EX.NO:1 **IMAGE SAMPLING AND QUANTIZATION**

DATE:

AIM:

To display grayscale and color images and perform sampling and quantizationusing MATLAB.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

Step 1: Read the image using imread().

Step 2: Display the color image using imshow()

Step 3: Convert the color image into RGB image using rgb2gray().

Step 4: Display the grayscale image using imshow().

Step 5: To display all the images in a single window use subplot().

Step 6: Vary the sampling rate and view the image.

THEORY:

A grayscale image is a data matrix whose values represent shades of gray. When the elements of a grayscale image are of class uint8 or uint16, they have integer values in the range [0,255] or [0, 65535] respectively.

The RGB image is stored as m by n by 3 data that define the red, green, and blue color components for each pixel. Graphics file formats store RGB images as 24-bit images where the Red, Green and Blue are 8 bits each. RGB MATLAB array can be class double, uint8 or uint16. Pixel whose colour components are {0,0,0} are represented as black and pixel with{1,1,1} are represented as white.

PROGRAM:

Sampling (spatial)

```
clc;
clear all;
close all;
a = imread('peppers.png');
b = rgb2gray(a);
figure();
imshow(b);
title('Normal gray');
c= imresize(b,2);
```

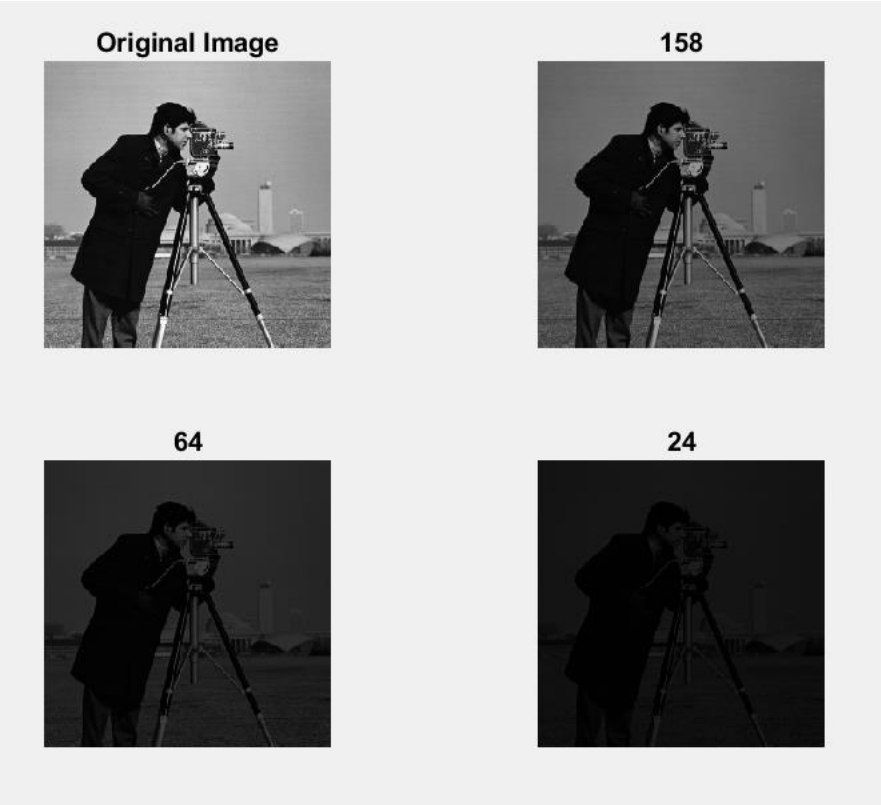
```
figure();
imshow(c);
title('Double');
d= imresize(b,0.5);
figure();
imshow(d);
title('Half');
f= imresize(b,[100,100]);
figure();
imshow(f);
title('100*100');
```

Quantisation

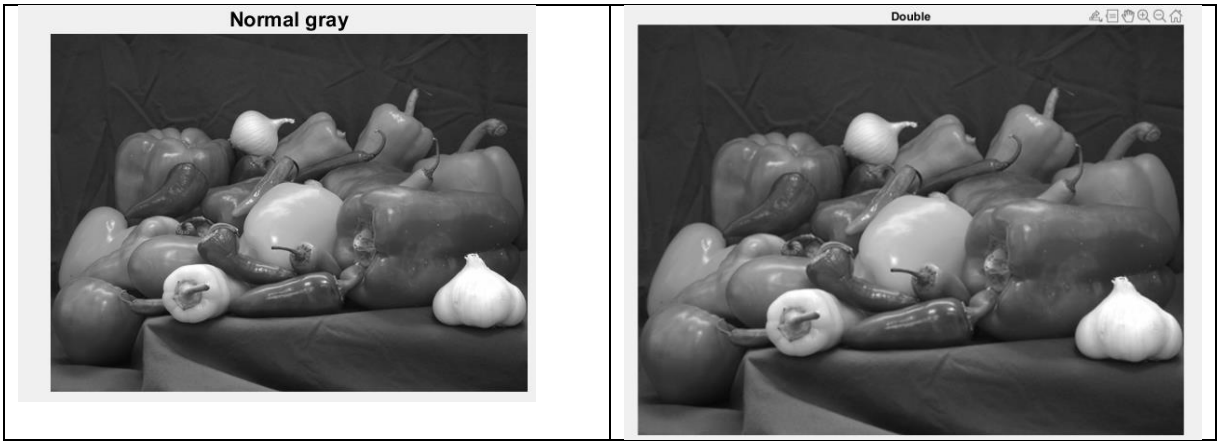
```
clc;
clear all;
close all;
a = imread('peppers.png');
b = rgb2gray(a);
subplot(2,3,1);
imshow(a);
title('Original');
c = grayslice(a,128);
subplot(2,3,2);
imshow(c);
title('128');
d = grayslice(a,64);
subplot(2,3,3);
imshow(d);
title('64');
e = grayslice(a,32);
subplot(2,3,4);
imshow(e);
title('32');
f = grayslice(a,16);
subplot(2,3,5);
imshow(f);
title('16');
g = grayslice(a,8);
subplot(2,3,6);
imshow(g);
title('8');
```

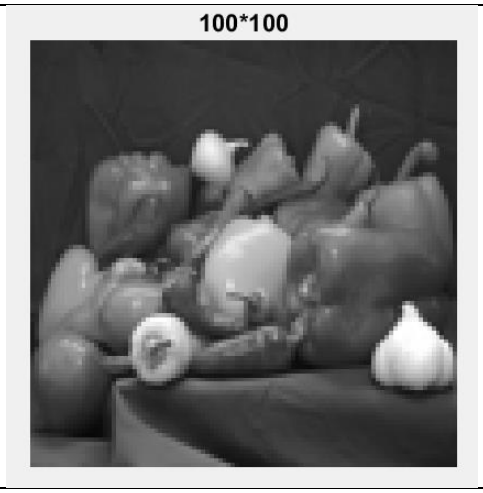
OUTPUT:

Quantization



Sampling





RESULT:

Thus, the program for displaying gray scale images and colour images by varying the sampling rate and quantization has been executed using MATLAB and output verified.

EX.NO: 2 **INTENSITY TRANSFORMATION OF IMAGES**

DATE:

AIM:

To write a program to visualize the various intensity transformation of an image.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

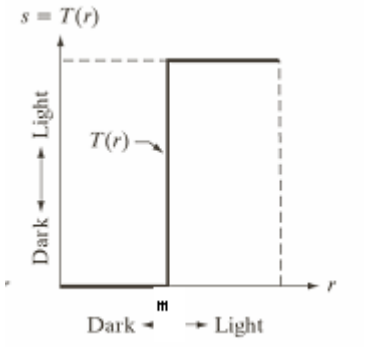
ALGORITHM:

1. Read the input image and convert the input image into grayscale image.
2. Display the gray scale image.
3. Perform basic mathematical operations.
4. Perform thresholding operation to convert gray scale to binary image.
5. Perform negative, power law and log transformation.
6. View the image.

THEORY:

Thresholding:

This method produces a two-level (binary image) by making all the pixelswith gray levels below and above m as black and white respectively.



In addition to the above two transformations there are three basic types of functions used frequently for image enhancement.

1. Linear (negative and identity transformations)
2. Logarithmic (log and inverse log transformations)
3. Power law (n^{th} power and n^{th} root transformations

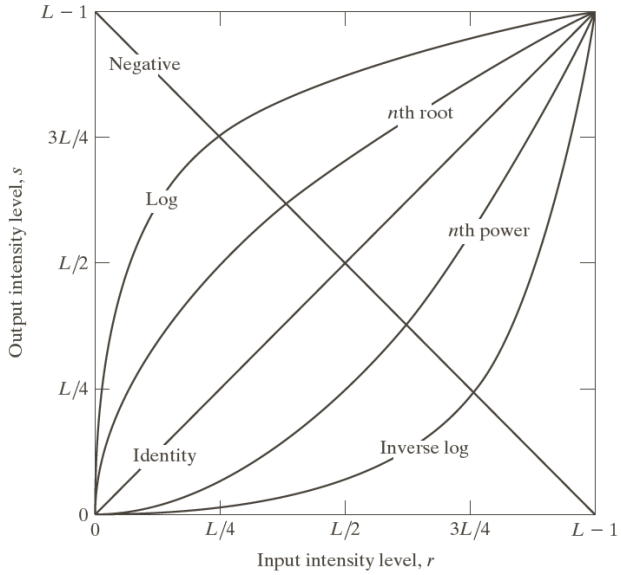


Image negative:

The negative of an image with gray levels in the range [0, L-1] is obtained by using negative transformation given by the expression s = L – 1 – r.

This processing is suited for enhancing white or gray details embedded in dark regions of image especially when black areas are dominant in size.

Log transformations:

The log transformation is given by

$$s = c \log (1+r)$$

where c is a constant and $r \geq 0$.

This transformation maps a narrow range of low gray levels in the input image into a wider range of out put levels and vice versa for high gray levels i.e. this transformation expands the values of dark pixels in an image while compressing the higher level values.

It is just the opposite form inverse log transformations.

The log functions are used to compress the dynamic range of images with large variations in pixel values, such as Fourier spectrum.

Power law transformations:

The process is given by the expression $s = cr^{\gamma}$ or $c(r+\varepsilon)^{\gamma}$ where c and γ are positive constants and ε is constants to account for an offset (i.e. a measurable output when input is zero). Power law curves with $\gamma < 1$ map a narrow range of dark (low) input values into wider range of output values and vice versa for high input levels. It is just opposite for curves with $\gamma > 1$.

The power law transformations can be used in gamma correction. Most of the devices used for image capture, printing and display respond according to the power law. The process used to correct this power law response phenomenon is called gamma correction.

Gamma correction is important when

1. The image is to be displayed accurately on a compute screen. Images that are not corrected properly will look bleached out or too dark.
2. For accurate reproduction of colors.

In addition to gamma correction, power law transformations are useful for contrast manipulation.

