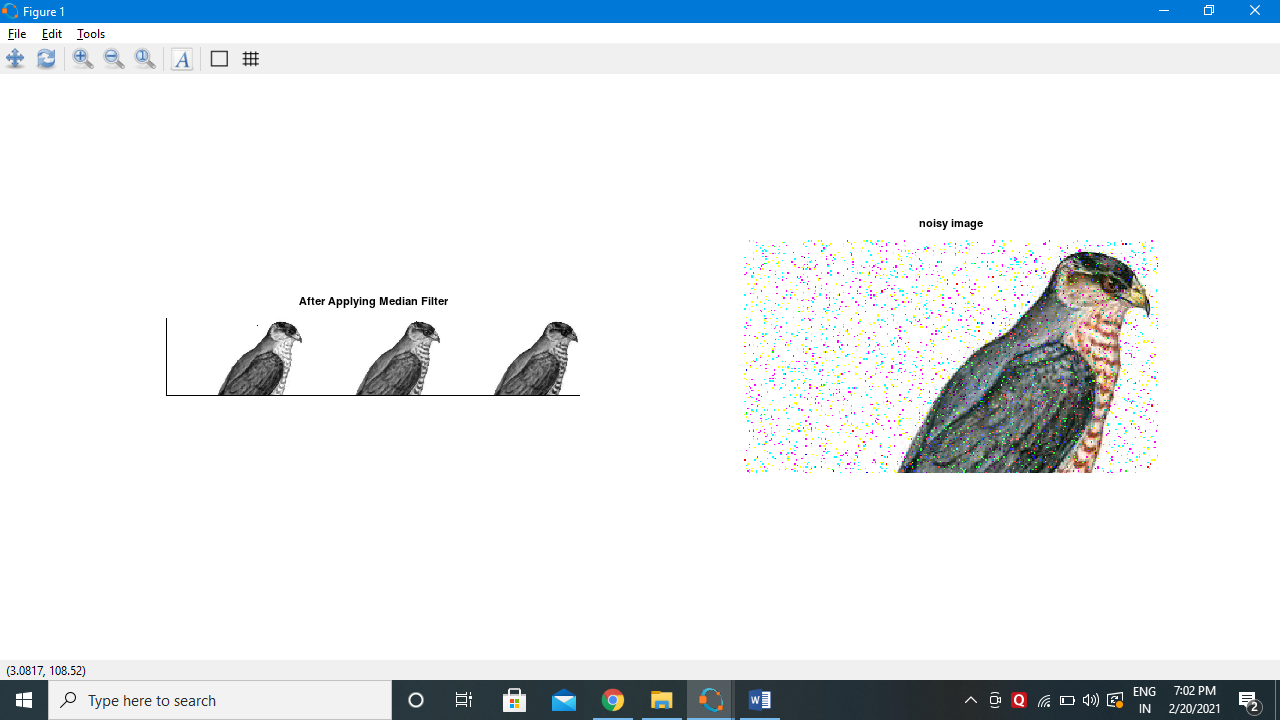
title('Denoised Image using Weighted Average Operation of Box Filter');

**Low Pass Filter (Median Filter)**

**Code**

pkg load image; # Read the image a=imread('hawk.jpeg'); img\_noisy1=imnoise(a,'salt & pepper' ); # Obtain the number of rows and columns of the image [m, n] = size(img\_noisy1) # Traverse the image. For every 3X3 area, # find the median of the pixels and # replace the center pixel by the median img\_new1 = zeros(m, n); for i=2: m-1 for j =2: n-1 temp = [img\_noisy1(i-1, j-1), img\_noisy1(i-1, j), img\_noisy1(i-1, j + 1), img\_noisy1(i, j-1), img\_noisy1(i, j), img\_noisy1(i, j + 1), img\_noisy1(i + 1, j-1), img\_noisy1(i + 1, j), img\_noisy1(i + 1, j + 1)] ; temp = sort(temp); img\_new1(i, j)= temp(4); endfor endforimg\_new1 = uint8(img\_new1);figuresubplot(1,2,1); imshow(img\_new1); title('After Applying Median Filter');subplot(1,2,2); imshow(img\_noisy1);title('noisy image');

**Output**



**Second Order Derivative-The Laplacian filter**

**Code**

clear all;

A=imread('coin.png');

size(A);

figure,

subplot(2,2,1);imshow(A); title('original Image');

%Preallocate the matrices with zeros

I1=A;

I=zeros(size(A));

I2=zeros(size(A));

%Filter Masks

F1=[0 2 0;2 -8 2; 0 2 0];

#F2=[1 1 1;1 -8 1; 1 1 1];

%Padarray with zeros

A=padarray(A,[1,1]);

A=double(A);

size(A);

%Implementation of the equation in Fig.D

for i=1:size(A,1)-2

for j=1:size(A,2)-2

I(i,j)=sum(sum(F1.\*A(i:i+2,j:j+2)));

end

end

I=uint8(I);

