****

**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

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

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2020 - 2021**

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

**Image vision and processing**

**Name: Uzma Siddiqui**

**Roll No.: 15**

**CERTIFICATE**

This is to certify that Miss. **Uzma Shehzad Anwar Siddiqui** with Roll No. **15** has successfully completed the necessary course of experiments in the subject of **Image vision and 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

|  |  |  |
| --- | --- | --- |
| **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   1. A) Low-pass Filter/Smoothing Filters   (Average, Weighted Average, Median and Gaussian)   1. 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  A) Implement Huffman Coding  B) Add a watermark to the image |  |
| 8. | Basic Morphological Transformations  A) Boundary Extraction  B) Thinning and Thickening  C) Hole filling and Skeletons |  |

**Practical 1: Implement Basic Intensity transformation functions**

**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('airplane.jfif'); #Reading Image

x=rgb2gray(x);               # Converting RGB image into Gray level

x=im2double(x);             #converting Image into Double

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

for i=1:row    #Reading rows value from starting to end and storing in[i] variable

  for j=1:col    #Reading columns 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);

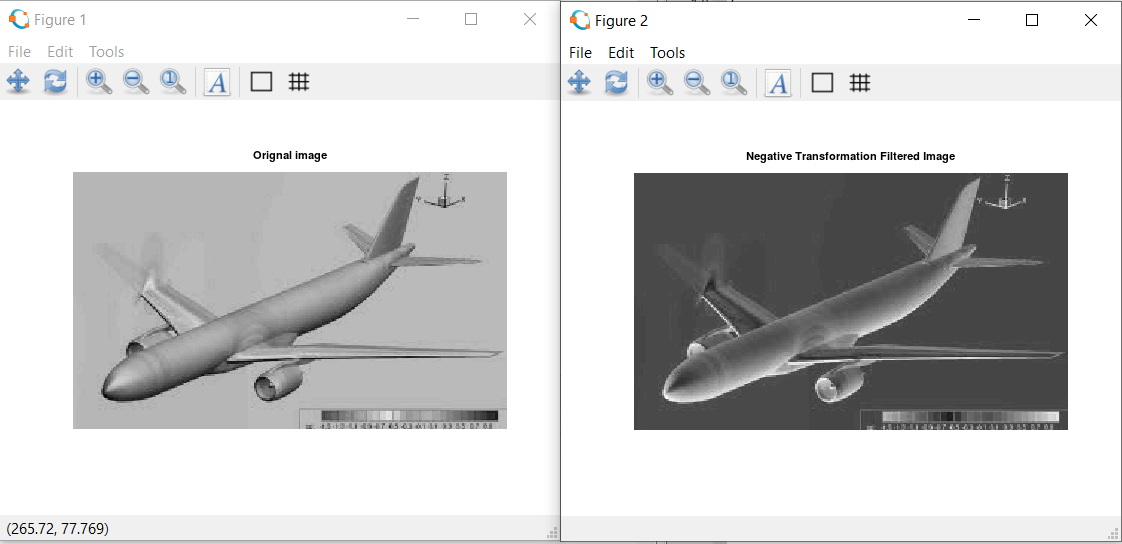
title("Orignal image");

figure

imshow(N);

title('Negative Transformation Filtered Image');

Output:



**Practical 1(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:**

x=imread('fruit.png');

x=rgb2gray(x);

x=im2double(x);

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

c=2;

for i=1:row    #Reading rows value from starting to end and storing in[i] variable

  for j=1:col    #Reading columns value from starting to end and storing in[j] variable

    N(i,j)=c\*log(1+x(i,j)); #Subtracting input matrix values from 255 and storing in new matrix[n]

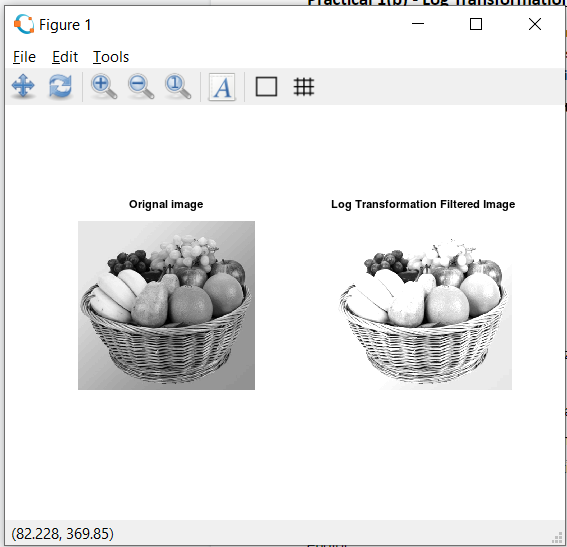
  endfor

endfor

subplot(1,2,1);imshow(x);title("Orignal image");

subplot(1,2,2);imshow(N);title('Log Transformation Filtered Image');

