312CCS-3 Image Processing Lab Manual
1444-1445 (2023-2024)
Tri-Semester-I
King Khalid University
College of computer sciences
Department of computer sciences

Course Coordinator:
Dr. Gran Badshah
Email : gdostan@kku.edu.sa

Course Description (As per course Specification)

This course is designed to provide introduction of Image Processing, an overview of Image processing principles, algorithm and other important operation applicable in Digital Image processing. The course begins with an introduction to image processing and its applications. Further contents include fundamental concepts of visual perception of digital images and their acquisition, including basic techniques of image manipulation such as segmentation, compression, watermarking and other preliminary understanding of pattern recognition.

Course Learning Outcomes (As per syllabus)

Digital image processing is universal with applications ranging from television to tomography, from photography to printing, from robotics to remote sensing. This course will introduce the basic techniques of digital image processing in different fields such as medical. The student will learn modern approaches to image acquisition and display, image enhancement, image compression, segmentation, watermarking, and their analysis.

On successful completion of this course, students should be able to:

1. Understand how computers can process digital images.
2. Understand the processes of two-dimensional data together with any perceived similarity with the human vision system.
3. Learn the theoretical foundations of modern image processing techniques.
4. Apply theory in a practical image processing environment.

Lab Description

This Lab is designed to provide the students with an overview of digital image processing using MATLAB. The course covers the fundamentals of image formation, low-level image processing and enhancement in both the spatial and frequency domains and gives a flavor of higher level tasks by studying image analysis, color image processing, segmentation, compression and watermarking.

Hardware / Software Requirements

- MATLAB with Normal PC.

Marks Distribution with assessment criteria

S. No	Assessment	Week Due	Marks
1	Lab Activity 1		1.5
2	Lab Activity 2		1.5
3	Lab Activity 3		1.5
4	Lab Activity 4		1.5

5	Lab Activity 5		1.5
6	Lab Activity 6		1.5
7	Lab Activity 7		1.5
8	Lab Activity 8		1.5
9	Lab Activity 9		1.5
10	Lab Activity 10		1.5
11	Lab Mini Project		5
Total Marks			20

No	List of Topics	Contact Hours	
1	**Introduction to MATLAB:** • Use of MATLAB Environment, Variables and workspace, dealing with matrices, Plots. • Help in MATLAB. • Execution of basic commands.	2	
2	**Extraction of Image Attributes:** • Basic Image import, processing, export. • Image Acquisition, Sampling and Pixel extraction.	2	
3	**Demonstrate Intensity Transformation and Spatial Filtering:** • Image Smoothing (Low pass) and Sharpening (High pass). • Intensity and Histogram Processing.	2	
4	**Implementation of Discrete Fourier Transformation (DFT):** • The one-dimensional DFT • The two-dimensional DFT • Inverse DFT	2	
5	**Color Image Processing Methods** • Color Models • Color image Smoothing and Sharpening • Segmentation	2	
6	**Use of some lossy and lossless Image Compression Algorithms** PNG, JPEG, LZW etc.	2	
7	**Image Segmentation** • Edge Detection • Image Thresholding • Region Detection Types	4	
8	**Image Enhancement Techniques:** • Applications of Enhancement Techniques on Digital	2	

	Images		
9	**Watermarking Applications** Watermark insertion into and its extraction from a digital image	2	

Week-wise Practical Schedule (10 weeks)

Week	Lab/ Exercise #	Title of the Experiment / Exercise	Page No.
Week 1 to 3	1 to 3	**Introduction to MATLAB:** Use of MATLAB Environment, Variables and workspace, dealing with images such as to flip an image vertically and horizontally along a line.	5
Week 4	4	**Extraction of Image Attributes:** Histogram Equalization Calculate the histogram of a given image	8
Week 5	5	**Color Image Processing Methods** Display the color components of the image	10
Week 6	6	**Image Enhancement Techniques:** Contrast stretching	13
Week 7	7	**Extraction of Image Attributes:** Display both images in one figure using subplot	14
Week 8	8	**Image Segmentation** Edge detections with gradient and convolution	15
Week 9	9	**Demonstrate Intensity Transformation and Spatial Filtering:** Filter the image using the Average, Median, Laplacian filters	17-28
Week 10	10	**Revision/ exams**	

Lab Session #1 to 3 / Exercise #1 to 3 (Week #1 to 3)

Program Outcome:

The image will flip vertically and horizontally along with the line.

The Negative of the image will be displayed.

The image will be rotated to given angle.

Description:
Flipping images is easy and straightforward in MATLAB. If you want to flip your image vertically or horizontally

Flipping and transposing both belong to image rotation, with rotating angle 90, 180, or 270(equals to -90) degrees. Sometimes we may want to rotate our images by some arbitrary degrees. This can be done by calling the MATLAB internal function **imrotate(array, angle)**.

Negative of the Image will be found using formula **neg = 255-org;**

First of all, we must rotate the data array which generates the image:

RotateA = imrotate(A, 45);

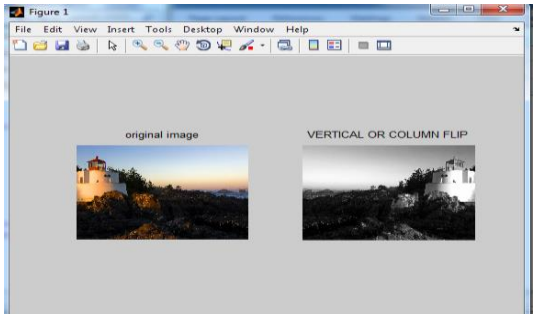
This means we want to rotate data array **A** by **45** degrees and save it in array **RotateA**. Note that the rotation is in a counterclockwise direction around the array's center point. You can specify a negative value for **angle** to rotate the image clockwise.