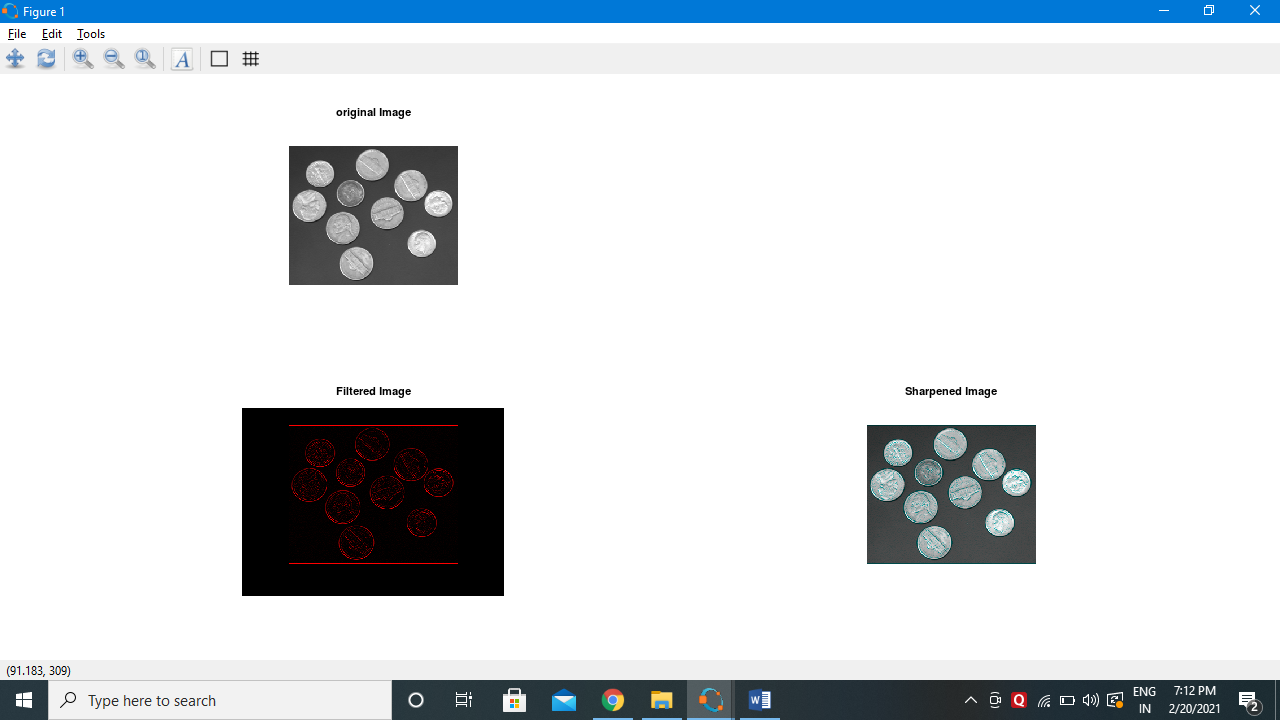
subplot(2,2,3);imshow(I);title('Filtered Image');

%Sharpenend Image

B=I1-I;

subplot(2,2,4); imshow(B);title('Sharpened Image');

**Output**

****

1. **High Pass Filter/Sharpening Filter**

clear all;

A=imread('peppers.png');

figure,

subplot(1,2,1);

imshow(A); title('Original');

C=double(A);

size(C)

for i=1:size(C,1)-2

for j=1:size(C,2)-2

%Sobel mask for x-direction:

Gx=((C(i+2,j)+2\*C(i+2,j+1)+C(i+2,j+2))-(C(i,j)+2\*C(i,j+1)+C(i,j+2)));

%Sobel mask for y-direction:

Gy=((C(i,j+2)+2\*C(i+1,j+2)+C(i+2,j+2))-(C(i,j)+2\*C(i+1,j)+C(i+2,j)));

%The gradient of the image

# B(i,j)=abs(Gx)+abs(Gy);

A(i,j)=sqrt(Gx.^2+Gy.^2);

end

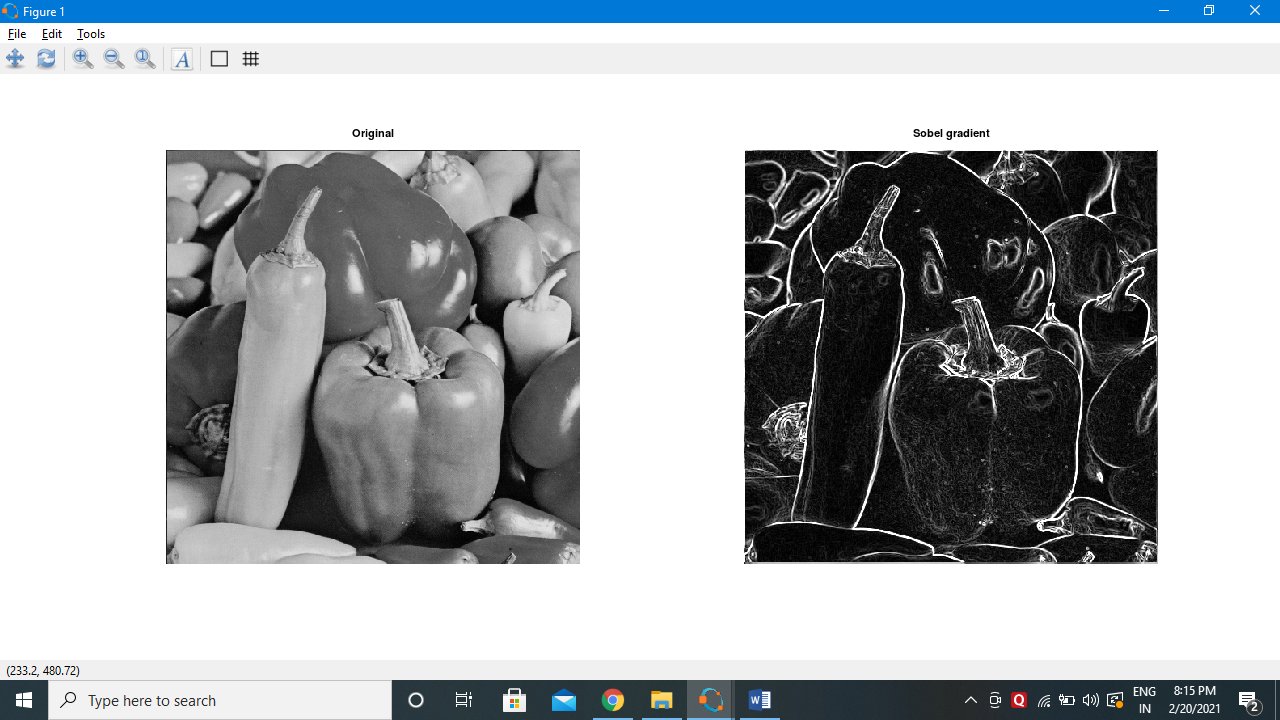
end

subplot(1,2,2);

imshow(A);

title('Sobel gradient');