Output:

****

**Practical No. 1 ( 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 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;

a=imread('cameramen.tiff');

#ad=rgb2gray(a);

ad=im2double(a);

y=ad;

cc=1;

[r c]=size(ad);        #taking image size into matrix form.

for i=1:r     #Reading rows value from starting to end and storing in[i] variable

  for j=1:c     #Reading columns value from starting to end and storing in[j] variable

    y(i,j)=cc\*ad(i,j)^.6;  #Subtracting input matrix values from 255 and storing in new matrix[n]

  endfor

endfor

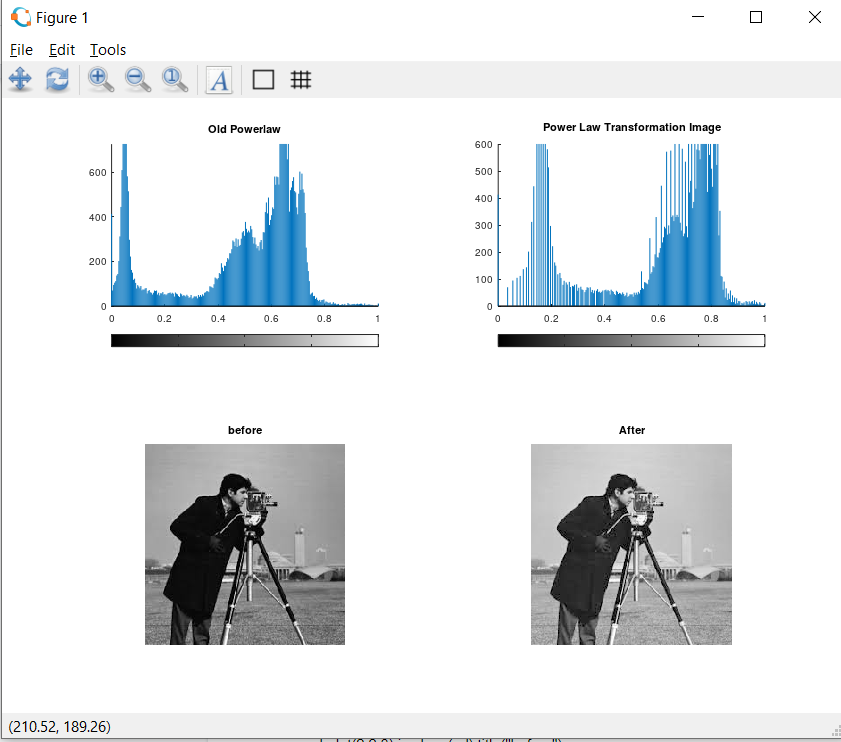
subplot(2,2,1);imhist(ad);title("Old Powerlaw");

subplot(2,2,2);imhist(y);title("Power Law Transformation Image");

subplot(2,2,3);imshow(ad);title("before");

subplot(2,2,4);imshow(y);title("After")

Output:



**Practical 2(a)-** Contrast Stretching

**Using imadjust function-**

a=imread("barbra.jfif");

#agray=rgb2gray(a);

ad=in2double(a);

subplot(2,2,1); imshow(ad); title("before");

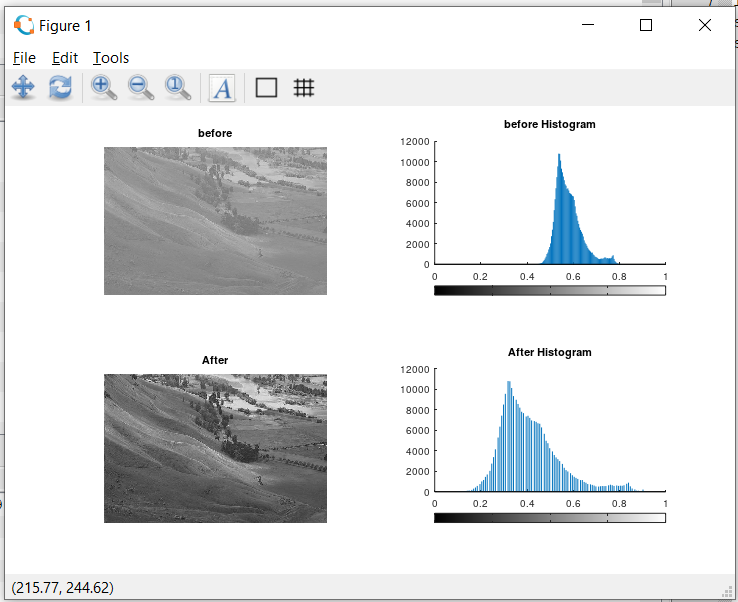
subplot(2,2,2); imhist(ad); title("before Histogram");

imad=imadjust(ad,(0.1 0.3),(0.2 0.8));

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

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

Output:



