EX. NO: 1 **DATE:**

ACQUIRE AND DISPLAY OF AN IMAGE, NEGATIVE OF AN IMAGE

AIM

To acquire an image and display it alongside its negative version.

PROCEDURE

Step 1: Acquiring the image

- imread('your_image.jpg') loads an image from the specified file. Replace 'your_image.jpg' with the path to your own image file.

Step 2: Displaying the original image

- imshow(image) is used to display the original image. It is shown in the first subplot for comparison.

Step 3: Convert to grayscale (if necessary)

- If the image is in RGB format (color), rgb2gray(image) converts it to grayscale. If the image is already grayscale, it remains unchanged.

Step 4: Negative of the image

- The negative of the image is calculated by subtracting the pixel values from 255.
- The formula is:
  negativeImage = 255 - grayImage. OR imcomplement

Step 5: Displaying the negative image

- imshow(negativeImage) shows the negative image in the second subplot.

PROGRAM

Display color Image

```
I=imread("images.jpg");
subplot(2,2,1);
imshow(I);
subimage(I);
title('Color Image');
```

Negative of An Image

```
c=imcomplement(I);
subplot(2,2,2);
imshow(c);
subimage(c);
title('Complement of color Image');
```

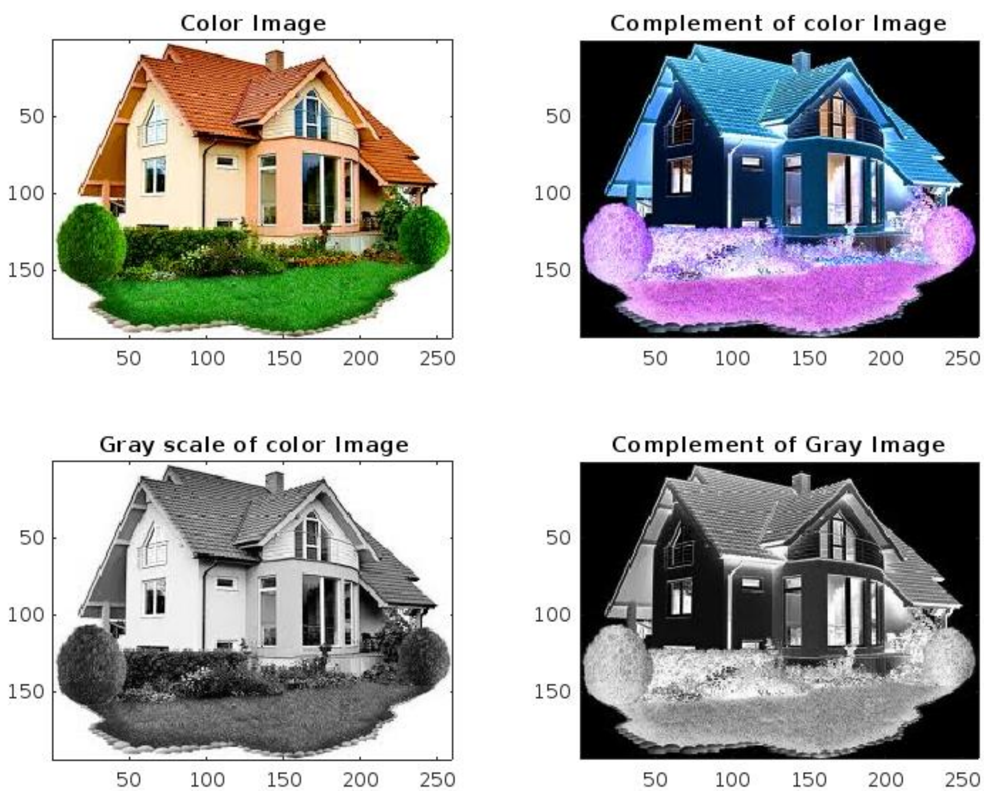
Gray Image

```
r=rgb2gray(I);
subplot(2,2,3);
imshow(r);
subimage(r);
title('Gray scale of color Image');
```

Complement of Gray Image

```
b=imcomplement(r);
subplot(2,2,4);
imshow(b);
subimage(b);
title('Complement of Gray Image');
```

OUTPUT



RESULT

Thus, the acquisition, display, and negative transformation of an image are effectively executed using MATLAB.

EX. NO: 2 **DATE:**

IMPLEMENTATION OF RELATIONSHIPS BETWEEN PIXELS

AIM

To analyze and model the relationships between pixels within an image.

PROCEDURE

Step 1: Create the Matrix : A **magic matrix** of size 5x5 is generated using magic(5), where the sum of elements in any row, column, or diagonal is the same

Step 2: Get User Input for Row and Column Selection

Step 3: Identify 4-Point Neighbors

Step 4: Identify 8-Point Neighbors

Step 5: Identify Diagonal Neighbors

Step 6: Displaying the relationships between pixels.

PROGRAM

To find Neighbour of a given Pixel

```
a=magic(5);
disp(‘a=’);
disp(a);
b=input('Enter the row < size of the Matrix');
c=input(' Enter the Column < size of matrix'); disp(‘Element’);
disp(a(b,c));
```

4 Point Neighbour

```
N4=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1)];
disp(‘N4=’);
disp(N4);
```

8 Point Neighbour

```
N8=[a(b+1,c), a(b-1,c), a(b,c+1), a(b,c-1), a(b+1,c+1), a(b+1,c-1), a(b-1,c-1), a(b-1,c+1)];
disp(‘N8=’);
disp(N8);
```

Diagonal Neighbour

```
ND=[ a(b+1,c+1), a(b+1,c-1), a(b-1,c-1), a(b-1,c+1)];
disp(‘ND=’);
disp(ND);
```

OUTPUT

```
a=
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

Enter the row < size of the Matrix
2
 Enter the Column < size of matrix
3
Element
     7

N4=
    13     1    14     5

N8=
    13     1    14     5    20     6    24     8

ND=
    20     6    24     8
```

RESULT

Thus the implementation of relationships between pixels is done using Matlab.

EX. NO: 3 **DATE:**

ANALYSIS OF IMAGES WITH DIFFERENT COLOR MODELS

AIM

To perform analysis of images with different color models.

PROCEDURE

Step 1: Load the Image (peppers.png)

- Load an image into MATLAB. The function imread('peppers.png') loads the image and stores it in a variable.

Step 2: Modify the Image Channels

- Images in MATLAB are stored in a 3D matrix where the third dimension corresponds to the color channels (Red, Green, Blue).
- Modify the color channels by accessing the matrix at specific indices for each color.
  - To set the green channel to zero: a(:,:,2)=0; — This means you zero out all green values in the image matrix a.
  - To set the blue channel to zero: a(:,:,3)=0; — This removes all blue values in the image matrix a.
  - Similarly, for other images (b and c), different channels are removed.

