SRM VALLIAMMAI ENGINEERING COLLEGE

(AN AUTONOMOUS INSTITUTION)
SRM NAGAR, KATTANKULATHUR – 603 203.

DEPARTMENT
OF
MEDICAL ELECTRONICS

1910706 DIGITAL IMAGE PROCESSING LABORATORY

IV YEAR / VII SEMESTER MDE

LABORATORY MANUAL

ACADEMIC YEAR: 2025-2026 ODD SEMESTER

1910706 DIGITAL IMAGE PROCESSING LABORATORY

PREPARED BY
Dr D RAVIKUMAR, ASSOCIATE PROFESSOR / MDE

Syllabus

1910706 DIGITAL IMAGE PROCESSING LABORATORY

L T P C 0 0 4 2

SIMULATION USING MATLAB

1. Image sampling and quantization.
2. Analysis of spatial and intensity resolution of images.
3. Intensity transformation of images.
4. DFT analysis of images.
5. Transforms (Walsh, Hadamard, DCT, Haar).
6. Histogram Processing and Basic Thresholding functions.
7. Image Enhancement-Spatial filtering.
8. Image Enhancement- Filtering in frequency domain.
9. Image segmentation – Edge detection, line detection and point detection.
10. Basic Morphological operations.
11. Region based Segmentation.
12. Segmentation using watershed transformation.
13. Analysis of images with different color models.
14. Study of DICOM standards.
15. Image compression techniques.
16. Image restoration.
17. A mini project based on medical image processing.

TOTAL: 60 PERIODS

LIST OF EXPERIMENTS

CYCLE-I

SIMULATION USING MATLAB

1. Image sampling and quantization.
2. Analysis of spatial and intensity resolution of images.
3. Intensity transformation of images.
4. DFT analysis of images.
5. Transforms (Walsh, Hadamard, DCT, Haar).
6. Histogram Processing and Basic Thresholding functions.
7. Image Enhancement-Spatial filtering.
8. Image Enhancement- Filtering in frequency domain.

CYCLE-II

SIMULATION USING MATLAB

1. Image segmentation – Edge detection, line detection and point detection.
2. Basic Morphological operations.
3. Region based Segmentation.
4. Segmentation using watershed transformation.
5. Analysis of images with different color models.
6. Study of DICOM standards.
7. Image compression techniques.
8. Image restoration.
9. A mini project based on medical image processing.

1910706 DIGITAL IMAGE PROCESSING LABORATORY

LIST OF EXPERIMENTS

Simulation using MATLAB

1. Image sampling and quantization
2. Analysis of spatial and intensity resolution of images.
3. Intensity transformation of images.
4. DFT analysis of images
5. Transforms (Walsh, Hadamard, DCT, Haar)
6. Histogram Processing
7. Image Enhancement-Spatial filtering
8. Image Enhancement- Filtering in frequency domain
9. Image segmentation – Edge detection, line detection and point detection
10. Basic Morphological operations.
11. Basic Thresholding functions
12. Analysis of images with different color models.

MINI PROJECTS:

1. Applications to Biometric and security
2. Applications to Medical Images
3. Texture analysis with statistical properties
4. Boundary detection

INDEX

S.No	Title of the Experiment
	Introduction to MATLAB
1	Image sampling and quantization
2	Analysis of spatial and intensity resolution of images.
3	Intensity transformation of images.
4	DFT analysis of images
5a	Walsh Transform
5b	Hadamard Transform
5c	DCT Transform
5d	Haar Transform
6	Histogram Processing
7	Image Enhancement-Spatial filtering
8	Image Enhancement- Filtering in frequency domain
9a	Edge detection
9b	line detection
9c	point detection
10	Basic Morphological operations
11	Basic Thresholding functions
12	Analysis of images with different color models

Introduction

MATLAB stands for MATrix LABoratory and the software is built up around vectors and matrices. It is a technical computing environment for high performance numeric computation and visualization. It integrates numerical analysis, matrix computation, signal processing and graphics in an easy-to-use environment, where problems and solutions are expressed just as they are written mathematically, without traditional programming. MATLAB is an interactive system whose basic data element is a matrix that does not require dimensioning. It enables us to solve many numerical problems in a fraction of the time that it would take to write a program and execute in a language such as FORTRAN, BASIC, or C. It also features a family of application specific solutions, called toolboxes. Areas in which toolboxes are available include signal processing, image processing, control systems design, dynamic systems simulation, systems identification, neural networks, wavelength communication and others. It can handle linear, non- linear, continuous-time, discrete-time, multivariable and multirate systems.

How to start MATLAB

Choose the submenu "Programs" from the "Start" menu. From the "Programs" menu, open the "MATLAB" submenu. From the "MATLAB" submenu, choose "MATLAB".

Procedure

1. Open Matlab.
2. File → New → Script.
3. Type the program in untitled window
4. File → Save → type filename.m in Matlab workspace path.
5. Debug → Run.
6. Output will be displayed at Figure dialog box.

Library Functions

clc:

Clear command window

Clears the command window and homes the cursor.

clear all:

Removes all variables from the workspace.

close all:

Closes all the open figure windows.

exp:

Y = exp(X) returns the exponential e^x for each element in array X.

linespace:

y = linspace(x1,x2) returns a row vector of 100 evenly spaced points between x1 and x2.

rand:
X = rand returns a single uniformly distributed random number in the interval (0,1).

ones:
X = ones(n) returns an n-by-n matrix of ones.

zeros:
X = zeros(n) returns an n-by-n matrix of zeros.

plot:
plot(X,Y) creates a 2-D line plot of the data in Y versus the corresponding values in X.

subplot:
subplot(m,n,p) divides the current figure into an m-by-n grid and creates an axes for a subplot in the position specified by p.

stem:
stem(Y) plots the data sequence, Y, as stems that extend from a baseline along the *x*-axis. The data values are indicated by circles terminating each stem.

title:
title(str) adds the title consisting of a string, str, at the top and in the center of the current axes.

xlabel:
xlabel(str) labels the *x*-axis of the current axes with the text specified by str.

ylabel:
ylabel(str) labels the *y*-axis of the current axes with the string, str.

A Summary of MATLAB Commands Used

imread	Read image from graphics file
imwrite	Write image to graphics file
imfinfo	Information about graphics file
imshow	Display image
implay	Play movies, videos, or image sequences
gray2ind	Convert grayscale or binary image to indexed image
ind2gray	Convert indexed image to grayscale image
mat2gray	Convert matrix to grayscale image
rgb2gray	Convert RGB image or colormap to grayscale
imbinarize	Binarize image by thresholding
adapthresh	Adaptive image threshold using local first-order statistics
otsuthresh	Global histogram threshold using Otsu's method
im2uint16	Convert image to 16-bit unsigned integers
im2uint8	Convert image to 8-bit unsigned integers
imcrop	Crop image

imresize	Resize image
imrotate	Rotate image
imadjust	Adjust image intensity values or colormap
imcontrast	Adjust Contrast tool
imsharpen	Sharpen image using unsharp masking
histeq	Enhance contrast using histogram equalization
adapthisteq	Contrast-limited adaptive histogram equalization (CLAHE)
imhistmatch	Adjust histogram of image to match N-bin histogram of reference image
imnoise	Add noise to image
imfilter	N-D filtering of multidimensional images
fspecial	Create predefined 2-D filter
weiner2	2-D adaptive noise-removal filtering
medfilt2	2-D median filtering
ordfilt2	2-D order-statistic filtering
imfill	Fill image regions and holes
imclose	Morphologically close image
imdilate	Dilate image
imerode	Erode image
imopen	Morphologically open image
imreconstruct	Morphological reconstruction
watershed	Watershed transform
dct2	2-D discrete cosine transform
hough	Hough transform
graydist	Gray-weighted distance transform of grayscale image
fft2	2-D fast Fourier transform
ifftshift	Inverse FFT shift
imcomplement	Complement image
immultiply	Multiply two images or multiply image by constant
imsubtract	Subtract one image from another or subtract constant from image
imdivide	Divide one image into another or divide image by constant
imadd	Add two images or add constant to image

Ex. No.1 **IMAGE SAMPLING AND QUANTIZATION**

AIM

To perform image sampling and quantization using Matlab.

SOFTWARE USED

MATLAB

THEORY

In order to become suitable for digital processing, an image function f(x,y) must be digitized both spatially and in amplitude. Typically, a frame grabber or digitizer is used to sample and quantize the analogue video signal. Hence in order to create an image which is digital, we need to covert continuous data into digital form. There are two steps in which it is done:

- Sampling
- Quantization

The sampling rate determines the spatial resolution of the digitized image, while the quantization level determines the number of grey levels in the digitized image. A magnitude of the sampled image is expressed as a digital value in image processing. The transition between continuous values of the image function and its digital equivalent is called quantization.

The number of quantization levels should be high enough for human perception of fine shading details in the image. The occurrence of false contours is the main problem in image which has been quantized with insufficient brightness levels.