**Practical 2(a)-** Contrast Stretching

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

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);

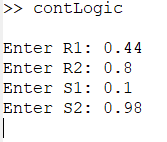
title("Contrast Streched Image");

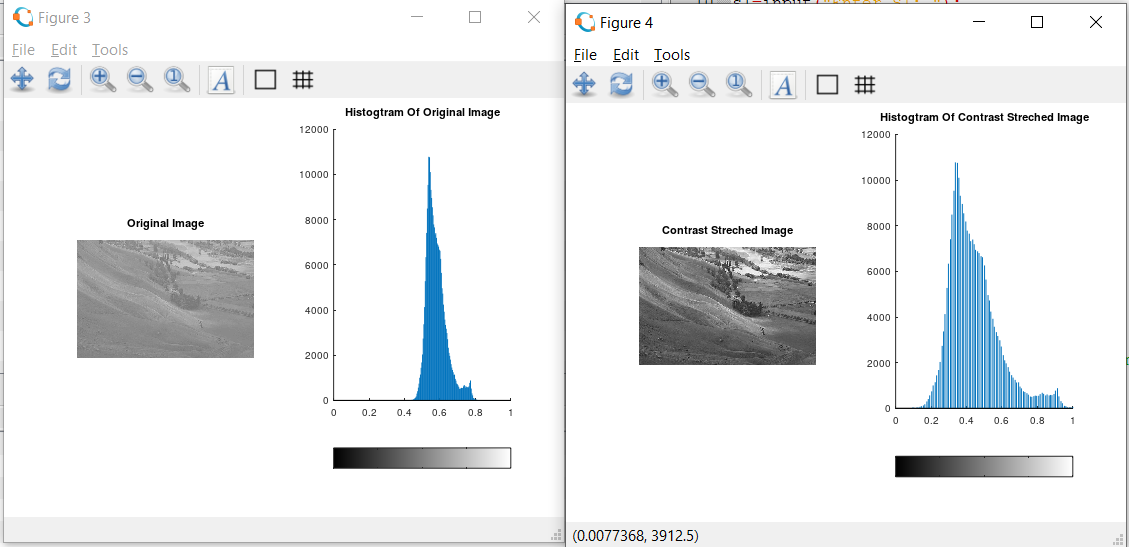
subplot(1,2,2)

imhist(s);

title('Histogtram Of Contrast Streched Image');

Output:



.

**Practical 2(b)-** Thresholding

Code-

pkg load image;

r=imread("baby.jpg");

r=rgb2gray(r);

threshold=150;  #here we take value for thresholding

[m n]=size(r);

OutImage=zeros(m,n);

#thresholding

for i=1:m

  for j=1:n

    if (r(i,j)>threshold)   #if dimension value is greater can user input then convert into 1

      OutImage(i,j)=1;

    else

      OutImage(i,j)=0;

    endif

  endfor

endfor

#Displaying The Orignal Image Histogram

figure

subplot(1,2,1)

imshow(r);

title("Orignal Image");

subplot(1,2,2)

imhist(r);

title("Histogram of Orignal Image");

#Displaying The Threshold Image Histogram

figure

subplot(1,2,1)

imshow(OutImage);

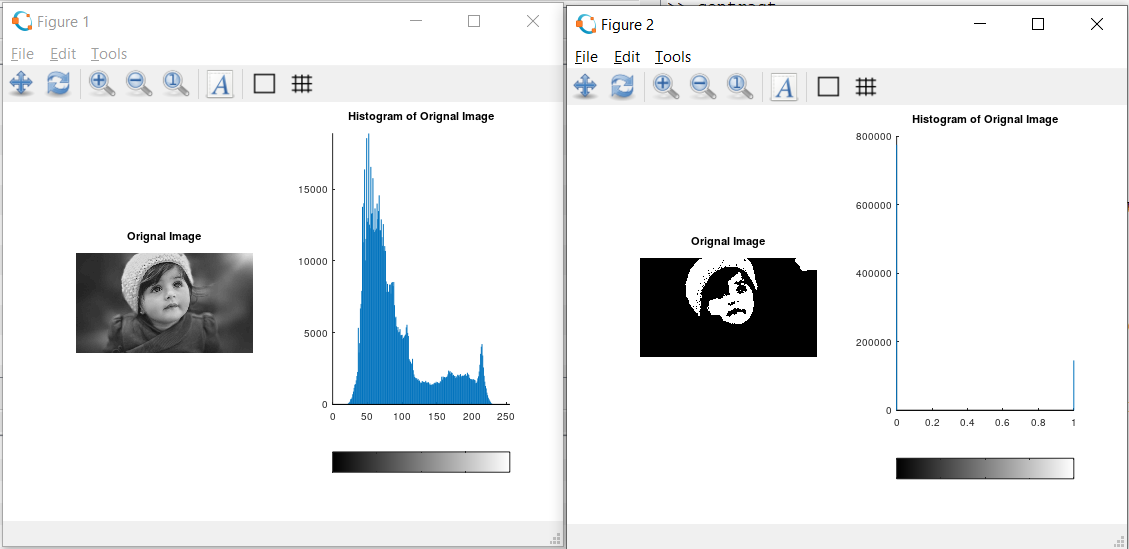
title("Orignal Image");

subplot(1,2,2)

imhist(OutImage);

title("Histogram of Orignal Image");

