Practical 1

Aim: To Understand And Implement Basic Commands Of MATLAB.

Image Acquisition Toolbox

Device Connection

clear	Clear image acquisition object from MATLAB workspace
delete	Remove image acquisition object from memory
disp	Display method for image acquisition objects
start	Obtain exclusive use of image acquisition device
stop	Stop video input object

Image Preview and Device Configuration

get	Return image acquisition object properties
imaqhelp	Image acquisition object function and property help
imaqtool	Launch Image Acquisition Tool
propinfo	Property characteristics for image acquisition objects
set	Configure or display image acquisition object properties

Image Data Acquisition
Acquisition Using Any Hardware

Functions

imaqtool	Launch Image Acquisition Tool
getdata	Acquired image frames to MATLAB workspace
peekdata	Most recently acquired image data
getsnapshot	Immediately return single image frame
set	Configure or display image acquisition object properties
start	Obtain exclusive use of image acquisition device
islogging	Determine whether video input object is logging
isrunning	Determine whether video input object is running
isvalid	Determine whether image acquisition object is associated with image acquisition device
wait	Wait until image acquisition object stops running or logging
stop	Stop video input object
clear	Clear image acquisition object from MATLAB workspace
delete	Remove image acquisition object from memory
flushdata	Remove data from memory buffer used to store acquired image frames
load	Load image acquisition object into MATLAB workspace
save	Save image acquisition objects to MAT-file
trigger	Initiate data logging

Image Processing Toolbox

- **Import, Export, and Conversion**
  - Image data import and export, conversion of image types and classes
- **Basic Import and Export**
  - Read and write image data, get information about contents of image files

Functions

imread	Read image from graphics file
imwrite	Write image to graphics file
imfinfo	Information about graphics file

Reading Image Data

To import an image from any supported graphics image file format, in any of the supported bit depths, use the imread function. This example reads a truecolor image into the MATLAB workspace as the variable RGB.

RGB = imread('football.jpg');

If the image file format uses 8-bit pixels, imread stores the data in the workspace as a uint8 array. For file formats that support 16-bit data, such as PNG and TIFF, imread creates a uint16 array.

Writing Image Data to Files

To export image data from the MATLAB workspace to a graphics file in one of the supported graphics file formats, use the imwrite function. When using imwrite, you specify the MATLAB variable name and the name of the file. If you include an extension in the filename, imwrite attempts to infer the desired file format from it. For example, the file extension .jpg infers the Joint Photographic Experts Group (JPEG) format. You can also specify the format explicitly as an argument to imwrite.

This example loads the indexed image X from a MAT-file, clown.mat, along with the associated colormap map, and then exports the image as a bitmap (BMP) file.

load clown

whos

Your output appears as shown:

Name	Size	Bytes Class	Attributes
X	200x320	512000	double
caption	2x1	4	char
map	81x3	1944	double

Export the image as a bitmap file:

imwrite(X,map,'clown.bmp')

Specify Format-Specific Parameters

When using imwrite with some graphics formats, you can specify additional format-specific parameters. For example, with PNG files, you can specify the bit depth. This example writes a grayscale image I to a 4-bit PNG file.

imwrite(I,'clown.png','BitDepth',4);

This example writes an image A to a JPEG file, using an additional parameter to specify the compression quality parameter.

imwrite(A, 'myfile.jpg', 'Quality', 100);

For more information about these additional format-specific syntaxes, see the imwrite reference page.

Image Type Conversion

Convert between the image types, such as RGB (truecolor), binary, grayscale, and indexed.

Functions

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
ind2rgb	Convert indexed image to RGB image
im2bw	Convert image to binary image, based on threshold

gray2ind

Convert grayscale or binary image to indexed image

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

[X, map] = gray2ind(BW,n)

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

[X, map] = gray2ind(BW,n) converts the binary image BW to an indexed image X. n specifies the size of the colormap, gray(n). If n is omitted, it defaults to 2.

gray2ind scales and then rounds the intensity image to produce an equivalent indexed image.

rgb2gray
Convert RGB image or colormap to grayscalecollapse all in page

Syntax

I = rgb2gray(RGB) example
newmap = rgb2gray(map) example

Description
I = rgb2gray(RGB) converts the truecolor image RGB to the grayscale intensity image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance. If you have Parallel Computing Toolbox™ installed, rgb2gray can perform this

newmap = rgb2gray(map) returns a grayscale colormap equivalent to map.

Display and Exploration
Interactive tools for image display and exploration

Basic Display
View image data, view multi-frame images (movies), set display preferences

Functions

imshow	Display image
montage	Display multiple image frames as rectangular montage
subimage	Display multiple images in single figure

imshow

Display imagecollapse all in page

Syntax

imshow(I) example
imshow(I,RI) example
imshow(X,map) example
imshow(X,RX,map)
imshow(filename) example
imshow(___,Name,Value...)
imshow(gpuarrayIM,___) example
imshow(I,[low high]) example
imshow(___,Name,Value,...)
himage = imshow(___)

Description
imshow(I) displays the image I in a Handle Graphics® figure, where I is a grayscale, RGB (truecolor), or binary image. For binary images, imshow displays pixels with the value 0 (zero) as black and 1 as white.

imshow(I,RI) displays the image I with associated 2-D spatial referencing object RI.

imshow(X,map) displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.

imshow(X,RX,map) displays the indexed image X with associated 2-D spatial referencing object RX and colormap MAP.

imshow(filename) displays the image stored in the graphics file specified by the text string filename.

imshow(___,Name,Value...) displays the image, specifying additional options with one or more Name,Value pair arguments, using any of the previous syntaxes.

imshow(gpuarrayIM,___) displays the image contained in a gpuArray. This syntax requires the Parallel Computing Toolbox.

imshow(I,[low high]) displays the grayscale image I, specifying the display range as a two-element vector, [low high]. For more information, see theDisplayRange parameter.

imshow(___,Name,Value,...) displays an image, using name-value pairs to control aspects of the operation.

himage = imshow(___) returns the handle to the image object created by imshow.

Interactive Exploration with the Image Viewer App
View and explore images using the Image Viewer app, set display preferences

Functions

imtool	Image Viewer app
imageinfo	Image Information tool
imcontrast	Adjust Contrast tool
imdisplayrange	Display Range tool
imdistline	Distance tool
impixelinfo	Pixel Information tool
impixelinfoval	Pixel Information tool without text label
impixelregion	Pixel Region tool
immagbox	Magnification box for scroll panel
imoverview	Overview tool for image displayed in scroll panel

imtool
Image Viewer app

Syntax

imtool
imtool(I)
imtool(I,[low high])
imtool(RGB)
imtool(BW)
imtool(X,map)
imtool(filename)
imtool close all

Description
imtool opens the Image Viewer app in an empty state. Use the File menu options Open or Import from Workspace to choose an image for display.

imtool(I) displays the grayscale image I in the Image Viewer.

imtool(I,[low high]) displays the grayscale image I in the Image Viewer, specifying the display range for I in the vector [low high]. The value low (and any value less than low) is displayed as black, the value high (and any value greater than high) is displayed as white. Values in between are displayed as intermediate shades of gray. The Image Viewer uses the default number of gray levels. If you use an empty matrix ([]) for [low high], the Image Viewer uses [min(I(:)) max(I(:))]; the minimum value in I is displayed as black, and the maximum value is displayed as white.

imtool(RGB) displays the truecolor image RGB in the Image Viewer.

imtool(BW) displays the binary image BW in the Image Viewer. Pixel values of 0 display as black; pixel values of 1 display as white.

imtool(X,map) displays the indexed image X with colormap map in the Image Viewer.

imtool(filename) displays the image contained in the graphics file filename in the Image Viewer. The file must contain an image that can be read by imread or dicomread or a reduced resolution dataset (R-Set) created by rsetwrite. If the file contains multiple images, the first one is displayed. The file must be in the current directory or on the MATLAB path.

imtool close all closes all open Image Viewers.

Geometric Transformation, Spatial Referencing, and Image Registration
Scale, rotate, perform other N-D transformations, provide spatial information, align images using automatic or control point registration

Geometric Transformations
Resize, rotate, and crop images; perform geometric transformation of multidimensional arrays

Functions

imcrop	Crop image
imresize	Resize image
imrotate	Rotate image
imtranslate	Translate image
impyramid	Image pyramid reduction and expansion
imwarp	Apply geometric transformation to image
makeresampler	Create resampling structure
tformfwd	Apply forward spatial transformation
tforminv	Apply inverse spatial transformation
checkerboard	Create checkerboard image

imresize
Resize image

Syntax

B = imresize(A, scale)
gpuarrayB = imresize(gpuarrayA,scale)
B = imresize(A, [numrowsnumcols])

Description
B = imresize(A, scale) returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If scale is between 0 and 1.0, B is smaller than A. If scale is greater than 1.0, B is larger than A. By default, imresize uses bicubic interpolation.

gpuarrayB = imresize(gpuarrayA,scale) performs the resize operation on a GPU. The input image and the output image are gpuArrays. When used with gpuArrays, imresize only supports cubic interpolation and always performs antialiasing. This syntax requires the Parallel Computing Toolbox™.

B = imresize(A, [numrowsnumcols]) returns image B that has the number of rows and columns specified by [numrowsnumcols]. Either numrows or numcols may be NaN, in which case imresize computes the number of rows or columns automatically to preserve the image aspect ratio.

Spatial Referencing
Associate spatial information with an image, use this information in image processing operations

Functions

imwarp	Apply geometric transformation to image
imregister	Intensity-based image registration
imregtform	Estimate geometric transformation that aligns two 2-D or 3-D images
imshow	Display image
imshowpair	Compare differences between images
imfuse	Composite of two images

imregister
Intensity-based image registration collapse all in page

Syntax
moving_reg = imregister(moving,fixed,transformType,optimizer,metric) example
[moving_reg,R_reg] = imregister(moving,Rmoving,fixed,Rfixed,transformType,optimizer,metric)

Description
moving_reg = imregister(moving,fixed,transformType,optimizer,metric) transforms the 2-D or 3-D image, moving, so that it is registered with the reference image, fixed. Both moving and fixed images must be of the same dimensionality, either 2-D or 3-D. transformType is a character string that defines the type of transformation to perform. optimizer is an object that describes the method for optimizing the metric and metric is an object that defines the quantitative measure of similarity between the images to optimize. Returns the aligned image, moving_reg.

[moving_reg,R_reg] = imregister(moving,Rmoving,fixed,Rfixed,transformType,optimizer,metric) transforms the spatially referenced image moving so that it is registered with the spatially referenced image fixed. Rmoving and Rfixed are spatial referencing objects that describe the world coordinate limits and resolution of moving and fixed.

Image Enhancement
Contrast adjustment, morphological filtering, deblurring, and other image enhancement tools

Contrast Adjustment
Contrast adjustment, histogram equalization, decorrelation stretching

Functions

imadjust	Adjust image intensity values or color map
imcontrast	Adjust Contrast tool
imsharpen	Sharpen image using unsharp masking
histeq	Enhance contrast using histogram equalization

adapthisteq	Contrast-limited adaptive histogram equalization (CLAHE)
stretchlim	Find limits to contrast stretch image
intlut	Convert integer values using lookup table
imnoise	Add noise to image

imadjust
Adjust image intensity values or colormapcollapse all in page

Syntax

J = imadjust(I) example
J = imadjust(I,[low_in; high_in],[low_out; high_out])
J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma)
newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma)

Description
J = imadjust(I) maps the intensity values in grayscale image I to new values in J such that 1% of data is saturated at low and high intensities of I. This increases the contrast of the output image J. This syntax is equivalent to imadjust(I,stretchlim(I)).

This function supports code generation

J = imadjust(I,[low_in; high_in],[low_out; high_out]) maps the values in I to new values in J such that values between low_in and high_in map to values between low_out and high_out.

Note If high_out is less than low_out, imadjust reverses the output image, as in a photographic negative.

J = imadjust(I,[low_in; high_in],[low_out; high_out],gamma) maps the values in I to new values in J, where gamma specifies the shape of the curve describing the relationship between the values in I and J.

newmap = imadjust(map,[low_in; high_in],[low_out; high_out],gamma) transforms the m-by-3 array colormap associated with an indexed image. low_in, high_in, low_out, and high_out must be 1-by-3 vectors. gamma can be a 1-by-3 vector that specifies a unique gamma value for each channel or a scalar that specifies the value used for all three channels. The rescaled colormapnewmap is the same size as map.

imnoise
Add noise to imagecollapse all in page

Syntax

J = imnoise(I,type)
J = imnoise(I,type,parameters)
J = imnoise(I,'gaussian',M,V)
J = imnoise(I,'localvar',V)
J = imnoise(I,'localvar',image_intensity,var)
J = imnoise(I,'poisson')
J = imnoise(I,'salt&pepper',d)
J = imnoise(I,'speckle',v)
gpuarrayJ = imnoise(gpuarrayI,___)

Description

J = imnoise(I,type) adds noise of a given type to the intensity image I. type is a string that specifies any of the following types of noise. Note that certain types of noise support additional parameters. See the related syntax.

Value	**Description**
'gaussian'	Gaussian white noise with constant mean and variance
'localvar'	Zero-mean Gaussian white noise with an intensity-dependent variance
'poisson'	Poisson noise
'salt & pepper'	On and off pixels
'speckle'	Multiplicative noise

J = imnoise(I,type,parameters) Depending on type, you can specify additional parameters to imnoise. All numerical parameters are normalized— they correspond to operations with images with intensities ranging from 0 to 1.

J = imnoise(I,'gaussian',M,V) adds Gaussian white noise of mean m and variance v to the image I. The default is zero mean noise with 0.01 variance.

J = imnoise(I,'localvar',V) adds zero-mean, Gaussian white noise of local variance V to the image I. V is an array of the same size as I.

J = imnoise(I,'localvar',image_intensity,var) adds zero-mean, Gaussian noise to an image I, where the local variance of the noise, var, is a function of the image intensity values in I. The image_intensity and var arguments are vectors of the same size, and plot(image_intensity,var) plots the functional relationship between noise variance and image intensity. The image_intensity vector must contain normalized intensity values ranging from 0 to 1.

J = imnoise(I,'poisson') generates Poisson noise from the data instead of adding artificial noise to the data. If I is double precision, then input pixel values are interpreted as means of Poisson distributions scaled up by 1e12.

J = imnoise(I,'salt&pepper',d) adds salt and pepper noise to the image I, where d is the noise density. This affects approximately d*numel(I) pixels. The default for d is 0.05.

J = imnoise(I,'speckle',v) adds multiplicative noise to the image I, using the equation J = I+n*I, where n is uniformly distributed random noise with mean 0 and variance v. The default for v is 0.04.

Note The mean and variance parameters for 'gaussian', 'localvar', and 'speckle' noise types are always specified as if the image were of class double in the range [0, 1]. If the input image is of class uint8 or uint16, the imnoise function converts the image to double, adds noise according to the specified type and parameters, and then converts the noisy image back to the same class as the input.

gpuarrayJ = imnoise(gpuarrayI,___) adds noise to the gpuArray intensity image gpuarrayI, performing the operation on a GPU. Returns a gpuArray image J of the same class. This syntax requires the Parallel Computing Toolbox™.

Image Filtering
Convolution and correlation, predefined and custom filters, nonlinear filtering, edge-preserving filters.

Functions

imfilter	N-D filtering of multidimensional images
imgaussfilt	2-D Gaussian filtering of images
imgaussfilt3	3-D Gaussian filtering of 3-D images
fspecial	Create predefined 2-D filter
imguidedfilter	Guided filtering of images
normxcorr2	Normalized 2-D cross-correlation
wiener2	2-D adaptive noise-removal filtering
medfilt2	2-D median filtering
ordfilt2	2-D order-statistic filtering
stdfilt	Local standard deviation of image
rangefilt	Local range of image
entropyfilt	Local entropy of grayscale image

imgaussfilt
2-D Gaussian filtering of imagescollapse all in page

Syntax

B = imgaussfilt(A)
B = imgaussfilt(A,sigma)
B = imgaussfilt(___,Name,Value,...)
gpuarrayB = imgaussfilt(gpuarrayA,___) example

Description
B = imgaussfilt(A) filters image A with a 2-D Gaussian smoothing kernel with standard deviation of 0.5. Returns B, the filtered image.

B = imgaussfilt(A,sigma) filters image A with a 2-D Gaussian smoothing kernel with standard deviation specified by sigma.