PROGRAM

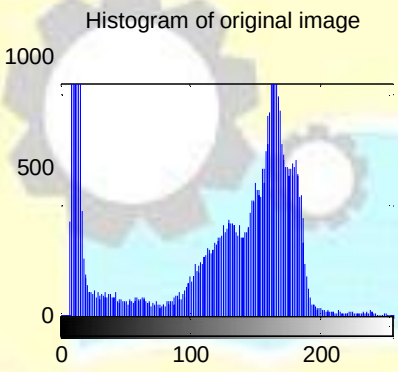
% Uniform Quantization

```
a=imread('cameraman.tif');
subplot(2,2,1)
imshow(a);
title('Original image');
subplot(2,2,2);
imhist(a);
title('Histogram of original image');
[m n]=size(a);
for i=1:1:m
for j=1:1:n
if a(i,j)<16 a(i,j)=7;
elseif a(i,j)>=16 && a(i,j)<32 a(i,j)=23;
elseif a(i,j)>=32 && a(i,j)<48 a(i,j)=39;
elseif a(i,j)>=48 && a(i,j)<64 a(i,j)=55;
elseif a(i,j)>=64 && a(i,j)<80 a(i,j)=71;
elseif a(i,j)>=80 && a(i,j)<96 a(i,j)=87;
```

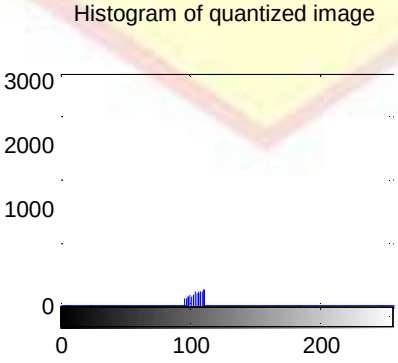
```
elseif a(i,j)>=96 && a(i,j)<96 a(i,j)=103;
elseif a(i,j)>=112 && a(i,j)<128 a(i,j)=119;
elseif a(i,j)>=128 && a(i,j)<144 a(i,j)=135;
elseif a(i,j)>=144 && a(i,j)<160 a(i,j)=151;
elseif a(i,j)>=160 && a(i,j)<176 a(i,j)=167;
elseif a(i,j)>=176 && a(i,j)<192 a(i,j)=183;
elseif a(i,j)>=192 && a(i,j)<208 a(i,j)=199;
elseif a(i,j)>=208 && a(i,j)<224 a(i,j)=215;
elseif a(i,j)>=224 && a(i,j)<240 a(i,j)=231;
elseif a(i,j)>=240 && a(i,j)<256 a(i,j)=247;
end
end
end
subplot(2,2,3)
imshow(a);
title('Quantised image')
subplot(2,2,4)
imhist(a);
title('Histogram of quantized image')
```

OUTPUT

Original image

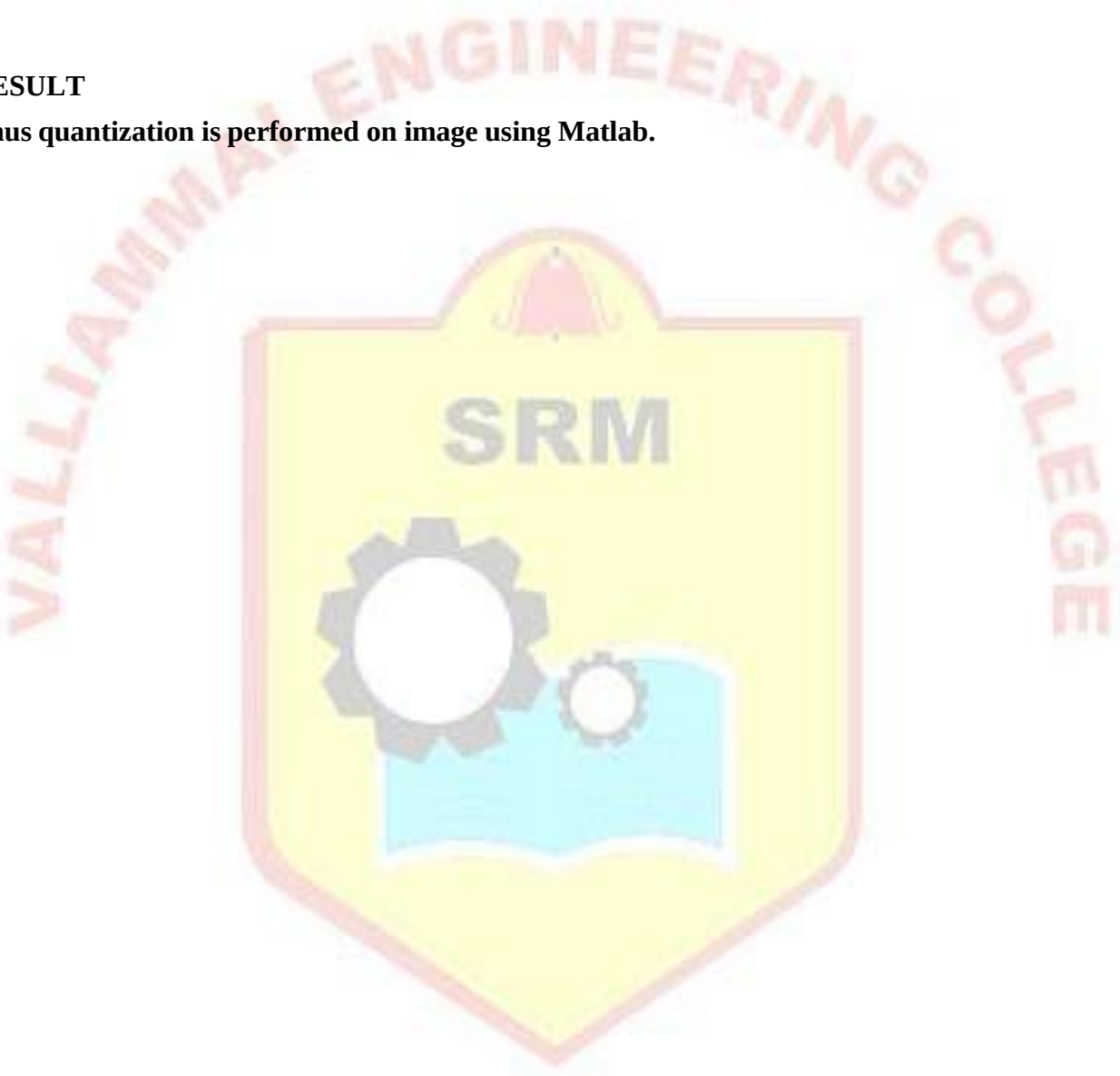


Quantised image



RESULT

Thus quantization is performed on image using Matlab.



Ex. No.2 ANALYSIS OF SPATIAL AND INTENSITY RESOLUTION OF IMAGES

AIM
To analyze spatial and intensity resolution of images using Matlab.

SOFTWARE USED
MATLAB

THEORY
Spatial resolution

Spatial resolution can be defined as the smallest discernible detail in an image. In other way spatial resolution is defined as the number of independent pixels values per inch. Since the spatial resolution refers to clarity, so for different devices, different measure has been made to measure it.

For example

- Dots per inch
- Lines per inch
- Pixels per inch

They are discussed in more detail in the next tutorial but just a brief introduction has been given below.

Dots per inch

Dots per inch or DPI is usually used in monitors.

Lines per inch

Lines per inch or LPI is usually used in laser printers.

Pixel per inch

Pixel per inch or PPI is measure for different devices such as tablets , Mobile phones e.t.c.

PROGRAM
SPATIAL RESOLUTION

```
z=imread('cameraman.tif');
z=imresize(z,[1024,1024]);
[r c]=size(z);
l=1;
for i=1:2:r
   k=1;
   for j=1:2:c
      a(l,k)=z(i,j);
      k=k+1;
   end
   l=l+1;
end
```

```
l=1;
for i=1:4:r
   k=1;
   for j=1:4:c
      b(l,k)=z(i,j);
      k=k+1;
   end
   l=l+1;
end
l=1;
for i=1:8:r
    k=1;
    for j=1:8:c
       e(l,k)=z(i,j);
       k=k+1;
    end
    l=l+1;
end
l=1;
for i=1:16:r
   k=1;
   for j=1:16:c
      d(l,k)=z(i,j);
      k=k+1;
   end
   l=l+1;
end
subplot(2,2,1),imshow(a)
subplot(2,2,2),imshow(b)
subplot(2,2,3),imshow(e)
subplot(2,2,4),imshow(d)
```

INTENSITY RESOLUTION

```
% Reading the image and converting it to a gray-level image.
I=imread('saturn.png');
I=rgb2gray(I);
% A 256 gray-level image:
[I256,map256]=gray2ind(I,256);
subplot(2,2,1);
```

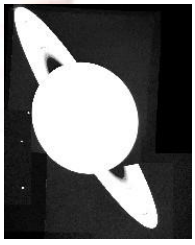
```
imshow(I256,map256);
% A 128 gray-level image:
[I128,map128]=gray2ind(I,128);
subplot(2,2,2);
imshow(I128,map128);
% A 64 gray-level image:
[I64,map64]=gray2ind(I,64)
subplot(2,2,3);
imshow(I64,map64);
% A 32 gray-level image:
[I32,map32]=gray2ind(I,32);
subplot(2,2,4);
imshow(I32,map32);
% A 16 gray-level image:
[I16,map16]=gray2ind(I,16);
figure,
subplot(2,2,1);
imshow(I16,map16);
% A 8 gray-level image:
[I8,map8]=gray2ind(I,8);
subplot(2,2,2);
imshow(I8,map8);
% A 4 gray-level image:
[I4,map4]=gray2ind(I,4);
subplot(2,2,3);
imshow(I4,map4);
% A 2 gray-level image:
[I2,map2]=gray2ind(I,2);
subplot(2,2,4);
imshow(I2,map2);
```

OUTPUT

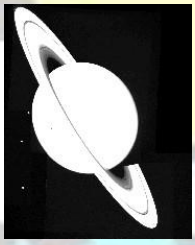
SPATIAL RESOLUTION

INTENSITY RESOLUTION

32



64



2

4

128

256

8

16

RESULT:

Thus analysis of spatial and intensity resolution is performed in image using Matlab.

Ex. No.3 **INTENSITY TRANSFORMATION OF IMAGES.**

AIM

To perform intensity transformation of images using Matlab.

SOFTWARE USED

MATLAB

THEORY

Photographic Negative:

The negative of an image with gray levels in the range [0, L-1] is obtained by using the negative transformation

$$s=L-1-r$$

Reversing the intensity levels of an image produces the equivalent of a photographic negative. This type of processing is suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.

Gamma Transformation

With Gamma Transformations, it is possible to curve the grayscale components either to brighten the intensity (when gamma is less than one) or darken the intensity (when gamma is greater than one).

Logarithmic Transformation

The general form of the log transformation

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

Where c is a constant and it is assumed that $r \geq 0$. The shape of the log curve shows that the transformation maps a narrow range of low gray-level values in the input image into a wider range of output levels. The opposite is true of higher values of input levels. This transformation is used to expand the values of dark pixels in an image while compressing the higher-level values. The opposite is true of the inverse log transformation.

Contrast stretching Transformation

Contrast-stretching transformations increase the contrast between the darks and the lights.

