Experiment No. 1
Experiment Title: Image Processing Concept

Aim: To study the Image Processing concept.

Tools Required: MATLAB R2015a

Commands used:

- **imread** → Loads an image.
- **imcomplement** → Computes the negative of an image.
- **imshow** → Displays an image.
- **figure** → Opens a new window for displaying an image.
- **imadjust** → Adjusts image contrast based on intensity mapping.

Theory: Digital images play an important role both in daily life applications as well as in the areas of research technology. Digital image processing refers to the manipulation of an image using a processor. The different elements of an image processing system include Image acquisition, image storage, image processing and display

An image is a two-dimensional function that represents a message of sum characteristics cs such as brightness or color of the viewed scene in the first MATLAB program. The command used from MATLAB is imcomplement.

Program:

% Program to study the image processing concept

```
I=imread(pout.tif);
J=imcomplement(I),
figure.imshow(I)
figure.imshow(J)
K=imadjust(I, [0,0.4],[05,1])
figure.imshow(K)
```

Results:

Original Image

Complement Image

Conclusion: Thus, we have studied how to obtain a complement image from the original image.

Experiment No. 2
Experiment Title: Histogram Equalization Image

Aim: To obtain a histogram equalization image.

Tools Required: MATLAB R2015a

Commands used:

- **imread** → Reads an image.
- **imshow** → Displays an image.
- **imhist** → Displays the histogram of an image.
- **imcomplement** → Computes the negative (complement) of an image.
- **histeq** → Performs histogram equalization to enhance contrast.
- **numel** → Counts the total number of pixels in an image.
- **plot** → Plots the normalized histogram.
- **imadjust** → Adjusts image contrast using intensity mapping.
- **maketform('affine', ...)** → Creates an affine transformation matrix.
- **imtransform** → Applies a geometric transformation to an image.

Theory: Histogram equalization is a technique used in image processing to enhance contrast by utilizing the image's histogram. While it can lead to unrealistic effects in photographs, it is particularly beneficial for scientific images, such as thermal or X-ray images, which often undergo false colour processing. However, applying histogram equalization to images with low colour depth can yield undesirable results. For instance, if it is applied to an 8-bit image displayed in 8-bit grayscale, it may further decrease the colour depth, resulting in fewer unique shades of gray. Histogram equalization is most effective when used on images with a much higher depth than the palette size, such as continuous data or 16-bit grayscale images.

Program:

```
I = imread('trees.tif');
figure, imshow(I);
title('Original Image');
figure, imhist(I, 100);
title('Original Histogram');
J = imcomplement(I);
figure, imshow(J);
title('Complemented Image');
I_eq = histeq(I);
figure, imshow(I_eq);
title('Equalized Image');
figure, imhist(I_eq, 64);
title('Equalized Histogram');
n = numel(I);
p = imhist(I) / n;
figure, plot(p);
title('Normalized Histogram');
```

```
K = imadjust(I, [0; 1], [0.4; 1], 0.5);
figure, imshow(K);
title('Adjusted Image');
T = maketform('affine', [0.3 0 0; 0.5 1 0; 0 0 1]);
I_transformed = imtransform(I, T);
figure, imshow(I_transformed);
title('Transformed Image');
```

Results:

Conclusion: Thus, we have obtained the Equalized Histogram from the original Histogram.

Experiment No. 3
Experiment Title: Averaging Filter in Spatial Domain

Aim: To implement a smoothing or averaging filter in the spatial domain.

Tools Required: MATLAB R2015a

Commands used:

- **imread** → Reads an image.
- **subplot(m,n,p)** → Creates a grid of plots (m × n) and selects position p.
- **imshow** → Displays an image.
- **title** → Adds a title to the displayed image.
- **ones(m,n)** → Creates a matrix of ones.
- **imfilter** → Applies a filter to an image.
- **'circular' padding** → Wraps around pixel values at the borders.

Theory: Filtering is a technique for modifying or enhancing an image. Masks or filters will be defined. The general process of convolution and correlation will be introduced via an example. Also smoothing linear filters such as box and weighted average filters will be introduced.

In statistics and image processing, to smooth a data set is to create an approximating function that attempts to capture important patterns in the data while leaving out noise or other in-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so that individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased, leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis by being able to extract more information from the data as long as the assumption of smoothing is reasonable by being able to provide analyses that are both flexible and robust. Different algorithms are used in smoothing.