Step 3: Displaying the Image with Different Color channels.

Step 4: Convert the image to NTSC, YCbCr, HSV,Complement color space

Step 5: Display the Modified Images

PROGRAM

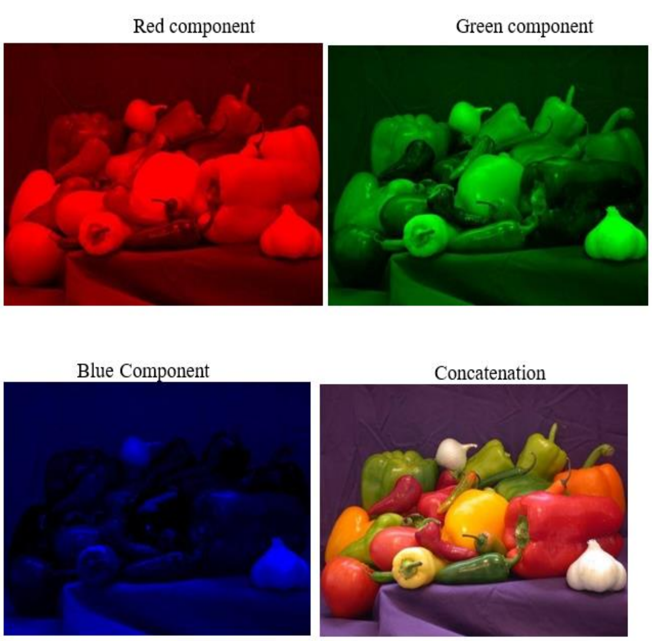
```
a=imread('peppers.png');
a(:,:,2)=0;
a(:,:,3)=0;
imshow(a);

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);
```

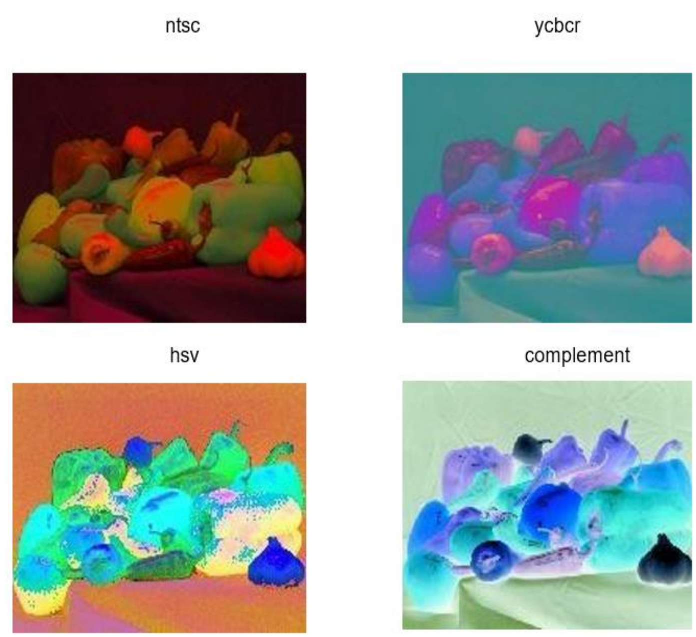
```
B=a(:,:,3);
new=cat(3,R,G,B);
figure,imshow(new);
```

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





RESULT

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

EX. NO: 4 **DATE:**

IMPLEMENTATION OF TRANSFORMATIONS OF AN IMAGE

AIM

To enhance and analyze the quality of an image through various transformation techniques.

PROCEDURE

Step 1: Load the image

- The image pexels-pixabay-56866.jpg is loaded using imread() and stored in the variable I.

Step 2: Display the original image

- The original image is shown in the first subplot using imshow(I).

Step 3: Scale (Resize) the image

- The user is prompted to enter a scaling factor using input().
- The imresize() function is used to resize the image I by the scaling factor s, and the result is stored in j.
- The scaled image j is displayed in the second subplot.

Step 4: Rotate the image

- The image j is rotated by 60 degrees using imrotate(j, 60) and stored in K. The rotated image is displayed in the third subplot.
- The image j is rotated again by 45 degrees using imrotate(j, 45) and stored in R. The rotated image is displayed in the fourth subplot.

Step 5: Resize the Image Using Bilinear, Nearest Neighbor, Bicubic Interpolation

Step 6: Display the Modified Images

PROGRAM

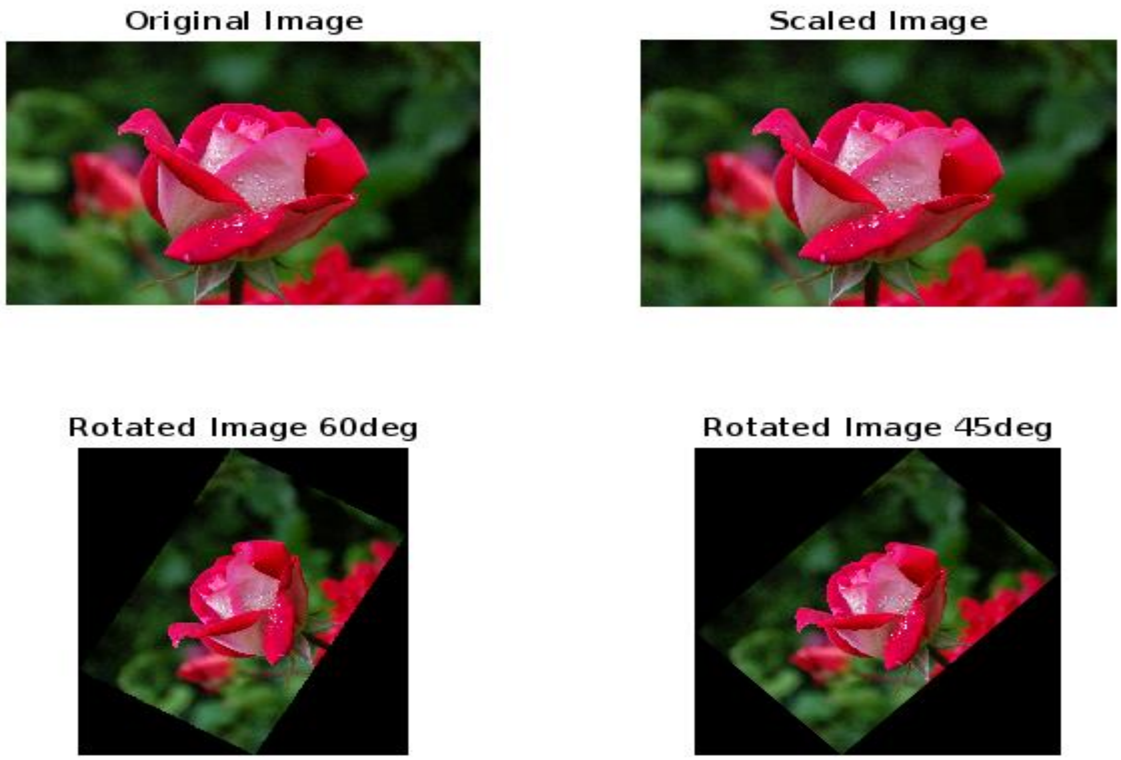
Scaling (Resize)

```
I=imread("pexels-pixabay-56866.jpg");
subplot(2,2,1);
imshow(I);
title('Original Image');

s=input("Enter Scaling Factor");
j=imresize(I,s);
subplot(2,2,2);
imshow(j);
title('Scaled Image');
```

