

**DEPARTMENT OF BIOMEDICAL ENGINEERING**

Course code -ECA36

Course Name- Digital Image processing

LAB MANUAL

**ECA3646 – DIGITAL IMAGE PROCESSING LABORATORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Exp. No.** | **Date** | **INDEX** | **Pg.**  **No.** | **Marks** | **Staff Sign** |
| **1.** |  |  |  |  |  |
| **2.** |  |  |  |  |  |
| **3.** |  |  |  |  |  |
| **4.** |  |  |  |  |  |
| **5.** |  |  |  |  |  |
| **6.** |  |  |  |  |  |
| **7.** |  |  |  |  |  |
| **8.** |  |  |  |  |  |
| **9.** |  |  |  |  |  |
| **10.** |  |  |  |  |  |
| **11.** |  |  |  |  |  |
| **12.** |  |  |  |  |  |
| **13.** |  |  |  |  |  |
| **14.** |  |  |  |  |  |
| **15.** |  |  |  |  |  |
| **16.** |  |  |  |  |  |

# Ex. No. 1

## AIM:

**Task 1:**

# IMAGE SAMPLING AND QUANTIZATION

To write a Matlab program to do sampling process of input image using sampling techniques. Various sampling techniques processed are image resizing, image zooming, image cropping.

## Task 2:

To write computer code using Matlab software for quantization using dithering techniqueand subplot the output image (with dithering and without dithering).

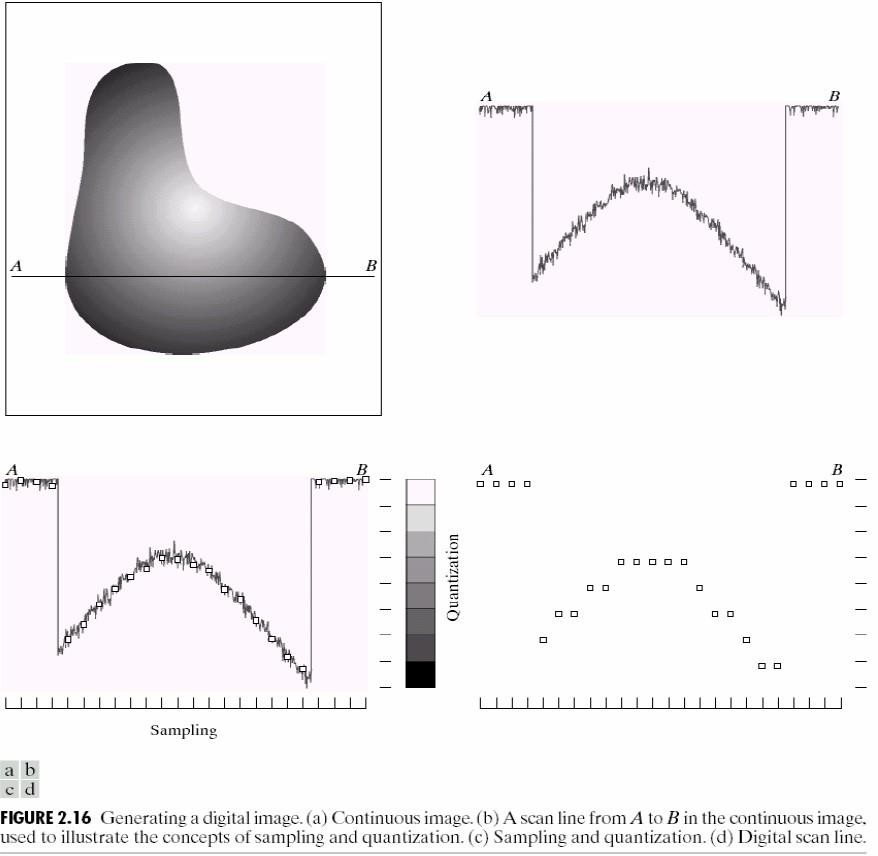
## SOFTWARE REQUIRED:

MATLAB R2017a

## THEORY:

In order to become suitable for computer processing, an image function f(x,y) must be digitized both spatially and in amplitude. A continuous image f(x,y) is normally approximated by equally spaced samples arranged in the form of an N x M array where each element of the array is a discrete quantity.

## CONCEPT OF SAMPLING AND QUANTIZATION:



**Image Resizing**

## SYNTAX

**B = imresize(A,m)**

It Returns image B that is m times the size of A. A can be an indexed image, grayscale image, RGB, or binary image. If m is between 0 and 1.0, B is smaller than A. If m is greater than 1.0, B is larger than A. When resizing the image, imresize uses nearest-neighbor interpolation.

## Image Cropping SYNTAX

**I2 = imcrop(I,rect)**

In the syntax above imcrop crops an input image I to a specified rectangle.**rect** is a four- element vector with the form [xmin ymin width height]; these values are specified in spatial coordinates.**rect** is specified in terms of spatial coordinates, the width and height elements of **rect** do not always correspond exactly with the size of the output image.

For example, suppose **rect** is [20 20 40 30], using the default spatial coordinate system. The upper-left corner of the specified rectangle is the center of the pixel (20,20) and the lower- right corner is the center of the pixel (50,60). The resulting output image is 31-by-41, not 30- by-40, because the output image includes all pixels in the input image that are completely or partially enclosed by the rectangle.

## Image Rotate SYNTAX

**B = imrotate(A,angle)**

This function of sampling process rotates image A by angle degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for angle. imrotate makes the output image B large enough to contain the entire rotated image. imrotate uses nearest neighbor interpolation, setting the values of pixels in B that are outside the rotated image to 0 (zero). The input image can be numeric or logical. The output image is of the same class as the input image.

## PROGRAM:

1. **Image sampling Resizing the input image**

clc; clear all; close all;

im=imread('Tulips.jpg'); im7=imshow(im);

im1=imresize(im,[1024 1024]); im2=imresize(im1,[1024 1024]/2); im3=imresize(im1,[1024 1024]/4); im4=imresize(im1,[1024 1024]/8); im5=imresize(im1,[1024 1024]/16); im6=imresize(im1,[1024 1024]/32); figure;

imshow(im1); figure; imshow(im2); figure; imshow(im3); figure; imshow(im4); figure; imshow(im5); figure; subplot(2,3,1); imshow(im1); title('1024');

subplot(2,3,2); imshow(im2); title('512');

subplot(2,3,3); imshow(im3); title('256');

subplot(2,3,4); imshow(im4); title ('128');

subplot(2,3,5); imshow(im5); title('64');

subplot(2,3,6); imshow(im6); title('32');

## Image zooming and shrinking

1. **Shrinking**

clc; clear all; close all;

A=imread('cameraman.tif'); display('input image==>cameraman.tif');

f=input('enter the shrinking factor of the image:'); s=size(A);

s1=s/f; k=1; l=1;

for i=1:s1 for j=1:s1

B(i,j)=A(k,l);

l=l+f; end l=1;

k=k+f; end figure;

imshow(A); title('original image'); figure;

imshow(B); title('shrinked version');

## Zooming

clc; clear all; close all;

A=imread('cameraman.tif'); display('Input Image==>cameraman.tif');

f1=input('enter the factor by which the image is to be zoomed'); s=size(A);

s2=s\*f1; k=1; l=1;

for (i=1:f1:s2) for (j=1:f1:s2)

C(i,j)=A(k,l);

l=l+1; end

l=1;

k=k+1; end

for (i=1:f1:s2)

for (j=2:f1:s2-1)

C(i,j)=[C(i,j-1)+C(i,j+1)]\*0.5;

end end

for (j=1:f1:s2) for (i=2:f1:s2-1)

C(i,j)=[C(i-1,j)+C(i+1,j)]\*0.5;

end end

for (i=2:f1:s2-1)

for(j=2:f1:s2-1)

C(i,j)=[C(i,j-1)+C(i,j+1)]\*0.5;

end end figure

imshow(C); title('zoomed image');

## Image cropping

clc; clear all; close all;

i=imread('cameraman.tif'); subplot(2,2,1);

imshow(i); title('original image');

j=imcrop(i,[50,100,100,300]);

subplot(2,2,2); imshow(j); title('cropped image');

## Image with degree of rotation

clc; clear all; close all;

a=imread('penguins.jpg'); subplot(2,3,1:2:3); imshow(a);

title('original image'); subplot(2,3,4); imshow(imrotate(a,30)); title('30 degree rotated image'); subplot(2,3,5); imshow(imrotate(a,60)); title('60 degree rotated image'); subplot(2,3,6); imshow(imrotate(a,90)); title('90 degree rotated image');

## Image quantization

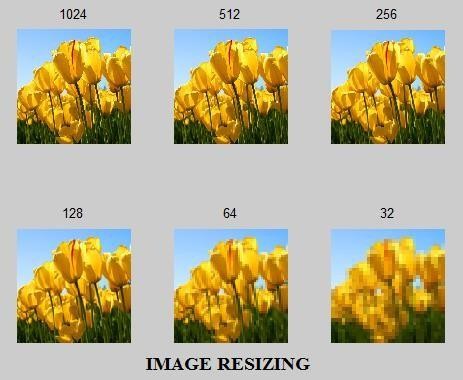
clc; clear all; close all;

rgb=imread('onion.png'); subplot(2,2,1); imshow(rgb);

[X\_no\_dither,map]=rgb2ind(rgb,8,'nodither'); subplot(2,2,2);

imshow(X\_no\_dither,map); [X\_dither,map]=rgb2ind(rgb,8,'dither'); subplot(2,2,3); imshow(X\_dither,map);

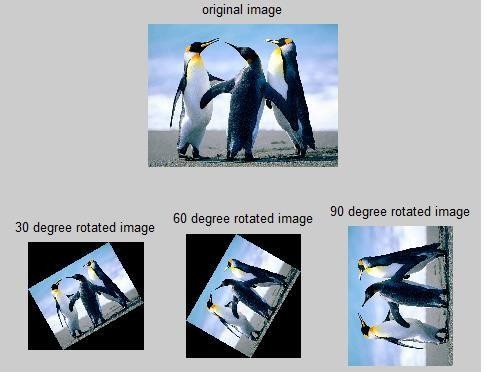
# OUTPUT:

**(a)**

# ORI.pngZOOM.jpg(b)

**(c)**

# (d)

**(e)**

## RESULT:

Sampling and Quantization techniques were applied in different input images and outputs areplotted and analyzed using MATLAB software successfully.