Program:

```
% Program for the implementation of smoothing or averaging filter in spatial domain

I = imread('trees.tif');
subplot(2,2,1);
imshow(I);
title('Original Image');
f = ones(3,3) / 9;
h = imfilter(I, f, 'circular');
subplot(2,2,2);
imshow(h);
title('Averaged Image');
```

Result:

Conclusion: Thus, we have performed the smoothing or averaging filter operation on the original image, and we get a filtered image.

Experiment No. 4
Experiment Title: Opening And Closing of the Image

Aim: Program for opening and closing of the image.

Tools Required: MATLAB R2015a

Commands Used:

- **imread** → Reads an image.
- **strel** → Creates a structuring element for morphological operations.
- **imopen** → Performs morphological opening (erosion → dilation).
- **imclose** → Performs morphological closing (dilation → erosion).
- **figure** → Opens a new figure window.
- **imshow** → Displays an image.
- **title** → Adds a title to the displayed image.

Theory: In mathematical morphology, opening is the dilation of the erosion of a set by structuring elements B:

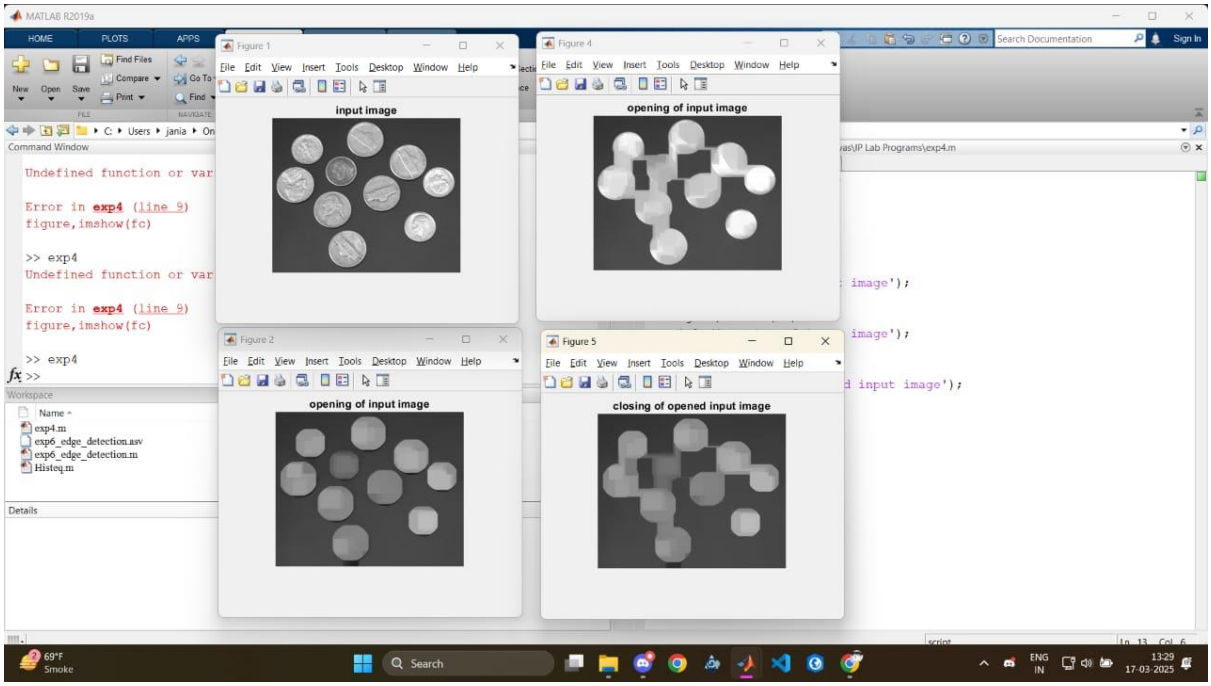
Together with closing, the opening serves in computer vision and image processing as a basic workhorse of morphological noise removal. Opening removes small objects from the foreground (usually taken as the bright pixels) of an image, placing them in the background, while closing removes small holes in the foreground, changing small islands of background into the foreground. These techniques can also be used to find specific shapes in an image. The opening can be used to find things into which a specific structuring element can it (edges, corners, ...).

In mathematical morphology, the closing of a set (binary image) A by a structuring element B is the erosion of the dilation of that set. In image processing, closing is, together with opening, the basic workhorse of morphological image removal. Opening removes small objects, while closing removes small holes.