Morphological Operations
Dilate, erode, reconstruct, and perform other morphological operations

Functions

Bwhitmiss	Binary hit-miss operation
Bwmorph	Morphological operations on binary images
bwulterode	Ultimate erosion
bwareaopen	Remove small objects from binary image
Imbothat	Bottom-hat filtering
imclearborder	Suppress light structures connected to image border
Imclose	Morphologically close image
Imdilate	Dilate image

Imerode	Erode image
imextendedmax	Extended-maxima transform
imextendedmin	Extended-minima transform
Imfill	Fill image regions and holes
Imhmax	H-maxima transform
Imhmin	H-minima transform
imimposemin	Impose minima
Imopen	Morphologically open image
imreconstruct	Morphological reconstruction
imregionalmax	Regional maxima
imregionalmin	Regional minima
imtophat	Top-hat filtering
watershed	Watershed transform

imerode
Erode image

Syntax

IM2 = imerode(IM,SE)
IM2 = imerode(IM,NHOOD)

Description
IM2 = imerode(IM,SE) erodes the grayscale, binary, or packed binary image IM, returning the eroded image IM2. The argument SE is a structuring element object or array of structuring element objects returned by the strel function.

If IM is logical and the structuring element is flat, imerode performs binary erosion; otherwise it performs grayscale erosion. If SE is an array of structuring element objects, imerode performs multiple erosions of the input image, using each structuring element in SE in succession.

imdilate
Dilate imagecollapse all in page

Syntax

IM2 = imdilate(IM,SE)
IM2 = imdilate(IM,NHOOD)

Description
IM2 = imdilate(IM,SE) dilates the grayscale, binary, or packed binary image IM, returning the dilated image, IM2. The argument SE is a structuring element object, or array of structuring element objects, returned by the strel function.

If IM is logical and the structuring element is flat, imdilate performs binary dilation; otherwise, it performs grayscale dilation. If SE is an array of structuring element objects, imdilate performs multiple dilations of the input image, using each structuring element in succession.

This function supports code generation

IM2 = imdilate(IM,NHOOD) dilates the image IM, where NHOOD is a matrix of 0's and 1's that specifies the structuring element neighborhood. This is equivalent to the syntax imdilate(IM,strel(NHOOD)). The imdilate function determines the center element of the neighborhood by floor((size(NHOOD)+1)/2).

Image Analysis
Object Analysis

Functions

bwboundaries	Trace region boundaries in binary image
bwtraceboundary	Trace object in binary image
visboundaries	Plot region boundaries
edge	Find edges in intensity image
imfindcircles	Find circles using circular Hough transform
viscircles	Create circle
corner	Find corner points in image
cornermetric	Create corner metric matrix from image
imgradient	Gradient magnitude and direction of an image
imgradientxy	Directional gradients of an image
hough	Hough transform
houghlines	Extract line segments based on Hough transform
houghpeaks	Identify peaks in Hough transform

edge
Find edges in intensity imagecollapse all in page

Syntax

BW = edge(I)
BW = edge(I,method) example
BW = edge(I,method,threshold)
BW = edge(I,method,threshold,direction)

Description
BW = edge(I) returns a binary image BW containing 1s where the function finds edges in the input image I and 0s elsewhere. By default, edge uses the Sobel edge detection method.

This function supports code generation

BW = edge(I,method) detects edges in image I, where method specifies the edge detection method used.

BW = edge(I,method,threshold) detects edges in image I, where threshold specifies the sensitivity threshold. edge ignores all edges that are not stronger than threshold.

BW = edge(I,method,threshold,direction) detects edges in image I, where direction specifies the direction in which the function looks for edges in the image: horizontally, vertically, or in both directions. Used only with the Sobel and Prewitt methods.

Region and Image Properties
Get information about the objects in an image

Functions

regionprops	Measure properties of image regions
bwarea	Area of objects in binary image
bwareafilt	Extract objects from binary image by size
bwconncomp	Find connected components in binary image
bwconvhull	Generate convex hull image from binary image
bwdist	Distance transform of binary image
bwpropfilt	Extract objects from binary image using properties
bwselect	Select objects in binary image
imhist	Histogram of image data

imhist
Histogram of image datacollapse all in page

Syntax

imhist(I) example
imhist(I,n)
imhist(X,map)
[counts,binLocations] = imhist(I)

Description
imhist(I) calculates the histogram for the intensity image I and displays a plot of the histogram. The number of bins in the histogram is determined by the image type.

This function supports code generation (see Tips).

imhist(I,n) calculates the histogram, where n specifies the number of bins used in the histogram. n also specifies the length of the colorbar displayed at the bottom of the histogram plot.

imhist(X,map) displays a histogram for the indexed image X. This histogram shows the distribution of pixel values above a colorbar of the colormap map. The colormap must be at least as long as the largest index in X. The histogram has one bin for each entry in the colormap.

[counts,binLocations] = imhist(I) returns the histogram counts in counts and the bin locations in binLocations so that stem(binLocations,counts) shows the histogram. For indexed images, imhist returns the histogram counts for each colormap entry. The length of counts is the same as the length of the colormap.

Image Quality
Mean-squared error, peak signal-to-noise ratio, and Structural Similarity Index (SSIM) image quality metrics

Functions

immse	Mean-squared error
psnr	Peak Signal-to-Noise Ratio (PSNR)
ssim	Structural Similarity Index (SSIM) for measuring image quality

Image Segmentation
Segment images

Functions

Activecontour	Segment image into foreground and background using active contour
Imsegfmm	Binary image segmentation using Fast Marching Method
imseggeodesic	Segment image into two or three regions using geodesic distance-based color segmentation
gradientweight	Calculate weights for image pixels based on image gradient
graydiffweight	Calculate weights for image pixels based on grayscale intensity difference

Image Transforms
Perform Fourier, Discrete Cosine, Radon, and Fan-beam transforms

Functions

Bwdist	Distance transform of binary image
bwdistgeodesic	Geodesic distance transform of binary image
Graydist	Gray-weighted distance transform of grayscale image
Hough	Hough transform
dct2	2-D discrete cosine transform
dctmtx	Discrete cosine transform matrix
idct2	2-D inverse discrete cosine transform
fft2	2-D fast Fourier transform
ifft2	2-D inverse fast Fourier transform
ifftshift	Inverse FFT shift

Color
Color space conversions, support for International Color Consortium (ICC) profiles

Various color spaces exist because they present color information in ways that make certain calculations more convenient or because they provide a more intuitive way to identify colors. The toolbox represents colors as RGB values and provides tools for converting color data from one color space to another. The toolbox also support International Color Consortium (ICC) profiles for describing colors.

Functions

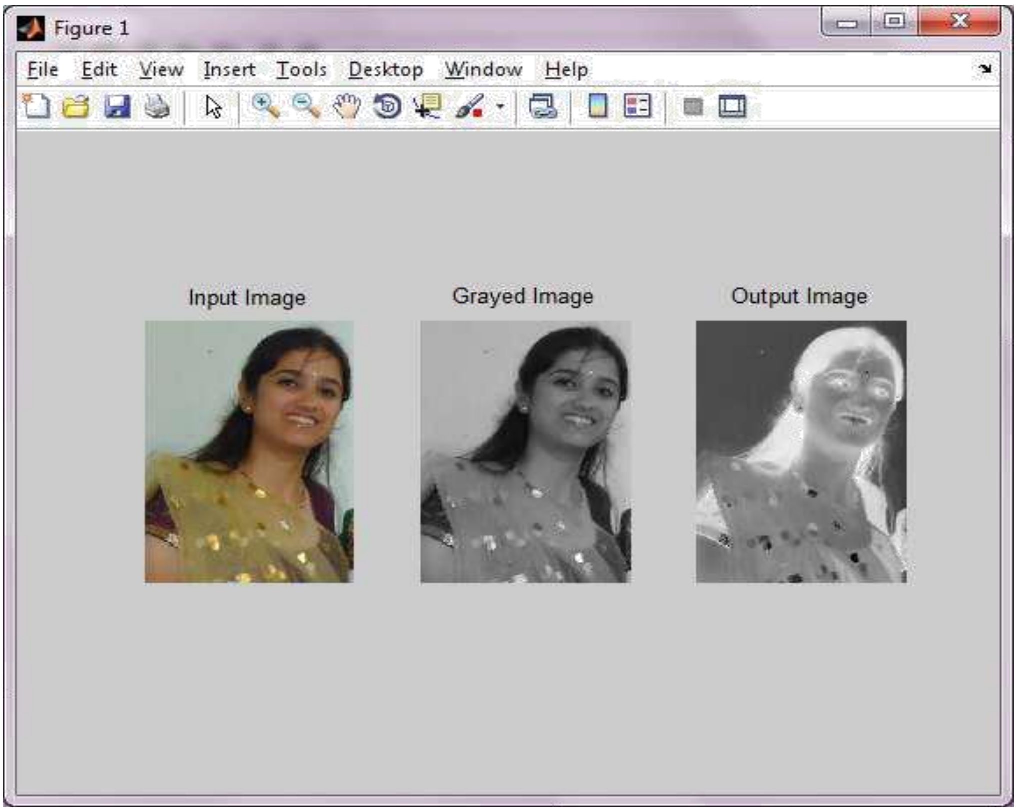
rgb2lab	Convert RGB to CIE 1976 L*a*b*
rgb2ntsc	Convert RGB color values to NTSC color space
rgb2xyz	Convert RGB to CIE 1931 XYZ
rgb2ycbcr	Convert RGB color values to YCbCr color space
lab2rgb	Convert CIE 1931 L*a*b* to RGB
lab2xyz	Convert CIE 1931 L*a*b* to CIE 1931 XYZ
xyz2lab	Convert CIE 1931 XYZ to CIE 1976 L*a*b*
xyz2rgb	Convert CIE 1931 XYZ to RGB
ycbcr2rgb	Convert YCbCr color values to RGB color space
ntsc2rgb	Convert NTSC values to RGB color space

Practical 2

Aim: To write a program for implementing Image Negatives.

Source Code:

```
clear all;
close all;
clc;

imx=imread('D:\Photos\me.jpg');
im=rgb2gray(imx);
siz=size(im);
L=max(max(im));
for i=1:siz(1,1)
for j=1:siz(1,2)
im1(i,j)=L-im(i,j);

end
end
figure(1)
subplot(1,3,1),imshow(imx)
title('Input Image');
subplot(1,3,2),imshow(im)
title('Grayed Image');
subplot(1,3,3),imshow(im1)
title('Output Image');
```

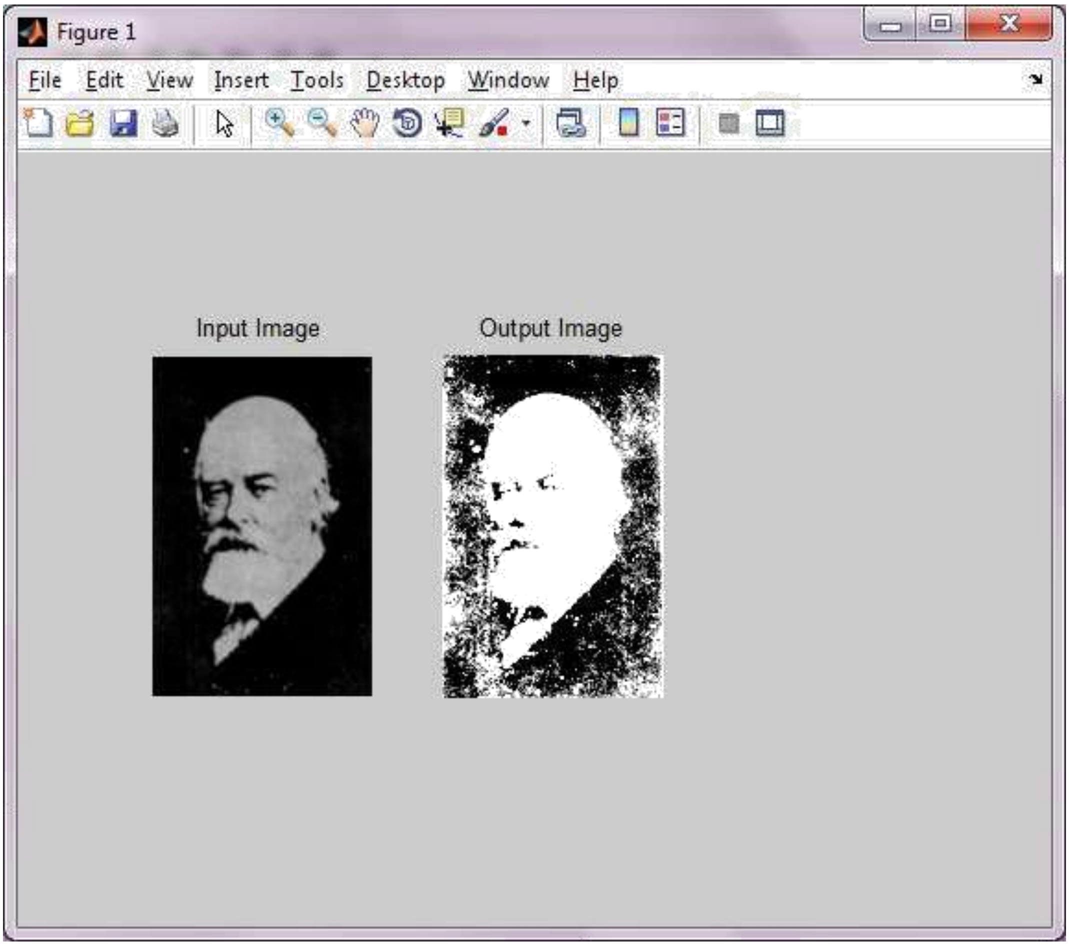


Practical 2

1). Aim: To write a program for implementing Log Transformations.

Source Code:

```
clear all;
close all;
clc;
im2=imread('H:\DIPimages\2nd\3rd\4th\Fig0102(1922 digital image).tif');
im=double(im2);
siz=size(im);
for i=1:siz(1,1)
for j=1:siz(1,2)
        s=1+im(i,j);
im1(i,j)=1 * log10(s);
end
end
figure(1)
subplot(1,3,1),imshow(im2)
title('Input Image');
subplot(1,3,2),imshow(im1)
title('Output Image');
```



2). Aim: To write a program for implementing Power Law (Gamma) Transformations.

Source Code:

```
clear all;
close all;
clc;
im2=imread('H:\DIP3E_CH02_Original_Images\DIP3E_Original_Images_CH02\fractured_s
pine.tif');
im=double(im2);
siz=size(im);
for i=1:siz(1,1)
for j=1:siz(1,2)
im1(i,j)=0.09*(im(i,j)^0.9);
end
end
figure(1)
subplot(1,3,1),imshow(im2)
title('Input Image');
subplot(1,3,2),imshow(im1)
title('Output Image');
```

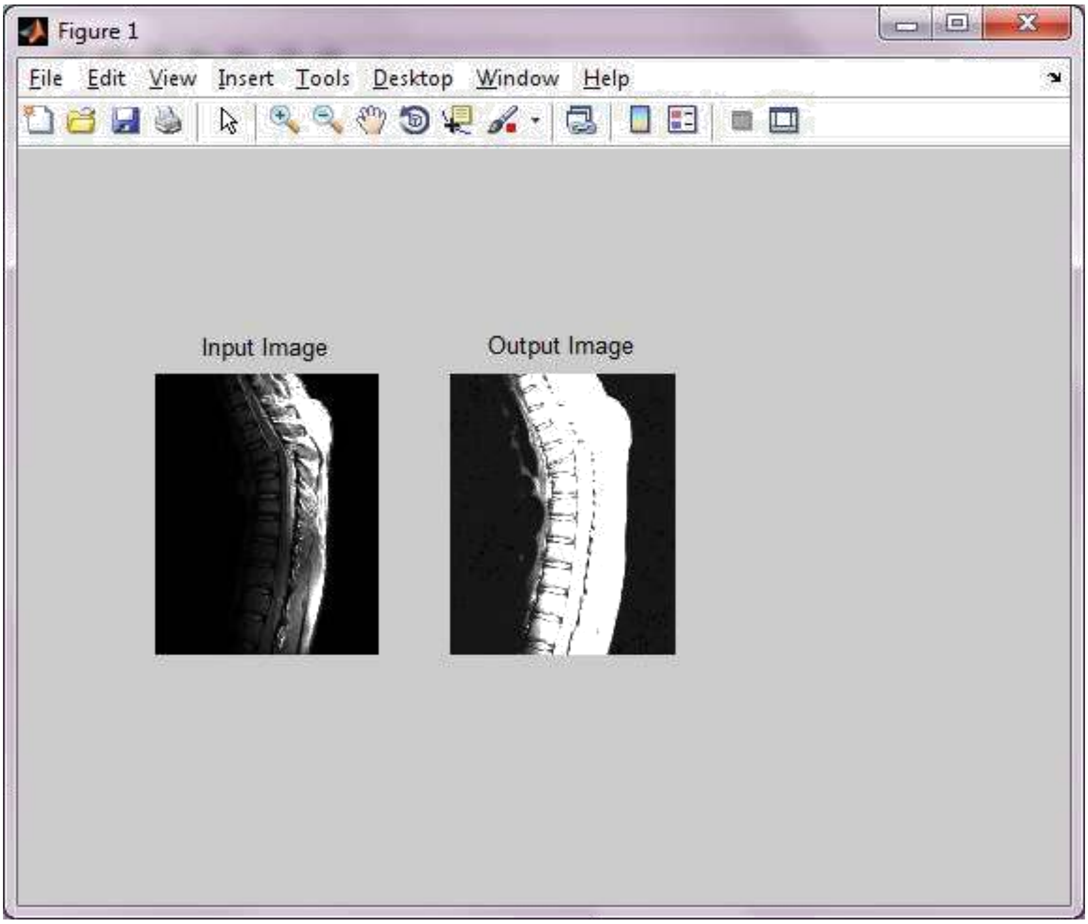


Figure 1
File Edit View Insert Tools Desktop Window Help
Input Image
Output Image

