CHEMBUR TROMABY EDUCATION SOCIETY

N. G. ACHARYA & D. K. MARATHE COLLEGE OF ARTS, SCIENCE & COMMERCE

N. G. Acharya Marg, Chembur, Mumbai – 400 071.

NAAC ACCREDITED COLLEGE

M.Sc (Computer Science /Information Technology)

Certificate

Certified that the work entered in this journal was done in the Computer laboratory by the student Mr. / Miss Dalvi Siddhesh Dinesh Roll No. A1-5 Of Class M.Sc IT & A Semester II During the Year 2021-2022 in a Satisfactory manner.

Lecture Incharge	*External Examiner*	*Head of Dept*

INDEX

1. Basics

A) Aim : Program to study the effects of reducing the spatial resolution of a digital image.

Software : Matlab – Image Processing Tool Box

Theory :

A digital image is a two-dimensional array of size *M x N* where *M* is the number of rows and *N* is the number of columns in the array. A digital image is made up of a finite number of discrete picture elements called a pixel. The location of each pixel is given by coordinates (x, y) and the value of each pixel is given by intensity value *f*. Hence, the elements in a digital image can be represented by *f(x, y)*.

Sampling is the principal factor determining the spatial resolution of the image. Spatial resolution is defined as the smallest discernible detail in an image. The term spatial resolution corresponds to the total number of pixels in the given image. If the number of pixels is more, then the resolution of the image is more. Measuring the discernible changes in gray level s highly subjective process. It is possible to have considerable discretion regarding the number of samples used to generate the digital image.

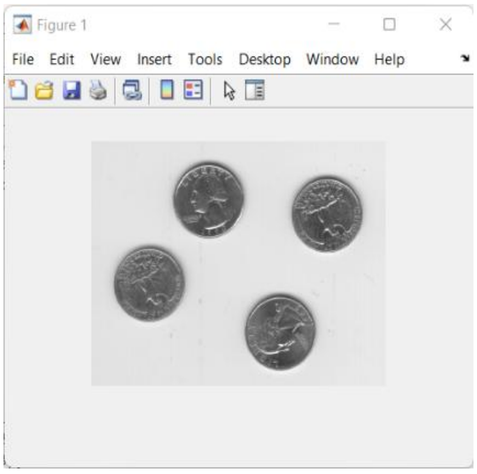
The sub sampling of the image is accomplished by deleting the appropriate number of rows and columns from the original image by deleting every other row and column. In first level sub sampling, it is very difficult to differentiate between the original and sub sampled image. These images can be compared by reconstructing the sub sampled image and bringing it back to the original image.

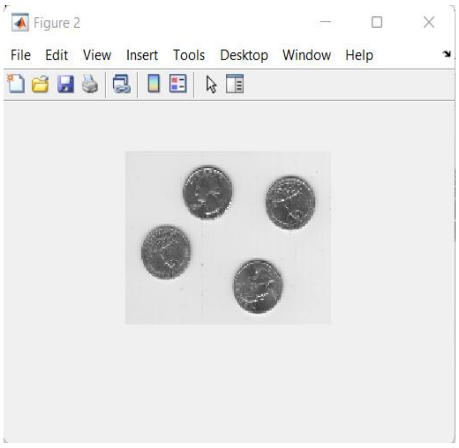
CODE :-

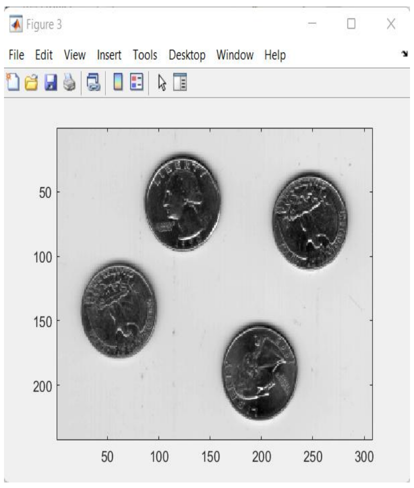
```
%Down sampling % Shrinking the image to 1/2
clc;
close all;
clear all;
a = imread('eight.tif');
[row, col]= size(a);
i=1;
j=1;
for x=1:2:row
for y=1:2:col
c(i,j)= a(x,y);
j= j+1;
end
j=1;
i= i+1;
end
figure(1), imshow(a)
figure(2), imshow(c)
figure(3),
imagesc(a),colormap(gray)
figure(4)
```

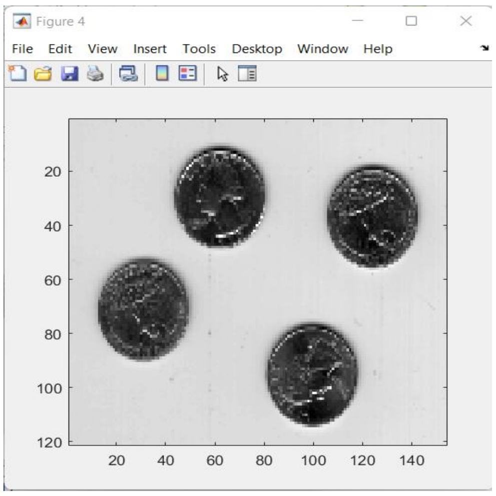
```
imagesc(c),colormap(gray)
```

OUTPUT:-









B) Aim : Program to study the effects of varying the number of intensity levels in a digital image.`

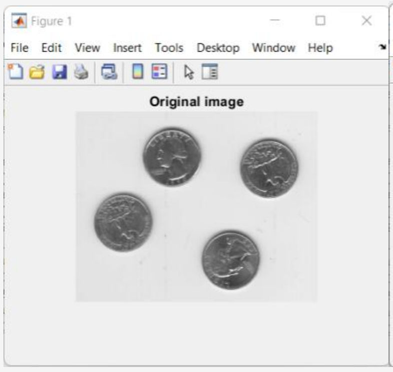
Theory :

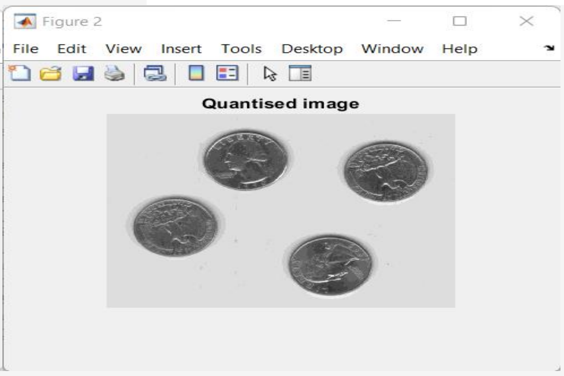
Gray level resolution is defined as the smallest discernable changes in gray level. It is possible to have considerable discretion regarding the number of samples used to generate the digital image, but this is not true for the number of gray levels. The number of gray levels will be in powers of 2 due to hardware restrictions.

In gray level resolution, as the number of gray levels are reduced, the details present in the goes on reducing. Sometimes it even leads to false contouring. In both the cases, it is not possible to get the original image after reconstruction.

CODE

```
close all;
clc;
i= imread('eight.tif');
%To quantize the given image
i=double(i);
i=i+1;
j=max(max(i));
a=input('how many levels you want to
have 1 2 4 8');
b=j/(2^a);
c=floor(i/(b+1));
c1=(c*255)/max(max(c));
% normalizing
figure(1);
imshow(uint8(i));
title('Original image');
figure(2);
imshow(uint8(c1));
title('Quantised image');
```





C]Aim: Program to perform image arithmetic operations.

Theory :

Image arithmetic is the implementation of standard arithmetic operations, such as addition, subtraction, multiplication, and division, on images. Image arithmetic has many uses in image processing both as a preliminary step in more complex operations and by itself. For example, image subtraction can be used to detect differences between two or more images of the same scene or object.

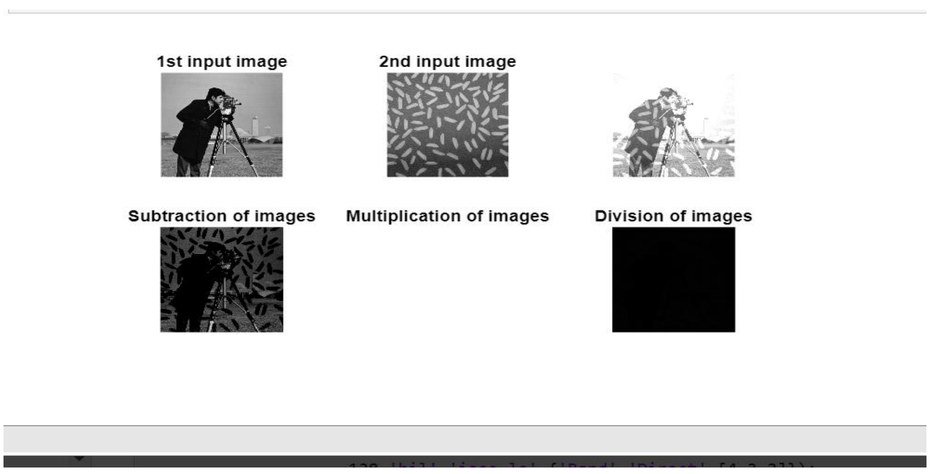
Image arithmetic can be done using the MATLAB® arithmetic operators. The Image Processing Toolbox™ software also includes a set of functions that implement arithmetic operations for all numeric, nonsparse data types. The toolbox arithmetic functions accept any numeric data type, including uint8, uint16, and double, and return the result image in the same format. The functions perform the operations in double precision, on an element-by-element basis, but do not convert images to double-precision values in the MATLAB workspace. Overflow is handled automatically. The functions clip return values to fit the data type.

CODE :-

```
clc;
clear all;
close all;
a=imread('cameraman.tif');
subplot(3,3,1);
imshow(a)
title('1st input image')
b=imread('rice.png');
subplot(3,3,2);
imshow(b)
title('2nd input image')
c=imadd(a,b);
subplot(3,3,3);
imshow(c)
d=imsubtract(a,b);
subplot(3,3,4);
imshow(d)
title('Subtraction of images')
e=immultiply(a,b);
subplot(3,3,5);
imshow(e)
title('Multiplication of images')
f=imdivide(a,b);
```

```
subplot(3,3,6);
imshow(f)
title('Division of images');

%%%Another way to add two images

I1=imread('cameraman.tif');
I2=imread('rice.png');
I3= I1+I2;
subplot(3,3,1);
 imshow(I1)
subplot(3,3,2);
 imshow(I2)
subplot(3,3,3);
 imshow(I3)
title('Addition of images')
%%%Another way to subtract two images
I1=imread('penguin.jpg');
I2=imread('lena.jpg');

I3=I1-I2;

subplot(3,3,1);
figure;
 imshow(I1)
subplot(3,3,2);
 figure;
imshow(I2)
subplot(3,3,3);
 figure;
 imshow(I3)
title('Addition of images')
```

OUTPUT:-



PRACTICAL NO 2. Intensity transformation and Spatial Filtering

A) Aim : **Program to perform Basic Intensity Transformation functions**:

i. Image Negation

ii. Thresholding.

iii. Log transformation

iv. Power-law transformations

v. Piecewise linear transformations

  a. Contrast Stretching

  b. Gray-level slicing with and without background.

  c. Bit-plane slicing

Theory :

Enhancing an image provides better contrast and a more detailed image as compare to non-enhanced image. It is used to enhance medical images, images captured in remote sensing, images from satellite. The transformation

function has been given below
s = T (r) where r is the pixels of the input image and s is the pixels of the output image.
T is a transformation function that maps each value of r to each value of s.
Basic Gray level transformation operations are: Image Negatives, Thresholding, Log Transformations, Power Law Transformations Piecewise Linear Transformations- Contrast stretching, Intensity Level Slicing, Bit Plane Slicing.

