BHARATI VIDYAPEETH DEEMED UNIVERSITY
COLLEGE OF ENGINEERING, PUNE - 43

DEPARTMENT OF COMPUTER ENGINEERING

Lab Manual

Image Processing and Pattern Recognition

B. Tech Computer Sem VIII

VISION OF THE INSTITUTE

Vision of the Institute

"To be World Class Institute for Social Transformation through Dynamic Education"

MISSION OF THE INSTITUTE

A. To provide quality technical education with advanced equipment, qualified faculty members, infrastructure to meet needs of profession and society.

B. To provide an environment conducive to innovation, creativity, research, and entrepreneurial leadership.

C. To practice and promote professional ethics, transparency and accountability for social community, economic and environmental conditions. VISION OF THE DEPARTMENT To pursue and excel in the endeavor for creating globally recognized Computer Engineers

VISION OF THE DEPARTMENT

"To pursue and excel in the endeavor for creating globally recognized computer engineers through quality education."

Mission of the Department

- To impart engineering knowledge and skills confirming to a dynamic curriculum.
- To develop professional, entrepreneurial & research competencies encompassing continuous intellectual growth.
- To produce qualified graduates exhibiting societal and ethical responsibilities in working environment

PROGRAM EDUCATIONAL OBJECTIVES(PEOs):

1. Demonstrate technical and professional competencies by applying engineering fundamentals, computing principles and technologies.
2. Learn, Practice, and grow as skilled professionals/ entrepreneur/researchers adapting to the evolving computing landscape.
3. Demonstrate professional attitude, ethics, understanding of social context and interpersonal skills leading to a successful career.

ROGRAM SPECIFIC OUTCOMES(PSO)s:

PSO 1: To design, develop and implement computer programs on hardware towards solving problems.

PSO 2: To employ expertise and ethical practise through continuing intellectual growth and adapting to the working environment.

PROGRAMME OUTCOMES(POs):

Upon completion of the course the graduate engineers will be able to:
1. Apply the knowledge of mathematics, science, engineering fundamentals, and computing for the solution of complex engineering problems.
2. Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using computer engineering foundations, principles, and technologies.
3. Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.
7. Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and the need for sustainable development.
8. Apply ethical principles while committed to professional responsibilities and norms of the engineering practice.
9. Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10. Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. 11. Apply the engineering and management principles to one's work, as a member and leader in a team.
12. Recognise the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

GENRAL INSTUCTIONS:

- Equipment in the lab is meant for the use of students. Students need to maintain aproper decorum in the computer lab. Students must use the equipment with care.
- Students are required to carry their reference materials, files and records with completed assignment while entering the lab.
- Students are supposed to occupy the systems allotted to them and are not supposed totalk or make noise in the lab.
- All the students should perform the given assignment individually.
- Lab can be used in free time/lunch hours by the students who need to use the systemsshould take prior permission from the lab in-charge.
- All the Students are instructed to carry their identity cards when entering the lab.
- Lab files need to be submitted on or before date of submission.
- Students are not supposed to use pen drives, compact drives or any other storage devices in the lab.
- For Laboratory related updates and assignments students should refer to the noticeboard in the Lab.

Course Name: **-**Image Processing and Pattern Recognition

EEKLY PLAN:

Week No.	Practical/Assignment Topic	Problem Definition
1	Grayscale Images	Display of Grayscale Images.
2	Flipped Image	Write a MATLAB code that reads a gray scale image and generates the flipped image of original image.
3	Enhance contrast	To enhance contrast using Histogram Equalization
4	Image enhancement	Write a program for image enhancement.
5	Image compression	Write a program for image compression
6	Edge detection	Write a program for Edge detection
7	Image segmentation	Write a program for image segmentation
8	Image morphology	Write a program for image morphology
9	Pattern recognition	Illustrate and discuss use of various method of pattern recognition.
10	Face detection	Write a program for face detection in MATLAB.