Program:

```
f = imread('coins.png');
se = strel('square', 20);
fo = imopen(f, se);
figure, imshow(f)
title('Input Image');
figure, imshow(fo)
title('Opening of Input Image');
fc = imclose(f, se);
figure, imshow(fc)
title('Closing of Input Image');
foc = imclose(fo, se);
figure, imshow(foc)
title('Closing of Opened Input Image');
```

Results:



Conclusion: Thus, we have obtained the opened image and closed image from the original Image.

Experiment No. 5
Experiment Title: Region of Interest for the Image

Aim: To fill the region of interest for the image.

Tools Required: MATLAB R2015a

Commands Used:

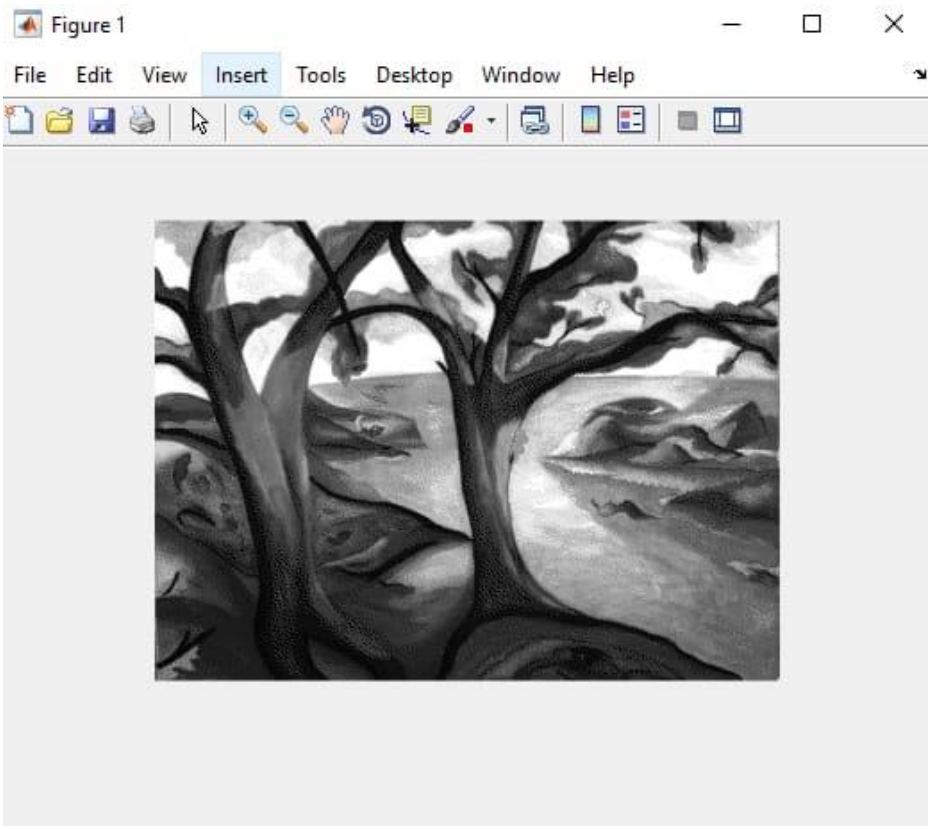
- **close all** → Closes all figure windows.
- **load trees** → Loads the built-in trees image dataset.
- **ind2gray** → Converts an indexed image to grayscale.
- **figure** → Opens a new figure window.
- **imshow** → Displays an image.
- **title** → Adds a title to the displayed image.
- **roi = I;** → Assigns the grayscale image to a new variable roi.

