

## RAMNIRANJAN JHUNJHUNWALA COLLEGE GHATKOPAR (W), MUMBAI - 400 086

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2020 – 2021**

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

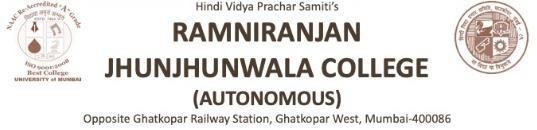
**Image And Vision Processing**

**Name :Sneha Balu Agale**

**Roll No.: 01**

SNEHA AGALE

1



### CERTIFICATE

This is to certify that Miss. **Sneha Balu Agale** with Roll No.**01** 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

SNEHA AGALE

2

**Index**

|  |  |  |
| --- | --- | --- |
| **NO** | **PRACTICAL** | **PAGE** |
| 1 | Implement Basic Intensity transformation functions   1. A) Image Inverse 2. B) Log Transformation 3. C) Power-law Transformation |  |
| 2 | Piecewise Transformation   1. A) Contrast Stretching 2. B) Thresholding 3. C) Bit-Plane Slicing |  |
| 3 | Implement Histogram Equalization |  |
| 4 | Image filtering in Spatial Domain  A. A) Low-pass Filter/Smoothing Filters  (Average, Weighted Average, Median and Gaussian)  A. B) High-pass Filter / Sharpening Filter  (Laplacian Filter, Sobel, Robert and Prewitt Filter to detect edge) |  |
| 5 | Analyze image in Frequency Domain   1. A) Low Pass/Smoothing filter 2. B) High Pass/Sharpening filter |  |
| 6 | Color Image Processing   1. A) Pseudocoloring 2. B) Separating the RGB Channels 3. C) Color Slicing |  |
| 7 | Image Compression Techniques and watermarking   1. Implement Huffman Coding 2. Add a watermark to the image |  |
| 8 | Basic Morphological Transformations   1. Boundary Extraction 2. Thinning and Thickening 3. Hole filling and Skeletons |  |

SNEHA AGALE

3

# Practical 1

### Implement Basic Intensity transformation functions

**A. 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)

### Code:

x = imread('doller.png'); #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

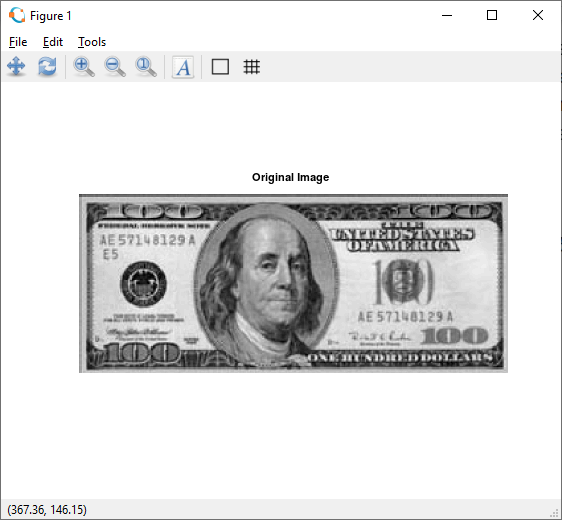
imshow(N);

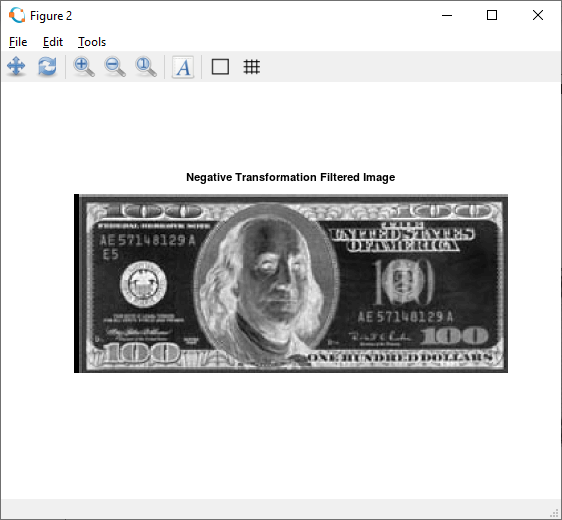
title('Negative Transformation Filtered Image');

SNEHA AGALE

4

### OutPut:

****

****

##### imread():

The *imread()* function reads images from the graphics files.

[**A**](https://in.mathworks.com/help/matlab/ref/imread.html#btnczv9-1-A) **= imread(**[**filename**](https://in.mathworks.com/help/matlab/ref/imread.html#mw_abb56802-ede7-4bfd-a64e-0743144c1db9)**)** reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.

[**A**](https://in.mathworks.com/help/matlab/ref/imread.html#btnczv9-1-A) **= imread(**[**filename**](https://in.mathworks.com/help/matlab/ref/imread.html#mw_abb56802-ede7-4bfd-a64e-0743144c1db9)[**,fmt**](https://in.mathworks.com/help/matlab/ref/imread.html#btnczv9-1-fmt)**)** additionally specifies the format of the file with the standard file extension indicated by fmt. If imread cannot find a file with the name specified

by filename, it looks for a file named *filename.fmt*.

##### im2double():

SNEHA AGALE

5

##### [I2 = im2double(I)](https://in.mathworks.com/help/matlab/ref/im2double.html#d122e624739)

[**I2 = im2double(I,'indexed')**](https://in.mathworks.com/help/matlab/ref/im2double.html#d122e624760)

[I2](https://in.mathworks.com/help/matlab/ref/im2double.html#mw_2f2def76-c130-4671-87d4-457d8bfbcd11) = im2double([I](https://in.mathworks.com/help/matlab/ref/im2double.html#buhz5vt-1-I)) converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a binary image. im2double rescales the output from integer data types to the range [0, 1].

[I2](https://in.mathworks.com/help/matlab/ref/im2double.html#mw_2f2def76-c130-4671-87d4-457d8bfbcd11) = im2double([I](https://in.mathworks.com/help/matlab/ref/im2double.html#buhz5vt-1-I),'indexed') converts the indexed image I to double

precision. im2double adds an offset of 1 to the output from integer data types.

##### subplot():

**subplot(**[**m**](https://in.mathworks.com/help/matlab/ref/subplot.html#btw1t4b-1-m)[**,n**](https://in.mathworks.com/help/matlab/ref/subplot.html#btw1t4b-1-n)[**,p**](https://in.mathworks.com/help/matlab/ref/subplot.html#btw1t4b-1-p)**)** divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes

##### imshow():

**imshow(**[**I**](https://in.mathworks.com/help/images/ref/imshow.html#bvmnrxi-1-I)**)** displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.

**B) 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:

pkg load image;

clear all;

close all;

x = imread('lena.png'); #Reading the image

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

SNEHA AGALE

6

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);

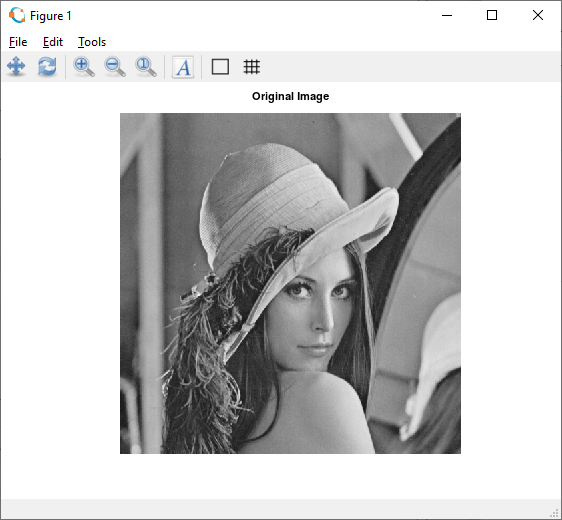
title('Original Image');

figure

imshow(N);

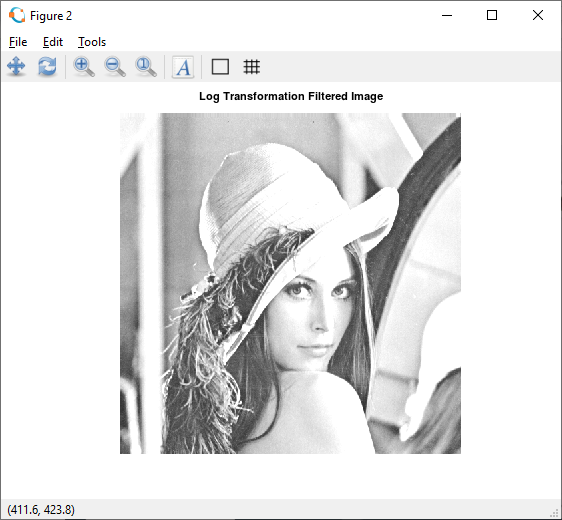
title('Log Transformation Filtered Image');

### OutPut:

****

SNEHA AGALE

7



##### rgb2gray():

[**I**](https://in.mathworks.com/help/matlab/ref/rgb2gray.html#buiz8mj-1-I) **=rgb2gray(**[**RGB**](https://in.mathworks.com/help/matlab/ref/rgb2gray.html#buiz8mj-1-RGB)**)** converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

##### imwrite():

**imwrite(**[**A**](https://in.mathworks.com/help/matlab/ref/imwrite.html#btv3cny-1-A)[**,filename**](https://in.mathworks.com/help/matlab/ref/imwrite.html#btv3cny-1-filename)**)** writes image data A to the file specified by filename, inferring the file format from the extension. imwrite creates the new file in your current folder. The bit depth of the output image depends on the data type of A and the file format.\

### ( C ) 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

SNEHA AGALE

8

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:

pkg load image; clear all;

close all;

x = imread('baby.jpg'); #Reading the image

#x = rgb2gray(x); #Converting RGB image to gray-level image x = im2double(x);

[row col] = size(x); gamma=2;

c=1;

for i = 1:row for j = 1:col

N(i,j)=c\*(x(i,j)^gamma); endfor

endfor

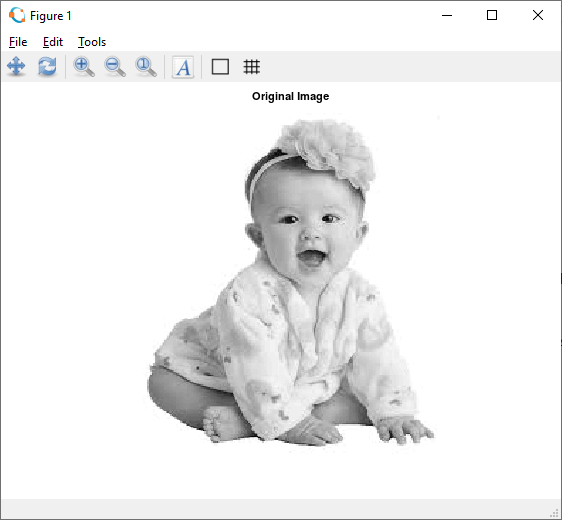
figure, imshow(x); title('Original Image');

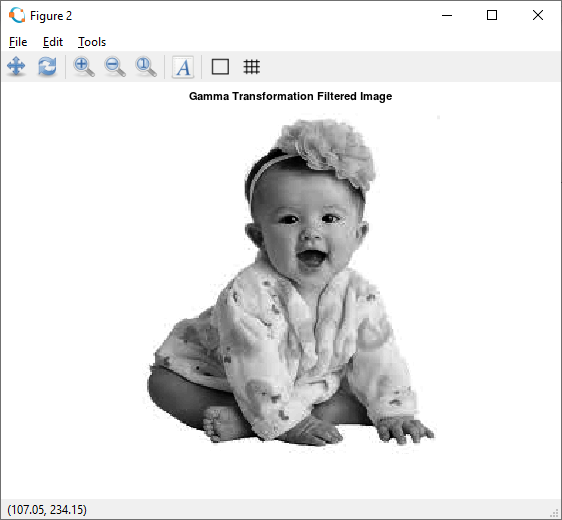
figure, imshow(N); title('Gamma Transformation Filtered Image');

SNEHA AGALE

9

### Output:





SNEHA AGALE

10

### Contrast Stretching

**Practical 2**

Contrast stretching is also known as normalization. It is a simple image enhancement technique. The quality of image is enhanced by stretching the range of intensity values.