3). Aim: To write a program to implement Bit Plane

Slicing Source Code:

```
clc;
clear all;
close all;
imx= imread('H:\bitplane.jpg');
imm=rgb2gray(imx);
[row,col,plane]=size(imm);
imm=imm(:,:,1);
im=zeros(row,col,8);
for k=1:8
for i=1:row
for j=1:col
im(i,j,k)=bitget(imm(i,j),k);
ifim(i,j,k)==1;
im(i,j,k)=255;
end
end
end
end
str = 'PLANE -- ';
for k=1:8
subplot(3,3,k)
imshow(im(:,:,k));
stri = strcat(str, num2str(k));
title(stri);
end

subplot(3,3,9);
imshow(imm);
title('Original Image');
```

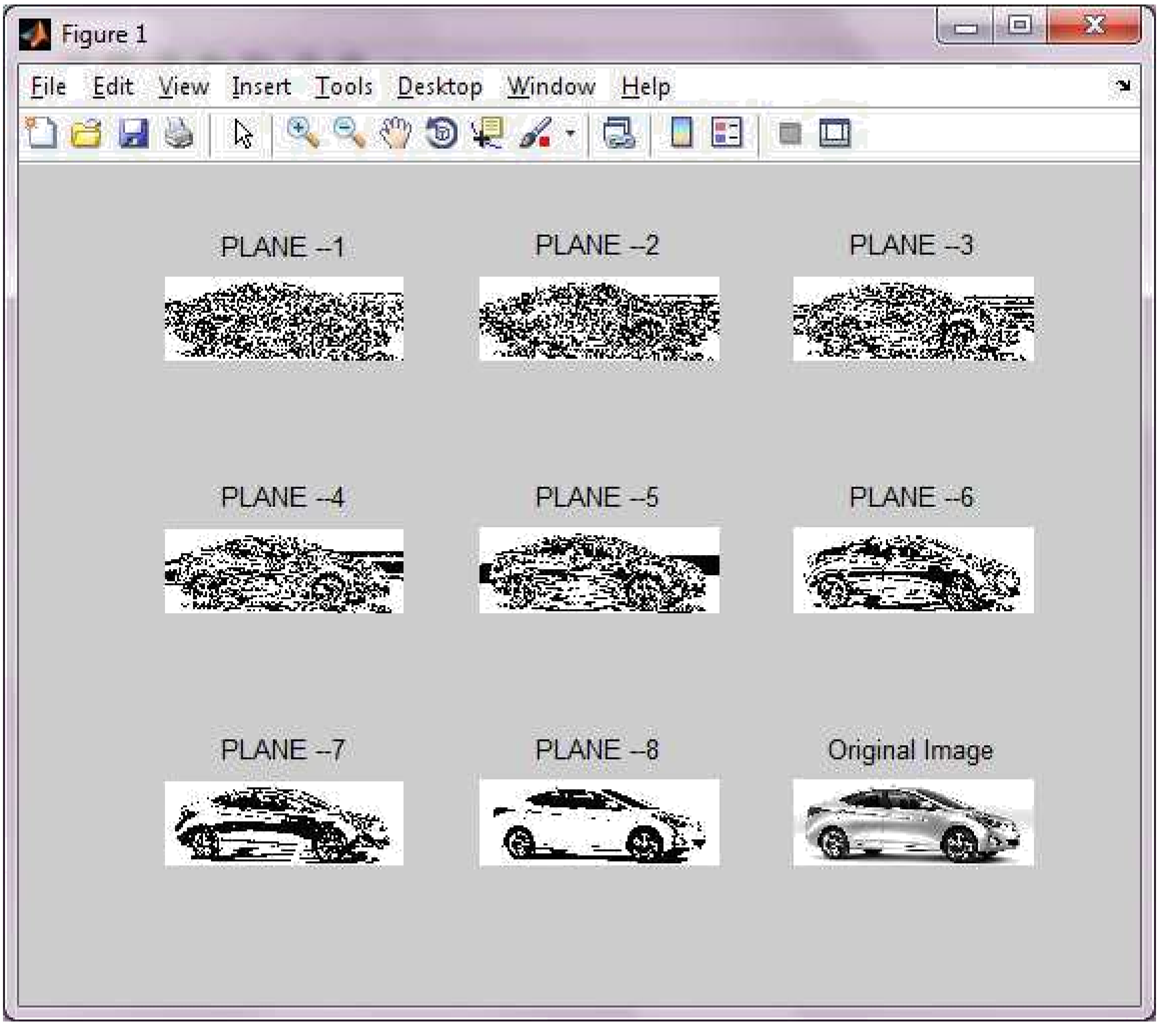


Figure 1
File Edit View Insert Tools Desktop Window Help
PLANE --1
PLANE --2
PLANE --3
PLANE --4
PLANE --5
PLANE --6
PLANE --7
PLANE --8
Original Image

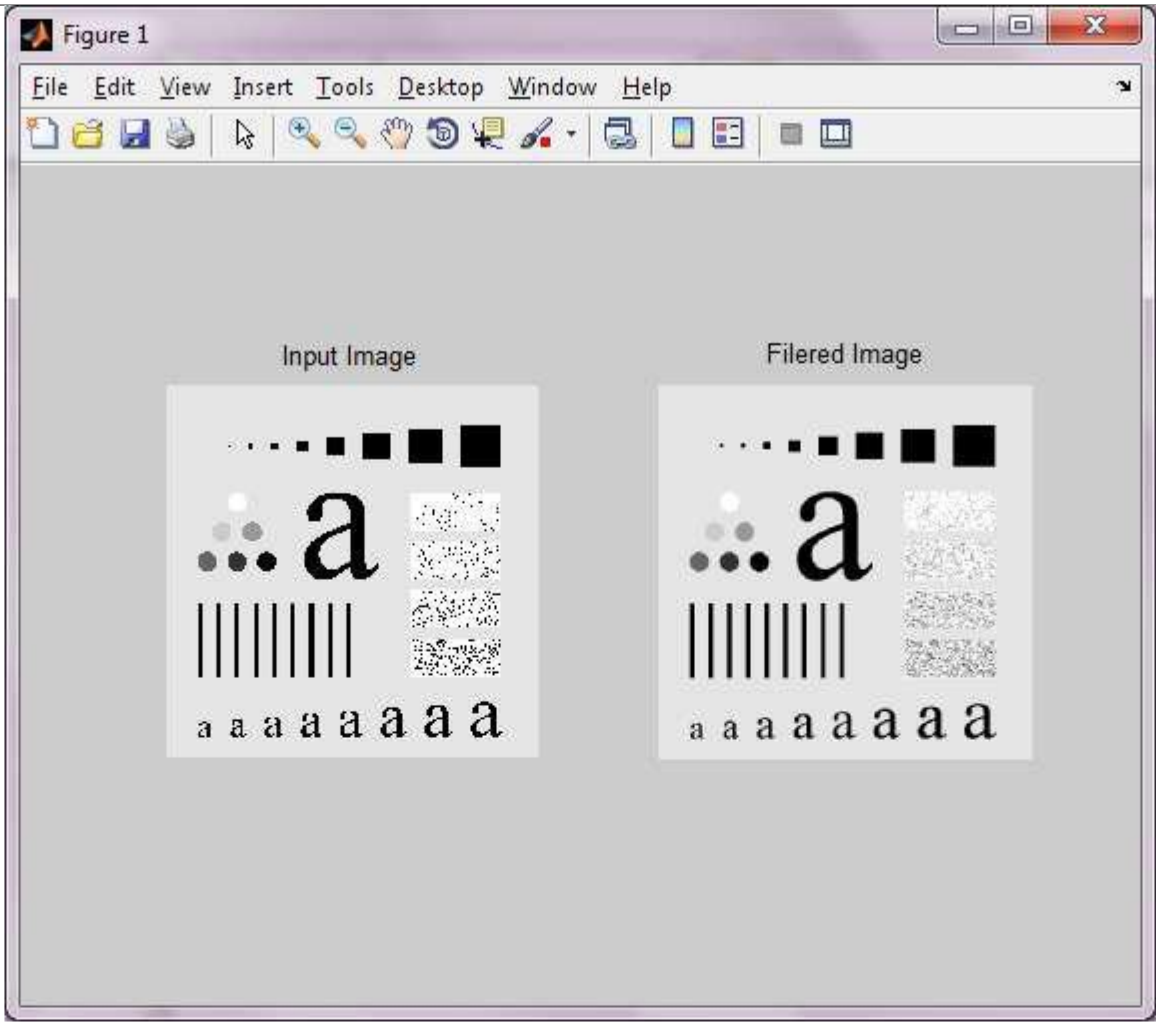
Practical 3

1). Aim: To write a program to implement spatial filter with dynamic mask (average filter)

Source Code:

```
clc;
close all;
clear all;

% a=[1,2,3,4,5;4,5,6,7,8;9,10,11,12,13;10,11,12,13,14]
  a1=imread('H:\image.tif');
  %a1=rgb2gray(b1);
%  imshow(a);
 a=im2double(a1);

in=input('Enter the input of mask ');
%performing on a mask with all co-efficient values as 1.
for m1=1:in
for n1=1:in
w(m1,n1)=1;
end
end
z1=0;
[m,n]=size(a);
[p,q]=size(w);
p1=in/2;
p1=floor(p1);
p2=p1;
 g=padarray(a,[p1,p1]);
for i=1:m
for j=1:n
img(i,j)=0;
for r1=1:in
for r2=1:in
img(i,j)=img(i,j)+ g(i+r1-1,j+r2-1)*w(r1,r2);
end
end
end
end
for i=1:m
for j=1:n
res(i,j)=img(i,j)/(p*q);
%if co-efficient values are not all 1s then divide by the summation of all co-efficients
end
end
figure(1)
subplot(1,2,1),imshow(a)
title('Input Image');
subplot(1,2,2),imshow(res)
```

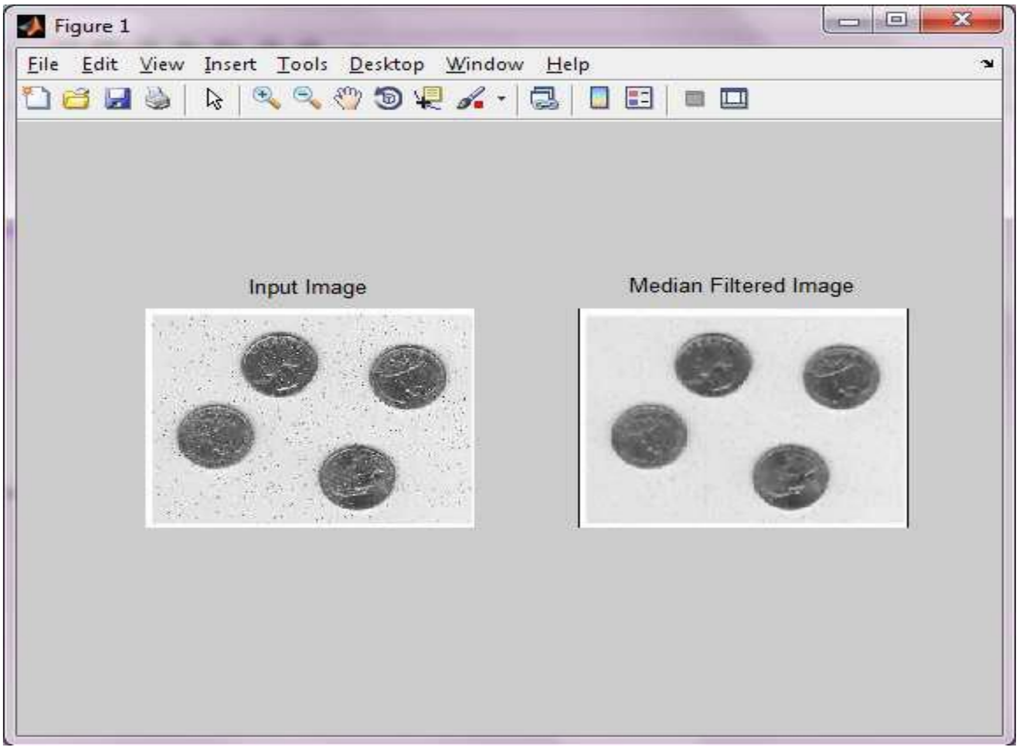
```
title('Filered Image');
```



2). Aim: To write a program to implement median filter
Source Code:

```
clear all;
clc;
close all;

im = imread('D:\1stSem_MTech\Digital_Image_Processing\mefi.tif'); b
= double(im);
c = padarray(b,[1,1],0,'both');
d = c;
[row col] = size(c);
for i = 2:row-1
for j = 2:col-1
        a1 = [c(i-1,j-1) c(i-1,j) c(i-1,j+1) c(i,j-1) c(i,j) c(i,j+1) c(i+1,j-1) c(i+1,j) c(i+1,j+1)];
        a2 = sort(a1);
med = a2(5);
d(i,j) = med;
end
end
```

```
figure(1)
subplot(1,2,1),imshow(uint8(im))
title('Input Image');
subplot(1,2,2),imshow(uint8(d))
title('Median Filtered Image');
```



Practical 4

Aim: Image Contrast Enhancement Using Histogram Equalization

Abstract - General framework based on histogram equalization for image contrast enhancement is discussed. In this framework, contrast enhancement is posed as an optimization problem that minimizes a cost function. Histogram equalization is an effective technique for contrast enhancement. However, conventional histogram equalization (HE) usually results in excessive contrast enhancement, which in turn gives the processed image an unnatural look and creates visual artifacts. By introducing specifically designed penalty terms, the level of contrast enhancement can be adjusted; noise robustness, white/black stretching and mean-brightness preservation may easily be incorporated into the optimization.
Keywords: Histogram equalization, histogram modification, image/video quality enhancement.

1. INTRODUCTION

Contrast enhancement plays a crucial role in image processing applications, such as digital photography, medical image analysis, remote sensing, LCD display processing, and scientific visualization. Image enhancement is a technique which reduces image noise, remove artifacts, and preserve details. Its purpose is to amplify certain image features for analysis, diagnosis and display. Contrast enhancement increases the total contrast of an image by making light colors lighter and dark colors darker at the same time. It does this by setting all color components below a specified lower bound to zero, and all color components above a specified upper bound to the maximum intensity (that is, 255). Color components between the upper and lower bounds are set to a linear ramp of values between 0 and 255. Because the upper bound must be greater than the lower bound, the lower bound must be between 0 and 254, and the upper bound must be between 1 and 255. Some users describe the enhanced image that if a curtain of fog has been removed from the image [1] . There are several reasons for an image/video to have poor contrast: • the poor quality of the used imaging device, • lack of expertise of the operator, and •The adverse external conditions at the time of acquisition. These effects result in under-utilization of the offered dynamic range. As a result, such images and videos may not reveal all the details in the captured scene, and may have a washed-out and unnatural look.

2. IMAGE ENHANCEMENT

Image enhancement processed consist of a collection of techniques that seek to improve the visual appearance of an image or to convert the image to a form better suited for analysis by a human or machine[2] . Enhancement of an image can be implemented by using different operations of brightness increment, sharpening, blurring or noise removal. Unfortunately, there is no general theory for determining what _good‘ image enhancement, when it comes to human perception. If it looks good, it is good! While categorizing Image Enhancement operations can be divided in two categories:

2.1 TECHNIQUES OF CONTRAST ENHANCEMENT

These techniques can be broadly categorized into two groups:

- direct methods and,
- Indirect methods.