Exercises (To be solved in the Lab):

1) Write a MATLAB program to flip the image around vertical line.
PROGRAM:

```
clear all;
close all;
a = imread('C:\Users\Public\Pictures\Sample Pictures\Lighthouse.jpg');
c = size(a);
for i = 1:c(1)
for j = 1:c(2)
    b(i,c(2)+1-j) = a(i,j);
end
end
subplot 121,imshow(a);title('original image');
subplot 122,imshow(b),title('VERTICAL OR COLUMN FLIP');
```

OUTPUT:

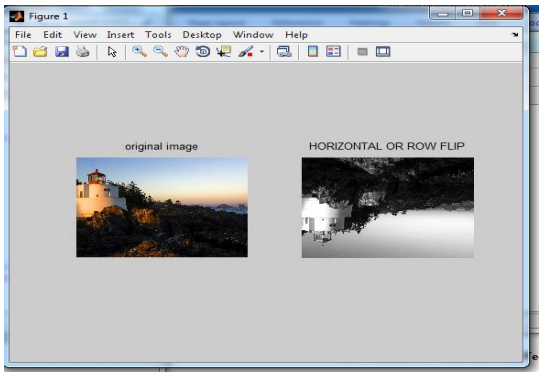


2) Flip the image around horizontal line

PROGRAM:

```
clear all;
close all;
a = imread('C:\Users\Public\Pictures\Sample Pictures\Lighthouse.jpg');
c = size(a);
z=c(1);
for i = 1:c(1)
for j = 1:c(2)
    b(z,j)= a(i,j);
end
z=z-1;
end
subplot 121,imshow(a);title('original image');
subplot 122,imshow(b),title('HORIZONTAL OR ROW FLIP');
```

OUTPUT:



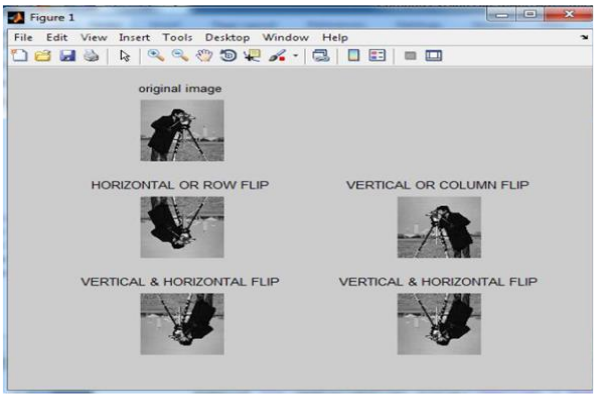
Activities:

1. Write a **MATLAB** code to flip image around the horizontal and vertical line using built-in function.

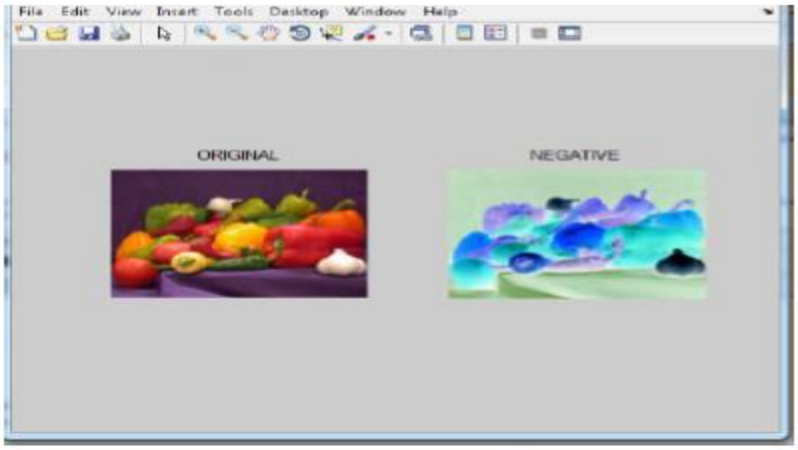
2. Write a **MATLAB** code to find the negative of Image 'peppers.png'.

3. Write a **MATLAB** code to rotate the image for 70 degree.

Screen shots for flipping Images



Screen shots for Image Negative



Screen shots for Image Rotation

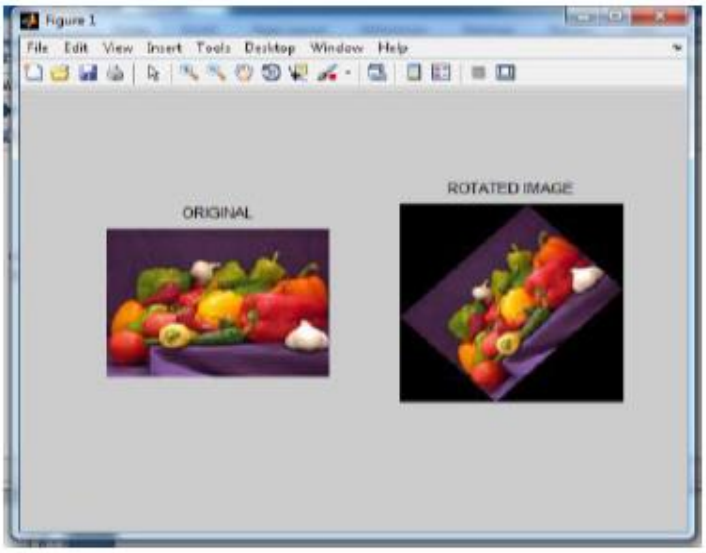


Figure 1
File Edit View Insert Tools Desktop Window Help
ORIGINAL
ROTATED IMAGE

Lab Session #4/ Exercise #4 (Week #4)

Histogram Equalization

The process of adjusting intensity values can be done automatically using *histogram equalization*. Histogram equalization involves transforming the intensity values so that the histogram of the output image approximately matches a specified histogram. By default, the histogram equalization function,

Program Outcome:

Display the Histogram of an Image.

Display the contrast-adjusted image and its new histogram.

Description:

HIST Histogram.

N = HIST(Y) bins the elements of Y into 10 equally spaced containers and returns the number of elements in each container. If Y is a matrix, HIST works down the columns.

N = HIST(Y,M), where M is a scalar, uses M bins.

N = HIST(Y,X), where X is a vector, returns the distribution of Y among bins with centers specified by X. The first bin includes data between -inf and the first center and the last bin includes data between the last bin and inf. Note: Use HISTC if it is more natural to specify bin edges instead.