Rotation

```
K=imrotate(j,60);
subplot(2,2,3);
imshow(K);
title('Rotated Image 60deg');
```

```
R=imrotate(j,45);
subplot(2,2,4);
imshow(R);
title('Rotated Image 45deg');
```



Display the color image and its Resized images by different methods

Display the color image

```
I=imread("Simple-House-step-by-step-drawing-tutorial-step-10.jpg");
figure,
subplot(2,2,1);
imshow(I);
title('Original Image');
```

Display Resized image by Bilinear method

```
B=imresize(I,5);
subplot(2,2,2);
imshow(B);
title('Bilinear Image');
```

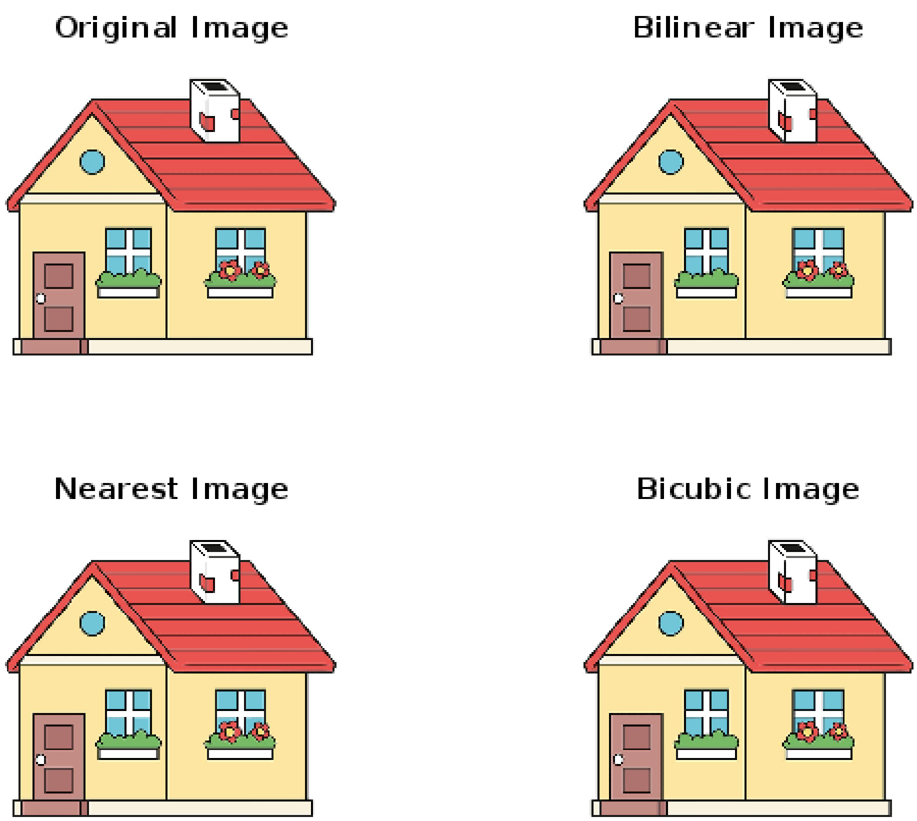
Display Resized image by Nearest method

```
C=imresize(I,5,'nearest');
subplot(2,2,3);
imshow(C);
title('Nearest Image');
```

Display Resized image by Bicubic method

```
D=imresize(I,5,'Bicubic');
subplot(2,2,4);
imshow(D);
title('Bicubic Image');
```

OUTPUT



RESULT

Thus the implementation of transformations of an image is done using Matlab.

EX.NO: 5 **DATE:**

HISTOGRAM PROCESSING AND BASIC THRESHOLDING FUNCTIONS

AIM

To implement histogram processing and basic thresholding functions.

PROCEDURE

Step 1: Image Reading: The image file "cameraman.png" is read using imread.

Step 2: Grayscale Conversion: If the image is in RGB format (i.e., has three color channels), it is converted to grayscale with rgb2gray. This simplifies the processing since the operations are designed for single-channel images.

Step 3: Displaying the Original Image and Its Histogram

Step 4: Equalization Process: histeq is applied to the image, which redistributes the intensity values to improve the global contrast.

Step 5: The resulting equalized image is displayed.

Step 6: Determine Intensity Range: The minimum and maximum intensity values of the original image are calculated. These values are divided by 255 (assuming an 8-bit image) to normalize the intensities between 0 and 1.

Step 7: Contrast Stretching: imadjust stretches the intensity range so that the darkest pixel becomes 0 and the brightest becomes 1, improving the contrast and image is displayed.

Threshold Setting:

- A fixed threshold value (100) is set.
- **Binary Conversion:** Each pixel in the original image is compared against the threshold. Pixels with intensity greater than 100 are set to `true` (or white) and others to `false` (or black), creating a binary image.