Output:



**Practical -2 ( C )**

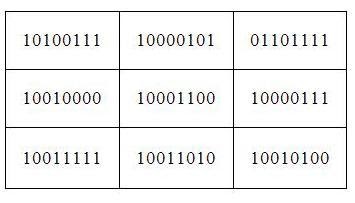
**Image reconstruction using n bit planes.**

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]

**MATLAB CODE:**

%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);

1. Explanation:

‘bitset’ is used to set  a bit at a specified position. Use ‘bitget’ to get the bit at the positions 7 and 8 from all the pixels in matrix A and use ‘bitset’ to set these bit values at the positions 7 and 8 in the matrix B.

**Code-**

clear all

close all

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);

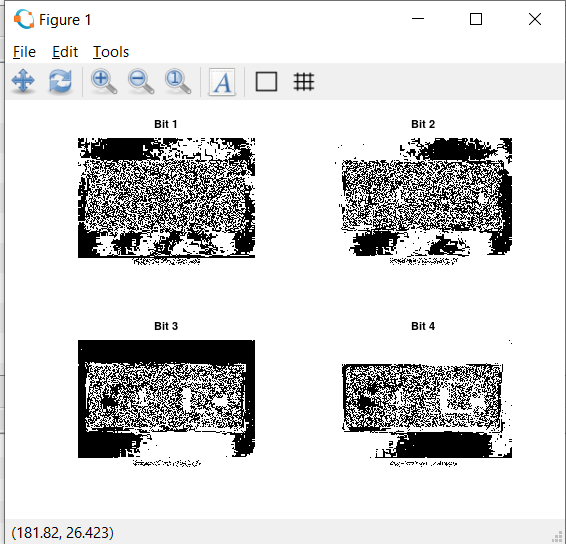
B=uint8(B);

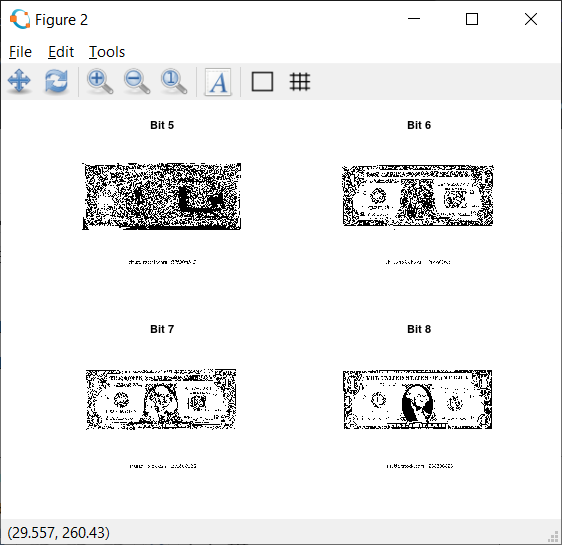
figure,

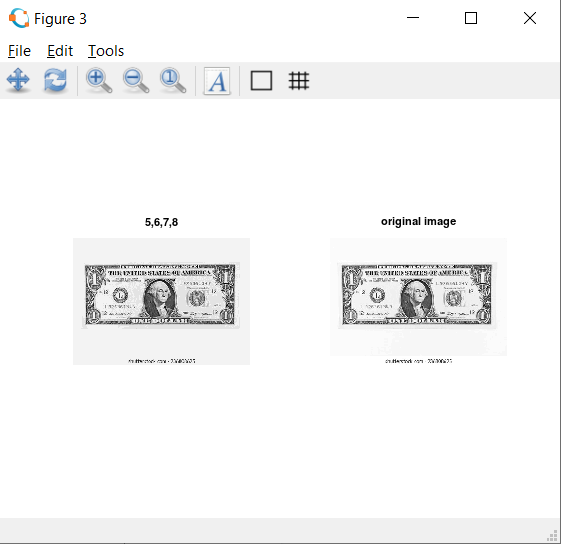
subplot(1,2,1),imshow(B); title("5,6,7,8")

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

Output:

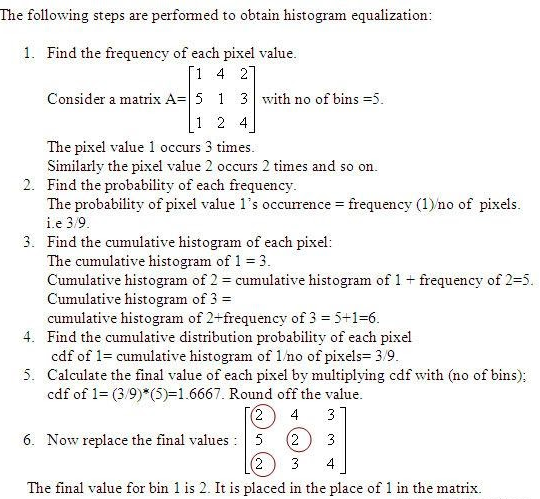






**Practical-3**

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



**Code:**

clear all;

close all;

pkg load image;

a=imread('histo.jpg');

#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');

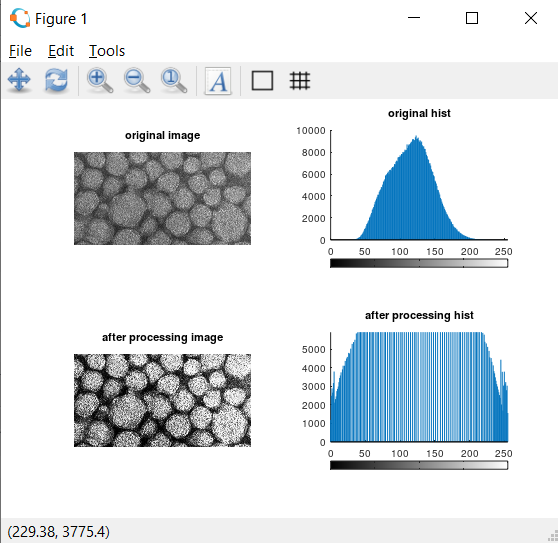
#he=histeq(a);

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

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

 #imhist(he);

Output:



**Practical -4 (Low Pass- Average filter using inbuilt functions)**

**Low Pass filtering:** It is also known as the smoothing filter. It removes the high-frequency content from the image. It is also used to blur an image. A low pass averaging filter mask is as shown.

1/9 1/9 1/9

1/9 1/9 1/9

1/9 1/9 1/9

**Code:**

clear all;

close all;

pkg load image;

a=imread('lowpassf.jpg');

sa=rgb2gray(a);

#imwrite(a,'hawk1.png');

c=im2double(sa);

r=imnoise(c,'salt & pepper' );

f=ones(3,3)/9;

af=filter2(f,r);

figure

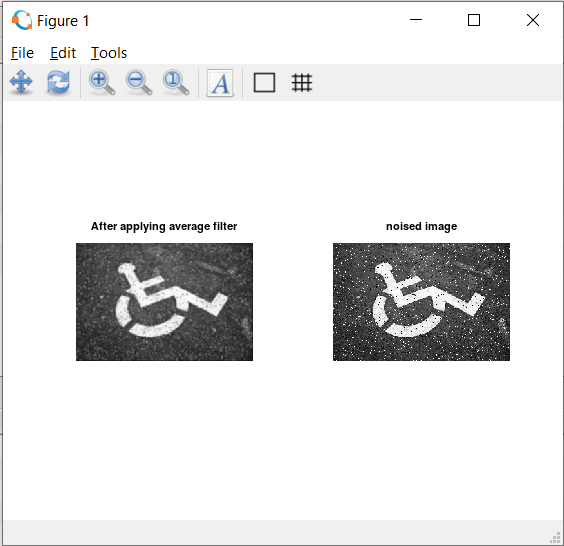
#imshow(a); title('original');

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

subplot(1,2,2)

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

Output:



**Practical-4 (Low pass- Average filter without using inbuilt functions)**

Code:

close all;

pkg load image;

im=imread('lowpassf.jpg'); % 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);

figure

subplot(1,2,1)

imshow(Nim);

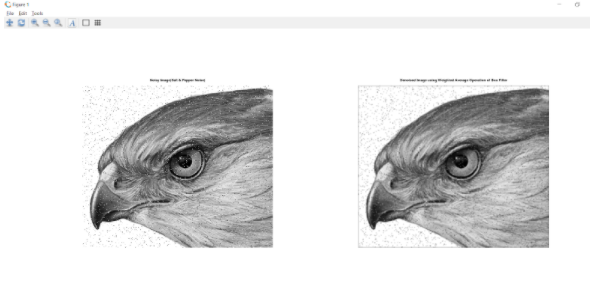
title('Noisy Image(Salt & Pepper Noise)');

subplot(1,2,2)

imshow(uint8(c));

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

Output:



**Practical No- 4 Low Pass Filter- (Median Filter)**

**Median Spatial Domain Filtering:** It is also known as nonlinear filtering. It is used to eliminate salt and pepper noise. Here the pixel value is replaced by the median value of the neighboring pixel.