2.1.1

Direct method In direct method of contrast enhancement, a contrast measure is first defined, which is then modified by a mapping function to generate the pixel value of the enhanced image. Various mapping functions such as the square root function, the exponential function, etc., have been introduced for the contrast measure modification. However, these functions do not produce satisfactory contrast enhancement results and are usually sensitive to noise and digitization effects [4]. In addition, they are computationally complex from the point of view of implementation. The polynomial function is ready to implement on digital computers and provides very satisfactory contrast enhancement.

2.1.2

Indirect method Indirect methods, on the other hand, improve the contrast through exploiting the underutilized regions of the dynamic range without defining a specific contrast term. Most methods in the literature fall into the second group [4]. Indirect methods can further be divided into several subgroups:

- techniques that decompose an image into high and low frequency signals for manipulation, e.g., homomorphic filtering,
- Histogram modification techniques, and
- Transform-based techniques. Out of these three subgroups, the second subgroup received the most attention due to its straightforward and intuitive implementation qualities.

3. HISTOGRAM EQUALIZATION

Contrast enhancement techniques in the second subgroup modify the image through some pixel mapping such that the histogram of the processed image is more spread than that of the original image. Techniques in this subgroup either enhance the contrast globally or locally. Ifa single mapping derived from the image is used then it is a global method; if the neighborhood of each pixel is used to obtain a local mapping function then it is a local method. Using a single global mapping cannot (specifically) enhance the local contrast [5], [6]. One of the most popular global contrast enhancement techniques is histogram equalization (HE). The histogram in the context of image processing is the operation by which the occurrence of each intensity value in the image is shown. Normally, the histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. For an 8- bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grayscale values [7]. Histogram equalization is the technique by which the dynamic range of the histogram of an image is increased. HE assigns the intensity values of pixels in the input image such that the output image contains a uniform distribution of intensities. It improves contrast and the goal of HE is to obtain a uniform histogram. This technique can be used on a whole image or just on a part of an image. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast without affecting the global contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values. The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed.

- Advantage: A key advantage of the method is that it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered.
- Disadvantage: A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal.

3.1 HE

often produces unrealistic effects in photographs; however it is very useful for scientific images like thermal, satellite or x-ray images, often the same class of images that user would apply false-color to. Also histogram equalization can produce undesirable effects (like visible image gradient) when applied to images with low color depth. For example if applied to 8-bit image displayed with 8-bit gray-scale palette it will further reduce color depth (number of unique shades of gray) of the image. Histogram equalization will work the best when applied to images with much higher color depth than palette size, like continuous data or 16-bit grayscale images. Histogram equalization is a specific case of the more general class of histogram remapping methods. These methods seek to adjust the image to make it easier to analyze or improve visual quality. The above describes histogram equalization on a grey-scale image. However it can also be used on color images by applying the same method separately to the Red, Green and Blue components of the RGB color values of the image. Still, it should be noted that applying the same method on the Red, Green, and Blue components of an RGB image may yield dramatic changes in the image's color balance since the relative distributions of the color channels change as a result of applying the algorithm. However, if the image is first converted to another color space, Lab color space, or HSL/HSV color space in particular, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image [3]. The histogram is a discrete function $h(r=k) = nk$, Where nk is the number of pixels in the image having gray level k It is a common practice to normalize a histogram by dividing each of its values by the total number of pixels in the image (n) $p(r=k) = nk/n$, $k=0, 1, \dots, L-1$ Where $p(r=k)$ is an estimate of the probability of occurrence of gray level k [8]. Following graph

4. CONCLUSION

The contrast of the image can be improved without introducing visual artifacts that decrease the visual quality of an image and cause it to have an unnatural look. The experimental results show the effectiveness of the algorithm in comparison to other contrast enhancement algorithms. Obtained images are visually pleasing, artifact free, and natural looking. A desirable feature of this paper is that it does not introduce flickering.

This is mainly due to the fact that the method uses the input (conditional) histogram, which does not change significantly within the same scene, as the primary source of information. This method is applicable to a wide variety of images. It also offers a level of controllability and adaptability through which different levels of contrast enhancement, from histogram equalization to no contrast enhancement, can be achieved.

5. REFERENCES

1. G.de Haan, ―Video Processing for Multimedia Systems‖, Eindhoven, The Netherlands, 2000.

2. http://www.cromwell-intl.com/3d/histogram/

3. J.A. Stark, ―Adaptive image contrast enhancement using generalizations of histogram equalization‖, IEEE Trans. Image Process, vol. 9, no. 5, pp. 889-896, May 2000. 4. J.-Y. Kim, L.-S Kim, and S.-H. Hwang, ‖An advanced contrast enhancement using partially overlapped sub-block histogram equalization‖, IEEE Trans. Circuits Syst. Video Technol, vol. 11, no.
4, pp. 475-484, Apr. 2001.

5. N.R.Mokhtar, Nor HazlynaHarun, M.Y. Mashor, H.Roseline, Nazahaha Mustafa, R.Adollah, H. Adilah, N.F.ModhNasir, ―Image Enhancement Techniques Using Local, Global, Bright, Dark and Partial Contrast Stretching‖, Proceedings of the world Congress on Engineering 2009 vol. I, WCE 2209, July 1-3, 2009, London U.K.

6. Rafael C. Gonzalez, Richard E. Woods, ―Digital Image Processing‖, Pearson Education, Inc.

7. Rafael C. Gonzalez, Richard E. Woods,‖Digital Image Processing Using MATLAB‖, Pearson Education, Inc.

8. TarikArici, SalihDikbas, Member, IEEE, and YucelAltunbasak, Senior Member, IEEE, ―A Histogram Modification Framework and Its Application for Image Contrast enhancement‖, IEEE Transactions on Image Processing, VOL.18, No. 9, Sept. 2009.

Practical 5

AIM : Study of Various Edge Detector (Sobel, canny etc.) , implement any one edge detector with any soft computing technique (FUZZY,NN,GA).and Compare the results.

INTRODUCTION

Edge detection refers to the process of identifying and locating sharp discontinuities in an image. The discontinuities are abrupt changes in pixel intensity which characterize boundaries of objects in a scene. Classical methods of edge detection involve convolving the image with an operator (a 2-D filter), which is constructed to be sensitive to large gradients in the image while returning values of zero in uniform regions. There are an extremely large number of edge detection operators available, each designed to be sensitive to certain types of edges. Variables involved in the selection of an edge detection operator include Edge orientation, Noise environment and Edge structure. The geometry of the operator determines a characteristic direction in which it is most sensitive to edges. Operators can be optimized to look for horizontal, vertical, or diagonal edges. Edge detection is difficult in noisy images, since both the noise and the edges contain high frequency content. Attempts to reduce the noise result in blurred and distorted edges. Operators used on noisy images are typically larger in scope, so they can average enough data to discount localized noisy pixels. This results in less accurate localization of the detected edges. Not all edges involve a step change in intensity. Effects such as refraction or poor focus can result in objects with boundaries defined by a gradual change in intensity. The operator needs to be chosen to be responsive to such a gradual change in those cases. So, there are problems of false edge detection, missing true edges, edge localization, high computational time and problems due to noise etc. Therefore, the objective is to do the comparison of various edge detection techniques and analyze the performance of the various techniques in different conditions. There are many ways to perform edge detection. However, the majority of different methods may be grouped into two categories:

Gradient based Edge Detection:

The gradient method detects the edges by looking for the maximum and minimum in the first derivative of theimage.

Laplacian based Edge Detection:

The Laplacian method searches for zero crossings in the second derivative of the image to find edges. An edgehas the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location.

Edge Detection Operators:

1. Sobel Operator:
The operator consists of a pair of 3×3 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient [3]. The gradient magnitude is given byTypically, an approximate magnitude is computed using:
G = Gx + Gy which is much faster to compute.

2. Robert‟s cross operator:
The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point. The operator consists of a pair of 2×2 convolution kernels as shown in Figure. One kernel is simply the other rotated by 90°

+1	0
0	-1

Gx

0	+1
-1	0

Gy

3. Prewitt‟s operator:
Prewitt operator [5] is similar to the Sobel operator and is used for detecting vertical and horizontal edges in images.