EXAMINATION SCHEME

End Semester Examination: 60
Internal Assessment: 40
Term Work & Practical: 50

PROCEDURE OF EVALUATION

Each practical/assignment shall be assessed continuously on the scale of 25 marks. The distribution of marks as follows.

Sr. No	Evaluation Criteria	Marks for each Criteria	Rubrics
1	**Timely Submission**	**07**	➢ Punctuality reflects the work ethics. Students should reflect that work ethics by completing the lab assignments and reports in a timely manner without being reminded or warned.
2	**Presentation**	**06**	➢ Student are expected to write the technical document (lab report) in their own words. The presentation of the contents in the lab report should be complete, unambiguous, clear, understandable. The report should document approach/algorithm/design and code with proper explanation.
3	**Understanding**	**12**	➢ Correctness and Robustness of the code is expected. The Learners should have an in-depth knowledge of the practical assignment performed. The learner should be able to explain methodology used for designing and developing the program/solution. He/she should clearly understand the purpose of the assignment and its outcome.

LABORATORY USAGE

Students use computers for executing the lab experiments, document the results and to prepare the technical documents for making the lab reports.

OBJECTIVE

1 Students should be able to understand digital image processing and advanced concepts.

2 Students should be able to properly implement algorithms using modern computing tools such as MATLAB, and to interpret and present the results.

3 To study fundamentals of color Image Processing.

PRACTICAL PRE-REQUISITE

Set theory, Linear algebra and statistics, Computer Graphics and visualization, Signals and system, Digital signal processing.

SOFTWARE REQUIREMENTS

- Intel based desktop PC with minimum of 166 MHZ or faster processor with at least 64MB RAM and 100MB free disk space, MATLAB.

Hours Per Week: 2 Hrs.

Course Outcomes: Upon Completion of the course the students will able to

1. Explain the digital image processing and digital image formation.
2. Illustrate different mathematical preliminaries to deal with digital imageprocessing
3. Explain the concept of Image restoration and image segmentation.
4. Apply the concept of pattern recognition and its different phases
5. Apply knowledge/ skills for solving real world problems.
6. Describe applications of DSP in speech and Image Processing

HOW ARE OUTCOMES ASSESSED?

Course Outcome	Assignment Number	Level	Proficiency evaluated by
To explain the digital image processing and digital image formation.	**1,2,3,4,5,7,9**	**3,3,3,3,3,3,3**	Performing Practical and reporting results
To illustrate different mathematical preliminaries to deal with digital image processing	**6**	**3**	Problem definition &Performing Practical and reporting results
To explain the concept of Image restoration and image segmentation.	**6**	**2**	Performing experiments and reporting results
To apply the concept of patter recognition and its different phases	**8,9,10**	**3,3,3**	Performing experiments and reporting results
To apply knowledge/ skills for solving real world problems.	**9,10**	**3,3**	Performing experiments and reporting results
To describe Applications of DSP in speech and image processing	**9,10**	**2,3**	Performing experiments and reporting results

CONTRIBUTION TO PROGRAM OUTCOMES and PROGRAM SPECIFIC OBJECTIVES

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	lPO12	**PSO1**	**PSO2**
CO1	3	2	3	2	3	3		3		3		2		
CO2	2	3		2		3		2	1			3		
CO3				2								3		
CO4	2	3	2	3		2		2	2			3		
CO5	1	-	3	-	-			3	3	3		3		

DESIGN EXPERIENCE GAINED

The students gain moderate design experience by Identifying, Demonstrating and applying their knowledge by analyzing image processing problems and recognizing and employing (or proposing) effecti**ve** solutions

LEARNING EXPERIENCE GAINED

Students will Design and create practical solutions to a range of common image processing problems and to critically assess the results of their solutions, including shortcomings.

MATLAB - Overview

MATLAB (matrix laboratory) is a fourth-generation high-level programming language and interactive environment for numerical computation, visualization, and programming.

MATLAB is developed by Math Works.