PROGRAM:

ARITHMETIC TRANSFORMATION:

```
clc;
close all;
a = imread('C:\Users\pavithra\Desktop\College Stuffs\MATlab\dandel.jpeg')
a = rgb2gray(a);
subplot(2,3,1);
imshow(a);
title('Original Image');
b = 150 + a;
subplot(2,3,2);
imshow(b);
title('150 + a')
c = 2*a;
subplot(2,3,3);
imshow(c);
title('2*a');
e = a-34;
subplot(2,3,4);
imshow(e);
title('a-34');
f = a/4;
subplot(2,3,5);
imshow(f);
title('a/4');
```

NEGATIVE TRANSFORMATION :

```
clc;
close all;
a = imread('peppers.png')
a = rgb2gray(a);
subplot(1,2,1);
imshow(a);
title('Original Image');

b = 255 - a;
subplot(1,2,2);
imshow(b);
title('Negative Image');
```

GAMMA TRANSFORMATION:

```
clc;
clear all;
close all;
a=imread('cameraman.tif');
subplot(2,4,1);
imshow(a);
title('Original');
c=1;
g1=2;
d=im2double(a);
s1=c*(d.^g1);
subplot(2,4,2);
imshow(s1);
title('g1=2');
g2=5;
s2=c*(d.^g2);
subplot(2,4,3);
imshow(s2);
title('g2=5');
g3=10;
s3=c*(d.^g3);
subplot(2,4,4);
imshow(s3);
title('g3=10');
g4=0.1;
s4=c*(d.^g4);
subplot(2,4,5);
imshow(s4);
```

```
title('g4=0.1');
g5=0.2;
s5=c*(d.^g5);
subplot(2,4,6);
imshow(s5);
title('g5=0.2');
g6=0.3;
s6=c*(d.^g6);
subplot(2,4,7);
imshow(s6);
title('g6=0.3');
g7=0.5;
s7=c*(d.^g7);
subplot(2,4,8);
imshow(s7);
title('g7=0.5');
```

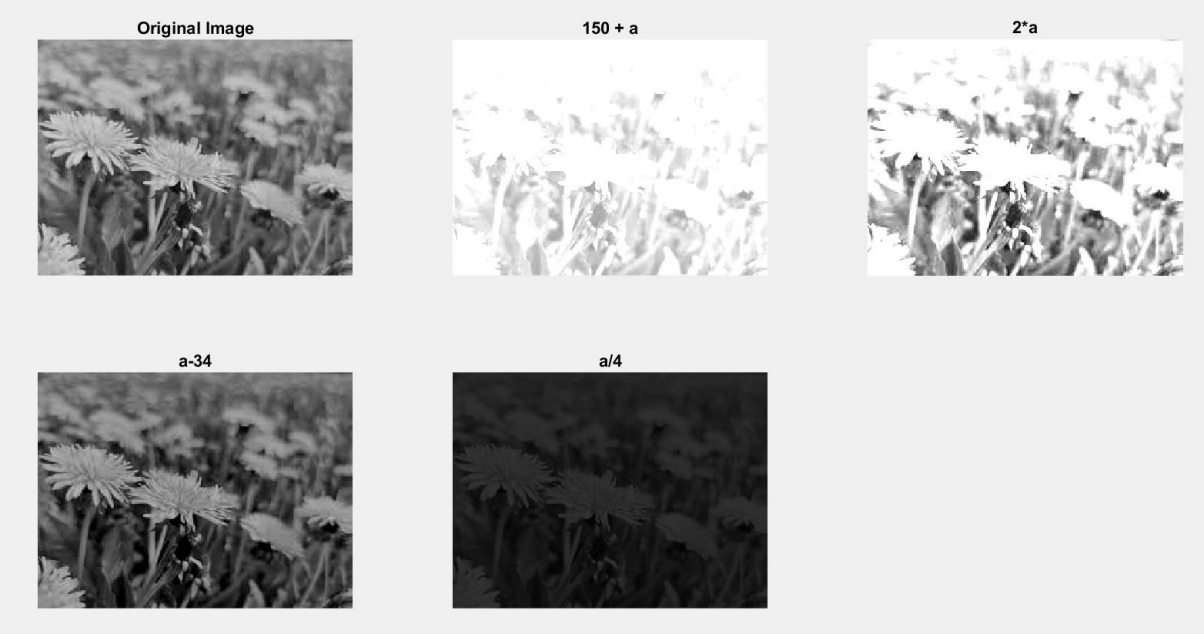
LOG TRANSFORMATION:

```
clc;
clear all;
close all;
a=imread('cameraman.tif');
subplot(2,4,1);
imshow(a);
title('Original');
c=1;
g1=2;
d=im2double(a);
s1=c*(d.^g1);
subplot(2,4,2);
imshow(s1);
title('g1=2');
g2=5;
s2=c*(d.^g2);
subplot(2,4,3);
imshow(s2);
title('g2=5');
g3=10;
s3=c*(d.^g3);
subplot(2,4,4);
imshow(s3);
title('g3=10');
g4=0.1;
s4=c*(d.^g4);
subplot(2,4,5);
imshow(s4);
title('g4=0.1');
g5=0.2;
s5=c*(d.^g5);
subplot(2,4,6);
```

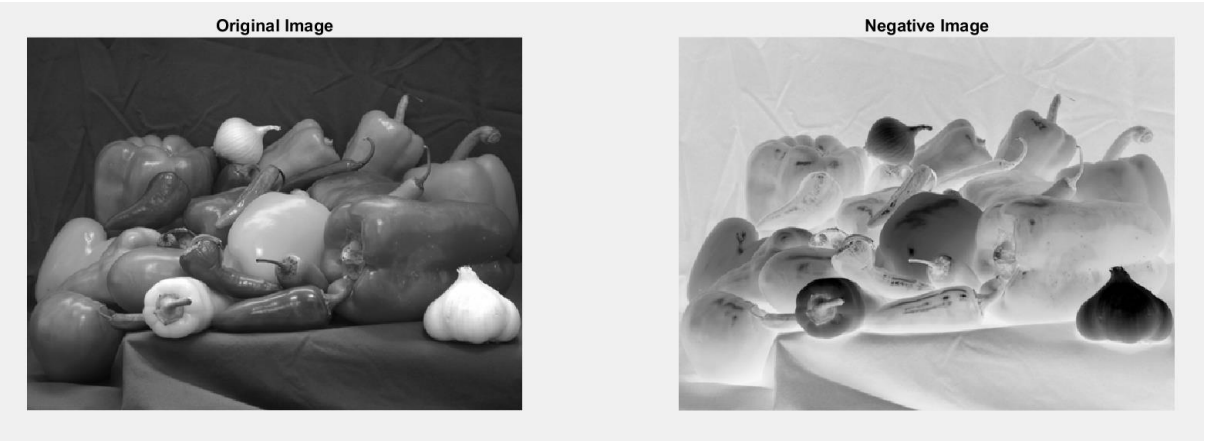
```
imshow(s5);
title('g5=0.2');
g6=0.3;
s6=c*(d.^g6);
subplot(2,4,7);
imshow(s6);
title('g6=0.3');
g7=0.5;
s7=c*(d.^g7);
subplot(2,4,8);
imshow(s7);
title('g7=0.5');
```

OUTPUT:

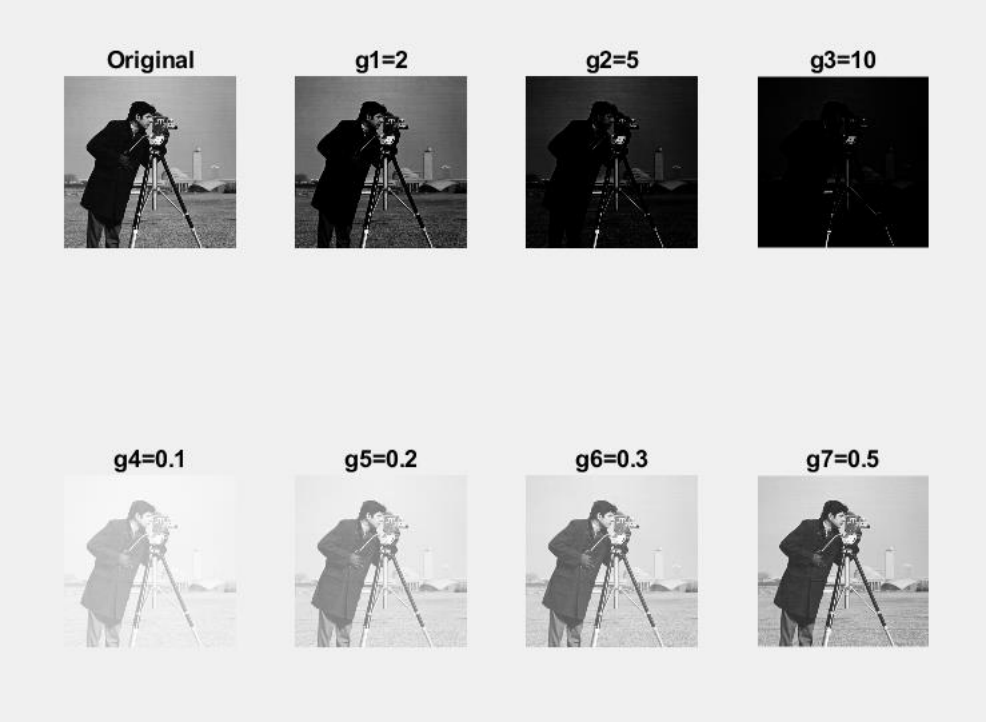
Arithmetic Transformation



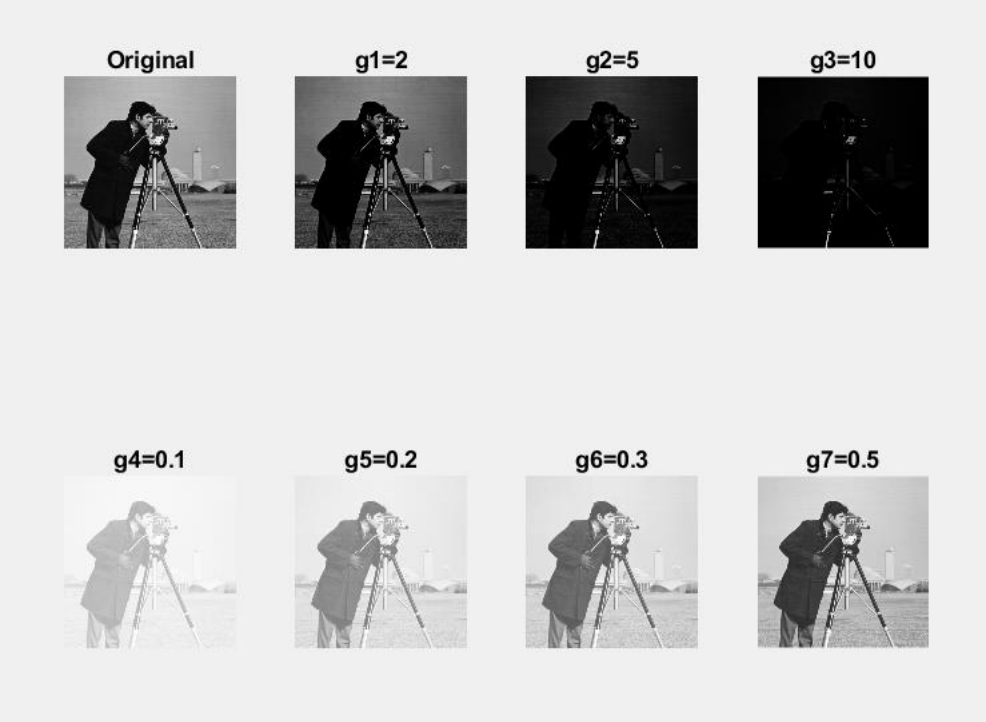
Negative Transformation



Gamma Transformation



Log Transformation



RESULT:

Thus the various methods of intensity transformations has been verified using MATLAB programs.

EX NO: 3 **HISTOGRAM PROCESSING**

DATE:

AIM:

To improve the contrast of an image using histogram equalization technique.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

With built-in function:

1. Read the input image and convert the input image into grayscale image.
2. Display the gray scale image.
3. Obtain histogram of input image using imhist() and display it.
4. Apply histogram equalization to the input image using histeq().
5. Display the equalized histogram and the histogram equalized image.
6. Apply adaptive histogram equalization to the input image using adapthisteq().
7. Display the adaptive equalized histogram and the corresponding equalized image.

Without built in function:

1. Read the input image and convert the input image into grayscale image.
2. Display the gray scale image and its histogram.
3. Define the number of gray levels L and find the number of pixels N in the image.
4. For each gray level r_k, find the corresponding number of pixels n_k (histogram).
5. Find the probability of each gray level $p_r(r_j)$ (normalized histogram) and the cumulative histogram.
6. Determine the output gray levels s_k by applying transformation function on the cumulative histogram.
7. Obtain the output image by replacing each gray level r_k in the input image with the corresponding output gray level s_k.
8. Display the output image and its histogram.

THEORY:

Histogram of an image is a plot of the number (r_k) of occurrences of gray level in the image against the gray level values. Histogram provides a summary of the intensity of the images and provides more insight about the image contrast and brightness.

For dark image, histogram is concentrated in the lower (dark) side of gray scale. For a bright image, the histogram is concentrated on higher side of the gray scale. For a low contrast image, has a narrow histogram centered towards middle of gray scale. For a high contrast image, the histogram covers a broad range of gray scale and the pixels are uniformly distributed over the entire range. Thus an image with pixels distributed uniformly over an entire range of possible gray levels

Equalization is a process that attempts to spread out the gray levels in an image so that they are evenly distributed across the range. The result of this intensity level equalization process is an image with increased dynamic range and higher contrast.