CODE:-

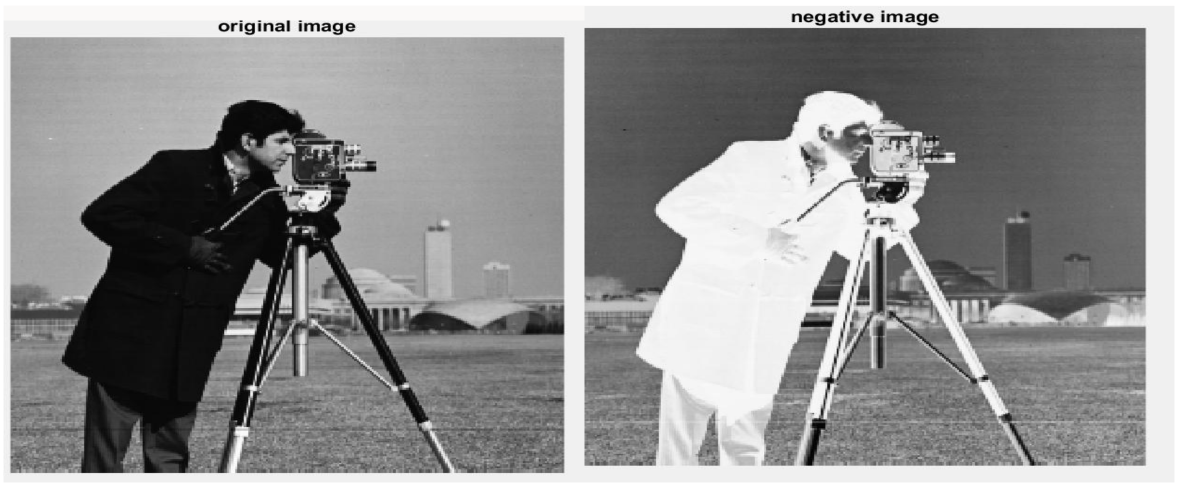
```
clc;
close all;
clear all;
%Read Input Image
a=imread('cameraman.tif');
figure(1);
imshow(a);
title('original image');
%negative image
%b=rgb2gray(a);
%figure(2);
%imshow(b);
b=255-a;
figure(3);
imshow(b);
title('negative image');

%logtransform
l=255;
d=l/log10(1+l);
e=d*log10(1+double(a));
f=uint8(e);
figure(4);
imshow(f);

% title('log transform');
%power law transform
gamma = 1.1;
g=double(a).^gamma;
figure(5);
imshow(uint8(g));
title('power law transform');
image thresholding
[r,c] = size(a);
Thre = input('Enter the value of Thr =');
for i = 1:r
   for j=1:c
      if (a(i,j)>Thre)
         Out(i,j) = 1;
      else
         Out(i,j) = 0;
      end
   end
end
subplot(121),
imshow(a);
title('Input Image');
subplot(122),
imshow(Out);
title('Threshold Image');
```

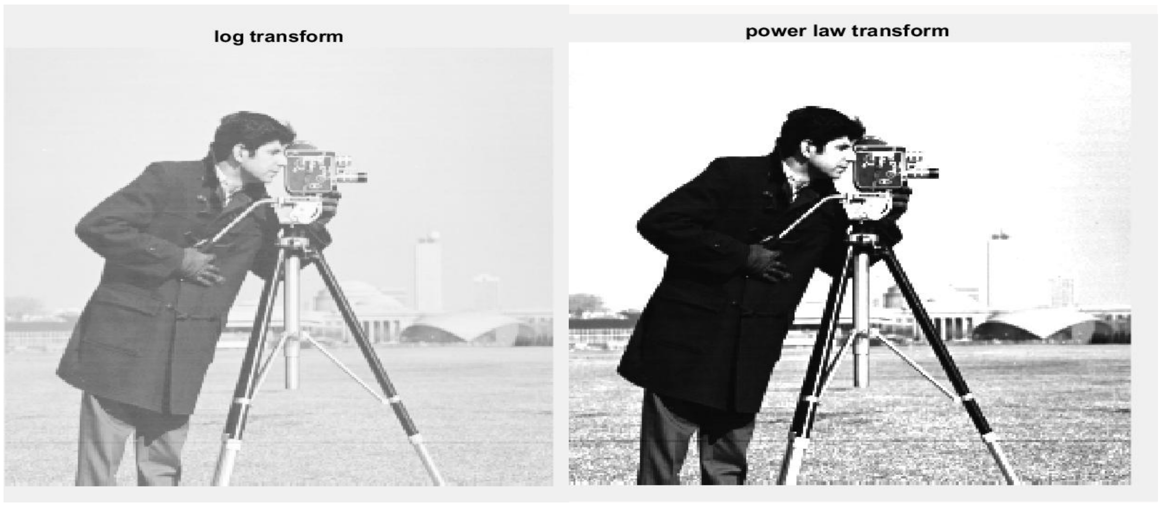
```
%inensity slicing with background
p = imread('cameraman.tif');
z = double(p);
[r,c] = size(z);
for i = 1:1:r
   for j=1:1:c
      if ((z(i,j) > 50)) && (z(i,j) < 150)z(i,j) = 255;
      else
         z(i,j) = p(i,j);
      end
   end
end


%Contrast stretching (using inbuilt functions)
a=imread('pout.tif');
s= imadjust(a,stretchlim(a),[]);
subplot(221),imshow(a);
title('Input Image');
subplot(222),imshow(s);
title('Stretched Image');
subplot(223),imhist(a);
title('Hist of Input Image');
subplot(224),imhist(Out);
title(' Hist ofThreshold Image');
%Contrast stretching

r=imread('pout.tif');
[m,n] = size(r);
r1 = input('enter r1 : ');
r2 = input('enter r2 : ');
s1 = input('enter s1 : ');
s2 = input('enter s2 : ');
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;
      end
   end
end
         z(i,j) = 255;
      else
         z(i,j)= 0;
      end
   end
end
```

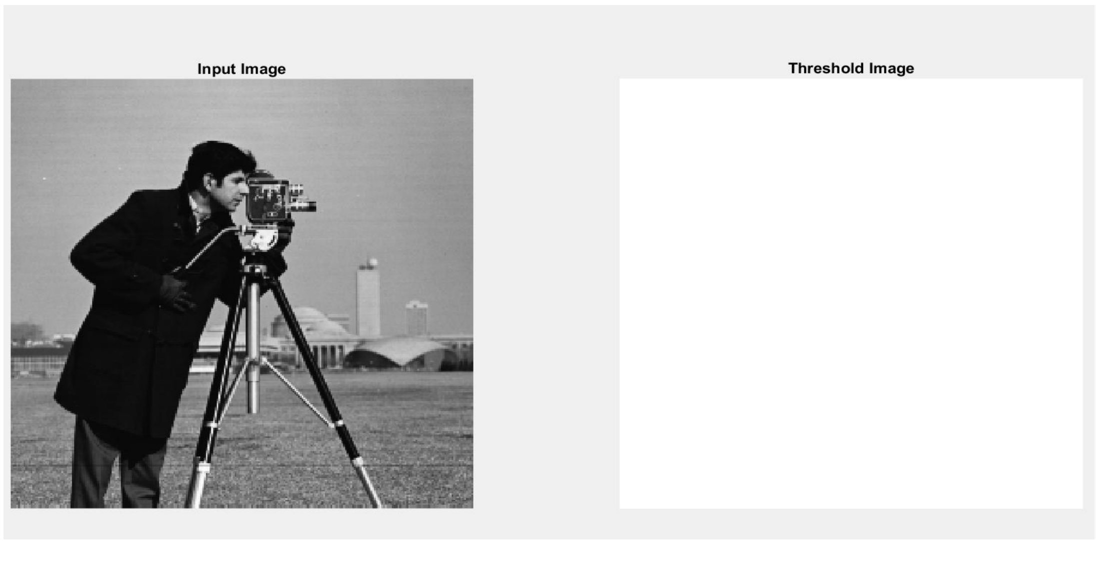
```
subplot(2,2,1);
imshow(p);
title('original img');
subplot(2,2,2);
imshow(uint8(z));
title('intensity slicing without BG');

figure;
imshow(uint8(r));
title('original img');
figure(1)
imshow(uint8(s));
title('contrast stretching img');
inensity slicimg without background
p = imread('rice.png');
z = double(p);
[r,c] = size(z);
for i = 1:1:r
    for j=1:1:c
        if ((z(i,j) > 50)) && (z(i,j) < 150)
figure (1);
subplot(2,2,1)

title('original img');
subplot(2,2,2)
imshow(uint8(z));
title('intensity slicing with BG');
```

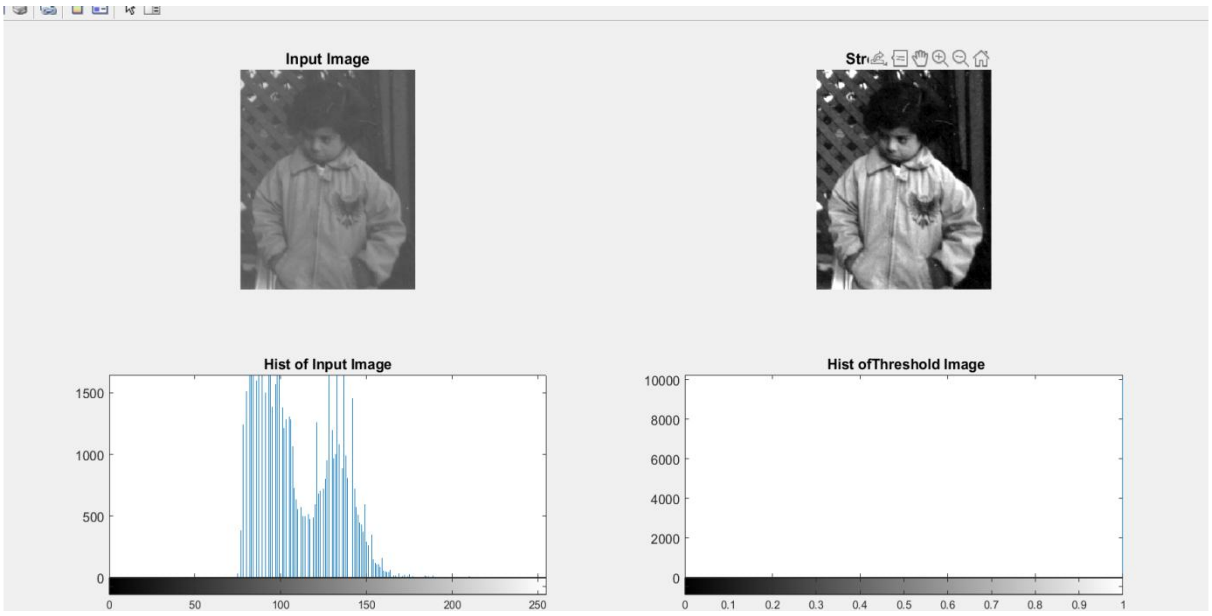
```
%bit plane extraction
a = imread('cameraman.tif');
a = double(a);
[x,y] = size(a);
I = input('enter bit image u want to see  1= MSB 8=
LSB');
for i = 1:1:x
    for j=1:1:y
        c = dec2bin(a(x,y),8);
        d = c(I);
        w(x,y)=double(d);
        if  w(x,y)== 49;
            w(x,y)=255;
        else
            w(x,y)=0;
        end
    end
end
subplot(2,2,1);
imshow(uint8(a));
title('original img');
subplot(2,2,2);
imshow(uint8(a));
title("Bit plane extraction image")
```