**Output**

****

**First Order Derivative -Sobel Operator for edge detection**

**Code**

#load package of image

pkg load image;

#Take input image

img=imread("peppers.png");

#function to find edge using sobel filter

sobel = edge(img,'Sobel');

figure 1,

subplot(1,2,1)

imshow(img);

title('Original Image');

subplot(1,2,2)

imshow(sobel);

title("Edge detection using sobel filter");

#function to find edge using sobel filter

robert = edge(img,'Roberts');

prewitt = edge(img,'Prewitt');

figure 2,

subplot(1,2,1)

imshow(robert);

title('Edge detection using robert filter');

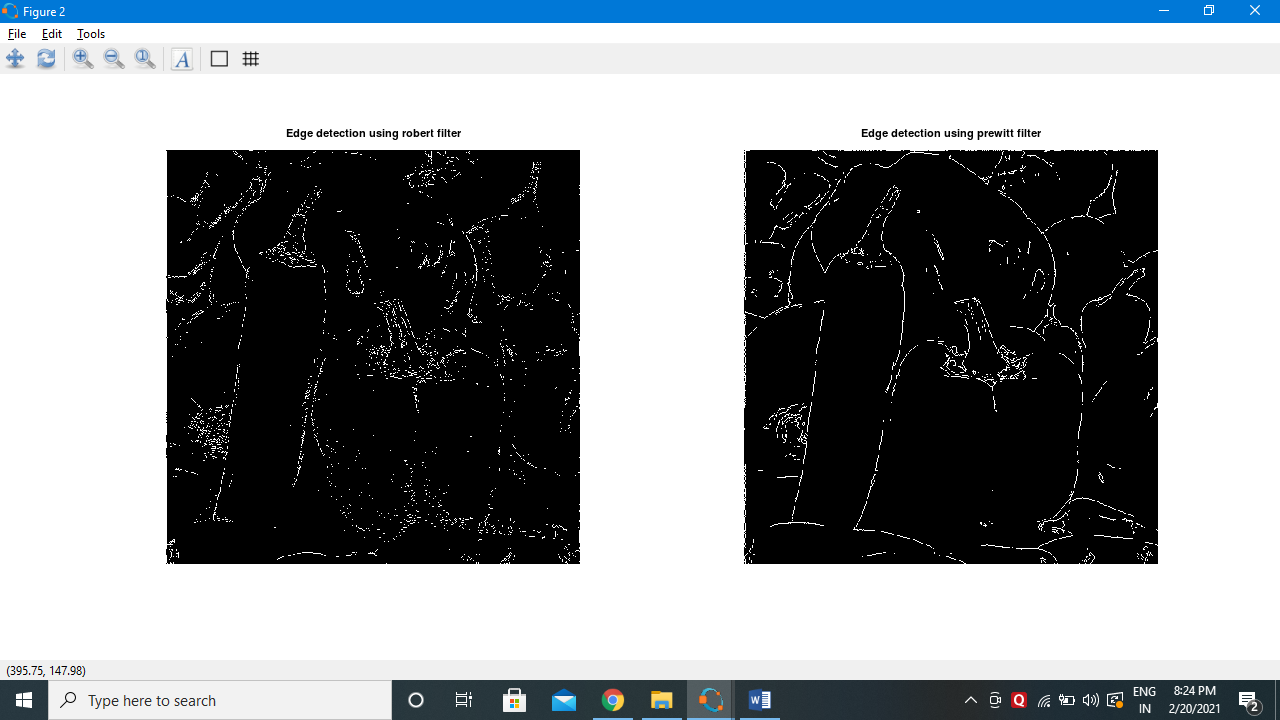
subplot(1,2,2)

imshow(prewitt);

title("Edge detection using prewitt filter");

**Output**

****

****

**Practical No. 6**

**Color Image Processing**

1. **Pseudocoloring**

**Code**

pkg load image;

close all;

clear all;

%READ INPUT IMAGE

A = imread('coin.png');

%RESIZE IMAGE

A = imresize(A,[256 256]);

%PRE-ALLOCATE THE OUTPUT MATRIX

Output = ones([size(A,1) size(A,2)]);

%COLORMAPS

#maps={'jet(256)';'hsv(256)';'cool(256)';'spring(256)';'summer(256)';'parula(256)';'hot(256)'};

%COLORMAP 1

map = colormap(jet(256));

Red = map(:,1);

Green = map(:,2);

Blue = map(:,3);

R1 = Red(A);

G1 = Green(A);

B1 = Blue(A);

%COLORMAP 2

map = colormap(cool(256));

Red = map(:,1);

Green = map(:,2);

Blue = map(:,3);

%RETRIEVE POSITION OF UPPER TRIANGLE

[x,y]=find(triu(Output)==1);

Output(:,:,1) = Red(A);

Output(:,:,2) = Green(A);

Output(:,:,3) = Blue(A);

for i=1:numel(x)

Output(x(i),y(i),1)=R1(x(i),y(i));

Output(x(i),y(i),2)=G1(x(i),y(i));

Output(x(i),y(i),3)=B1(x(i),y(i));