# Ex. No. 2

**ANALYSIS OF SPATIAL AND INTENSITY RESOLUTION OF IMAGES**

## AIM:

To analyze spatial and intensity resolution of given input image using MATLAB

software. The analyses to perform are

* 1. The effect of changing the number of gray levels on the quality of image.
  2. The effect of changing spatial resolution on the quality of image.

**SOFTWARE REQUIRED:**

MATLAB R2017a

## THEORY:

**Changing the number of gray Levels (Intensity Resolution):**

The quality of a gray-level image is significantly affected by its gray-levelresolution. Other word, increasing the number of bits per pixel has a greateffect in improving the quality of gray-level images. This is because that ahigher number of gray levels would give a smooth transition along thedetails of the image and hence improving its quality to the human eye.

## Changing Spatial Resolution on Indexed image:

Changing the spatial resolution of a digital image, by zooming or shrinking, is anoperation of great importance in a wide range of applications (i.e. in digitalcameras, biomedical image processing and astronomical images). Also spatial resolution can be performed using indexing of input image.

## Note: Zooming and Shrinking technique can also used in image processing of spatial resolution in an original image.

**COMMANDS USED:**

## [X,map] = gray2ind(I,n)

**[X,map] = gray2ind(I,n)** converts the grayscale image I to an indexed image X. n specifies the size of the colormap, gray(n). n must be an integer between 1 and 65536. If n is omitted, it defaults to 64.

## B = imresize(A,m)

It Returns image B that is m times the size of A. A can be an indexed image, grayscale image, RGB, or binary image. If m is between 0 and 1.0, B is smaller than A. If m is greater than 1.0, B is larger than A. When resizing the image, imresize uses nearest-neighbor interpolation.

## PROGRAM:

1. **changing the intensity gray resolution from 256 to 2.**

I=imread('cameraman.tif'); [r,c]=size(I); I2=uint8(zeros(r,c));

for i=1:r for j=1:c

if (I(i,j)>128)

I2(i,j)=256;

else I2(i,j)=1;

end end end figure;

subplot(1,2,1); imshow(I); title('original image'); subplot(1,2,2); imshow(I2);

title('change in intensity resolution');

## Changing the spatial resolution from 1-bit image to 8-bit image

clc; clear all; close all;

A=imread('onion.png'); C=rgb2gray(A); [B,map]=gray2ind(C,2^4); nImage=ind2gray(B,map); subplot(2,1,1); imshow(C);

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

title('change in spatial resolution'); whos A;

whos C; whos nImage;

# OUTPUT:

**(a)**

# (b)

## RESULT:

Analysis spatial and intensity resolution of given input image using MATLAB software are performed and output are plotted successfully.

# Ex. No.3

**INTENSITY TRANSFORMATION OF IMAGES**

## AIM:

**Task 1:**

To write a computer program to plot Photographic Negative and Logarithmic Transformation with given constant of c = 1, 2, 5 on input image.

## Task 2:

Write Matlab code to do Gamma Transformations, on the grayscale components of given input image with gamma value as 1, 3, and 0.4.

## Task 3:

To write a Matlab program to do Contrast stretching intensity transformation for the given input image with E value as 0.4, 0.5 and -1. Comment on the output image by comparing with input image.

## Task 4:

Perform intensity level slicing on original input image with minimum intensity level as 100 and maximum intensity level as 180 and plot the corresponding output.

**SOFTWARE REQUIRED:**

MATLAB R2017a

## THEORY:

**Photographic Negative**

The Photographic Negative is probably the easiest of the intensity transformations to describe. Assume that we are working with grayscale double arrays where black is 0 and white is 1. The idea is that 0's become 1's, 1's become 0's, and any gradients in between are also reversed. In intensity, this means that the true black becomes true white and vise versa.

## Syntax:

**B =(L-1) – A;**

Where, A is input image and L is maximum intensity level of image.

## Logarithmic Transformation

Logarithmic Transformations can be used to brighten the intensities of an image. More often, it is used to increase the detailof lower intensity values.

## Syntax:

**g = c\*log(1 + double(A));**

Where,A is input image and ‘c’ is constant is usually used to scale the range of the log function to match the input domain.

## Gamma Transformations

With Gamma Transformations, you can curve the grayscale components either to brighten the intensityor to darken the intensity.

## Syntax:

**I = imadjust(f, [low\_in high\_in], [low\_out high\_out], gamma);**

Where, f is the input image, [low\_in high\_in] and [low\_out high\_out] are used for clipping. we can use [] for both [low\_in high\_in] and [low\_out high\_out]. This means that the full range of the input is mapped to the full range of the output.

## Contrast-Stretching Transformations

Contrast-stretching transformations increase the contrast between the darks and the lights. This transformation kept everything at relatively similar intensities and merely stretched the histogram to fill the image's intensity domain.

## Syntax:

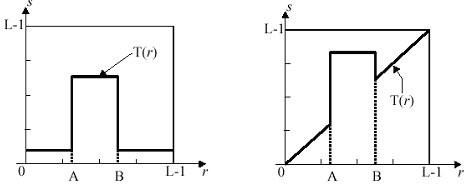
**x=1. /(1 + (m./(double(f) + eps)).^E)**

Where, E controls the slope of the function and m is the mid-line where you want to switch from dark values to light values. eps is a MATLAB constant that is the distance between 1.0 and the next largest number that can be represented in double-precision floating point.

## Intensity Level Slicing

Highlighting a specific range of gray-levels in an image is often desired. There are two main different approaches:

* Highlight a range of intensities while diminishing all others to a constant low level.
* Highlight a range of intensities but preserve all others.

Schematic representation of intensity level slicing:

We can set Maximum intensity and Minimum intensity level for transformation.

## PROGRAM:

1. **Negative transformation and log transformation.**

clc; clear all; close all;

I=imread('tire.tif'); imshow(I)

J=225-I;

subplot(3,2,1) imshow(I);title('original image') subplot(3,2,2)

imshow(I);title('negative transformation'); I2=im2double(I);

K1=1\*log(1+ I2); K2=2\*log(1+I2); K3=5\*log(1+I2);

subplot(3,2,3)

imshow(K1);title('log transformation(c=1)'); subplot(3,2,4)

imshow(K2);title('log transformation(c=2)'); subplot(3,2,5)

imshow(K3);title('log transformation(c=5)');

## Gamma Transformation

clc; clear all; close all;

I=imread('tulips.jpg'); J1=imadjust(I,[],[],1);

J2=imadjust(I,[],[],0.4);

J3=imadjust(I,[],[],0.5); subplot(2,2,1) imshow(I);title('original image'); subplot(2,2,2)

imshow(J1);title('gamma transformation(gamma=1)'); subplot(2,2,3)

imshow(J2);title('gamma transformation(gamma=0.4)'); subplot(2,2,4)

imshow(J3);title('gamma transformation(gamma=0.5)');

## Contrast stretching

clc; clear all; close all;

I=imread('tulips.jpg'); I2=im2double(I); m=mean2(I2)

contrast1=1./(1+(m./(I2+eps)).^4); contrast2=1./(1+(m./(I2+eps)).^5); contrast3=1./(1+(m./(I2+eps)).^-1); subplot(2,2,1) imshow(I);title('original image'); subplot(2,2,2)

imshow(contrast1);title('contrast stretching(E=0.4)'); subplot(2,2,3)

imshow(contrast2);title('contrast stretching(E=0.5)'); subplot(2,2,4)

imshow(contrast3);title('contrast stretching(E=-1)');

## Intensity level slicing

clc; clear all; close all;

I=imread('tulips.jpg'); image=I(:,:,1); rmin=100; rmax=180; [r,c]=size(image); s=zeros(r,c);

for i=1:r for j=1:c

if(rmin<image(i,j)&&image(i,j)<rmax) s(i,j)=0;

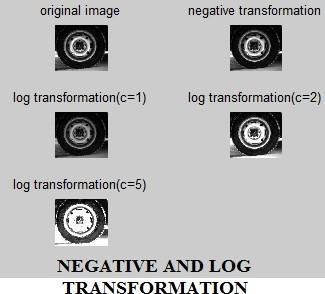
else s(i,j)=image(i,j); end

end end

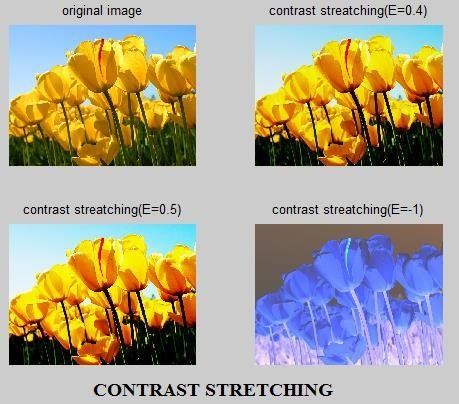
subplot(1,2,1) imshow(image);title('original image'); subplot(1,2,2)

imshow(s);title('intensity level slicing');

# OUTPUT:

**(a)**

# (b)

**(c)**

# (d)

## RESULT:

Intensity transformations with different transformation function are performed on input grayscale image using MATLAB software and output are plotted successfully.

# Ex. No.4

**AIM:**

# DFT ANALYSIS OF IMAGES

To obtain 2D DFT of an image and subsequent image restoration reconstruction using MATLAB programming.

## SOFTWARE REQUIRED:

MATLABR2017a

## THEORY:

The discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence.

The DFT is therefore said to be a frequency domain representation of the original input sequence. If the original sequence spans all the non-zero values of a function, its DTFT is continuous (and periodic), and the DFT provides discrete samples of one cycle. If the original sequence is one cycle of a periodic function, the DFT provides all the non-zero values of one DTFT cycle.