PROGRAM

1. Photographic Negative

```
I=imread('cameraman.tif');
imshow(I)
J=imcomplement(I);
figure, imshow(J)
```

2. Gamma Transformation

```
I=imread('tire.tif');
subplot(2,2,1);
imshow(I)
J=imadjust(I,[],[],1);
J2=imadjust(I,[],[],3);
J3=imadjust(I,[],[],0.4);
subplot(2,2,2);
imshow(J);
subplot(2,2,3);
imshow(J2);
subplot(2,2,4);
imshow(J3);
```

3. Logarithmic Transformation

```
tire = imread('tire.tif');
d = im2double(tire);
figure, imshow(d);
%log on domain [0,1]
f = d;
c = 1/log(1+1);
j1 = c*log(1+f);
figure, imshow(j1);
%log on domain [0, 255]
f = d*255;
c = 1/log(1+255);
j2 = c*log(1+f);
figure, imshow(j2);
%log on domain [0, 2^16]
f = d*2^16;
c = 1/log(1+2^16);
j3 = c*log(1+f);
figure, imshow(j3);
```

4. Contrast Stretching with changing E

```
I=imread('tire.tif');
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)).^10);
imshow(I2)
figure,imshow(contrast1)
figure,imshow(contrast2)
figure,imshow(contrast3)
```

5. Contrast Stretching with changing m

```
I=imread('tire.tif');
I2=im2double(I);
contrast1=1./(1+(0.2./(I2+eps)).^4)
contrast2=1./(1+(0.5./(I2+eps)).^4);
contrast3=1./(1+(0.7./(I2+eps)).^4);
imshow(I2)
figure,imshow(contrast1)
figure,imshow(contrast2)
figure,imshow(contrast3)
```

OUTPUT

Photographic Negative

Original Image

Photographic Negative

Gamma Transformation

Original Image

Gamma=1

Gamma=3

Gamma=0.4

Logarithmic Transformation

Original Image

log on domain [0, 1]

log on domain [0, 255]

log on domain [0, 65535]

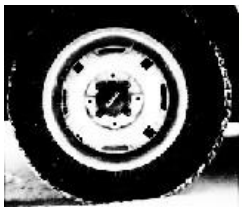
Contrast Stretching

Changing E

Original Image

E=4

E=5

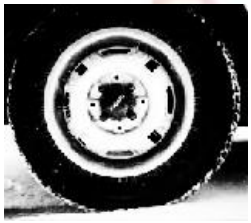


E=10

Changing m

Original image

m=0.2



m=0.5

m=0.7

RESULT

Thus intensity transformation of images is performed using Matlab.

Ex. No.4 **DFT ANALYSIS OF IMAGES**

AIM

To apply Discrete Fourier Transform on image and study its properties using Matlab.

APPARATUS REQUIRED

PC with Matlab

THEORY

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency containedin the spatial domain image.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression.

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 number of frequencies corresponds to the number of pixels in the spatial domain image, *i.e.* the image in the spatial and Fourier domains are of the same size.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-i2\pi\left(\frac{ki}{N} + \frac{lj}{N}\right)}$$

where f(a,b) is the image in the spatial domain and the exponential term is the basis function corresponding to each point F(k,l*)* in the Fourier space.

The inverse Fourier transform is given by:

$$f(a,b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} F(k,l)\, e^{i2\pi\left(\frac{ka}{N} + \frac{lb}{N}\right)}$$

PROGRAM

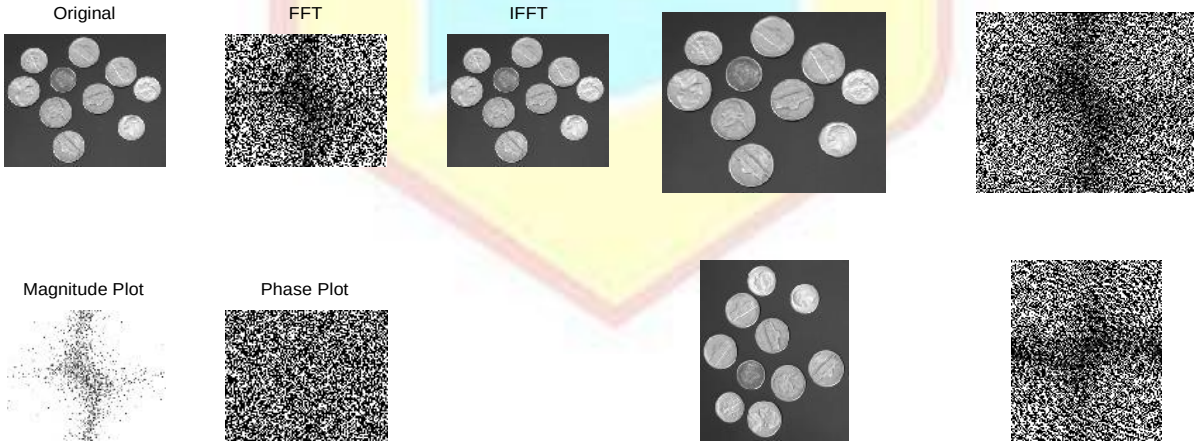
```
a=imread('Coins.png');
subplot(2,3,1);
imshow(a);
title('Original');
b=im2double(a);
c=fft2(b);
subplot(2,3,2);
imshow(c);
title('FFT');
d=ifft2(c);
```

```
subplot(2,3,3);
imshow(d);
title('IFFT');
mag=abs(c);
subplot(2,3,4);
imshow(mag);
title('Magnitude Plot');
ang=angle(c);
subplot(2,3,5);
imshow(ang);
title('Phase Plot');
```

ROTATION PROPERTY:

```
a=imread('coins.png');
subplot(2,2,1);
imshow(a);
a1=im2double(a);
b=fft2(a1);
subplot(2,2,2);
imshow(b);
c=imrotate(a1,90);
subplot(2,2,3);
imshow(c);
d=fft2(c);
subplot(2,2,4);
imshow(d);
```

OUTPUT



RESULT

Thus Discrete Fourier Transform is applied on image and its properties is studied using Matlab.

Ex. No.5a **WALSH TRANSFORM**

AIM

To implement Walsh transform on image.

SOFTWARE USED

MATLAB

THEORY

When $N=2^n$, the 2-D forward and inverse Walsh kernels are given by the relations

$$g(x,y,u,v) = \frac{1}{N}\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_{n-1-i}(u)+b_i(y)b_{n-1-i}(v)]} \quad \text{and} \quad h(x,y,u,v) = \frac{1}{N}\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_{n-1-i}(u)+b_i(y)b_{n-1-i}(v)]}$$

Where $b_k(z)$ is the kth bit in the binary representation of z.

So the forward and inverse Walsh transforms are equal in form; that is:

$$W(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_{n-1-i}(u)+b_i(y)b_{n-1-i}(v)]}$$

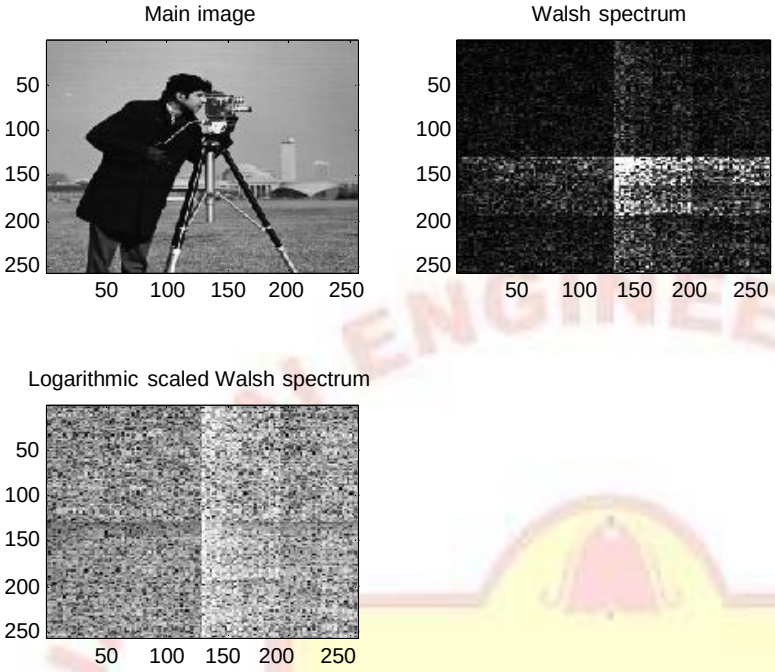
$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} W(u,v)\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_{n-1-i}(u)+b_i(y)b_{n-1-i}(v)]}.$$

PROGRAM

```
% Getting the name and extension of the image file from the user.
a=imread('cameraman.tif');
N=length(a);
% Computing Walsh Transform of the image file.
n=log2(N);
n=1+fix(n);
f=ones(N,N);
for x=1:N;
for u=1:N
p=dec2bin(x-1,n);
q=dec2bin(u-1,n);
for i=1:n;
f(x,u)=f(x,u)*((-1)^(p(n+1-i)*q(i)));
end;
end;
```

```
end;
F=(1/N)*f*double(a)*f;
% Shifting the Fourier spectrum to the center of the frequency square.
for i=1:N/2;    for j=1:N/2
G(i+N/2,j+N/2)=F(i,j);
end;
end
for i=N/2+1:N;
for j=1:N/2
G(i-N/2,j+N/2)=F(i,j);
end;
end
for i=1:N/2;
for j=N/2+1:N
G(i+N/2,j-N/2)=F(i,j);
end;
end
for i=N/2+1:N;
for j=N/2+1:N;
G(i-N/2,j-N/2)=F(i,j);
end;
end;
% Computing and scaling the logarithmic Walsh spectrum.
H=log(1+abs(G));
for i=1:N
H(i,:)=H(i,:)*255/abs(max(H(i,:)));
end
% Changing the color map to gray scale (8 bits).
colormap(gray(255));
% Showing the main image and its Walsh spectrum.
subplot(2,2,1),image(a),title('Main image');
subplot(2,2,2),image(abs(G)),title('Walsh spectrum');
subplot(2,2,3),image(H),title('Logarithmic scaled Walsh spectrum');
```

OUTPUT



RESULT

Thus Walsh transform is implemented in the image using Matlab.