It allows matrix manipulations; plotting of functions and data; implementation of algorithms; creation of user interfaces; interfacing with programs written in other languages, including C, C++, Java, and FORTRAN; analyse data; develop algorithms; and create models and applications.

It has numerous built-in commands and math functions that help you in mathematical calculations, generating plots, and performing numerical methods.

MATLAB's Power of Computational Mathematics

MATLAB is used in every facet of computational mathematics. Following are some commonly used mathematical calculations where it is used most commonly

- Dealing with Matrices and Arrays
- 2-D and 3-D Plotting and graphics
- Linear Algebra
- Algebraic Equations
- Non-linear Functions
- Statistics
- Data Analysis
- Calculus and Differential Equations
- Numerical Calculations
- Integration
- Transforms
- Curve Fitting
- Various other special functions

Features of MATLAB

Following are the basic features of MATLAB

- It is a high-level language for numerical computation, visualization and application development.

- It also provides an interactive environment for iterative exploration, design and problem solving.
- It provides vast library of mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration and solving ordinary differential equations.
- It provides built-in graphics for visualizing data and tools for creating custom plots.
- MATLAB's programming interface gives development tools for improving code quality maintainability and maximizing performance.
- It provides tools for building applications with custom graphical interfaces.
- It provides functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET and Microsoft Excel.

Uses of MATLAB

MATLAB is widely used as a computational tool in science and engineering encompassing the fields of physics, chemistry, math and all engineering streams. It is used in a range of applications including

- Signal Processing and Communications
- Image and Video Processing
- Control Systems
- Test and Measurement
- Computational Finance
- Computational Biology

MATLAB - Environment Setup

Try it Option Online

You really do not need to set up your own environment to start learning MATLAB Octave. Reason is very simple, we already have set up MATLAB Octave environment online, so that you can execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try the following example using **Try it** option available at the top right corner of the below sample code box −

```
x =[12345678910];
y1 =[.16.08.04.02.013.007.004.002.001.0008];
y2 =[.16.07.03.01.008.003.0008.0003.00007.00002];

semilogy(x,y1,'-bo;y1;',x,y2,'-kx;y2;');
title('Plot title');
xlabel('X Axis');
ylabel('Y Axis');
print-deps graph.eps
```