The histogram equalization automatically determines a transformation function $T(r_k)$ based on the normalized histogram $p_r(r_k)$ of input image and applies this transformation on the input image to produce an output image that has uniform histogram.

The transformation function used often is

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = (L-1)\sum_{j=0}^{k} n_j/n \text{ where } k = 0 \text{ to } L\text{-}1$$

This transformation or mapping is called histogram equalization or linearization.

The output image is obtained by mapping each gray level r_k in the input image, into corresponding gray level s_k in the output image.

PROGRAM

Histogram with built-in function

```
clc;
close all;
a = imread('peppers.png');
a = rgb2gray(a);
subplot(2,2,1);
imshow(a);
title('Original Image');
subplot(2,2,2);
imhist(a);
title('Histogram of the Image');
subplot(2,2,3);
histeq(a);
title('Equalized Histogram');
```

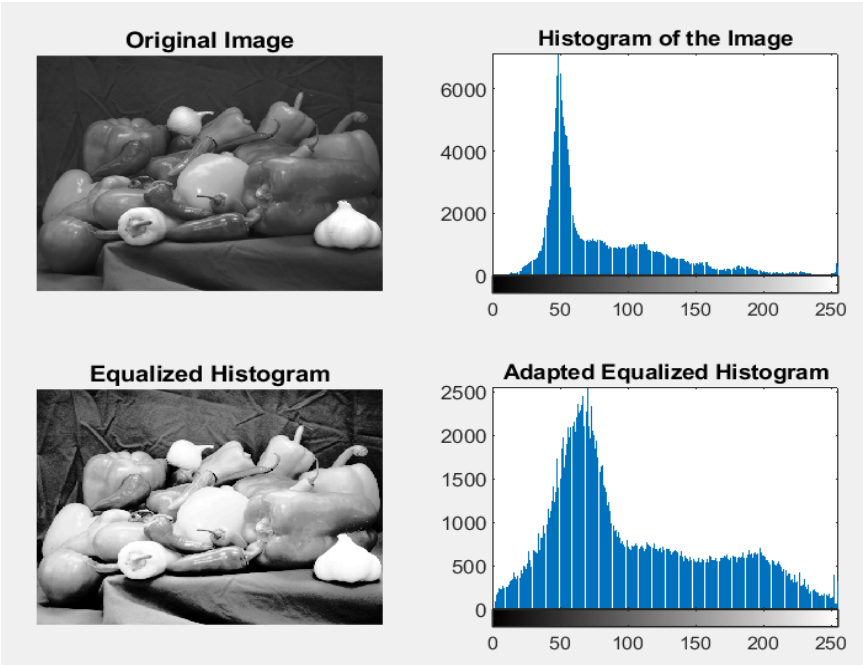
```
subplot(2,2,4);
b = adapthisteq(a);
imhist(b);
title('Adapted Equalized Histogram');
```

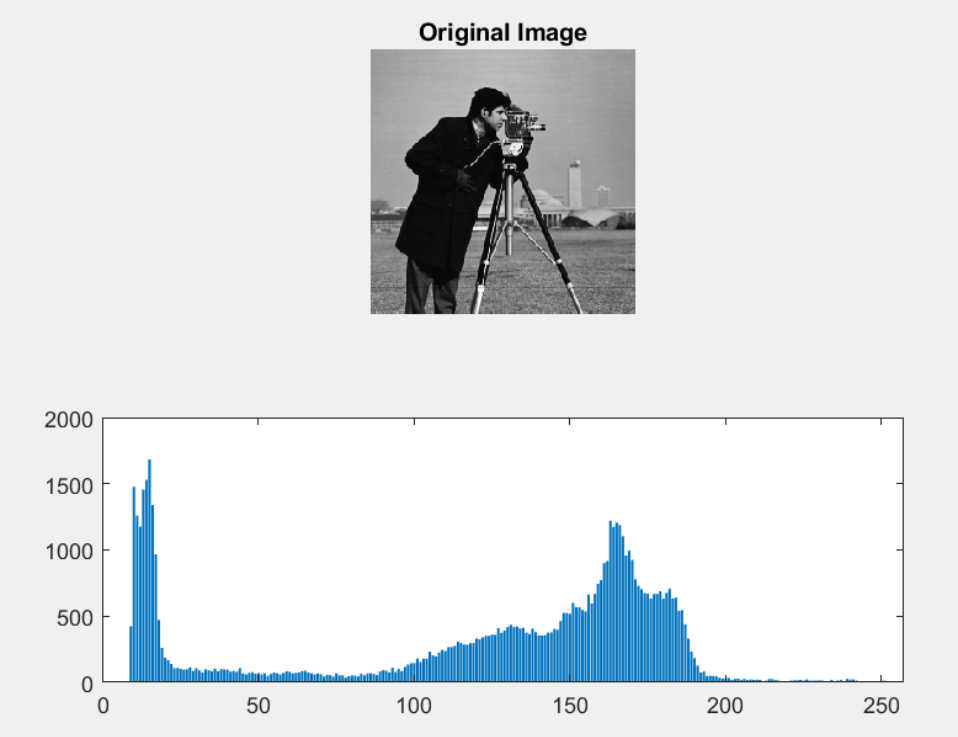
Histogram without built-in function

```
clc;
close all;
a = imread('cameraman.tif');
subplot(2,1,1);
imshow(a);
title('Original Image');
subplot(2,1,2);
z = im2double(a);
k = zeros(256,1);
for i=1:size(z,1)
        for j=1:size(z,1)
        for k1=0:255
        if a(i,j)==k1
                k(k1+1) = k(k1+1)+1;
        end
        end
        end
end
bar(k)
```

OUTPUT:

Histogram with built-in function





RESULT:

Thus, a program to enhance a poor contrast image using histogram equalization technique was written and output verified.

EX.NO: 4 IMAGE TRANSFORMS

DATE:

AIM:

To write a program to find Discrete Fourier Transform, Discrete Cosine Transform and Discrete Wavelet Transform of input image and reconstruct the image from the transform.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

DISCRETE FOURIER TRANSFORM

Step 1: Read and display the input image.

Step 2: Obtain the kernel for Discrete Fourier Transform using function fft();

Step 3: Obtain the DFT of the input image using the kernel and display it.

Step 4: Reconstruct the image from the DFT of input image using inverse kernel.

DISCRETE COSINE TRANSFORM:

Step 1. Read the input image and convert the input image into grayscale image.

Step 2. Display the gray scale image.

Step 3. Obtain the DCT of the input image and display it.

Step 4. Reconstruct the input image by applying inverse DCT to DCT of input image.

Step 5. Find the error between input image and the reconstructed image and display it

Step 6. Obtain the DCT for each 8x 8 area of the image and display it.

Step 7. Reconstruct the input image by applying inverse DCT to the result of step 6

DISCRETE WAVELET TRANSFORM:

Step 1. Read the input image and convert the input image into a grayscale image.

Step 2. Display the grayscale image.

Step 3. Obtain the DWT of the input image and display the result.

Step 4: Perform second level decomposition and display the result.

THEORY:

DISCRETE FOURIER TRANSFORM:

It is used to decompose an image into its sine and cosine components. The 2D-DFT converts an input image u(m,n) in spatial domain into image v(k,*l*) in the Fourier or frequency domain. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The image in the spatial and Fourier domains are of the same size.

The two dimensional DFT of an N x N image {u(m,n)} is a separable transform defined as

$$v(k,\ell) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n) W_N^{km} W_N^{\ell n} \qquad 0 \le k, l \le N-1$$

Inverse transform is

$$u(m,n) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} v(k,\ell) W_N^{-km} W_N^{-\ell n} \qquad 0 \le m,n \le N-1$$

Two dimensional unitary DFT pair is defined as

$$v(k,\ell) = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} u(m,n) W_N^{km} W_N^{\ell n} \qquad 0 \le k, l \le N-1$$

$$u(m,n) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} v(k,\ell) W_N^{-km} W_N^{-\ell n} \qquad 0 \le m,n \le N-1$$

This becomes $V = FUF^T$ 5.65

Where U in the input image and V is the output image.

The Fourier Transform produces a complex number valued output image which can be displayed with two images, either with the *real*and*imaginary*part or with*magnitude* and *phase.*

Inimage processing, often only the magnitude of the Fourier Transform is displayed as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image.

The Fourier Transform is used for accessing the geometric characteristics of a spatial domain image. Because the image in the Fourier domain is decomposed into its sinusoidal components, it is easy to examine or process certain frequencies of the image, thus influencing the geometric structure in the spatial domain.

In most implementations the Fourier image is shifted in such a way that the DC-value (*i.e.* the image mean) v(0.0) is displayed in the center of the image. The image points move further away from the center, as their frequency increases.

DISCRETE COSINE TRANSFORM:

A DCT expresses a sequence of finite data points in terms of sum of cosine functions oscillating at different frequencies unlike Fourier transform which uses sine and cosine waves to represent a signal.

2D-DCT:

Let u(m,n) be the input image of size N x N and v(k,*l*) be the transformed image also of size N x N.

Transformed image $$v(k,\ell) = \alpha(k)\alpha(\ell)\sum_{m,n=0}^{N-1} u(m,n)\cos[\frac{\pi(2m+1)k}{2N}]\cos[\frac{\pi(2n+1)\ell}{2N}]$$

Where $\alpha(k) \triangleq \sqrt{\frac{1}{N}}$ $k = 0$

$\alpha(k) \triangleq \sqrt{\frac{2}{N}}$ $1 \le k \le (N-1)$

$\alpha(\ell) \triangleq \sqrt{\frac{1}{N}}$ $\ell = 0$

$\alpha(\ell) \triangleq \sqrt{\frac{2}{N}}$ $1 \le \ell \le (N-1)$

The inverse DCT is given by

$$u(m,n) = \sum_{k,\ell=0}^{N-1} \alpha(k)\alpha(\ell)v(k,\ell)\cos[\frac{\pi(2m+1)k}{2N}]\cos[\frac{\pi(2n+1)\ell}{2N}]$$

The Discrete Cosine Transform has become the method of choice for image data compression. Most practical transform coding systems are based on the Discrete Cosine transform which provides a good compromise between information packing ability and computational complexity.

Compared to other input independent transforms, it has advantages of having been implemented in a single integrated circuit, packing the most information into the fewest coefficients and minimizing the block like appearance, called blocking artifact that results when boundaries between subimages become visible.

DISCRETE WAVELET TRANSFORM:

Wavelets are small waves of varying frequency and limited duration. they are used to analyze the signals or images at both time and frequency scaling.

The transformed signal is a function of two variable τ and s where τ and s are translational scaling parameter. Ψ (t) is a transforming function and is called mother wavelet. When digital images are to be viewed or processed at multiple resolutions, the DWT is the mathematical tool. DWT provides powerful insight into an image's spatial and frequency characteristics, whereas Fourier transform reveals an image's frequency attributes. The term DWT refers to class of transformations that differ not only in transformation kernels employed but also in the fundamental nature of those functions and in the way in which they are applied.