-1	0	+1
-1	0	+1
-1	0	+1

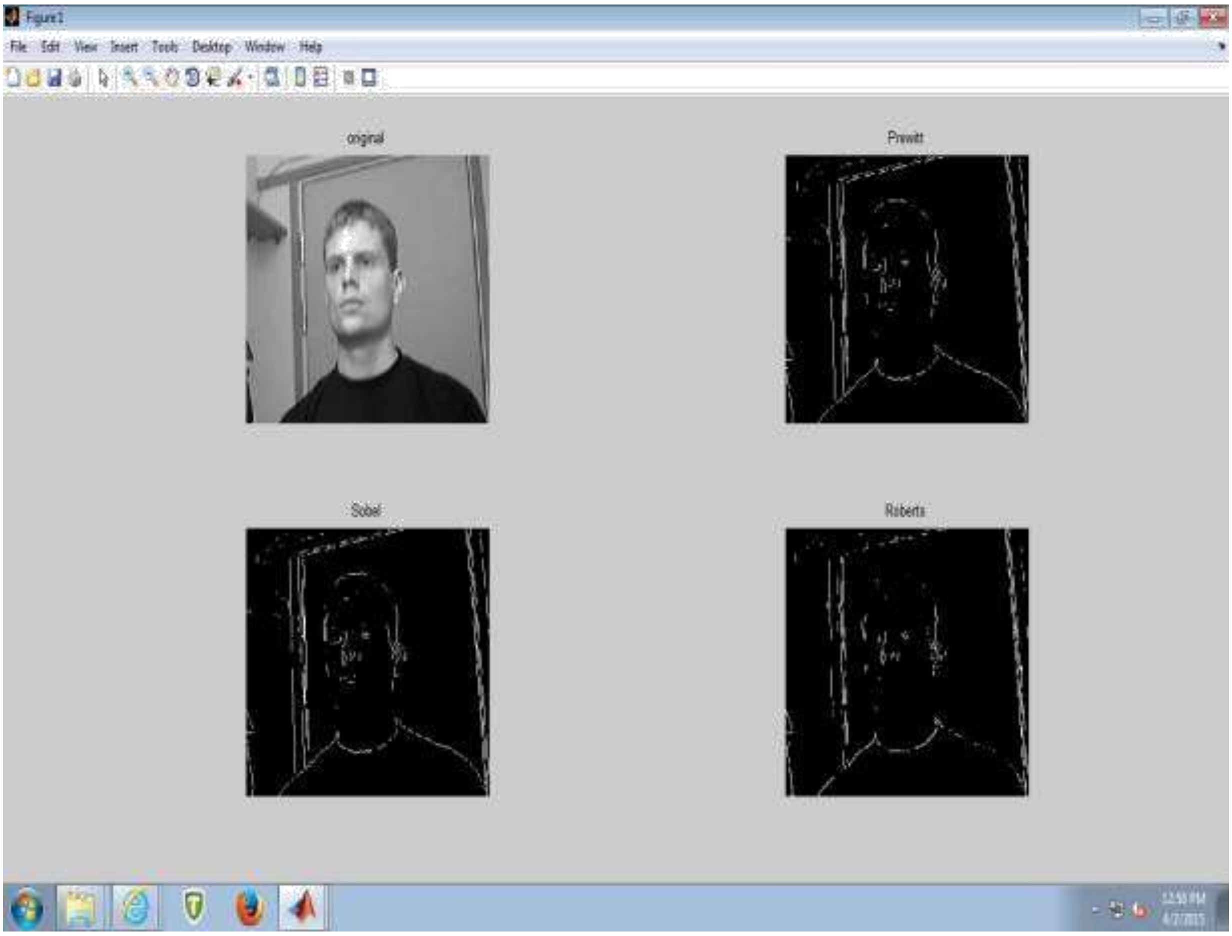
Gx

+1	+1	+1
0	0	0
-1	-1	-1

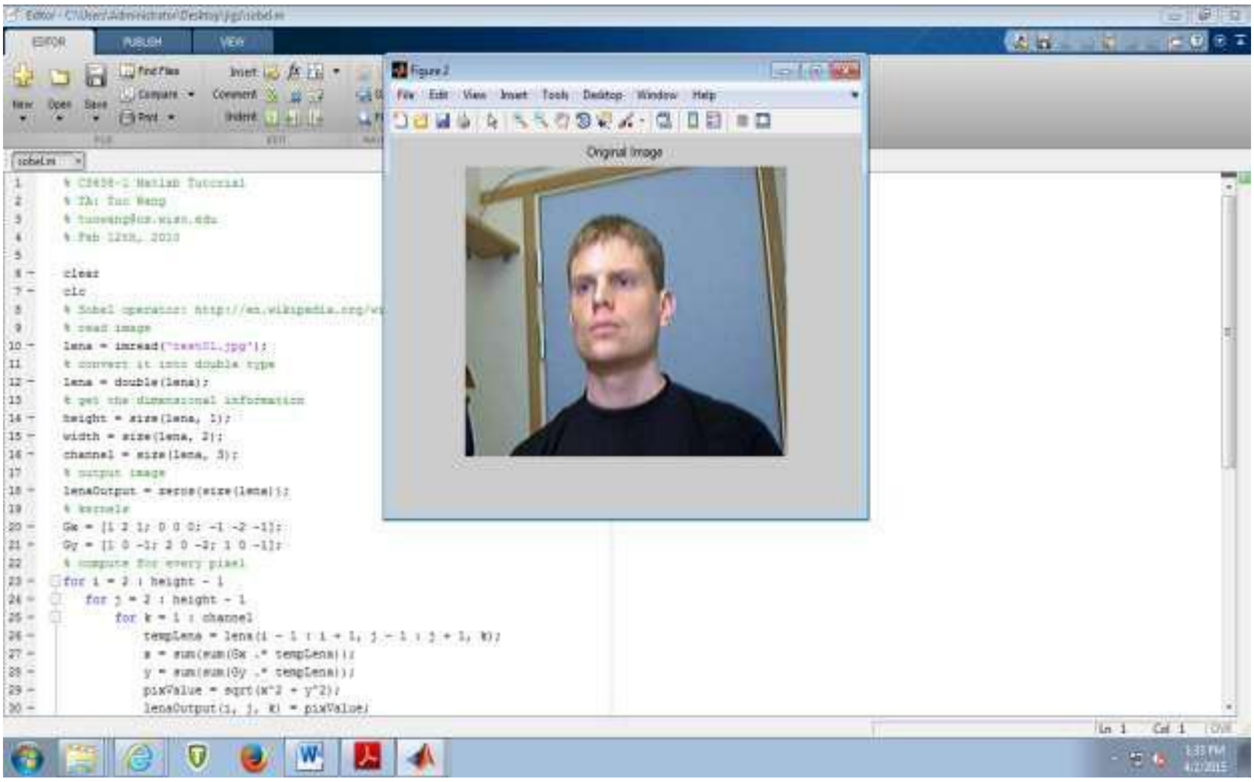
Gy

Implementation using MATLAB inbuilt Prewitt, SObel, Roberts Operators

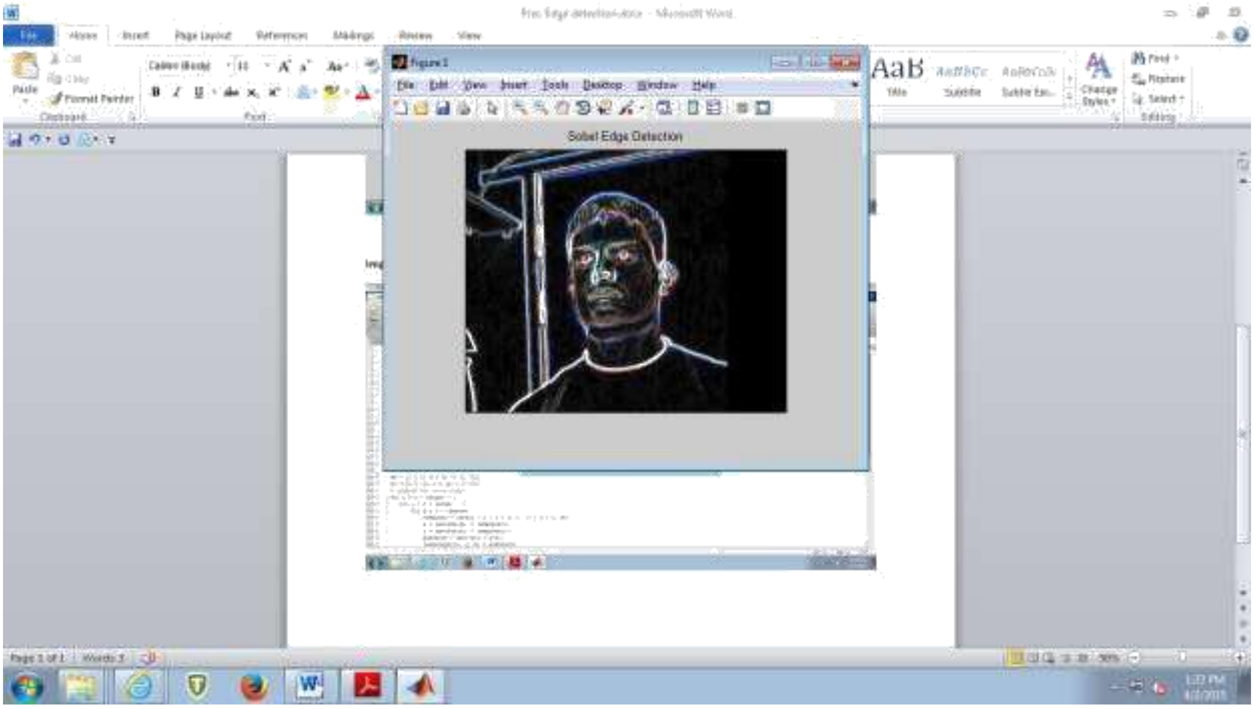
```
% BW = edge(I) takes a grayscale or a binary image I as its input, and
% returns a binary image BW of the same size as I, with 1's where the
% function finds edges in I and 0's elsewhere.
%
%  By default, edge uses the Sobel method to detect edges but the following
%  provides a complete list of all the edge-finding methods supported by
%  this function:
% The Sobel method finds edges using the Sobel approximation to the
% derivative. It returns edges at those points where the gradient of I is
% maximum. The Prewitt method finds edges using the Prewitt approximation
% to the derivative. It returns edges at those points where the gradient of
% I is maximum. The Roberts method finds edges using the Roberts
% approximation to the derivative. It returns edges at those points where
% the gradient of I is maximum. The Laplacian of Gaussian method finds
% edges by looking for zero crossings after filtering I with a Laplacian of
% Gaussian filter. The zero-cross method finds edges by looking for zero
% crossings after filtering I with a filter you specify. The Canny method
% finds edges by looking for local maxima of the gradient of I. The
% gradient is calculated using the derivative of a Gaussian filter. The
% method uses two thresholds, to detect strong and weak edges, and includes
% the weak edges in the output only if they are connected to strong edges.
% This method is therefore less likely than the others to be fooled by
% noise, and more likely to detect true weak edges. The parameters you can
% supply differ depending on the method you specify. If you do not specify
% a method, edge uses the Sobel method.

i = imread('test01.jpg');
I = rgb2gray(i);
BW1 = edge(I,'prewitt');
BW2= edge(I,'sobel');
BW3= edge(I,'roberts');
subplot (2,2,1);
imshow(I);
title('original');
subplot(2,2,2);
imshow(BW1);
title('Prewitt');
subplot(2,2,3);
imshow(BW2);
title('Sobel');
subplot(2,2,4);
imshow(BW3);
title('Roberts');
```



Implemented with developed Sobel operator:



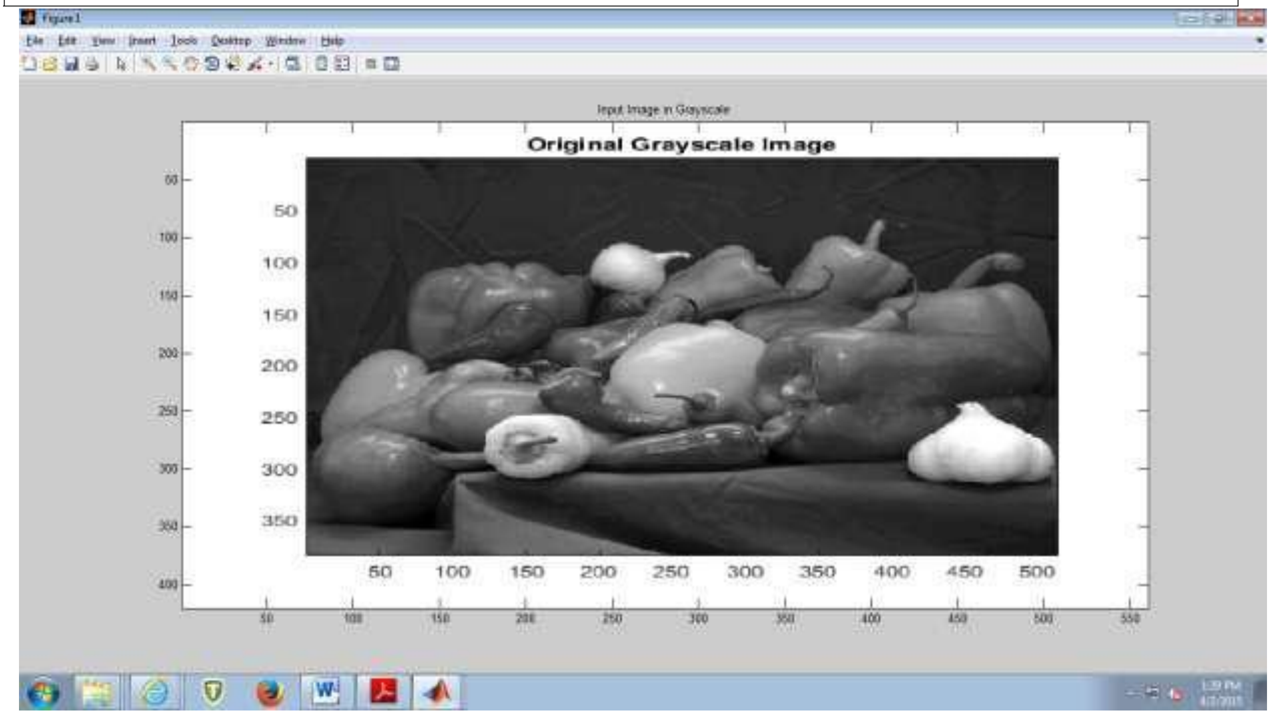
```
% read image
lena = imread('test01.jpg');
% convert it into double
type lena = double(lena);
% get the dimensional information
height = size(lena, 1);
width = size(lena, 2);
channel = size(lena, 3);
% output image
lenaOutput = zeros(size(lena));
% kernels
Gx = [1 2 1; 0 0 0; -1 -2 -1];
Gy = [1 0 -1; 2 0 -2; 1 0 -1];
% compute for every pixel
for i = 2 : height - 1
for j = 2 : height - 1
for k = 1 : channel
tempLena = lena(i - 1 : i + 1, j - 1 : j + 1, k);

        x = sum(sum(Gx .* tempLena));
        y = sum(sum(Gy .* tempLena));
pixValue = sqrt(x^2 + y^2);
lenaOutput(i, j, k) = pixValue;
end
end
end
% display the processed image
lenaOutput = uint8(lenaOutput);
figure;
imshow(lenaOutput);
title('Sobel Edge Detection');
% write the output to disk
imwrite(lenaOutput, 'lenaOutput.jpg', 'jpg')

% original image
figure;
imshow(uint8(lena));
title('Original Image');
```



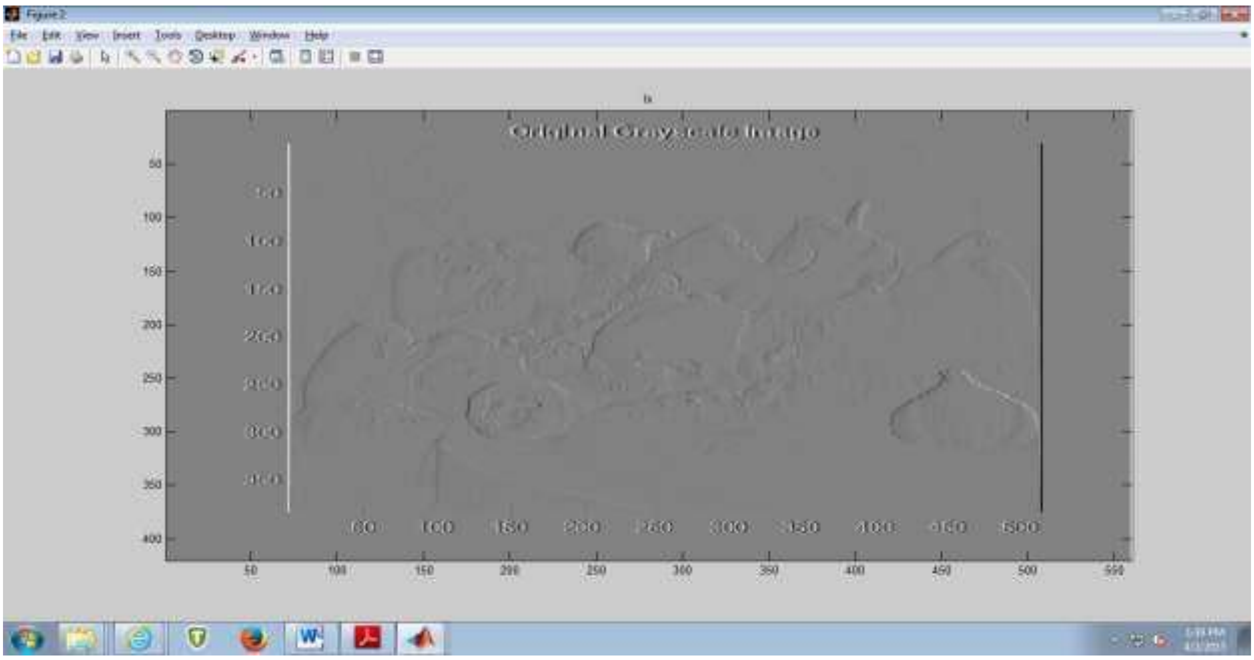
Implemented using Fuzzy logic (FIS)

```
clc
close all
clear all
k=readfis('ed');
% [filename path]=uigetfile('*.*','Select the
 image') a=imread('peppers.png');
   a=double(a);
d=a;
[r,c]=size(a);
r1=r-1;
c1=c-1;
for i=1:1:r1
for j=1:1:c1
        b=a(i:i+1,j:j+1);
        b=b';
        b=b(:);


%i=[255 255 255 0];
e=evalfis(b,k);
e=round(e);
d(i+1,j+1)=e;
end
end

edge1=d;
```

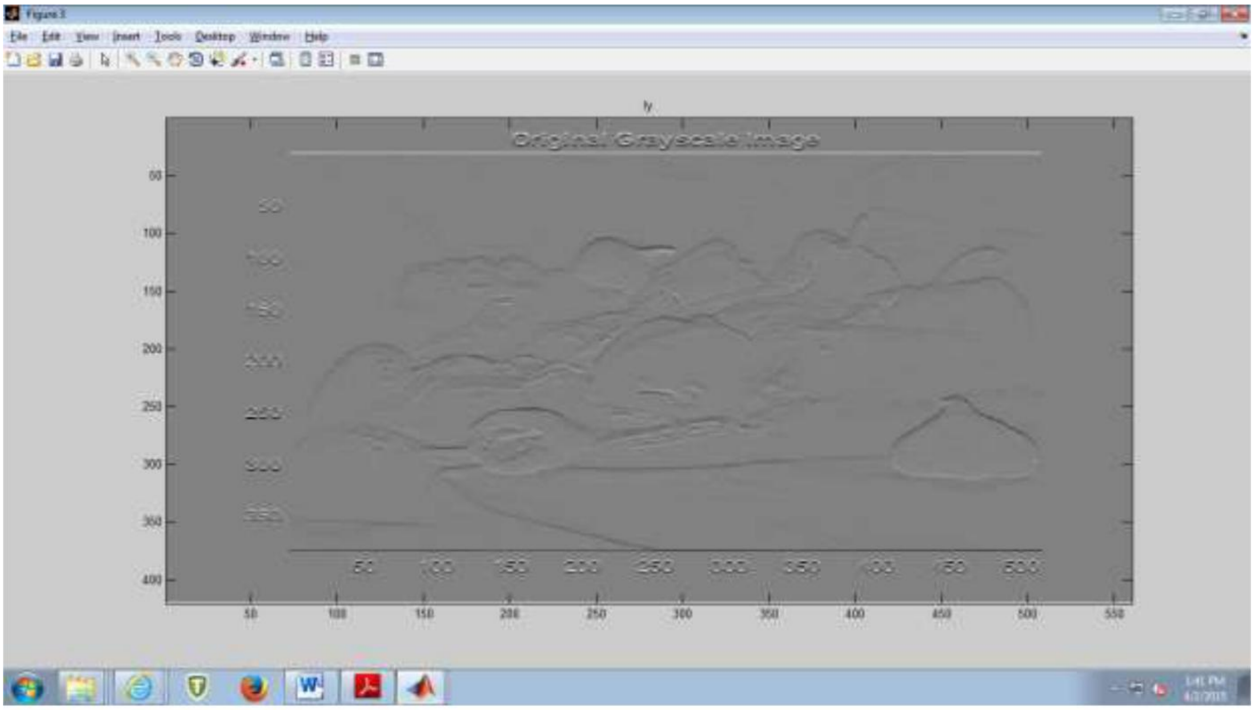
```
edge1(edge1>134)=0;
edge1(edge1<130)=0;
edge1(edge1~=0)=1;
figure
imshow(uint8(a))
title('Original image')
figure
imshow(edge1)
title('Edge detection using fuzzy logic')
```