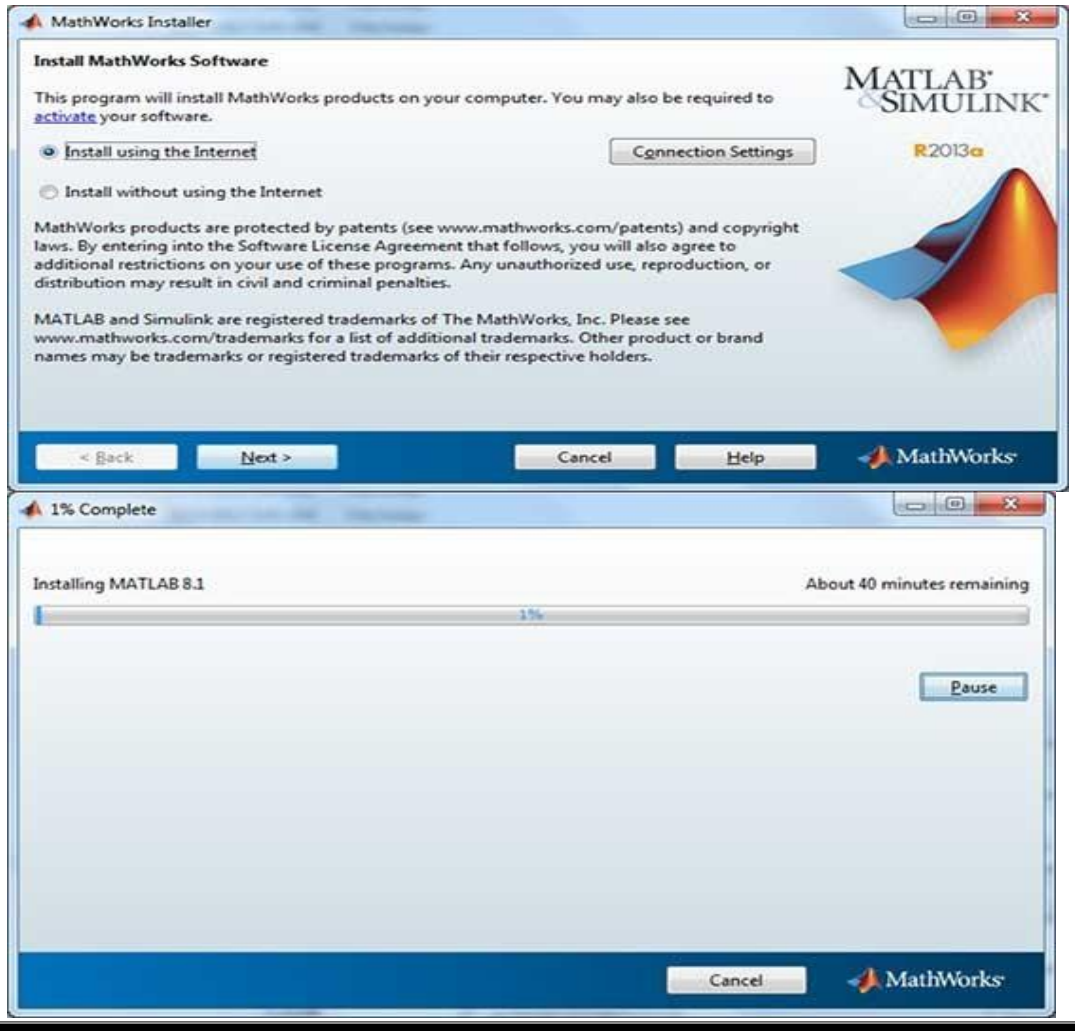
For most of the examples given in this tutorial, you will find **Try it** option, so just make use of it and enjoy your learning.

Local Environment Setup

Setting up MATLAB environment is a matter of few clicks. The installer can be downloaded from here.

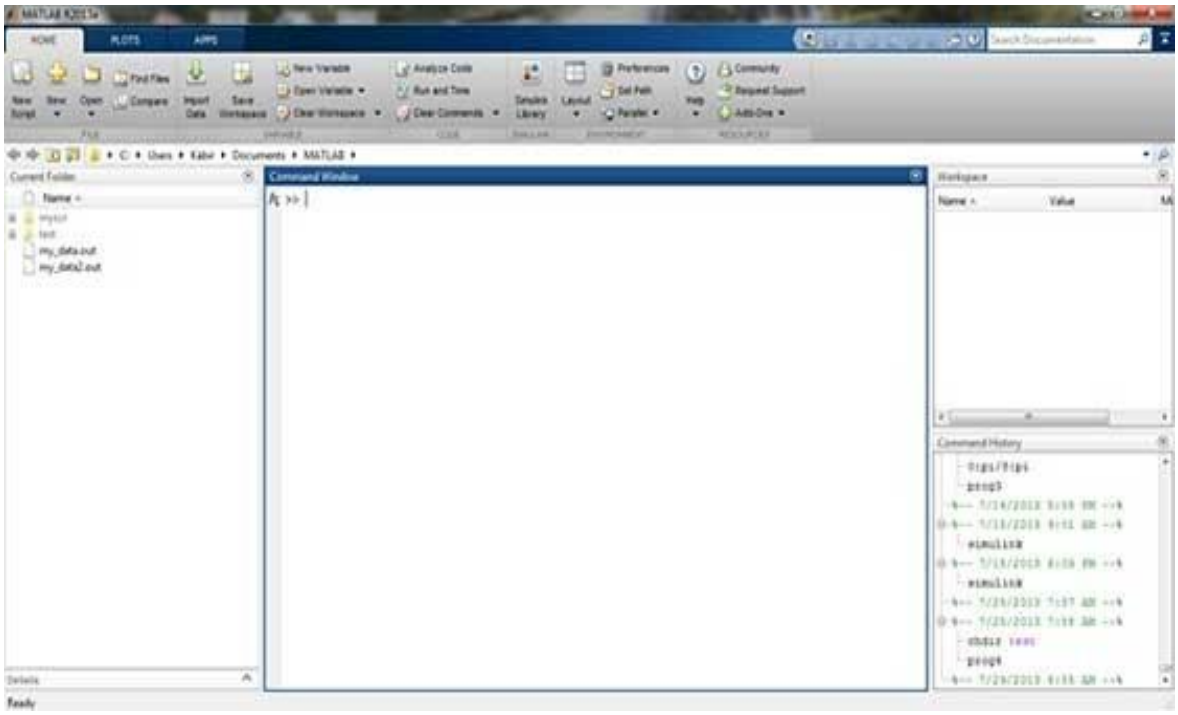
MathWorks provides the licensed product, a trial version and a student version as well. You need to log into the site and wait a little for their approval.