Ex. No.5b **HADAMARD TRANSFORM**

AIM

To implement Hadamard transform on image.

SOFTWARE USED

MATLAB

THEORY

When $N=2^n$, the 2-D forward and inverse Hadamard kernels are given by the relations

$$g(x,y,u,v) = \frac{1}{N}\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_i(u)+b_i(y)b_i(v)]} \quad \text{and} \quad h(x,y,u,v) = \frac{1}{N}\prod_{i=0}^{n-1}(-1)^{[b_i(x)b_i(u)+b_i(y)b_i(v)]}$$

Where $b_k(z)$ is the kth bit in the binary representation of z.

So the forward and inverse Hadamard transforms are equal in form; that is:

$$H(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)(-1)^{\sum_{i=0}^{n-1}[b_i(x)b_i(u)+b_i(y)b_i(v)]}$$

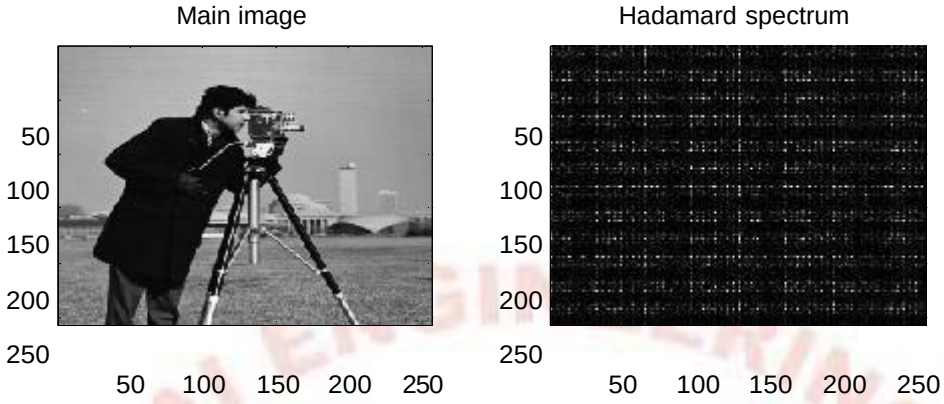
$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u,v)(-1)^{\sum_{i=0}^{n-1}[b_i(x)b_i(u)+b_i(y)b_i(v)]}$$

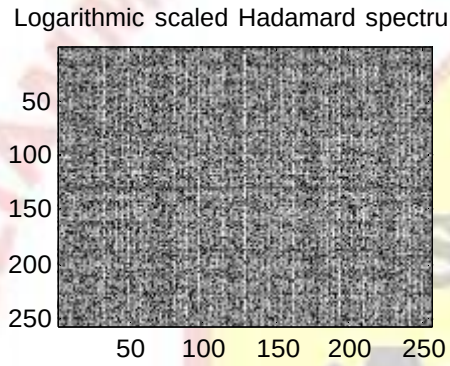
PROGRAM

```
% Getting the name and extension of the image file from the user
a=imread('cameraman.tif');
N=length(a);
%Computing Hadamard Transform of the image file
n=log2(N);
n=1+fix(n);
f=ones(N,N);
for x=1:N;
for u=1:N
p=dec2bin(x-1,n);
q=dec2bin(u-1,n);
for i=1:n;
f(x,u)=f(x,u)*((-1)^(p(n+1-i)*q(n+1-i)));
end;
end;
end;
```

```
F=(1/N)*f*double(a)*f;
% Shifting the Fourier spectrum to the center of the frequency square.
for i=1:N/2;
   for j=1:N/2
G(i+N/2,j+N/2)=F(i,j);
end;
end
for i=N/2+1:N;
for j=1:N/2
G(i-N/2,j+N/2)=F(i,j);
end;
end
for i=1:N/2;
for j=N/2+1:N
G(i+N/2,j-N/2)=F(i,j);
end;
end
for i=N/2+1:N;
for j=N/2+1:N;
G(i-N/2,j-N/2)=F(i,j);
end;
end;
% Computing and scaling the logarithmic Hadamard spectrum.
H=log(1+abs(G));
for i=1:N
H(i,:)=H(i,:)*255/abs(max(H(i,:)));
end
% Changing the color map to gray scale (8 bits).
colormap(gray(255));
% Showing the main image and its Hadamard spectrum.
subplot(2,2,1),image(a),title('Main image');
subplot(2,2,2),image(abs(G)),title('Hadamard spectrum');
subplot(2,2,3),image(H),title('Logarithmic scaled Hadamard spectrum');
```

OUTPUT





RESULT

Thus Hadamard transform is implemented in the image using Matlab.

Ex. No.5c **DISCRETE COSINE TRANSFORM**

AIM

To apply Discrete Cosine Transform on image using Matlab.

SOFTWARE USED

MATLAB

THEORY

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]\cos\left[\frac{(2y + 1)v\pi}{2N}\right]$$

for $u, v = 0, 1, 2, \ldots, N - 1$, and

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u, v)\cos\left[\frac{(2x + 1)u\pi}{2N}\right]\cos\left[\frac{(2y + 1)v\pi}{2N}\right]$$

for $x, y = 0, 1, 2, \ldots, N - 1$, where α is

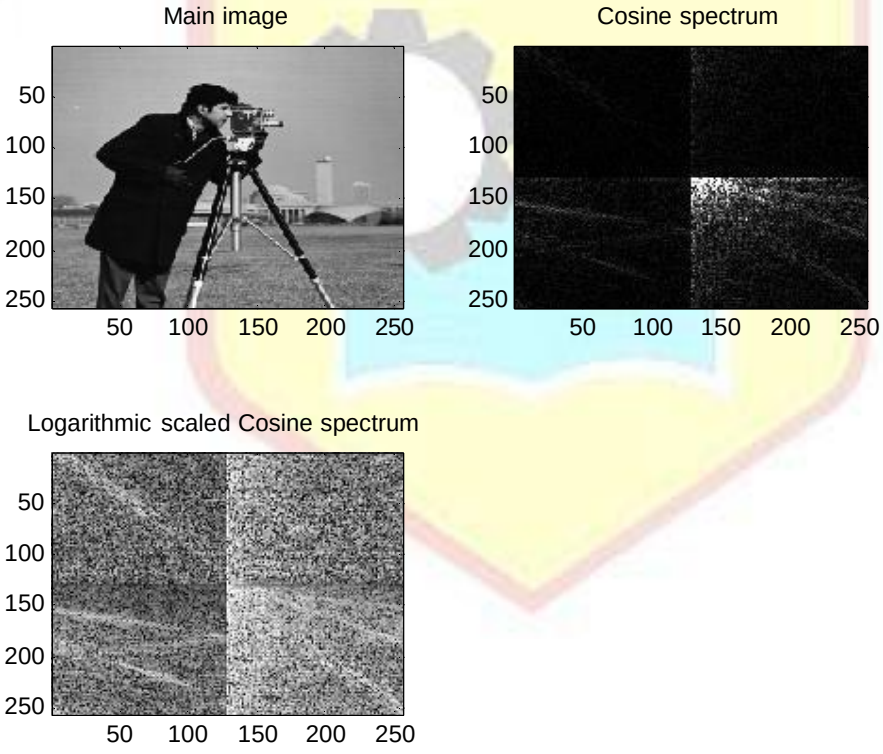
$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \ldots, N - 1. \end{cases}$$

PROGRAM

```
a=imread('cameraman.tif');
N=length(a);
F=dct2(double(a));
% Shifting the Fourier spectrum to the center of the frequency square.
for i=1:N/2;
   for j=1:N/2
G(i+N/2,j+N/2)=F(i,j);
end;
end
for i=N/2+1:N;
for j=1:N/2
G(i-N/2,j+N/2)=F(i,j);
end;
end
for i=1:N/2;
for j=N/2+1:N
G(i+N/2,j-N/2)=F(i,j);
```

```
end;
end
for i=N/2+1:N;
for j=N/2+1:N;
G(i-N/2,j-N/2)=F(i,j);
end;
end;
% Computing and scaling the logarithmic Cosine spectrum.
H=log(1+abs(G));
for i=1:N
H(i,:)=H(i,:)*255/abs(max(H(i,:)));
end
% Changing the color map to gray scale (8 bits).
colormap(gray(255));
% Showing the main image and its Cosine spectrum.
subplot(2,2,1),image(a),title('Main image');
subplot(2,2,2),image(abs(G)),title('Cosine spectrum');
subplot(2,2,3),image(H),title('Logarithmic scaled Cosine spectrum');
```

OUTPUT



RESULT

Thus cosine transform is implemented in the image using Matlab.

Ex. No.5d **HAAR TRANSFORM**

AIM

To apply Harr Transform on image using Matlab.

SOFTWARE USED

MATLAB

THEORY

The Haar transform is based on the Haar functions, $h_i(z)$, which are defined over the continuous, closed interval [0,1] for z, and for k=0,1,2,…,N-1, where $N=2^n$. The first step in generating the Haar transform is to note that the integer k can be decomposed uniquely as $k=2^p+q-1$

where $0\le p\le n-1$, q=0 or 1 for p=0, and $1\le q\le 2^p$ for $p\ne 0$.