## PROGRAM:

**DFT Analysis of Images**

clc; clear all; close all;

%a=zeros(256,256);

%a(50:150,50:150)=1;

%subplot(3,3,1);

%imshow(a); a=imread('pout.tif'); subplot(3,3,1); imshow(a); title('given image');

%fft

y=fft2(a); subplot(3,3,2); imshow(y);

title('fft of the given image');

%shifting of origin z=fftshift(y); subplot(3,3,3); imshow(z);

title('fft with the shifted origin');

%reshifting of the origin m=ifftshift(z); subplot(3,3,4); imshow(m);

title('fft reshifted to the original position');

%inverse fft n=ifft2(m); n1=uint8(n); subplot(3,3,5); imshow(n1);

title('ifft of the given image');

%discrete cosine transformation b=dct(a);

subplot(3,3,6); imshow(b);

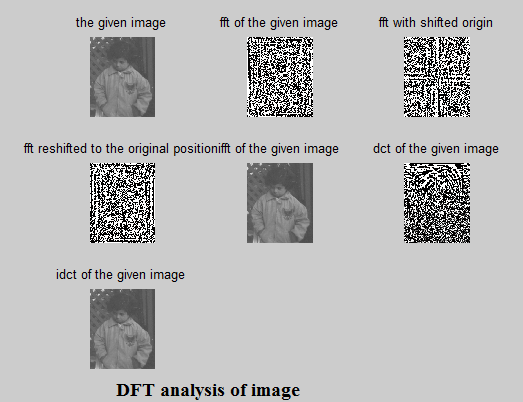
title('dct of the given image'); c=idct(b);

c1=uint8(c);

%z1=uint8(z); subplot(3,3,7); imshow(c1);

title('idct of the given image');

# OUTPUT:



## RESULT:

Thus we have obtained an image using 2D DFT and subsequent image restoration / reconstruction using MATLAB programming.

# Ex. No.5

## AIM:

**TRANSFORMS (WALSH, HADAMARD, DCT, HAAR)**

To perform different types of transforms of an image and reconstruct it using MATLAB programming.

## SOFTWARE REQUIRED:

MATLABR2017a

## THEORY:

**Discrete Cosine Transform:**

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations.The use of cosine rather than sine functions is critical for compression, since it turns out, that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions.

## PROGRAM:

clc; clear all; close all;

d=imread('cameraman.tif'); e=im2double(d); e=imresize(e,[256 256]); [M N]=size(e);

[u x]=meshgrid(0:N-1); ker1=sqrt(2/N)\*cos(pi\*(2.\*x+1).\*u/(2\*N)); ker1(1,:)=ker1(1,:)/sqrt(2); trans2=ker1\*e\*ker1;

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

xlabel('fig 5.4 original image'); subplot(2,2,2); imshow(trans2);

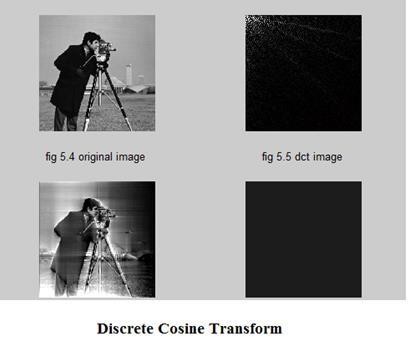
xlabel('fig 5.5 dct image'); orig2=ker1\*trans2\*ker1; subplot(2,2,3); imshow(orig2);

xlabel('fig 5.6 dct reconstructed image'); error2=mse(e-orig2);

subplot(2,2,4); imshow(error2);

xlabel('fig 5.8 dct error image');

# OUTPUT:



## RESULT:

Thus we have obtained an image using DCT and restored using MATLAB programming.

# Ex. No. 6

## AIM:

**HISTOGRAM PROCESSING AND BASICTHRESHOLDING FUNCTIONS**

**Task 1:**To perform histogram processing for the given input image using MATLAB software and to apply histogram equalization on given input image. The tasks to perform are,

* 1. To plot the histogram for a given input image.
  2. To plot the histogram of its corresponding equalized image.

**Task 2:** To perform the basic thresholding using MATLAB.

## SOFTWARE REQUIRED:

MATLAB R2017a

## THEORY:

**Histogram**is defined as a graph plotted between gray level value and no of pixels. Histograms are the basis for numerous spatial domain processing technique. Histogram manipulation can be used effectively for image enhancement. Histograms are simple to calculate in software and also lend themselves to economic hardware implementations thus making them a popular tool for real time image processing.

**Thresholding** is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images.

The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity I(i,j) is less than some fixed constant T (that is,I(i,j)<T), or a white pixel if the image intensity is greater than that constant. In the example image on the right, this results in the dark tree becoming completely black, and the white snow becoming completely white.

## COMMANDS USED:

**IMHIST:**

Displays the histogram of image data.

## SYNTAX:

**I = IMHIST (FILENAME)** displays a histogram for the image of given filename above a grayscale color bar. If the given filename is a binary image, then imhist uses two pins. If given filename is a grayscale image, then imhist uses a default values of 256 pins.

## HISTEQ:

To enhance contrast using histogram equalization

## SYNTAX:

**J=HISTEQ (FILENAME,HGRAM)** transforms the intensity image of the given filename. So that the output intensity image J with length (hgram) bins approximately matches histogram.

## PROGRAM:

**Histogram:**

clc; clear all; close all;

A=imread('Penguins.jpg'); I=rgb2gray(A); subplot(2,2,1); imshow(I);

title('original image'); subplot(2,2,2); imhist(I);

title('before equalization'); subplot(2,2,3);

histeq(I); title('equalized image'); subplot(2,2,4);

plot(I);

title('after equalization');

# OUTPUT:

## PROGRAM:

**Thresholding:**

I=imread('coins.png'); imshow(I); level=multithresh(I,1); seg\_I=imquantize(I,level); figure

imshow(seg\_I,[])

## OUTPUT:

**Original Image Image after thresholding**

## RESULT:

Histograms are processed for the given input image using MATLAB software and Histogram equalization and the basic thresholding outputs are plotted using MATLAB software successfully.

# Ex. No. 7

**IMAGE ENHANCEMENT-SPATIAL FILTERING**

## AIM:

To do enhancement process in spatial domain for an input image using MATLAB

software.

## IMAGE SMOOTHING AND SHARPENING PROCESS:

**Task 1:**

1. Compute blurring effect on an image using motion and disk command.
2. Enhance input image by sharpening technique.

## Task 2:

1. Apply Box Filter (3\*3) and (5\*5) and do smoothing process of input image.
2. Remove the noise by applying Average filter and Median Filter (Add Salt and Pepper Noise).

**SOFTWARE REQUIRED:**

MATLAB R2017a

## THEORY:

**Spatial Domain Methods:**

* + Spatial domain refers to the image plane.
  + Approaches in this category are based on direct manipulation of pixels in an image.
  + The process of spatial filtering consists simply of moving the filter mask from point to point in an image.
  + At each point (*x, y*), the *response* of the filter at that point is the sum of the products of values of each entry of the mask with its corresponding image pixel value.Few syntax for the filters used in spatial filtering process during enhancement of input image are given below:

## SYNTAX:

**h = fspecial ('motion', len, theta)**

It returns a filter to approximate, once convolved with an image, the linear motion of a camera by len pixels, with an angle of theta degrees in a counterclockwise direction. The filter becomes a vector for horizontal and vertical motions. The default len is 9 and the default theta is 0, which corresponds to a horizontal motion of nine pixels.

## h = fspecial('disk',radius)

It returns a circular averaging filter (pillbox) within the square matrix of side 2\*radius+1.

The default radius is 5.

## h = fspecial('unsharp',alpha)

It returns a 3-by-3 unsharp contrast enhancement filter. fspecial creates the unsharp filter from the negative of the Laplacian filter with parameter alpha. Alpha controls the shape of the Laplacian and must be in the range 0.0 to 1.0. The default value for alpha is 0.2.

## J = imnoise(I,'salt & pepper',d)

The above command adds salt and pepper noise to the the intensity image I, where d is the noise density. The default for d is 0.05.

## h = fspecial('average',hsize)

It returns an averaging filter h of size hsize. The argument hsize can be a vector specifying the number of rows and columns in h, or it can be a scalar, in which case h is a square matrix. The default value for hsize is [3 3].

## Y = filter2(h,J)

Filters the data in X with the two-dimensional FIR filter in the matrix h. It computes the result, Y, using two-dimensional correlation, and returns the central part of the correlation that is the same size as J

## B = medfilt2(A)

The above operator performs median filtering of the matrix A using the default 3-by-3 neighborhood.

## PROGRAM:

1. **Sharpening Technique:** I=imread('cameraman.tif'); subplot(2,2,1);

imshow(I); title('orginal image');

H=fspecial('motion',20,45); MotionBlur=imfilter(I,H,'replicate');

subplot(2,2,2); imshow(MotionBlur); title('motion blurred image'); H=fspecial('disk',10); blurred=imfilter(I,H,'replicate'); subplot(2,2,3); imshow(blurred);

title('blurred image'); H=fspecial('unsharp'); sharpened=imfilter(I,H,'replicate'); subplot(2,2,4); imshow(sharpened); title('sharpened image');

## Smoothing Technique:

clc; clear all; close all;

a=imread('Coins.png'); h1=1/9\*ones(3,3); h2=1/25\*ones(5,5); b1=conv2(a,h1,'same');

b2=conv2(a,h2,'same');

subplot(2,2,1); imshow(a); title('original image'); subplot(2,2,2); imshow(uint8(b1));

title('output using 3\*3 mask'); subplot(2,2,3); imshow(uint8(b2)); title('output using 5\*5 mask');

## Average and Median Filtering:

clc; clear all;

A=imread('Tulips.jpg'); figure;

I=rgb2gray(A);

J=imnoise(I,'salt & pepper',0.02); K=filter2(fspecial('average',3),J)/255; M=medfilt2(J);

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

title('salt and pepper noise'); subplot(2,2,3);

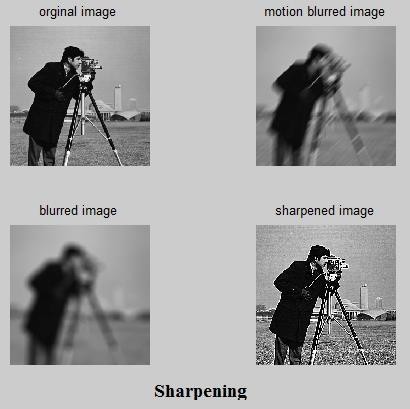
imshow(K);