After downloading the installer the software can be installed through few clicks.



Understanding the MATLAB Environment

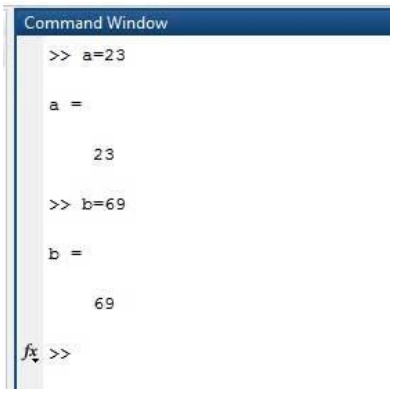
MATLAB development IDE can be launched from the icon created on the desktop. The main working window in MATLAB is called the desktop. When MATLAB is started, the desktop appears in its default layout −



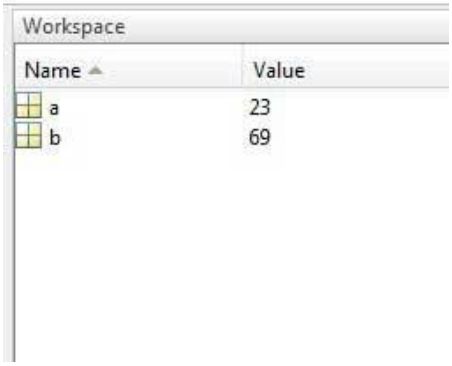
The desktop has the following panels

- **Current Folder** This panel allows you to access the project folders and files.

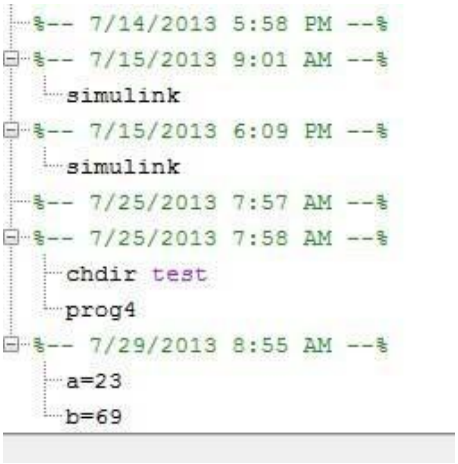
- **Command Window** − This is the main area where commands can be entered at the command line. It is indicated by the command prompt (>>).



- **Workspace** − The workspace shows all the variables created and/or imported from files.



- **Command History** − This panel shows or rerun commands that are entered at the command line.



Set up GNU Octave

If you are willing to use Octave on your machine (Linux, BSD, OS X or Windows), then kindly download latest version from Download GNU Octave. You can check the given installation instructions for yourmachine.