OUTPUT

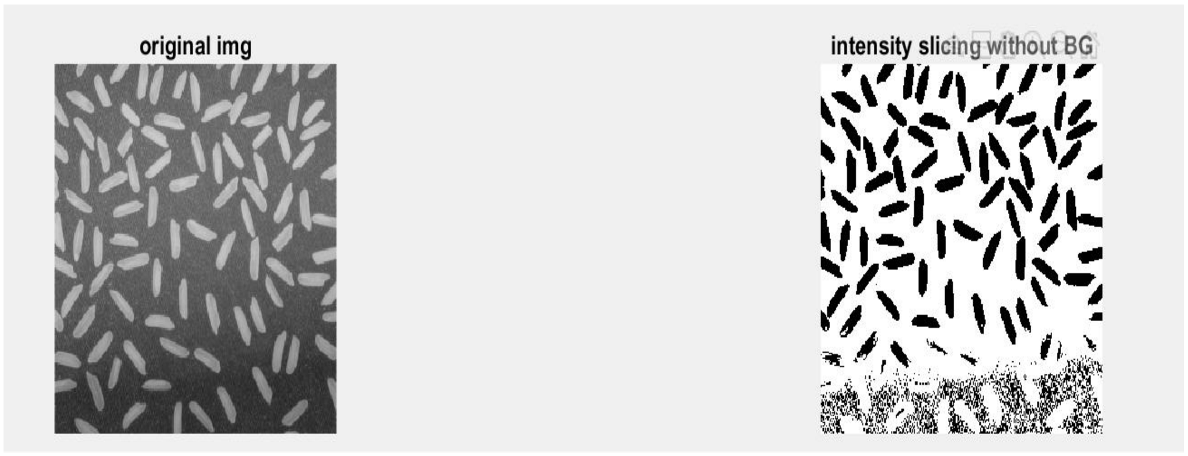

original image
negative image

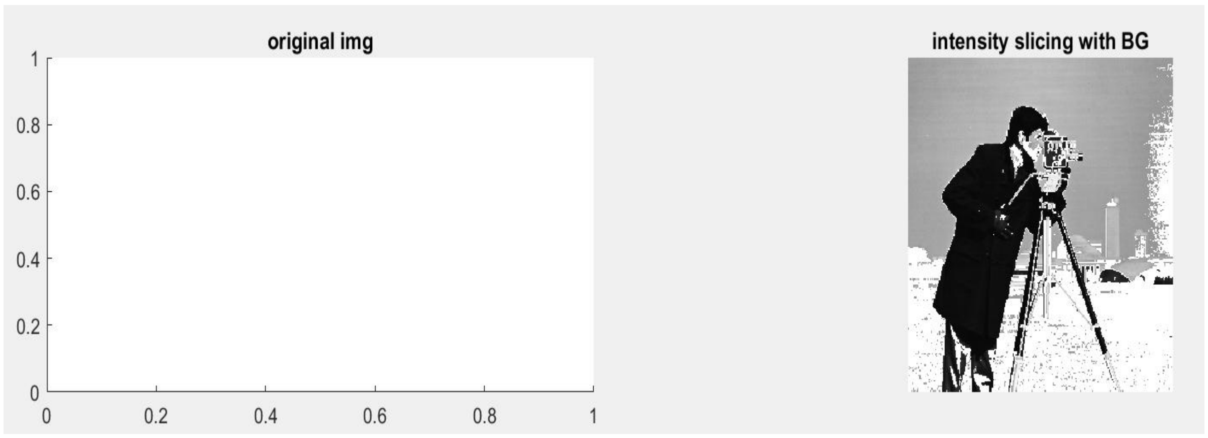

log transform
power law transform

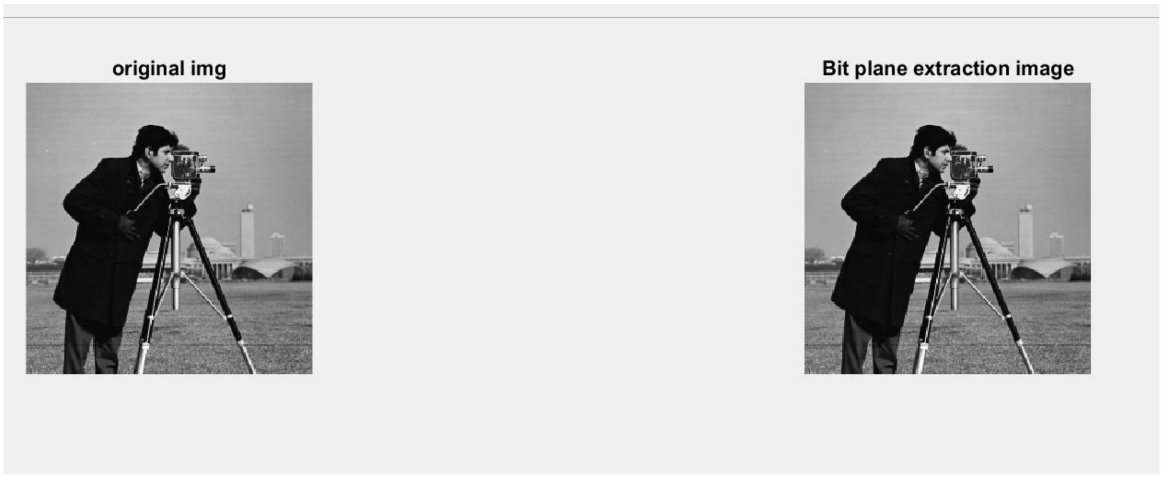

Input Image
Threshold Image


Input Image
Str
Hist of Input Image
Hist ofThreshold Image
1500
1000
500
0
0
50
100
150
200
250
10000
8000
6000
4000
2000
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

contrast stretching img


original img
intensity slicing without BG


original img
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
intensity slicing with BG


original img
Bit plane extraction image

B) Aim : Program to plot the histogram of an image and categorise.

Theory:

Histogram of an image with L total possible intensity levels in the range [0,G] is defined as the discrete function **h(rk) = nk** where rk is the kth intensity level in the interval [0,G] and nk is the number of pixels in the image whose intensity level is rk. A histogram shows us the distribution of grey levels in the image. Histograms are the basis for numerous spatial domain techniques. Histogram manipulation can be used for image enhancement. In histogram equalisation, the pixels are made to occupy the entire range of possible intensity levels and in addition the pixels are made to be distributed uniformly. The net effect will be an image that shows a great deal of gray level detail and has high dynamic range.

CODE:-

```
clc;
clear all;
close all;
 a=imread('tire.tif')
 %a=imread('C:\deer1.jpg');
 %perform histogram equalization
 b=histeq(a);
 subplot(2,2,1),
 imshow(a),
 title('original image'),
 subplot(2,2,2),
 imshow(b),
 title('After histogram equalization'),
 subplot(2,2,3),
 imhist(a),
 title('original histogram');
 subplot(2,2,4),
  imhist(b),
 title('After histogram equalization')

img = imread('cameraman.tif');
figure(1);
imshow(img);
```

```
%img = rgb2gray(img);
[x,y] = size(img);
%Create frequency array of size 256
frequency = 1:256;
count = 0;
for i = 1:256
    for j = 1:x
        for k = 1:y
            if img(j,k) == i-1
                count = count + 1;

 end
        end
        frequency(i) = count;
        count = 0;
    end
end
    n = 0:255;
    stem(n, frequency);
    %grid on;
    ylabel("No of pixels with such
intensity level");
    xlabel("Intensity level");
```

OUTPUT :-

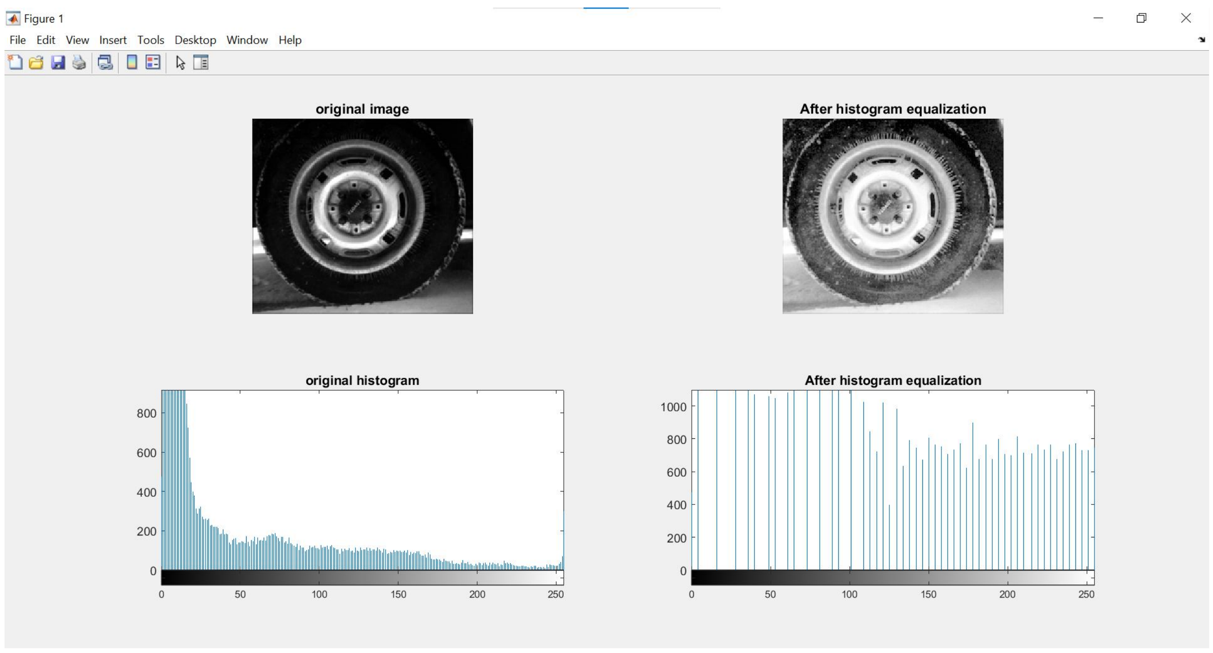
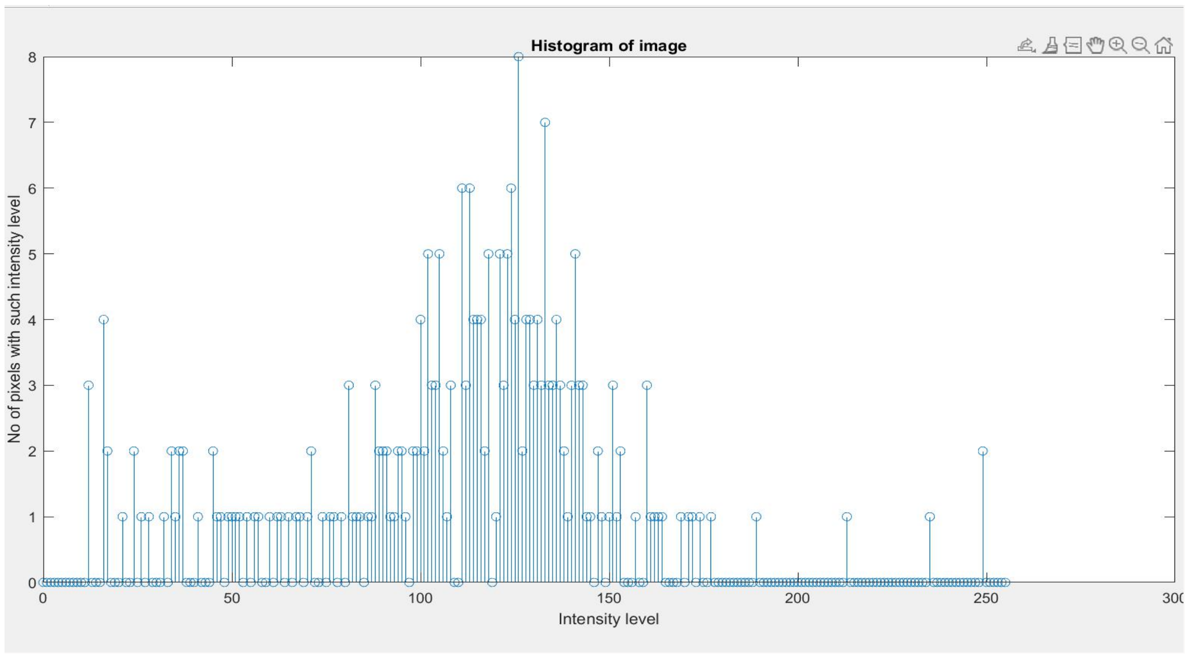


Figure 1
File Edit View Insert Tools Desktop Window Help
original image
After histogram equalization
original histogram
After histogram equalization



Histogram of image
No of pixels with such intensity level
Intensity level

B) Aim: Program to apply histogram equalization.

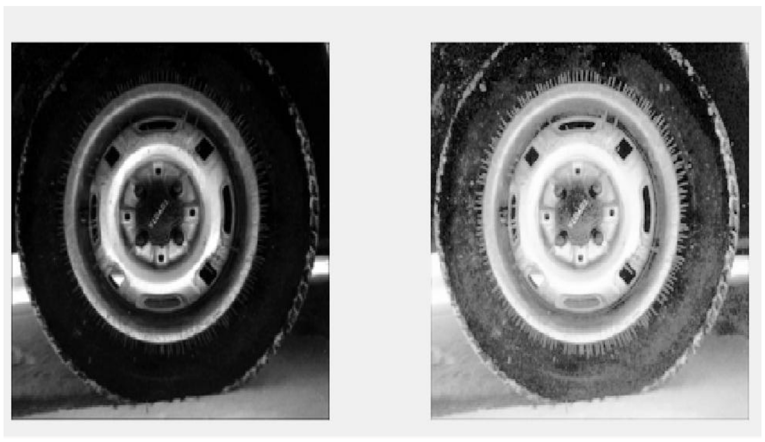
Theory :

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.

CODE:-

```
A=imread('tire.tif');
subplot(1,3,1);
imshow(A);
%Specify the bin range (0,255)
bin  = 255;
%Find the histogram of image
val = reshape(A, [], 1);
val = double(val);
I=hist(val,0:bin);
output = I/(numel(A));
%Calculate cumalative sum
CSum = cumsum(output);
%Perform the transformation s=T(R)
where s & R in range[0,1]
HIm=CSum(A+1);
%Convert to image into int 8
HIm = uint8(HIm*bin);
subplot(1,3,2);
imshow(HIm);
```

OUTPUT:-



C) Aim: Write a program to apply smoothing andsharpening filters on grayscale and color images .