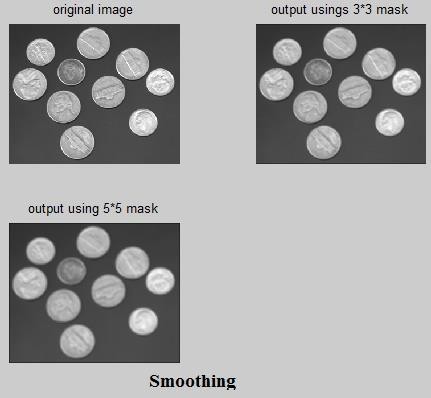
title('filtering by using average filter'); subplot(2,2,4);

imshow(M);

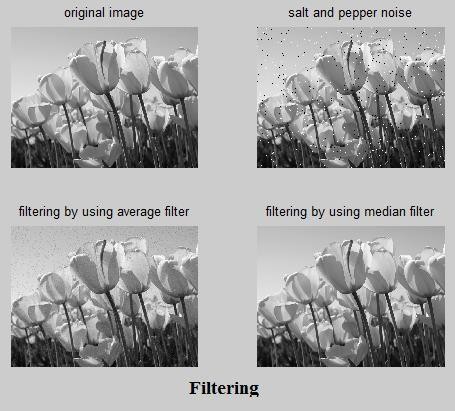
title('filtering by using median filter');

# OUTPUT:



**(a)**

# (b)



## (C)

**RESULT:**

Image Enhancement process is carried out in spatial domain for an input image using various spatial filtering techniques and corresponding outputs are plotted and analyzed.

# Ex. No. 8

**AIM****:**

# IMAGE ENHANCEMENT - FILTERING IN FREQUENCY DOMAIN

To filter an image in frequency domain using MATLAB programming.

## SOFTWARE REQUIRED:

MATLAB R2017a

## THEORY:

The concept of filtering is easier to visualize in the frequency domain. Therefore, enhancement of image f (m,n) can be done in the frequency domain, based on its DFT F(u,v).

The enhancement in the frequency domain as follows:

## G(u,v) = H(u,v)F(u,v)

Where, G(u,v)->Enhanced image, F(u,v)->Given image & H(u,v)->Transfer function

## PROGRAM:

clc; clear all;

x1=imread('koala.jpg'); x=rgb2gray(x1); subplot(2,2,1); imshow(x);

title('the given image');

%butterworth lpf fc=30; fc2=fc\*fc;

r=1;

[m,n]=size(x); f=fft2(x); subplot(2,2,2);

imshow(f);

title('fft of the given image'); fs=fftshift(f); mc=round(m/2); nc=round(n/2); b=zeros(m,n);

for i=1:m for j=1:n

d=(i-mc).^2+(j-nc).^2; b(i,j)=1/(1+((d/fc2).^(2\*r))); end

end h=fs.\*b;

%y=ifftshift(h); y1=abs(ifft2(h)); y2=uint8(y1); subplot(2,2,3); imshow(y2);

title('smoothened image using buttorworth LPF');

%buttorworth HPF fc=1.2;

fc2=fc\*fc; for i=1:m for j=1:n

d=(i-mc).^2+(j-nc).^2; b(i,j)=1/(1+((fc2/d).^(2\*r))); end

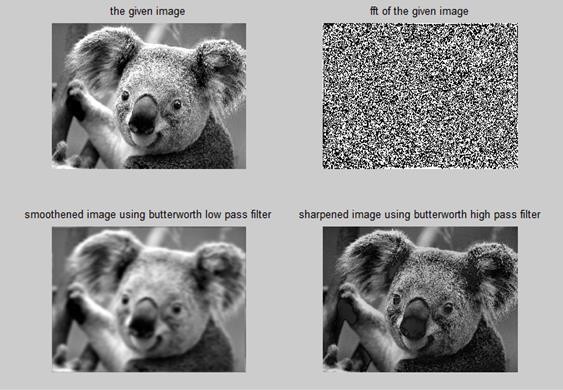
end h=fs.^b;

y1=abs(ifft2(h)); y2=uint8(y1); subplot(2,2,4);

imshow(y2);

title('sharpened image using buttorworth HPF');

## OUTPUT:



**RESULT:**

Thus, the image filtering in frequency domain was performed using MATLAB.

# Ex. No. 9

**IMAGE SEGMENTATION- EDGE DETECTION, LINE DETECTION AND POINT DETECTION**

## AIM:

To perform image segmentation using MATLAB program.

## SOFTWARE REQUIRED:

MATLABR2017a

## THEORY:

**Image segmentation:**

Segmentation subdivides an image into its constituent regions or objects that have similar Features (Intensity, Histogram, mean, variance, Energy, Texture, ...etc.) according to a set of predefined criteria. Most segmentations algorithms we will consider are based on one of two basic properties of intensity values:

***Discontinuity:***

The strategy is to partition an image based on abrupt changes in intensity detection of gray level discontinuities:

\*Edge detection, \*Line detection and \*Point detection

## Edge detection:

**Edge detection** is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges.Edge detection is the most common approach for detecting meaningful discontinuities.

Edge is detected by:

\*First‐order derivative (gradient operator)

\* Second‐order derivative (laplacian operator)

Two general criteria for edge detection:

\* Find places where the first derivative of the intensity is greater magnitude than a Specified threshold

\*Find places where the second derivative of the intensity has a zero crossing.

## Edge function:

Edge function find edges in intensity image.

## Syntax:

[g, t] = edge(f, ‘method’, parameters)

Edge supports six different edge-finding methods:

* 1. The Sobel, 2. The Prewitt, 3. The Robert, 4.The Laplacian, 5.The zero-cross, 6.The canny

## PROGRAM:

clear all; clc;

x=imread('mri.tif'); subplot(3,3,1); imshow(x);

title('given image for edge detection'); b=edge(x,'prewitt',0.05); subplot(3,3,2);

imshow(b);

title('edge detection using prewitt edge detector'); c=edge(x,'sobel',0.05);

subplot(3,3,3); imshow(c);

title('edge detection using sobel edge detector'); d=edge(x,'canny',0.05);

subplot(3,3,4); imshow(d);

title('edge detection using canny edge detector'); e=edge(x,'roberts',0.07);

subplot(3,3,5); imshow(e);

title('edge detection using roberts edge detectors'); f=edge(x,'log',0);

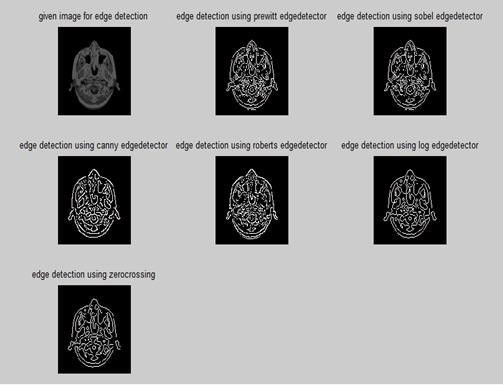
subplot(3,3,6); imshow(f);

title('edge detection using log edge detector'); g=edge(x,'zerocross',0);

subplot(3,3,7); imshow(g);

title('edge detection using zercrossing');

## OUTPUT:



**Point Detection:**

The detection of isolate points embedded in constant or nearly constant areas is detected. T is a nonnegative threshold. Point detection is implemented in MATLAB using Function imfilter. If T is given, the following command implements the point detection.

## Syntax:

G = abs(imfilter(double(f),w)) >= T;

Where f is the input image, w is an appropriate point-detection mask, and g is the Resulting image.

## PROGRAM:

clc; clear all; close all;

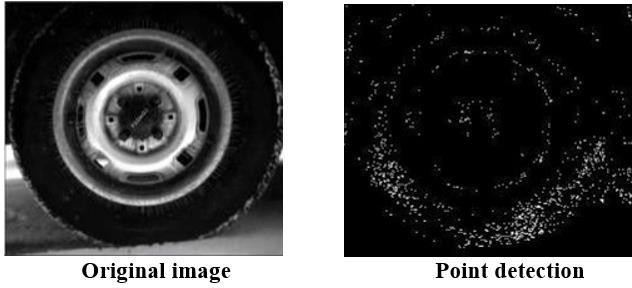
f = imread('tire.tif');

w=[-1,-1,-1;-1,8,-1;-1,-1,-1];

g=abs(imfilter(double(f),w)); T=max(g(:));

g=g>=T; %(T given threshold) subplot(121),imshow(f),title('original image'); subplot(122),imshow(g),title('Result of point detection');

# OUTPUT:



## Line Detection:

These filter masks would respond more strongly to lines. Note that the coefficients in eachask sum to zero, indicating a zero response from the masks in areas of constant gray level. Ifwe are interested in detecting lines in a specified direction (e.g. vertical), we coulduse the mask associated with that direction and threshold its output.If interested in lines of any directions run all 4 masks and select the highest response.These filters respond strongly to lines of one pixel thick of the designated directionand correspond closest to the direction defined by the mask.

## PROGRAM:

clc; clear all; close all;

f = imread('cell.tif');

w=[2,-1,-1;-1,2,-1;-1,-1,2]; %%-45

g1=imfilter(double(f),w); gtop=g1(1:120,1:120);

gtop=pixeldup(gtop,4); %%Duplicates pixels of an image in both directions. gbot=g1(end-119:end,end-119:end);

gbot=pixeldup(gbot,4); g=abs(g1); T=max(g(:));

g=g>=T;

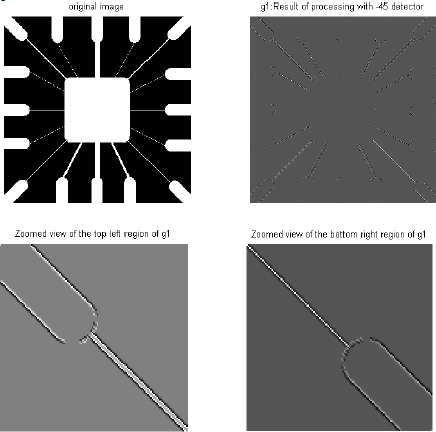
subplot(121),imshow(f),title('original image');

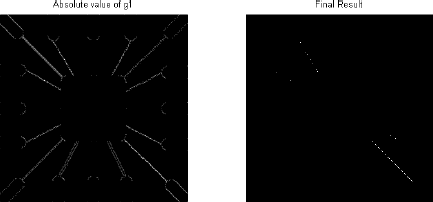
subplot(122),imshow(g1,[]),title('g1:Result of processing with -45 detector'); figure

subplot(121),imshow(gtop,[]),title('Zoomed view of the top left region of g1'); subplot(122),imshow(gbot,[]),title('Zoomed view of the bottom right region of g1'); figure

subplot(121),imshow(g,[]),title('Absolute value of g1'); subplot(122),imshow(gg),title('Final Result');

# OUTPUT:





## RESULT:

Thus, the image discontinuities were highlighted using various edge detection operations.