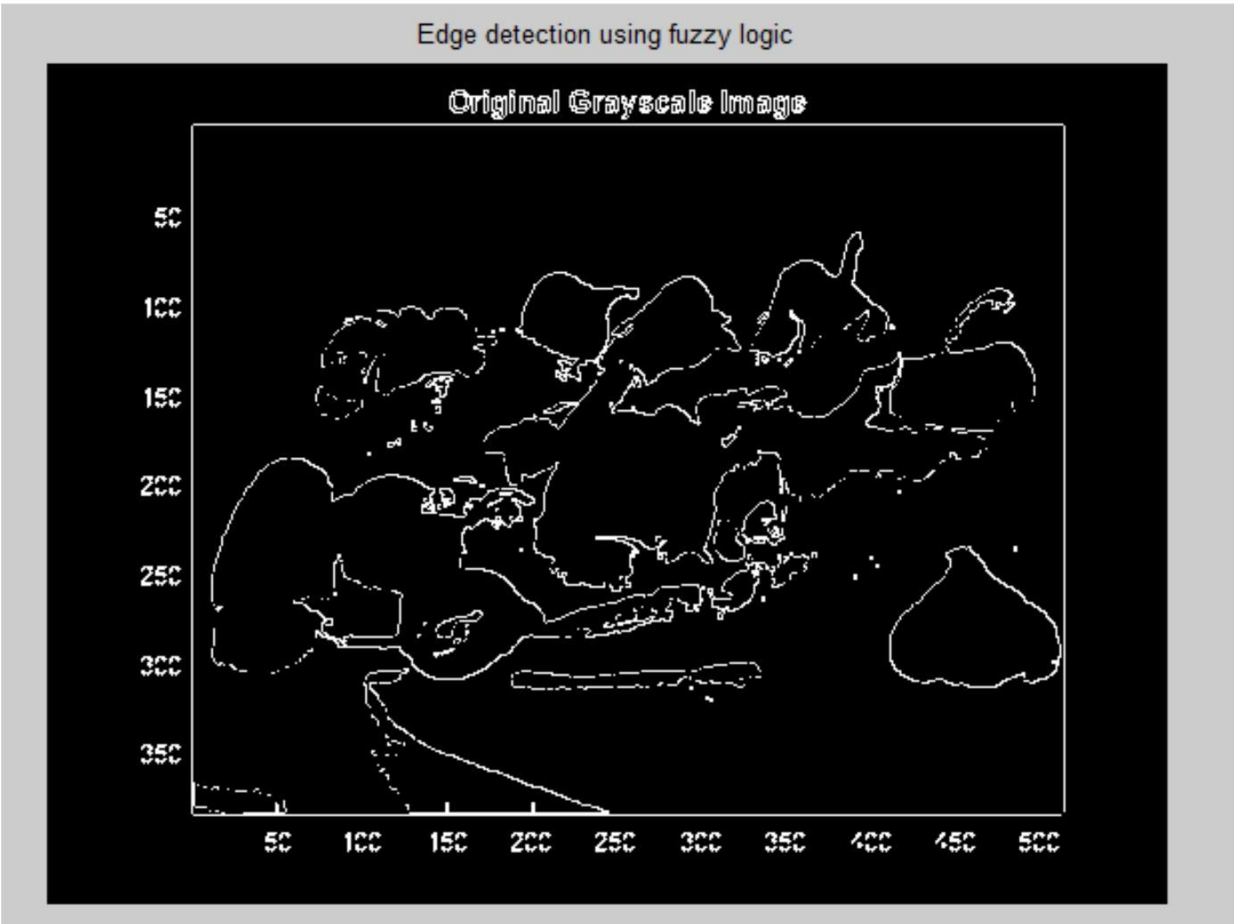


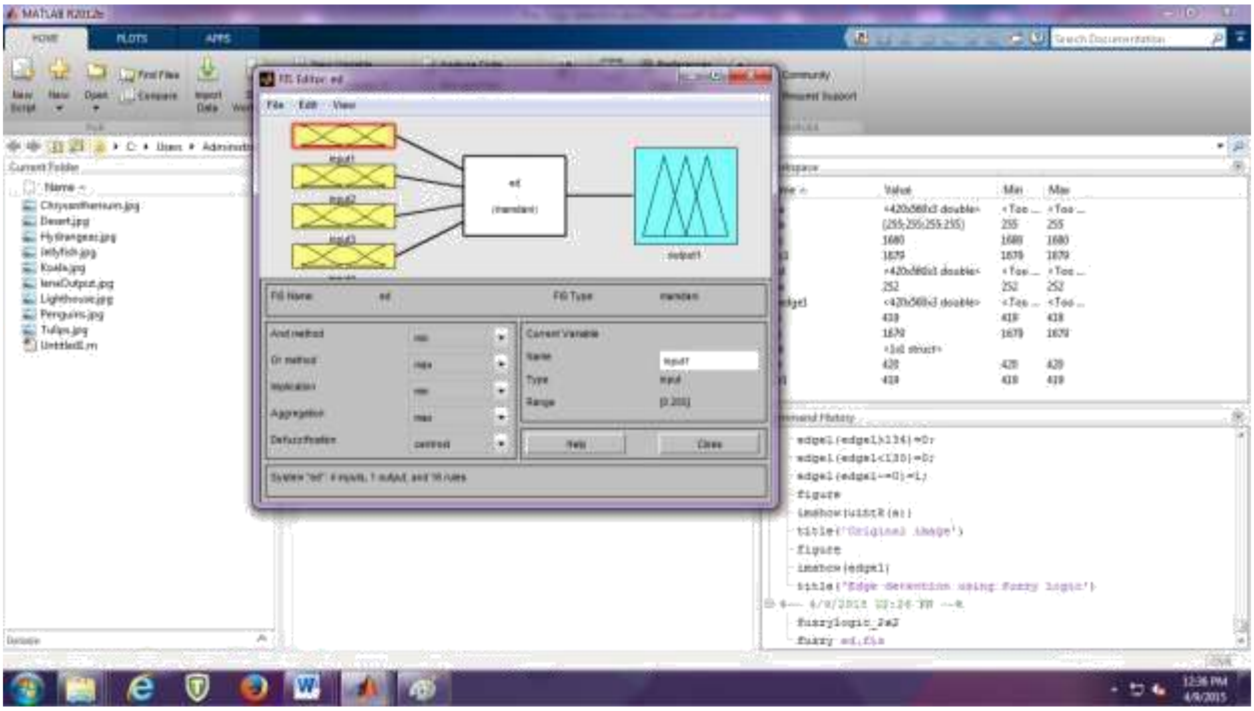
X axis Image gradient



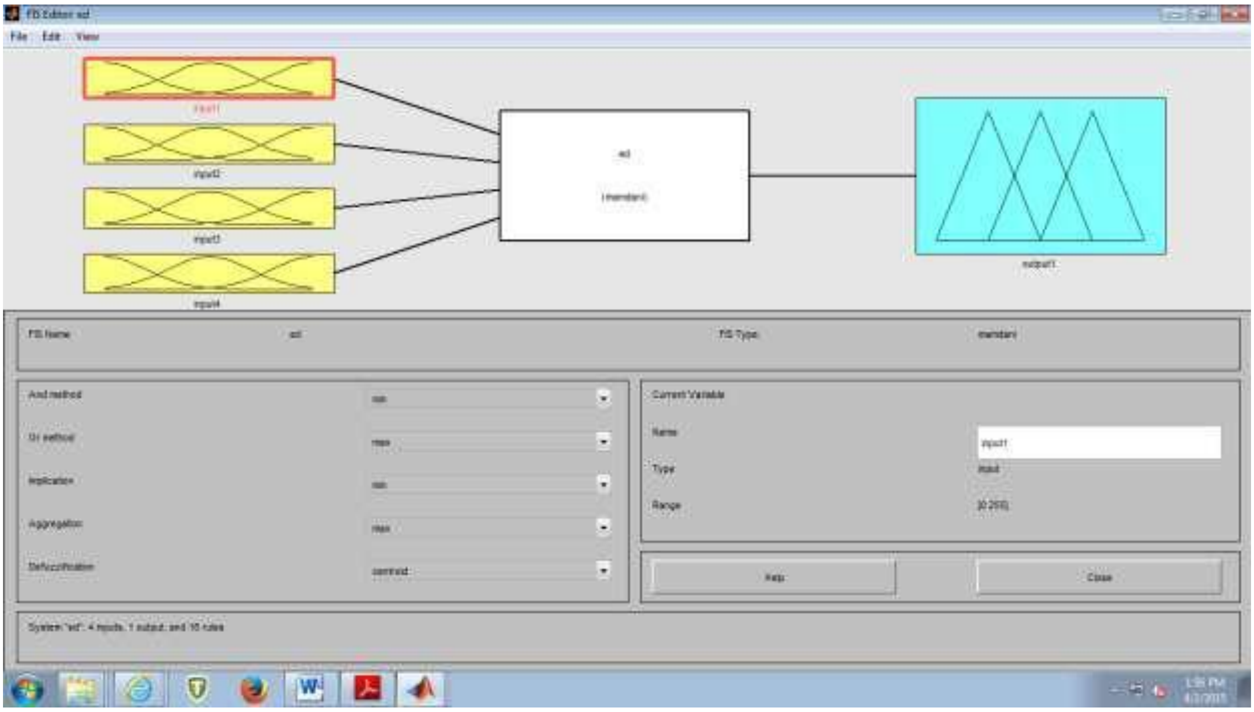
Y axis image gradient

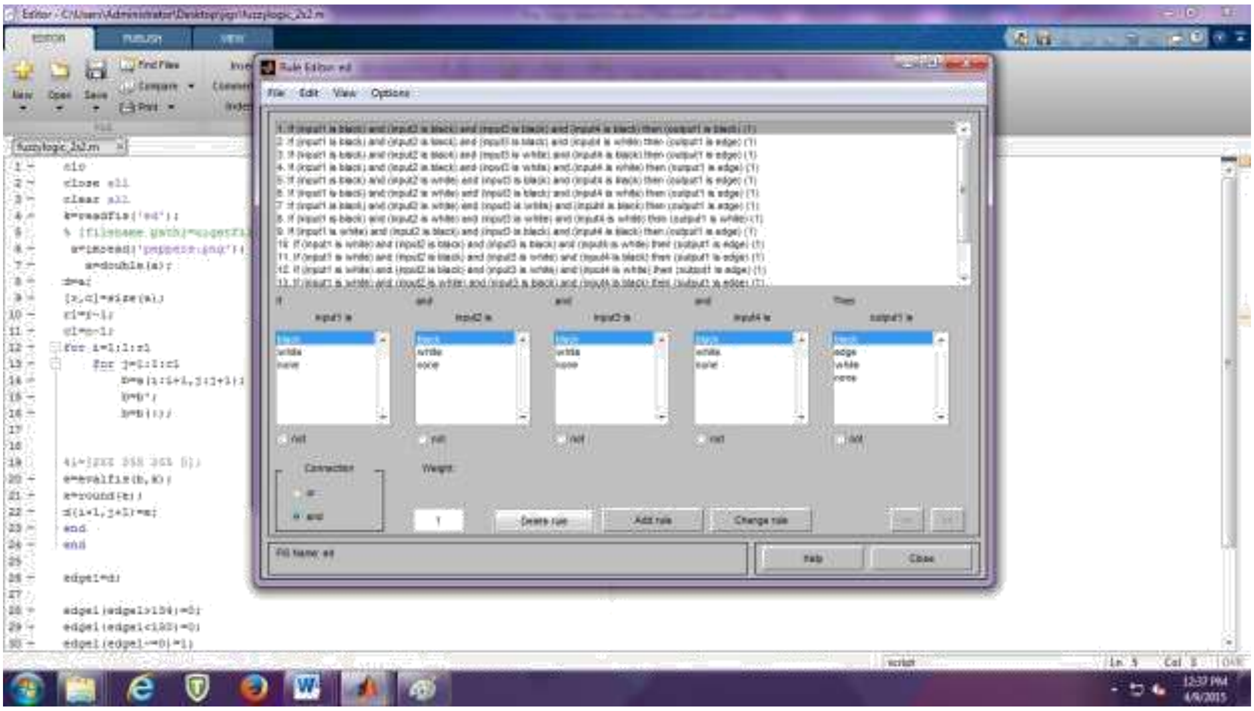




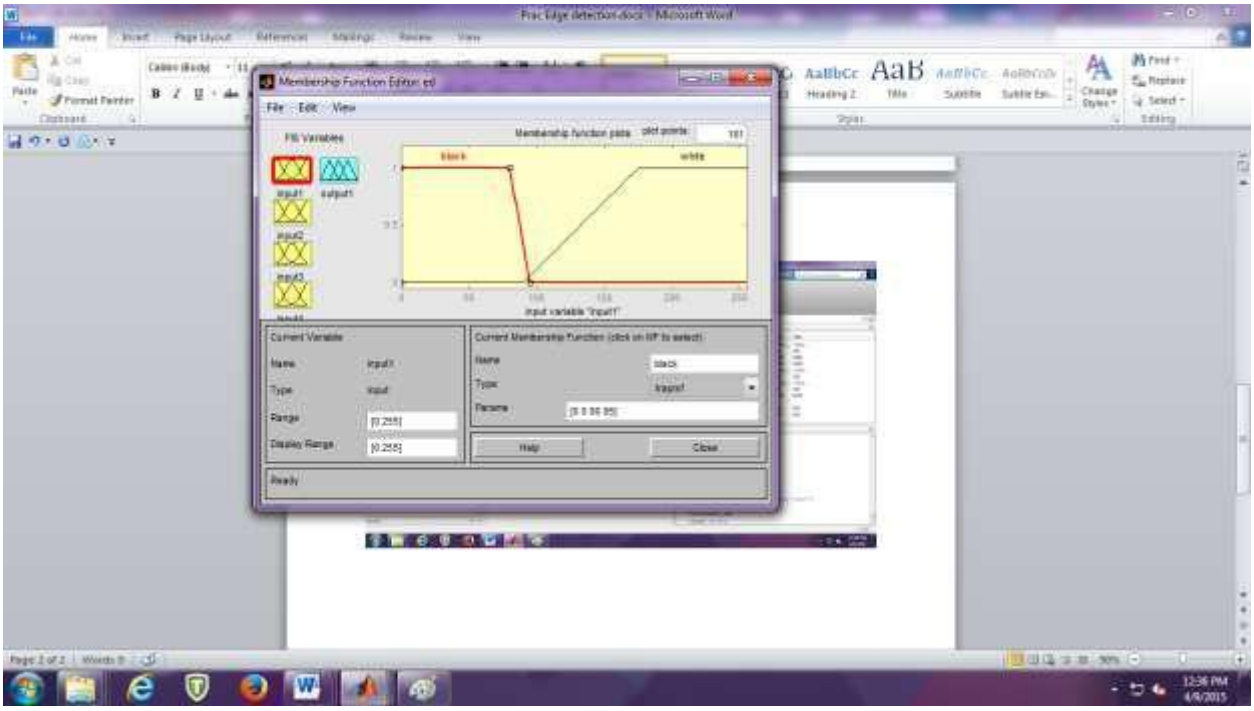


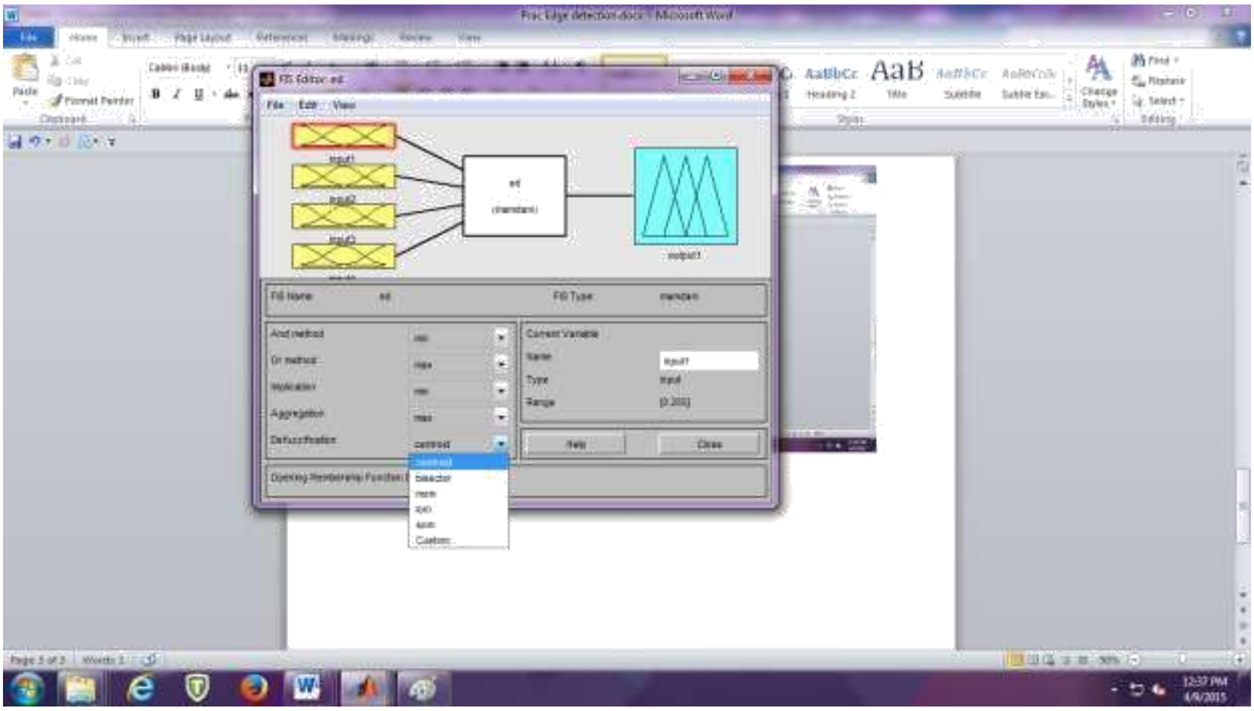
FIS





s





Observation:

- Results of several operators were compared visually and with help of Regionprops function in MATLAB.
- Among gradient operators, Prewitt's results were found better than Sobel and Roberts. Gradient operators are noise sensitive.
- Operator implemented using Fuzzy logic was able to give even better result with sharp edges and noise reduced.

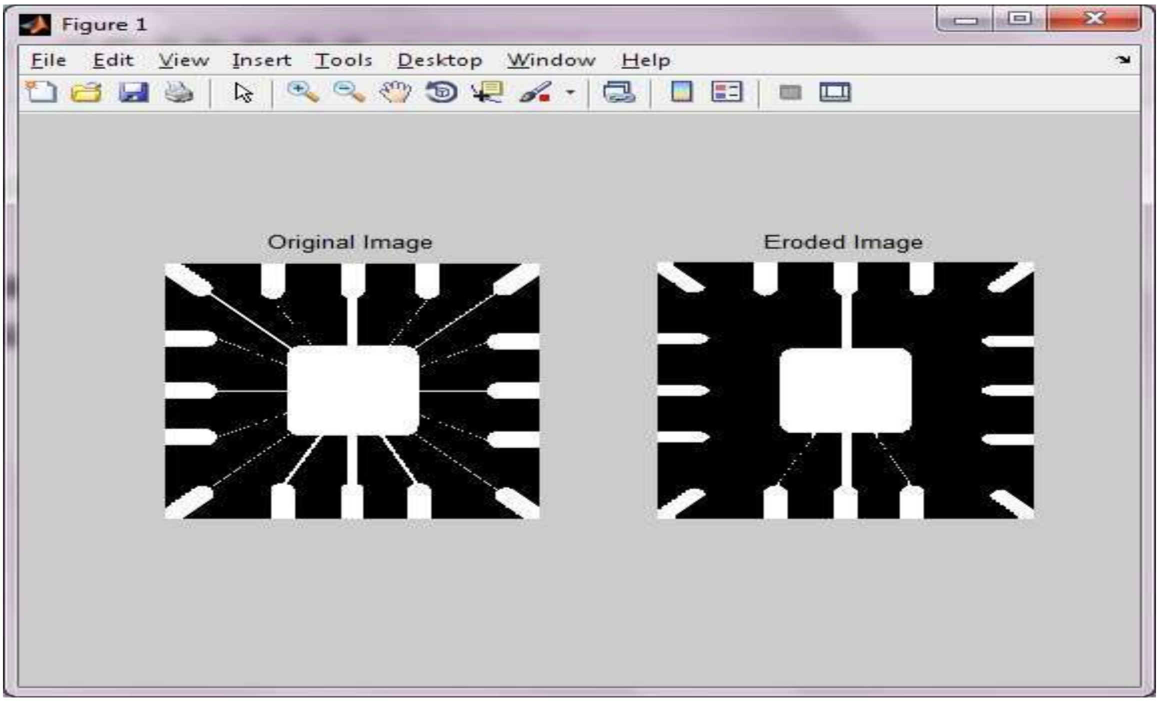
Practical 6

1).Aim: write a program to implement erosion on an image using „imerode" function.