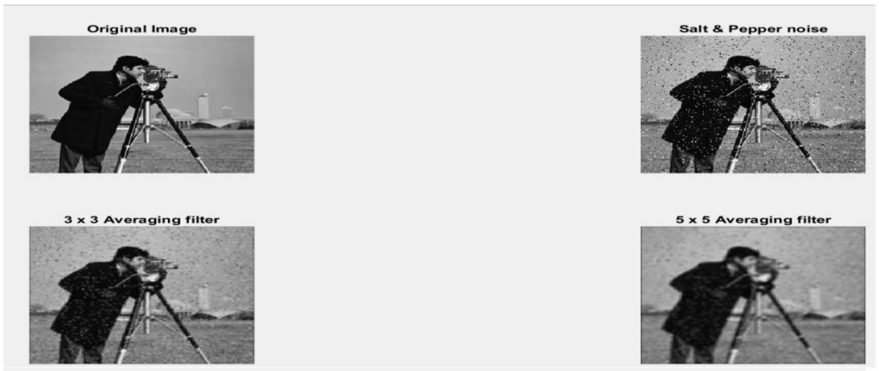
a) Low Pass

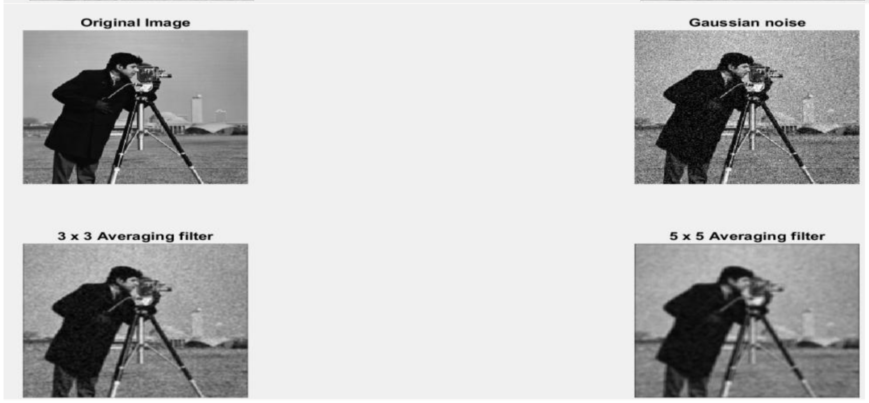
B) High Pass

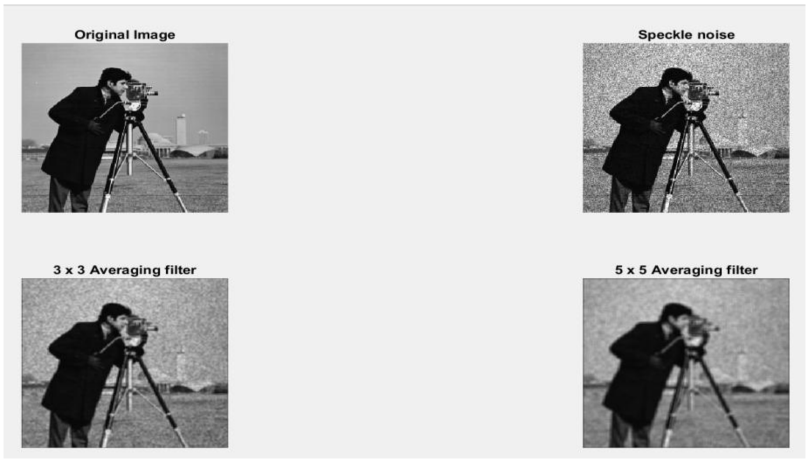
CODE:-

```
%This program is for Averaging spatial Filter
a=imread('cameraman.tif');
% Addition of noise to the input image
b=imnoise(a,'salt & pepper');
c=imnoise(a,'gaussian');
d=imnoise(a,'speckle');
% Defining 3x3 and 5x5 kernel
h1=1/9*ones(3,3);
h2=1/25*ones(5,5);
% Attempt to recover the image
b1=conv2(b,h1,'same');
b2=conv2(b,h2,'same');
c1=conv2(c,h1,'same');
c2=conv2(c,h2,'same');
d1=conv2(d,h1,'same');
d2=conv2(d,h2,'same');
% displaying the result
figure,
subplot(2,2,1),imshow(a),title('Original Image');
subplot(2,2,2),imshow(b),title('Salt & Pepper noise');
subplot(2,2,3),imshow(uint8(b1)),title('3 x 3 Averaging filter');
subplot(2,2,4),imshow(uint8(b2)),title('5 x 5 Averaging filter')
figure,subplot(2,2,1),imshow(a),title('Original Image');
subplot(2,2,2),imshow(c),title('Gaussian noise');
subplot(2,2,3),imshow(uint8(c1)),title('3 x 3 Averaging filter');
subplot(2,2,4),imshow(uint8(c2)),title('5 x 5 Averaging filter');
figure,subplot(2,2,1),imshow(a),title('Original Image');
subplot(2,2,2),imshow(d),title('Speckle noise');
subplot(2,2,3),imshow(uint8(d1)),title('3 x 3 Averaging filter');
subplot(2,2,4),imshow(uint8(d2)),title('5 x 5 Averaging filter'),
```

OUTPUT:-


Original Image
Salt & Pepper noise
3 x 3 Averaging filter
5 x 5 Averaging filter

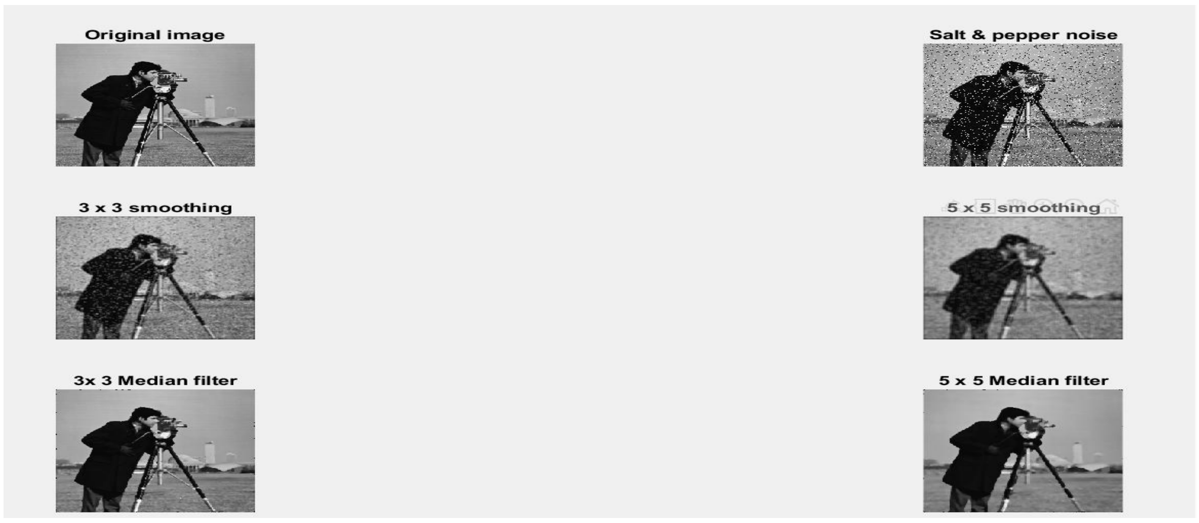

Original Image
Gaussian noise
3 x 3 Averaging filter
5 x 5 Averaging filter


Original Image
Speckle noise
3 x 3 Averaging filter
5 x 5 Averaging filter

CODE:-

```
%%% this program is for comparing
averaging & median filter%%%
clc;
clear all;
close all;
a=imread('cameraman.tif');
%Addition of salt and pepper noise
b=imnoise(a,'salt & pepper',0.1);
%Defining the box and median filters
h1=1/9*ones(3,3);
h2=1/25*ones(5,5);
c1=conv2(b,h1,'same');
c2=conv2(b,h2,'same');
c3=medfilt2(b,[3 3]);
c4=medfilt2(b,[5 5]);
subplot(3,2,1),imshow(a),title('Original
image');
subplot(3,2,2),imshow(b),title('Salt &
pepper noise');
subplot(3,2,3),imshow(uint8(c1)),title('3 x
3 smoothing');
subplot(3,2,4),imshow(uint8(c2)),title('5 x
5 smoothing');
subplot(3,2,5),imshow(uint8(c3)),title('3x
3 Median filter');
subplot(3,2,6),imshow(uint8(c4)),title('5 x
5 Median filter');
```

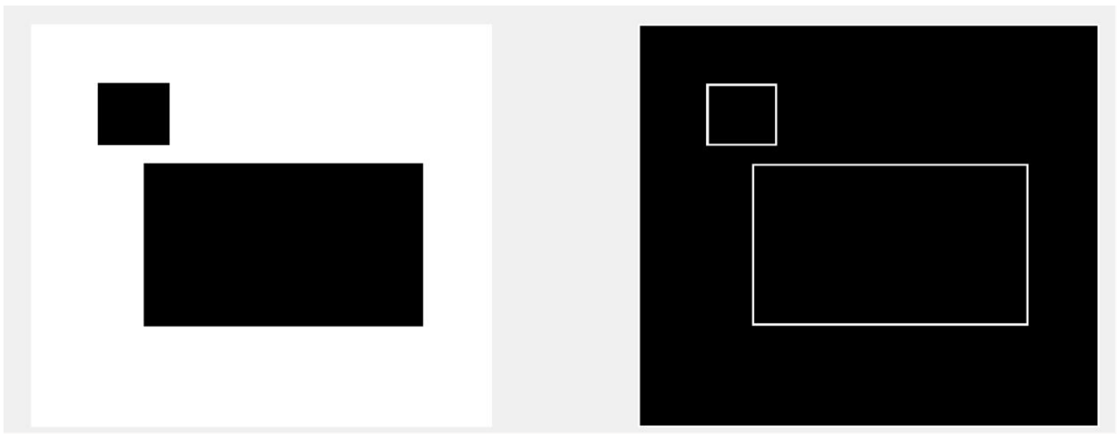
OUTPUT :-

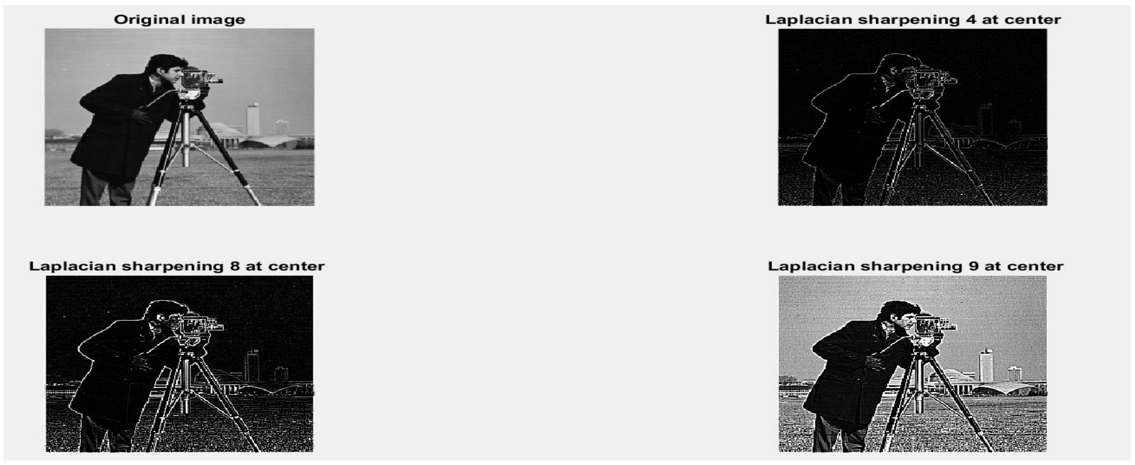


CODE:-

```
% % this program is for sharpening spatial
domain filter%%
% %Sharpening Filters%%
A=ones(200,200);
A(30:60,30:60)=0;
A(70:150,50:170)=0
figure(1) ,subplot(1,2,1),imshow(A);
AM=[1 1 1;1 -8 1;1 1 1];
B=conv2(A,AM);
subplot(1,2,2), imshow(B);

% this program is for sharpening spatial
domain filter
% a=imread('D:\horse.jpg');
%Defining the laplacian filters
h1=[0 -1 0;-1 4 -1;0 -1 0];
h2=[-1 -1 -1;
   -1 8 -1; -1 -1 -1];
h3=[-1 -1 -1;
   -1 9 -1; -1 -1 -1];
c1=conv2(a,h1,'same');
c2=conv2(a,h2,'same');
c3=conv2(a,h3,'same');
subplot(2,2,1),imshow(a),title('Original
image');
subplot(2,2,2),imshow(uint8(c1)),title('Lap
lacian sharpening 4 at center');
subplot(2,2,3),imshow(uint8(c2)),title('Lap
lacian sharpening 8 at center ');
subplot(2,2,4),imshow(uint8(c3)),title('
Laplacian sharpening 9 at center');
```

OUTPUT




Original image
Laplacian sharpening 4 at center
Laplacian sharpening 8 at center
Laplacian sharpening 9 at center

PRACTICAL NO 3 Filtering in Frequency Domain

A] Aim : Program to apply Discrete Fourier Transform on an image

Software : Matlab – Image Processing Tool Box

Theory :

The fourier transform, developed by Jean Baptiste Joesph Fourier, is widely used in the field of image processing. An image is a spatially varying function. One way to analyse spatial variations is to decompose an image into a set of orthogonal functions, one such being the fourier functions. A fourier transform is used to transform an intensity image into the domain of spatial frequency.