Objectives:

- To write and execute programs in C/C++ to solve problems using data structures such as arrays, linked lists, stacks, queues.
- To write and execute write programs in C/C++ to implement various sorting and searching methods.

Outcome

- Ability to identify the appropriate data structure for given problem
- Graduate able to design and analyze the time and space complexity of algorithm or program.
- Ability to effectively use compilers includes library functions, debuggers, and trouble shooting.

LIST OF PRACTICAL ASSIGNMENTS:

No	Detail of Assignment
1	Display of Grayscale Images.
2	Write a MATLAB code that reads a gray scale image and generates the flipped image of original image.
3	To enhance contrast using Histogram Equalization
4	Write a program for image enhancement.
5	Write a program for image compression
6	Write a program for Edge detection
7	Write a program for image segmentation
8	Write a program for image morphology
9	Illustrate and discuss use of various method of pattern recognition.
10	Write a program for face detection in MATLAB.

Assignment No.1

Title : Display of Grayscale Images.

Aim : To implement Matlab grayscale image functions.

Theory :

Syntax

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

imshow(I) displays image I in a 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 optimizes figure, axes, and image object properties for image display.

imshow(X,map) displays the indexed image X with the colormap map. A colormap matrix can 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(filename) displays the image stored in the graphics file specified by filename.

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

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

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

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

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

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

Example:

Display a grayscale image by reading an RGB image into the workspace and converting it to a grayscale image.

Read RGB image into the workspace.

```
RGB = imread('peppers.png');
```

Convert the image to grayscale.

```
I = rgb2gray(RGB);
```

Display the grayscale image.

```
imshow(I)
```

Assignment No.2

Title : Write a MATLAB code that reads a gray scale image and generates the flipped image of original image.

Aim : To implement imrotate functions to flipped image

Theory:

Syntax

```
B = imrotate(A,angle)

B = imrotate(A,angle,method)

B = imrotate(A,angle,method,bbox)

gpuarrayB = imrotate(gpuarrayA,method)
```

Description

B = imrotate(A,angle) rotates image A by angle degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for angle. imrotate makes the output image B large enough to contain the entire rotated image. imrotate uses nearest neighbor interpolation, setting the values of pixels in B that are outside the rotated image to 0 (zero).

B = imrotate(A,angle,method) rotates image A, using the interpolation method specified by method.

B = imrotate(A,angle,method,bbox) rotates image A, where bbox specifies the size of the output image. If you specify 'cropped', imrotate makes the output image the same size as the input image. If you specify 'loose', imrotate makes the output image large enough to include the entirety of the rotated image.

gpuarrayB = imrotate(gpuarrayA,method) perform operation on a graphics processing unit (GPU), where gpuarrayA is a gpuArray object that contains a grayscale or binary image, and the output image is a gpuArray object. This syntax requires the Parallel Computing Toolbox™.

Examples

Rotate Image Clockwise for Better Horizontal Alignment
Try this Example

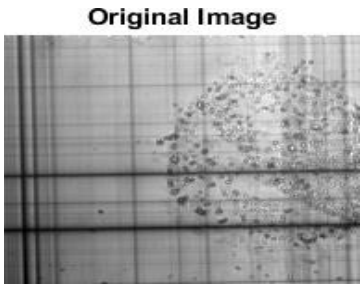
Read an image into the workspace, and convert it to a grayscale image.

```
I = fitsread('solarspectra.fts');
I = rescale(I);
```

Display the original image.

```
figure
imshow(I)
```

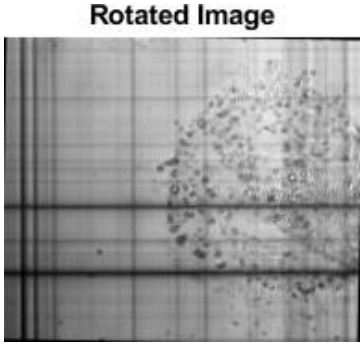
title('Original Image')



Rotate the image 1 degree clockwise to bring it into better horizontal alignment. The example specified bilinear interpolation and requests that the result be cropped to be the same size as the original image.