### Using imadjust fucation:

#### Code:

pkg load image; a=imread('doller.png’);#agray=rgb2gray(a); ad=im2double(a);

subplot(2,2,1); imshow(ad); title("before"); subplot(2,2,2); imhist(ad); title("before hist"); imad=imadjust(ad,[0.44 0.8],[0.1 0.9]);

subplot(2,2,3); imshow(imad); title("After");

subplot(2,2,4); imhist(imad); title("After hist");

### output:

****

##### imadjust():

Adjust image or colormap intensity (values).

Returns an image of equal dimensions to I, cmap, or RGB, with its intensity values adjusted, usually for the purpose of increasing the image contrast.

The values are rescaled according to the input and output limits, low\_in and high\_in,

and low\_out and high\_out respectively. The first pair sets the lower and upper limits on the input image, values above and below them being clipped. The second pair sets the lower and upper limits for the output image, the interval to which the image will be scaled after clipping the input limits.

For example:

##### imadjust (img, [0.2; 0.9], [0; 1])

will clip all values in img outside the range [0.2 0.9], and then rescale them linearly into the range [0 1].

### Contrast Stretching

**Using Inputs from user r1,r2,s1,s2**

**Code:**

pkg load image; clear all;

close 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: ");

SNEHA AGALE

12

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);

SNEHA AGALE

13

subplot(1,2,1) imshow(s);

title("Contrast Streched Image"); subplot(1,2,2)

imhist(s);

title('Histogtram Of Contrast Streched Image');

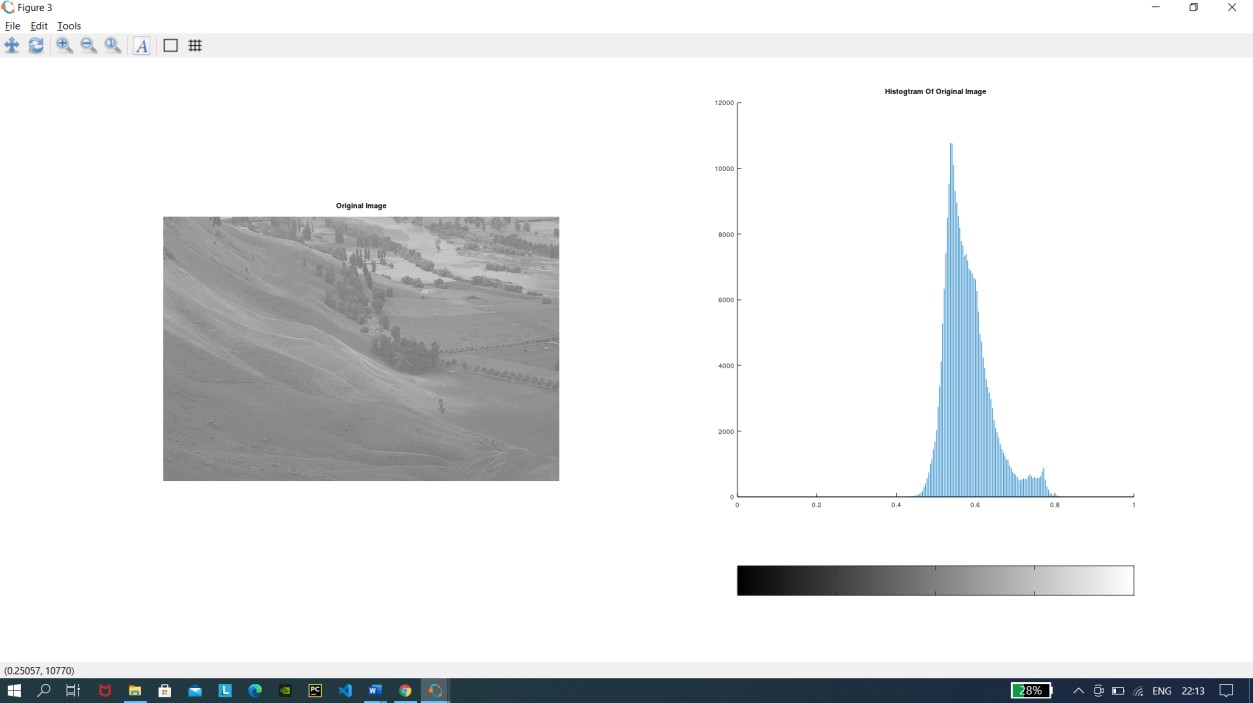
### Output:

Enter R1: 0.44

Enter R2: 0.8

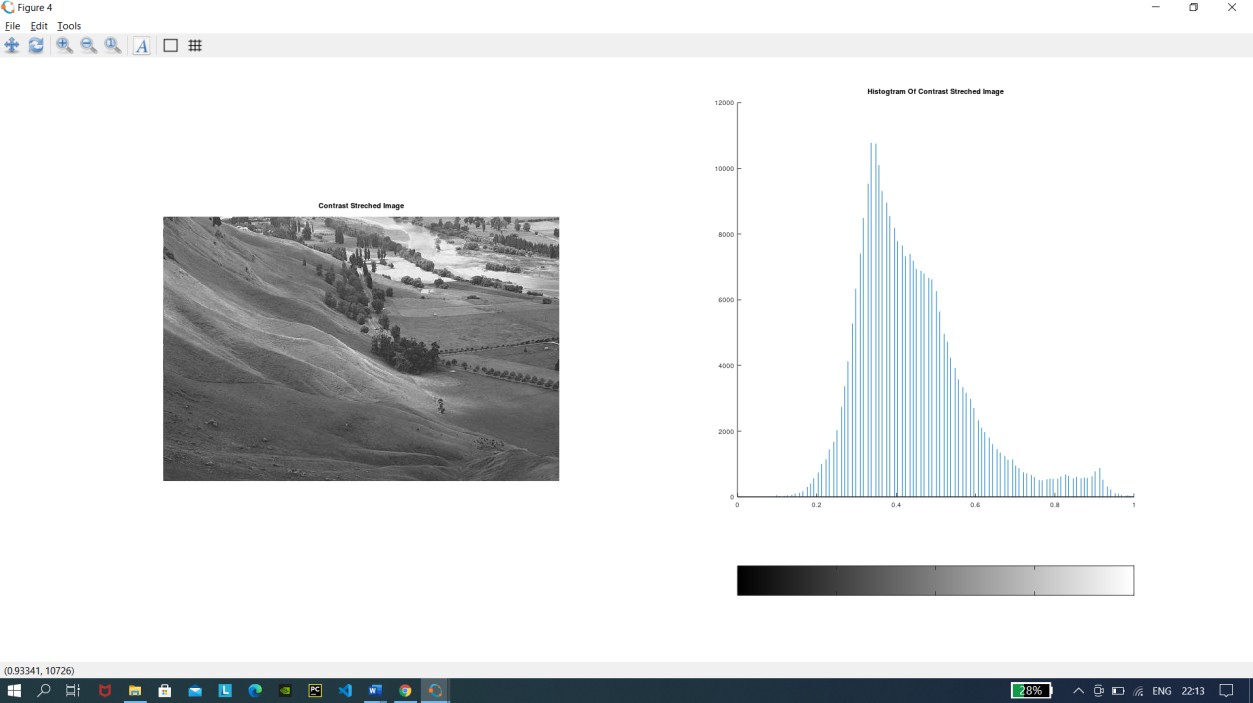
Enter S1: 0.1

Enter S2: 0.98



SNEHA AGALE

14



### Thresholding:

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This [image analysis](https://in.mathworks.com/discovery/image-analysis.html) 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("fields.jpg"); #r=rgb2gray(r); #r=im2double(r); imhist(r);

thr=150;

[m n]=size(r);

s=zeros(m,n); for i=1:m

SNEHA AGALE

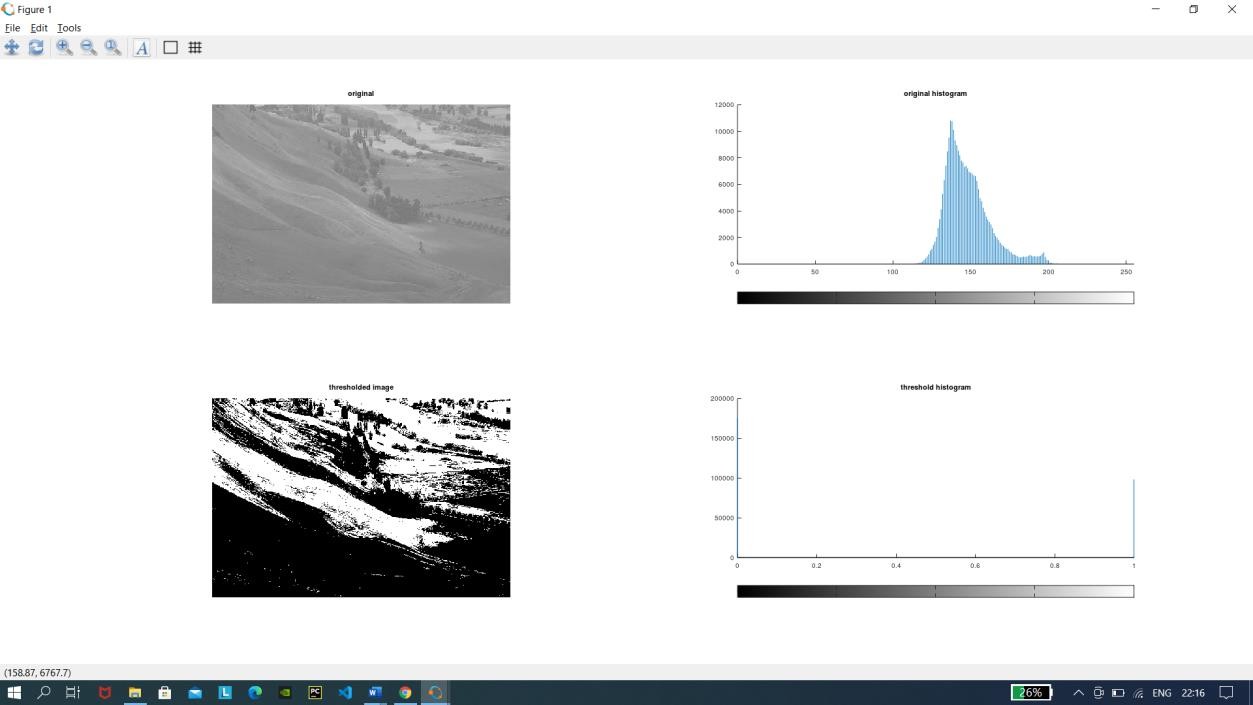
15

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:



##### zeros():

**B = zeros(n)** returns an n-by-n matrix of zeros. An error message appears if n is not a scalar.

**B = zeros(m,n)** or B = zeros([m n]) returns an m-by-n matrix of zeros.

SNEHA AGALE

16

### Image reconstruction using n bit planes

#### Code:

pkg load image A=imread('doller.png'); 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');

SNEHA AGALE

17

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");