With this background, the Haar functions are defined as

$$h_0(z)\triangleq h_{00}(z)=\frac{1}{\sqrt{N}} \qquad \text{for } z\in[0,1]$$

and

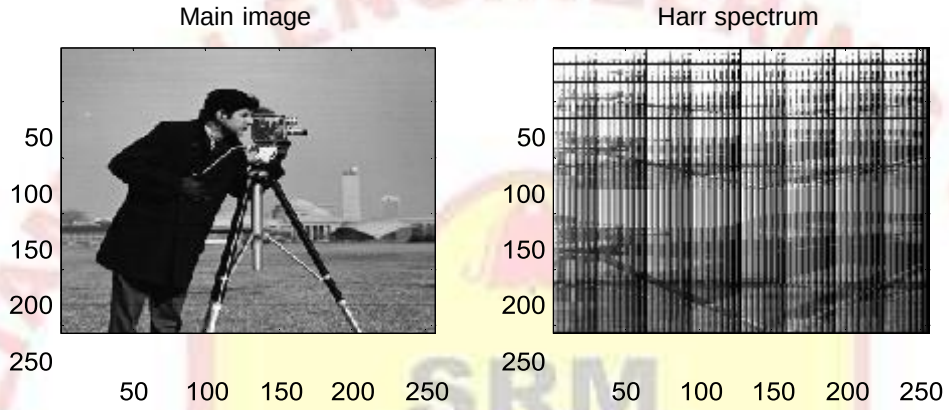
$$h_k(z)\triangleq h_{00}(z)=\frac{1}{\sqrt{N}}\begin{cases}2^{p/2} & \frac{q-1}{2^p}\le z<\frac{q-1/2}{2^p}\\ -2^{p/2} & \frac{q-1/2}{2^p}\le z<\frac{q}{2^p}\\ 0 & \text{otherwise for } z\in[0,1]\end{cases}$$

PROGRAM

```
a=imread('cameraman.tif');
N=length(a);
for i=1:N;
p=fix(log2(i));
q=i-(2^p);
for j=1:N
z=(j-1)/N;
if(z>=(q-1)/(2^p))&&(z<(q-1/2)/2^p)
f(i,j)=(1/(sqrt(N)))*(2^(p/2));
elseif(z>=(q-1)/(2^p))&&(z<(q/2)/2^p)
f(i,j)=(1/(sqrt(N)))*(-2^(p/2));
else f(i,j)=0;
end;
end;
end;
F=f*double(a)*f;
```

```
% Changing the color map to gray scale (8 bits).
colormap(gray(255));
% Showing the main image and its Harr spectrum.
subplot(2,2,1),image(a),title('Main image');
subplot(2,2,2),image(abs(F)),title('Harr spectrum');
```

OUTPUT



RESULT

Thus Harr transform is applied on image using Matlab.

Ex. No.6 **HISTOGRAM PROCESSING**

AIM

To study the histogram and histogram equalization.

SOFTWARE USED

MATLAB

THEORY

The histogram of a digital image with gray values r_0, r_1….r^{L-1} is the discrete function

$$P(r_k)=n_k/n$$

n_k : Number of pixels with gray value r_k .

n: Total number of pixels in the image

The function $P(r_k)$ represents the fraction of the total number of pixels with gray value r_k. Histogram provides a global description of the appearance of the image. The shape of the histogram provides useful information for contrast enhancement. The histogram equalization is an approach to enhance a given image. The approach is to design a transformation T(.) such that the gray values in the output is uniformly distributed in [0,1].In terms of histograms, the output image will have all gray values in equal proportion.

PROGRAM

% histogram without inbuilt function.

```
histo=zeros(1,256);
I=imread('cameraman.tif');
imshow(I);
si=size(I);
for i=1:si(1)
for j=1:si(2)
for g=1:256
if I(i,j)==g
histo(g)=histo(g)+1;
end
end
end
end
figure,stem(histo)
```

%histogram with inbuilt function

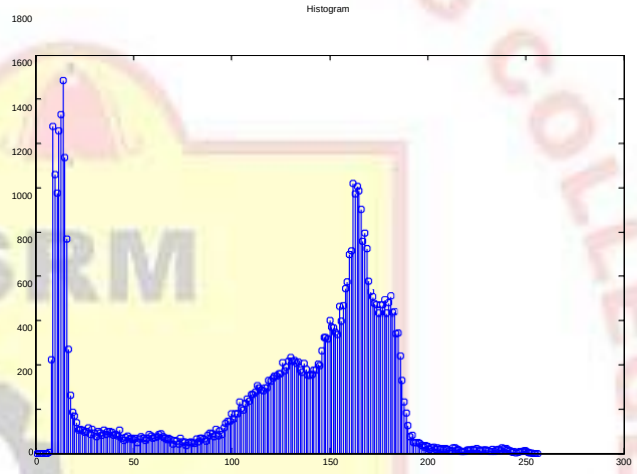
```
x=imread(‘cameraman.tif’);
imhist(x);
```

%histogram equalization

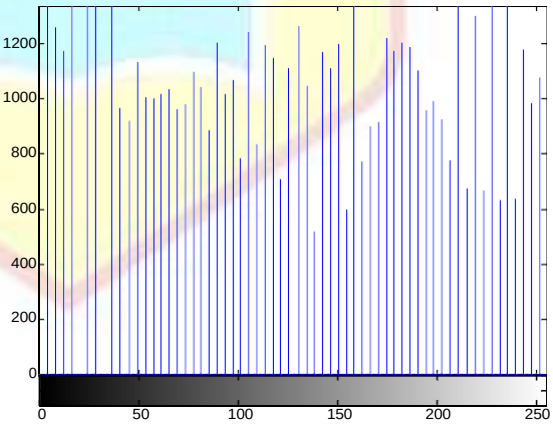
```
I=imread('cameraman.tif');
a=histeq(I);
imshow(a);
figure,imhist(a)
```

OUTPUT

Histogram



Histogram Equalization



RESULT:

Thus the histogram and its equalization are performed using Matlab.

Ex. No.7 **IMAGE ENHANCEMENT BY SPATIAL FILTERING**

AIM:
To perform image enhancement by spatial filtering.

SOFTWARE
MATLAB

THEORY:

In Spatial filtering, the filtering operations are performed directly on the pixels of an image. There are two types of spatial filters, linear and non linear filters. The mechanics of spatial filtering is that it is the process which consists of simply 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 calculated using a predefined relationship. For linear spatial filtering, the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask. Smoothing filters are used for blurring and for noise reduction. The output of a smoothing linear spatial filter is simply the average of the pixels contained in the neighbourhood of the filter mask. These filters are called averaging or lowpass filters.

Order statistics filters are non-linear spatial filters whose response is based on ordering the pixels contained in the image area encompassed by the filter and then replacing the value of the center pixel with the value determined by the ranking result. Median filter replaces the value of a pixel with the median of the graylevels in the neighbourhood of the pixel.

PROGRAM

Average

```
i=imread('cameraman.tif');
imshow(i);
w=fspecial('average',[3 3]);
g=imfilter(i,w,'symmetric');
figure,imshow(g,[])
```

Gaussian

```
i=imread('cameraman.tif');
w=fspecial('gaussian',[3 3],0.5);
g=imfilter(i,w,'symmetric');
imshow(g,[])
```

Laplacian

```
i=imread('cameraman.tif');
```

```
w=fspecial('laplacian', 0.5);
g=imfilter(i,w,'symmetric');
imshow(g,[])
```

Sobel

```
i=imread('cameraman.tif');
w=fspecial('sobel');
g=imfilter(i,w,'symmetric');
imshow(g,[])
```

Non linear order statistic filter

```
i=imread('cameraman.tif');
h=ordfilt2(i,1,ones(3,3));
h1=ordfilt2(i,3*3,ones(3,3));
h2=ordfilt2(i,median(1:3*3),ones(3,3));
subplot(2,2,1)
imshow(i);
subplot(2,2,2)
imshow(h,[]);
subplot(2,2,3)
imshow(h1,[]);
subplot(2,2,4)
imshow(h2,[]);
```

Median Filter

```
g=imread('cameraman.tif');
m=medfilt2(g,[3 3]);
imshow(m,[]);
```

OUTPUT

Original Image

Average

Gaussian

Laplacian

Sobel

Median filter

Original Image

Order statistic filter (1)

Order statistic filter (2)

Order statistic filter (3)

RESULT

Thus spatial filtering is performed on image using Matlab.

Ex. No.8a FREQUENCY DOMAIN FILTERS FROM SPATIAL DOMAIN

AIM

To obtain frequency domain filters from spatial domain.

SOFTWARE USED

MATLAB

THEORY

Filtering in the frequency domain consists of modifying the Fourier transform of an image and then computing the inverse transform to obtain the processed result.

Thus a given image of digital f(x,y) of size M x N the basic filtering equation in which interested is in the form

$$\mathbf{g(x,y)=F^{-1}[H(u,v)\ F(u,v)]}$$

F^{-1} – IDFT, F(u,v) is the DFT of input image f(x,y)

H(u,v) – Filter function

g(x,y) – Filtered (output) image

PROGRAM

Average

```
f=imread('cameraman.tif');
h=fspecial('average',[5 5]);
Fs=size(f);
F=fft2(f);
H=freqz2(h,Fs(1),Fs(2));
G=F.*H;
g=ifft2(G);
imshow(real(g),[]);
figure,imshow(abs(H));
```

Gaussian

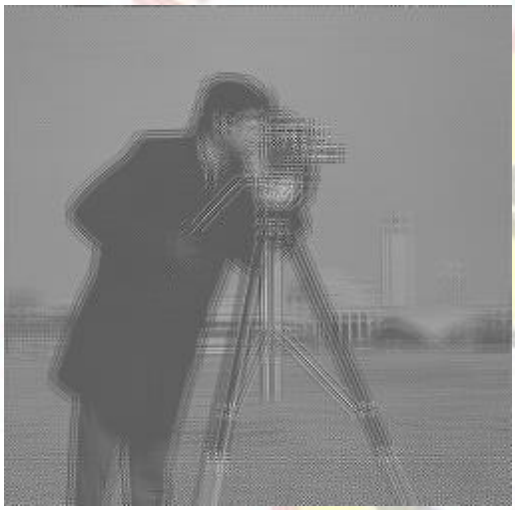
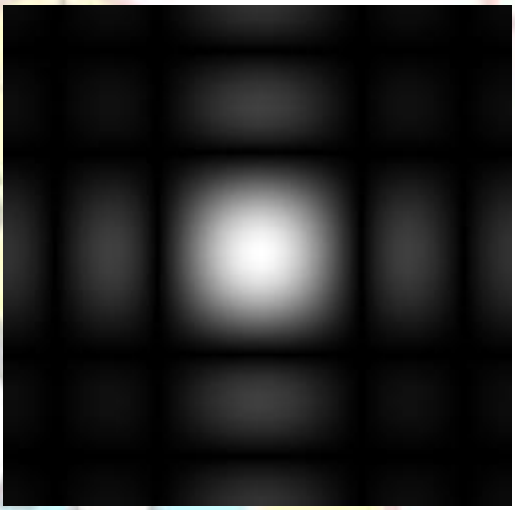
```
f=imread('cameraman.tif');
h=fspecial('gaussian',[3 3],2);
Fs=size(f);
F=fft2(f);
H=freqz2(h,Fs(1),Fs(2));
G=F.*H;
g=ifft2(G);
imshow(real(g),[]);
figure,imshow(abs(H));
```