**Code:**

  pkg load image;

# Read the image

 a=imread('lowpassf.jpg');

 r=rgb2gray(a);

 img\_noisy1=imnoise(r,'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

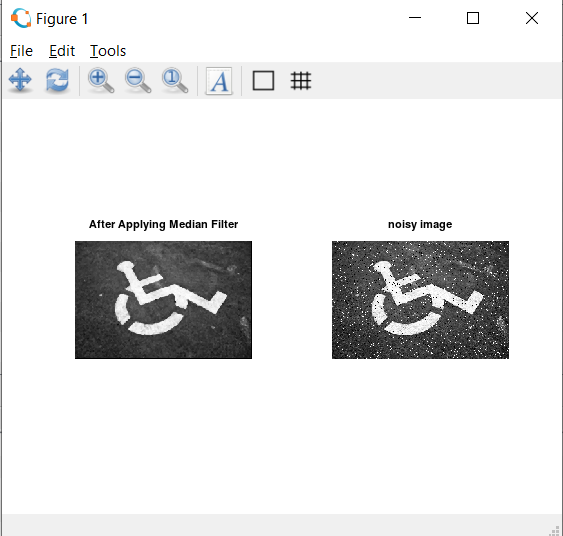
img\_new1 = uint8(img\_new1);

figure

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

subplot(1,2,2); imshow(img\_noisy1);title('noisy image');

Output:



**Practical – 4  Second order derivative – The Laplacian Filter**

The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for https://lh3.googleusercontent.com/_ibDFVf0sgfvOBZZpI2813k8YjooXrpCv9csRH6NWftFagoVGZVpg93jFY0oiEgqdEwNJKwjKtqOIbvAGnRiPjf756ZKktJfMJll3rMmnbOMiab974bN7Fd7H3kq8dZks5Ke9JENedge detection.The operator normally takes a single graylevel image as input and produces another graylevel image as output.

The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

Eqn:eqnlog1

This can be calculated using a convolution filter.

**Code:**

%Input Image

clear all;

A=imread('coin.jpg');

 r=rgb2gray(A);

size(r);

figure,

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

%Preallocate the matrices with zeros

I1=r;

I=zeros(size(r));

I2=zeros(size(r));

%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

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

r=double(r);

size(r);

%Implementation of the equation in Fig.D

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

    for j=1:size(r,2)-2

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

    end

end

I=uint8(I);

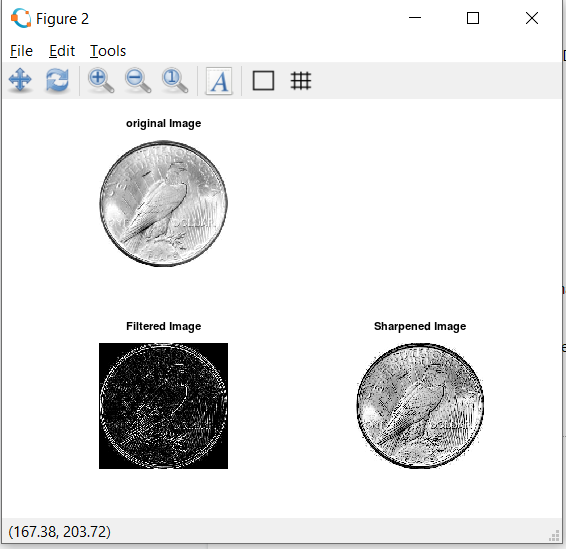
subplot(2,2,3);imshow(I);title('Filtered Image');

%Sharpenend Image

B=I1-I;

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

Output:



**Practical-4 First Order Derivative -Sobel Operator for edge detection without using edge function**

**Code:**

clear all;

A=imread('coin.jpg');

g=rgb2gray(A);

figure,

subplot(1,2,1); imshow(g); title('Original');

C=double(g);

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);

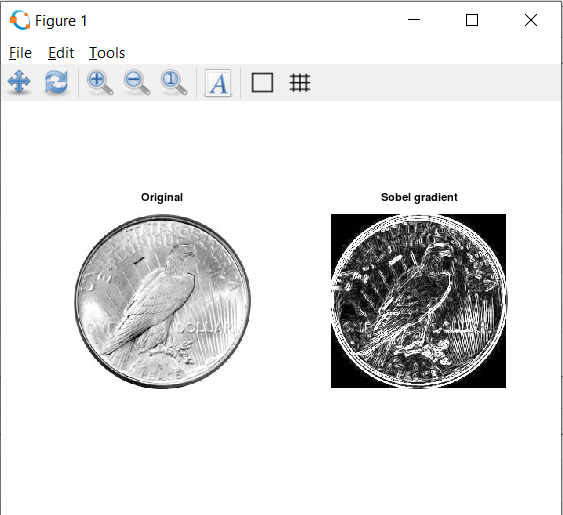
        g(i,j)=sqrt(Gx.^2+Gy.^2);

    end

end

subplot(1,2,2); imshow(g); 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("fruitnew.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);