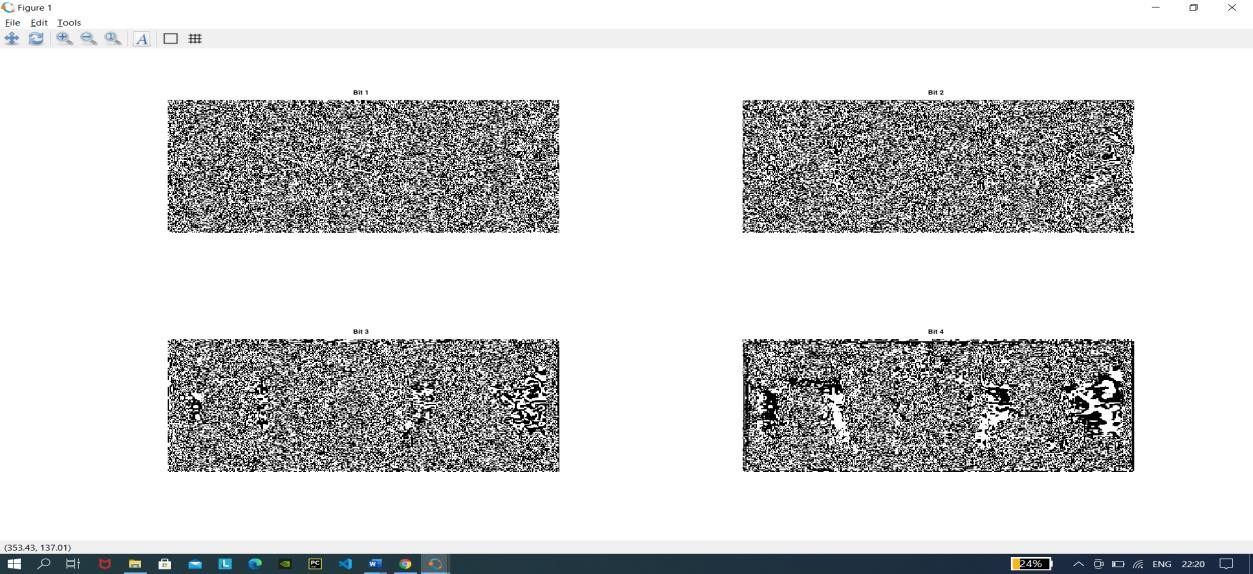
SNEHA AGALE

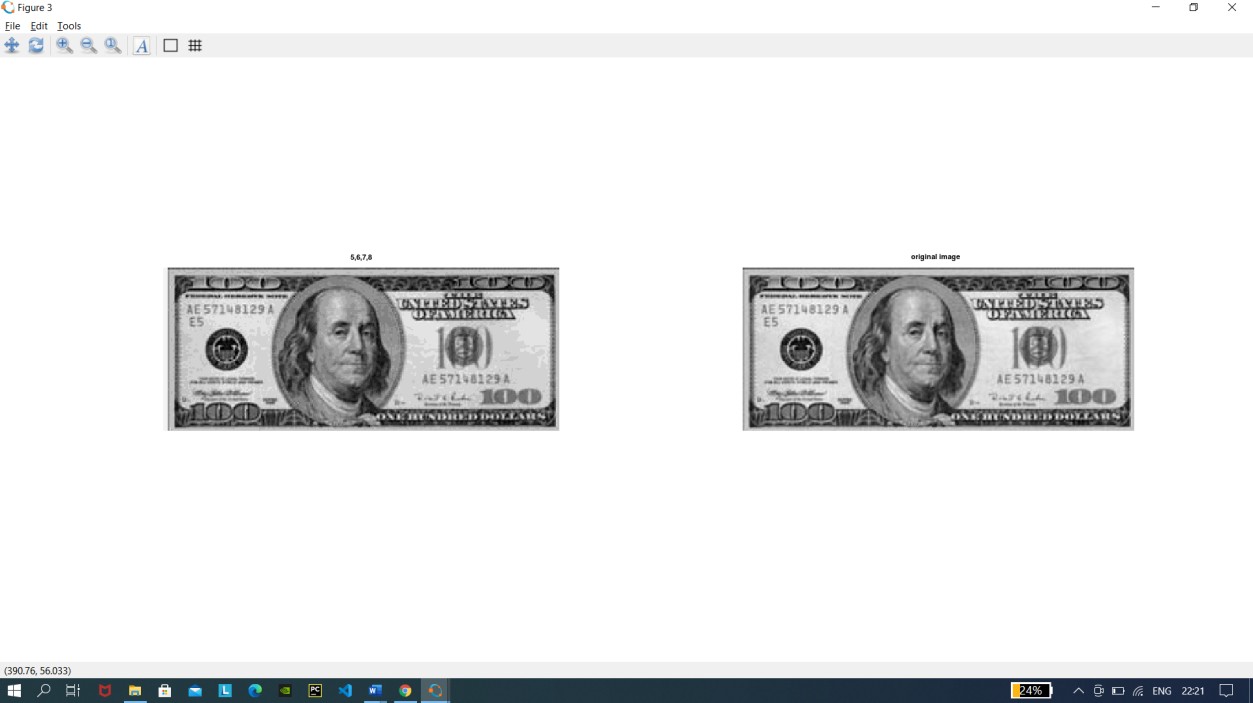
18

### Output:

19

### 

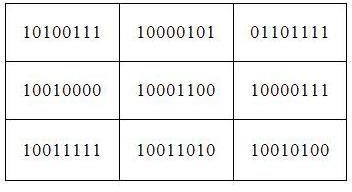




1. The nth plane in the pixels are multiplied by the constant 2^n-1
2. For instance, consider the matrix A= A=[167 133 111

144 140 135

159 154 148] and the respective bit format



1. Combine the 8 bit plane and 7 bit plane.

For 10100111, multiply the 8 bit plane with 128 and 7 bit plane with 64.\ (1x128) + (0x64) + (1x0) + (0x0) + (0x0) + (1x0) + (1x0) + (1x0) = 128

1. Repeat this process for all the values in the matrix and the final result will be [128 128 64

128 128 128

128 128 128]

#### Code:

pkg load image;

%Image reconstruction by combining 8 bit plane and 7 bit plane A=imread('coins.png');

B=zeros(size(A)); B=bitset(B,7,bitget(A,7));

B=bitset(B,8,bitget(A,8)); B=uint8(B); figure,imshow(B);

%Image reconstruction by combining 8,7,6 and 5 bit planes A=imread('coins.png');

B=zeros(size(A)); B=bitset(B,8,bitget(A,8));

B=bitset(B,7,bitget(A,7));

B=bitset(B,6,bitget(A,6));

B=bitset(B,5,bitget(A,5)); B=uint8(B); figure,imshow(B);

SNEHA AGALE

20

### OutPut:



##### biget():

Get bit at specified position Syntax

##### C = bitget(A, *bit*)

SNEHA AGALE

21

Description

C = bitget(A, *bit*) returns the value of the bit at position *bit* in A. Operand A must be an unsigned integer or an array of unsigned integers, and *bit* must be a number between 1 and the number of bits in the unsigned integer class of A (e.g., 32 for the uint32 class).

##### bitset():

Set bit at specified position Syntax

##### C = bitset(A, *bit*)

* **C = bitset(A, *bit*, v)**

Description

C = bitset(A, *bit*) sets bit position *bit* in A to 1 (on). A must be an unsigned integer or an array of unsigned integers, and *bit* must be a number between 1 and the number of bits in the unsigned integer class of A (e.g., 32 for the uint32 class).

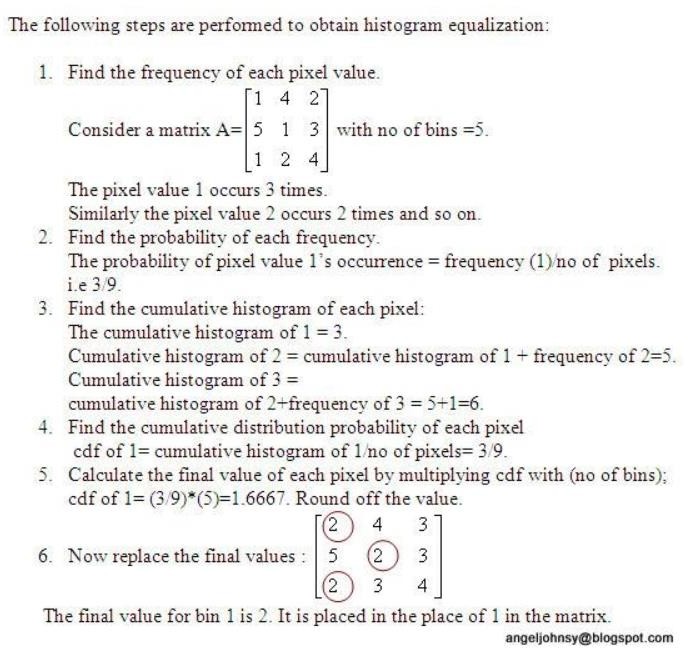
C = bitset(A, *bit*, v) sets the bit at position *bit* to the value v, which must be either 0 or 1.

SNEHA AGALE

22

# Practical-3

### Histogram equalization without using histeq() function.



#### Code:

pkg load image; a=imread('fields.jpg'); #a=rgb2gray(a); #a=a(1:10,1:10)

r=size(a,1);

c=size(a,2);

SNEHA AGALE

23

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

SNEHA AGALE

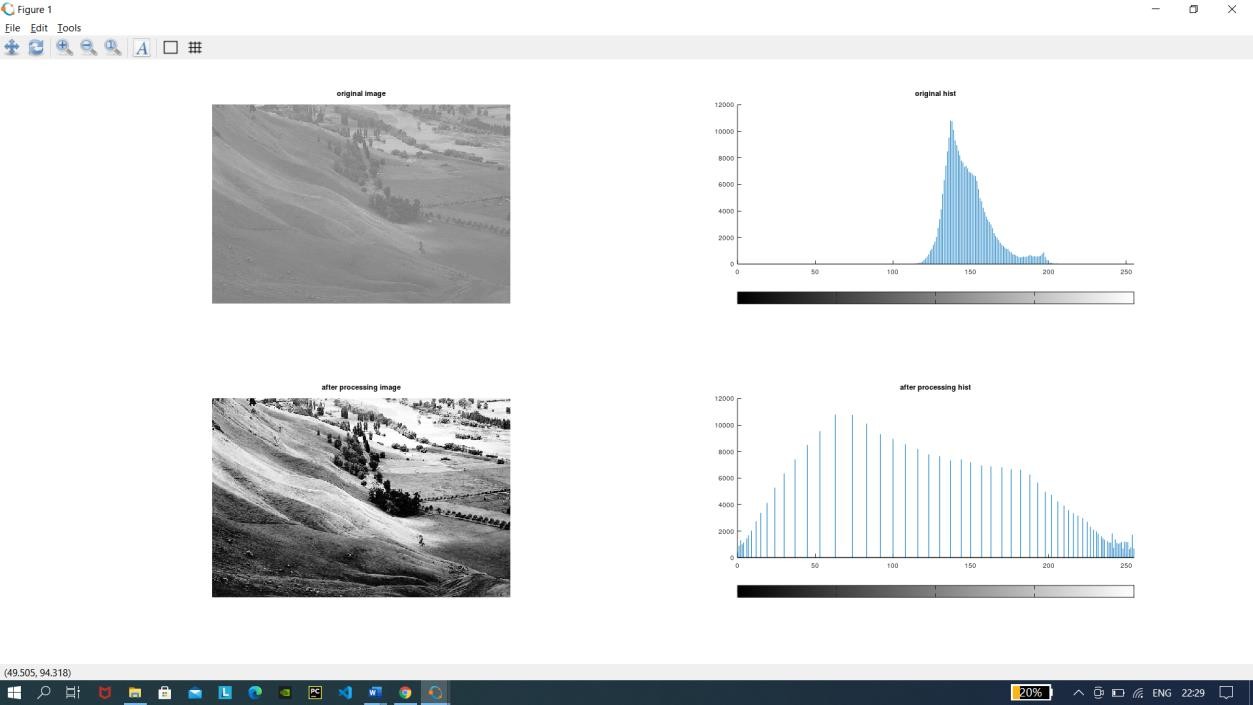
24

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:



##### round():

**Y =** [**round(X**](https://in.mathworks.com/help/matlab/ref/round.html#buftmpz-X)**)** rounds each element of X to the nearest integer. In the case of a tie, where an element has a fractional part of exactly 0.5, the round function rounds away from zero to the integer with larger magnitude.

Y = [round(X](https://in.mathworks.com/help/matlab/ref/round.html#buftmpz-X),[N](https://in.mathworks.com/help/matlab/ref/round.html#buftmpz-N)) rounds to N digits:

* N > 0: round to N digits to the *right* of the decimal point.
* N = 0: round to the nearest integer.
* N < 0: round to N digits to the *left* of the decimal point.