Sobel

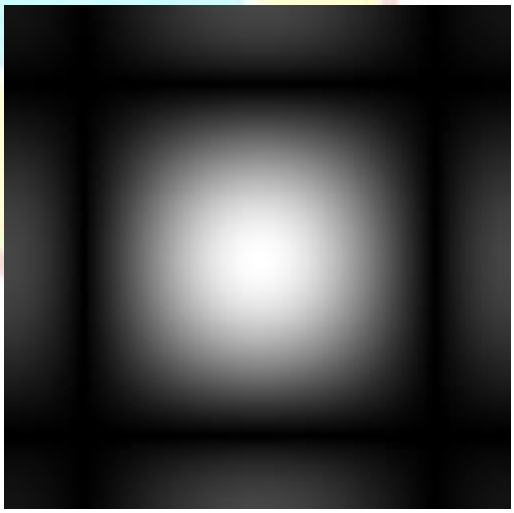
```
f=imread('cameraman.tif');
h=fspecial('sobel');
Fs=size(f);
F=fft2(f);
H=freqz2(h,Fs(1),Fs(2));
G=F.*H;
g=ifft2(G);
imshow(real(g),[]);
figure,imshow(abs(H));
```

OUTPUT

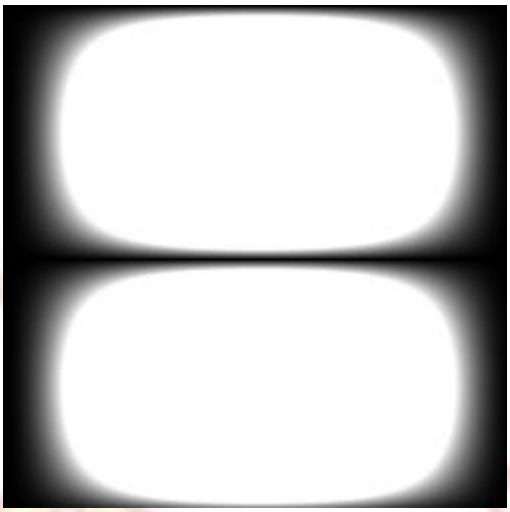
Average

Gaussian

Sobel



RESULT

Thus frequency domain filters are obtained from spatial domain.

Ex. No.8b GENERATING FILTERS DIRECTLY IN THE FREQUENCY DOMAIN

AIM

To generate filters directly in the frequency domain using Matlab.

SOFTWARE USED

MATLAB

THEORY

Frequency filtering is based on the Fourier Transform. There are basically three different kinds of filters: lowpass, highpass and bandpass filters.

A low-pass filter attenuates high frequencies and retains low frequencies unchanged. The result in the spatial domain is equivalent to that of a smoothing filter. A highpass filter, on the other hand, yields edge enhancement or edge detection in the spatial domain, because edges contain many high frequencies. Areas of rather constant graylevel consist of mainly low frequencies and are therefore suppressed. A bandpass attenuates very low and very high frequencies, but retains a middle range band of frequencies. Bandpass filtering can be used to enhance edges (suppressing low frequencies) while reducing the noise at the same time (attenuating high frequencies).

Ideal Lowpass Filter

It is a filter that cuts off all high frequency components of the Fourier Transform that are at a distance greater than a specified distance D_0 from the origin of the transform. The transfer function is

$$H(u,v) = \begin{cases} 1 \text{ if } D(u,v) \le D0 \\ 0 \text{ if } D(u,v) > D0 \end{cases}$$

Where $D(u,v) = [(u - M/2)^2 + (v - N/2^2)]^{1/2}$

Butterworth Lowpass Filter

The transfer function of a butterworth lowpass filter of order n, and with cutoff frequency at a distance D_0 from the origin is

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

Gaussian Lowpass Filter

The form of Gaussian filters in two dimensions is given by

$$H(u,v) = e^{-D^2(u,v)/2\sigma^2}$$

The transfer function of high pass filter is obtained by $H_{hp}(u,v)=1-H_{lp}(u,v)$

PROGRAM

Butterworth Low Pass:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=1/(1+((d(i,j))/rad).^(2*N));
   end
end
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(2,2,1);
imshow(img);
title('Original');
subplot(2,2,2);
imshow(fin1);
title('After LPF');
subplot(2,2,3);
surf(h);
title('LPF in 3D');
subplot(2,2,4);
imshow(h);
title('LPF as Image');
```

Butterworth High Pass:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
```

```
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=1/(1+(rad/d(i,j)).^(2*N));
   end
end
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(2,2,1);
imshow(img);
title('Original');
subplot(2,2,2);
imshow(fin1);
title('After HPF');
subplot(2,2,3);
surf(h);
title('HPF in 3D');
subplot(2,2,4);
imshow(h);
title('HPF as Image');
```

Gaussian Low Pass Filter:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=exp(-(d(i,j).^2)/(2*((rad).^2)));
   end
end
```

```
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(2,2,1);
imshow(img);
title('Original');
subplot(2,2,2);
imshow(fin1);
title('After Gaussian LPF');
subplot(2,2,3);
surf(h);
title('Gaussian LPF in 3D');
subplot(2,2,4);
imshow(h);
title('Gaussian LPF as Image');
```

Gaussian High Pass Filter:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=1-exp(-(d(i,j).^2)/(2*((rad).^2)));
   end
end
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(221);
imshow(img);
title('Original');
subplot(2,2,2);
```

```
imshow(fin1);
title('After Gaussian HPF');
subplot(2,2,3);
surf(h);
title('Gaussian HPF in 3D');
subplot(2,2,4);
imshow(h);
title('Gaussian HPF as Image');
```

Ideal Low Pass Filter:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=double(d(i,j)<=rad);
   end
end
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(2,2,1);
imshow(img);
title('Original');
subplot(2,2,2);
imshow(fin1);
title('After LPF');
subplot(2,2,3);
surf(h);
title('LPF in 3D');
subplot(2,2,4);
imshow(h);
title('LPF as Image');
```

Ideal High Pass Filter:

```
clear;
clc;
img=imread('Coins.png');
[X,Y]=size(img);
N=input('Order of Filter=');
x=ceil(X/2);
y=ceil(Y/2);
rad=26;
for i=1:X
   for j=1:Y
      d(i,j)=sqrt((i-x).^2+(j-y).^2);
      h(i,j)=double(d(i,j)>rad);
   end
end
fft1=fftshift(fft2(img));
fil=h.*fft1;
fin=ifft2(fil);
fin1=uint8(fin);
subplot(2,2,1);
imshow(img);
title('Original');
subplot(2,2,2);
imshow(fin1);
title('After HPF');
subplot(2,2,3);
surf(h);
title('HPF in 3D');
subplot(2,2,4);
imshow(h);
title('HPF as Image');
```

RESULT:

Thus filters are directly generated in the frequency domain.

Ex. No.9a **EDGE DETECTION**

AIM:

To detect edges in the image.

SOFTWARE USED

MATLAB

THEORY

The Sobel and prewitt operators are used in image processing and computer vision, particularly within edge detection algorithms where they create an image emphasizing edges. They are a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel and prewitt operatorsis either the corresponding gradient vector or the norm of this vector. The Sobel and prewitt operators are based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions.

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. Canny uses the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

The Roberts cross operator is used in image processing and computer vision for edge detection. It is a differential operator. The gradient of an image is approximated through discrete differentiation which is achieved by computing the sum of the squares of the differences between diagonally adjacent pixels.

PROGRAM

```
a=imread('house.jpg');
imshow(a);
f=rgb2gray(a);
figure,imshow(f);
[g,t]=edge(f,'sobel','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,t]=edge(f,'sobel','horizontal');
subplot(3,1,2);
imshow(g);
[g,t]=edge(f,'sobel','both');
subplot(3,1,3);
```

```
imshow(g);
[g,t]=edge(f,'prewitt','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,t]=edge(f,'prewitt','horizontal');
subplot(3,1,2);
imshow(g);
[g,t]=edge(f,'prewitt','both');
subplot(3,1,3);
imshow(g);
[g,t]=edge(f,'roberts','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,t]=edge(f,'roberts','horizontal');
subplot(3,1,2);
imshow(g);
[g,t]=edge(f,'roberts','both');
subplot(3,1,3);
imshow(g);
[g,t]=edge(f,'canny','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,t]=edge(f,'canny','horizontal');
subplot(3,1,2);
imshow(g);
[g,t]=edge(f,'canny','both');
subplot(3,1,3);
imshow(g);
```

OUTPUT

Original Image

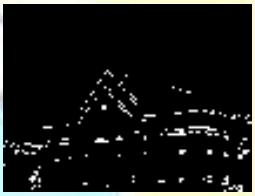
Original Image

Sobel- Vertical

Prewitt- Vertical

Sobel-Horizontal

Prewitt- Horizontal



Sobel- Horizontal and Vertical

Prewitt-Horizontal and Vertical

Roberts- Vertical

Canny- Vertical

Roberts- Horizontal

Canny - Horizontal

Roberts- Horizontal and Vertical

Canny- Horizontal and Vertical

RESULT

Thus the edges are detected using Matlab.

Ex. No.9b **LINE DETECTION**

AIM

To detect lines in the images using Matlab.

SOFTWARE USED

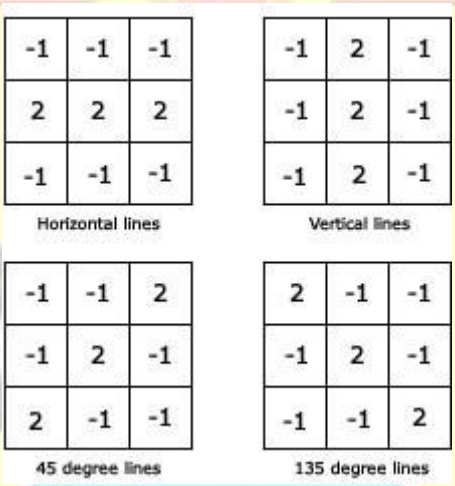
MATLAB

THEORY

Horizontal mask will result with maximum response when a line is passed through the middle row of the mask with a constant background. Let R1, R2, R3, R4 denote the response of the horizontal, +45 degree, vertical and +45 degree masks respectively.

|Ri|>|Rj|

For detecting all lines in an image in the direction defined by a given mask, we simply run the mask through the image and threshold the absolute value of the result.



PROGRAM

Horizontal lines