PROGRAM

Read the grayscale image

```
img = imread("cameraman.png");
if size(img,3) == 3
    img = rgb2gray(img);
end
```

Display Original Image and Histogram

```
figure; subplot(2,3,1);
imshow(img);
title('Original Image');
```

```
subplot(2,3,2); imhist(img);
title('Histogram of Original Image');
```

Histogram Equalization

```
equalized_img = histeq(img);
subplot(2,3,3); imshow(equalized_img);
title('Histogram Equalized Image');
subplot(2,3,4); imhist(equalized_img);
title('Histogram After Equalization');
```

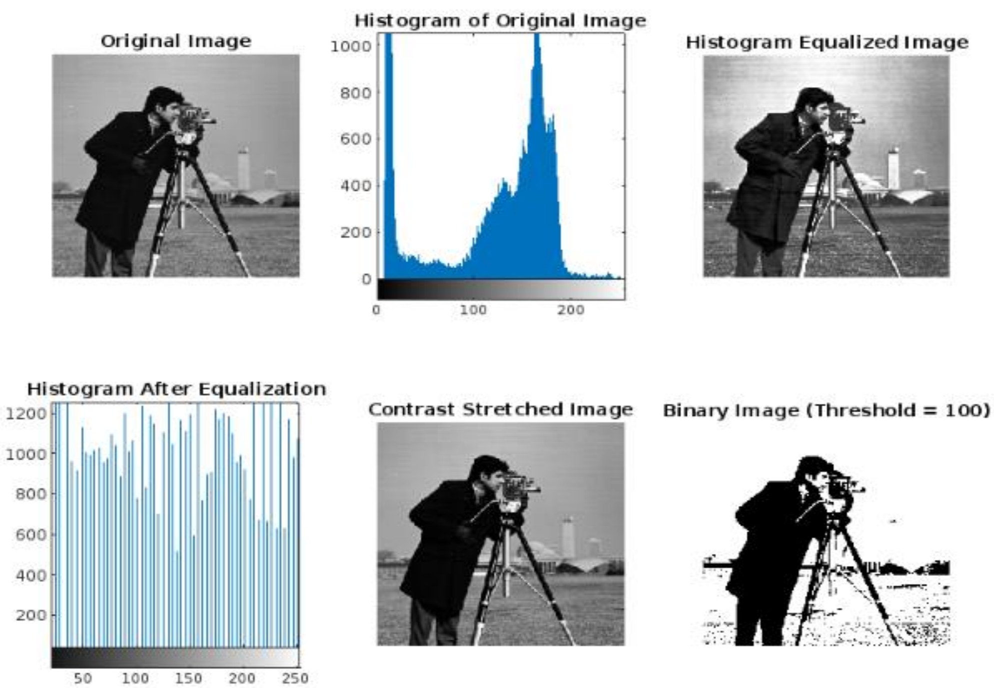
Contrast Stretching

```
min_val = double(min(img(:)));
max_val = double(max(img(:)));
stretched_img = imadjust(img, [min_val/255 max_val/255], []);
subplot(2,3,5);
imshow(stretched_img);
title('Contrast Stretched Image');
```

Basic Thresholding

```
threshold = 100; % Change threshold value as needed
binary_img = img > threshold;
subplot(2,3,6);
imshow(binary_img);
title(['Binary Image (Threshold = ' num2str(threshold) ')']);
```

OUTPUT



RESULT

Thus the MATLAB implementation for histogram processing and basic thresholding has been successfully executed.

EX.NO: 6 **DATE:**

COMPUTATION OF MEAN, STANDARD DEVIATION, CORRELATION COEFFICIENT OF THE GIVEN IMAGE

AIM

To compute of mean, standard deviation, correlation coefficient of the given image.

PROCEDURE

Step 1: Read and Display the Original Image

Step 2: Convert Image to Grayscale

Step 3: Crop the image to focus on a specific region of interest.

Step 4: Calculate and display statistical measures (mean and standard deviation) of the cropped image

Step 5: Generate checkerboard patterns and display them with different thresholds.

Step 6: Compute and display the correlation between the two checkerboard patterns.

PROGRAM

Read the original image

```
i = imread('cancer-cells.jpg');
figure(); subplot(2,2,1); imshow(i); title('Original Image');
```

Convert the image to grayscale

```
g = rgb2gray(i);
subplot(2,2,2); imshow(g); title('Gray Image');

rect = [100, 100, 200, 200];  % Adjust as needed
c = imcrop(g, rect);
subplot(2,2,3); imshow(c); title('Cropped Image');
```

Calculate mean and standard deviation of the cropped image

```
m = mean2(c); disp('m'); disp(m);
s = std2(c); disp('s'); disp(s);
```

Generate a checkerboard pattern

```
checkerboard1 = checkerboard(50, 5, 5);  % 50 pixels per square, 5x5 pattern
subplot(2,1,1); imshow(checkerboard1 > 0.8); title('Image1');

checkerboard2 = checkerboard(50, 5, 5);
subplot(2,1,2); imshow(checkerboard2 > 0.5); title('Image2');
```

Compute the correlation between the two checkerboard patterns

```
r = corr2(checkerboard1 > 0.8, checkerboard2 > 0.5);
disp('r'); disp(r);
```

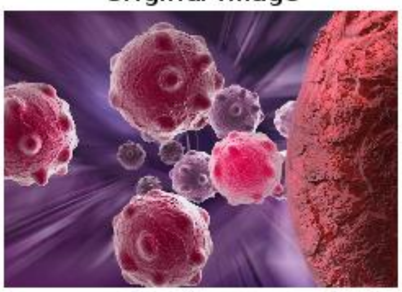
OUTPUT

m
87.7879

s
44.9134

r
0.5774

Original Image



Gray Image

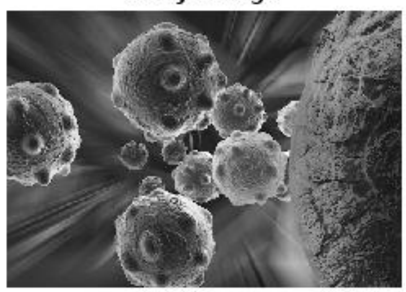
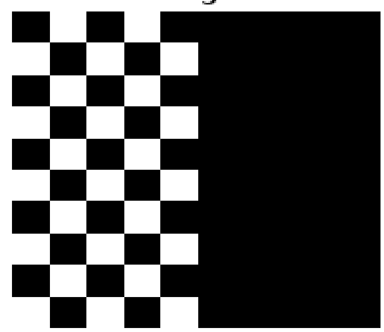
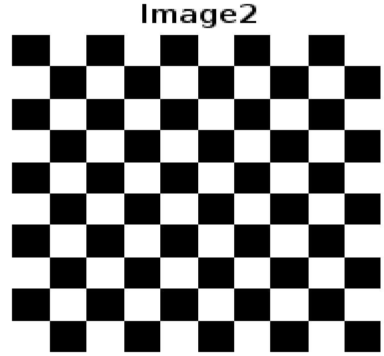


Image1



Cropped Image

Image2



RESULT

Thus compute of mean, standard deviation, correlation coefficient of the given image is performed using Matlab.

EX.NO: 7 **DATE:**

IMPLEMENTATION OF IMAGE ENHANCEMENT – SPATIAL FILTERING

AIM

To perform how different filters affect the image characteristics and how to choose the appropriate filter for specific tasks like noise reduction, edge detection, or image smoothing.

PROCEDURE

Step 1: Read and Display the Original Image
Step 2: Apply Spatial Filters

- Create a 3×3 averaging filter using fspecial('average', [3 3]).
- Create a 3×3 Gaussian filter with standard deviation 0.5 using fspecial('gaussian', [3 3], 0.5).
- Generate a Laplacian filter using fspecial('laplacian', 0.5).
- Generate a Sobel filter using fspecial('sobel').