SNEHA AGALE

25

# Practical 4

#### Code:

**(Low Pass-Average filter using inbuilt functions)**

clear all; close all;

pkg load image; a=imread('hawk1.png'); #a=rgb2gray(a); #imwrite(a,'hawk1.png'); a=im2double(a); r=imnoise(a,'salt & pepper' ); f=ones(3,3)/9;

af=filter2(f,r); figure

#imshow(a); title('original');

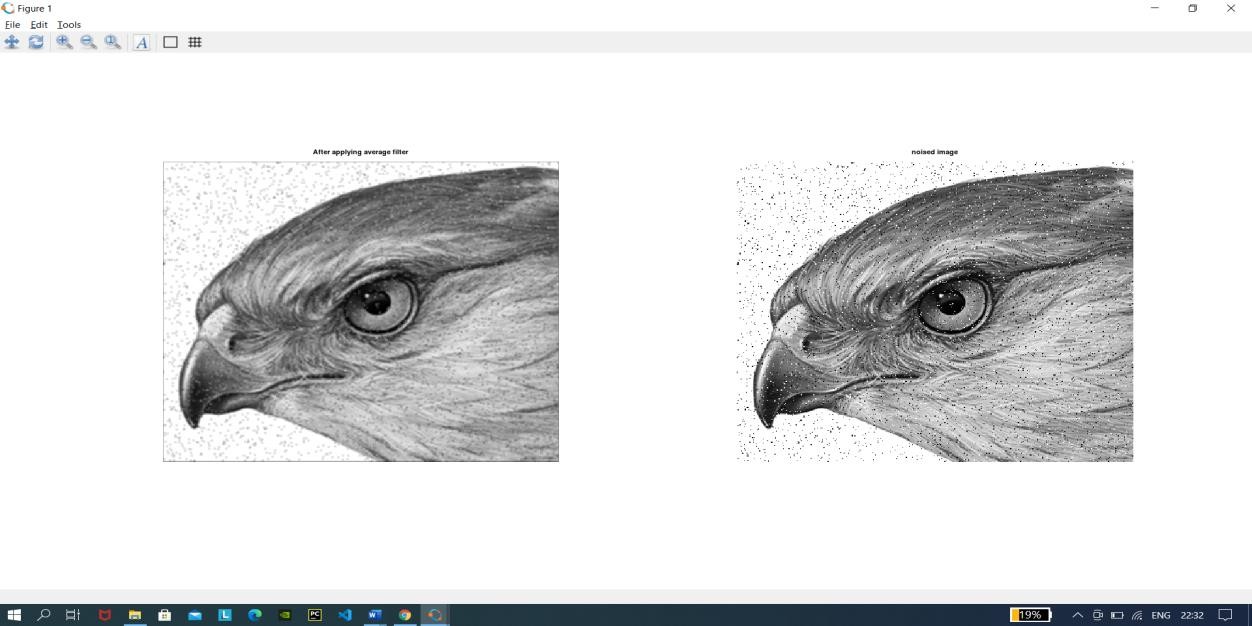
subplot(1,2,1);imshow(af); title('After applying average filter'); subplot(1,2,2)

imshow(r); title('noised image');

SNEHA AGALE

26

**OutPut:**



##### imnoise():

Add noise to an image

##### Syntax

* + **J = imnoise(I,type)**
  + **J = imnoise(I,type,parameters)**

**Description**

J = imnoise(I,type) adds noise of a given type to the intensity image I. type is a string that can have one of these values.

|  |  |
| --- | --- |
| **Value** | **Description** |
| 'gaussian' | Gaussian white noise with constant mean and variance |
| 'localvar' | Zero-mean Gaussian white noise with an intensity-dependent variance |
| 'poisson' | Poisson noise |
| 'salt & pepper' | On and off pixels |
| 'speckle' | Multiplicative noise |

##### filter2():

Perform two-dimensional linear filtering.

##### Y = filter2 (B, X)

**Y = filter2 (B, X, SHAPE)**

Apply the 2-D FIR filter B to X.

If the argument SHAPE is specified, return an array of the desired shape. Possible values are:

27

SNEHA AGALE

"full"

pad X with zeros on all sides before filtering.

"same"

unpadded X (default)

"valid"

trim X after filtering so edge effects are no included.

**Code:**

**(Low Pass Average filter without using inbuilt functions)**

close all;

pkg load image;

im=imread('hawk1.png'); % To read image #f=rgb2gray(CIm); % To convert RGB to Grayimage

Nim=imnoise(im,'salt & pepper'); % Adding salt & pepper noise to image w=(1/16)\*[1 2 1;2 4 2;1 2 1]; % Defining the box filter mask

% get array sizes [ma, na] = size(Nim) [mb, nb] = size(w)

% To do convolution

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

% extract region of size(a) from c r1 = floor(mb/2) + 1;

r2 = r1 + ma - 1;

c1 = floor(nb/2) + 1; c2 = c1 + na - 1;

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

SNEHA AGALE

28

figure subplot(1,2,1) imshow(Nim);

title('Noisy Image(Salt & Pepper Noise)'); subplot(1,2,2)

imshow(uint8(c));

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

### OutPut:

##### floor():

**Y =** [**floor(X**](https://in.mathworks.com/help/matlab/ref/floor.html#bug_5mn-1-X)**)** rounds each element of X to the nearest integer less than or equal to that element.

**Y =** [**floor(t**](https://in.mathworks.com/help/matlab/ref/floor.html#bug_5mn-1-t)**)** rounds each element of the duration array t to the nearest number of seconds less than or equal to that element.

**Y =** [**floor(t**](https://in.mathworks.com/help/matlab/ref/floor.html#bug_5mn-1-t)**,**[**unit**](https://in.mathworks.com/help/matlab/ref/floor.html#bug_5mn-1-unit)**)** rounds each element of t to the nearest number of the specified unit of time less than or equal to that element.

SNEHA AGALE

29

### low Pass Filter(Median Filter)#Meadian Spatial Domain Filtering

#### Code:

pkg load image; # Read the image

a=imread('hawk1.png'); 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 endfor

img\_new1 = uint8(img\_new1); figure

subplot(1,2,1); imshow(img\_new1); title('After Applying Median Filter'); subplot(1,2,2); imshow(img\_noisy1);title('noisy image');

SNEHA AGALE

30

### OutPut:

##### sort():

Sort array elements in ascending or descending order

##### Syntax

* + **B = sort(A)**
  + **B = sort(A,dim)**
  + **B = sort(...,mode)**
  + **[B,IX] = sort(...)**

**Description**

B = sort(A) sorts the elements along different dimensions of an array, and arranges those elements in ascending order.

|  |  |
| --- | --- |
| **If A is a ...** | **sort(A) ...** |
| Vector | Sorts the elements of A. |
| Matrix | Sorts each column of A. |
| Multidimensional array | Sorts A along the first non-singleton dimension, and returns an array of sorted vectors. |
| Cell array of strings | Sorts the strings in ASCII dictionary order. |

SNEHA AGALE

31

### Second order derivative-The Laplacian Filter

#### Code:

%Input Image clear all;

A=imread('coins.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');

SNEHA AGALE

32

### OutPut:

##### padarray():

[**B**](https://in.mathworks.com/help/images/ref/padarray.html#d122e210839) **= padarray(**[**A**](https://in.mathworks.com/help/images/ref/padarray.html#d122e210575)[**,padsize**](https://in.mathworks.com/help/images/ref/padarray.html#d122e210625)**)** pads array A with an amount of padding in each dimension specified by padsize. The padarray function pads numeric or logical images with the

value 0 and categorical images with the category <undefined>. By default, paddarray adds padding before the first element and after the last element of each dimension.

##### uint8():

1. bit unsigned integer arrays y = uint8(10);

### First Order Derivative-Sobel Operator for edge detection without using edge function

**Code:**

clear all; A=imread('peppers.jfif'); figure,

subplot(1,2,1); imshow(A); title('Original'); C=double(A);

size(C)

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

SNEHA AGALE

33

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)));

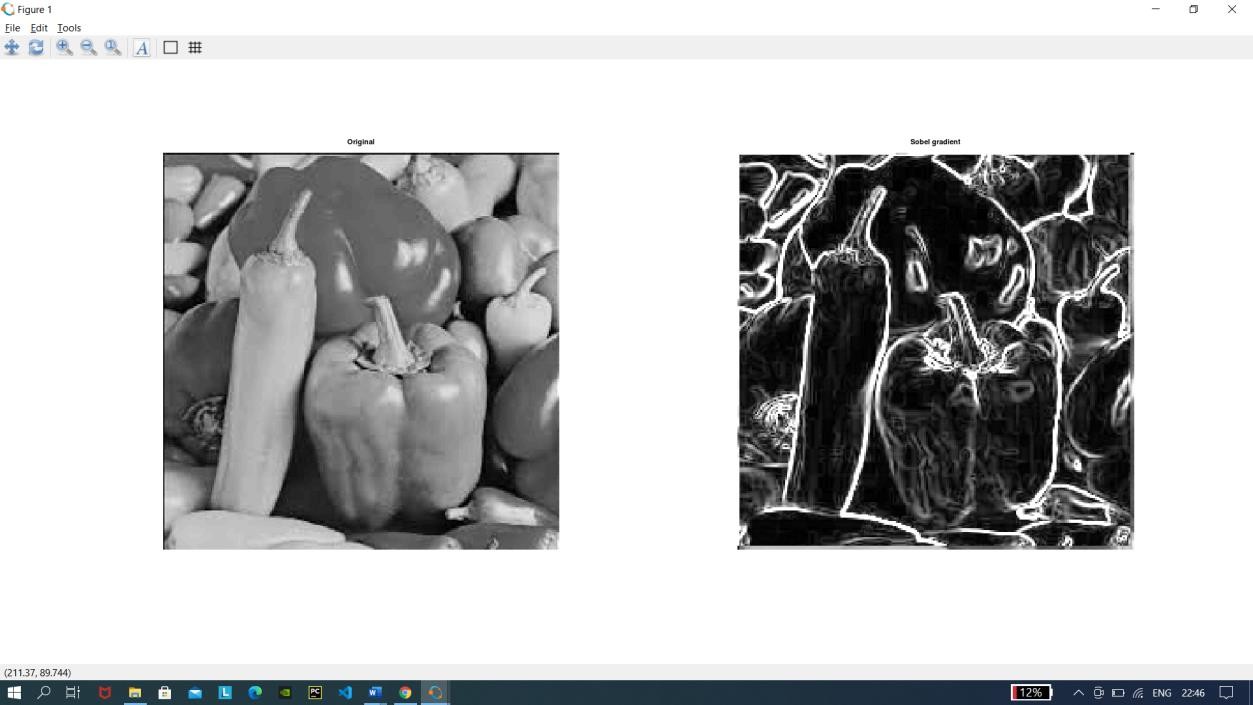
%The gradient of the image # B(i,j)=abs(Gx)+abs(Gy);

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 using edge function**

#### Code:

#load package of image pkg load image;

#Take input image img=imread("peppers.jfif");

#function to find edge using sobel filter sobel = edge(img,'Sobel');

figure 1,

34

SNEHA AGALE

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,

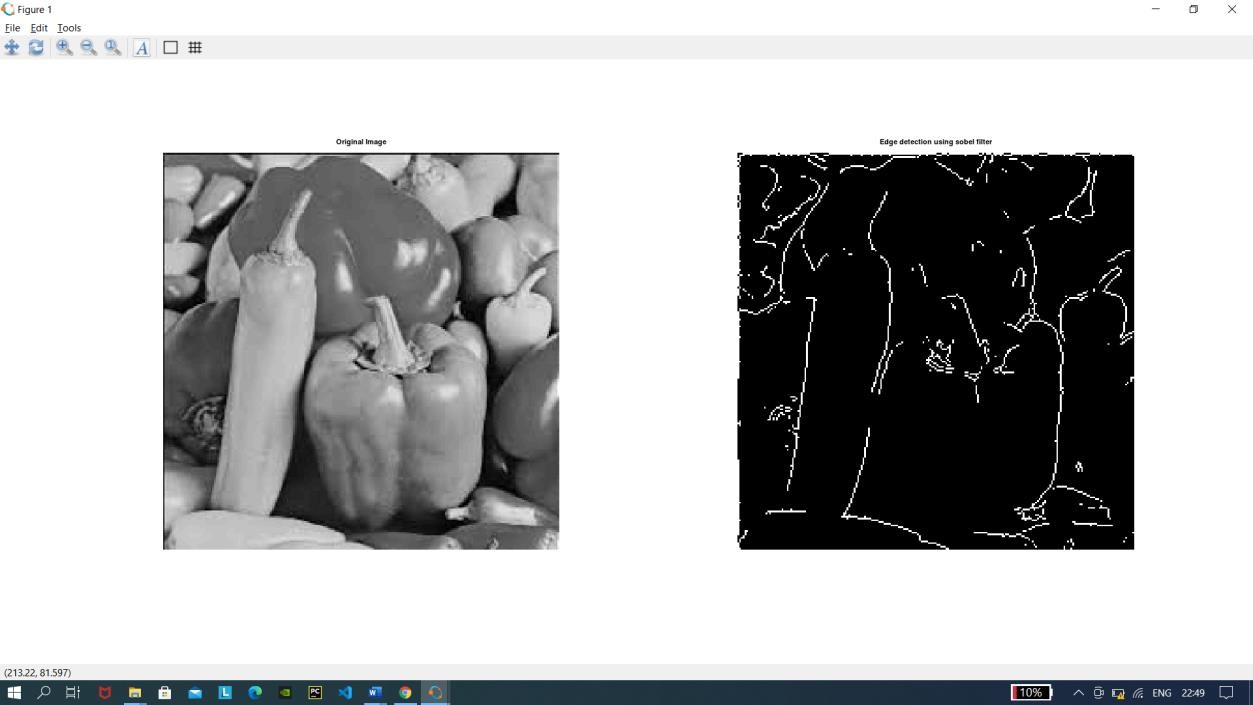
subplot(1,2,1) imshow(robert);

title('Edge detection using robert filter'); subplot(1,2,2)

imshow(prewitt);