```
a=imread('line.jpg');
f=rgb2gray(a);
imshow(f);
w=[-1,-1,-1;2,2,2;-1,-1,-1]
g=abs(imfilter(double(f),w));
T=300;
g=g>=T;
figure,imshow(g);
```

Vertical lines

```
a=imread('line.jpg');
f=rgb2gray(a);
imshow(f);
w=[-1,2,-1;-1,2,-1;-1,2,-1]
g=abs(imfilter(double(f),w));
T=300;
g=g>=T;
figure,imshow(g);
```

45 degree lines

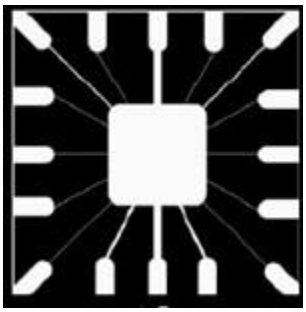
```
a=imread('line.jpg');
f=rgb2gray(a);
imshow(f);
w=[-1,-1,2;-1,2,-1;2,-1,-1]
g=abs(imfilter(double(f),w));
T=300;
g=g>=T;
figure,imshow(g);
```

135 degree lines

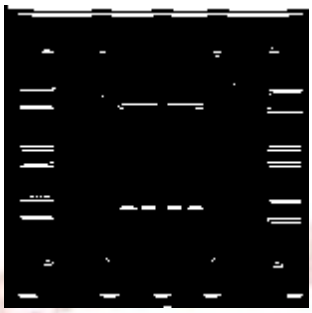
```
a=imread('line.jpg');
f=rgb2gray(a);
imshow(f);
w=[2,-1,-1;-1,2,-1;-1,-1,2]
g=abs(imfilter(double(f),w));
T=300;
g=g>=T;
figure,imshow(g);
```

OUTPUT

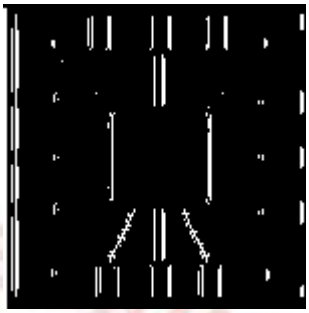
Original Image



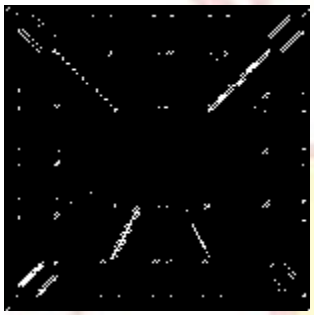
Horizontal



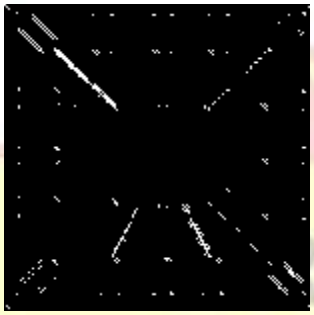
Vertical



45 degree



135 degree



RESULT

Thus lines are detected in images using Matlab.

Ex. No.9c **POINT DETECTION**

AIM
To detect points in an image.

SOTWARE USED
MATLAB

THEORY

Points are detected at those pixels in the subsequent filtered image that are above a set threshold. Point detection can be achieved simply using the mask below: $R \geq T$

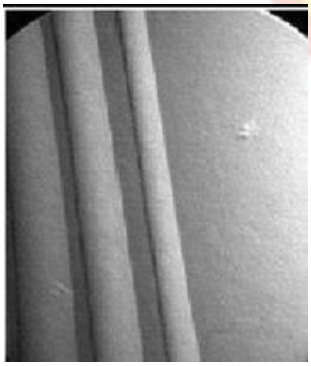
–1	–1	–1
–1	8	–1
–1	–1	–1

PROGRAM

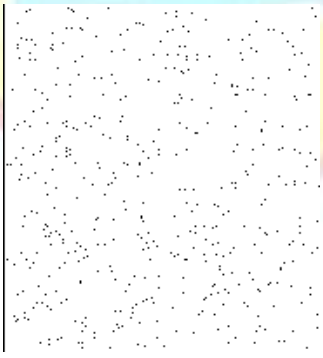
```
a=imread('point.jpg');
f=rgb2gray(a);
imshow(f);
w=[-1,-1,-1;-1,8,-1;-1,-1,-1];
g=abs(imfilter(double(f),w));
imshow(g);
```

OUTPUT

Input image



Point detection



RESULT
Thus the points are detected in the image.

Ex. No.10 **MORPHOLOGICAL OPERATIONS**

AIM

To perform morphological operations on image using Matlab**.**

SOFTWARE USED

MATLAB

THEORY

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image. The most basic morphological operations are dilation and erosion.

Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image.

Opening and closing are derived from the fundamental operations of erosion and dilation. The basic effect of an opening is somewhat like erosion in that it tends to remove some of the foreground (bright) pixels from the edges of regions of foreground pixels. Closing is similar to dilation in that it tends to enlarge the boundaries of foreground (bright) regions in an image.

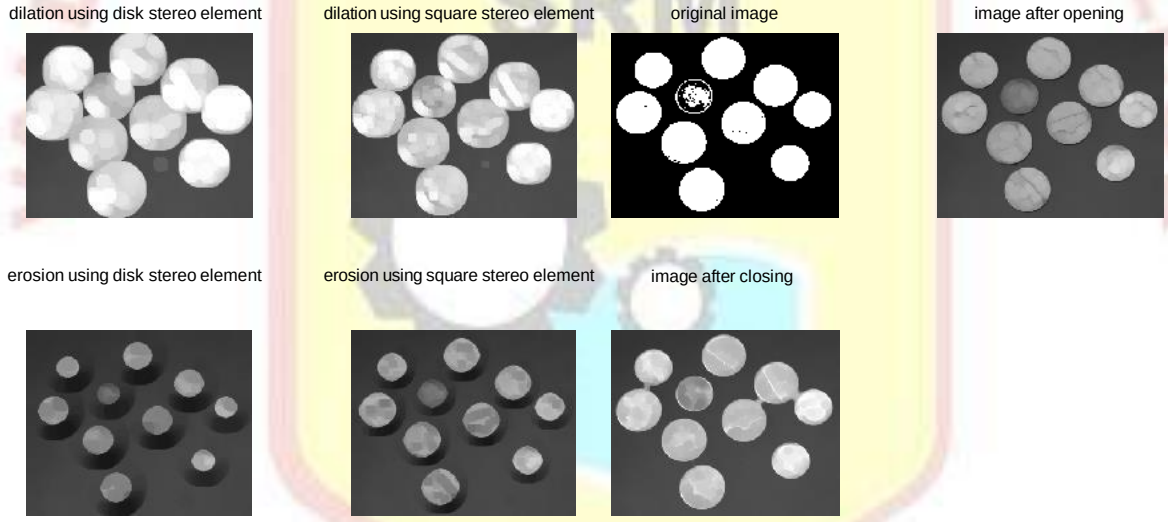
PROGRAM

```
imread('coins.png');
b=strel('disk',10);
c=imdilate(a,b);
figure;
subplot(2,2,1);
imshow(c);
title('dilation using disk stereo element');
b=strel('square',10);
c=imdilate(a,b);
subplot(2,2,2);
imshow(c);
title('dilation using square stereo element');
%Image erosion
%a=imread('coins.png');
b=strel('disk',10);
```