PROGRAM:

DISCRETE FOURIER TRANSFORM:

```
clc;
clear all;
close all;
b=imread('cameraman.tif');
subplot(2,3,1);
imshow(b);
subplot(2,3,2);
imshow(b);
a=im2double(b);
[N N]=size(a);
[k n]=meshgrid(0:N-1,0:N-1);
ker=(exp(-(j*(2*pi*(n.*k))/N)))/sqrt(N);
trans1=ker*a*ker;
```

```
d=fftshift(abs(trans1));
subplot(2,3,3);
imshow(d);
invker=(exp(j*2*pi*(n.*k)/N))/sqrt(N);
recon=invker*trans1*invker;
subplot(2,3,4);
imshow(recon);
for i=1:N
for j=1:N
        error(i,j)=(a(i,j)-recon(i,j))^2;
end
end
error1=sum(sum(error))/(N*N);
subplot(2,3,5);
imshow(error1);
```

DISCRETE COSINE TRANSFORM:

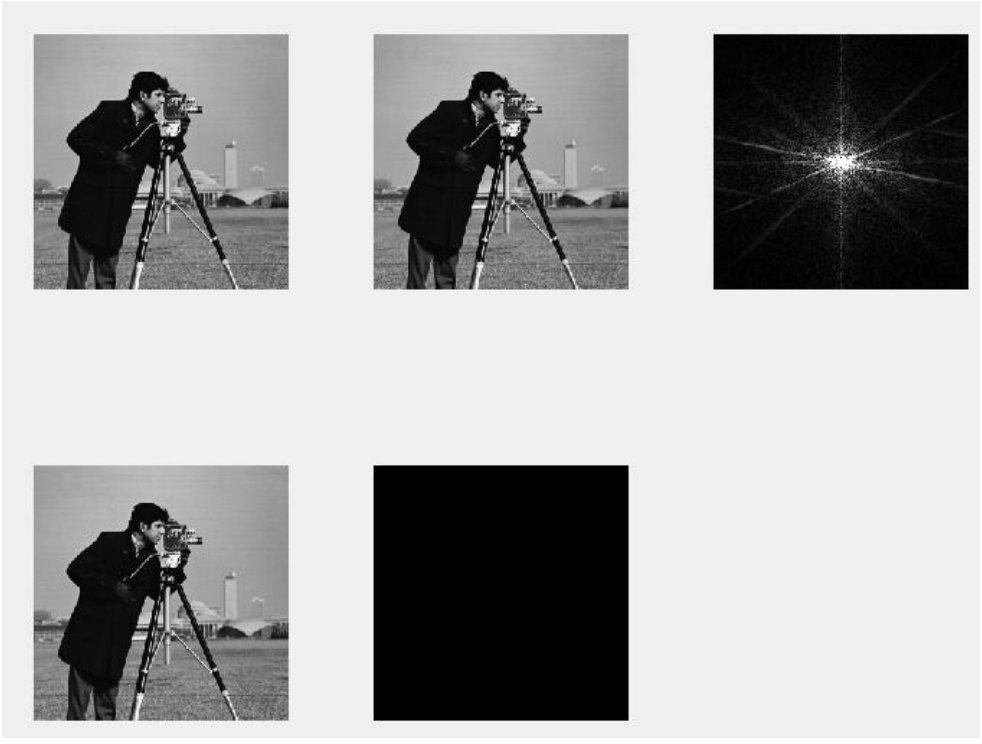
```
clc;
close all;
clear all;
a=imread('peppers.png');
a = imresize(a,[256 256]);
b = rgb2gray(a);
subplot(2,2,1);
imshow(b)
title('original');
c=dct2(b);
subplot(2,2,2);
imshow(c,[0 255])
title('dctimg');
d=idct2(c);
subplot(2,2,3);
imshow(d, [0 255])
title('idctimg');
[N N]=size(b)
for i=1:N
        for j=1:N
        error(i,j)=(b(i,j)-d(i,j))^2;
        end
end
error1=sum(sum(error))/(N*N);
subplot(2,2,4);
imshow(error1);
title('error');
```

DISCRETE WAVELET TRANSFORM:

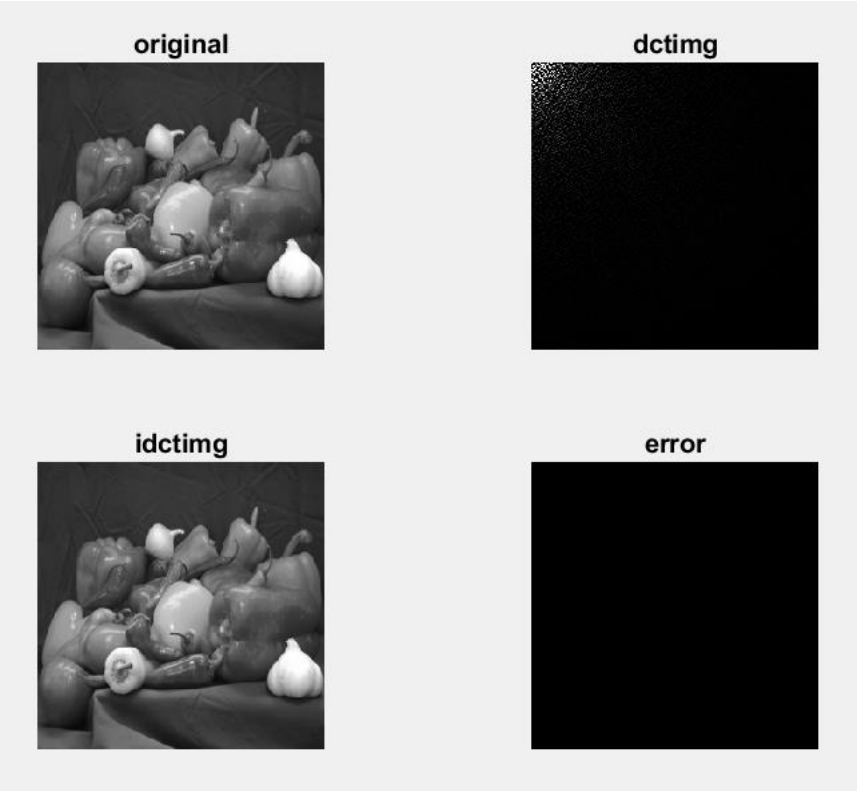
```
clc;
clear all;
close all;
in=imread('peppers.png');
subplot(1,3,1);
imshow(in);
title('Original image');
[a,b,c,d]=dwt2(in,'haar');
p=[uint8(a),b;c,d]
subplot(1,3,2);
imshow(p);
title('First level decomposition');
[a1,a2,a3,a4]=dwt2(a,'haar');
p1=[a1,a2;a3,a4];
p2=[uint8(p1),b;c,d]
subplot(1,3,3);
imshow(p2);
title('Second level decomposition');
```

OUTPUT:

Discrete Fourier Transform



Discrete Cosine Transform



Discrete Wavelet Transform

RESULT:

Thus a program to find the Discrete Fourier Transform, Discrete Cosine Transform and Discrete Wavelet Transform of the input image and reconstruct the image from the transform was written and output verified.

EX NO: 5 IMAGE ENHANCEMENT- SPATIAL DOMAIN FILTERING

DATE:

AIM:

To write a program for removing noise from an image by applying linear and nonlinear filtering and to compare the filtered images.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

THEORY:

Linear Filtering:

The linear filtering is similar to convolution. Hence linear spatial filtering is referred as convolving a filter mask with an image. The filter masks are also called as kernel, template or window.

The response at each pixel of an image is obtained by multiplying each pixel in the neighborhood spanned by the filter mask, by the corresponding filter coefficient and summing the results. For e.g. in averaging filters, the value of every pixel in an image is replaced by the average of pixels contained in the neighborhood defined by the filter mask.

Non Linear Filtering:

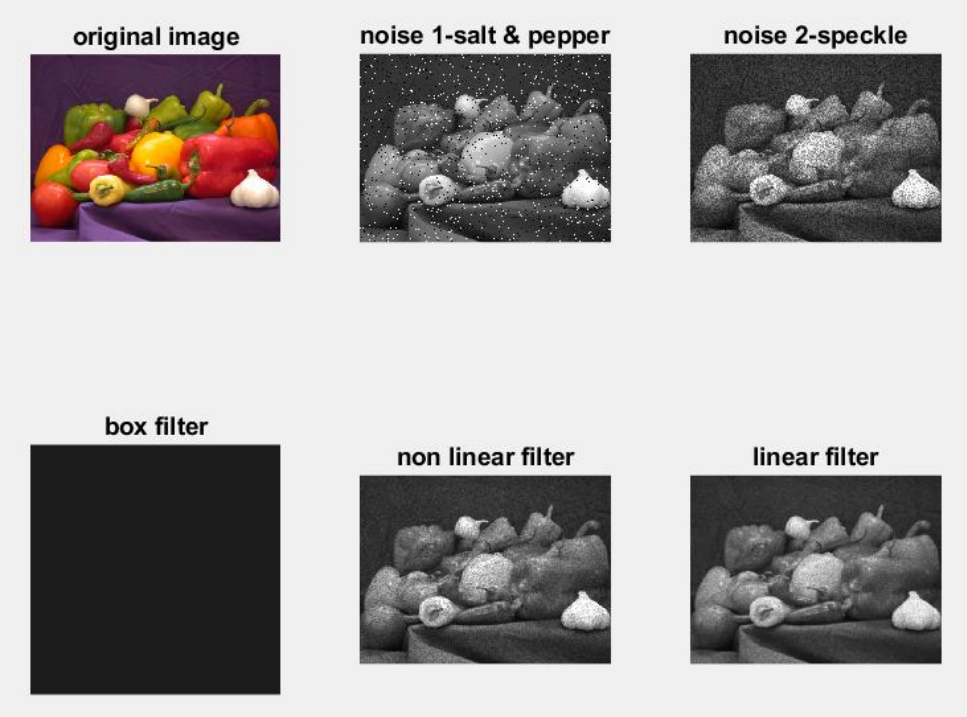
The response is given by the median or variance of gray level in the area spanned by the filter mask. Non-linear filtering which computes the median gray level in the neighbourhood, is used to reduce noise effectively. Median filtering is a useful in reducing salt and pepper noise in an image.

PROGRAM:

```
a =imread('peppers.png');
subplot(2,3,1);
imshow(a);
title('original image');
a=rgb2gray(a);
b = imnoise(a,'salt & pepper')
subplot(2,3,2);
imshow(b);
title('noise 1-salt & pepper');
b = imnoise(a,'speckle')
```

```
subplot(2,3,3);
imshow(b);
title('noise 2-speckle');
c = medfilt2(b);
subplot(2,3,5);
imshow(c);
title('non linear filter');
d=fspecial('average');
subplot(2,3,4);
imshow(d);
title('box filter');
e=imfilter(b,d,'conv');
subplot(2,3,6);
imshow(e);
title('linear filter');
```

OUTPUT:



RESULT:

Thus, a program for removing noise from an image applying linear and nonlinear filtering was written and output verified. It was found that non-linear filter is more effective it removing the noise.

EX.NO:6 IMAGE ENHANCEMENT – FILTERING IN FREQUENCY DOMAIN

DATE:

AIM:

To write a program to perform frequency domain filtering of input image using Butterworth low pass filter and Butterworth high pass filter.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

(i) Butterworth low and high pass filters:

Step 1: Read and display the input image.

Step 2: Find the Fourier transform of input image and display it.

Step 3: Find the Butterworth Low Pass Filter function.

Step 4: Apply Butterworth low pass filter to Fourier transform of input image and displaying the result.

Step 5: Obtain low frequency filtered image in spatial domain by applying inverse Fourier transform to result of step 4.

Step 6: Repeat steps 1 – 5 for Butterworth High pass filter.

THEORY:

There are basically three different kinds of filters: *lowpass*, *highpass* and *bandpass* filters.

(i) Butterworth Low pass filter:

A low-pass filter attenuates high frequencies and retains low frequencies unchanged. It acts as a smoothing filter because the blocked high frequencies correspond to sharp intensity changes *i.e.* fine-scale details and noise in the spatial domain image.

The transfer function of BLPF of order n and cutoff frequency at a distance D_0 from the origin is defined as $H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$.

The BLPF transfer function does not have a sharp discontinuity between passed and filtered frequencies. For such filters with smooth transfer functions, the cutoff frequency is defined as the point where H(u,v) is down to certain fraction (50%) of its maximum value (1).

There is smooth transition in blurring as cutoff frequency increases. Also, there is no ringing.

A Butterworth filter of order 1 has no ringing and negative values. The filter of order 2 shows mild ringing and negative values but less than ideal LPF. As order increases, ringing becomes significant.

In general, BLPFs or order 2 are a good compromise between effective low pass filtering and acceptable ringing characteristics.

(ii) Butterworth High pass filter:

A high pass filter yields edge enhancement or edge detection in the spatial domain, because edges contain many high frequencies. Areas of constant gray level consist of mainly low frequencies and are therefore suppressed.

The transfer function of BLPF of order n and cutoff frequency at a distance D_0 from the origin is defined as $H(u,v) = \frac{1}{1+[D_0/D(u,v)]^{2n}}$.

(iii) Band pass filters:

A band pass attenuates very low and very high frequencies, but retains a middle range band of frequencies. Band pass filtering can be used to enhance edges (suppressing low frequencies) while reducing the noise at the same time (attenuating high frequencies).