- **Description:Erosion is the set of all points in the image, where the structuring element "fits into".**
- **Consider each foreground pixel in the input image**
  - **If the structuring element fits in, write a "1" at the origin of the structuring element!**
- **Simple application of pattern matching**
- **Input:**
  - **Binary Image (Gray value)**
  - **Structuring Element, containing only 1s!**

Source Code:

```
close all;
clear all;
clc;
originalBW = imread('J:\DIPBOOK\DIP_morphological\erode.tif'); se
= strel('line',11,90);
erodedBW = imerode(originalBW,se);

figure(1),subplot(1,2,1), imshow(originalBW),title('Original Image');
figure(1),subplot(1,2,2), imshow(erodedBW),title('Eroded Image');
```

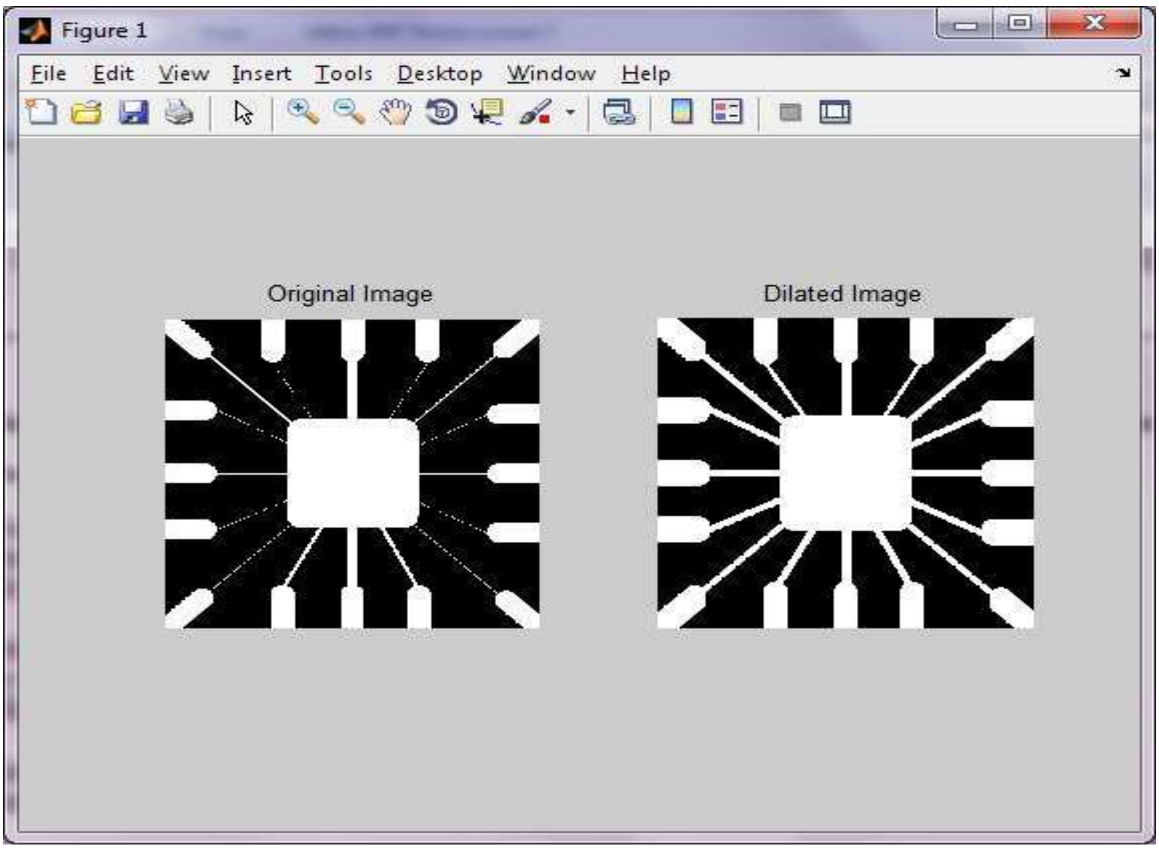


2). Aim: write a program to implement dilation on an image using „imdilate" function.

- **Description:Dilation is the set of all points in the image, where the structuring element "touches" the foreground.**
- **Consider each pixel in the input image**
  - **If the structuring element touches the foreground image, write a "1" at the origin of the structuring element!**
- **Input:**
  - **Binary Image**
  - **Structuring Element, containing only 1s!!**

Source Code:

```
close all;
clear all;
clc;

originalBW = imread('J:\DIPBOOK\DIP_morphological\erode.tif'); se
= strel('line',11,90);
dilatedBW = imdilate(originalBW,se);

figure(1),subplot(1,2,1), imshow(originalBW),title('Original Image');
figure(1),subplot(1,2,2), imshow(dilatedBW),title('Dilated Image');
```

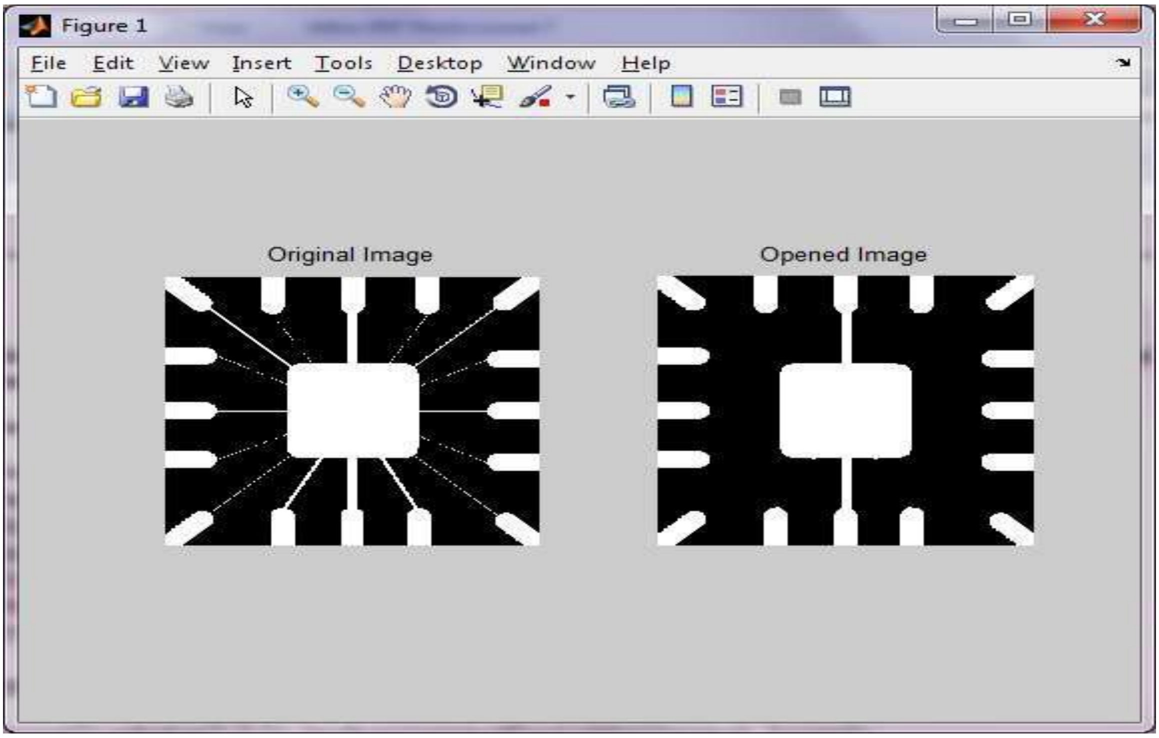


3). Aim:write a program to implement opening operation on an image using „imopen" function.

- **Description:Similar to Erosion**
  - **Spot and noise removal**
  - **Less destructive**
- **Erosion next dilation**
- **the s*ame structuring element for both operations.***
- **Input:**
  - **Binary Image**
  - **Structuring Element, containing only 1s!**

Source Code:

```
%opening operation
%erode first and then dilate
close all;
clear all;
clc;
originalImg=imread('J:\DIPBOOK\DIP_morphological\erode.tif');
sr=strel('square',5);

opened=imopen(originalImg,sr);

figure(1),subplot(1,2,1), imshow(originalImg),title('Original Image');
figure(1),subplot(1,2,2), imshow(opened),title('Opened Image');
```

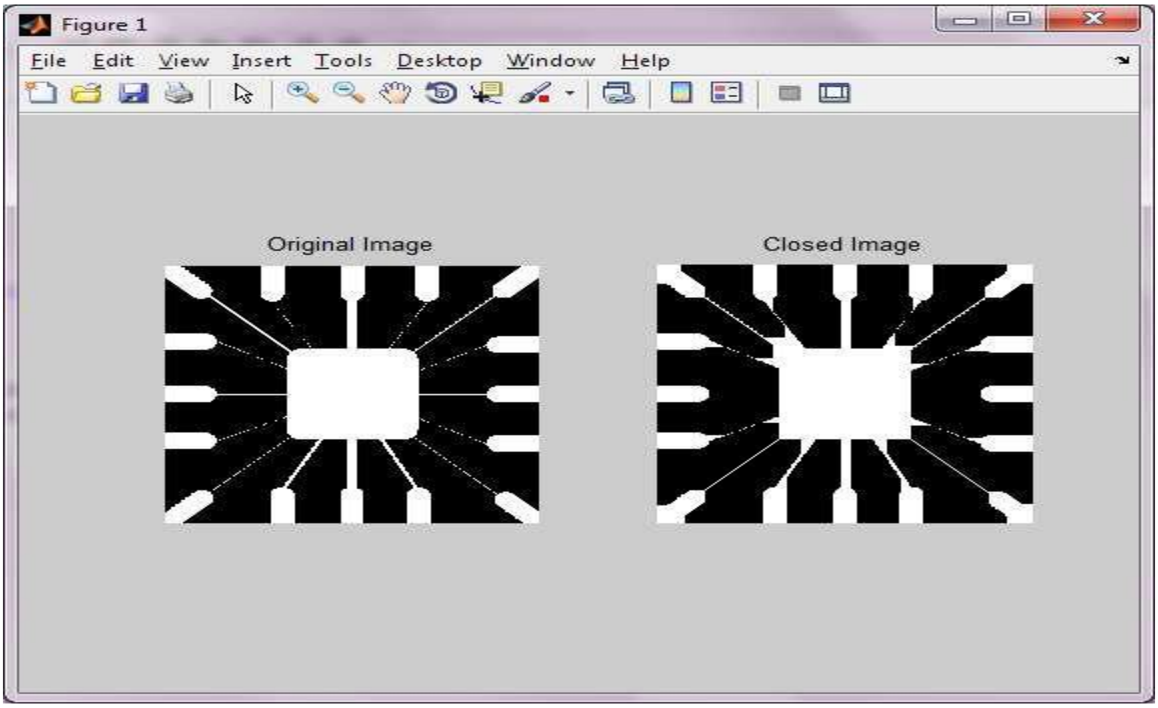


4). Aim:write a program to implement closing operation on an image using „imclose" function.

- **Description:Similar to Dilation**
  - **Removal of holes**
  - **Tends to enlarge regions, shrink background**
- **Closing is defined as a Dilatation, followed by an Erosion*using the same structuring element for both operations.***
- **Dilation next erosion!**
- **Input:**
  - **Binary Image**
  - **Structuring Element, containing only 1s!**

Source Code:

```
%closing operation
%dilate first and then erode

close all;
clear all;
clc;
originalImg=imread('J:\DIPBOOK\DIP_morphological\erode.tif');
sr=strel('square',25);
closed=imclose(originalImg,sr);
figure(1),subplot(1,2,1), imshow(originalImg),title('Original Image');
figure(1),subplot(1,2,2), imshow(closed),title('Closed Image');
```



5). Aim: To write a program to implement binary hit miss operation on an image using „bwhitmiss" function.

- **Description:Used tolook for particular patterns of foreground and background pixels**
- **Very simple object recognition**
- **All other morphological operations can be derived from it!!**
- **Input:**
  - **Binary Image**
  - **Structuring Element, containing 0s and 1s!!**

Source Code:

```
close all;
clear all;
clc;

bw = [0 0 0 0 0 0
      0 0 1 1 0 0
      0 1 1 1 1 0
      0 1 1 1 1 0
      0 0 1 1 0 0
      0 0 1 0 0 0]

interval = [0 -1 -1
1  1 -1
0  1 0]

bw2 = bwhitmiss(bw,interval)
```

Output:

```
bw =

     0     0   0      0     0   0
     0     0   1      1     0   0
     0     1   1      1     1   0
     0     1   1      1     1   0
     0     0   1      1     0   0
     0     0   1      0     0   0



interval =

     0    -1  -1
   1     1   -1
   0     1   0
```

```
bw2 =

     0     0     0     0     0     0
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

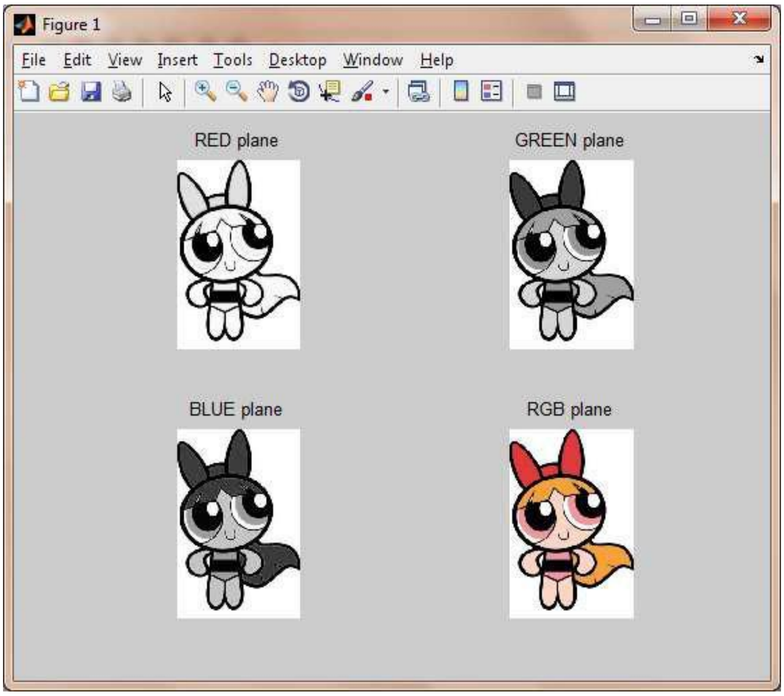
Practical 7

1). Aim: Separate R, G and B planes from an RGB image.

Source Code:

```
closeall;
clearall;
clc;
i1 = imread('D:\Photos\blossom1.jpg');
r1 = i1(:,:,1);
g1 = i1(:,:,2);
b1 = i1(:,:,3);
figure(1);
subplot(2,2,1);
imshow(r1);title('RED plane');
subplot(2,2,2);
imshow(g1); title('GREEN plane');
subplot(2,2,3);
imshow(b1); title('BLUE plane');
i1 = double(i1);
[row col dim] = size(i1);
plane = padarray(row,col);
im = cat(3,r1,g1,b1);
subplot(2,2,4);
imshow(im);title('RGB plane');
```

Output:

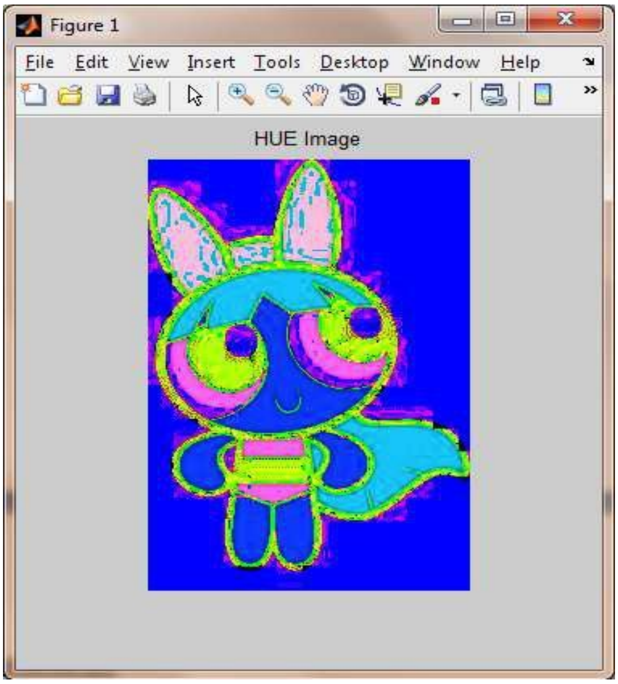


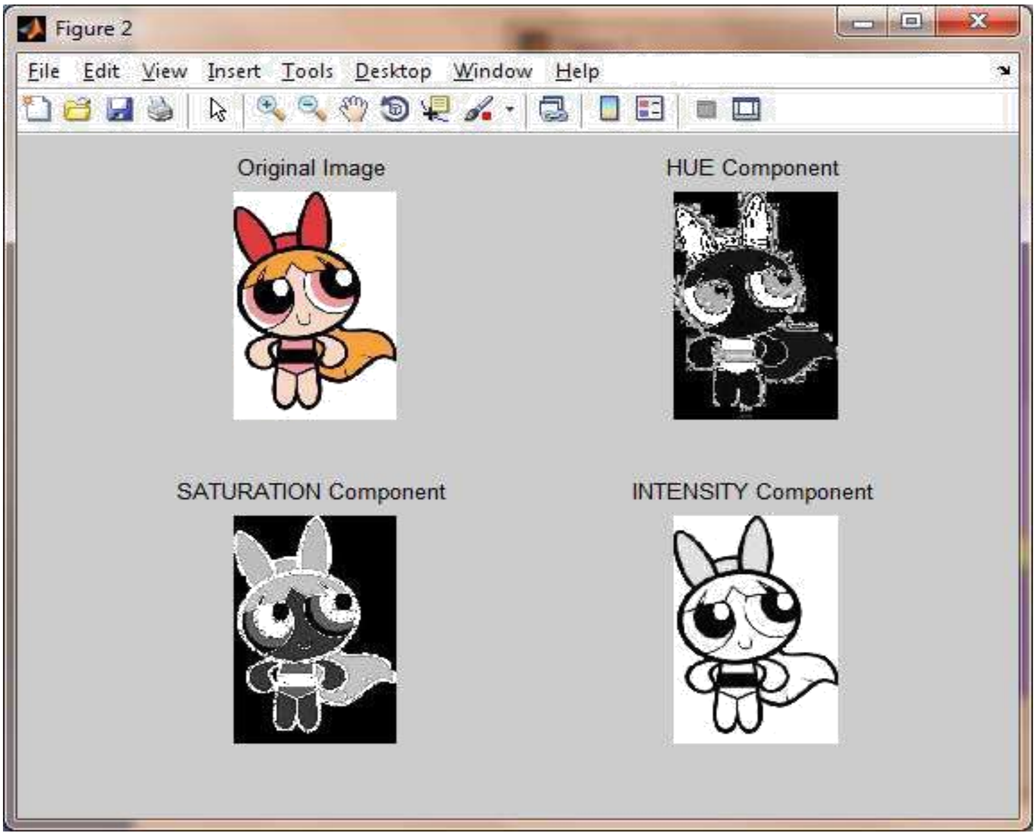
2). Aim: Convert RGB image to HSI.

Source Code:

```
closeall;
clearall;
clc;
im1 = imread('D:\Photos\blossom1.jpg');
HSV = rgb2hsv(im1);
figure(1);
imshow(HSV);%HUE image combine
title('HUE Image');
H=HSV(:,:,1);
S=HSV(:,:,2);
V=HSV(:,:,3);
figure(2);
subplot(2,2,1), imshow(im1); title('Original Image');
subplot(2,2,2), imshow(H); title('HUE Component');
subplot(2,2,3), imshow(S); title('SATURATION Component');
subplot(2,2,4), imshow(V); title('INTENSITY Component');
```

Output:





Aim: Convert Image RGB to CMY.

Source Code:

```
clearall;
closeall;
clc;
im = imread('D:\Photos\blossom1.jpg');
R = im(:,:,1);
G = im(:,:,2);
B = im(:,:,3);
R1 = im2double(R);
G1 = im2double(G);
B1 = im2double(B);
c = 1-R1;
m = 1-G1;
y = 1-B1; figure(2);
subplot(2,4,1);
imshow(im);title('orignal image');
subplot(2,4,2)

imshow(c); title('cyan plane');
subplot(2,4,3);
imshow(m); title('magenta plane');
subplot(2,4,4);
imshow(y); title('yellow plane');
```

Output:

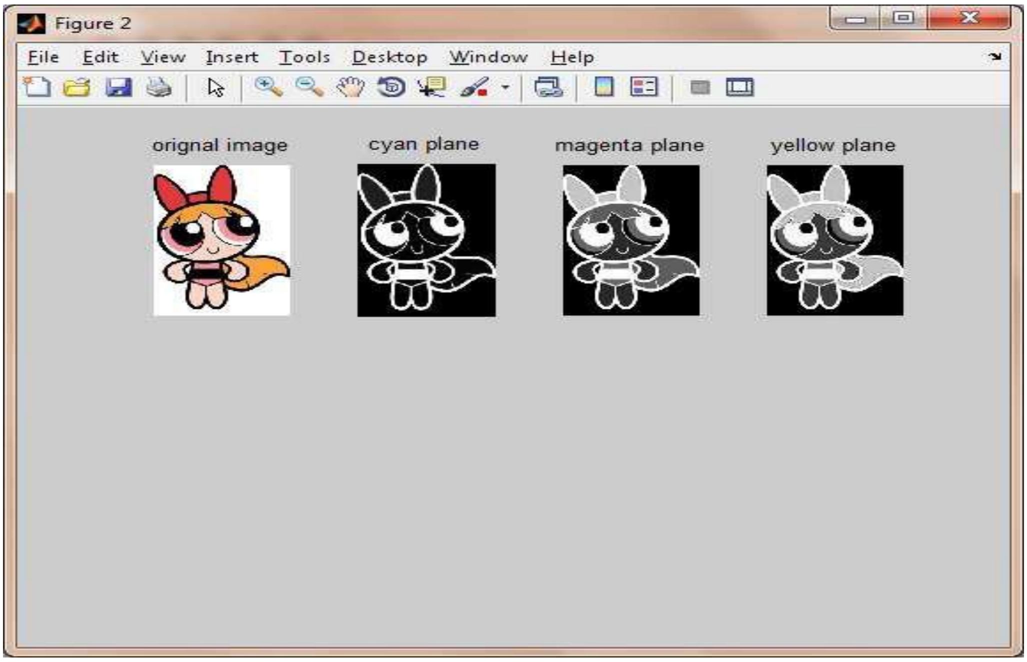


Figure 2
File Edit View Insert Tools Desktop Window Help
orignal image
cyan plane
magenta plane
yellow plane

Exp-8 study of image filtering in transform domain.

```
% Filtering in the transform domain
clear all;

f = double(imread('.\img\cameraman.tif'));
F = fftshift(fft2(f));

[u,v] = meshgrid(1:256,1:256);
z = (u-129).^2 + (v-129).^2;
for r = 128:-9:1;                % sequence of 'ideal' disk-shaped filters
    H = double(z<r.^2);        %  lowpass
%    H = double(z>r.^2);      %  highpass
    F1 = F;  F1(round(sqrt(z))==r)=1e6;
    figure(1);  imshow(log(abs(F1)),[]);
    G = F .* H;
    g = real(ifft2(ifftshift(G)));
    figure(2); imshow(g,[]); pause;
    mean(g(:).^2)       % note: image energy is reduced
    end;
return;

%% ------------------------------------------------------------------------

for r = 128:-9:1;                % sequence of Gaussian filters
    H = exp(-z/(2*r.^2));        %  lowpass
%    H = 1 - H;                     %  highpass
    figure(1); mesh(H);
    G = F .* H;
    g = real(ifft2(ifftshift(G)));
    figure(2); imshow(g,[]); pause;
    end;

%% ---------------------------------------------------------------------

H = -z;         %   Laplacian filter (see dip04, slide 55)
 figure(1); mesh(H);        % beware!! the Laplacian has huge gain!!
G = F .* H;
g = real(ifft2(ifftshift(G)));
 figure(2); imshow(g,[]);

%%
 H = 1+z;          %  subtraction of the Laplacian,
% H = 1 + .0001 * z;    %  i.e. unsharp masking (slides 58-59)
 figure(1); mesh(H)
G = F .* H;
g = real(ifft2(ifftshift(G)));
 figure(2); imshow(g,[]);  % pause;
% figure(3); imshow(histeq(g/max(g(:))));  % sharpening + hist.eq., slide
60
return;
```

Exp- 9 Implementation of homomorphic filter.

```
% Homomorphic filtering
clear all;

 f = double(imread('.\img\cameraman.tif'));
% f = double(imread('.\img\clip_bw_av.tif'));

figure(1);  imshow(f,[]);
f = log(f+1);
F = fftshift(fft2(f));

[nc,nr] = size(f);
```

```
nr2 = nr/2+1;      nc2 = nc/2+1;     % (image size should be even)
[u,v] = meshgrid(1:nr,1:nc);
z = (u-nr2).^2 + (v-nc2).^2;
 gamma_h = 1;  gamma_l = .2;  wdth = 200;      % cameraman
% gamma_h = 1.4;  gamma_l = .5;  wdth = 100;     % clip_bw_av
H = (gamma_h - gamma_l).*(1 - exp(-.9 .* z/(wdth.^2))) + gamma_l;

% figure(1); mesh(H); return;
G = F .* H;
g = real(ifft2(ifftshift(G)));
g = exp(g)-1;
figure(2); imshow(g,[]);
return;
```

Exp- 10 implementation of non-linear filter.

```
% Nonlinear filters
clear all;

f = double(imread('.\img\cameraman.tif'));
[m,n] = size(f);

 fn = f + randn(m,n)*20;       %  additive Gaussian noise
% fn = f .* ((rand(m,n)-.5)*.6+1);      % uniform multiplicative noise in
[0.7 , 1.3]
% fn = f;  z = rand(m,n);   fn(z<.005)=0;  fn(z>1-.005)=255;       %
impulse noise, prob. 2*.005
% z = rand(m,n);   fn(z<.005)=0;  fn(z>1-.005)=255;      % mixed impulse-
other noise

fn(fn<0) = 0;   fn(fn>255) = 255;   %  avoid negative values, in order to
use log(fn) later
                                    %  note: actual noise variance will be
less than expected

w = [1 1 1; 1 1 1; 1 1 1] / 9;

return;
%%
g1 = filter2(w,fn);              % averaging filter (arithmetic mean)
g2 = exp(filter2(w,log(fn+.01)));        % geometric mean
 imshow([f, fn; g1, g2],[0 255]);
 PSNR_noisy = 10 * log10( 256^2 / (mean(mean((fn-f).^2))) )
 PSNR_g1 = 10 * log10( 256^2 / (mean(mean((g1-f).^2))) )
 PSNR_g2 = 10 * log10( 256^2 / (mean(mean((g2-f).^2))) )
%%
q = 1.7;      % q>0 for dark impulse noise, q<0 for light impulse noise
g3 = filter2(w,fn.^(q+1)) ./ filter2(w,fn.^q);   % contraharmonic mean
g4 = medfilt2(fn, [3 3]);                         % median filter on a 3*3
support
 imshow([f, fn; g3, g4],[0 255]);
 PSNR_noisy = 10 * log10( 256^2 / (mean(mean((fn-f).^2))) )
 PSNR_g3 = 10 * log10( 256^2 / (mean(mean((g3-f).^2))) )
 PSNR_g4 = 10 * log10( 256^2 / (mean(mean((g4-f).^2))) )
 %%
dum(:,:,1) = fn;                            % median filter on a 3*3 'plus'-
shaped support
dum(:,:,2) = [zeros(1,n); fn(1:m-1,:)];
dum(:,:,3) = [fn(2:m,:); zeros(1,n)];
dum(:,:,4) = [zeros(m,1)  fn(:,1:n-1)];
dum(:,:,5) = [fn(:,2:n)  zeros(m,1)];
g5 = median(dum,3);
 imshow([f, fn; g4, g5],[0 255]);
```

```
 PSNR_noisy = 10 * log10( 256^2 / (mean(mean((fn-f).^2))) )
 PSNR_g4 = 10 * log10( 256^2 / (mean(mean((g4-f).^2))) )
 PSNR_g5 = 10 * log10( 256^2 / (mean(mean((g5-f).^2))) )
 %%
dum(:,:,6) = fn;                          % center-weighted median filter,
with weight 3
dum(:,:,7) = fn;
g6 = median(dum,3);
 imshow([f, fn; g5, g6],[0 255]);
 PSNR_noisy = 10 * log10( 256^2 / (mean(mean((fn-f).^2))) )
 PSNR_g5 = 10 * log10( 256^2 / (mean(mean((g5-f).^2))) )
 PSNR_g6 = 10 * log10( 256^2 / (mean(mean((g6-f).^2))) )
 %%
g7 = fn;
for i=2:m-1;   for j=2:n-1;            % alpha-trimmed, 3*3, d=4
   dum = fn(i-1:i+1,j-1:j+1);
   dum = sort(dum(:));
   g7(i,j) = mean(dum(3:7));
   end; end;
 imshow([f, fn; g1, g7],[0 255]);
 PSNR_noisy = 10 * log10( 256^2 / (mean(mean((fn-f).^2))) )
 PSNR_g1 = 10 * log10( 256^2 / (mean(mean((g1-f).^2))) )
 PSNR_g7 = 10 * log10( 256^2 / (mean(mean((g7-f).^2))) )

return;

% la media geom. espande i neri e quindi sul rumore impulsivo evidenzia il
pepper e
% ripulisce il salt
```

Exp – 11 Implementation of inverse filter and wiener filter.

```
% Motion blur and turbulence blur restoration with inverse filter and
Wiener filter
clear all;

% f = double(imread('.\img\lenar.tif'));
f = double(imread('.\img\lena.tif'));  f = f(101:450,51:400);
[m,n] = size(f);

[u,v] = meshgrid(1:n,1:m);
m2 = m/2+1;  n2 = n/2+1;
a = 1/32;     % equivalent to the average of a*m shots, each with a one-
pixel shift
                    % for m=256, try with a = 1/128 ... 1/32 ... 1/8
PSF = sin(pi*(u-m2)*a) .* exp(-j*pi*(u-m2)*a) ./ (pi*(u-m2)*a);        %
Motion degradation model
PSF(:,m2)=1;   %  when u=m2 the PSF is sin(0)/0, a Matlab NaN

% PSF = exp(-.0002*((u-m2).^2+(v-n2).^2).^(5/6));     %  Atmospheric
turbulence, k = .0002 .. .002
 figure(1);  mesh(abs(PSF));

F = fftshift(fft2(f));
Fb = F .* PSF;
fb = real(ifft2(ifftshift(Fb)));            %  blurred image
sdn = 10;   fb = fb + randn(m,n)*sdn;                %  ...plus noise
% fb = round(fb);  fb(fb>255)=255;  fb(fb<0)=0;    % image 'saved' as uint8
--> quantiz. noise

 G1 = Fb ./ PSF;      %  not realistic, only for reference
 G2 = fftshift(fft2(fb)) ./ PSF;      %  inverse filter
 H = PSF;   H(abs(PSF)<.2)=.2;  % 'raise' zeros in the PSF
 G3 = fftshift(fft2(fb)) ./ H;         %  clipped inverse filter
```

```
g1 = real(ifft2(ifftshift(G1)));
g2 = real(ifft2(ifftshift(G2)));
g3 = real(ifft2(ifftshift(G3)));
figure(2);  imshow([fb, g1; g2, g3],[0 255]);

Hw = (conj(PSF) .* F.^2) ./ (PSF.^2 .* F.^2 + sdn^2);   %  Wiener filter
% Hw(abs(Hw)>5)=5;  % clip peaks in the Wiener filter
 figure(3);  mesh(abs(Hw));
Gw = fftshift(fft2(fb-mean(fb(:)))) .* Hw;
gw = real(ifft2(ifftshift(Gw))) + mean(fb(:));
figure(4);  imshow([fb, gw],[0 255]);

psf = real(fftshift(ifft2(ifftshift(PSF))));
nsr = sdn.^2 ./ sum( (f(:)/255).^2);
gwm = deconvwnr(fb/255,psf,nsr);  % Matlab's Wiener filter
% figure(5);  mesh(abs(psf));
 figure(6);  imshow([fb, gwm*255],[0 255]);

return;
```