Exercises (To be solved in the Lab):

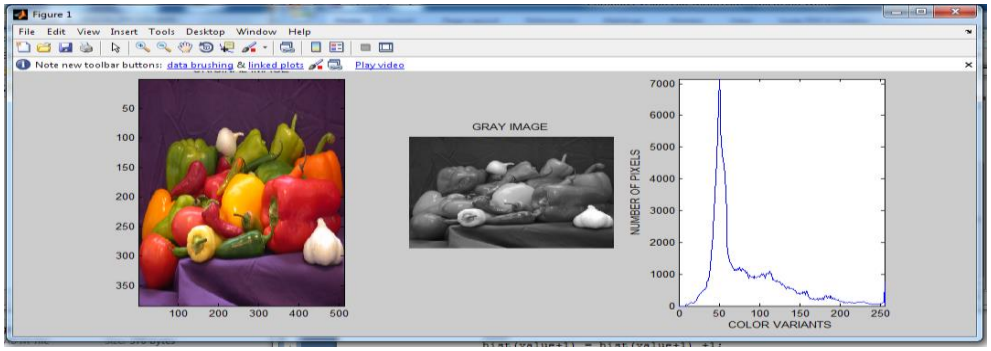
3) Write a program to determine the Histogram of an Image

PROGRAM:

```
close all;
clear all;
org = imread('peppers.png');
colormap(gray);
subplot 131; imagesc(org);title('ORIGINAL IMAGE');
orggray = rgb2gray(org);
subplot 132; imshow(orggray);title('GRAY IMAGE');
hist = zeros(256,1);
c = size(org);
for i = 1: c(1)
    for j = 1:c(2)
        value = round(orggray(i,j));
        if value < 0 value = 0;
        elseif value > 255 value = 255;
        end;
        hist(value+1) = hist(value+1) +1;
```

```
    end;
end;
subplot 133; plot(hist);
axis([0,255,0,max(hist)]);
xlabel('COLOR VARIANTS');
ylabel('NUMBER OF PIXELS');
```

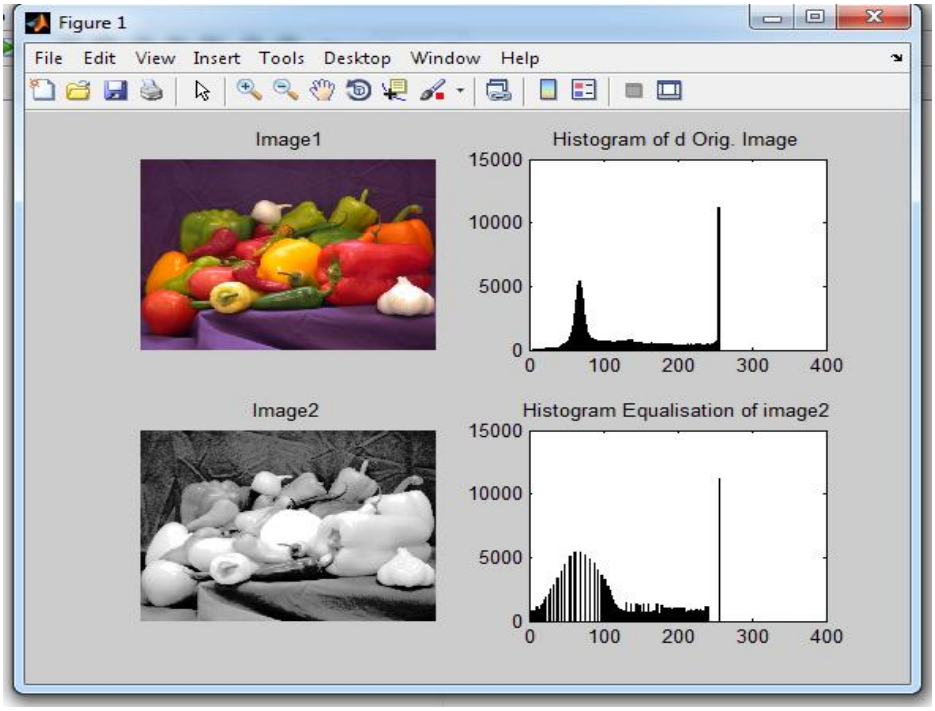
OUTPUT:



Activity:

4. Write a **MATLAB** code to show Histogram Equalization.

Screen shots for histogram equilization



Lab Session #5/ Exercise #5 (Week #5)

The **RGB color model** is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.

Program Outcome:

The image consist the color component of RGB

Description:

An RGB MATLAB array can be of class double, uint8, or uint16. In an RGB array of class double, each color component is a value between 0 and 1. A pixel whose color components are (0,0,0) is displayed as black, and a pixel whose color components are (1,1,1) is displayed as white. The three color components for each pixel are stored along the third dimension of the data array. For example, the red, green, and blue color components of the pixel (10,5) are stored in RGB(10,5,1), RGB(10,5,2), and RGB(10,5,3), respectively.

To display the true color image RGB, use the image function:

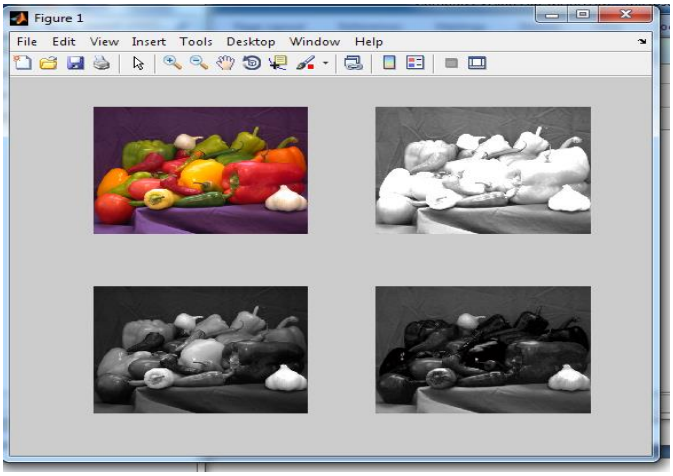
image(RGB)

Exercises (To be solved in the Lab):
4) Write a program to show the color component of an Image

PROGRAM:

```
clear all;
close all;
x = imread('peppers.png');
red = x(:,:,1);
green = x(:,:,2);
blue = x(:,:,3);
subplot 221;imshow(x);
subplot 222;imshow(red+green);
subplot 223;imshow(green);
subplot 224;imshow(blue);
```