PROGRAM:

(i) Butterworth Low pass filter:

```
clc;
clear all;
close all;
im=imread('peppers.png');
subplot(2,3,1);
imshow(im);
title('original image');
im=rgb2gray(im);
ftim=fft2(im);
cftim=fftshift(ftim);
subplot(2,3,2);
imshow(cftim);
title('fourier transform');
```

```
D0=20;
n=2;
[M,N]=size(im);
Cx=round(M/2);
Cy=round(N/2);
H=zeros(M,N);
for i=1:M
        for j=1:N
        d=sqrt((i-Cx).^2+(j-Cy).^2);
        H(i,j)=1/(1+((d/D0).^(2*n)));
        end;
end;
subplot(2,3,3);
imshow(H);
title('H(i,j)');
fftim=cftim.*H;
subplot(2,3,4);
imshow(fftim);
title('fft');
sim=abs(ifft2(fftim));
sim=uint8(sim);
subplot(2,3,5);
imshow(sim);
title('LPF');
```

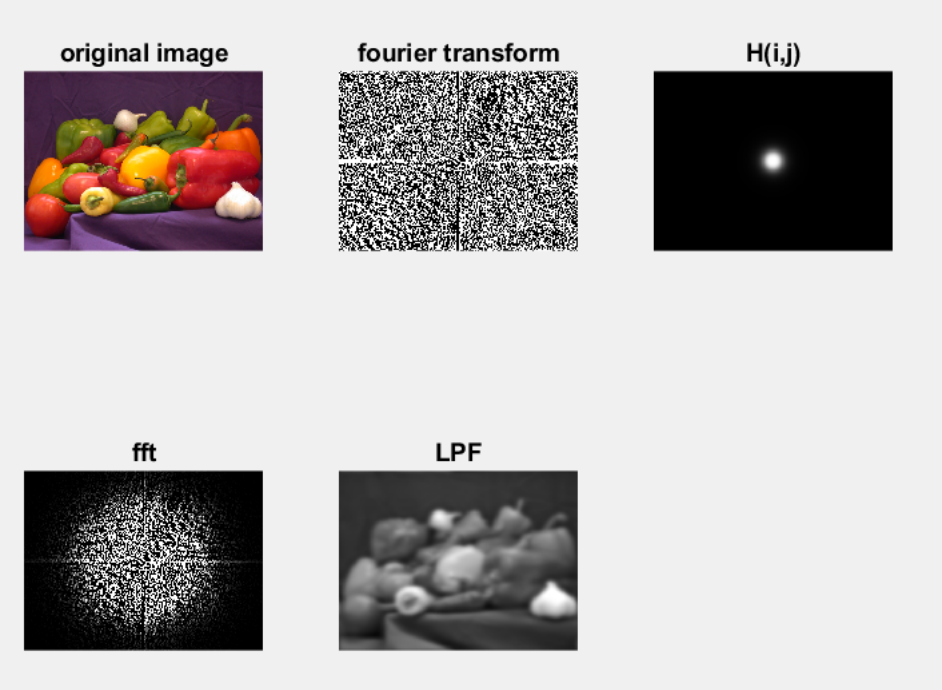
(ii) Butterworth High pass filter:

```
clc;
clear all;
close all;
im=imread('peppers.png');
subplot(2,3,1);
imshow(im);
```

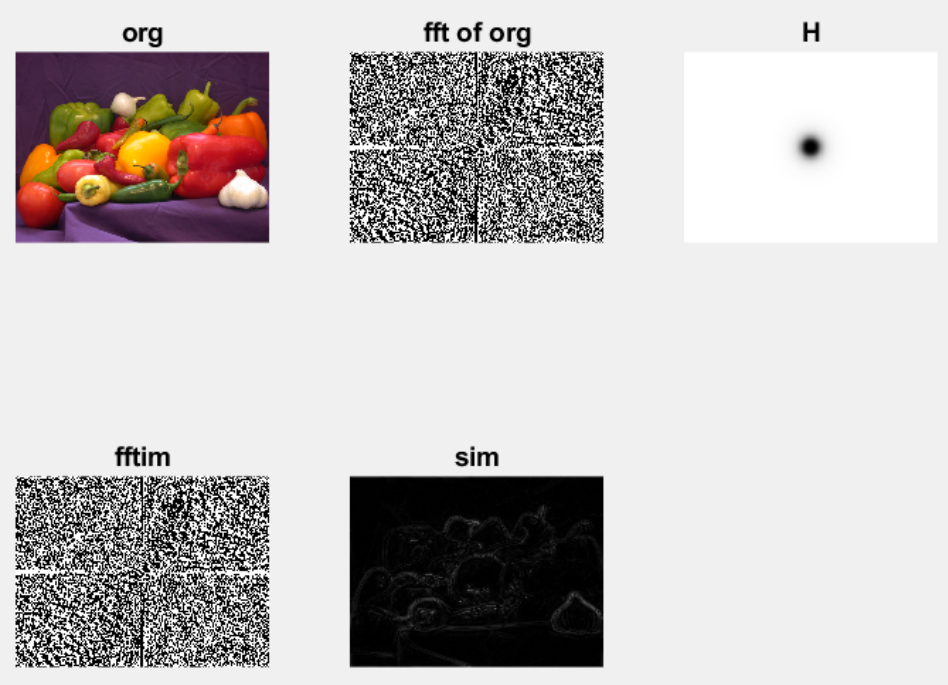
```
title('org')
im=rgb2gray(im);
ftim=fft2(im);
cftim=fftshift(ftim);
subplot(2,3,2);
imshow(cftim);
title('fft of org');
D0=20;
n=2;
[M,N]=size(im);
cx=round(M/2);
cy=round(N/2);
H=zeros(M,N);
for i=1:M
        for j=1:N
        d=sqrt((i-cx).^2+(j-cy).^2);
        H(i,j)=1/(1+((D0/d).^(2*n)));
        end;
end;
%H1=3+log10(double(H));
subplot(2,3,3);
imshow(H);
title('H');
%H=H./max(H(:));

fftim=cftim.*H;
subplot(2,3,4);
imshow(fftim);
title('fftim');
sim=abs(ifft2(fftim));
subplot(2,3,5);
imshow(uint8(sim));
title('sim');
```

OUTPUT:

Low Pass Filter



High Pass Filter



RESULT:

Thus, a program to perform frequency domain filtering of input image using Butterworth low pass filter and Butterworth high pass filter was written and output verified.

EX NO: 7 IMAGE SEGMENTATION – USING EDGE DETECTION OPERATORS

DATE:

AIM:

To write a program for image segmentation using edge detection, line detection and point detection

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

THEORY:

Edge Detection:

Edge detection is the most common approach for detecting meaningful discontinuities in intensity values. An edge is a set of connected pixels that lie on the boundary between two regions. Edge is local concept whereas a region boundary is a global idea.

Edge can also be defined a set of connected points with two-dimensional first order derivative greater than a specified threshold.

An edge can also be defined as a set of connected points falling on the zero crossings of its second order derivative.

Such discontinuities are detected using first and second order derivatives.

First and second order derivatives:

First order derivative: Nonzero along entire ramp and zero in constant gray level areas.

Second order derivative: Nonzero only at the onset and end of the ramp.

From the diagram it can be seen that, for a transition from dark to light, second order derivative is positive at the dark side, -ive at the light side and zero along the ramp and constant gray level areas. It is vice versa for transitions from light to dark.

It is also seen that

(i) Second derivative produces two values for every edge in an image

(ii) Imaginary straight line joining the extreme positive and negative values of the second derivative would cross zero near the midpoint of the edge. This zero crossing property of second derivative is useful in locating the centers of thick edges.

Hence

(i) The magnitude of first derivative is used to detect the presence of edge at a point in an image i.e. to determine if a point is on a ramp.

(ii) The sign of the second derivative can be used to determine whether the edge pixel lies on the dark or light side of an edge.

Method of edge detection:

A point in an image is said to be an edge point if its two-dimensional first-order derivatives is greater than a specified threshold or it falls on the zero crossings of its second order derivative.

Sobel Edge Detector:

Sobel edge detector computes gradient by using the discrete differences between rows and columns of a 3x2 neighborhood where the center pixel in each row or column is weighted by 2 to provide smoothening.

Prewitt Edge Detector:

The Prewitt edge detector uses the mask to approximate digitally the first derivatives. The Prewitt detector tends to produce somewhat noisier results.

Roberts Edge Detector:

Roberts edge detector uses mask to approximate digitally the first derivatives as differences between adjacent pixels. Roberts edge detector is one of the oldest edge detectors and has limited functionality.

Laplacian of a Gaussian Detector:

In a laplacian of a Gaussian filter (LoG), the second derivative is a linear operation , convolving an image with $\Delta 2\ G(x,y)$ is same as convolving the image with the smoothing function first and then computing the laplacian of the result. Convolving results is smoothening of the image and yields double edge image.

Zero cross Edge Detector:

This is based on the concept of LoG method, but the convolution is carried out using a specialized filter function.

Canny Edge Detector:

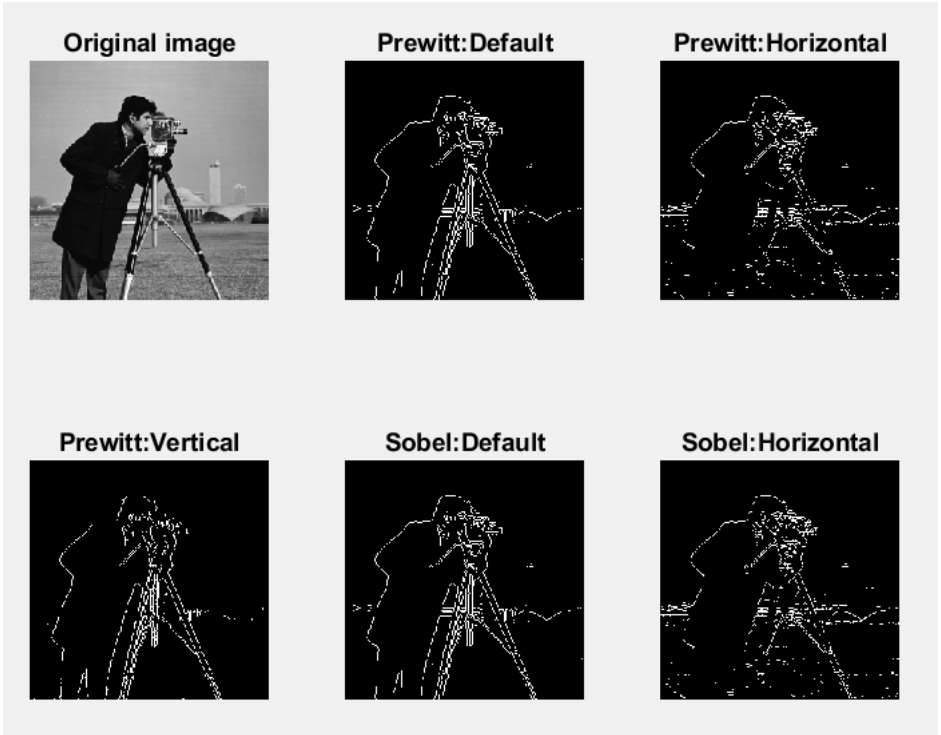
In a canny detector the image smoothed using Gaussian filter with specified standard deviation, to reduce noise. The local gradient and edge gradient are computed at a point. An edge is defined as a point whose strength is locally maximum in the direction of the gradient. The edge points gives rise to ridges in the image. An algorithm tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top. Finally an algorithm performs edge linking by incorporating the weak pixels that are connected to strong pixels.