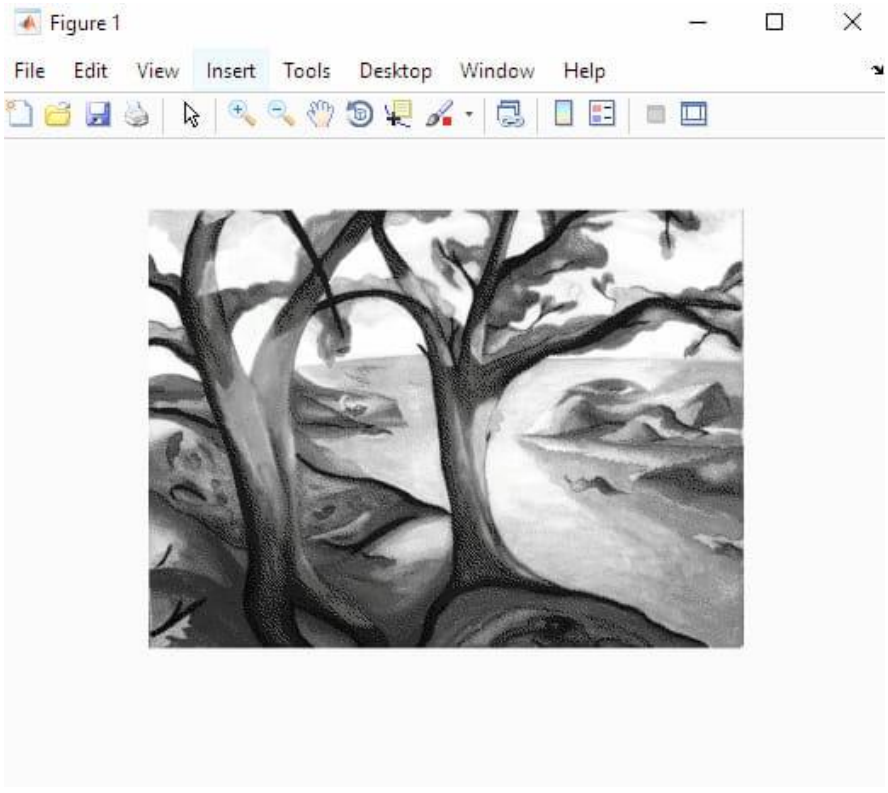
Theory: A region of interest (often abbreviated ROI), a sample within a data set identified for a particular purpose. The concept of an ROI is commonly used in many application areas. For example, in medical imaging, the boundaries of a tumour may be defined on an image or in a volume to measure its size. The endocardial border may be defined on an image, perhaps during different phases of the cardiac cycle, for example, end-systole and end-diastole, to assess cardiac function. In geographical information systems (GIS), an ROI can be taken literally as a polygonal selection from a 2D map. In computer vision and optical character recognition, the ROI defines the borders of an object under consideration. In many applications, symbolic (textual) labels are added to an ROI, to describe its content compactly. Within an ROI may lie individual points of interest (POIs).

Program:

```
% Program for ROI

close all;
load trees;
I = ind2gray(X, map);
figure, imshow(I);
title('Original Image');
roi = I;
figure, imshow(roi);
title('Output Image');
```

Results:





Conclusion: We have filled the interested region in the original image.

Experiment No. 6
Experiment Title: Edge Detection Algorithm

Aim: Program for edge detection algorithm.

Tools Required: MATLAB

Commands Used:

- **imread('coins.png')** → Reads the input image and stores it in a variable I.
- **imshow(I)** → Displays the original image.
- **title('Figure 1: Original Image')** → Adds a title to the displayed image.
- **ones(5,5) / 25** → Creates a 5×5 averaging filter (mean filter).
- **imfilter(I, h)** → Applies the averaging filter to smooth the image.
- **edge(b, 'sobel')** → Detects edges using the Sobel operator.
- **edge(b, 'prewitt')** → Detects edges using the Prewitt operator.
- **edge(b, 'roberts')** → Detects edges using the Roberts operator.
- **edge(b, 'canny')** → Detects edges using the Canny operator.
- **Each figure, imshow()** → Creates a new figure window and displays an image.

Theory: The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a computational theory of edge detection explaining why the technique works.

The Process of the Canny edge detection algorithm can be broken down into 5 different steps:

1. Apply a Gaussian filter to smooth the image to remove the noise

2. Find the intensity gradients of the image

3. Apply non-maximum suppression to get rid of spurious responses to edge detection

4. Apply a double threshold to determine potential edges

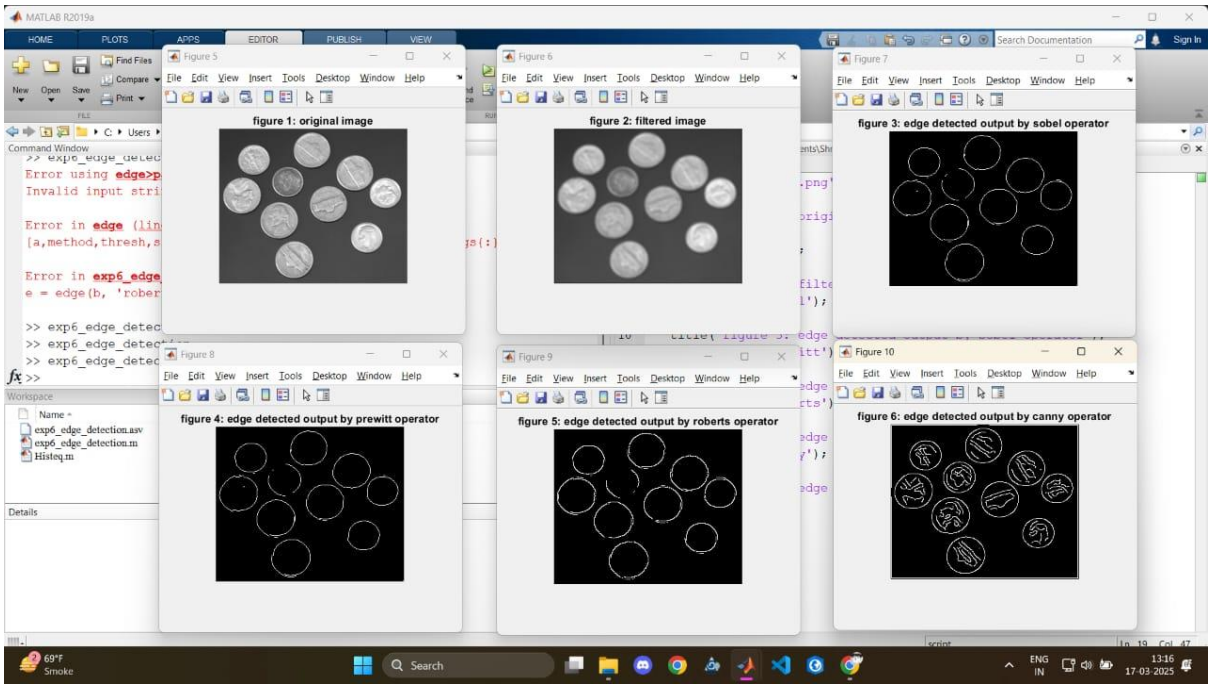
5. Track edges by hypothesis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Program:

```
% Program for edge detection algorithm

I = imread('coins.png');
figure, imshow(I);
title('Figure 1: Original Image');
h = ones(5,5) / 25;
b = imfilter(I, h);
figure, imshow(b);
title('Figure 2: Filtered Image');
c = edge(b, 'sobel');
figure, imshow(c);
```

```
title('Figure 3: Edge Detection using Sobel Operator');
d = edge(b, 'prewitt');
figure, imshow(d);
title('Figure 4: Edge Detection using Prewitt Operator');
e = edge(b, 'roberts');
figure, imshow(e);
title('Figure 5: Edge Detection using Roberts Operator');
f = edge(b, 'canny');
figure, imshow(f);
title('Figure 6: Edge Detection using Canny Operator');
```

Results:



Conclusion: Thus, we have detected the edges in the original image.

Experiment No. 7
Experiment Title: Sharpen Image Using Gradient Mask

Aim: Program to sharpen the image using a gradient mask

Tools Required: MATLAB R2015a

Commands Used:

- **imread('inspag.png')** → Reads the input image and stores it in a variable I.
- **subplot(2,2,1);** → Divides the figure into a **2×2 grid** and selects the **first section**.
- **imshow(I);** → Displays the original image in the first subplot.
- **title('Original Image');** → Adds a title to the first subplot.
- **fspecial('sobel');** → Creates a **Sobel filter kernel** for edge detection.
- **imfilter(I, h);** → Applies the **Sobel filter** to the image.
- **subplot(2,2,2);** → Selects the **second section** in the 2×2 grid.
- **imshow(F);** → Displays the filtered image in the second subplot.
- **title('Filtered Image using Sobel Mask');** → Adds a title to the second subplot.
- **edge(I, 'sobel');** → Applies the **Sobel edge detection algorithm**.
- **subplot(2,2,4);** → Selects the **fourth section** in the 2×2 grid.
- **imshow(C);** → Displays the edge-detected image in the fourth subplot.
- **title('Edge Detection using Sobel');** → Adds a title to the fourth subplot.

Theory: An image gradient is a directional change in the intensity or color in an image The gradient of the image is one of the fundamental building blocks in image processing. For example, the Canny edge detector uses image gradients for edge detection. Mathematically, the gradient of a two-variable function (here the image intensity function) at each image point is a 20 vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of the largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction

Program:

%Program of sharpened image using a gradient mask

```
I = imread('inspag.png');
subplot(2,2,1);
imshow(I);
title('Original Image');
h = fspecial('sobel');
F = imfilter(I, h);
subplot(2,2,2);
imshow(F);
title('Filtered Image using Sobel Mask');
C = edge(I, 'sobel');
subplot(2,2,4);
imshow(C);
title('Edge Detection using Sobel');
```

Results:

Conclusion: Thus, we have performed the sharpening opera using a gradient mask on the original image.