# Ex. No. 10

**BASIC MORPHOLOGICAL OPERATION**

## AIM:

To perform basic morphological operations using MATLAB program.

## SOFTWARE REQUIRED:

MATLABR2017a

## THEORY:

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. Morphological operations rely only on the relative ordering of pixel values, not on their numerical values, and therefore are especially suited to the processing of binary images. Morphological operations can also be applied to grayscale images such that their light transfer functions are unknown and therefore their absolute pixel values are of no or minor interest. Morphological techniques probe an image with a small shape or template called a structuring element. The structuring element is positioned at all possible locations in the image and it is compared with the corresponding neighborhood of pixels.

## TYPES OF MORPHOLOGICAL OPERATION:

**Dilation:** Dilation adds pixels on the object boundaries.

**Erosion**: Erosion removes pixels on Object boundaries

**Open:** The opening operation on erodes an image and then dilates the eroded image using and then dilates the eroded image using the same structuring element for both operation.

**Close:** The closing operation dilates an image and the eroded the dilated image using the same structuring element for both operation.

The number of pixels added or removed from object in an image depends on the shape and size of the structuring element used to processing the image. In the morphological dilation and erosion operation, the state of any given pixel in the out put is determined by applying a rule to the corresponding pixel and it’s neighbors in the input image. The rule used to process the pixel defines the morphological operation as a dilation or an erosion.

## DILATED IMAGE:

**Syntax:**

J= imdilate(I, SE)

J= imdilate(I, SE) dilated the grayscale binary, or packed binary image I, returning the dilated image, J, SE is a structuring element object, returned by the stream or offsetstrel functions.

## EROSION IMAGE:

**Syntax:**

J= imerode(I, SE)

J= imerode (I, SE) erodes the grayscale, binary, or packed binary image I, returning the eroded image, J, SE is a structuring element object or array of structuring element objects, returned by the stream or offsetstrel functions.

## OPEN IMAGE:

**Syntax:**

J= imopen(I, SE)

J= imopen(I, SE) performs morphological opening on the grayscale or binary image I, returning the opened image, J, SE is a single structuring element object returned by the strel or offsetstrel function. The morphological open operation is an erosion followed by a dilation, using the same structuring element for both operation.

## CLOSED IMAGE:

**Syntax:**

J= imclose(I, SE)

J= imclose(I, SE) performs morphological closing on the grayscale or binary image, In returning the closed image, J, SE is a single structuring element object returned by the strel or offsetstrel function. The morphological close operation is a dilated followed by an erosion using the same structuring element for both operation.

## PROGRAM:

I = imread('football.jpg'); subplot(2, 3, 1); imshow(I);

title ('Original image');

% Dilated Image

se = strel('line', 7, 7); dilate = imdilate(I, se); subplot (2, 3, 2); imshow(dilate);

title ('Dilated image');

% Eroded image

erode = imerode(I, se); subplot(2, 3, 3);

imshow(erode); title('Eroded image');

% Opened image open = imopen(I, se); subplot(2, 3, 4); imshow(open); title('Opened image');

% Closed image

close = imclose(I, se); subplot(2, 3, 5); imshow(close); title('Closed image');

## OUTPUT:

**RESULT:**

Thus, the basic morphological operations are performed using MATLAB programming.

## Ex. No. 11

**AIM:**

# REGION BASED SEGMENTATION

To write a MATLAB program for region-based segmentation.

## SOFTWARE REQUIRED:

MATLAB R2020b

## THEORY:

Region based segmentation is classified into three types

1. Region growing2. Region splitting 3. Region merging

Region based segmentation is a technique for determining the region directly. Region growing is a Simple region-based Image segmentation method. It also classified into pixel- based image segmentation method since it involves the selection of initial seed points

## Region growing

Region growing is a procedure that groups of pixels or sub regions into larger regions The simplest of these approaches is Pixel aggregation, which starts with a set of seed Points and from these grows region by appending to each seed points those neighboring Pixels that have similar properties(such as gray level, texture, color, shape)

Region growing based technique are better than the edge based technique in noisy images Where edge are difficult to detect

## Region splitting

Region growing starts from a set of seed points

An alternative is to start With the whole image as a single Region and subdivide the regions That do not satisfied a condition of homogeneity

## Region merging

Region merging is the opposite of region splitting

Start with small regions (example. 2\*2 or 4\*4 regions) and merge the region that have similar characteristic (such as gray level, variance)

Typically, splitting and merging approaches are used iteratively

## Program (region growing):

function J=region growing(I,x,y,reg\_maxdist)

%This function perform" region growing"in an image from a specified

% seed point(x,y)

%

% J = regiongrowing(I,x,y,t)

%

% I : input image

% J : logical output image of region

% x,y : the position of the seedpoint (if not given uses function getpts)

% t : maximum intensity distance (defaults to 0.2)

%

% The region is iteratively grown by comparing all unallocated neighbouring pixels to the region.

% The difference between a pixel's intensity value and the region's mean,

% is used as a measure of similarity. The pixel with the smallest difference

% measured this way is allocated to the respective region.

% This process stops when the intensity difference between region mean and

% new pixel become larger than a certain treshold (t)

%

% Example:

%

% I = im2double(imread('medtest.png'));

% x=198; y=359;

% J = regiongrowing(I,x,y,0.2);

% figure, imshow(I+J);

%

if(exist('reg\_maxdist','var')==0), reg\_maxdist=0.2; end

if(exist('y','var')==0), figure, imshow(I,[]); [y,x]=getpts; y=round(y(1)); x=round(x(1)); end J = zeros(size(I)); % Output

Isizes = size(I); % Dimensions of input image

reg\_mean = I(x,y); % The mean of the segmented region reg\_size = 1; % Number of pixels in region

% Free memory to store neighbours of the (segmented) region neg\_free = 10000; neg\_pos=0;

neg\_list = zeros(neg\_free,3);

pixdist=0; % Distance of the region newest pixel to the regio mean

% Neighbor locations (footprint) neigb=[-1 0; 1 0; 0 -1;0 1];

% Start regiogrowing until distance between regio and posible new pixels become

% higher than a certain treshold while(pixdist<reg\_maxdist&&reg\_size<numel(I))

% Add new neighbors pixels for j=1:4,

% Calculate the neighbour coordinate xn = x +neigb(j,1); yn = y +neigb(j,2);

% Check if neighbour is inside or outside the image ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));

% Add neighbor if inside and not already part of the segmented area if(ins&&(J(xn,yn)==0))

neg\_pos = neg\_pos+1;

neg\_list(neg\_pos,:) = [xn yn I(xn,yn)]; J(xn,yn)=1;

end end

% Add a new block of free memory

if(neg\_pos+10>neg\_free), neg\_free=neg\_free+10000; neg\_list((neg\_pos+1):neg\_free,:)=0; end

% Add pixel with intensity nearest to the mean of the region, to the region dist = abs(neg\_list(1:neg\_pos,3)-reg\_mean);

[pixdist, index] = min(dist); J(x,y)=2; reg\_size=reg\_size+1;

% Calculate the new mean of the region

reg\_mean= (reg\_mean\*reg\_size + neg\_list(index,3))/(reg\_size+1);

% Save the x and y coordinates of the pixel (for the neighbour add proccess) x = neg\_list(index,1); y = neg\_list(index,2);

% Remove the pixel from the neighbour (check) list neg\_list(index,:)=neg\_list(neg\_pos,:); neg\_pos=neg\_pos-1;

end

% Return the segmented area as logical matrix J=J>1;

## Program:

**Region Based Splitting and Merging:**

function g = splitmerge(f, mindim, fun)

%SPLITMERGE Segment an image using a split-and-merge algorithm.

% G = SPLITMERGE(F, MINDIM, @PREDICATE) segments image F by using a

% split-and-merge approach based on quadtree decomposition. MINDIM

% (a positive integer power of 2) specifies the minimum dimension

% of the quadtree regions (subimages) allowed. If necessary, the

% program pads the input image with zeros to the nearest square

% size that is an integer power of 2. This guarantees that the

% algorithm used in the quadtree decomposition will be able to

% split the image down to blocks of size 1-by-1. The result is

% cropped back to the original size of the input image. In the

% output, G, each connected region is labeled with a different

% integer.

%

% Note that in the function call we use @PREDICATE for the value of

% fun. PREDICATE is a function in the MATLAB path, provided by the

% user. Its syntax is

%

% FLAG = PREDICATE(REGION) which must return TRUE if the pixels

% in REGION satisfy the predicate defined by the code in the

% function; otherwise, the value of FLAG must be FALSE.

%

% The following simple example of function PREDICATE is used in

% Example 10.9 of the book. It sets FLAG to TRUE if the

% intensities of the pixels in REGION have a standard deviation

% that exceeds 10, and their mean intensity is between 0 and 125.

% Otherwise FLAG is set to false.

%

% function flag = predicate(region)

% sd = std2(region);

% m = mean2(region);

% flag = (sd > 10) & (m > 0) & (m < 125);

% Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins

% Digital Image Processing Using MATLAB, Prentice-Hall, 2004

% $Revision: 1.6 $ $Date: 2003/10/26 22:36:01 $

% Pad image with zeros to guarantee that function qtdecomp will

% split regions down to size 1-by-1. Q = 2^nextpow2(max(size(f)));

[M, N] = size(f);

f = padarray(f, [Q - M, Q - N], 'post');

%Perform splitting first.