Step 3: Apply Order-Statistic Filters

- Apply a **minimum filter** using ordfilt2(i, 1, ones(3,3)).
- Apply a **maximum filter** using ordfilt2(i, 9, ones(3,3)).
- Apply a **median filter** using medfilt2(i, [3 3])

Step 4: Apply Median Filter for Noise Removal

- Reload the original image using imread('cameraman.png').
- Apply a median filter using medfilt2(i, [3 3]).
- Display the final result.

PROGRAM

Averaging Filter

```
i = imread('cameraman.png');
imshow(i);
title('Original Image'); % Title for the original image
w = fspecial('average', [3 3]);
g = imfilter(i, w, 'symmetric');
figure, imshow(g, []);
title('Average Filtered Image'); % Title for the filtered image
```

Gaussian filter

```
i = imread('cameraman.png');
w = fspecial('gaussian', [3 3], 0.5);
g = imfilter(i, w, 'symmetric');
figure, imshow(g, []);
title('Gaussian Filtered Image'); % Title for the filtered image
```

Laplacian filter

```
i = imread('cameraman.png');
w = fspecial('laplacian', 0.5);
g = imfilter(i, w, 'symmetric');
figure, imshow(g, []);
title('Laplacian Filtered Image'); % Title for the filtered image
```

Sobel Filtering (Edge Detection):

```
i = imread('cameraman.png');
w = fspecial('sobel');
g = imfilter(i, w, 'symmetric');
figure, imshow(g, []);
title('Sobel Filtered Image (Edge Detection)'); % Title for the filtered image
```

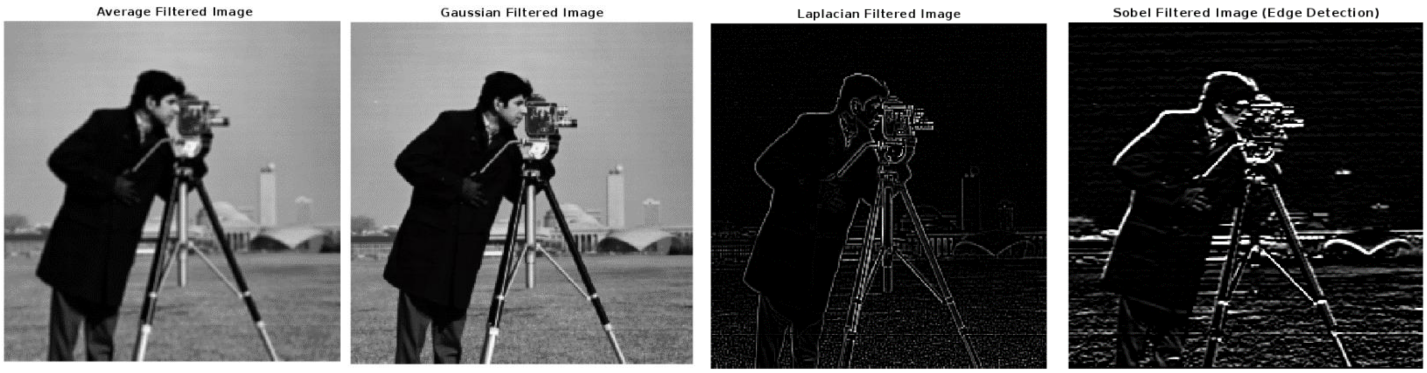
Order-Statistic Filtering:

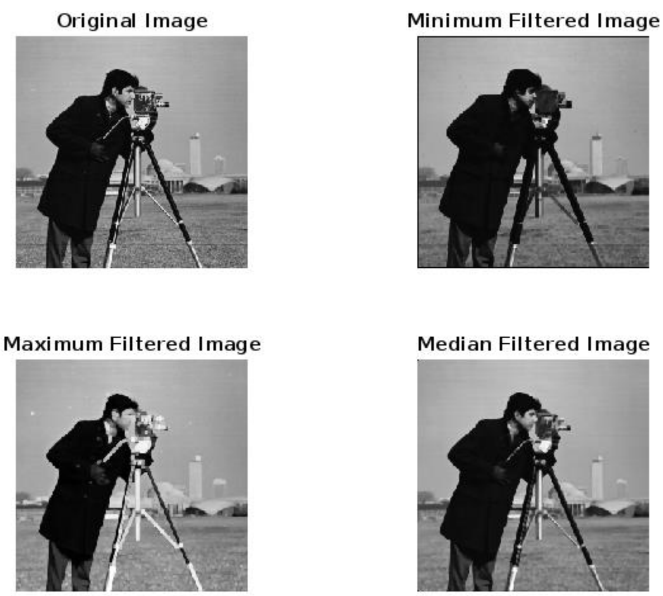
```
i = imread('cameraman.png');
h = ordfilt2(i, 1, ones(3, 3));
h1 = ordfilt2(i, 3*3, ones(3, 3));
h2 = ordfilt2(i, median(1:3*3), ones(3, 3));
subplot(2, 2, 1)
imshow(i);
title('Original Image'); % Title for the original image
subplot(2, 2, 2)
imshow(h, []);
title('Minimum Filtered Image'); % Title for the minimum filtered image
subplot(2, 2, 3)
imshow(h1, []);
title('Maximum Filtered Image'); % Title for the maximum filtered image
subplot(2, 2, 4)
imshow(h2, []);
title('Median Filtered Image'); % Title for the median filtered image
```

Median Filtering:

```
g = imread('cameraman.png');
m = medfilt2(g, [3 3]);
figure, imshow(m, []);
title('Median Filtered Image'); % Title for the filtered image
```

OUTPUT





RESULT

Thus the MATLAB implementation for spatial filtering has been successfully executed.

EX.NO: 8 **DATE:**

FREQUENCY DOMAIN FILTERS FROM SPATIAL DOMAIN

AIM

To obtain frequency domain filters from spatial domain.

PROCEDURE

Step 1: Load the input image using imread().

Step 2: Use fspecial() to generate different filters such as Average, Gaussian, and Sobel.

Step 3: Apply fft2() to convert the image into the frequency domain.

Step 4: Use freqz2() to compute the frequency response of the spatial filter.

Step 5: Multiply the Fourier-transformed image with the frequency response of the filter.

Step 6: Use ifft2() to convert the filtered image back to the spatial domain.

Step 7: Display the Results.