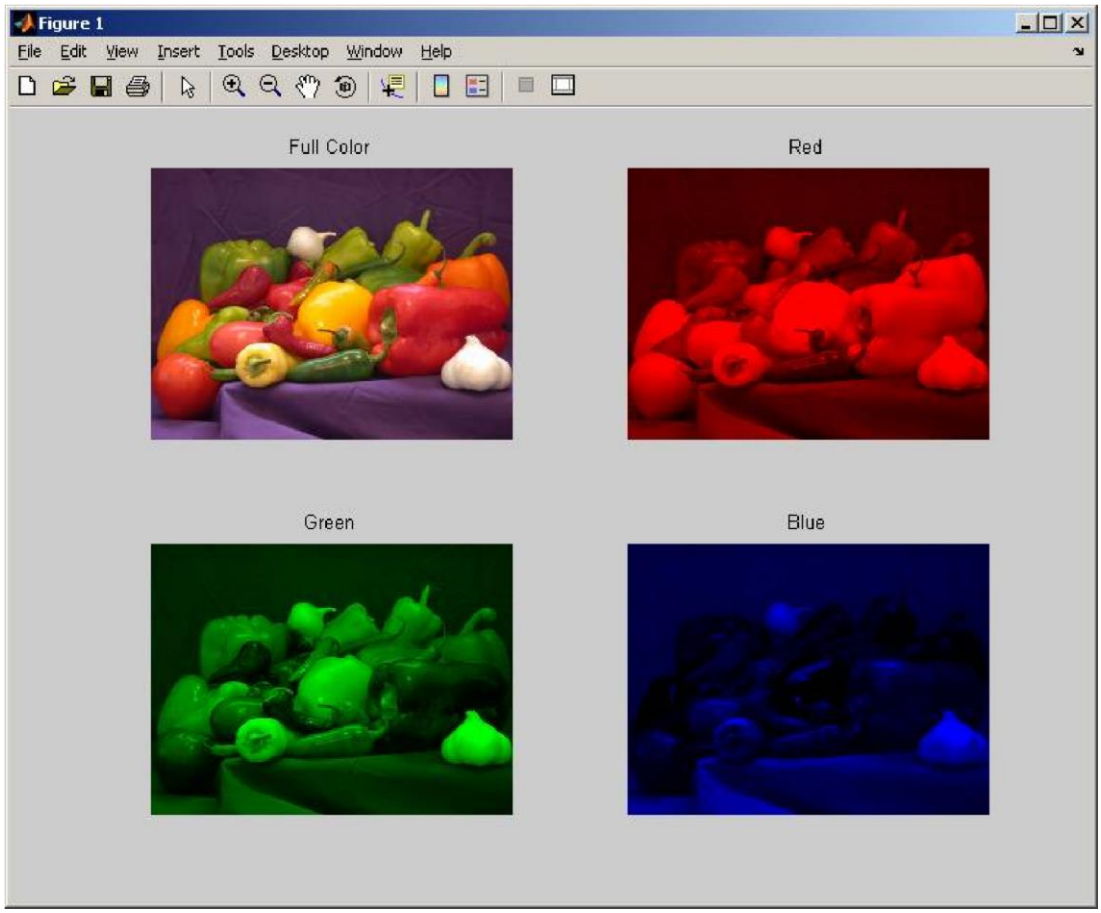
OUTPUT:



Activity:

5. Write a **MATLAB** code to Splitting a Color Image into its 3 RGB Channels.

Screen Shots for Splitting a Color Image into 3 RGB Channels



Lab Session #6/Exercise 6 (Week #6)

The contrast of an image is a measure of its dynamic range, or the "spread" of its histogram. The dynamic range of an image is defined to be the entire range of intensity values contained within an image, or put a simpler way, the maximum pixel value minus the minimum pixel value. An 8-bit image at full contrast has a dynamic range of 28-l=255, with anything less than that yielding a lower contrast image.

Program Outcome:

The resulted image will be enhanced by contrast stretching.

Description:

CONTRAST Gray scale color map to enhance image contrast.

CMAP = CONTRAST(X,M) returns a gray contrast enhancing color, map that is a Mby-3 matrix with 3 identical columns, so that IMAGE(X) . COLORMAP(CMAP) has a roughly equi-distributed gray scale histogram.

If M is omitted, the current colormap length is used.

CONTRAST works best when the image colors are ordered by intensity.

Algorithm:

INPUT: q-bit image I

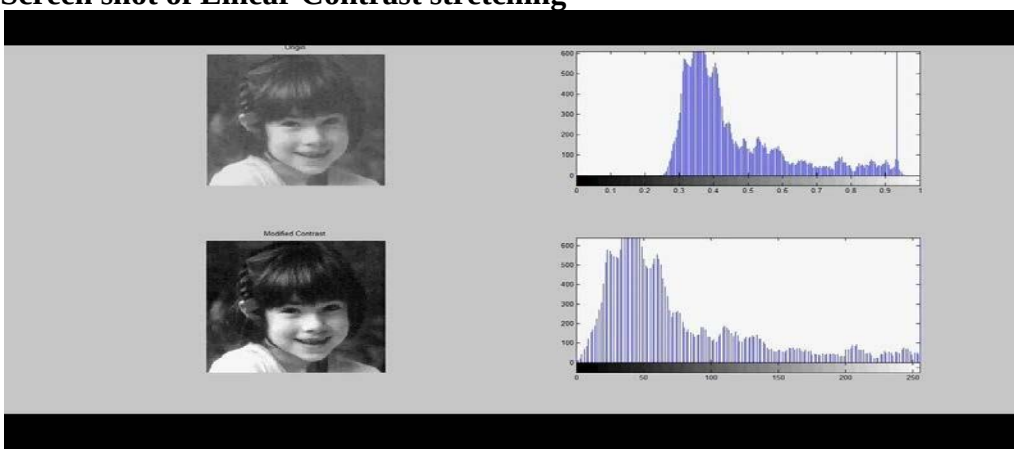
Step1: Find M1 and M2 respectively as Minimum and Maximum of the Image

Step2: Apply *Output(i,j) =(input(i,j)-m1)*255/(m2-m1)*
Step2: Display the output Image

Activity:

6. Write a **MATLAB** code to find Contrast Stretching of any Image.

Screen shot of Linear Contrast stretching



Lab Session #7 / Exercise #7 (Week #7)

Subplot divides the current figure into rectangular panes that are numbered row wise. Each pane contains an axes object which you can manipulate using Axes Properties. Subsequent plots are output to the current pane.

Program Outcome:

The multiple images will be displayed in one plot.

Description:

SUBIMAGE Display multiple images in single figure.