Experiment No. 8.
Experiment Title: DCT/IDCT Computation

Aim: Program for DCT/IDCT computation.

Tools Required: MATLAB R2015a

Commands Used:

- **clc;** – Clears the command window.
- **clear all;** – Clears all variables from the workspace.
- **close all;** – Closes all open figure windows.
- **input('Enter the basis matrix dimension: ');** – Prompts the user to enter a value.
- **ones(1, n);** – Creates a row vector of ones with length n.
- **sqrt(value);** – Computes the square root of a given value.
- **cell(m, n);** – Creates an m × n cell array.
- **for ... end** – Defines a loop structure to iterate over a range of values.
- **cos(value);** – Computes the cosine of the given value (in radians).
- **pi;** – Represents the mathematical constant π (pi).
- **figure(3);** – Creates a figure window with the specified number.
- **subplot(m, n, k);** – Creates a grid of m × n subplots and activates the kth subplot.
- **imshow(a{i, j}, []);** – Displays the image stored in cell array a{i, j} with auto-scaling.

Theory: A discrete cosine transform (DCT) expresses a inite 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 (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions.

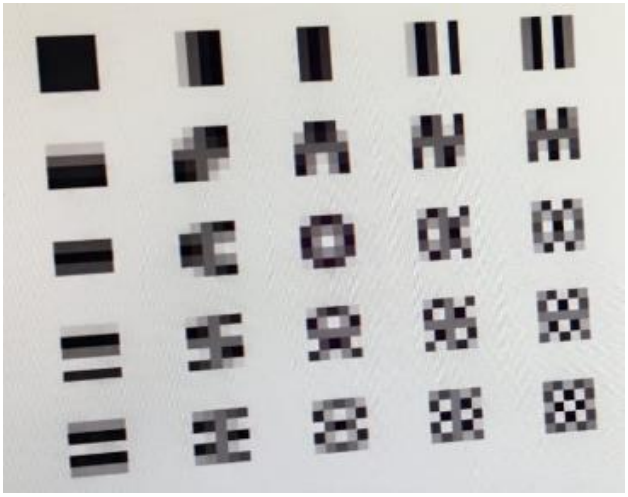
In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. The DCTs are generally related to Fourier Series coeficients of a periodically and symmetrically extended sequence whereas DFTS are related to Fourier Series coeficients of a periodically extended sequence. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), whereas in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common.

Like for the DFT, the normalization factor in front of these transform de initions is merely a convention and differs between treatments. For example, some authors multiply the transforms by so that the inverse does not require any additional multiplicative factor. Combined with appropriate factors of sqrt (2 / N) , this can be used to make the transform matrix orthogonal.

Program:

```
clc;
clear all;
close all;
m = input('Enter the basis matrix dimension: ');
n = m;
alpha2 = ones(1, n) * sqrt(2 / n);
alpha2(1) = sqrt(1 / n);
alpha1 = ones(1, m) * sqrt(2 / m);
alpha1(1) = sqrt(1 / m);
a = cell(m, n);
for u = 0:m-1
   for v = 0:n-1
      for x = 0:m-1
         for y = 0:n-1
            a{u+1, v+1}(x+1, y+1) = alpha1(u+1) * alpha2(v+1) * ...
               cos((2*x+1)*u*pi/(2*n)) * cos((2*y+1)*v*pi/(2*n));
         end
      end
   end
end
figure(3);
k = 1;
for i = 1:m
   for j = 1:n
      subplot(m, n, k);
      imshow(a{i,j}, []);
      k = k + 1;
   end
end
```

Results:



Conclusion: Thus, we have obtained the DCT/IDCT for the image.

Experiment 9
Experiment Title: Region of Interest

Aim: To fill the region of interest of the image.