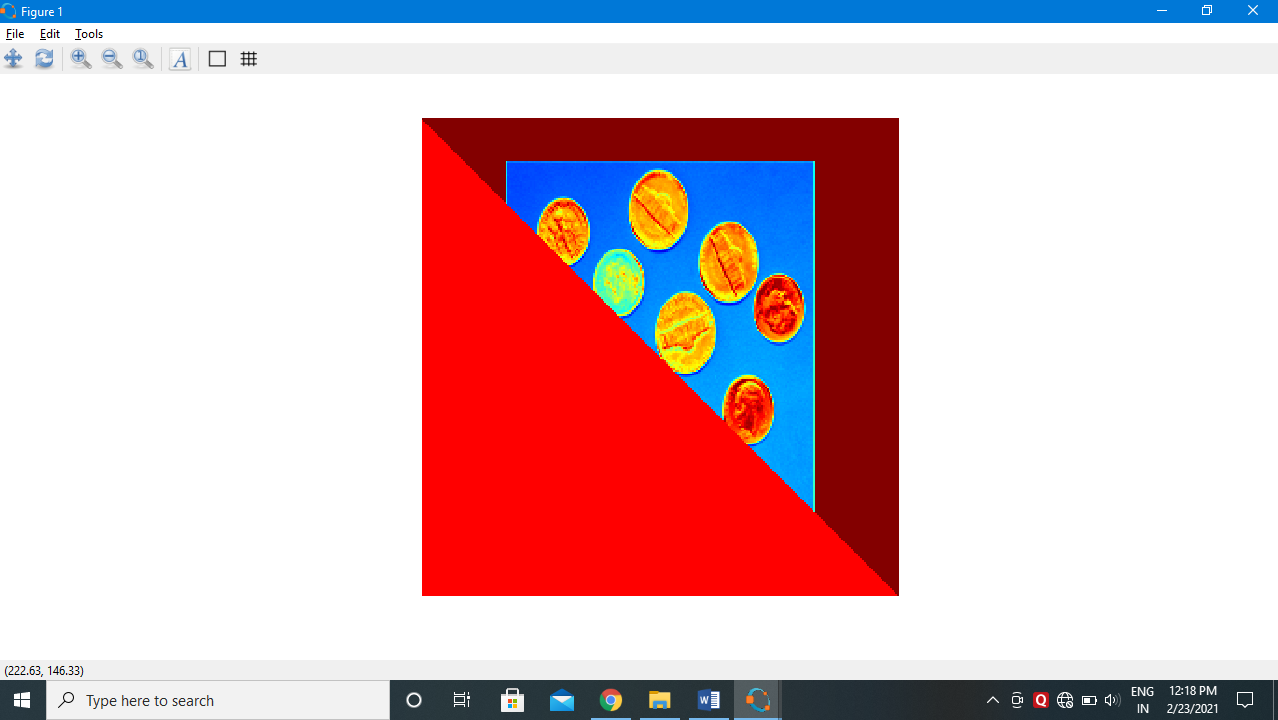
end

Output = im2uint8(Output);

%FINAL IMAGE

imshow(Output);

**Output**



1. **Intensity Slicing**

**Code**

clear all;

#im=input('Enter the file name);

input\_image=imread('hawk.jpeg');

k=rgb2gray(input\_image);

[x y z]=size(k);

% z should be one for the input image

k=double(k);

for i=1:x

for j=1:y

if k(i,j)>=0 && k(i,j)<50

m(i,j,1)=k(i,j,1)+25;

m(i,j,2)=k(i,j)+50;

m(i,j,3)=k(i,j)+60;

end

if k(i,j)>=50 && k(i,j)<100

m(i,j,1)=k(i,j)+55;

m(i,j,2)=k(i,j)+68;

m(i,j,3)=k(i,j)+70;

end

if k(i,j)>=100 && k(i,j)<150

m(i,j,1)=k(i,j)+52;

m(i,j,2)=k(i,j)+30;

m(i,j,3)=k(i,j)+15;

end

if k(i,j)>=150 && k(i,j)<200

m(i,j,1)=k(i,j)+50;

m(i,j,2)=k(i,j)+40;

m(i,j,3)=k(i,j)+25;

end

if k(i,j)>=200 && k(i,j)<=256

m(i,j,1)=k(i,j)+120;

m(i,j,2)=k(i,j)+60;

m(i,j,3)=k(i,j)+45;

end

end

end

figure,

imshow(uint8(k),[]);

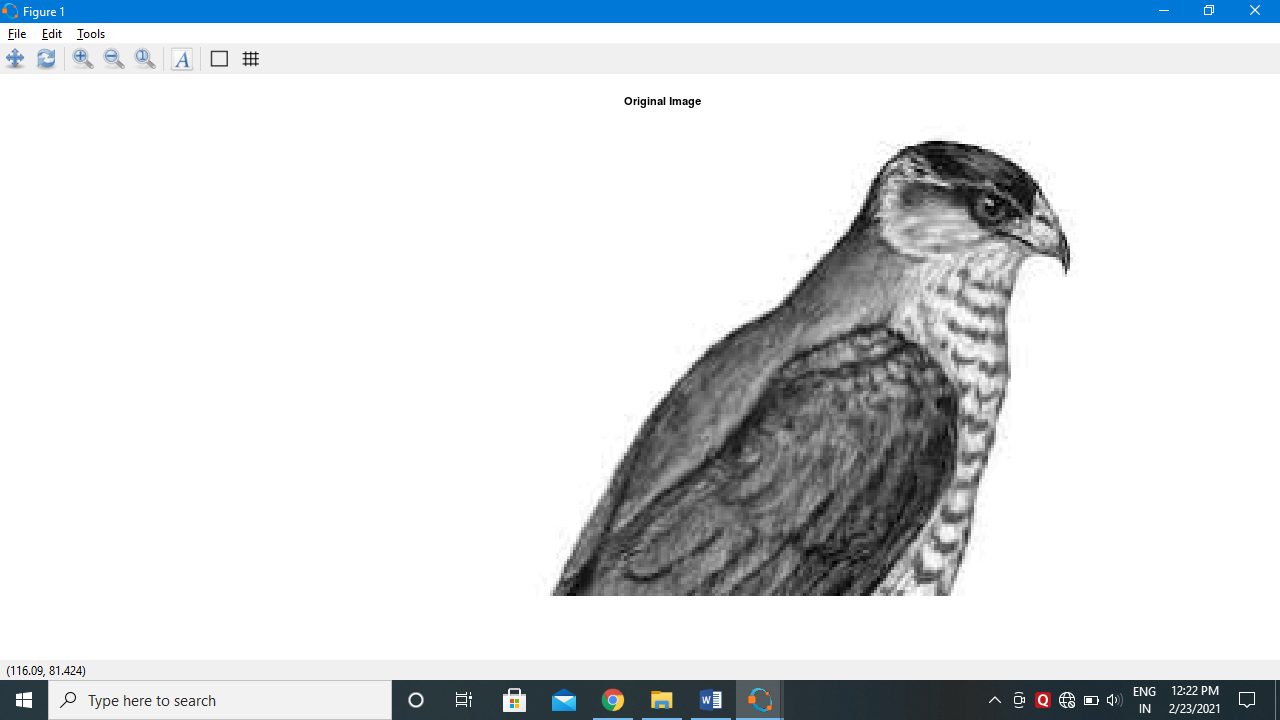
title('Original Image');