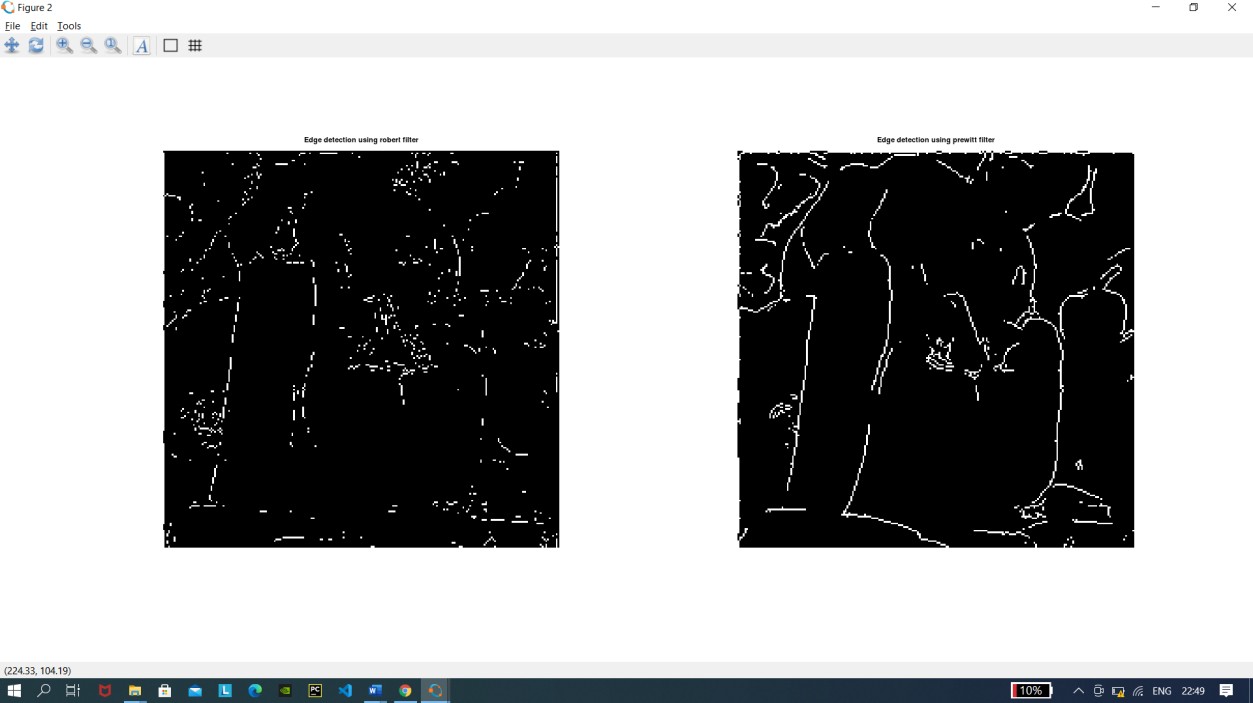
title("Edge detection using prewitt filter");

### OutPut:



SNEHA AGALE

35



SNEHA AGALE

36

# Practical -6

### Color Image Processing

* 1. **Pseudocoloring**

#### Code:

pkg load image; close all;

clear all;

%READ INPUT IMAGE

A = imread('coins.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);

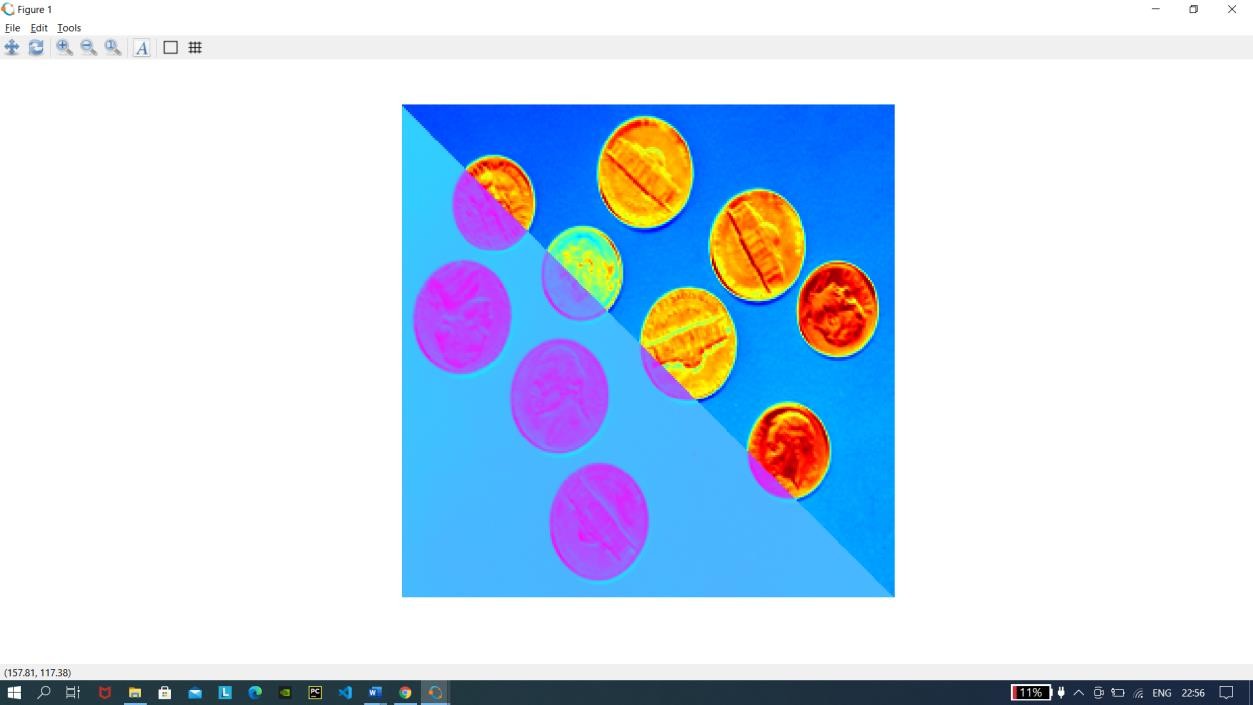
%FINAL IMAGE

37

SNEHA AGALE

imshow(Output);

### OutPut:



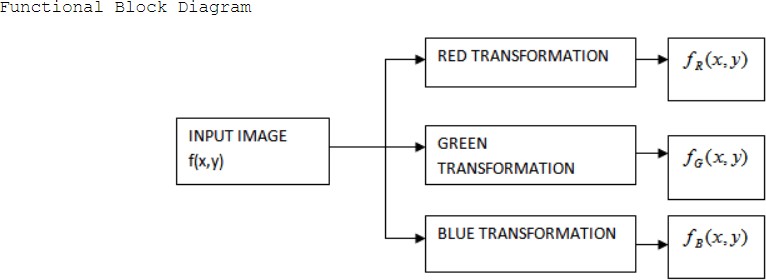
##### colormap():

the purpose of colormap is to define the colors of the graphics objects like image, surface and patch objects. A colormap is basically a matrix with values between 0 & 1. Colormaps can have any length, but width-wise they must have 3 columns. Each row of the matrix defines one color by using an RGB triplet.

SNEHA AGALE

38

##### Separating the RGB Channels



**Code:**

pkg load image; clear all;

close all;

img1 = imread('cartoon.jpg'); img\_r=img1;

img\_r(:,:,2)=0;

img\_r(:,:,3)=0;

img\_g=img1; img\_g(:,:,1)=0;

img\_g(:,:,3)=0;

img\_b=img1; img\_b(:,:,1)=0;

img\_b(:,:,2)=0;

subplot(2,2,1);imshow(img1);title("Original image"); subplot(2,2,2);imshow(img\_r);title("Red channel"); subplot(2,2,3);imshow(img\_g);title("Green channel"); subplot(2,2,4);imshow(img\_b);title("Blue channel");

SNEHA AGALE

39

### OutPut

### 

* 1. **Intensity Slicing**

#### Code:

clear all;

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

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

SNEHA AGALE

40

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");

SNEHA AGALE

41

### OutPut:

SNEHA AGALE

42

### Pseudo Image:

#### Code:

pkg load image; clear all;

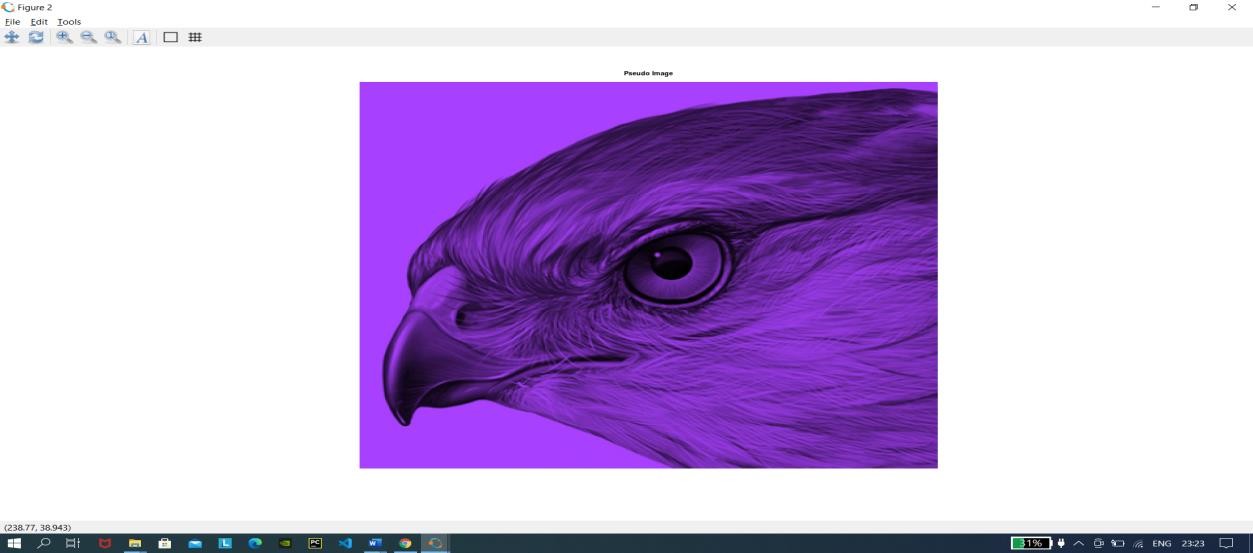
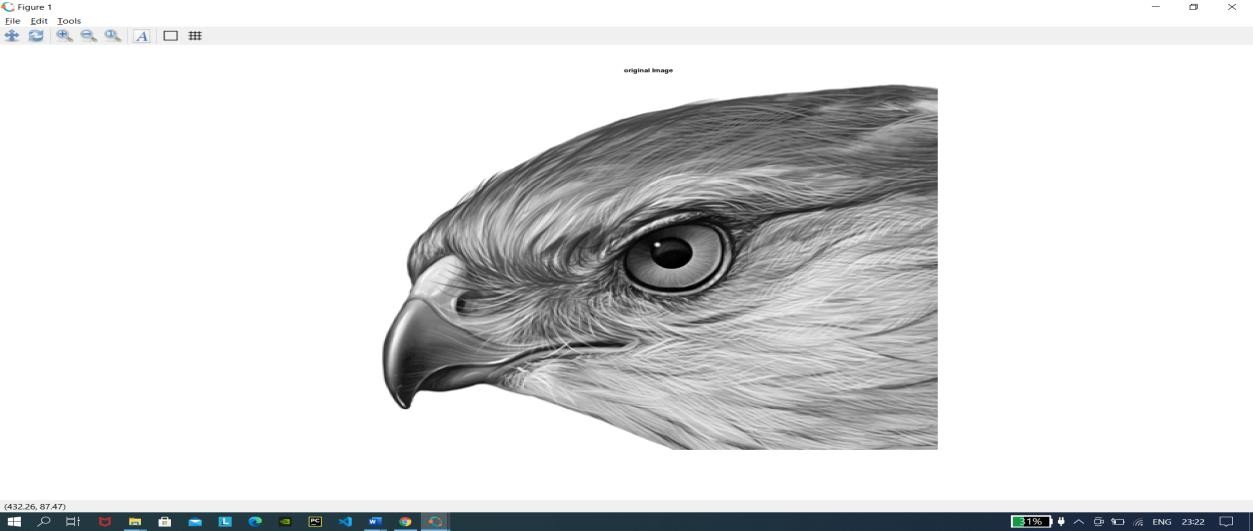
img = imread('hawk1.png'); % Read image figure, imshow(img);title("original Image"); red = 0.66\*img;