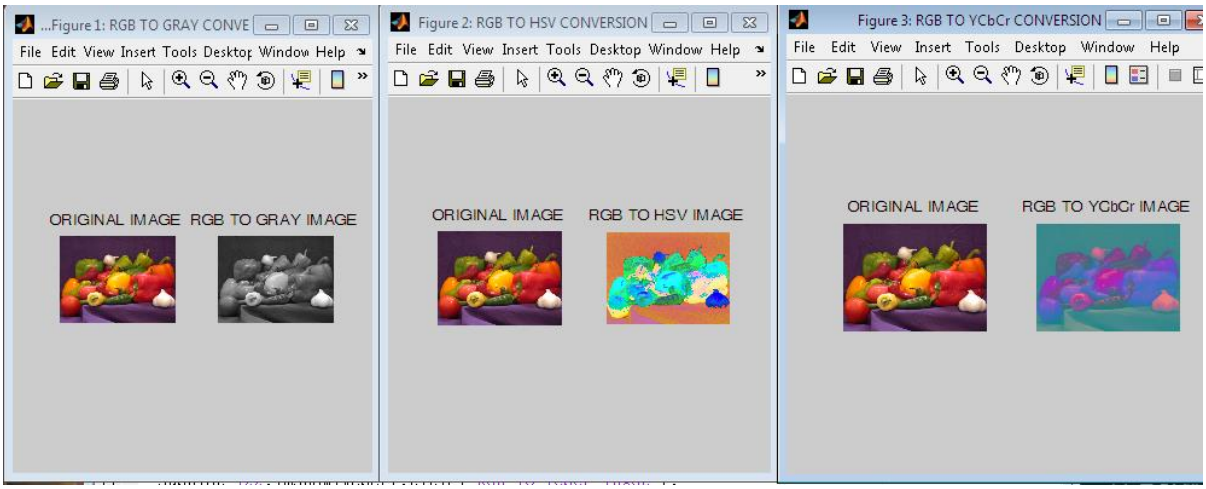
You can use SUBIMAGE in conjunction with SUBPLOT to create figures with multiple images, even if the images have different colormaps. SUBIMAGE works by converting images to truecolor for display purposes, thus avoiding colormap conflicts.

SUBIMAGE(X,MAP) displays the indexed image X with colormap MAP in the current axes.
SUBIMAGE(I) displays the intensity image I in the current axes.
SUBIMAGE(BW) displays the binary image BW in the current axes.
SUBIMAGE(RGB) displays the truecolor image RGB in the current axes.
SUBIMAGE(x,y,...) displays an image with non default spatial coordinates.
H = SUBIMAGE(...) returns a handle to the image object.

Activity:

7. Write a **MATLAB** code to do different color conversion and display the image in different window using figure and subplot.

Screen shots for Figure and subplot



Lab Session #8 / Exercise #8 (Week #8)

Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Program Outcome:

Detecting the edges of the image by using gradient and convolution operations.

Description:

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image.

To find edges, you can use the edge function. This function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

- Places where the first derivative of the intensity is larger in magnitude than some threshold
- Places where the second derivative of the intensity has a zero crossing

Edge provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal edges, vertical edges, or both. edge returns a binary image containing 1's where edges are found and 0's elsewhere. [FX,FY] = GRADIENT(F) returns the numerical gradient of the matrix F. FX corresponds to dF/dx, the differences in x

(horizontal) direction. FY corresponds to dF/dy, the differences in y (vertical) direction. The spacing between points in each direction is assumed to be one. When F is a vector, DF = GRADIENT(F)is the 1-D gradient.

C = conv2(A,B) computes the two-dimensional convolution of matrices A and B. If one of these matrices describes a two-dimensional finite impulse response (FIR) filter, the other matrix is filtered in two dimensions. The size of C is determined as follows: if [ma,na] = size(A), [mb,nb] = size(B), and [mc,nc] = size(C), then mc = max([ma+mb1,ma,mb]) and nc = max([na+nb-1,na,nb]).

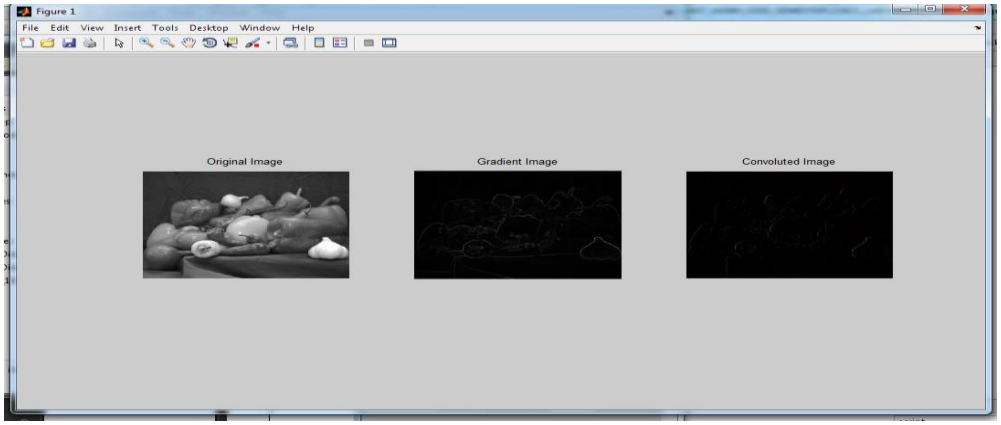
Exercises (To be solved in Lab):

5) Write a program for Edge detection with gradient and convolution of an Image

PROGRAM:

```
clc; clear all; close all;
%hat = imread('peppers.png');
hat = imread('cameraman.tif');
hat = double(rgb2gray(hat));
gradkernel = [-1 0 1];
dx=abs(conv2(hat,gradkernel,'same'));
[dx, dy] = gradient(hat);
gradmag = sqrt(dx.^2 + dy.^2);
subplot 131, imshow(uint8(hat)), title ('Original Image');
subplot 132, imshow(uint8(gradmag)), title ('Gradient Image');
subplot 133, imshow(uint8(dx)), title ('Convoluted Image');
```

OUTPUT:



Lab Session #9 / Exercise #9 (Week #9)

Mean filtering or Average filtering is easy to implement. It is used as a method of smoothing images, reducing the amount of intensity variation between one pixel and the next resulting in reducing noise in images.

Program Outcome:

The resulted image is a smoothing image reducing the noise.