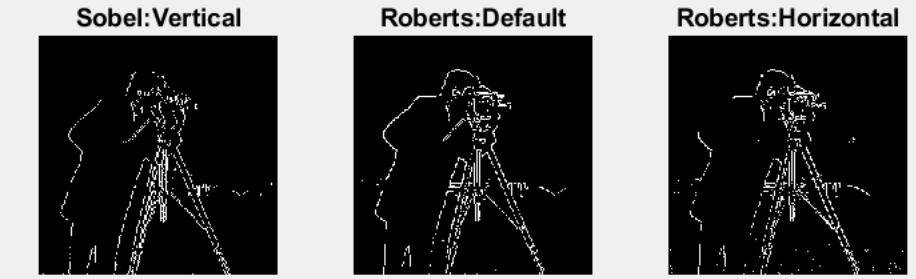
PROGRAM:

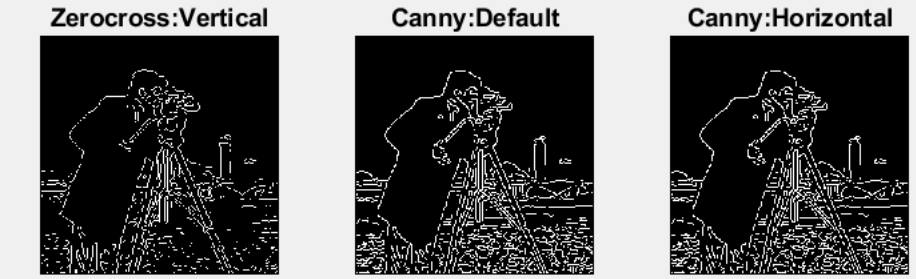
```
clc;
clear all;
close all;
in=imread('cameraman.tif');
subplot(6,3,2);
imshow(in);
title('Original image');
a=edge(in,'prewitt');
subplot(6,3,4);
imshow(a);
title('Prewitt:Default');
a1=edge(in,'prewitt','horizontal');
subplot(6,3,5);
imshow(a1);
title('Prewitt:Horizontal');
a2=edge(in,'prewitt','vertical');
subplot(6,3,6);
imshow(a2);
title('Prewitt:Vertical');
b=edge(in,'sobel');
subplot(6,3,7);
imshow(b);
title('Sobel:Default');
b1=edge(in,'sobel','horizontal');
subplot(6,3,8);
imshow(b1);
title('Sobel:Horizontal');
b2=edge(in,'sobel','vertical');
subplot(6,3,9);
imshow(b2);
title('Sobel:Vertical');
c=edge(in,'roberts');
subplot(6,3,10);
imshow(c);
title('Roberts:Default');
c1=edge(in,'roberts','horizontal');
subplot(6,3,11);
```

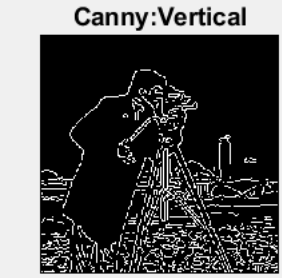
```
imshow(c1);
title('Roberts:Horizontal');
c2=edge(in,'roberts','vertical');
subplot(6,3,12);
imshow(c2);
title('Roberts:Vertical');
d=edge(in,'zerocross');
subplot(6,3,13);
imshow(d);
title('Zerocross:Default');
d1=edge(in,'zerocross','horizontal');
subplot(6,3,14);
imshow(d1);
title('Zerocross:Horizontal');
d2=edge(in,'zerocross','vertical');
subplot(6,3,15);
imshow(d2);
title('Zerocross:Vertical');
e=edge(in,'canny');
subplot(6,3,16);
imshow(e);
title('Canny:Default');
e1=edge(in,'canny','horizontal');
subplot(6,3,17);
imshow(e1);
title('Canny:Horizontal');
e2=edge(in,'canny','vertical');
subplot(6,3,18);
imshow(e2);
title('Canny:Vertical');
```

OUTPUT:









RESULT:

Thus, a program for edge detection using various operators has been written and executed

EX.NO: 8 **BASIC MORPHOLOGICAL OPERATIONS**
DATE:

AIM:

To perform segmentation of an image using Watershed Transform.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

1. Read and display the input image.
2. Determine the gradient of the input image.
3. Create structuring elements
4. Apply the structuring elements over the images
5. Record the output.

$f \ominus b$

THEORY:

Erosion **(): (f $\ominus$ b)**

The erosion of an image f at any location (x,y) by a flat structuring element b is defined as the minimum value of the image in the region coincident with b when the origin of b is at (x,y).

Dilation ($f \oplus b$)**:**

The dilation of an image f at any location (x,y) by a flat structuring element b is defined as the maximum value of the image in the region coincident with $\hat{b}$ when the origin of $\hat{b}$ is at (x,y).

Opening:

The opening of image f by structure element b is defined as erosion of f by b followed by the dilation of the result with b.

$$f \circ b = (f \ominus b) \oplus b$$

The opening decreases intensity of all bright features and removes small bright details while leaving overall intensity levels and larger bright features relatively undistributed.

Closing:

The closing of image f by structure element b is defined as dilation of f by b followed by the erosion of the result with b.

$$f \bullet b = (f \oplus b) \ominus b$$

The bright details and background are relatively unaffected, but the dark features were attenuated.

PROGRAM:

MORPHOLOGICAL OPERATIONS:

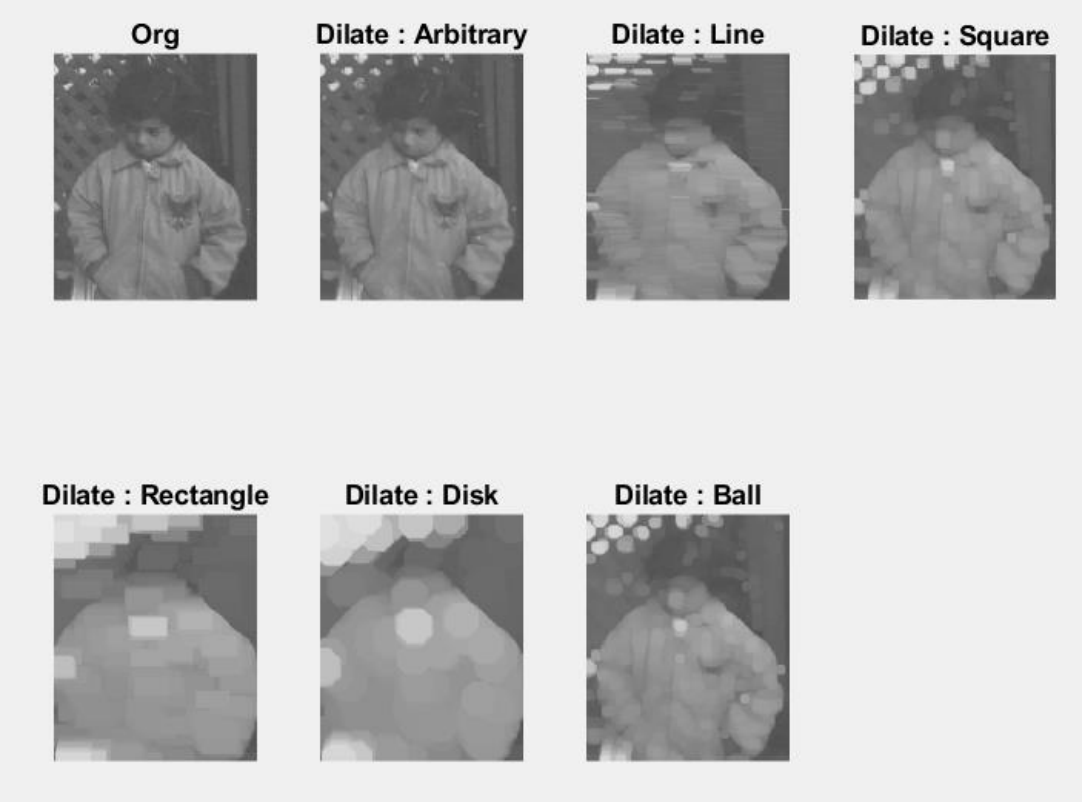
```
clc;
clear all;
close all;
im = imread('pout.tif');
se1 = strel('arbitrary',[1 1 1 ; 0 1 0 ; 1 1 1]);
se2 = strel('line',20,0);
se3 = strel('square', 10);
se4 = strel('rectangle',[20 40]);
se5 = strel('disk', 20);
se6 = strel('ball', 5, 4);
figure(1);
subplot(2,4,1);
imshow(im);
title('Org');
subplot(2,4,2);
imshow(imdilate(im, se1));
title('Dilate : Arbitrary');
subplot(2,4,3);
imshow(imdilate(im, se2));
title('Dilate : Line');
subplot(2,4,4);
imshow(imdilate(im, se3));
title('Dilate : Square');
subplot(2,4,5);
imshow(imdilate(im, se4));
title('Dilate : Rectangle');
subplot(2,4,6);
imshow(imdilate(im, se5));
title('Dilate : Disk');
subplot(2,4,7);
imshow(imdilate(im, se6));
title('Dilate : Ball');
figure(2);
subplot(2,4,1);
```

```
imshow(im);
title('Org');
subplot(2,4,2);
imshow(imerode(im, se1));
title('Erode : Arbitrary');
subplot(2,4,3);
imshow(imerode(im, se2));
title('Erode : Line');
subplot(2,4,4);
imshow(imerode(im, se3));
title('Erode : Square');
subplot(2,4,5);
imshow(imerode(im, se4));
title('Erode : Rectangle');
subplot(2,4,6);
imshow(imerode(im, se5));
title('Erode : Disk');
subplot(2,4,7);
imshow(imerode(im, se6));
title('Erode : Ball');
figure(3);
subplot(2,4,1);
imshow(im);
title('Org');
subplot(2,4,2);
imshow(imopen(im, se1));
title('Open : Arbitrary');
subplot(2,4,3);
imshow(imopen(im, se2));
title('Open : Line');
subplot(2,4,4);
imshow(imopen(im, se3));
title('Open : Square');
subplot(2,4,5);
imshow(imopen(im, se4));
title('Open : Rectangle');
subplot(2,4,6);
imshow(imopen(im, se5));
title('Open : Disk');
subplot(2,4,7);
imshow(imopen(im, se6));
title('Open : Ball');
figure(4);
subplot(2,4,1);
imshow(im);
title('Org');
subplot(2,4,2);
imshow(imclose(im, se1));
title('Close : Arbitrary');
subplot(2,4,3);
imshow(imclose(im, se2));
```

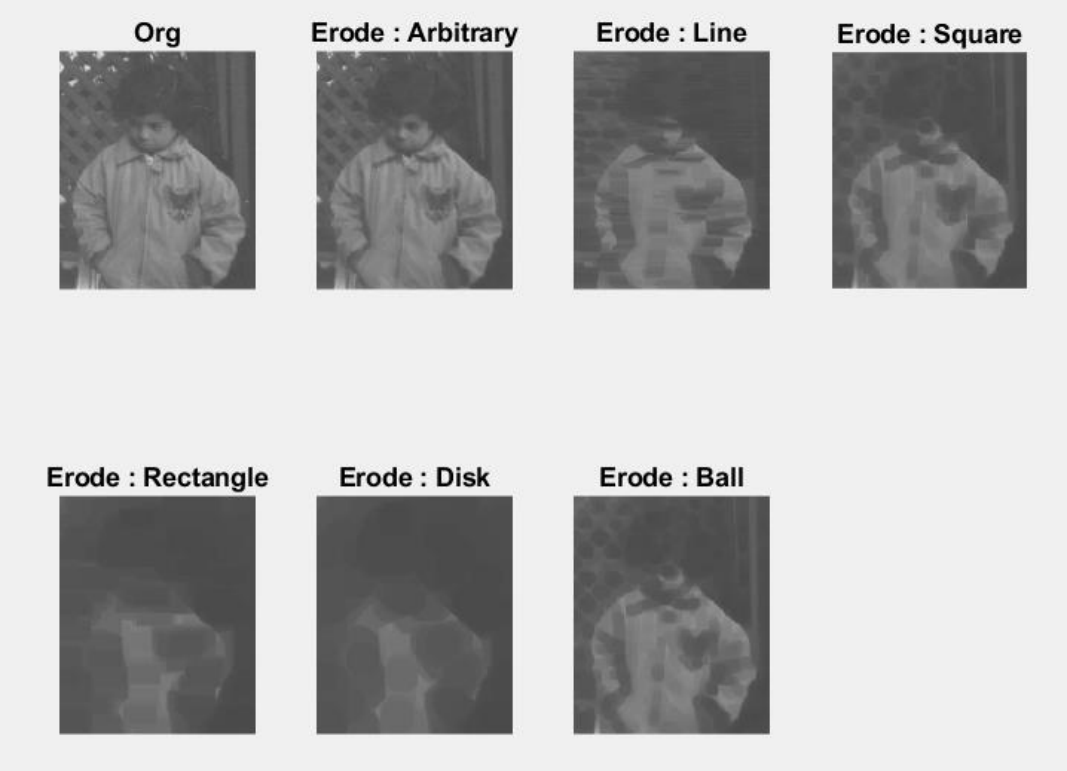
```
title('Close : Line');
subplot(2,4,4);
imshow(imclose(im, se3));
title('Close : Square');
subplot(2,4,5);
imshow(imclose(im, se4));
title('Close : Rectangle');
subplot(2,4,6);
imshow(imclose(im, se5));
title('Close : Disk');
subplot(2,4,7);
imshow(imclose(im, se6));
title('Close : Ball');
```

OUTPUT:

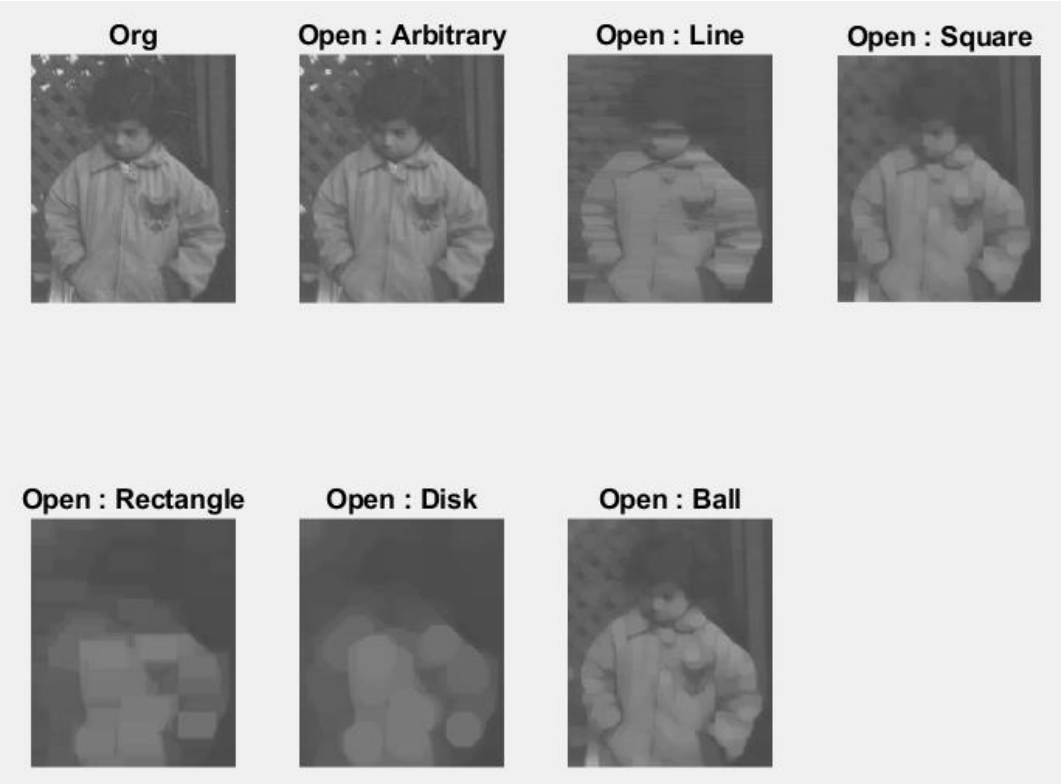
DILATE



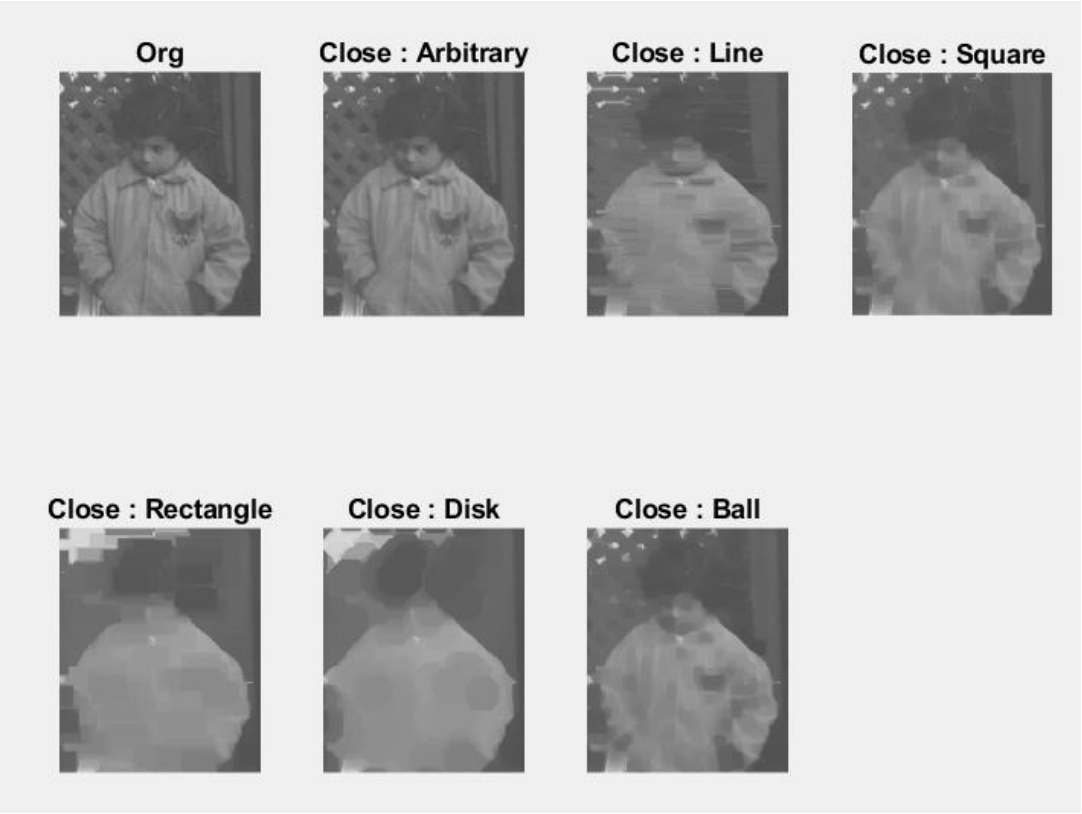
ERODE


Org
Erode : Arbitrary
Erode : Line
Erode : Square
Erode : Rectangle
Erode : Disk
Erode : Ball

OPEN


Org
Open : Arbitrary
Open : Line
Open : Square
Open : Rectangle
Open : Disk
Open : Ball

CLOSE



RESULT:

Thus, a program to perform segmentation of an image using Watershed Transform was written and output was verified.

EX NO: 9 ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS AND COLOR PLANES
DATE:

AIM:

To convert an image between various color spaces.

MATERIALS REQUIRED:

MATLAB R2009b, Personal Computer.

ALGORITHM:

Step 1. Read a colour input image.

Step 2. Convert the RGB image into HSV image using function rgb2hsv() and display it.

Step 3. Convert the RGB image into YCbCr image using function rgb2ycbcr() and display it.

Step 4. Convert the RGB image into NTSC image using function rgb2ntsc() and display it.

THEORY:

Color Space:

A colour space is a method by which we can specify, create and visualise colour. A colour is thus usually specified using three co-ordinates, or parameters. These parameters describe the position of the colour within the colour space being used. They do not tell us what the colour is, that depends on what colour space is being used. The various color spaces exist because they present color information in ways that make certain calculations more convenient or because they provide a way to identify colors that is more intuitive.

YIQ Color Space

NTSC is the color space used for television broadcast in the United States, and the only space among the above-mentioned ones that realizes a complete separation between the luminance and the chrominance information. NTSC has this property because when it was introduced, it had to separate the information used by the monochrome TV receivers from the supplementary one used by color receivers. The components of the NTSC color space are Y

(the luminance component), I (the cyan-orange component), and Q (the green-purple component). The first component, luminance, represents grayscale information, while the last two components make up chrominance (color information).

YCbCrColor Space

The YCbCrcolor space is widely used for digital video. In this format, luminance information is stored as a single component (Y), and chrominance information is stored as two color-difference components (Cb and Cr). Cb represents the difference between the blue component and a reference value. Cr represents the difference between the red component and a reference value. For uint8 images, the data range for Y is [16, 235], and the range for Cb and Cr is [16, 240].

HSV Color Space

The HSV color space (Hue, Saturation, Value) is often used by people who are selecting colors (e.g., of paints or inks) from a color wheel or palette, because it corresponds better to how people experience color than the RGB color space does. As hue varies from 0 to 1.0, the corresponding colors vary from red through yellow, green, cyan, blue, magenta, and back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter.

PROGRAM:

COLOR MODELS:

```
clc;
clear all;
close all;
a=imread('football.jpg');
subplot(2,2,1);
imshow(a);
title('original');
b=rgb2ntsc(a);
subplot(2,2,2);
imshow(b);
title('ntsc');
c=rgb2hsv(a);
subplot(2,2,3);
imshow(c);
title('hsv');
```

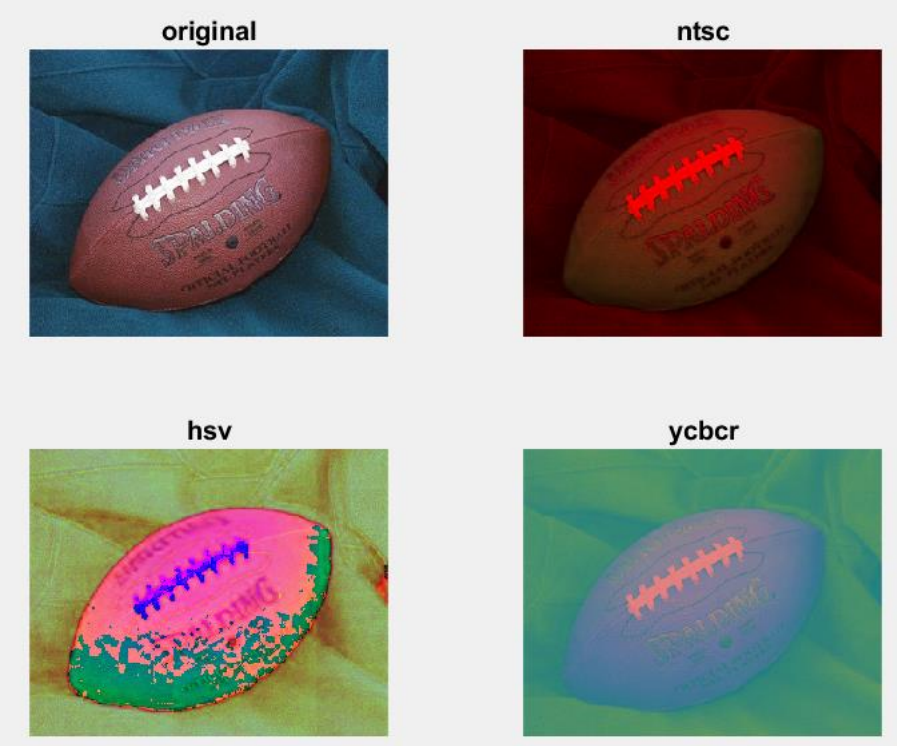
```
d=rgb2ycbcr(a);
subplot(2,2,4);
imshow(d);
title('ycbcr');
```

COLOR PLANES:

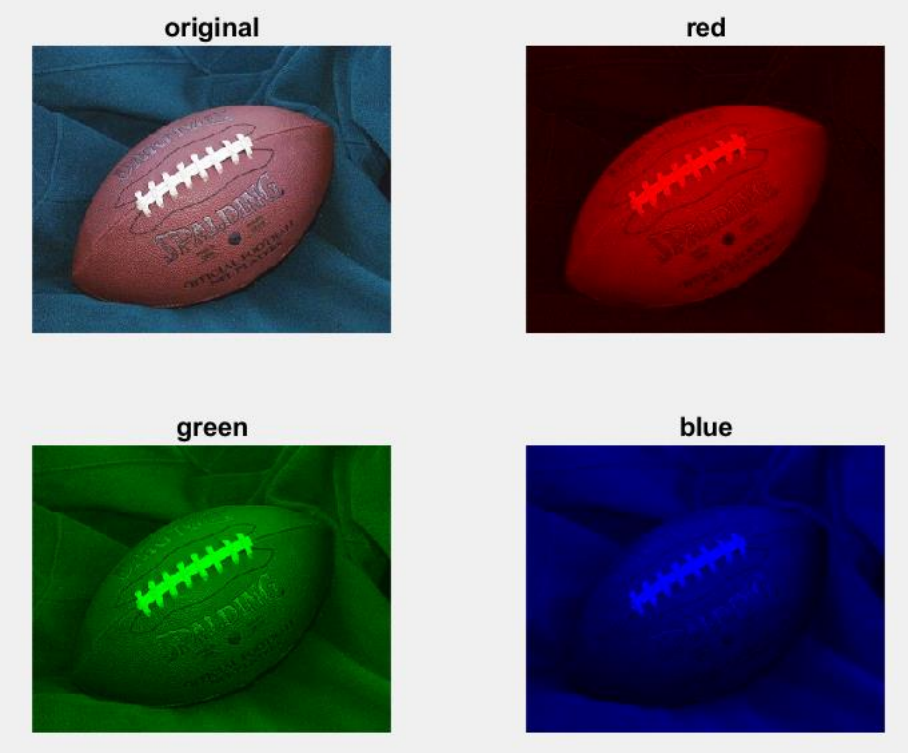
```
clc;
clear all;
close all;
a=imread('football.jpg');
subplot(2,2,1);
imshow(a);
title('original');
[r c d]=size(a);
z=zeros(r,c);
temp=a;
temp(:,:,2)=z; temp(:,:,3)=z;
subplot(2,2,2);
imshow(temp);
title('red');
z1=zeros(r,c);
temp=a;
temp(:,:,1)=z1; temp(:,:,3)=z1;
subplot(2,2,3);
imshow(temp);
title('green');
z2=zeros(r,c);
temp=a;
temp(:,:,1)=z1; temp(:,:,2)=z1;
subplot(2,2,4);
imshow(temp);
title('blue');
```

OUTPUT:

COLOR MODELS:



COLOR PLANES:



RESULT:

Thus, a program to convert the given RGB image into various colour spaces was written and output verified.