green=0.25\*img; blue = img;

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

figure, imshow(pseudo\_img), title('Pseudo Image');

### OutPut:



SNEHA AGALE

43

**Practical -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)

### OutPut:



SNEHA AGALE

44

##### repmat():

Repmat command repeats the elements of an array in output. Repetition is depended on parameter list, therefore every time we need to declare parameters within a bracket after repmat command.

Syntax:

##### Repmat(number,number of times) Huffmandict():

Generate Huffman code dictionary for source with known probability model. SYNTAX:

##### [[dict,avglen] = huffmandict(symbols,prob)](https://in.mathworks.com/help/comm/ref/huffmandict.html#d122e50782)

huffmandict([symbols,](https://in.mathworks.com/help/comm/ref/huffmandict.html#d122e50944)[prob](https://in.mathworks.com/help/comm/ref/huffmandict.html#mw_8454f6ac-0e06-4914-8780-b4a060166c01)) generates a binary Huffman code dictionary, dict, for the source symbols, symbols, by using the maximum variance algorithm. The input prob specifies the probability of occurrence for each of the input symbols. The length of prob must equal the

length of symbols. The function also returns average codeword length avglen of the dictionary, weighted according to the probabilities in the input prob.

##### Huffmanenco():

Encode sequence of symbols by Huffman encoding

##### SYNTAX:

[**code = huffmanenco(sig,dict)**](https://in.mathworks.com/help/comm/ref/huffmanenco.html#d122e51176)

[code](https://in.mathworks.com/help/comm/ref/huffmanenco.html#mw_3101491a-0082-4ca6-bb8e-f41a3552c432) = huffmanenco([sig,](https://in.mathworks.com/help/comm/ref/huffmanenco.html#d122e51319)[dict](https://in.mathworks.com/help/comm/ref/huffmanenco.html#mw_8375f27d-df67-434b-8b66-111d4bef63ce)) encodes input signal sig using the Huffman codes described by input code dictionary dict. sig can have the form of a vector, cell array, or alphanumeric cell array. If sig is a cell array, it must be either a row or a column. dict is an N-by-2 cell array, where *N* is the number of distinct possible symbols to encode. The first column

of dict represents the distinct symbols and the second column represents the corresponding codewords. Each codeword is represented as a row vector, and no codeword in dict can be the prefix of any other codeword in dict. You can generate dict using the [huffmandict](https://in.mathworks.com/help/comm/ref/huffmandict.html) function.

##### Huffmandeco():

Decode binary code by Huffman decoding. SYNTAX:

##### [sig = huffmandeco(code,dict)](https://in.mathworks.com/help/comm/ref/huffmandeco.html#d122e50478)

SNEHA AGALE

45

[sig](https://in.mathworks.com/help/comm/ref/huffmandeco.html#mw_f6897bf7-fbfb-47f4-a731-9d7df8e07a11) = huffmandeco([code,](https://in.mathworks.com/help/comm/ref/huffmandeco.html#d122e50635)[dict](https://in.mathworks.com/help/comm/ref/huffmandeco.html#mw_7cf75e06-46a8-425c-85b9-c3dfc90fd379)) decodes the numeric Huffman code vector, code, by using the Huffman codes described by input code dictionary dict. Input dict is an *N*-by-2 cell array, where *N* is the number of distinct possible symbols in the original signal that encodes code. The first column of dict represents the distinct symbols, and the second column represents the corresponding codewords. Each codeword is represented as a numeric row vector, and no codeword in dict can be the prefix of any other codeword in dict. You can generate dict by using the [huffmandict](https://in.mathworks.com/help/comm/ref/huffmandict.html) function and code by using the [huffmanenco](https://in.mathworks.com/help/comm/ref/huffmanenco.html) function. If all symbols

in dict are numeric, output sig is a vector. If any symbol in dict is alphabetic, sig is a one- dimensional cell array.

##### de2bi():

Convert decimal numbers to binary vectors. SYNTAX:

##### [b = de2bi(d)](https://in.mathworks.com/help/comm/ref/de2bi.html#d122e26788)

converts a nonnegative decimal integer d to a binary row vector. If d is a vector, the

output b is a matrix in which each row is the binary form of the corresponding element in d.

##### numel():

Number of elements in array or subscripted array expression

##### Syntax

* + **n = numel(A)**
  + **n = numel(A,varargin)**

**Description**

n = numel(A) returns the number of elements, n, in array A.

n = numel(A,varargin) returns the number of subscripted elements, n,

in A(index1,index2,...,indexn), where varargin is a cell array whose elements are index1, index2, ..., indexn.

SNEHA AGALE

46

### Watermarking:

##### Code:

pkg load image; clear all;

close all;

#Input Image where we want to apply watermark f=imread('lena\_gray\_256.tif');

#For watermarking, size of inputimage and watermarking image should be same #there for we changed the size of image using imresize and dispalyed

fr=imresize(f,[560 560]); figure;imshow(fr);

title('Original Image with resized');

#Watermarking Image w=imread('Sample1.png');

#Again Resized the Watermarking Image wr=imresize(w,[560 560]); figure;imshow(wr);

title('watermark');

#Applied watermarking alpha=0.7;

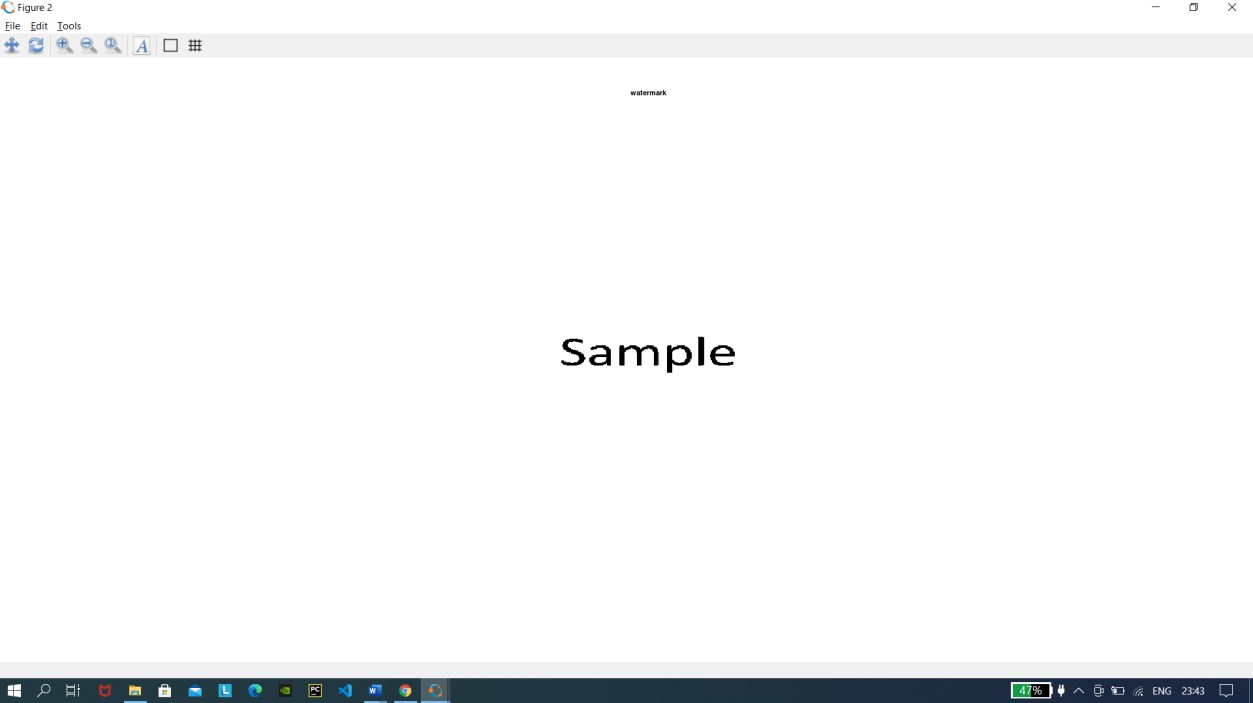
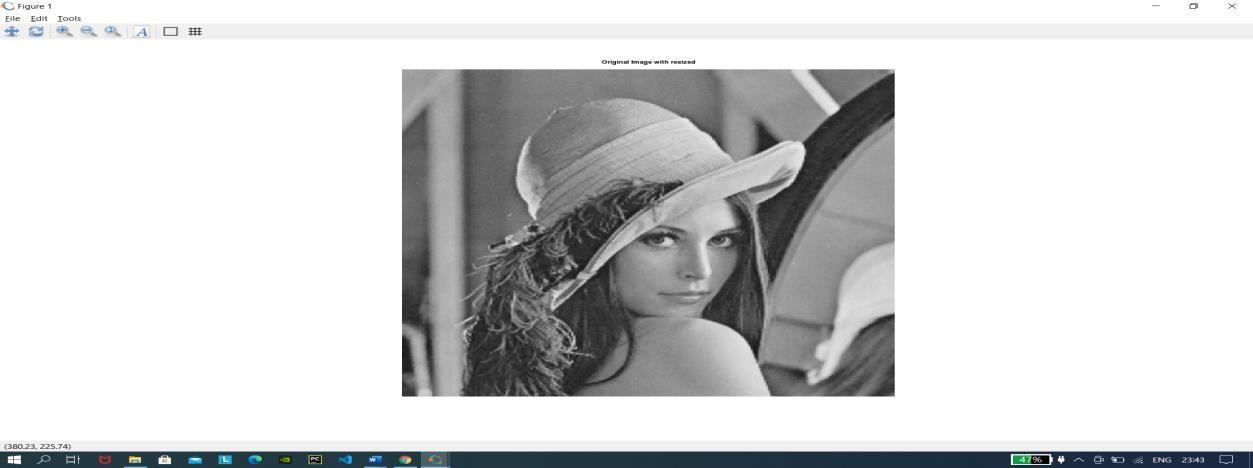
fw=(1-alpha)\*fr + alpha.\*wr;