PROGRAM

```
f = imread("cameraman.png");
```

Average Filter

```
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), []);
title('Filtered Image with Average Filter');
figure, imshow(abs(H));
title('Frequency Response of Average Filter');
```

Gaussian Filter

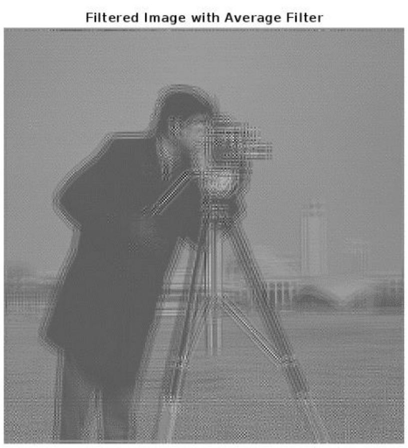
```
f = imread("cameraman.png");
h = fspecial('gaussian', [3 3], 2);
Fs = size(f);
F = fft2(f);
H = freqz2(h, Fs(1), Fs(2));
G = F .* H;
g = ifft2(G);
imshow(real(g), []);
```

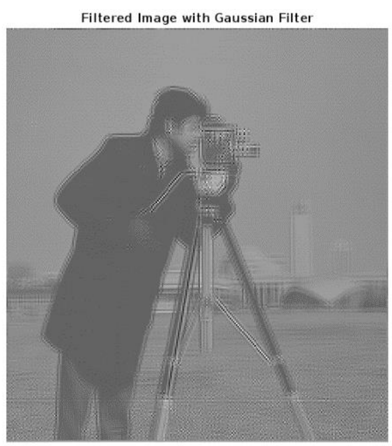
```
title('Filtered Image with Gaussian Filter');
figure, imshow(abs(H));
title('Frequency Response of Gaussian Filter');
```

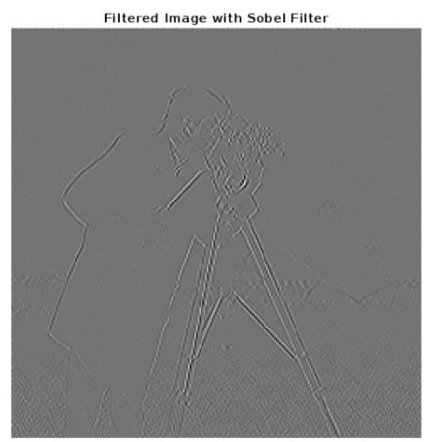
Sobel Filter

```
f = imread("cameraman.png");
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), []);
title('Filtered Image with Sobel Filter');
```

OUTPUT







RESULT

Thus, the spatial domain filters can be effectively converted into frequency domain filters, and filtering operations can be performed efficiently in the frequency domain.

EX.NO: 9 **DATE:**

IMAGE SEGMENTATION - EDGE DETECTION,LINE DETECTION AND POINT DETECTION

AIM

To perform edge detection on a grayscale image using different edge detection techniques in MATLAB.

PROCEDURE

EDGE DETECTION

Step 1: Read and Display the Original Image

Step 2: Apply Edge Detection Techniques

Use MATLAB's `edge()` function to detect edges in the image using different operators:

- **Sobel Operator:** Detects edges based on intensity differences.
- **Prewitt Operator:** Similar to Sobel but gives different weight to pixels.
- **Roberts Operator:** Detects edges with a diagonal response.
- **Canny Operator:** Uses a multi-stage algorithm for better edge detection.

LINE DETECTION

Step 1: Define Filters for Line Detection

Step 2: Apply Thresholding

POINT DETECTION

Step 1: Read and Convert Image to Grayscale

Step 2: Define the Laplacian Filter

Step 3: Apply the Laplacian Filter

Step 4: Display Results

PROGRAM

```
a=imread('images.jpg');
imshow(a);
f=rgb2gray(a);
figure,imshow(f);
```

Sobel Operator

```
[g,~]=edge(f,'sobel','vertical');
figure,subplot(3,1,1);
imshow(g);
```

```
[g,~]=edge(f,'sobel','horizontal');
subplot(3,1,2);
imshow(g);
[g,~]=edge(f,'sobel','both');
subplot(3,1,3);
imshow(g);
```

Prewitt Operator

```
[g,~]=edge(f,'prewitt','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,~]=edge(f,'prewitt','horizontal');
subplot(3,1,2);
imshow(g);
[g,~]=edge(f,'prewitt','both');
subplot(3,1,3);
imshow(g);
```

Roberts Operator:

```
[g,~]=edge(f,'roberts','vertical');
figure,subplot(3,1,1);
imshow(g);
[g,~]=edge(f,'roberts','horizontal');
subplot(3,1,2);
imshow(g);
[g,~]=edge(f,'roberts','both');
subplot(3,1,3);
imshow(g);
```

Canny Operator:

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

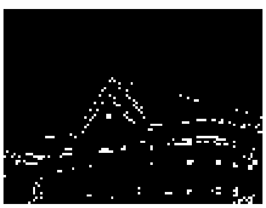
OUTPUT

Original Image

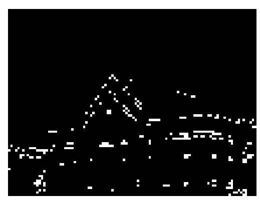
Sobel- Vertical

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

LINE DETECTION

PROGRAM

```
a = imread('robo.jpg');
f = rgb2gray(a);
```

Define Filters for Line Detection

```
horizontal_filter = [-1, -1, -1; 2, 2, 2; -1, -1, -1];
vertical_filter   = [-1,  2, -1; -1, 2, -1; -1,  2, -1];
diag45_filter     = [-1, -1,  2; -1,  2, -1; 2, -1, -1];
diag135_filter    = [2, -1, -1; -1,  2, -1; -1, -1, 2];
```

Apply Filters

```
g_hor  = abs(imfilter(double(f), horizontal_filter));
g_ver  = abs(imfilter(double(f), vertical_filter));
g_45   = abs(imfilter(double(f), diag45_filter));
g_135  = abs(imfilter(double(f), diag135_filter));
```