EX.NO:10 **FILTERING IN FREQUENCY DOMAIN –**

DATE: **INVERSE AND WIENER FILTERING**

AIM:

To write a program to restore a degraded image using Inverse filter and Wiener filter.

ALGORITHM:

(i) Inverse filter:

Step 1: Read and display the input image.

Step 2: Determine the Fourier transform of input image.

Step 3: Define a degradation function and find its Fourier transform. .

Step 4: Degrade the input image by multiplying it by FT of degradation function.

Step 5: Restore the degraded input image using inverse filter and display the result.

(ii) Wiener filter:

Step 1: Read and display the input image.

Step 2: Determine the Power spectrum of input image and center it.

Step 3: Degrade (blur) the input image and add noise to it.

Step 4: Restore the degraded image Wiener filter algorithm and display the restored image.

Step 5: Determine the power spectrum of restored image and display it.

THEORY:

(i) Inverse filter:

The estimate $\hat{F}(u,v)$ of the Fourier transform F(u,v) of the original image f(x,y) is obtained by simply dividing the transform G(u,v) of the degraded image g(x,y) by the degradation function H(u,v)

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)} \qquad (1)$$

However $$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

Hence $$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}$$

Thus we cannot find the estimate $\hat{F}(u,v)$ of the Fourier transform and recover the undegraded image f(x,y) exactly, even if we know the degradation function H(u,v) because N(u,v) is a random function whose Fourier transform is not known. Moreover if the value of H(u,v) is zero or very small, then ratio $\frac{N(u,v)}{H(u,v)}$ will dominate the estimate $\hat{F}(u,v)$ and also $\hat{F}(u,v)$ becomes infinite.

Thus the inverse filter eliminates blur in the image but noise still dominates.

Weiner filter:

Weiner filtering is a method of restoring images in the presence of blur and noise. The wiener filtering incorporates both degradation function and statistical characteristics of noise into the restoration process unlike inverse filter which does not take noise into account.

An estimate $\hat{f}$ of original input image f is found from the degraded image such that the mean square error $e^2 = E\{(f - \hat{f})^2\}$ (1)

between them is minimized.

Where E{.} is the expected value of the argument. It is assumed that

(i) Noise and the image are uncorrelated

(ii) Noise or image has zero mean

(iii) Gray levels in the estimate are linear function of the gray levels in the degraded image.

The frequency domain representation $\hat{F}(u,v)$ of the estimate $\hat{f}(x,y)$ is given by

$$\hat{F}(u,v) = \frac{H^*(u,v)S_f(u,v)}{S_f(u,v)|H(u,v)|^2 + S_\eta(u,v)} G(u,v) \quad (2)$$

$$\hat{F}(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)} G(u,v) \quad (3)$$

$$\hat{F}(u,v) = [\frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}]G(u,v) \quad (4)$$

Where H(u,v) is transform of the degradation function

H*(u,v) is complex conjugate of H(u,v)

$|H(u,v)|^2 = H^*(u,v)\, H(u,v)$

$S_\eta(u,v) = |N(u,v)|^2$ is power spectrum of the noise

$S_f(u,v) = |F(u,v)|^2$ is power spectrum of undegraded image

G(u,v) is the transform of the degraded image.

The restored image $\hat{f}(x, y)$ in the spatial domain is given by the inverse Fourier transform of the frequency-domain estimate $\hat{F}(u, v)$. The wiener filter consists of terms inside the brackets. The problem occurring in the inverse filter when the H(u,v) = 0 is absent unless both H(u,v) and $S_\eta(u,v)$ are zero for the same values of u and v. If noise is zero, the noise power spectrum $S_\eta(u,v)$ is zero and wiener filter reduces to an inverse filter.

For spectrally white noise, the spectrum $S_\eta(u,v) = |N(u,v)|^2$ is a constant. However power spectrum $S_f(u,v)$ of undegraded image is not known often. In such cases where these quantities are not known or cannot be estimated

$$\hat{F}(u,v) = [\frac{1}{H(u,v)} \quad \frac{|H(u,v)|^2}{|H(u,v)|^2 + K} \quad]G(u,v) \qquad (5)$$

where K is a constant chosen interactively to yield the best visual results. It is usually chosen as $2\sigma^2$ where σ^2 is the noise variance.

The Wiener filter produces slightly poor results for high and medium noise cases.

PROGRAM:

(i) Inverse filter:

```
clc;
close all;
clear all;

%reading and display of input image
f=imread('mri-1.jpg');
f=rgb2gray(f);
subplot(3,2,1), imshow(f),title('Original Image');

%Determination of Fourier transform of input image
F = fft2(f);

%Determination of degradation function
h = ones(29,29)./121;
subplot(3,2,2), imshow(h,[]),title('Degradation function');

%Determination of fourier transform of degradation function
[M N]=size(f);
H = fft2(h,M,N);
subplot(3,2,3), imshow(H),title('Fourier transform of Degradation function');
 %Degradation of input image
G = H.*F;
g = ifft2(G);
subplot(3,2,4),imshow(uint8(g)),title('Degraded Image');
```

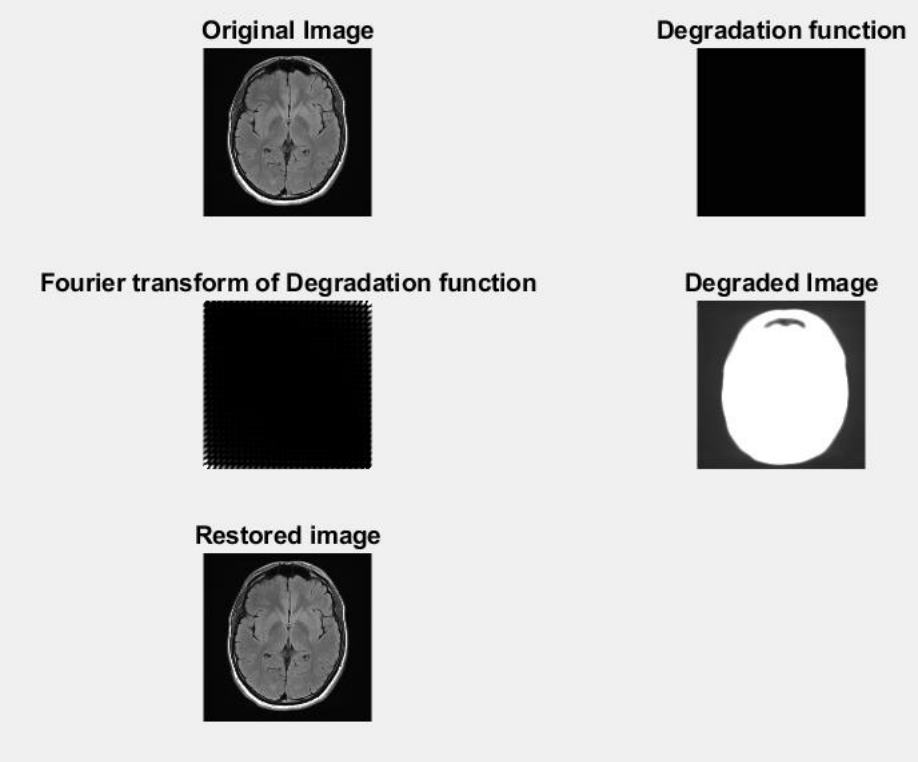
```
%restoration of degraded input image
RF = G./H;
Rf = ifft2(RF);
subplot(3,2,5),imshow(uint8(Rf)),title('Restored image');
```

ii) Wiener filter:

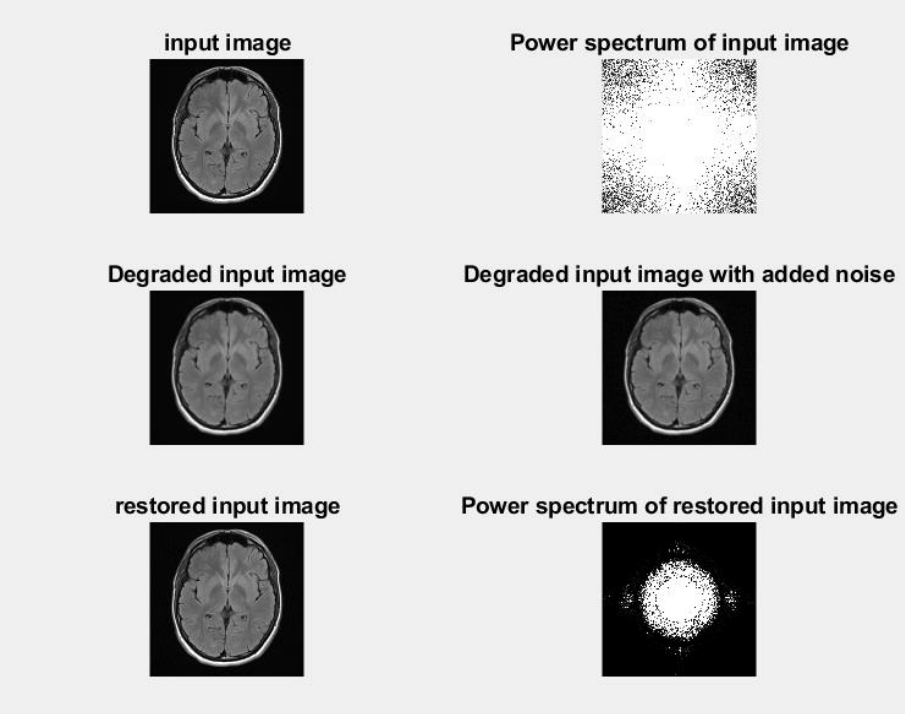
```
clc;
close all;
clear all;
%reading and display of input image
in=imread('mri-1.jpg');
in = im2double(in);
subplot(3,2,1);
imshow(in),title('input image');

%Determination of Power spectrum of input image
%Power spectrum of image is equal to square of Fourier transform of image
%Power spectrum is then centered
cf = abs(fft2(in)).^2;
ccf = fftshift(cf);
subplot(3,2,2);
imshow(ccf), title('Power spectrum of input image');
 %Degradation of input image
h = fspecial('disk',4);
in_blur = imfilter(in,h,'circular');
subplot(3,2,3);
imshow(in_blur), title('Degraded input image');


%Generation of noise
sigma = 0.01;
noise = sigma*randn(size(in));

%Adding noise to degraded image
in_blur_noise = in_blur + noise;
subplot(3,2,4);
imshow(in_blur_noise), title('Degraded input image with added noise');

%Determination of noise to signal ratio (nsr)
nsr=numel(in)*sigma^2./cf;

%Restoration of image degraded with degradation function 'h' and 'noise',
%by deconvolving the degraded image using Wiener filter algorithm.
res_in = deconvwnr(in_blur_noise,h,nsr);
subplot(3,2,5), imshow(res_in), title('restored input image');

%Determination of power spectrum of restored input image
cf1 = abs(fft2(res_in)).^2;
ccf1= fftshift(cf1);
subplot(3,2,6), imshow(ccf1), title('Power spectrum of restored input image');
```

OUTPUT:

(i) Inverse filter:



ii) Wiener filter:



RESULT:

Thus, a program to restore a degraded image using Inverse filter and Wiener filter is demonstrated.

MINI PROJECT