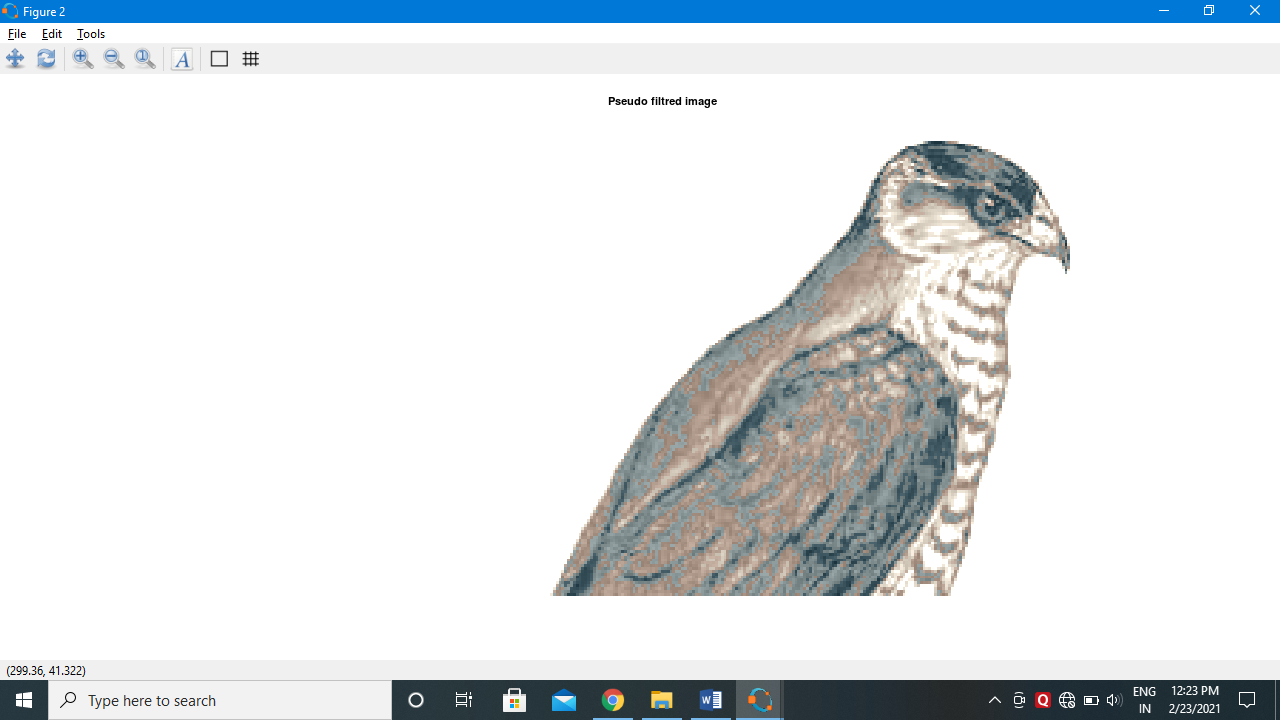
figure,

imshow(uint8(m),[]);

title("Pseudo filtred image");

**Output**





**Pseudo Image**

**Code**

pkg load image;

clear all;

img = imread('hawk1.png'); % Read image

figure, imshow(img);

title("original Image");

red = 0.66\*img;

green=0.25\*img;

blue = img;

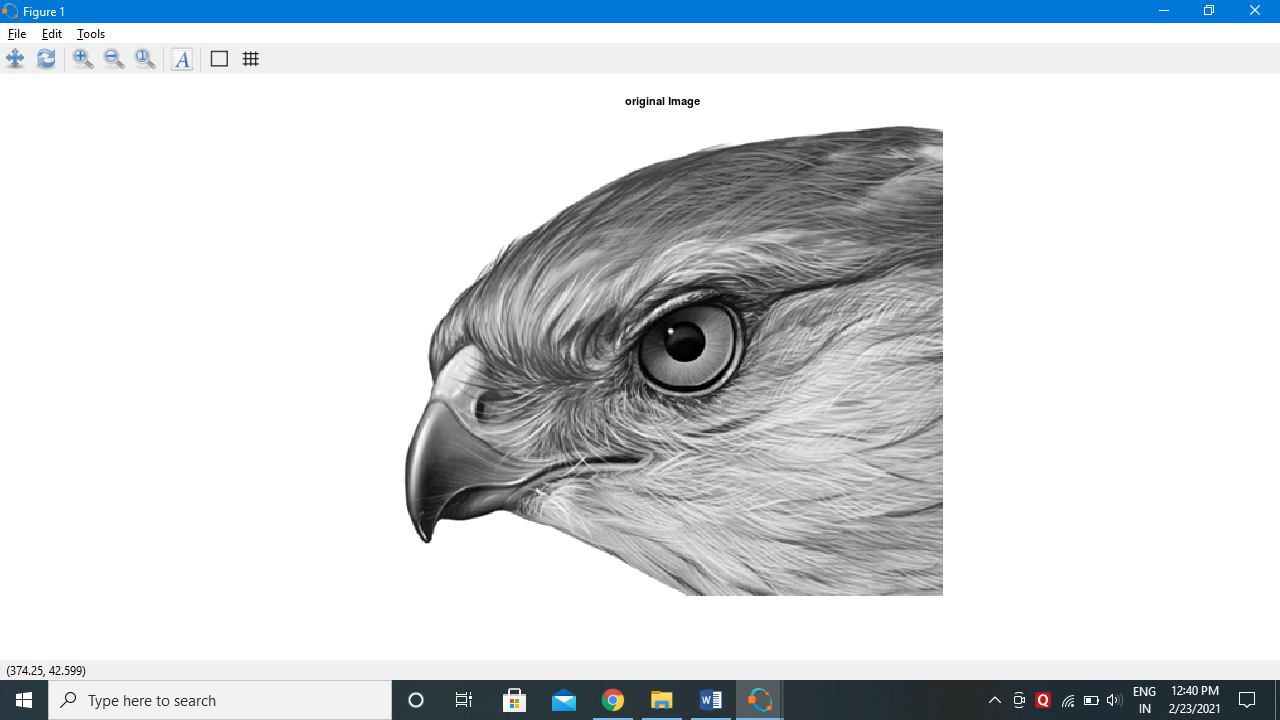
pseudo\_img = cat(3, red, green, blue);