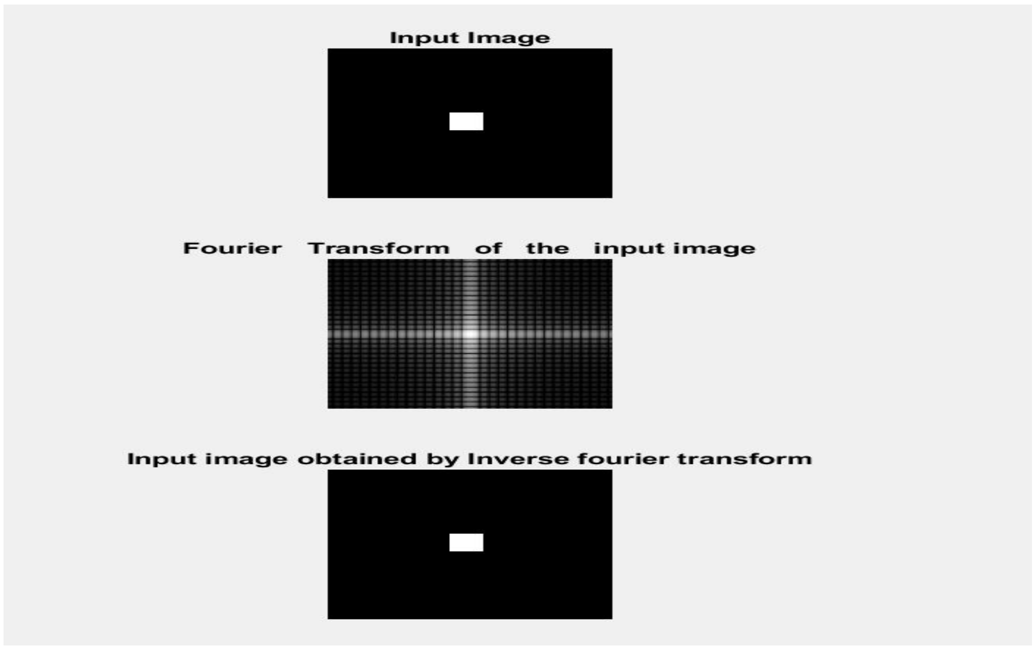
The program first creates an image, finds its fourier transform and displays it. The inverse fourier transform is found to obtain and display the input image.

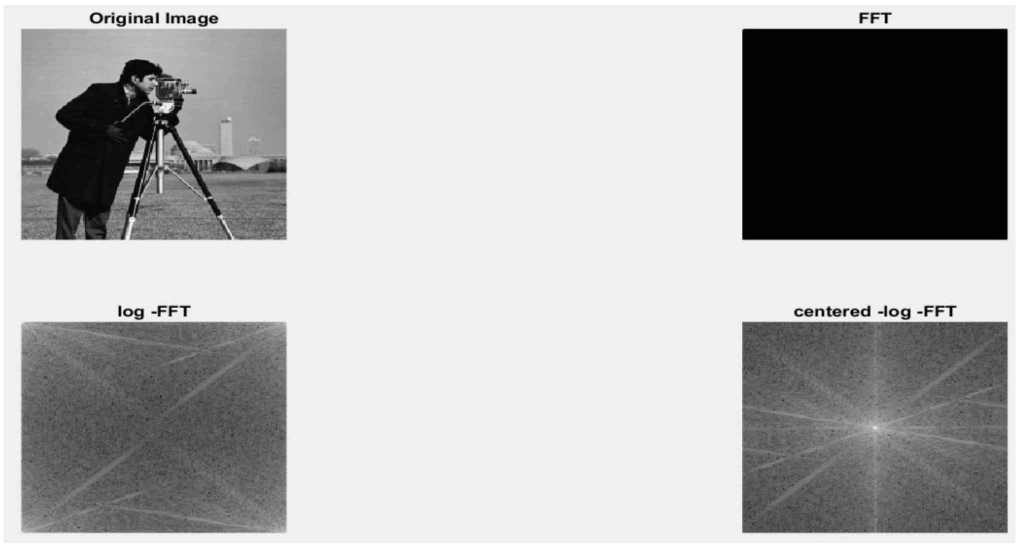
CODE:-

```
clc;
clear all;
close all;
% to create an image, find and display its
Fourier transform & also to find the
inverse fourier transform %
 a=zeros(256,256);
 a(110:140,110:140)=1;
subplot(3,1,1),
imshow(a),
title('Input Image');
a1=log(1+abs(fftshift(fft2(a))));
subplot(3,1,2),imshow(mat2gray(a1)),
title('Fourier Transform of the input
image');
a1=fft2(a);
b=ifft2(a1);
subplot(3,1,3),
imshow (b),
title('Input image obtained by Inverse
fourier transform');
% find and displayFourier transformof an
img
clc;
clear all;
close all;
I = imread('cameraman.tif');
subplot(2,2,1);imshow(I);
title('Original Image');
subplot(2,2,2);
F = fft2(I);
imshow(abs(F),[]);
title('FFT');
subplot(2,2,3);
imshow(log(abs(F)),[])
title('log -FFT');
F = fftshift(F);
subplot(2,2,4);
imshow(log(abs(F)),[])
```

```
title('centered -log -FFT')
```

OUTPUT:-

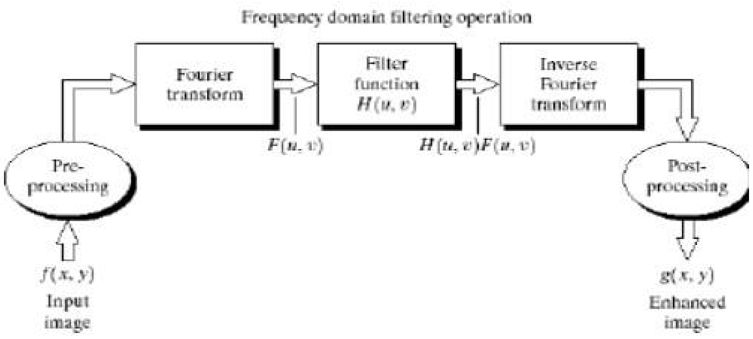

Input Image
Fourier Transform of the input image
Input image obtained by Inverse fourier transform



B] Aim: Program to apply Low pass and High pass filters in frequency domain.

Theory:-

Study of frequency domain filters .



Basic Steps for Filtering in the Frequency Domain:

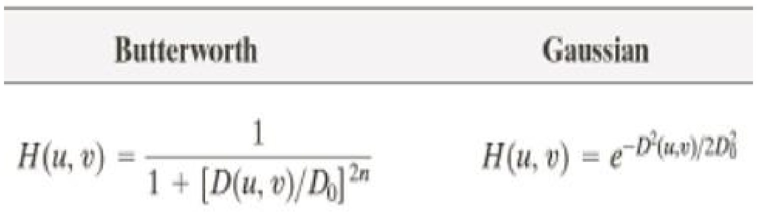
1. Multiply the input image by (-1)x+y to center the transform.

2. Compute F(u,v), the DFT of the image from (1).

3. Multiply F(u,v) by a filter function H(u,v).

4. Compute the inverse DFT of the result in (3).

5. Obtain the real part of the result in (4).
6. Multiply the result in (5) by (-1)x+y .

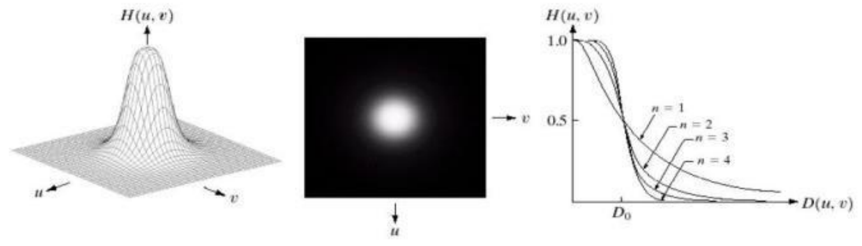


Butterworth	Gaussian
$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$	$H(u,v) = e^{-D^2(u,v)/2D_0^2}$

Low-pass filter: A filter that attenuates high frequencies while passing low frequencies. Used for blurring (smoothing).

- Lowpass filter: A filter that attenuates high frequencies while passing the low frequencies.
- Low frequencies represent the gray-level appearance of an image over smooth areas.

Frequency Domain Filtering



a b c

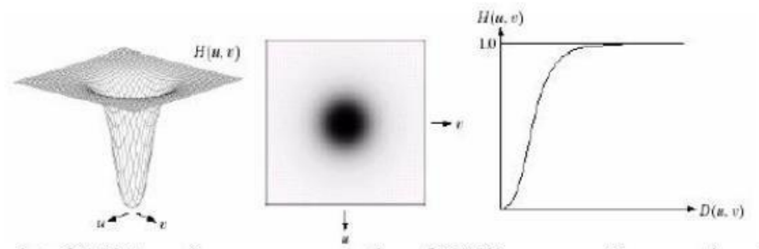
FIGURE 4.14 (a) Perspective plot of a Butterworth lowpass filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Sharpening Frequency-Domain Filters

High-pass filter: A filter that attenuates low frequencies while passing high frequencies is used for sharpening. High frequencies represent the details such as edges and noise. Sharpening is achieved by attenuating the LF (low frequency) components of DFT of an image. Where Edges and fine detail in images are associated with high frequency components (high pass filters).

- ***Butterworth High pass Filter (BHPF):*** The transfer function of BHPF of order n and with specified cutoff frequency is given by: Perspective plot of BHPF Image representation of BHPF Cross section of BHPF

- D(u,v) is the distance from the origin
- D0 is the cutoff frequency.
- n is the order of the filter



Perspective plot of BHPF   Image representation of BHPF   Cross section of BHPF

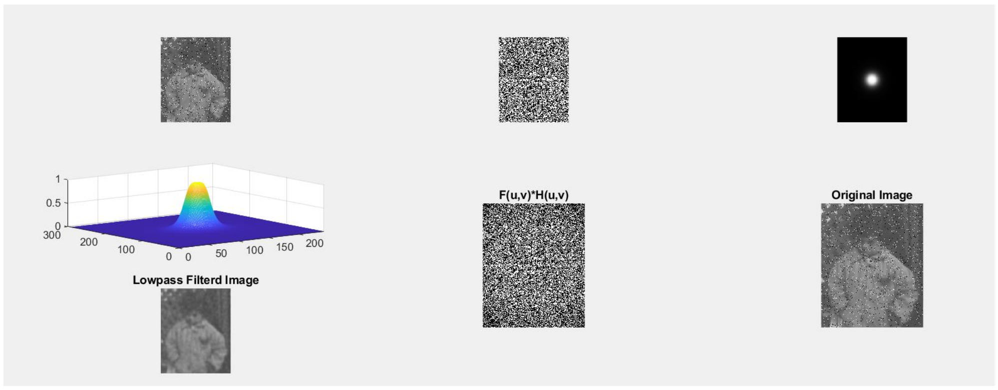
$$H(u,v) = \frac{1}{1+[D_0 / D(u,v)]^{2n}}$$

CODE:-

%Lowpass filter for Butterworth%

```
%This code is used to Butterworth
lowpass filter
close all;
clear all;
clc;
im=imread('pout.tif');
im1=imnoise(im,'salt & pepper',0.05);
subplot(4,3,1);
imshow(im1);
fc=20;
% Cutoff frequency
n=1;
[co,ro] = size(im);
cx = round(co/2);
% find the center of the image
cy = round (ro/2);
imf=fftshift(fft2(im1));

subplot(4,3,2);
imshow(imf);
H=zeros(co,ro);
for i = 1:co
    for j =1 : ro
        d = (i-cx).^2 + (j-cy).^ 2;
        H(i,j) = 1/(1+((d/fc/fc).^(2*n)));
    end
end
subplot(4,3,3);
imshow(H);
subplot(4,3,4);
mesh(H);
outf = imf .* H;
out = abs(ifft2(outf));
subplot(3,3,5);
imshow(outf);
title('F(u,v)*H(u,v)');
subplot(3,3,6);
imshow(im1);
title('Original Image');
subplot(4,3,7);
imshow(uint8(out));
title('Lowpass Filterd Image');
```

OUTPUT:-

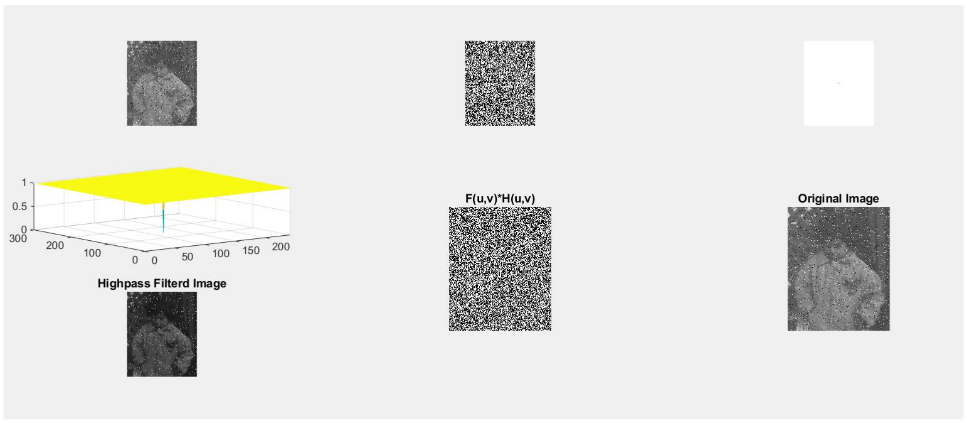


CODE:-

%Highpass Filter for Butterworth%

```
%This code is used to Butterworth
highpass filter
close all;
clear all;
clc;
im=imread('pout.tif');
im1=imnoise(im,'salt & pepper',0.05);
subplot(4,3,1);
imshow(im1);
fc=20;
% Cutoff frequency
n=1;
[co,ro] = size(im);
cx = round(co/2);
% find the center of the image
cy = round (ro/2);
imf=fftshift(fft2(im1));

subplot(4,3,2);
imshow(imf);
H=zeros(co,ro);
for i = 1:co
 for j =1 : ro
    d = (i-cx).^2 + (j-cy).^ 2;
    H(i,j) = 1/(1+((fc/fc/d).^(2*n)));
 end
end
subplot(4,3,3);
imshow(H);
subplot(4,3,4);
mesh(H);
outf = imf .* H;
out = abs(ifft2(outf));
subplot(3,3,5);
imshow(outf);
title('F(u,v)*H(u,v)');
subplot(3,3,6);
imshow(im1);
title('Original Image');
subplot(4,3,7);
imshow(uint8(out));
title('Highpass Filterd Image');
```