Software Used: MATLAB R2015a

Commands Used:

- **clc;** → Clears the command window.
- **close all;** → Close all open figure windows.
- **load trees;** → Loads the trees dataset containing an indexed image.
- **imshow(X, map);** → Displays the original image with its colourmap.
- **title('Original Image');** → Adds a title to the figure.
- **fprintf(...);** → Prints text in the command window.
- **roipoly();** → Allows the user to select a polygonal Region of Interest (ROI) interactively.
- **filled_image(roi_mask) = 255;** → Fills the selected region with white.
- **subplot(1,2,1);** → Creates a subplot (1 row, 2 columns, first section).
- **imshow(X, map);** → Displays the original image.
- **subplot(1,2,2);** → Moves to the second subplot.
- **imshow(filled_image, map);** → Displays the modified image.

Theory:

A region of interest (often abbreviated as ROI) consists of samples within a data set identified for a specific purpose. The concept of an ROI is commonly utilised in various application areas. For instance, in medical imaging, the boundaries of a tumour may be delineated on an image, potentially during different phases of the cardiac cycle, such as end-systole and end-diastole, to assess cardiac function.

In geographical information systems, i.e., GIS, a ROI can be interpreted literally as a polygonal selection from a 2D map. In computer vision and optical character recognition, the ROI delineates the borders of an object under consideration. Within an ROI may exist individual points of interest referred to as POIs.

Use ROIFILL to fill in a specified polygon in an image. ROIFILL smoothly interpolates inwards from the pixel values on the boundary of the polygon by solving Laplace's equation. ROIFILL can be employed, for instance, to erase objects in an image.

J = ROIFILL creates an interactive polygon tool associated with the image displayed in the current figure, called the target image. You can place the tool interactively, using the mouse to specify the vertices of the polygon. Double-click to add a final vertex to the polygon and complete it. Right-click to close the polygon without adding another vertex. You can adjust the position of the polygon and the individual vertices by clicking and dragging. To delete the polygon, press Backspace, Escape, or Delete, or select the Cancel option from the context menu. If the polygon is deleted, all return values are set to empty. After positioning and sizing the polygon, fill the region by either double-clicking on the polygon or selecting Fill Area from the tool's context menu. ROIFILL returns J, a version of the image with the region filled.

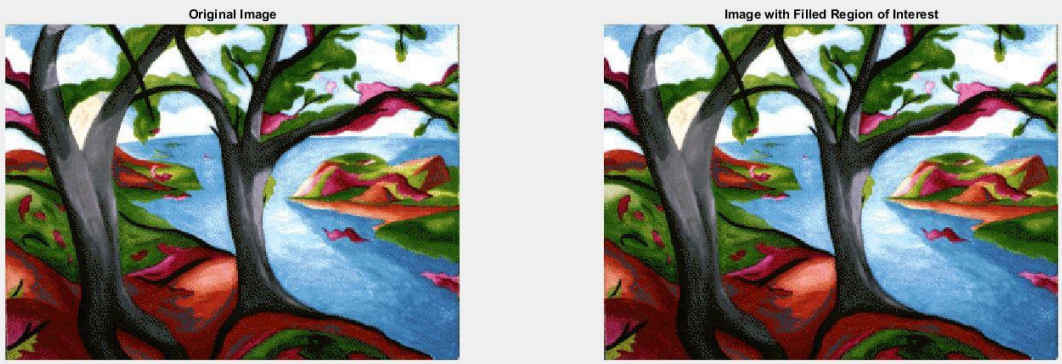
J = ROIFILL(I) displays the image I and creates an interactive polygon tool associated with that image.

J = ROIFILL(I, G, B) fills the polygon specified by C and R, which are equal-length vectors containing the row-column coordinates of the pixels at the vertices of the polygon. The kith vertex is in the pixel (R(k), C(k)).

Program:

```
clc;
close all;
load trees;
imshow(X, map);
title('Original Image');
fprintf('Select the region of interest using roipoly.\n');
roi_mask = roipoly(); % Returns a binary mask of the selected region
filled_image = X;
filled_image(roi_mask) = 255; % Fill the selected region with white
figure;
subplot(1,2,1);
imshow(X, map);
title('Original Image');
subplot(1,2,2);
imshow(filled_image, map);
title('Image with Filled Region of Interest');
```

Result:



Conclusion: Thus, we have filled the interested region in the original image.

Experiment 10
Experiment Title: Erosion and Dilation

Aim: Program for morphological operation: erosion and dilation.

Software Used: MATLAB R2015a

Commands Used:

- **imread('coins.png')** – Reads the input image.
- **B = [...]** – Defines a custom 3x3 structuring element for dilation.
- **strel('disk', 10)** – Creates a disk-shaped structuring element for erosion.
- **imdilate(f, B)** – Performs dilation using structuring element B.
- **imerode(f, se)** – Performs erosion using the disk-shaped element.
- **imshow(...)** – Displays images in figure windows.
- **title('...')** – Adds titles to the displayed images.