SNEHA AGALE

47

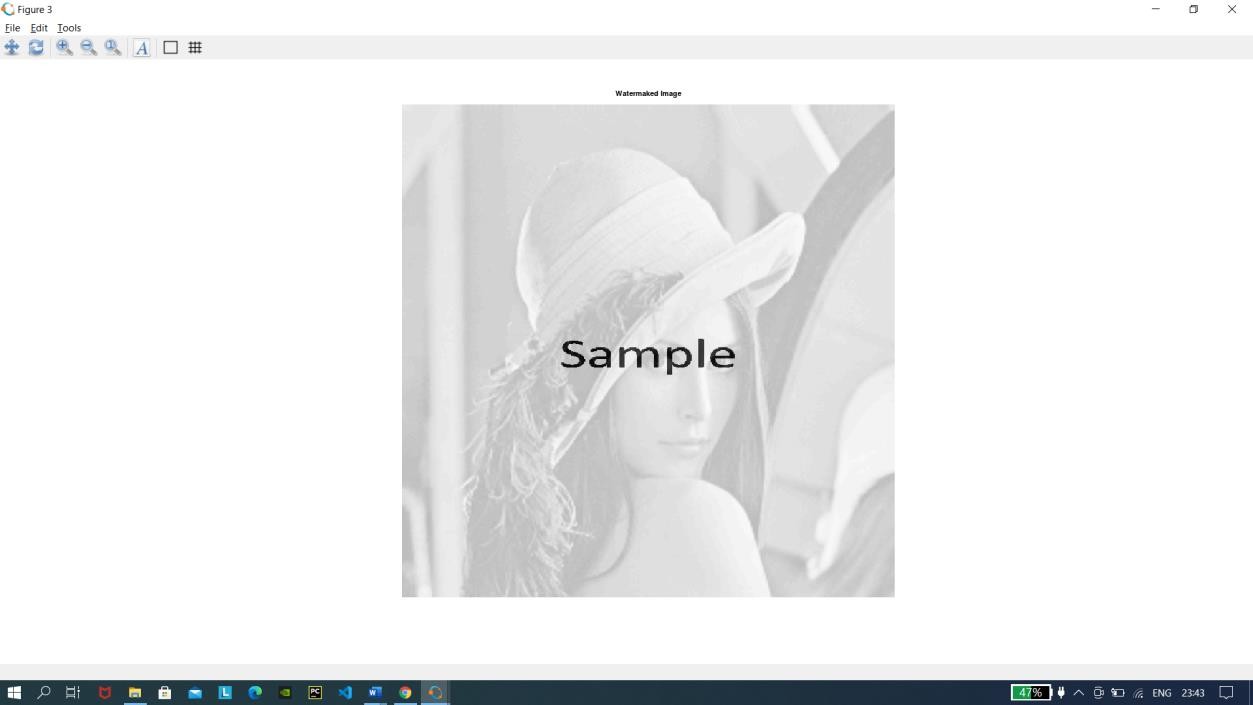
#Display the watermarked Image figure;imshow(fw); title('Watermaked Image');

### OutPut:



SNEHA AGALE

48



##### Imresize():

Resize image

[**B**](https://in.mathworks.com/help/matlab/ref/imresize.html#d122e653113) **= imresize(**[**A,**](https://in.mathworks.com/help/matlab/ref/imresize.html#d122e652513)[**scale**](https://in.mathworks.com/help/matlab/ref/imresize.html#d122e652552)**)** returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If A has more than two

dimensions, imresize only resizes the first two dimensions. If scale is in the range [0, 1], B is smaller than A. If scale is greater than 1, B is larger than A. By default, imresize uses bicubic interpolation.

SNEHA AGALE

49

# Practical 8

#### Basic Morphological Transformations

1. **Boundary Extraction**

Extracting the boundary is the important process to gain the information and understand the feature of an image. Boundary extraction is the first process in preprocessing in order to present the features of the image. This process can help the researcher to gain the data from the image.

Boundary Extraction in Octave

Let A be an Image matrix and B be a structuring element.

##### Formula for Boundary Extraction:

Steps to be followed:

* + Convert the image into binary image.
  + Perform Erosion:

Erode binary image A by structuring element B. (i.e) 

* + Subtraction:

Subtract the binary image A from the Eroded image.(i.e) 

##### imerode():

erodes the image.

[**J**](https://in.mathworks.com/help/images/ref/imerode.html#d122e125766) **= imerode(**[**I**](https://in.mathworks.com/help/images/ref/imerode.html#d122e125474)[**,SE**](https://in.mathworks.com/help/images/ref/imerode.html#d122e125517)**)** erodes the grayscale, binary, or packed binary image I, returning the eroded image, J. SE is a structuring element object or array of structuring element objects, returned by the [strel](https://in.mathworks.com/help/images/ref/strel.html) or [offsetstrel](https://in.mathworks.com/help/images/ref/offsetstrel.html) functions.

#### Code:

pkg load image; clear all;

close all; A=imread('tulips.png'); C=rgb2gray(A);

C(C<225)=0;

SNEHA AGALE

50

s=strel('disk',4,0);%Structuring element

D=~im2bw(C);%binary Image

F=imerode(D,s);%Erode the image by structuring element

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

figure,imshow(D);title('Binary Image');

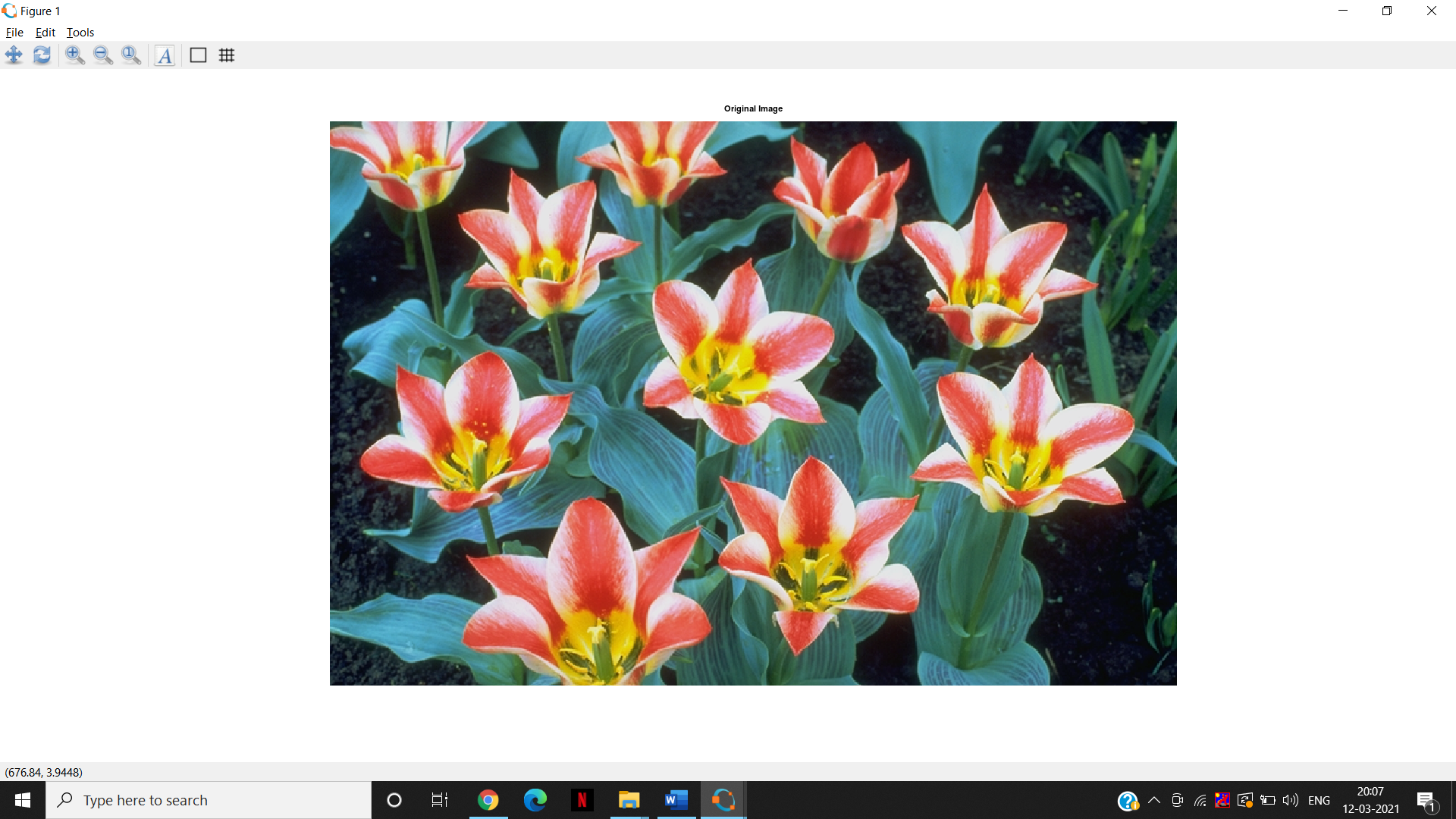
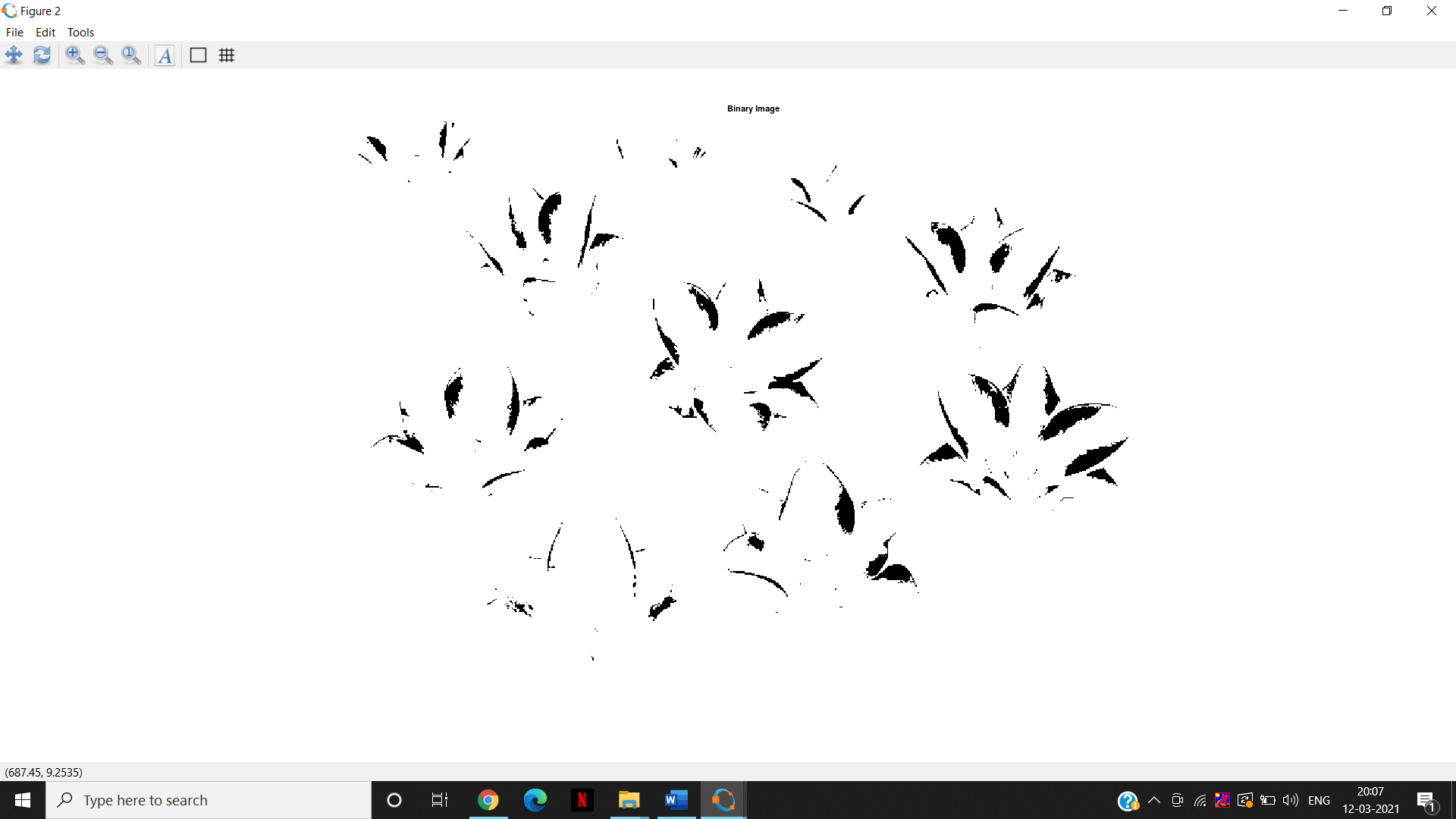
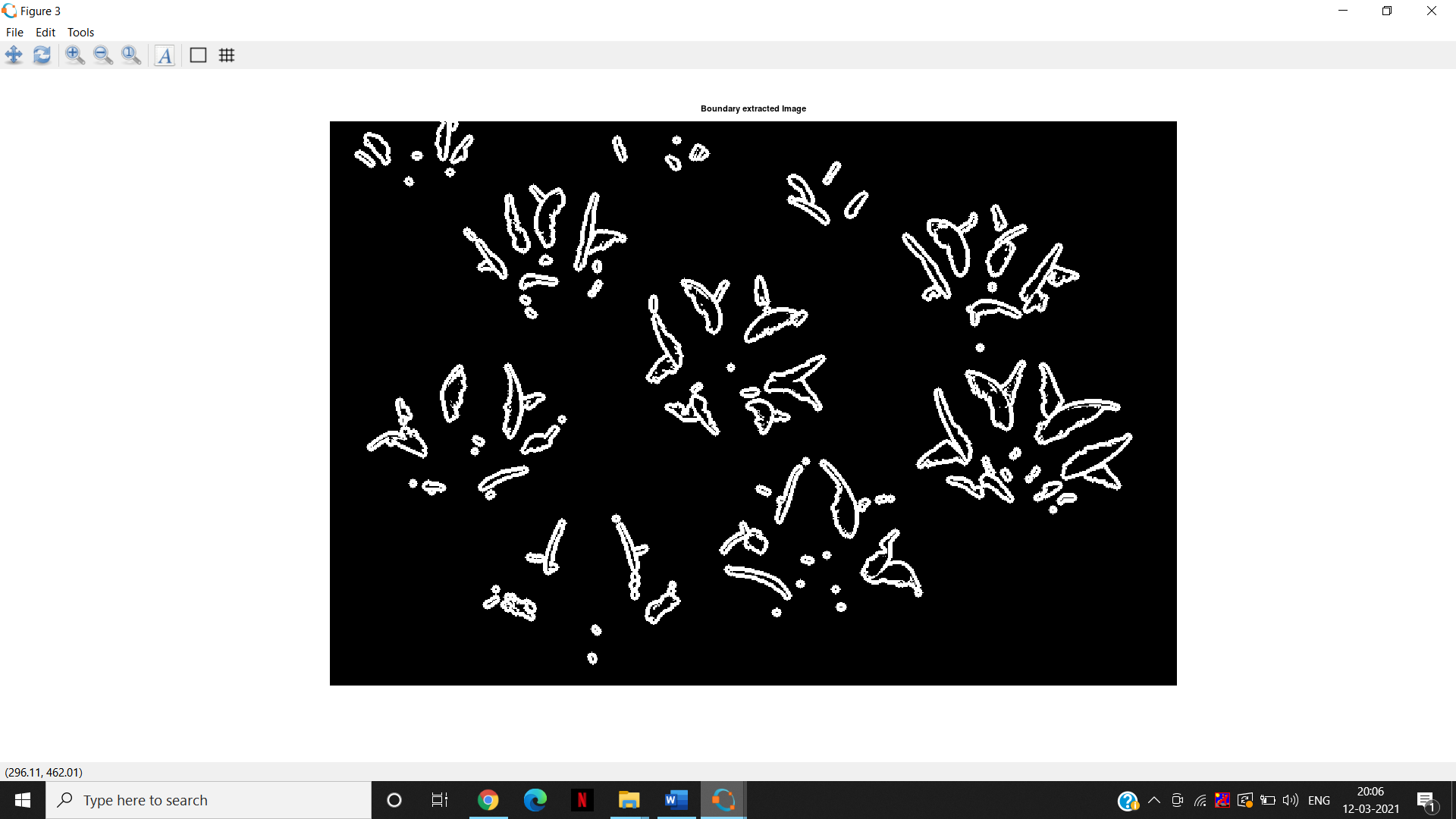
%Difference between binary image and Eroded image