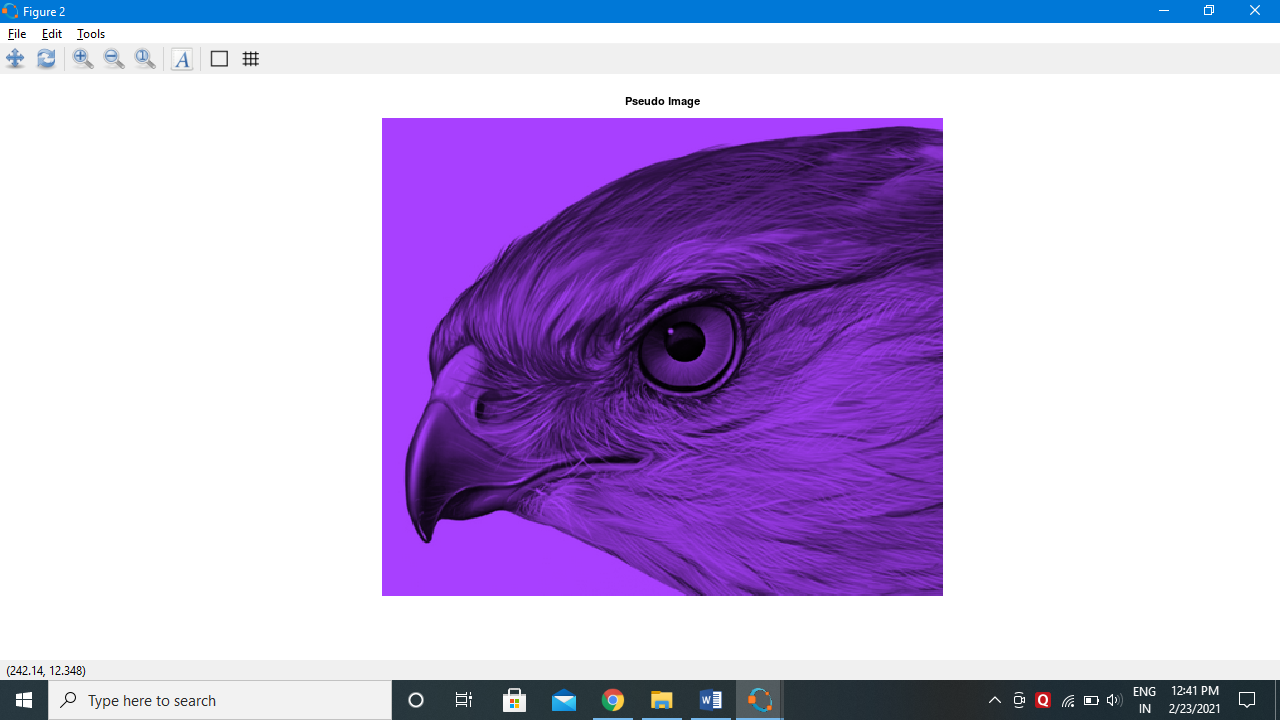
figure,

imshow(pseudo\_img);

title('Pseudo Image');

**Output**





**Practical No 7**

**Image Compression Techniques and Watermarking**

1. **Implement Huffman Coding**

**Code**

pkg load communications

sig = repmat([3 3 1 3 3 3 3 3 2 3],1,50);

symbols = [1 2 3];

p = [0.1 0.1 0.8];

dict = huffmandict(symbols,p);

hcode = huffmanenco(sig,dict);

dhsig = huffmandeco(hcode,dict);

isequal(sig,dhsig)

binarySig = de2bi(sig);

seqLen = numel(binarySig)

binaryhcode = de2bi(hcode);

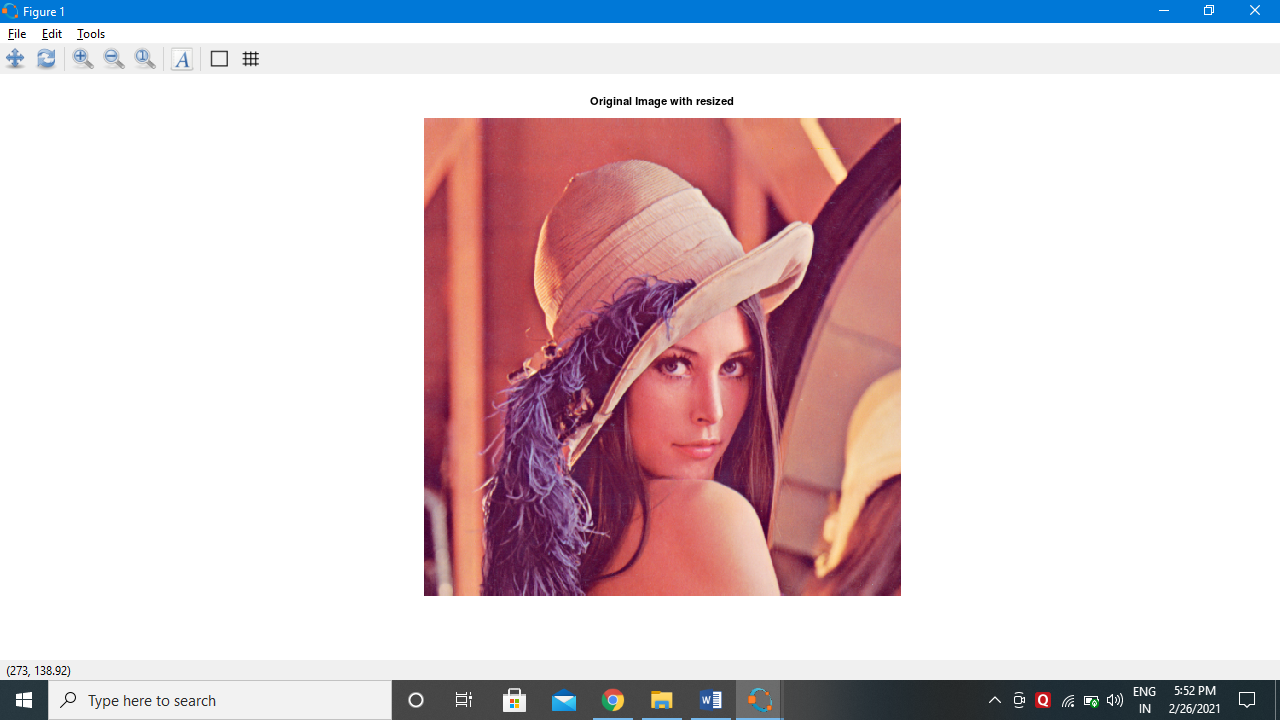
encodedLen = numel(binaryhcode)