OUTPUT:-

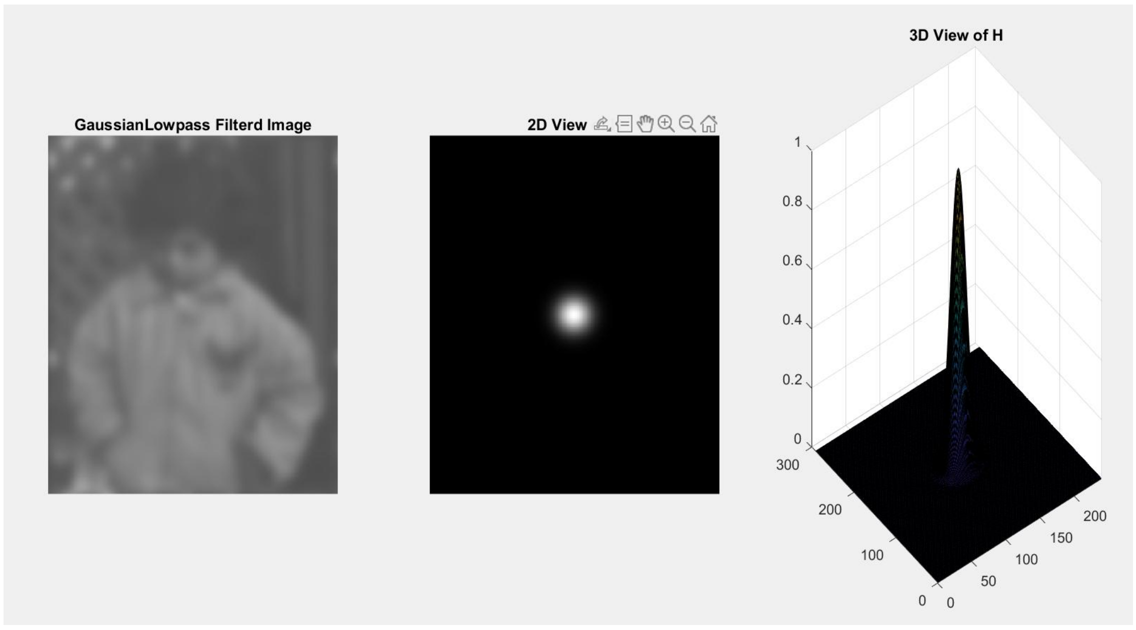


CODE:-

%Gaussian lowpass filter%

```
%This code is used to gaussian lowpass
filter
close all;
clear all;
clc;
im=imread('pout.tif');
fc=10;
imf = fftshift(fft2(im));
[co,ro]=size(im);
out = zeros(co,ro);
cx = round(co/2);
% find the center of the image
cy = round (ro/2);
H = zeros(co,ro);
for i = 1 : co
    for j = 1 : ro
        d = (i-cx).^2 + (j-cy).^2;
        H(i,j) = exp(-d/2/fc/fc);
    end
end
outf= imf.*H;
out=abs(ifft2(outf));
imshow(im);
title('Original Image');
subplot(1,3,1);
imshow(uint8(out));
title('GaussianLowpass Filterd Image');
subplot(1,3,2);
imshow(H);
title('2D View of H');
subplot(1,3,3);
surf(H);
title('3D View of H');
```

OUTPUT:-



CODE:-

%HighPass for Gaussian filter%

```
%This code is used to gaussian highpass
filter
close all;
clear all;
clc;
im=imread('pout.tif');
fc=10;
imf = fftshift(fft2(im));
[co,ro]=size(im);
out = zeros(co,ro);
cx = round(co/2);
% find the center of the image
cy = round (ro/2);
H = zeros(co,ro);
for i = 1 : co
    for j = 1 : ro
        d = (i-cx).^2 + (j-cy).^2;
        H(i,j) = 1-exp(-d/2/fc/fc);
    end
end
outf= imf.*H;
out=abs(ifft2(outf));
imshow(im);
title('Original Image');
subplot(1,3,1);
imshow(uint8(out));
title('Gaussian Highpass Filterd Image');
subplot(1,3,2);
imshow(H);
title('2D View of H');
subplot(1,3,3);
surf(H);
title('3D View of H');
```

OUTPUT:-

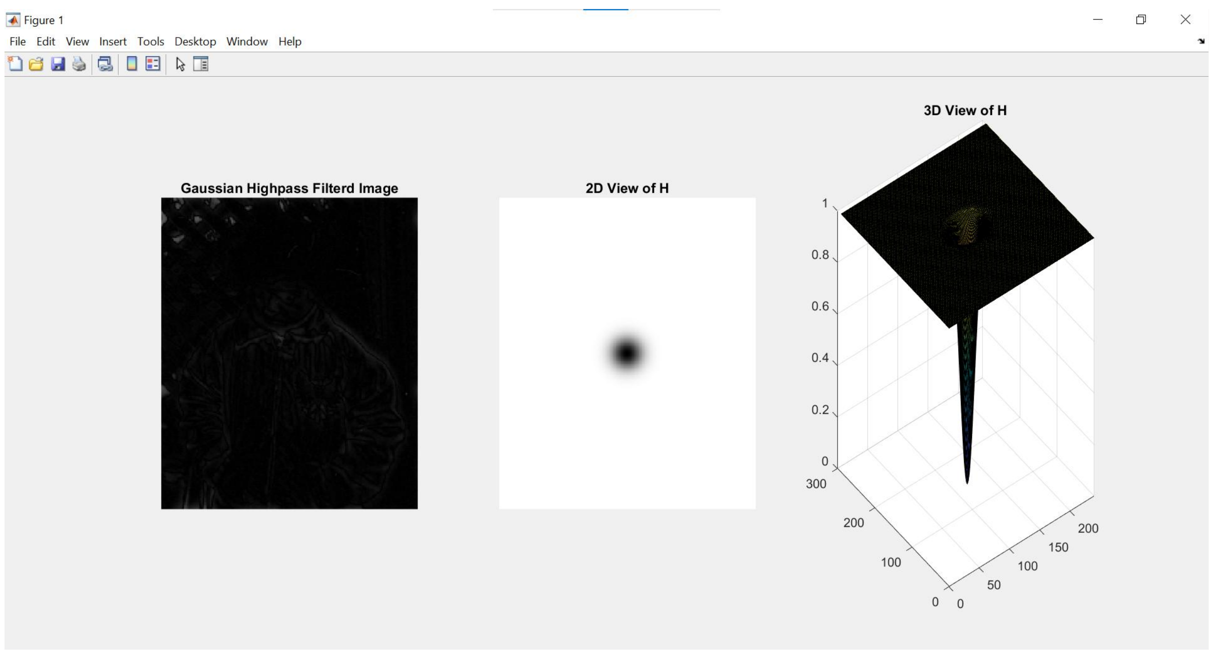


Figure 1
File Edit View Insert Tools Desktop Window Help
Gaussian Highpass Filterd Image
2D View of H
3D View of H
1
0.8
0.6
0.4
0.2
0
300
200
100
0
0
50
100
150
200

PRACTICAL NO 4 : Image Denoising

Aim: Program for image blurring using inverse, Weiner filter

Software : Matlab – Image Processing Tool Box

Theory:

INVERSE FILTERING

In the restoration of images degraded by noise and motion blur, the simplest approach is to ignore the noise term in the degradation model and form an estimate of the form F^(u,v) = G(u,v) / H(u,v). The corresponding estimate of the image is obtained by taking the inverse fourier transform of F^(u,v) . This approach is called the inverse filtering. Taking noise into account, we can express our estimate as F^(u,v) = F(u,v) + N(u,v)/H(u,v). Even if H(u,v) is known exactly, F(u,v) and thereby the input image cannot be recovered exactly because the noise component is a random function whose fourier transform N(u,v) is not known. In addition there usually is the problem of H(u,v) having numerous zeros. Even if the noise term N(u,v) were negligible, dividing it by vanishing values of H(u,v) would dominate restoration estimates. The typical approach when attempting inverse filtering is to form the ratio F^(u,v) = G(u,v) / H(u,v) and then limit the frequency range for obtaining the inverse, to frequencies near the origin. The idea is that zeros in H(u,v) are less likely to occur near the origin because the magnitude of the transform typically is at its highest values in that region. There are numerous variations of this basic theme.

In the program we first use direct inverse filtering to restore the image and observe that the process is dominated by noise. Having an estimations of the noise signal ratio improves the restoration process.

CODE :-

%Inverse Filter

```
clc;
clear all;
close all;
I = im2double(imread('cameraman.tif'));
subplot(4,1,1);
imshow(I);
title('Input Image');
LEN = 21;THETA = 11;
PSF = fspecial('motion', LEN, THETA);
blurred = imfilter(I, PSF, 'conv', 'circular');
noise_mean = 0;
noise_var = 0.0001;
blurred_noisy = imnoise(blurred, 'gaussian',
noise_mean, noise_var);
subplot(4,1,2);
imshow(blurred_noisy);
title('Input Image degraded by motion blur
and noise');
estimated_nsr = 0;
```

```
wnr2 = deconvwnr(blurred_noisy, PSF,
estimated_nsr);
subplot(4,1,3);
imshow(wnr2);
title('Restoration of Blurred, Noisy Image
Using Ideal Inverse Filter');
estimated_nsr = noise_var / var(I(:));
wnr3 = deconvwnr(blurred_noisy, PSF,
estimated_nsr);
subplot(4,1,4);
imshow(wnr3);
title('Restoration of Blurred, Noisy Image
Using Estimated NSR');
```

OUTPUT:-

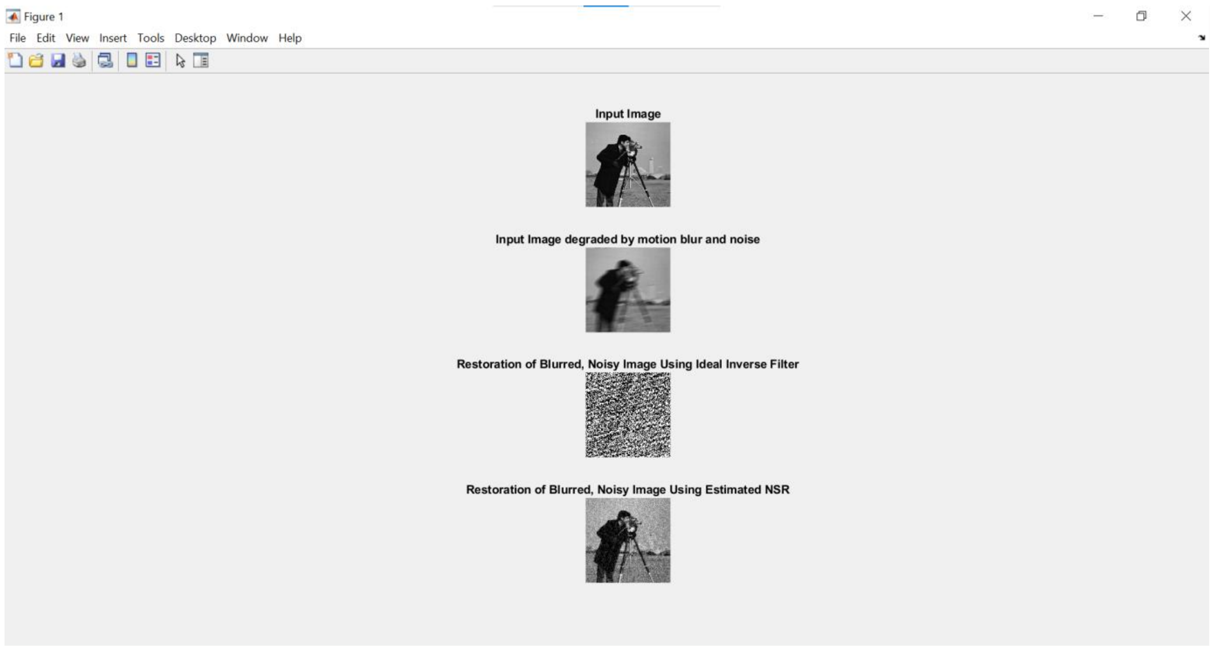


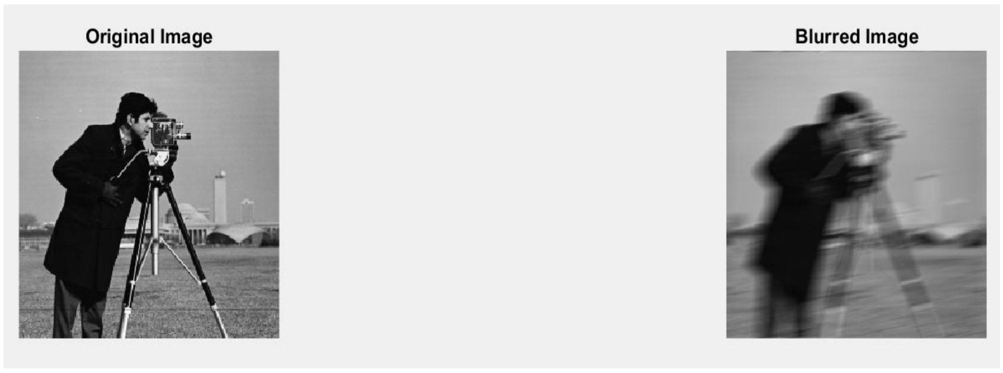
Figure 1
File Edit View Insert Tools Desktop Window Help
Input Image
Input Image degraded by motion blur and noise
Restoration of Blurred, Noisy Image Using Ideal Inverse Filter
Restoration of Blurred, Noisy Image Using Estimated NSR

CODE:-

%Weiner Filter%

```
close all;
clear all;
clc;
Ioriginal = imread('cameraman.tif');
subplot(2,2,1);
imshow(Ioriginal);
title('Original Image');
PSF = fspecial('motion',21,11);
Idouble = im2double(Ioriginal);
blurred = imfilter(Idouble,PSF,'conv','circular');
subplot(2,2,2);
imshow(blurred);
title('Blurred Image');
```

OUTPUT:-



PRACTICAL NO 5:- Color Image Processing

A] Aim:- Program to read a color image and segment into RGB planes, histogram of color image.