Description:

The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3×33×3 square kernel is used, as shown below:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Exercises (To be solved in Lab):

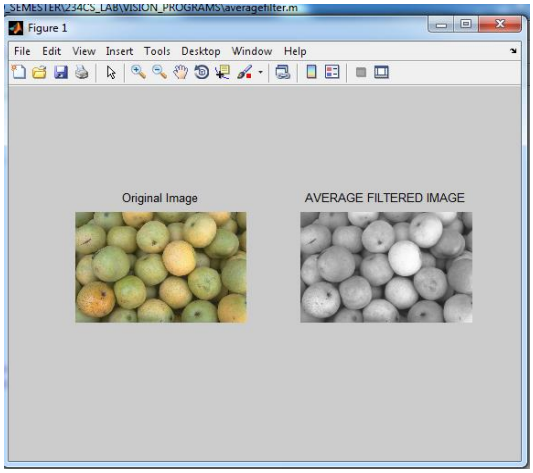
6) Write a program to filter the image by using Average filter.

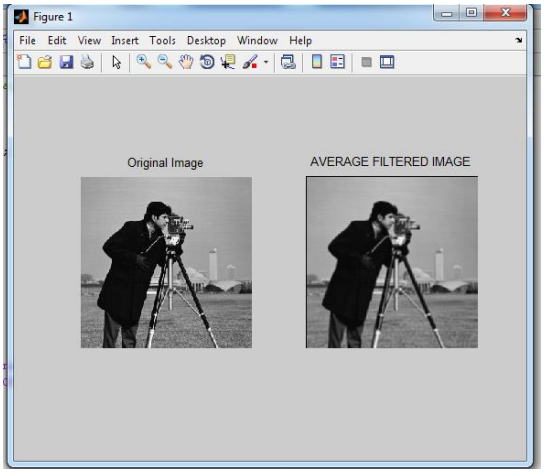
PROGRAM:

```
%Program to apply Average Filter on an Image
clear all;
close all;
%a=im2double(imread('pears.png'));
a=im2double(imread('cameraman.tif'));
c = size(a);
for i=2:c(1)-1
for j=2:c(2)-1
s=0;
for k=-1:1
for l=-1:1
s=s+a(i+k,j+l);
end
end
b(i,j)=s/9;
end
end
subplot 121,imshow(a); title ('Original Image');
```

```
subplot 122,imshow(b); title ('AVERAGE FILTERED IMAGE');
```

OUTPUT:

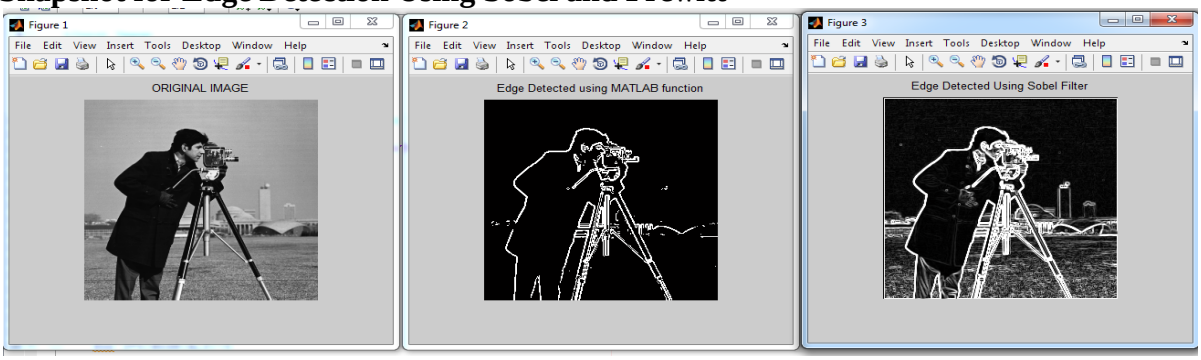




Activities:

8. Write a **MATLAB** code to find the edges of an image using Sobel filter.
9. Write a **MATLAB** code to apply Prewitt method to find the edges of an image.

Snapshot for Edge Detection Using Sobel and Prewitt



Lab Session #9 / Exercise #9 (Week #9)

Median filters are useful in removing impulse noise (also known as salt and pepper noise Salt=255, Pepper=0 gray levels).

Program Outcome:

The impulse corrupted image will be corrected using median Filter using built in and user defined function.

Description:

a. Consider a matrix $A=\begin{bmatrix}5 & 6 & 9\\ 2 & 5 & 3\\ 8 & 1 & 2\end{bmatrix}$

b. Now pad the matrix with zeros on all sides

$$A=\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 5 & 6 & 9 & 0\\ 0 & 2 & 5 & 3 & 0\\ 0 & 8 & 1 & 2 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

c. Consider a window of size 3 by 3. The window can be of any size. Starting from matrix A(1,1) , place the window.

$$\text{Window}=\begin{bmatrix}0 & 0 & 0\\ 0 & 5 & 6\\ 0 & 2 & 5\end{bmatrix}$$

d. $$\text{window}=\begin{bmatrix}0 & 0 & 0\\ 0 & (5) & 6\\ 0 & 2 & 5\end{bmatrix}$$

The value to be changed is the middle element.[Value of A(2,2)]

e. Sort the window matrix : $\begin{bmatrix}0 & 0 & 0\\ 0 & (0) & 2\\ 5 & 5 & 6\end{bmatrix}$

f. After sorting, the output matrix is placed with a value of 0 at (2,2) pixel position. The value of the output pixel is found using the median of the neighborhood pixels.

g. This procedure is repeated for all the values in the input matrix by sliding the window to next position ie A(1,2) and so on.

h. The output matrix: $\begin{bmatrix}0 & 3 & 0\\ 2 & 5 & 2\\ 0 & 2 & 0\end{bmatrix}$