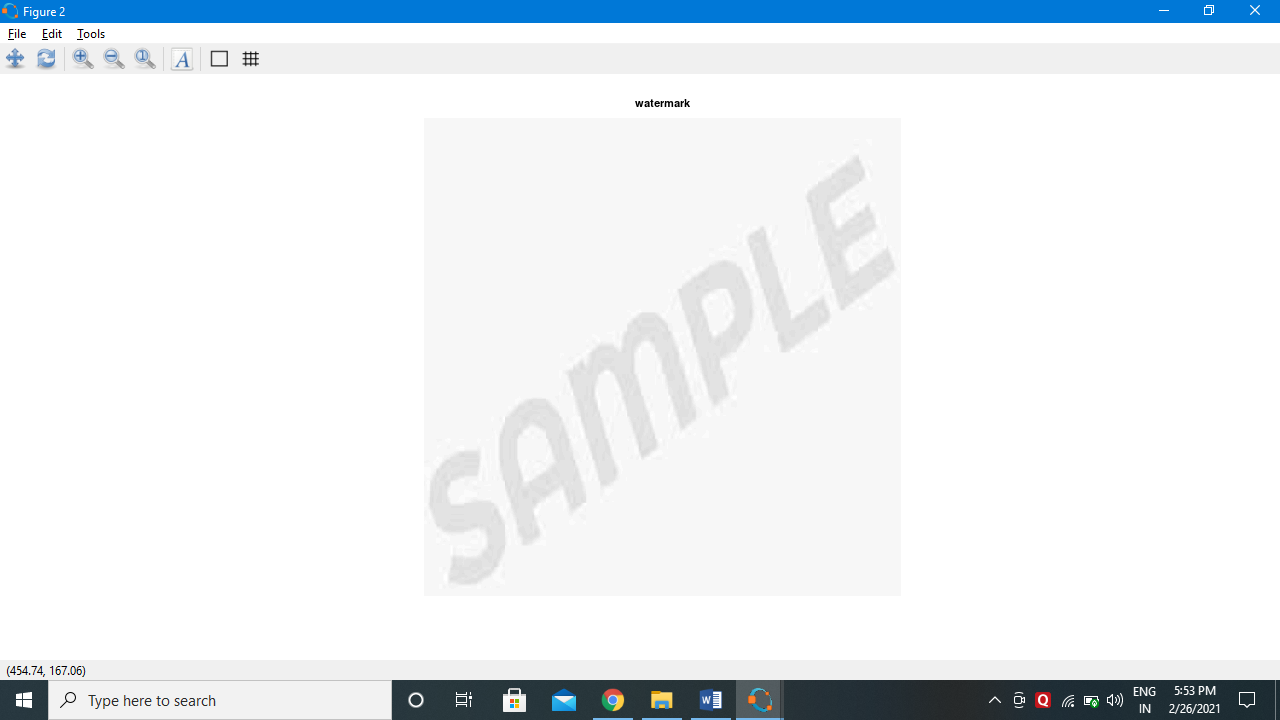
1. **Watermarking**

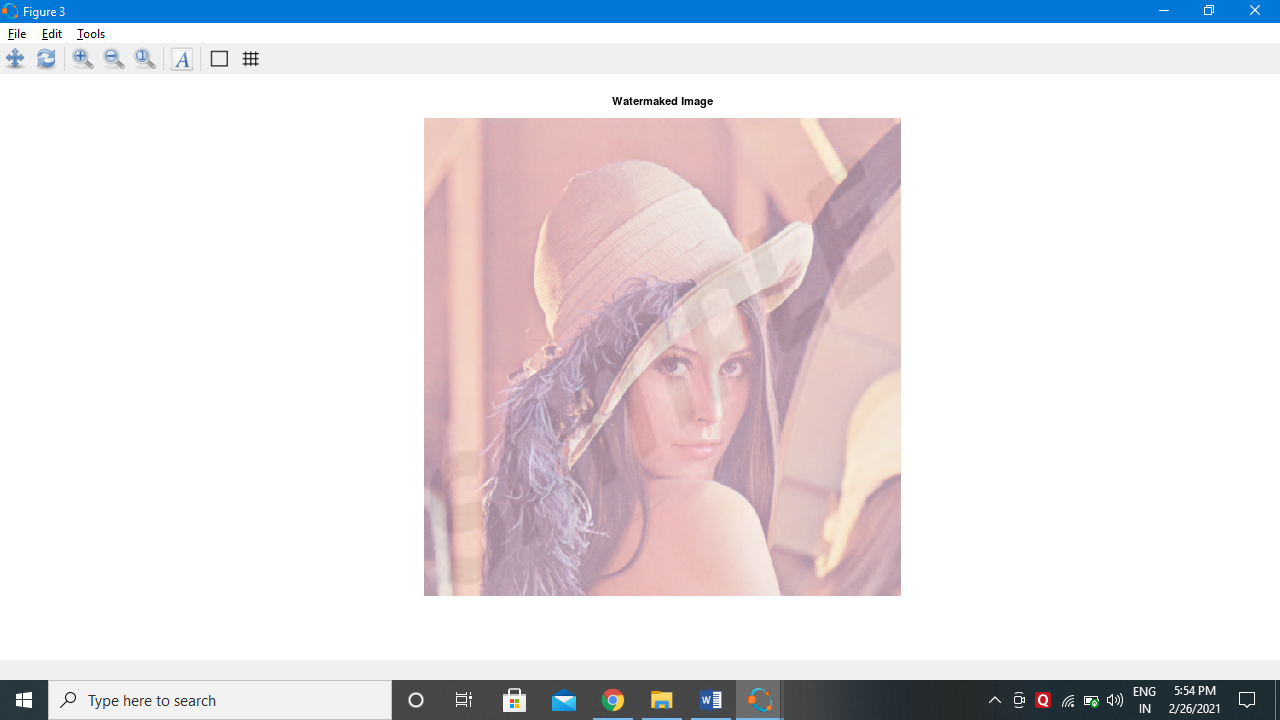
**Code**

pkg load image;clear all;close all;#Input Image where we want to apply watermarkf=imread('lena\_color\_512.tif');#For watermarking, size of inputimage and watermarking image should be same#there for we changed the size of image using imresize and dispalyedfr=imresize(f,[560 560]);figure;imshow(fr);title('Original Image with resized');#Watermarking Imagew=imread('watersample.jpeg');#Again Resized the Watermarking Imagewr=imresize(w,[560 560]);figure;imshow(wr);title('watermark');#Applied watermarkingalpha=0.6;fw=(1-alpha)\*fr + alpha.\*wr;#Display the watermarked Imagefigure;imshow(fw);title('Watermaked Image');

**Output**

****

****

****

**Practical No. 8**

**Basic Morphological Transformations**

1. **Boundary Extraction**