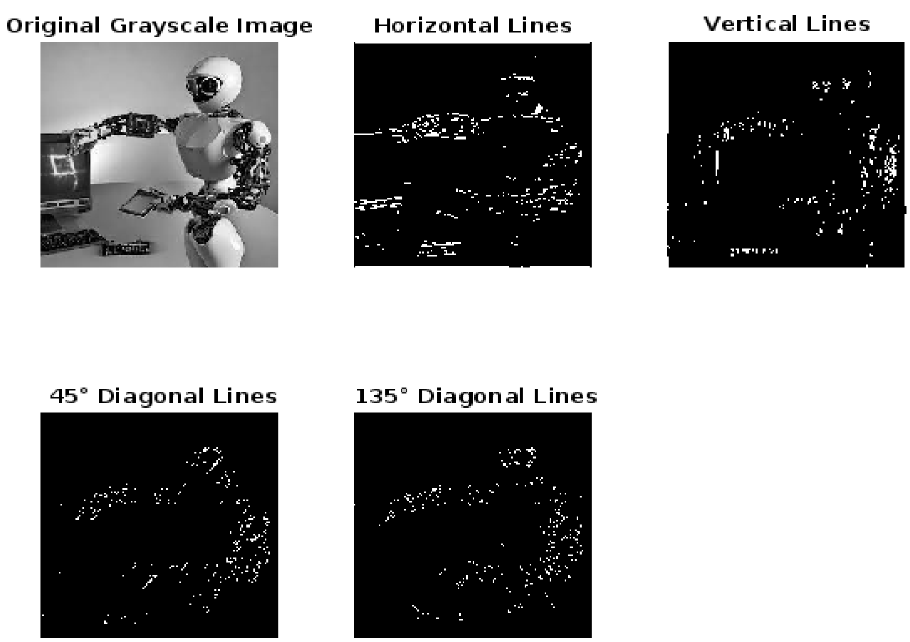
Thresholding to Enhance Line Detection

```
T = 300;
g_hor  = g_hor  >= T;
g_ver  = g_ver  >= T;
g_45   = g_45   >= T;
g_135  = g_135  >= T;
```

Display Results

```
figure;
subplot(2,3,1);
imshow(f);
title('Original Grayscale Image');

subplot(2,3,2);
imshow(g_hor);
title('Horizontal Lines');

subplot(2,3,3);
imshow(g_ver);
title('Vertical Lines');

subplot(2,3,4);
imshow(g_45);
title(' 45° Diagonal Lines');

subplot(2,3,5);
imshow(g_135);
title(' 135° Diagonal Lines');
```

OUTPUT



POINT DETECTION

PROGRAM

```
a = imread('robo.jpg');
f = rgb2gray(a);
```

Define Laplacian Filter

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

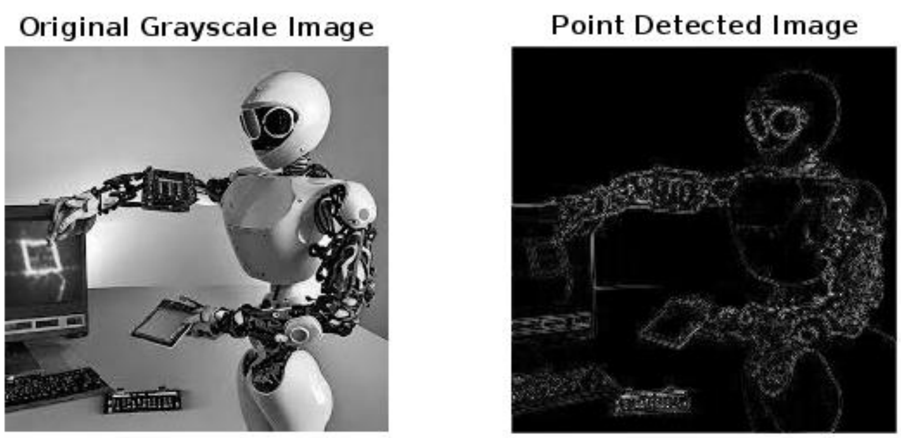
Apply the Filter

```
g = abs(imfilter(double(f), w));
```

Display Results

```
figure;
subplot(1,2,1);
imshow(f);
title('Original Grayscale Image');

subplot(1,2,2);
imshow(g, []);
title(' Point Detected Image');
```

OUTPUT



RESULT

The implementation of image segmentation techniques such as edge detection, line detection, and point detection was successfully carried out using MATLAB.

EX.NO: 10 **DATE:**

REGION BASED SEGMENTATION

AIM

region based segmentation of image using Matlab.

PROCEDURE

Step 1: Read and Preprocess the Image

Step 2: Apply Otsu's Thresholding

Step 3: Apply Manual Thresholding

Step 4: Display the Results

Step 5: Display the Histogram

PROGRAM

Read Image

```
a = imread(computers.jpg);
```

Convert to Grayscale if it is RGB

```
if size(a,3) == 3
    a = rgb2gray(a);
end
```

Display Original Image

```
subplot(2,2,1);
imshow(a);
title('Original Image');
```

Apply Otsu's Thresholding

```
level = graythresh(a);
b = im2bw(a, level);
```

Display Thresholded Image

```
subplot(2,2,2);
imshow(b);
title('Thresholded Image');
```

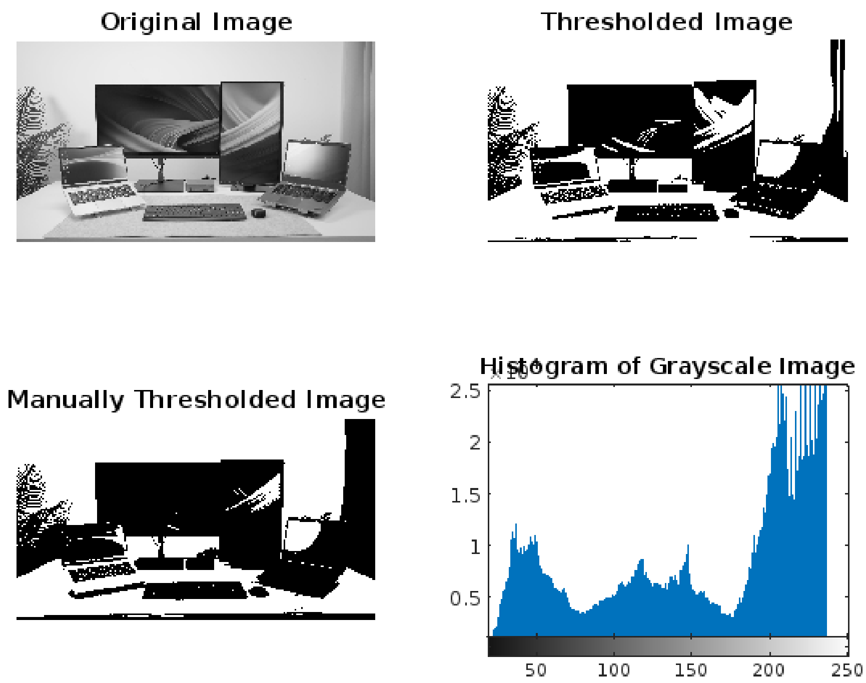
Manual Thresholding (Pixels > 180)

```
a1 = a > 180;
subplot(2,2,3);
imshow(a1);
title('Manually Thresholded Image');
```

Display Histogram

```
subplot(2,2,4);
imhist(a);
title('Histogram of Grayscale Image');
```

OUTPUT



RESULT

The implemented region based segmentation of image successfully executed.