```
c=imerode(a,b);
subplot(2,2,3);
imshow(c);
title('erosion using disk stereo element');
b=strel('square',10);
c=imerode(a,b);
subplot(2,2,4);
imshow(c);
title('erosion using square stereo element');
%opening
bw=im2bw(a);
figure;
subplot(2,2,1);
imshow(bw);
title('original image');
b=strel('disk',5);
c=imopen(a,b);
subplot(2,2,2);
imshow(c);
title('image after opening');
%closing
b=strel('disk',5);
c=imclose(a,b);
subplot(2,2,3);
imshow(c);
title('image after closing');
%morphological operations
a=imread('rice.png');
b=im2bw(a);
c=bwmorph(b,'remove');
figure;
subplot(2,3,1);
imshow(c);
title('image remove');
c=bwmorph(b,'clean');
subplot(2,3,2);
imshow(c);
title('image clean');
c=bwmorph(b,'shrink');
subplot(2,3,3);
```

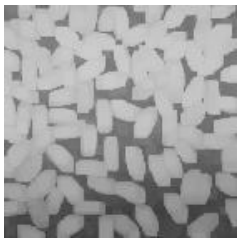
```
imshow(c);
title('image shrink');
c=bwmorph(b,'fill');
subplot(2,3,4);
imshow(c);
title('image fill');
c=bwmorph(b,'thin');
subplot(2,3,5);
imshow(c);
title('image thin');
c=bwmorph(b,'thick');
subplot(2,3,6);
imshow(c);
title('image thick');
```

OUTPUT



dilation using disk stereo element

dilation using square stereo element



erosion using disk stereo element

erosion using square stereo element

original image

image after opening

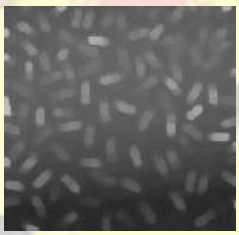


image after closing

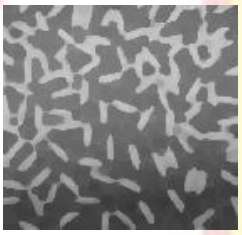


image remove

image clean

RESULT:

Thus morphological operations are performed in images using Matlab.

Ex. No.11 **REGION BASED SEGMENTATION**

AIM

To perform region based segmentation of image using Matlab.

SOFTWARE USED

MATLAB

THEORY

Thresholding is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images. Thresholding is used to extract an object from its background by assigning an intensity value T (threshold) for each pixel such that each pixel is either classified as an object point or a background point.

In general **T = T[x, y, p(x, y), f (x, y)]**

If T is a function of f(x, y) only – Global thresholding

If T is a function of both f(x, y) and local properties p(x, y) – Local thresholding

If T depends on the coordinates (x, y) – Dynamic/adaptive thresholding

Image thresholding classifies pixels into two categories. Those to which some property measured from the image falls below a threshold and those at which the property equals or exceeds a threshold.

In fixed (or global) thresholding, the threshold value is held constant throughout the image. Determine a single threshold value by treating each pixel independently of its neighborhood.

Global Thresholding is done when the modes of histogram can be clearly distinguished. Each mode represents either the background or an object

PROGRAM

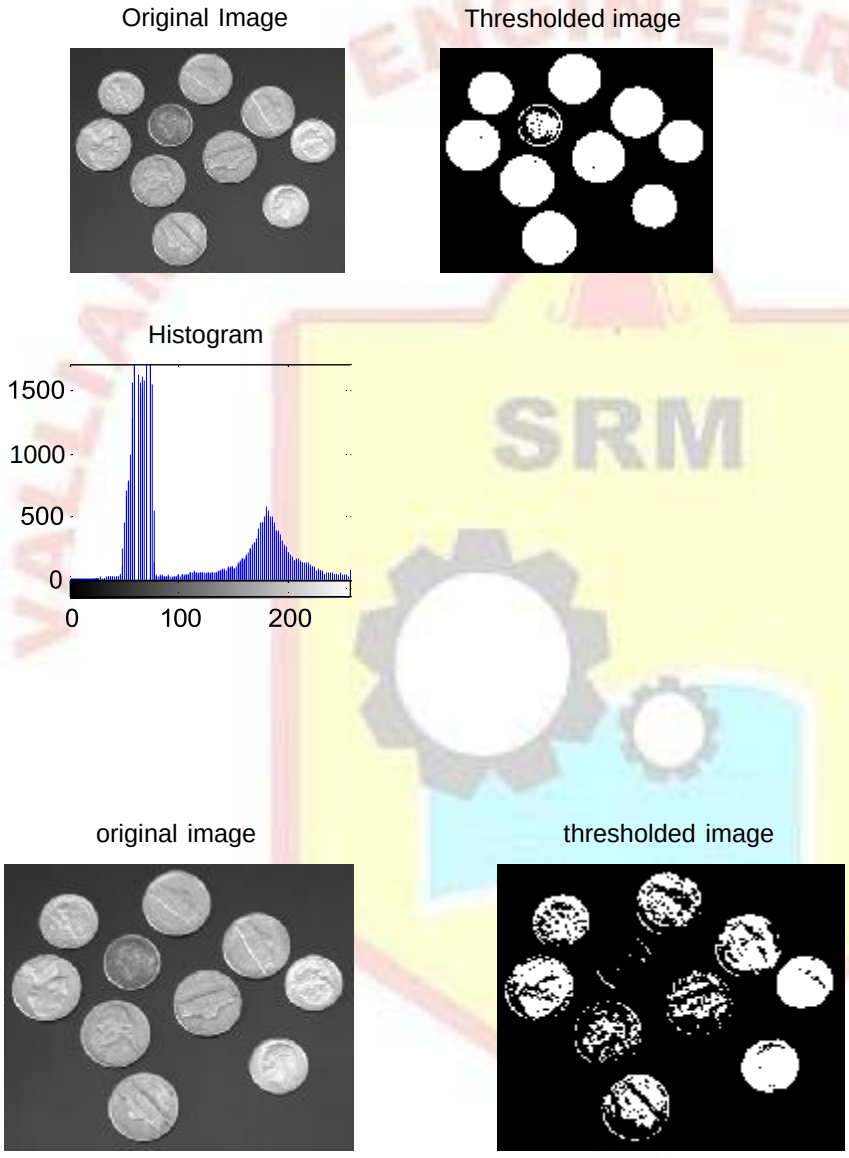
```
a=imread('coins.png');
subplot(2,2,1);
imshow(a);
level=graythresh(a);
b=im2bw(a,level);
subplot(2,2,2);
imshow(b);
subplot(2,2,3);
imhist(a);
```

Code2:

```
a=imread('coins.png');
a1=a>180;
```

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

OUTPUT



RESULT

Thus region based segmentation carried out by using thresholding technique in Matlab.

Ex. No.12 ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS

AIM

To perform analysis of images with different color models.

SOFTWARE USED

MATLAB

THEORY

An RGB colour image is an M x N x3 array of colour pixels, where each colour pixel is a triplet corresponding to the red, green and blue components of an RGB image at a specific spatial location. An RGB image is viewed as a 'stack' of three gray scale images that when fed into the red, green and blue inputs of a colour monitor, produce a colour image on the screen. The three images forming an RGB colour image are referred to as the red, green and blue component images.

A color map is an *m*-by-3 matrix of real numbers between 0.0 and 1.0. Each row is an RGB vector that defines one color. The *k*th row of the color map defines the kth color, where map (k,:) = [r(k) g(k) b(k)]) specifies the intensity of red, green, and blue.

ycbcrmap is an M-by-3 matrix that contains the YCbCr luminance (*Y*) and chrominance (Cb and Cr) color values as columns. In the NTSC color space, the luminance is the grayscale signal used to display pictures on monochrome (black and white) televisions. The other components carry the hue and saturation information. In HSV, H represents the hue component, S the saturation component, V the brightness component of an image.

PROGRAM

```
a=imread('peppers.png');
a(:,:,2)=0;
a(:,:,3)=0;
imshow(a);
b=imread('peppers.png');
b(:,:,1)=0;
b(:,:,3)=0;
figure,imshow(b);
c=imread('peppers.png');
c(:,:,1)=0;
c(:,:,2)=0;
figure,imshow(c);
```

```
a=imread('peppers.png');
R=a(:,:,1);
G=a(:,:,2);
B=a(:,:,3);
new=cat(3,R,G,B);
figure,imshow(new);
```

Colormap

```
a=imread('cameraman.tif');
subplot(2,2,1)
imshow(a,copper);
subplot(2,2,2)
rgbplot(copper);
figure,subplot(2,2,1)
imshow(a,jet(150));
subplot(2,2,2)
rgbplot(jet(150));
figure,subplot(2,2,1)
imshow(a,summer);
subplot(2,2,2)
rgbplot(summer);
```

Conversion between color spaces

```
a=imread('peppers.png');
b=rgb2ntsc(a);
subplot(2,2,1)
imshow(b);
c=rgb2ycbcr(a);
subplot(2,2,2)
imshow(c);
d=rgb2hsv(a);
subplot(2,2,3)
imshow(d);
e=imcomplement(a);
subplot(2,2,4)
imshow(e);
```

OUTPUT

Red component

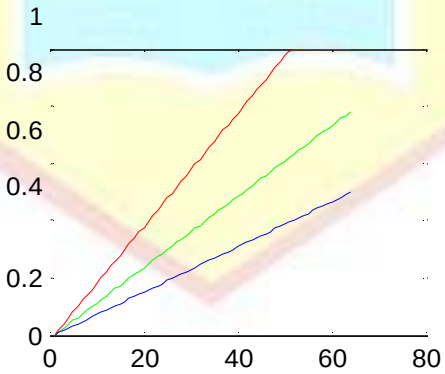
Green component

Blue Component

Concatenation

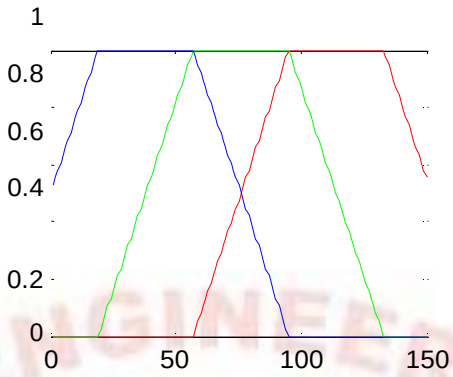
copper

rgbplot



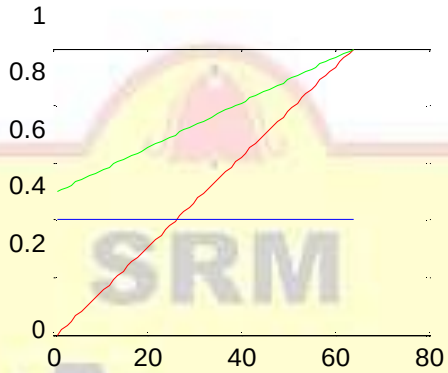
jet 150

rgbplot

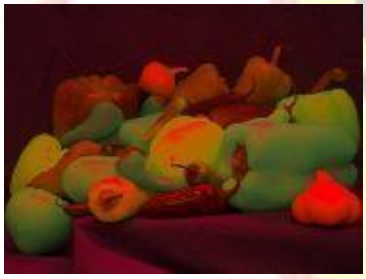


summer

rgbplot

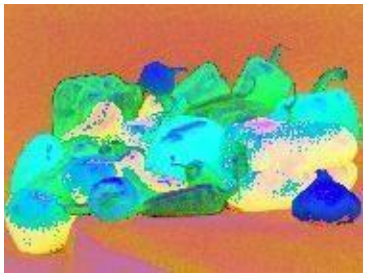


ntsc

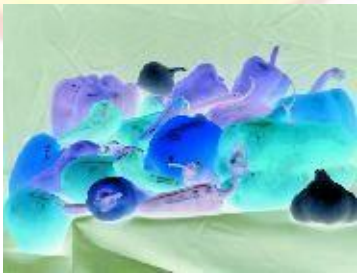


ycbcr

hsv



complement



RESULT

Thus the analysis of images with different color models is done using Matlab.