S = qtdecomp(f, @split\_test, mindim, fun);

% Now merge by looking at each quadregion and setting all its

% elements to 1 if the block satisfies the predicate.

% Get the size of the largest block. Use full because S is sparse. Lmax = full(max(S(:)));

% Set the output image initially to all zeros. The MARKER array is

% used later to establish connectivity. g = zeros(size(f));

MARKER = zeros(size(f));

% Begin the merging stage. for K = 1:Lmax

[vals, r, c] = qtgetblk(f, S, K); if ~isempty(vals)

% Check the predicate for each of the regions

% of size K-by-K with coordinates given by vectors

% r and c.

for I = 1:length(r)

xlow = r(I); ylow = c(I);

xhigh = xlow + K - 1; yhigh = ylow + K - 1; region = f(xlow:xhigh, ylow:yhigh);

flag = feval(fun, region); if flag

g(xlow:xhigh, ylow:yhigh) = 1; MARKER(xlow, ylow) = 1;

end end

end end

% Finally, obtain each connected region and label it with a

% different integer value using function bwlabel. g = bwlabel(imreconstruct(MARKER, g));

% Crop and exit g = g(1:M, 1:N);

% %

function v = split\_test(B, mindim, fun)

% THIS FUNCTION IS PART OF FUNCTION SPLIT-MERGE. IT DETERMINES

% WHETHER QUADREGIONS ARE SPLIT. The function returns in v

% logical 1s (TRUE) for the blocks that should be split and

% logical 0s (FALSE) for those that should not.

% Quadregion B, passed by qtdecomp, is the current decomposition of

% the image into k blocks of size m-by-m.

% k is the number of regions in B at this point in the procedure. k = size(B, 3);

% Perform the split test on each block. If the predicate function

% (fun) returns TRUE, the region is split, so we set the appropriate

% element of v to TRUE. Else, the appropriate element of v is set to

% FALSE.

v(1:k) = false; for I = 1:k

quadregion = B(:, :, I);

if size(quadregion, 1) <= mindim v(I) = false;

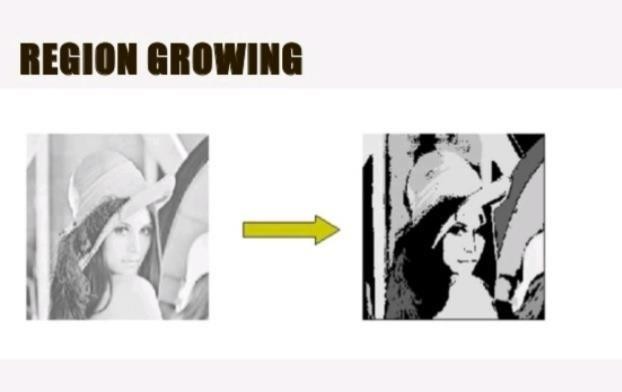
continue end

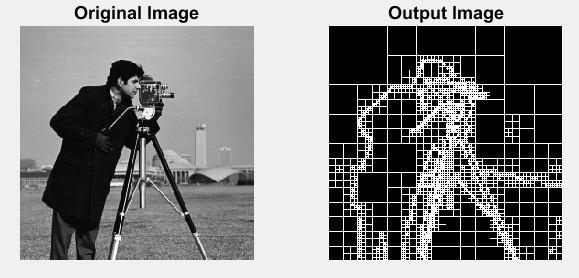
flag = feval(fun, quadregion); if flag

v(I) = true; end

end

## OUTPUT:





**REGION SPLITTING AND MERGING**

## RESULT:

Thus, the region-based segmentation operations are performed using MATLAB programming.

## Ex. No. 12

**SEGMENTATION USING WATERSHED TRANSFORMATION**

## AIM:

To write a Matlab program of segmentation using watershed transformation.

## SOFTWARE REQUIRED:

MATLAB R2012b

## THEORY:

Watershed Segmentation gets its name from the manner in which the algorithm segment region into Catchment basins. Instead of working on a image itself, this technique is often applied on its gradient image. The key behind using the watershed transform for segmentation is the change your image into another image whose catchment basins are the object you want to identify. Use internal markers to obtain watershed lines of the gradient of the image to be segmented.

## PROGRAM:

clc; clear all; close all;

I=imread('coins.png');

%I=rgb2gray(RGB); I1=imtophat(I, strel('disk',10));

%A very useful morphological transformaation to subtract the background from

%an image is the so called tophat. Tophat is the subtraction of an opened

%image from the original. One can do opening in gray images,removing all

%features smaller than the structuring element figure,imshow(I1);

I2=imadjust(I1); figure,imshow(I2); level=graythresh(I2); BW=im2bw(I2,level); figure,imshow(BW); C=~BW;

figure,imshow(C); D=-bwdist(C);

%bwdist compute the distance transform. The distance transform of a binary

%image is the distance from every pixel to the nearest nonzero-valued pixel D(C)=-Inf;

%modify the image so that the background pixels and the extended maxima

%pixels are forced to be the only local minima in the image. L=watershed(D);

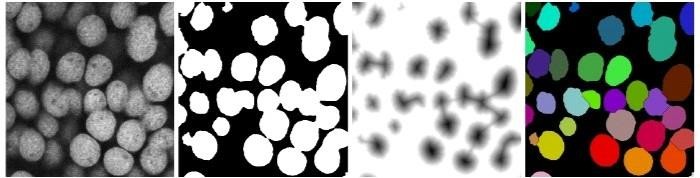
figure,imshow(L);

wi=label2rgb(L,'hot','w'); figure,imshow(wi); im=I;

im(L==0)=0;

figure,imshow(im);

## OUTPUT:



**RESULT:**

Segmentation using watershed transformation techniques were applied in different input image and output are plotted using MATLAB software successfully.

## Ex. No.13

**ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS**

## AIM:

**Task 1: Analysis of color image conversion models**

To write a computer program to read an RGB image and to convert RGB image into Grayscale image, Matrix to gray scale image, Gray scale to binary image.

## Task 2: Extraction and analysis of color components

To write a computer program to read an RGB image. Extract and separate the three colour components red, green and blue from the input image.

## Task 3: Analysis of HSI Model

To write a computer program to read an RGB image and display Hue, Intensity and Saturation level in the input image.

## SOFTWARE REQUIRED:

MATLAB R2017a

## THEORY:

Reading an image enable us to view that image through MATLAB. The command used to read is imread. This command reads the image and stores it in variable.

|  |  |  |
| --- | --- | --- |
| **FORMAT NAME** | **DESCRIPTION** | **RECOGNISED**  **EXTENSIONS** |
| TIFF JPEG  GIF BMP PNG  XWD | Tagged Image File Format Joint Photographic Experts Group  Graphic Interchange Format Windows Bitmap  Possible Network Graphics XWindows | .tif .tiff  .jpg .jpeg  .gif  .bmp  .png  .xwd |

Images are displayed on the MATLAB desktop. The command used for this purpose is imshow.

## BINARY IMAGES

In a binary image, each pixel assumes one of only two discrete values. Essentially, these two values correspond to on and off. A binary image is stored as a logical array of 0’s (off pixels) and 1’s (on pixels).

## RGB (TRUECOLOR) IMAGES

An RGB image, sometimes referred to as a “truecolor“image, is stored in MATLAB as an m- by-n-by-3 data array that defines red, green and blue color components for each individual pixel. The color of each pixel is determined by the combination of the red, green, blue intensities stored in each color plane at the pixel location. RGB MATLAB array can be of class double, uint8, or uint16. In an RGB array of class double, each color component is a value between 0 and 1. A pixel whose color components are (0,0,0) is displayed as black, and a pixel whose color components are (1,1,1) is displayed as white.

## COMMANDS USED:

1. **SYNTAX: IMAGE READ**

**A=imread (file name, format);** read a gray scale or color image from the file specified by the string FILE NAME, where the string FMT specifies the format of the file.

## SYNTAX: RGB TO GRAY CONVERSION

**B=rgb2gray(A);** converts the truecolor image Ato the grayscale intensity image B.

## SYNTAX: MATRIX TO INTENSITY IMAGE

**I = mat2gray(A, [amin amax]);** converts the matrix A to the intensity image I. The returned matrix I contains values in the range 0.0 (black) to 1.0 (full intensity or white). amin and amax are the values in A that correspond to 0.0 and 1.0 in I. Values less than amin become 0.0, and values greater than amax become 1.0.

## SYNTAX:GRAY SCALE TO BINARY IMAGE

**BW = im2bw(I, level)**; converts the grayscale image I to a binary image. The output image BW replaces all pixels in the input image with luminance greater than level with the value 1 (white) and replaces all other pixels with the value 0 (black).Specify level in the range [0,1].

## SYNTAX: COLOR MODEL ANALYSIS

**z = rgb2ntsc(rgbmap)**; converts the m-by-3 RGB values in rgbmap to NTSC color space. z is an m-by-3 matrix that contains the NTSC luminance (*Y*) and chrominance (*I* and *Q*) color components as columns that are equivalent to the colors in the RGB colormap.