Theory:

Erosion (usually represented by $\ominus$) is one of two fundamental operations (the other being dilation) in morphological image processing from which all other morphological operations are based. It was originally defined for binary images, later being extended to grayscale images, and subsequently to complete lattices.

With A and B as two sets in Z2 (2D integer space), the dilation of A and B is defined as

$$A(+)B=\{Z|(B\hat{})Z\cap A \neq \phi\}$$

In the above example, A is the image while B is called a structuring element.

In the equation,$(B\hat{})Z$ simply means taking the reflections of B about its origin and shifting it by Z. Hence dilation of A with

B is a set of all displacements, Z, such that $(B\hat{})Z$ and A overlap by at least one element. Flipping of B about the origin and then moving it past image A is analogous to the convolution process. In practice flipping of B is not done always.

Dilation adds pixels to the boundaries of object in an image. The number of pixels added depends on the size and shape of the structuring element. Based on this definition, dilation can be defined as

$$A(+)B=\{\{Z|(B\hat{})Z\cap A\} \epsilon A\}$$

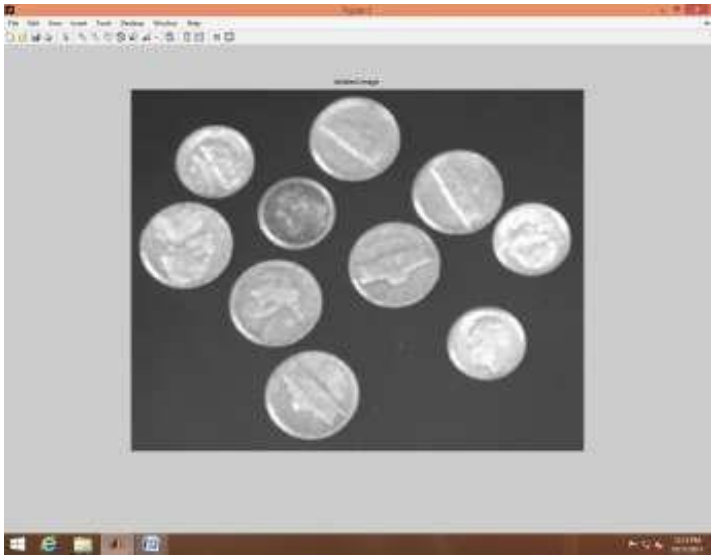
Program:

```
% Read the input image
f = imread('coins.png');

% Define structuring elements
B = [0 1 1; 1 1 1; 0 1 0]; % Structuring element for dilation
se = strel('disk', 10); % Structuring element for erosion
```

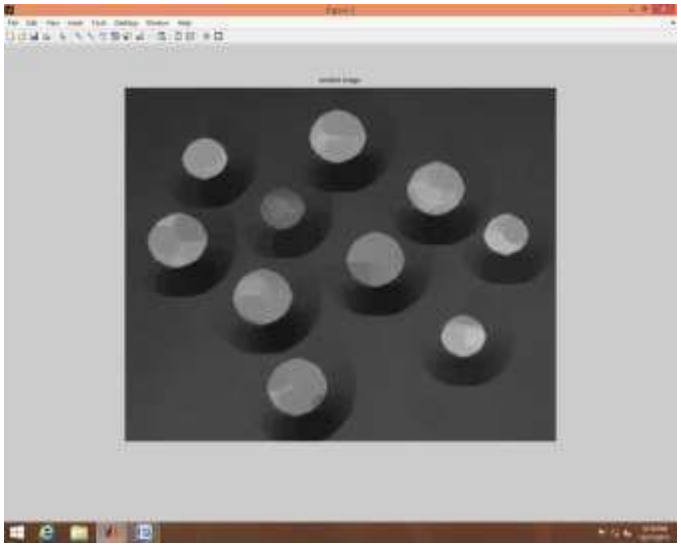
```
% Perform dilation
f1 = imdilate(f, B);

% Perform erosion
f2 = imerode(f, se);

% Display the results figure, imshow(f);
title('Input Image');

figure, imshow(f1);
title('Dilated Image');

figure, imshow(f2);
title('Eroded Image');
```

Result:

Original Image



Eroded Image



Dilated Image

Conclusion: Thus, we have obtained the eroded and dilated image for the original image.