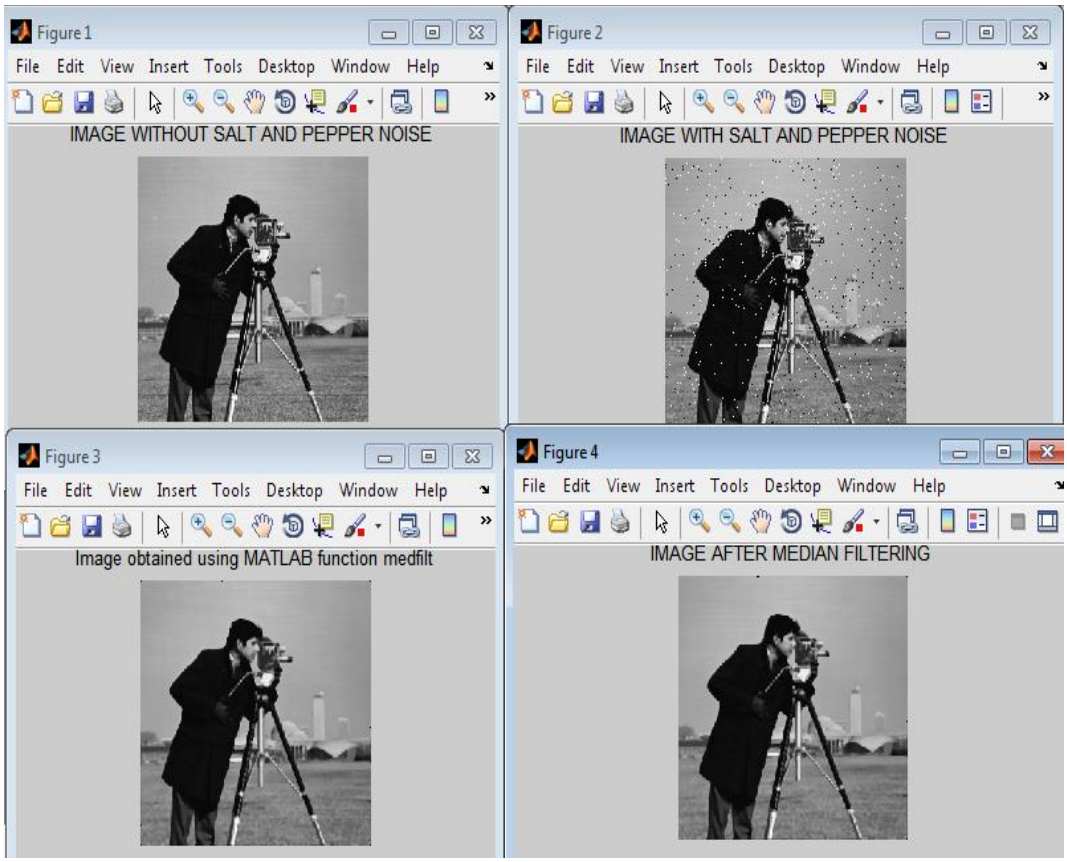
Exercises (To be solved in Lab):

7) Write a program to filter the image by using Median filter. (or)
Write a MATLAB Code to perform 2D Median filter using **medfilt2** function and without using medfilt2 function.

PROGRAM:

```
close all
clear all;

%Input Image
I = imread('cameraman.tif');
A = imnoise(I,'salt & pepper',0.02);
figure(1),imshow(I),title('IMAGE WITHOUT SALT AND PEPPER NOISE');
figure(2),imshow(A),title('IMAGE WITH SALT AND PEPPER NOISE');

%Image obtained using MATLAB BUILT_IN_function 'medfilt'
K = medfilt2(A);
figure(3),imshow(K);title('Image obtained using MATLAB function medfilt')

%PAD THE MATRIX WITH ZEROS ON ALL SIDES
A=padarray(A,[1,1]);
A=double(A);

%LET THE WINDOW BE AN ARRAY STORE THE 3-by-3 NEIGHBOUR VALUES IN THE
ARRAY
      %SORT AND FIND THE MIDDLE ELEMENT

for i= 1:size(A,1)-2
    for j=1:size(A,2)-2
        window=zeros(9,1);
        inc=1;
        for x=1:3
            for y=1:3
                window(inc)=A(i+x-1,j+y-1);
                inc=inc+1;
            end
        end

        med=sort(window);
        %PLACE THE MEDIAN ELEMENT IN THE OUTPUT MATRIX
        B(i,j)=med(5);

    end
end
%CONVERT THE OUTPUT MATRIX TO 0-255 RANGE IMAGE TYPE
B=uint8(B);
figure(4),imshow(B);title('IMAGE AFTER MEDIAN FILTERING')
```

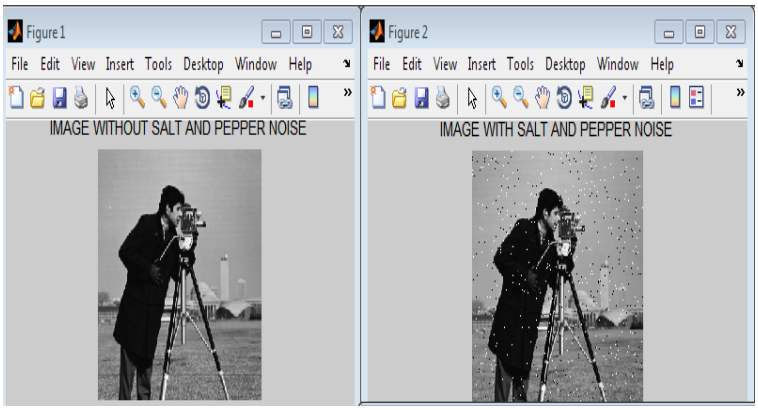
OUTPUT:



Activity:

10. Write a **MATLAB** code to Insert salt and pepper noise to a given image.

Snapshot for Inserting Salt and pepper noise



Lab Session #9 / Exercise #9 (Week #9)

Laplacian filter is a second derivative edge detector operator. Laplacian is more sensitive to noise than sobel and prewitt. Laplacian calculate the difference of center point to the surrounding pixels .

Program Outcome

Image reduces the noise by using Laplacian filter.

Description:

The derivative operator Laplacian for an Image is defined as

$$\Delta^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$ ➔ A

For X-direction,

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$ ➔ B

or Y-direction,

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$ ➔ C

By substituting, Equations in Fig.B and Fig.C in Fig.A, we obtain the following equation

$$\Delta^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4(f(x,y)$$ ➔ D

The equation represented in terms of Mask:

0	1	0
1	-4	1
0	1	0

When the diagonals also considered then the equation becomes,

$$\Delta^2 f(x,y) = f(x+1,y)+f(x-1,y)+f(x,y+1)$$
$$+f(x,y-1)+f(x-1,y-1)+f(x+1,y-1)$$
$$+f(x-1,y+1)+f(x+1,y+1)-8(f(x,y)) \rightarrow E$$

The Mask representation of the above equation,

1	1	1
1	-8	1
1	1	1

Now let's discuss further how image sharpening is done using Laplacian.