## PROGRAM:

1. **Analysis of color image conversion model**

clc; clear all; close all;

a=imread('Jellyfish.jpg'); b=rgb2gray(a); c=mat2gray(b);

d=im2bw(b); subplot(2,2,1); imshow(a); title('original image'); subplot(2,2,2); imshow(b); title('grayscale image'); subplot(2,2,3); imshow(c);

title('matrix converted image'); subplot(2,2,4);

imshow(d); title('binary image');

## Extraction and analysis of color components.

clc; clear all; close all;

RGB=imread('tulips.jpg'); R=RGB;

G=RGB; B=RGB; R(:,:,2)=0;

R(:,:,3)=0;

G(:,:,1)=0;

G(:,:,3)=0;

B(:,:,1)=0;

B(:,:,2)=0;

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

title('red component'); subplot(2,2,3); imshow(G);

title('green component'); subplot(2,2,4); imshow(B);

title('blue component'); whos RGB;

whos R;whos G;whos B;

## Separation of colors from RGB plane

clc; clear all; close all;

a=imread('tulips.jpg'); a1=a;

b1=a; c1=a; a1(:,:,1)=0;

b1(:,:,2)=0;

c1(:,:,3)=0;

subplot(2,2,1); imshow(a); title('original image'); subplot(2,2,2); imshow(a1);

title('red missing'); subplot(2,2,3); imshow(b1); title('green missing'); subplot(2,2,4); imshow(c1); title('blue missing'); whos a;

whos a1; whos b1; whos c1;

## Analysis of HSI model

clc; clear all; close all;

I=imread('tulips.jpg'); Y=rgb2ntsc(I); J=Y(:,:,1);

K=Y(:,:,2);

L=Y(:,:,3);

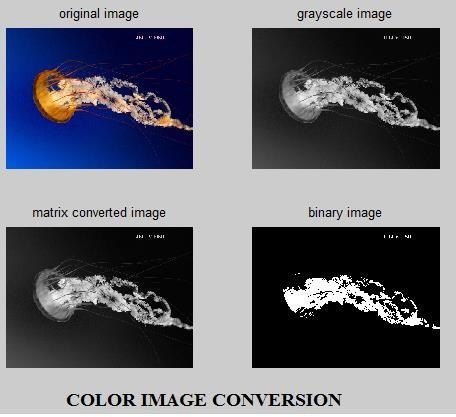
M=ntsc2rgb(Y); subplot(3,3,1); imshow(I); title('orginal image'); subplot(3,3,2); imshow(Y); title('RGB to NTSC');

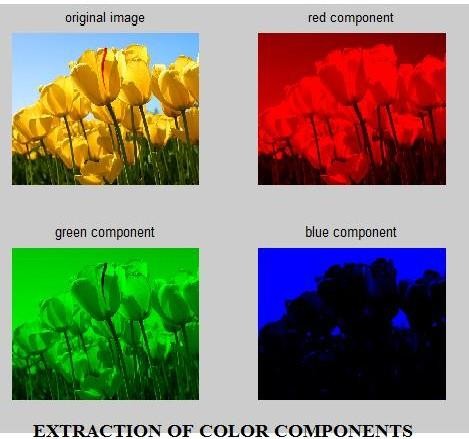
subplot(3,3,3); imshow(J); title('luminance'); subplot(3,3,4); imshow(K); title('Hue');

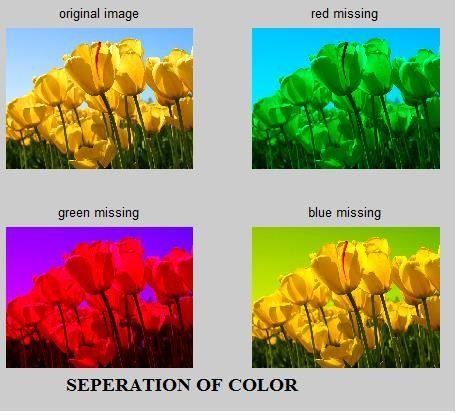
subplot(3,3,5); imshow(L); title('saturation'); subplot(3,3,6); imshow(M); title('NTSC to RGB');

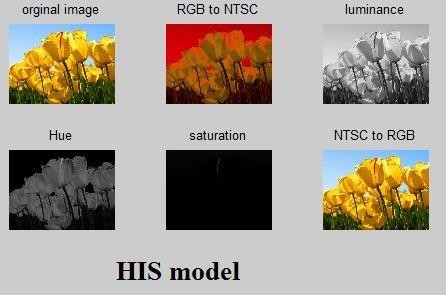
gtext('conversion of color space models');

## OUTPUT:









**RESULT:**

Analysis of color image with different color models are performed using MATLAB software successfully.

## Ex.No.14

**STUDY OF DICOM STANDARDS**

## AIM:

To study about dicom standards and different file formats.

## THEORY:

Digital Imaging and Communications in Medicine (DICOM) is the standard for the communication and management of medical imaging information and related data. The DICOM Standard facilitates interoperability of medical imaging equipment by specifying:

* For network communications, a set of protocols to be followed by devices claiming conformance to the Standard.
* The syntax and semantics of Commands and associated information that can be exchanged using these protocols.
* For media communication, a set of media storage services to be followed by devices claiming conformance to the Standard, as well as a File Format and a medical directory structure to facilitate access to the images and related information stored on interchange media.
* Information that must be supplied with an implementation for which conformance to the Standard is claimed.

The DICOM Standard does not specify:

* The implementation details of any features of the Standard on a device claiming conformance.
* The overall set of features and functions to be expected from a system implemented by integrating a group of devices each claiming DICOM conformance.

•A testing/validation procedure to assess an implementation's conformance to the Standard.

The DICOM Standard pertains to the field of Medical Informatics. Within that field, it addresses the exchange of digital information between medical imaging equipment and other systems. Because such equipment may interoperate with other medical devices and information systems, the scope of this Standard needs to overlap with other areas of medical informatics. However, the DICOM Standard does not address the breadth of this field.

This Standard has been developed with an emphasis on diagnostic medical imaging as practiced in radiology, cardiology, pathology, dentistry, ophthalmology and related disciplines, and image-based therapies such as interventional radiology, radiotherapy and surgery. However, it is also applicable to a wide range of image and non-image related information exchanged in clinical, research, veterinary, and other medical environments.

This Standard facilitates interoperability of systems claiming conformance in a multi-vendor environment, but does not, by itself, guarantee interoperability.

With the introduction of computed tomography (CT) followed by other digital diagnostic imaging modalities in the 1970's, and the in- creasing use of computers in clinical applications, the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) recognized the emerging need for a standard method for

transferring images and associated information between devices manufactured by various vendors. These devices produce a variety of digital image formats.

The American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA) formed a joint committee in 1983 to develop a standard to:

* Promote communication of digital image information, regardless of device manufacturer
* Facilitate the development and expansion of picture archiving and communication systems (PACS) that can also interface with other systems of hospital information.
* Allow the creation of diagnostic information data bases that can be interrogated by a wide variety of devices distributed geographically.

ACR-NEMA Standards Publication No. 300-1985, published in 1985 was designated version

1.0. The Standard was followed by two revisions: No. 1, dated October 1986 and No. 2, dated January 1988. These Standards Publications specified a hardware interface, a minimum set of software commands, and a consistent set of data formats.

ACR-NEMA Standards Publication No. 300-1988, published in 1988 was designated version

2.0. It included version 1.0, the published revisions, and additional revisions. It also included new material to provide command support for display devices, to introduce a new hierarchy scheme to identify an image, and to add data elements for increased specificity when describing an image.

In 1993, ACR-NEMA Standard 300 was substantially revised and replaced by this Standard, designated Digital Imaging and Communications in Medicine (DICOM). It embodies a number of major enhancements to previous versions of the ACR-NEMA Standard:

* It is applicable to a networked environment. The ACR-NEMA Standard was applicable in a point-to-point environment only; for operation in a networked environment a Network Interface Unit (NIU) was required. DICOM supports operation in a networked environment using the industry standard networking protocol TCP/IP.
* It is applicable to off-line media exchange. The ACR-NEMA Standard did not specify a file format or choice of physical media or logical filesystem. DICOM supports operation in an off-line media environment using industry standard media such as CD-R, DVD- R and USB and common file systems.
* It is a service-oriented protocol, specifying the semantics of commands and associated data, and how devices claiming conformance to the Standard react to commands and data being exchanged. Specified services include support for management of the workflow of an imaging department. The ACR-NEMA Standard was confined to the transfer of data with only implicit service requirements.
* It specifies levels of conformance. The ACR-NEMA Standard specified a minimum level of conformance. DICOM explicitly describes how an implementor must structure a Conformance Statement to select specific options.

In 1995, with the addition of DICOM capabilities for cardiology imaging supported by the American College of Cardiology, the ACR- NEMA Joint Committee was reorganized as the DICOM Standards Committee, a broad collaboration of stakeholders across all medical imaging specialties.

## Standards used by DICOM:

The best-known standards and protocols used by DICOM are:

* + DICOM Makes use of the OSI network model. It uses the 2 network protocols on which the Internet is based and which allow data transfer, [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) / [IP,](https://en.wikipedia.org/wiki/Internet_Protocol) and the [HTTP](https://en.wikipedia.org/wiki/HTTP) hypertext transfer protocol. Additionally, DICOM has its own [MIME](https://en.wikipedia.org/wiki/MIME) content type.
  + DICOM uses other protocols such as [DHCP,](https://en.wikipedia.org/wiki/DHCP) SAML ...
  + DICOM makes use of a coding system called [SNOMED CT](https://en.wikipedia.org/wiki/SNOMED_CT) that is based on medical and clinical terms.
  + DICOM uses an external alphabet known as [LOINC](https://en.wikipedia.org/wiki/LOINC).
  + In the case of breast images, use is made of other types of structured files known as [BI-RADS.](https://en.wikipedia.org/wiki/BI-RADS)

## Data format:

DICOM groups information into data sets. For example, a file of a chest x-ray image may contain the patient ID within the file, so that the image can never be separated from this information by mistake. This is similar to the way that image formats such as JPEG can also have embedded tags to identify and otherwise describe the image.

A DICOM data object consists of a number of attributes, including items such as name, ID, etc., and also one special attribute containing the image pixel data (i.e. logically, the main object has no "header" as such, being merely a list of attributes, including the pixel data). A single DICOM object can have only one attribute containing pixel data. For many modalities, this corresponds to a single image. However, the attribute may contain multiple "frames", allowing storage of cine loops or other multi-frame data. Another example is NM data, where an NM image, by definition, is a multi-dimensional multi-frame image. In these cases, three- or four-dimensional data can be encapsulated in a single DICOM object. Pixel data can be compressed using a variety of standards, including JPEG, lossless JPEG, JPEG 2000, and run-length encoding (RLE). LZW (zip) compression can be used for the whole data set (not just the pixel data), but this has rarely been implemented.