EX.NO: 11 **DATE:**

BASIC MORPHOLOGICAL OPERATIONS

AIM:

To implement and analyze basic morphological operations such as Erosion, Dilation, Opening, and Closing on a binary image using MATLAB.

PROCEDURE

Step 1: Read and Convert the Image

Step 2: Define Structuring Element

Step 3: Apply Morphological Operations

Step 4: Display the Results

PROGRAM

Read and convert the image to binary

```
img1 = imread('coins.jpg'); % Load binary image (black & white)
img = im2bw(img1); % Convert to binary (if not already)
```

Define Structuring Element (3x3 Square)

```
se = strel('square', 3);
```

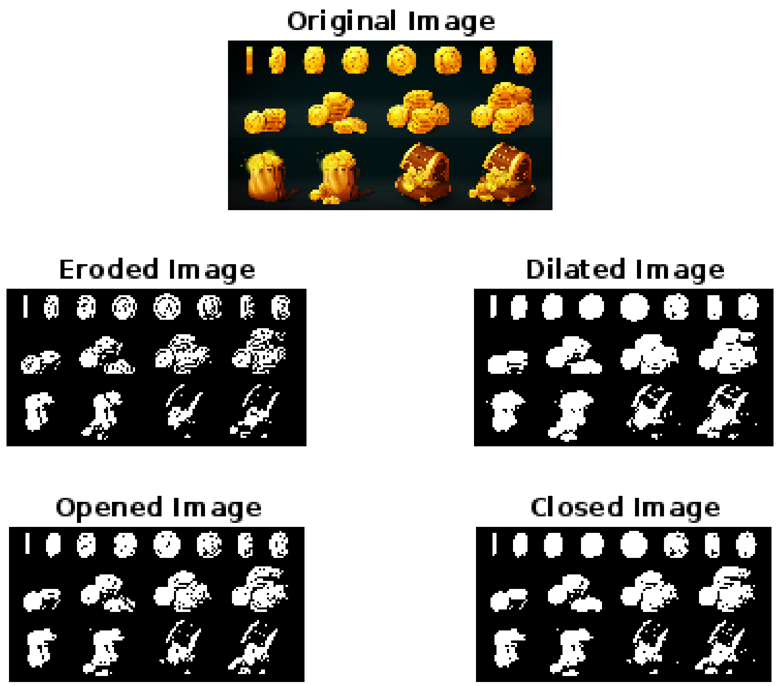
Apply Morphological Operations

```
eroded_img = imerode(img, se);     % Erosion
dilated_img = imdilate(img, se);   % Dilation
opened_img = imopen(img, se);      % Opening (Erosion → Dilation)
closed_img = imclose(img, se);     % Closing (Dilation → Erosion)
```

Display Results

```
figure;
subplot(3,2,[1 2]); imshow(img1); title('Original Image');
subplot(3,2,3); imshow(eroded_img); title('Eroded Image');
subplot(3,2,4); imshow(dilated_img); title('Dilated Image');
subplot(3,2,5); imshow(opened_img); title('Opened Image');
subplot(3,2,6); imshow(closed_img); title('Closed Image');
```

OUTPUT



RESULT

The implemented morphological operations successfully modified the structure of objects in the binary image.

EX.NO: 12 **DATE:**

IMPLEMENTATION OF IMAGE COMPRESSION TECHNIQUES

AIM:

To implement and analyze image compression techniques using MATLAB.

PROCEDURE

Huffman Coding for Image Compression

Step 1: Read the Image

- Convert the image into a **1D array** (flatten it).

Step 2: Compute Frequency of Pixel Values

- Count the occurrences of each pixel intensity value.

Step 3: Generate Huffman Dictionary

- Build a Huffman tree based on the pixel frequency.
- Assign shorter codes to frequently occurring values.

Step 4: Encode the Image

- Replace pixel values with their Huffman codes.

Step 5: Decode the Image

- Convert the Huffman-coded values back to pixel intensities using the dictionary.

Step 6: Display Results

PROGRAM

Read grayscale image

```
img = imread('cameraman.tif');
img = uint8(img(:));  % Convert image to a 1D array
```

Compute histogram for unique symbols

```
symbols = unique(img);
counts = histcounts(img, [symbols; max(symbols)+1]);
```

Generate Huffman dictionary

```
dict = huffmandict(symbols, counts / numel(img));
```

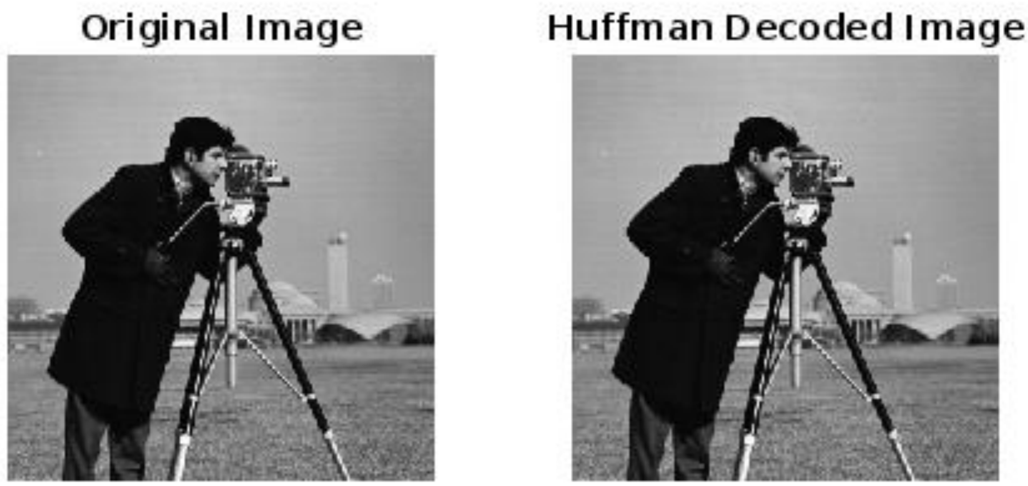
Encode image using Huffman coding

```
encoded_data = huffmanenco(img, dict);
```

Decode the Huffman encoding

```
decoded_data = huffmandeco(encoded_data, dict);
decoded_img = reshape(uint8(decoded_data), size(imread('cameraman.tif')));
figure;
subplot(1,2,1); imshow(imread('cameraman.tif')); title('Original Image');
subplot(1,2,2); imshow(decoded_img); title('Huffman Decoded Image');
```

OUTPUT



RESULT

The implemented image compression techniques successfully reduced image size while preserving important visual details.