Equation:

$$g(x,y) = f(x,y) + c\left[\Delta^2 f(x,y)\right] \rightarrow F$$

Where f(x,y) is the input image
g(x,y) is the sharpened image and
c= -1 for the above mentioned filter masks.(fig.D and fig.E)

The Laplacian derivative equation has produced grayish edge lines and other areas are made dark(background)

```
%Sharpenend Image Refer Equation in Fig.F
B=I1-I;
figure,imshow(B);title('Sharpened Image');
```

Exercises (To be solved in Lab):

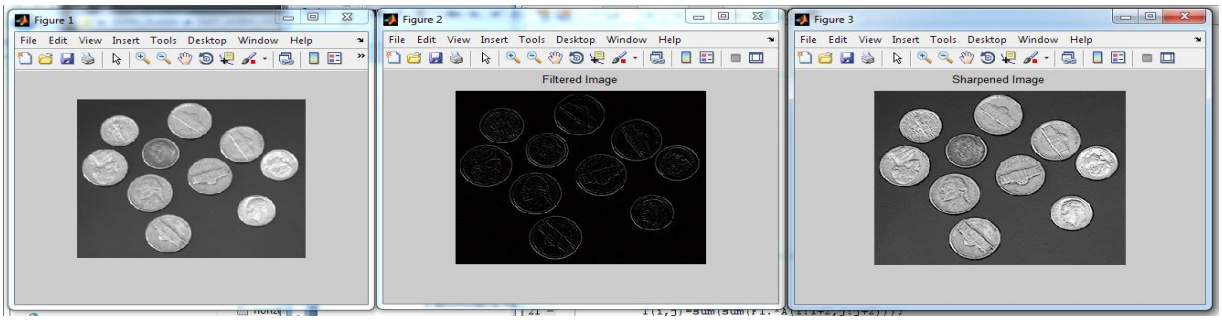
8) Write a MATLAB code to apply LAPLACIAN Filter mask and sharpened the image.

PROGRAM:

```
close all
clear all;

%Input Image
A=imread('coins.png');
figure,imshow(A);

%Preallocate the matrices with zeros
I1=A;
I=zeros(size(A));
I2=zeros(size(A));

%Filter Masks
F1=[0 1 0;1 -4 1; 0 1 0];
F2=[1 1 1;1 -8 1; 1 1 1];

%Padarray with zeros
A=padarray(A,[1,1]);
A=double(A);
```

```
%Implementation of the equation in Fig.D
for i=1:size(A,1)-2
    for j=1:size(A,2)-2
       I(i,j)=sum(sum(F1.*A(i:i+2,j:j+2)));
    end
end
I=uint8(I);
figure,imshow(I);title('Filtered Image');
%Sharpenend Image Refer Equation in Fig.F
B=I1-I;
figure,imshow(B);title('Sharpened Image');
```

OUTPUT:



Lab Session #9 / Exercise #9 (Week #9)

Image thresholding is simple in application for image segmentation.

Program Outcome:

The image extract the object from the background using threshold application of Image Segmentation.

Description:

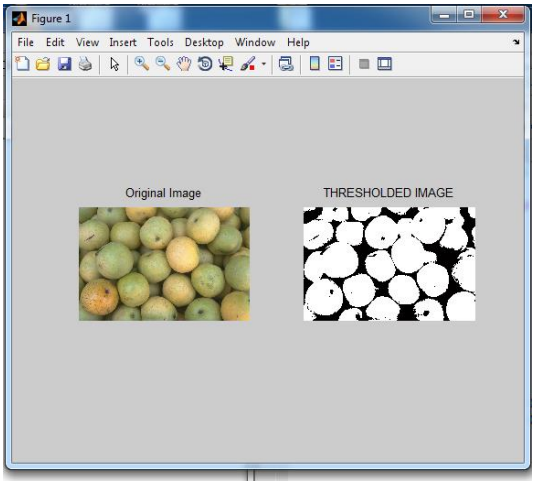
One way to extract the objects from the background by Thresholding is to select a threshold T that separates these modes. Then any point (x, y) for which f(x, y) >T is called an object point. otherwise, the point is called a background point.

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T. \end{cases}$$

Activity:

11. Write a **MATLAB** code to extract the object from the background using threshold application of Image Segmentation.

Snapshot for Threshold



Lab Session #9 / Exercise #9 (Week #9)

Quad tree decomposition is an analysis technique that involves subdividing an image into blocks that are more homogeneous than the image itself. This technique reveals information about the structure of the image. It is also useful as the first step in adaptive compression algorithms.

Program Output:

The resulted image is decomposition of blocks.

Description:

A **quad tree** is a tree data structure in which each internal node has exactly four children. Quad trees are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes.

You can perform quadtree decomposition using the qtdecomp function. This function works by dividing a square image into four equal-sized square blocks, and then testing each block to see if it meets some criterion of homogeneity (e.g., if all the pixels in the block are within a specific dynamic range). If a block meets the criterion, it is not divided any further. If it does not meet the criterion, it is subdivided again into four blocks, and the test criterion is applied to those blocks. This process is repeated iteratively until each block meets the criterion. The result might have blocks of several different sizes. Blocks can be as small as 1-by-1, unless you specify otherwise.

qtdecomp returns the quadtree decomposition as a sparse matrix, the same size as I. The nonzero elements represent the upper left corners of the blocks. The value of each nonzero element indicates the block size.

S = qtdecomp(I) performs a quadtree decomposition on the intensity image I and returns the quadtree structure in the sparse matrix S. If S(k,m) is nonzero, then (k,m) is the upper left corner of a block in the decomposition, and the size of the block is given by S(k,m). By default, qtdecomp splits a block unless all elements in the block are equal.

S = qtdecomp(I, threshold) splits a block if the maximum value of the block elements minus the minimum value of the block elements is greater than threshold. threshold is specified as a value between 0 and 1, even if I is of class uint8 or uint16. If I is uint8, the threshold value you supply is multiplied by 255 to determine the actual threshold to use; if I is uint16, the threshold value you supply is multiplied by 65535.

Activity:

12. Write a **MATLAB** code to segregate a digital image into non-overlapping rectangular blocks of homogenous gray levels using Quad Tree Decomposition.

Snapshot for Quad tree decomposition