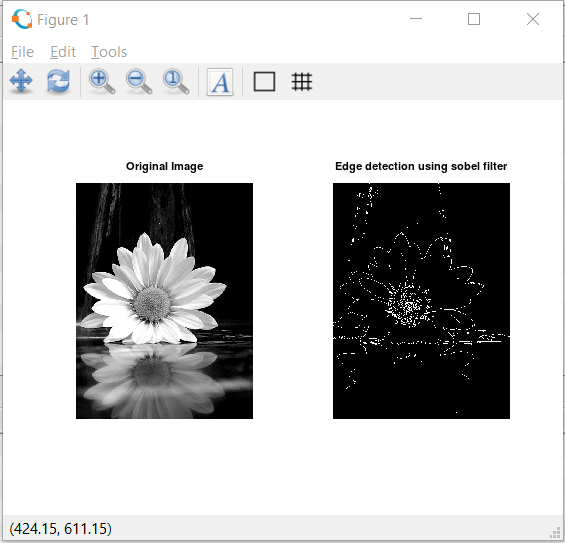
title('Edge detection using robert filter');

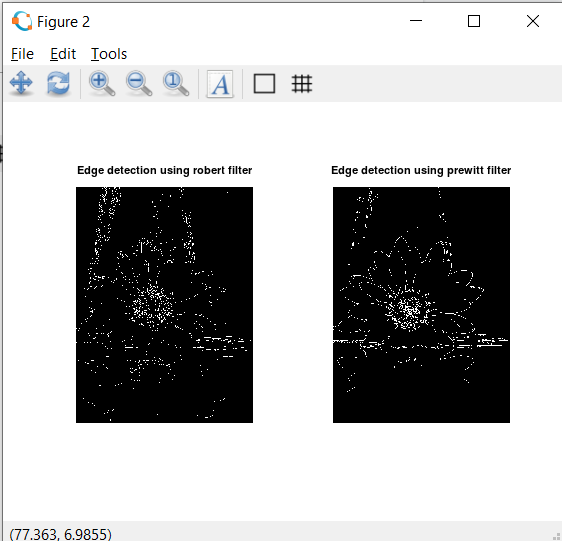
subplot(1,2,2)

imshow(prewitt);

title("Edge detection using prewitt filter");

Output:





**Practical no. 6 Color Image Processing**

1. **A) Pseudocoloring**

**Pseudocoloring** is a technique to artificially assign colors to a grey scale.The idea is to perform 3 transformations on a particular grey level and feed this to the three color inputs (RGB) of a color monitor. The result is a composite **image** whose color content depends on the grey level to color transformations.

**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));

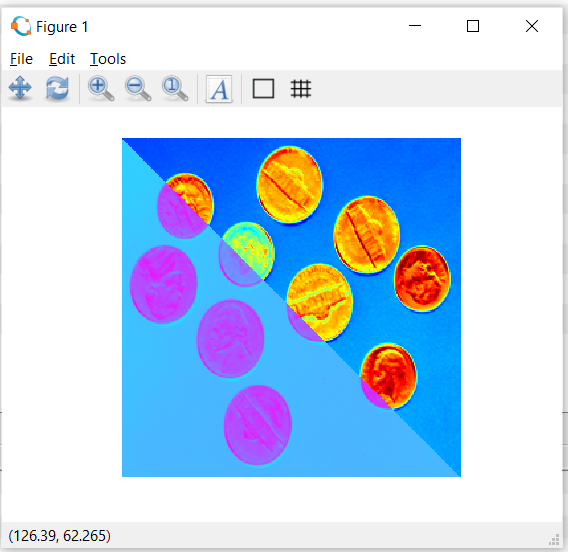
        Output(x(i),y(i),3)=B1(x(i),y(i));

end

Output = im2uint8(Output);

%FINAL IMAGE

imshow(Output);



**Intensity slicing**

**Intensity** level **slicing** means highlighting a specific range of **intensities** in an **image**. In other words, we segment certain gray level regions from the rest of the **image**

**Code :**

clear all;

#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;

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),[]);