Boundary Extraction is the process of Morphological Image processing.

There are five methods of Morphological Image Transformations:

1. Erosion

In erosion, the final image gets eroded from the original image. It is represented by



Erosion is defined as 

1. Dilation

In dilation, the final image get added by the structuring element and the size of image increased. It is represented by 

Dilation is defined by



1. Opening

Erosion followed by dilation.

Opening A by B:



1. Closing

Dilation follower by erosion

Closing a by B is given by



**Code**

A=imread('newimage.png');

s=strel('disk',2,0);

F=imerode(A,s);

figure,

subplot(2,1,1);

imshow(A);

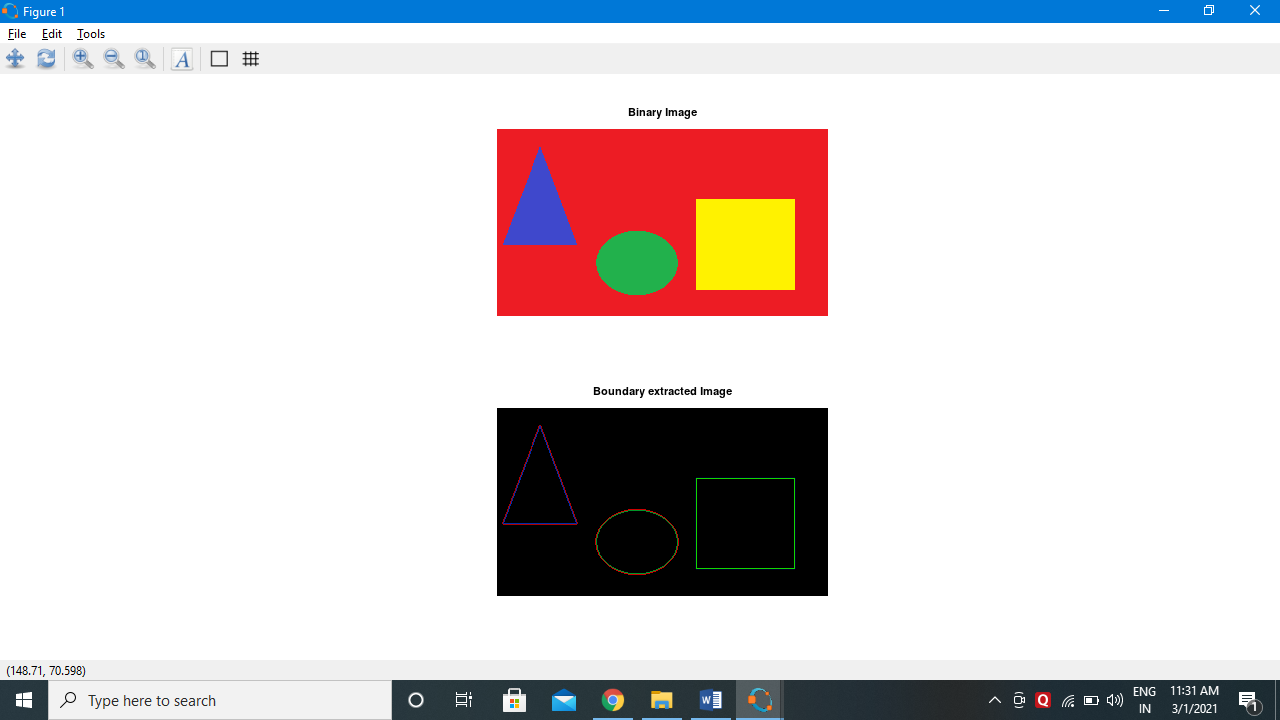
title('Binary Image');

subplot(2,1,2);

imshow(A-F);

title('Boundary extracted Image');

**Output**

****

1. **Thinning and Thickening**