figure,imshow(D-F);title('Boundary extracted Image');

SNEHA AGALE

51

**OutPut:**

SNEHA AGALE

52

#### Thining and thicking Code:

org\_im=imread("img.jpg"); subplot(2,2,1), imshow(org\_im);title("Original image"); binary=im2bw(org\_im);

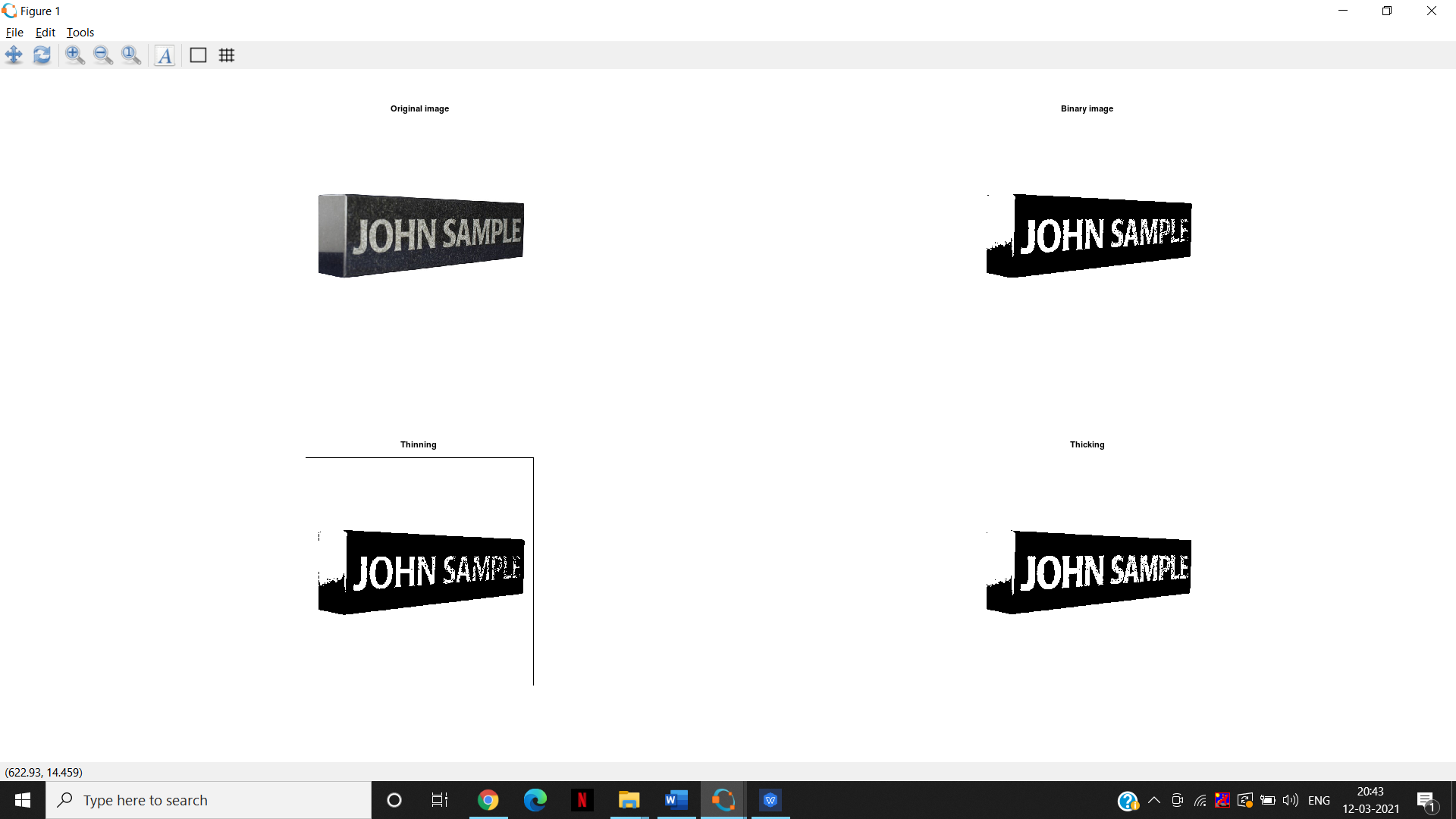
subplot(2,2,2), imshow(binary);title("Binary image");

thin=bwmorph(binary,'thin'); subplot(2,2,3), imshow(thin);title("Thinning "); thick=bwmorph(binary,'thicken'); subplot(2,2,4), imshow(thick);title("Thicking");

SNEHA AGALE

53

### OutPut:



##### bwmorph():

Morphological operations on binary images SYNTAX:

##### [BW2 = bwmorph(BW,operation)](https://in.mathworks.com/help/images/ref/bwmorph.html#d122e24066)

Operations can be ‘skel’ ,’thin’ ,’thicken’, ‘fill’, etc.

1. **Hole filling and sketoning**

**Code:** A=imread("coins.png"); B=im2bw(A); subplot(2,2,1)

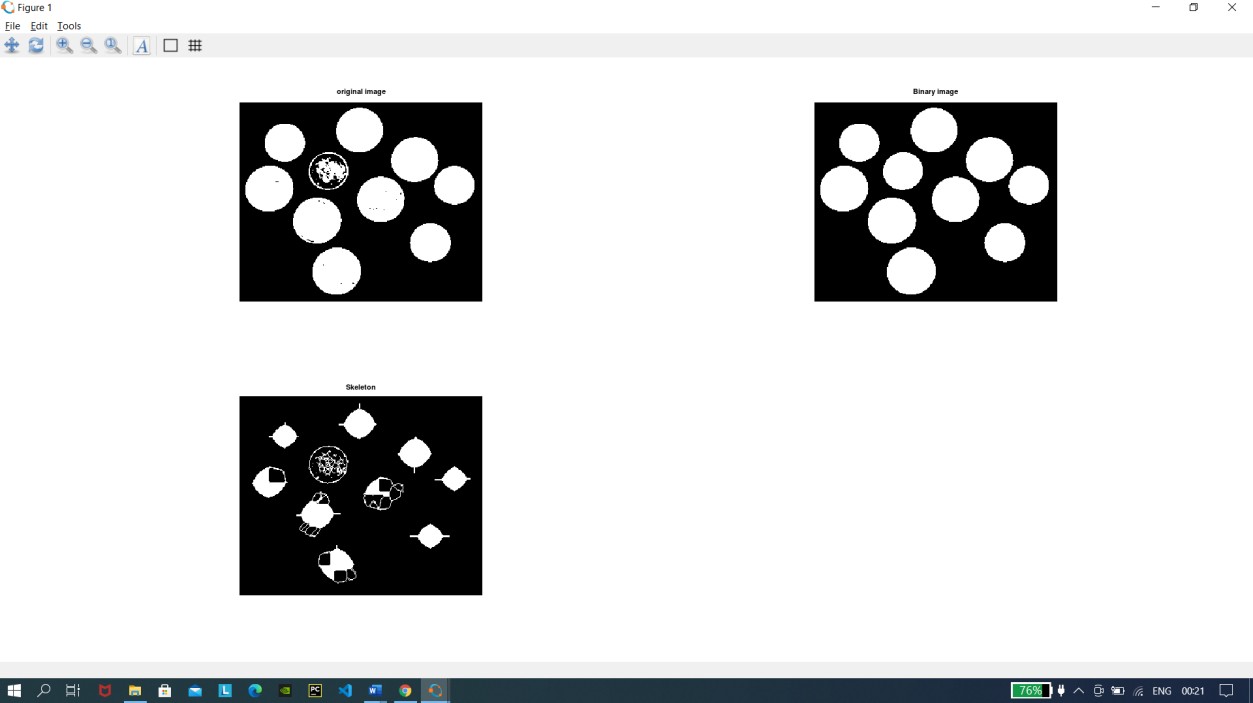
imshow(B);title("original image"); hole=bwfill(B,'holes'); subplot(2,2,2),

54

SNEHA AGALE

imshow(hole);title("Binary image"); skel=bwmorph(B,'skel',8); subplot(2,2,3), imshow(skel);title("Skeleton");

### OutPut:



SNEHA AGALE

55