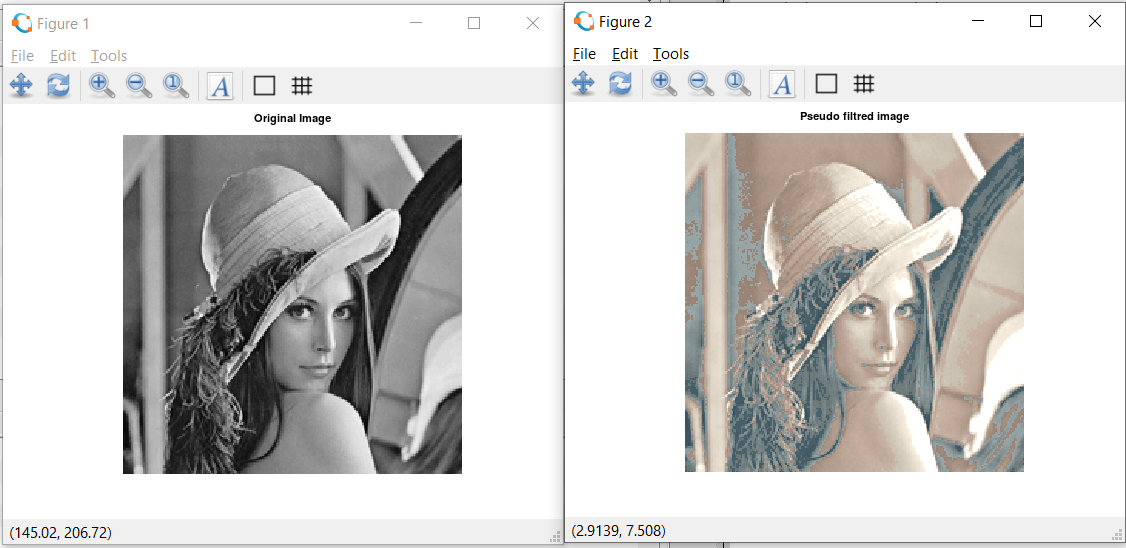
title('Original Image');

figure,

imshow(uint8(m),[]);

title("Pseudo filtred image");

Output:



**Pseudo Image**

pkg load image;

clear all;

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

figure, imshow(img);title("original Image");

red = 0.66\*img;

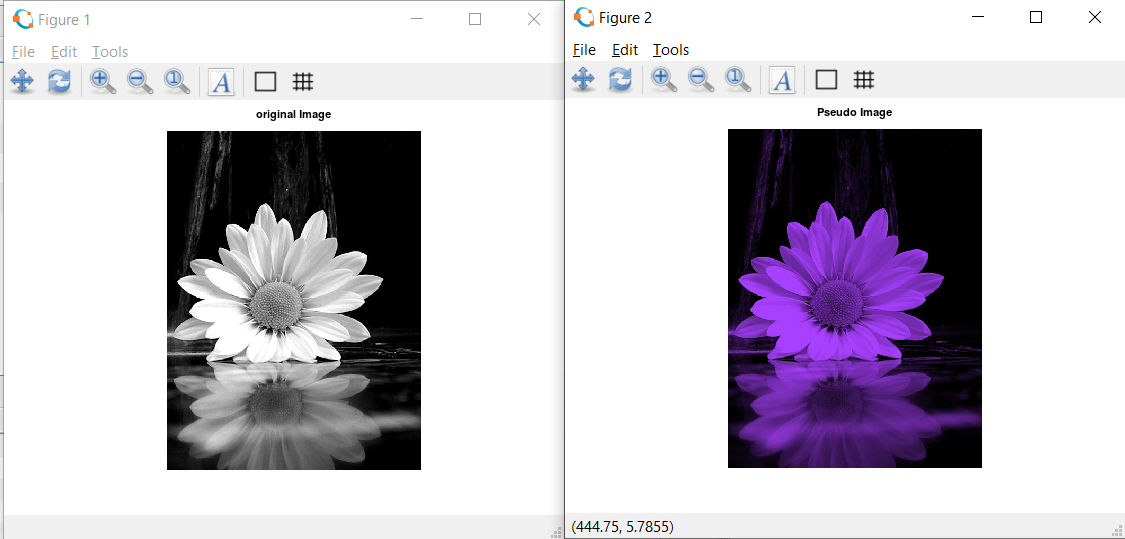
green=0.25\*img;

blue = img;

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

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

Output:



**Practical -7  Image Compression Techniques and watermarking**

**Implement Huffman Coding:**

When applying **Huffman encoding** technique on an **Image**, the source symbols can be either pixel intensities of the **Image**, or the output of an intensity mapping function. where numpix is the number of occurrence of a pixel with a certain intensity value and totalnum is the total number of pixels in the input **Image**.

**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);

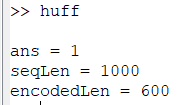
isequal(sig,dhsig)

binarySig = de2bi(sig);

seqLen = numel(binarySig)

binaryhcode = de2bi(hcode);

encodedLen = numel(binaryhcode)



**Watermarking**

**Digital image watermarking** is the method in which data is embedded in a multimedia file such as an **image** or a video, so as to verify the credibility of the content or the identity of the owner.

**Code:**

pkg load image;

clear all;

close all;

#Input Image where we want to apply watermark

f=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 dispalyed

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

figure;imshow(fr);

title('Original Image with resized');

#Watermarking Image

w=imread('watersample.jpg');

#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;

#Display the watermarked Image

figure;imshow(fw);

title('Watermaked Image');

**Output:**