DICOM uses three different data element encoding schemes. With explicit value representation (VR) data elements, for VRs that are not OB, OW, OF, SQ, UT, or UN, the format for each data element is: GROUP (2 bytes) ELEMENT (2 bytes) VR (2 bytes) LengthInByte (2 bytes) Data (variable length). For the other explicit data elements or implicit data elements, see section 7.1 of Part 5 of the DICOM Standard.

The same basic format is used for all applications, including network and file usage, but when written to a file, usually a true "header" (containing copies of a few key attributes and details of the application that wrote it) is added.

## RESULT:

DICOM standards and different file formats were studied.

## Ex.No.15

**IMAGE COMPRESSION TECHNIQUES**

## AIM:

To write a MATLAB program for image compression techniques.

## SOFTWARE REQUIRED:

MATLAB R2019b

## THEORY:

Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce the redundancy of the image and to store transmit data in an efficient form. A common characteristic of image data is that, they are redundant. The amount of data associated with visual information is so large and its storage requires enormous storage capacity. The transmission of this redundant data is wasteful of primary communication source. The image coding may be lossless or lossy based on application.

## TYPES:

**LOSSLESS COMPRESSION TECHNIQUE**:

In lossless compression scheme reconstructed image is same to the input image. Lossless image compression techniques first convert the images into the image pixels. Then processing is done each single pixel.

## LOSSY COMPRESSION TECHNIQUE:

Lossy compression technique provides higher compression. In this method, the compressed image is not same as the original image; there is some amount of information loss in the image.

## COMMANDS USED:

**Wfilters:**

## Syntax:

[Lo\_D,Hi\_D,Lo\_R,Hi\_R]=wfilters(wname)

wfilters(wname) returns the four low Spass and high pass, decomposition and reconstruction filters associated with the orthogonal or biorthogonal wavelet name.

## Reconstructed image:

**Syntax:**

J=Reconstructed image, it returned as numeric or logic array, depending on the input image, that is the same size as the input image.

## Resize image:

**Syntax:**

B=imresize(A,scale)

It returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If a has more than two dimensions. If scale is in the range [0,1], B is smaller than A.

## LOSSLESS COMPRESSION: PROGRAM:

clear all;

close all; input\_image1=imread('peppers.png’);

input\_image=imnoise(input\_image1,'speckle',.01); figure;

imshow(input\_image);

n=input('enter the decomposition level='); [Lo\_D,Hi\_D,Lo\_R,Hi\_R]=wfilters('haar'); [c,s]=wavedec2(input\_image,n,Lo\_D,Hi\_D); disp('the decomposition vector output is'); disp(c); [thr,nkeep]=wdcbm2(c,s,1.5,3\*prod(s(1,:)));

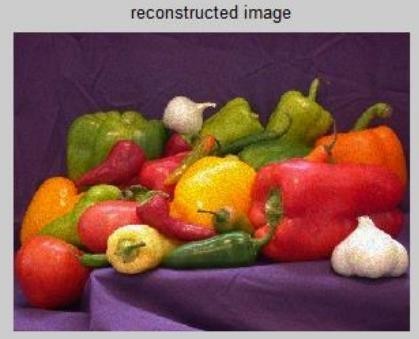
[compressed\_image,TREED,comp\_ratio,PERFL2]

=wpdencmp(thr,'s',n,'haar','threshold',5,1); disp('compression ratio in percentage'); disp(comp\_ratio); re\_ima1=wavedec2(c,s,'haar'); re\_ima=uint8(re\_ima1);

subplot(1,3,1); imshow(input\_image); title('i/p image'); subplot(1,3,2); imshow(compressed\_image); title('compressed image); subplot(1,3,3); imshow(re\_ima); title('reconstructed image');

## OUTPUT:





**LOSSY COMPRESSION:**

## PROGRAM:

clear all; close all;

input\_image1=imread('peppers.png'); input\_image=imnoise(input\_image1,'speckle',.01); figure;

imshow(input\_image);

n=input('enter the decomposition level='); [Lo\_D,Hi\_D,Lo\_R,Hi\_R] = wfilters('haar'); [c,s]=wavedec2(input\_image,n,Lo\_D,Hi\_D); disp(' the decomposition vector Output is'); disp(c);

[thr,nkeep] = wdcbm2(c,s,1.5,3\*prod(s(1,:))); [compressed\_image,TREED,comp\_ratio,PERFL2] =wpdencmp(thr,'s',n,'haar','threshold',5,1); disp('compression ratio in percentage');

disp(comp\_ratio);

re\_ima1 = waverec2(c,s,'haar'); re\_ima=uint8(re\_ima1); subplot(1,3,1); imshow(input\_image);

title('i/p image'); subplot(1,3,2); imshow(compressed\_image); title('compressed image'); subplot(1,3,3); imshow(re\_ima); title('reconstructed image');

## OUTPUT:



**Original image Transformation image**

## RESULT:

Image Compression techniques were applied in different input images and outputs are plotted and analyzed using MATLAB software successfully.

## Ex.No.16

**IMAGE RESTORATION**

## AIM:

To write a MATLAB program for image restoration.

## SOFTWARE REQUIRED:

MATLAB R2019b

## THEORY:

Image restoration is the operation of taking a corrupt/noisy image and estimating the clean, original image. Corruption may come in many forms such as motion blur, noise and camera mis-focus. Image restoration is performed by reversing the process that blurred the image and such is performed by imaging a point source and use the point source image, which is called the Point Spread Function (PSF) to restore the image information lost to the blurring process.

Image restoration is different from image enhancement in that the latter is designed to emphasize features of the image that make the image more pleasing to the observer, but not necessarily to produce realistic data from a scientific point of view. Image enhancement techniques (like contrast stretching or de-blurring by a nearest neighbor procedure) provided by imaging packages use no a priori model of the process that created the image.

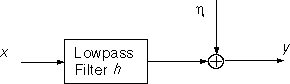
With image enhancement noise can effectively be removed by sacrificing some resolution, but this is not acceptable in many applications. In a fluorescence microscope, resolution in the z-direction is bad as it is. More advanced image processing techniques must be applied to recover the object.

The objective of image restoration techniques is to reduce noise and recover resolution loss Image processing techniques are performed either in the image domain or the frequency domain. The most straightforward and a conventional technique for image restoration is deconvolution, which is performed in the frequency domain and after computing the Fourier transform of both the image and the PSF and undo the resolution loss caused by the blurring factors. This deconvolution technique, because of its direct inversion of the PSF which typically has poor matrix condition number, amplifies noise and creates an imperfect deblurred image. Also, conventionally the blurring process is assumed to be shift-invariant. Hence more sophisticated techniques, such as regularized deblurring, have been developed to offer robust recovery under different types of noises and blurring functions. It is of 3 types:

1. Geometric correction 2. radiometric correction 3. noise removal.

The purpose of image restoration is to "compensate for" or "undo" defects which degrade an image. Degradation comes in many forms such as motion blur, noise, and camera misfocus. In cases like motion blur, it is possible to come up with an very good estimate of the actual blurring function and "undo" the blur to restore the original image. In cases where the image is corrupted by noise, the best we may hope to do is to compensate for the degradation it

caused. In this project, we will introduce and implement several of the methods used in the image processing world to restore images.

The block diagram for our general degradation model is

where g is the corrupted image obtained by passing the original image f through a low pass filter (blurring function) b and adding noise to it. We present four different ways of restoring the image.

1. Inverse Filter

In this method we look at an image assuming a known blurring function. We will see that restoration is good when noise is not present and not so good when it is.

1. Weiner Filtering

In this section we implement image restoration using wiener filtering, which provides us with the optimal trade-off between de-noising and inverse filtering. We will see that the result is in general better than with straight inverse filtering.

1. Wavelet Restoration

We implement three wavelet-based algorithms to restore the image.

1. Blind Deconvolution

In this method, we assume nothing about the image. We do not have any information about the blurring function or on the additive noise. We will see that restoring an image when we know nothing about it is very hard.

## PROGRAM:

I = im2double(imread('cameraman.tif')); imshow(I);

title('Original Image'); LEN = 21;

THETA = 11;

PSF = fspecial('motion', LEN, THETA); blurred = imfilter(I, PSF, 'conv', 'circular'); figure, imshow(blurred);

noise\_mean = 0;

noise\_var = 0.0001;

blurred\_noisy = imnoise(blurred, 'gaussian', ...

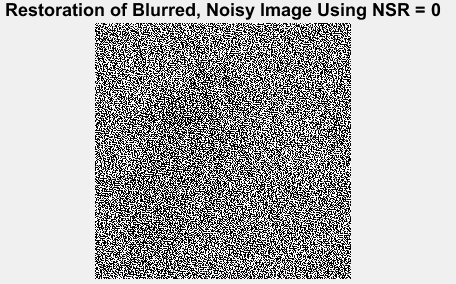
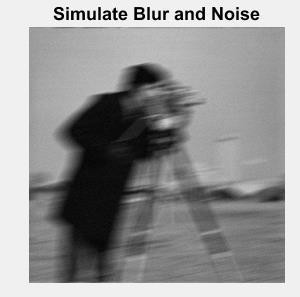
noise\_mean, noise\_var); figure, imshow(blurred\_noisy); title('Simulate Blur and Noise') estimated\_nsr = 0;

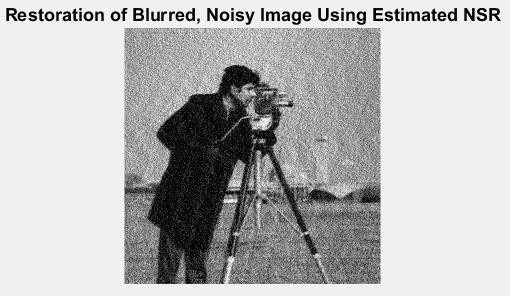
wnr2 = deconvwnr(blurred\_noisy, PSF, estimated\_nsr); figure, imshow(wnr2);

title('Restoration of Blurred, Noisy Image Using NSR = 0'); estimated\_nsr = noise\_var / var(I(:));

wnr3 = deconvwnr(blurred\_noisy, PSF, estimated\_nsr); figure, imshow(wnr3)

title('Restoration of Blurred, Noisy Image Using Estimated NSR');





## RESULT:

Image Restoration techniques were applied in input image and outputs are plotted and analyzed using MATLAB software successfully.