Theory:-

A colored image can be represented as a 3 order matrix. The first order is for the rows, the second order is for the columns and the third order is for specifying the color of the corresponding pixel. Here we use the RGB color format, so the third order will take 3 values of Red, Green ans Blue respectively. The values of the rows and columns depends on the size of the image.

Approach:
Load the image into a variable J by using imread().
Store the number of rows and columns of the image in variables, r and c.
Create 3 zero matrices R, G and B (one for each of the 3 colors) of size rXc.
Store the corresponding color plane of the image in the corresponding zero matrix.
1: Red
2: Green

3: Blue

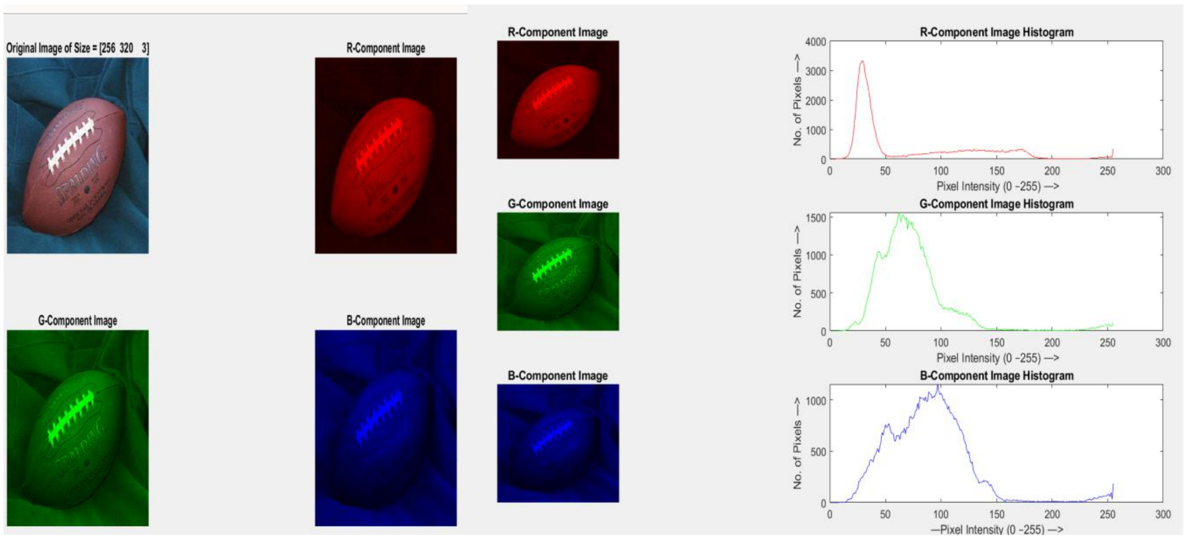
Display the images by using imshow(), but typecast them into uint8 first.

CODE:-

```
close all;
clear all;
clc;
Orig=imread('football.jpg');
PlotRow = 2;
PlotCol = 2;
subplot(PlotRow,PlotCol,1);
imshow(Orig);
[row col c] = size(Orig);
str = strcat('Original Image of Size =
[',num2str(size(Orig)),']');
title(str);
Blank = zeros(row,col);
R = Orig(:, :, 1);
R_Comp_Image = cat(3,R,Blank,Blank);
subplot(PlotRow,PlotCol,2);
imshow(R_Comp_Image);
title('R-Component Image');
G = Orig(:, :, 2);
G_Comp_Image = cat(3,Blank,G,Blank);
subplot(PlotRow,PlotCol,3);
imshow(G_Comp_Image);
title('G-Component Image');
B = Orig(:, :, 3);
B_Comp_Image = cat(3,Blank,Blank,B);
subplot(PlotRow,PlotCol,4);
imshow(B_Comp_Image);
title('B-Component Image');
PlotRow = 3;PlotCol = 2;
figure,subplot(PlotRow,PlotCol,1);
```

```
imshow(R_Comp_Image);
title('R-Component Image');
subplot(PlotRow,PlotCol,2);
[P,X] = imhist(R); plot(X,P,'r');
 title(['R-Component Image Histogram']);
   xlabel(' Pixel Intensity (0 –255) —>');
   ylabel('No. of Pixels —>');
   subplot(PlotRow,PlotCol,3);
   imshow(G_Comp_Image);
   title('G-Component Image');
   subplot(PlotRow,PlotCol,4);
   [P,X] =imhist(G);
   plot(X,P,'g');
title('G-Component Image Histogram');
xlabel(' Pixel Intensity (0 –255) —>');
ylabel('No. of Pixels —>');
subplot(PlotRow,PlotCol,5);
imshow(B_Comp_Image);
title('B-Component Image');
subplot(PlotRow,PlotCol,6);
[P,X] = imhist(B);
plot(X,P,'b');
title('B-Component Image Histogram');
xlabel('—Pixel Intensity (0 –255) —>');
ylabel('No. of Pixels —>');
```

OUTPUT:-


Original Image of Size = [256 320 3]
R-Component Image
G-Component Image
B-Component Image
R-Component Image
G-Component Image
B-Component Image
R-Component Image Histogram
G-Component Image Histogram
B-Component Image Histogram
No. of Pixels —>
Pixel Intensity (0 -255) —>
—Pixel Intensity (0 -255) —>
0
50
100
150
200
250
300
1000
2000
3000
4000
500
1500

B] Aim :- Program for converting from one color model to another model

Program to apply false colouring(pseudo) on a gray scale image.

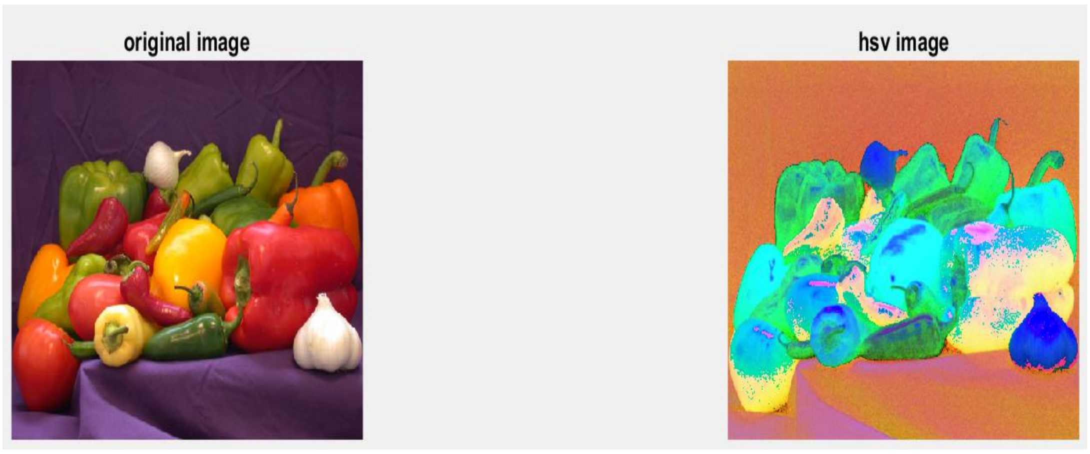
Theory:-

Grayscale image: It is a black and white image. The pixels values are shades of gray colour which is the combination of white and black shades. The image is represented in form of one 2-Dimensional matrix. Each value represents the intensity or brightness of the corresponding pixel at that coordinate in the image. Total 256 shades are possible for the grayscale images. 0 means black and 255 means white. As we increase the value from 0 to 255, the white component gets increases and brightness increases.
RGB color image: It is a colored image. It consists of three 2-Dimensional matrices, which are called channels. Red, Green and Blue channels contain the corresponding colour values for each pixel in the image. In integer format, the range of pixel intensity goes from 0 to 255. 0 means black and 255 represents the highest intensity of the primary colour. There exist 256 shades of each colour.

CODE:-

```
% Color Image segmentation
clc;
clear all;
close all;
a=imread('peppers.png');
%Conversion of RGB to HSV
b=rgb2hsv(a);
subplot(2,2,1);
imshow(a);
title('original image');
subplot(2,2,2);
imshow(b);
title('hsv image')
```

OUTPUT:-



PRACTICAL NO 6. Fourier Related transform

A] Aim: Program to compute Discrete Cosine Transforms.

Theory:-

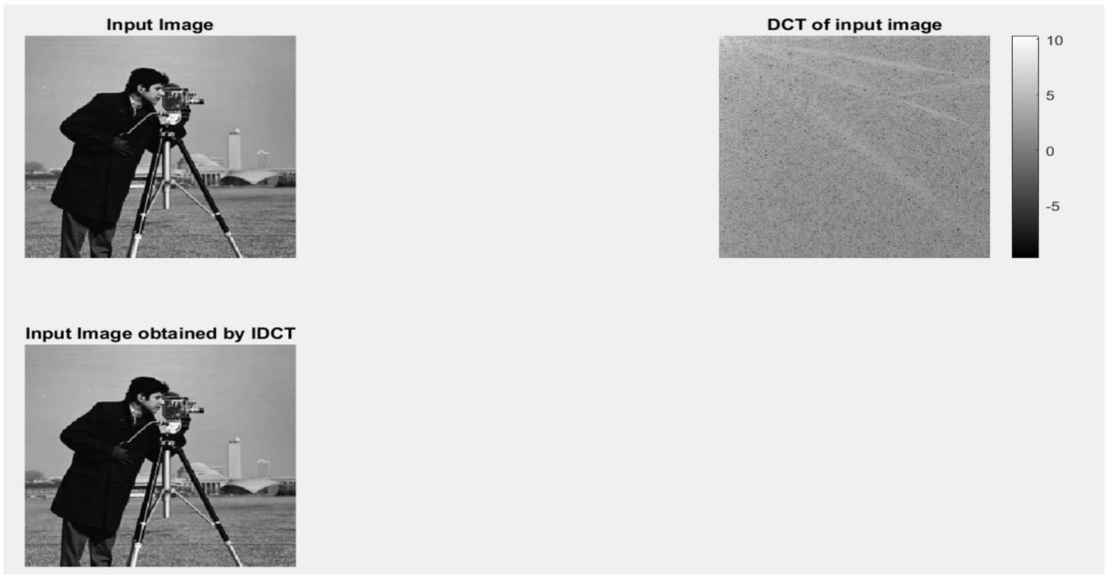
A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The DCT, first proposed by Nasir Ahmed in 1972, is a widely used transformation technique in signal processing and data compression. It is used in most digital media, including digital images (such as JPEG and HEIF, where small high-frequency components can be discarded), digital video (such as MPEG and H.26x), digital audio (such as Dolby Digital, MP3 and AAC), digital television (such as SDTV, HDTV and VOD), digital radio (such as AAC+ and DAB+), and speech coding (such as AAC-LD, Siren and Opus). DCTs are also important to numerous other applications in science and engineering, such as digital signal processing, telecommunication devices, reducing network bandwidth usage, and spectral methods for the numerical solution of partial differential equations.

CODE :-

```
clc;
clear all;
close all;
%to create an image, find and display its discrete cosine transform & also to find the inverse
fourier transform %
 I4=[1 1 1 1; 1 1 1 1; 1 1 1 1; 1 1 1 1];
 D4=dctmtx(4);
output4 = D4*I4*D4;
input4=D4'*output4*D4';
%To find the DCT of a given image and the inverse transform %
I = imread('cameraman.tif');
J = dct2(I);
K = idct2(J);
subplot(2,2,1);
imshow(I);
```

```
title('Input Image');
subplot(2,2,2),imshow(log(abs(J)),[]), colorbar, title('DCT of input image');
subplot(2,2,3),imshow(K,[0 255]), title('Input Image obtained by IDCT');
```

OUTPUT:-



B] Aim: Hadamard Transforms.

Theory:-

The Hadamard transform (also known as the Walsh–Hadamard transform, Hadamard–Rademacher–Walsh transform, Walsh transform, or Walsh–Fourier transform) is an example of a generalized class of Fourier transforms. It performs
an orthogonal, symmetric, involutive, linear operation on 2^m real numbers (or complex,
or hypercomplex numbers, although the Hadamard matrices themselves are purely real).

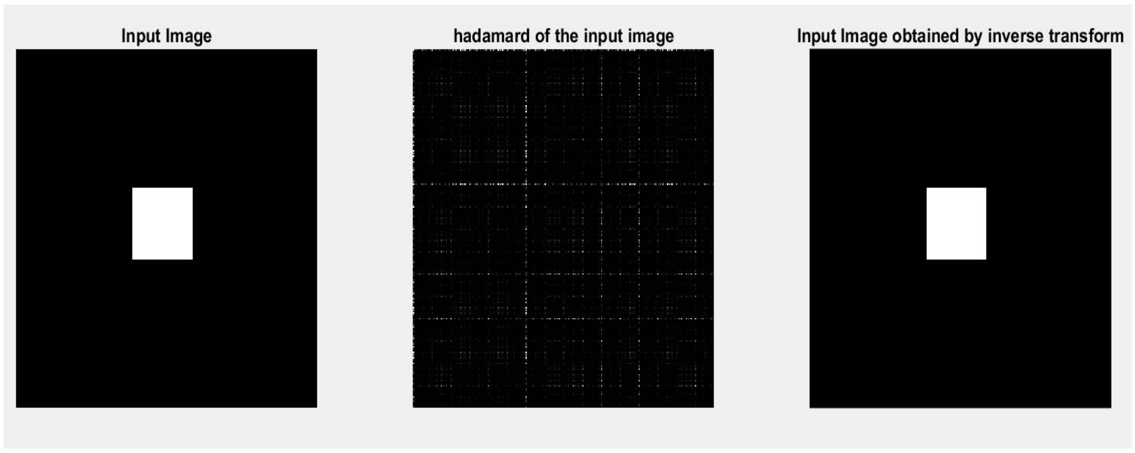
The Hadamard transform can be regarded as being built out of size-2 discrete Fourier transforms (DFTs), and is in fact equivalent to a multidimensional DFT of size $2 \times 2 \times \cdots \times 2 \times 2$. It decomposes an arbitrary input vector into a superposition of Walsh functions.

CODE :-

```
clc;

clear all;

close all;
A=[1 1 1 1; 1 1 1 1; 1 1 1 1; 1 1 1 1];
```

```
disp('The input matrix is'),A
H4=hadamard(4);
disp('The hadamard matrix for N=4 is: '), H4
B = 1/4*H4*A*H4;
disp('The hadamard transform of the input matrix is'),B
A=1/4*H4'*B*H4';
disp('The inverse hadamard is'),A
A=zeros(256,256);
A(100:150,100:150)=1;
subplot(1,3,1),imshow(A),title('Input Image');
H256=hadamard(256);
B=1/256*H256*A*H256;
subplot(1,3,2),imshow(B),title('hadamard of the input image');
A=1/256*H256'*B*H256';
subplot(1,3,3),imshow(A),title('Input Image obtained by inverse transform');
```

OUTPUT:-


Input Image
hadamard of the input image
Input Image obtained by inverse transform

PRACTICAL NO 7. Morphological Image Processing

A] Aim: Program to apply erosion, dilation, opening, closing.

Theory:-

Morphological Transformations are simple operations based on the shape of an image usually performed on a binary image. It takes our input image and a structuring element(kernel) which decides the nature of the operation.

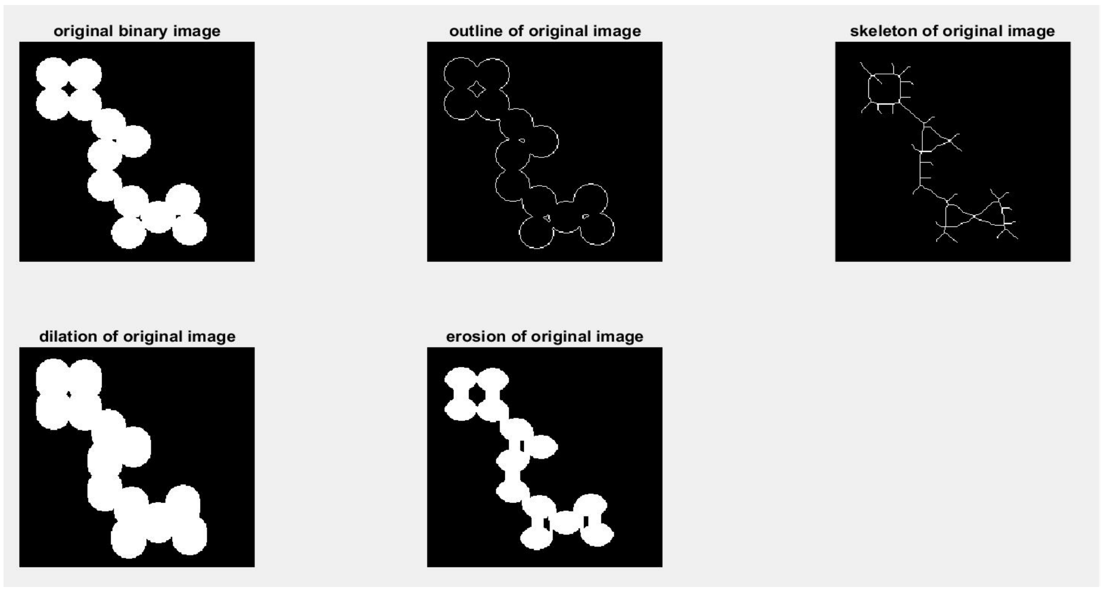
Erosion is the morphological operation used to diminish the size of the foreground object. It is just like soil erosion and erodes away the boundary of the foreground object.

Dilation is the opposite of Erosion, instead of shrinking it expands the foreground object. In this operation structuring element(kernel) is slid through the image.

CODE :-

```
clc;
close all;
a = imread ('circles.png');
b = im2bw(a,0.4);
subplot(2,3,1);
imshow(b);
title('original binary image');
c = bwmorph(b,'remove');
%removes interior pixels to obtain outline
subplot(2,3,2);
imshow(c);
title('outline of original image');
d = bwmorph(b,'skel',Inf);
%applies operation infinite times till image no longer changes
subplot(2,3,3);
imshow(d);
title('skeleton of original image');
se = strel('line',11,90);
%create a 'line' structuring element
e = imdilate(b,se);
%dilate image with a structuring element
subplot(2,3,4);
imshow(e), title('dilation of original image');
f = imerode(b,se);
%erode image with a structuring element
subplot(2,3,5);
imshow(f), title('erosion of original image');
g = bwmorph(b,'bothat');
%performs morphological " bottom hat" operation

subplot(2,3,6);

imshow(g);
title('bottom hat operation on original image');
```

OUTPUT :-


original binary image
outline of original image
skeleton of original image
dilation of original image
erosion of original image

PRACTICAL NO 9: Image Segmentation

Aim: Program for Edge detection using a. Sobel, Prewitt, Marr-Hildreth and Canny.

Theory:-

Sobel Operator: It is a discrete differentiation operator. It computes the gradient approximation of image intensity function for image edge detection. At the pixels of an image, the Sobel operator produces either the normal to a vector or the corresponding gradient vector.

Prewitt Operator: This operator is almost similar to the sobel operator. It also detects vertical and horizontal edges of an image. It is one of the best ways to detect the orientation and magnitude of an image.

Marr-Hildreth Operator or Laplacian of Gaussian (LoG): It is a gaussian-based operator which uses the Laplacian to take the second derivative of an image. This really works well when the transition of the grey level seems to be abrupt.

Canny Operator: It is a gaussian-based operator in detecting edges. This operator is not susceptible to noise. It extracts image features without affecting or altering the feature. Canny edge detector have advanced algorithm derived from the previous work of Laplacian of Gaussian operator.

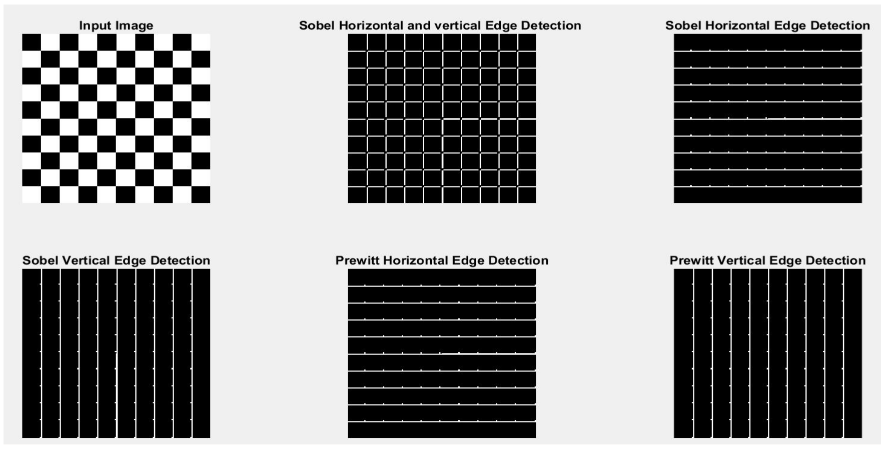
CODE:-

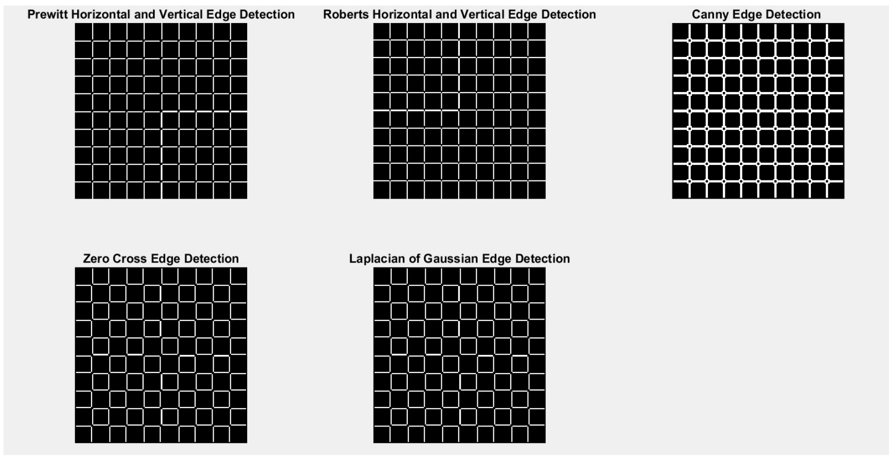
```
clc;
close all;
clear all;
a = checkerboard(15,5,5)>0.5;
subplot(2,3,1);
imshow(a);
title('Input Image');
b = edge(a,'sobel',0.1,'both');
subplot(2,3,2);
imshow(b);
title('Sobel Horizontal and vertical Edge
Detection ');
b = edge(a,'sobel',0.1,'horizontal');
subplot(2,3,3);
imshow(b);
title('Sobel Horizontal Edge Detection');
b = edge(a,'sobel',0.1,'vertical');
subplot(2,3,4);
imshow(b);
title('Sobel Vertical Edge Detection');
d = edge(a,'prewitt','horizontal',0.1);
subplot(2,3,5);
imshow(d);
title('Prewitt Horizontal Edge Detection');
```

```
d = edge(a,'prewitt','vertical',0.1);
subplot(2,3,6);
imshow(d);
title('Prewitt Vertical Edge Detection');
d = edge(a,'prewitt','both',0.1);
figure,
subplot(2,3,1);
imshow(d);
title('Prewitt Horizontal and Vertical Edge
Detection');
c = edge(a,'roberts','both');
subplot(2,3,2);
imshow(c);
title('Roberts Horizontal and Vertical Edge
Detection');
```

```
e = edge(a,'canny',.01);
subplot(2,3,3);
imshow(e);
title('Canny Edge Detection ');
f = edge(a,'zerocross');
subplot(2,3,4);
imshow(f);
title('Zero Cross Edge Detection');
c = edge(a,'log');
subplot(2,3,5);
imshow(c);
title('Laplacian of Gaussian Edge
Detection');
```

OUTPUT


Input Image
Sobel Horizontal and vertical Edge Detection
Sobel Horizontal Edge Detection
Sobel Vertical Edge Detection
Prewitt Horizontal Edge Detection
Prewitt Vertical Edge Detection


Prewitt Horizontal and Vertical Edge Detection
Roberts Horizontal and Vertical Edge Detection
Canny Edge Detection
Zero Cross Edge Detection
Laplacian of Gaussian Edge Detection