J = imrotate(I,-1,'bilinear','crop');
Display the rotated image.

figure
imshow(J)
title('Rotated Image')



Rotate Image on GPU

Read image into a gpuArray object.

X = gpuArray(imread('pout.tif'));
Rotate the image, performing the operation on the graphics processing unit (GPU).

Y = imrotate(X, 37, 'loose', 'bilinear');
Display the rotated image.

figure; imshow(Y)

Input Arguments

A — Image to be rotated

real, nonsparse, numeric or logical array

Image to be rotated, specified as a real, nonsparse, numeric, or logical array.

DataTypes: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical

Value	Description
'crop'	Make output image B the same size as the input image A, cropping the rotated image to fit
'loose'	Make output image B large enough to contain the entire rotated image. B is larger than A.

angle — Amount of rotation in degrees
numeric scalar

Amount of rotation in degrees, specified as a numeric scalar.

DataTypes: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical

method — Interpolation method
'nearest' (default) | 'bilinear' | 'bicubic'

Interpolation method, specified as one of the following values:

Value	Description
'nearest'	Nearest-neighbor interpolation; the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered.
'bilinear'	Bilinear interpolation; the output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood
'bicubic'	Bicubic interpolation; the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood Note Bicubic interpolation can produce pixel values outside the original range.

Data Types: char

bbox — Bounding box defining size of output image
'loose' (default) | 'crop'

Bounding box that defines the size of output image, specified as either of the following values:

Data Types: char

gpuarrayA — Image to be rotated
gpuArray

Image to be rotated, specified as a gpuArray.

Output Arguments

B — Rotated image
real, nonsparse, numeric, or logical array

Rotated image, returned as a real, nonsparse, numeric, or logical array.

gpuarrayB — Rotated image

gpuArray

Rotated image, returned as a gpuArray

Assignment No.3

Title : To enhance contrast using Histogram Equalization

Aim :To implement enhance contrast using MATLAB

Theory:

Syntax

```
J = histeq(I,hgram)

J = histeq(I,n)

[J,T] = histeq(I)

[gpuarrayJ,gpuarrayT] = histeq(gpuarrayI,___)

newmap = histeq(X,map)

newmap = histeq(X,map,hgram)

[newmap,T] = histeq(X,___)
```

Description

J = histeq(I,hgram) transforms the intensity image I so that the histogram of the output intensity image J with length(hgram) bins approximately matches the target histogram hgram.

J = histeq(I,n) transforms the intensity image I, returning in J an intensity image with n discrete gray levels. A roughly equal number of pixels is mapped to each of the n levels in J, so that the histogram of J is approximately flat. The histogram of J is flatter when n is much smaller than the number of discrete levels in I.

[J,T] = histeq(I) returns the grayscale transformation T that maps gray levels in the image I to gray levels in J.

[gpuarrayJ,gpuarrayT] = histeq(gpuarrayI,___) performs the histogram equalization on a GPU. The input image and the output image are of type gpuArray. This syntax requires the Parallel Computing Toolbox™.

newmap = histeq(X,map) transforms the values in the colormap so that the histogram of the gray component of the indexed image X is approximately flat. It returns the transformed colormap in newmap.

newmap = histeq(X,map,hgram) transforms the colormap associated with the indexed image X so that the histogram of the gray component of the indexed image (X,newmap) approximately matches the target histogram hgram. The histeq function returns the transformed colormap in newmap. length(hgram) must be the same as size(map,1).

[newmap,T] = histeq(X,___) returns the grayscale transformation T that maps the gray component of map to the gray component of newmap.

Examples

Enhance Contrast Using Histogram Equalization

Try this Example

Read an image into the workspace.

```
I = imread('tire.tif');
```

Enhance the contrast of an intensity image using histogram equalization.

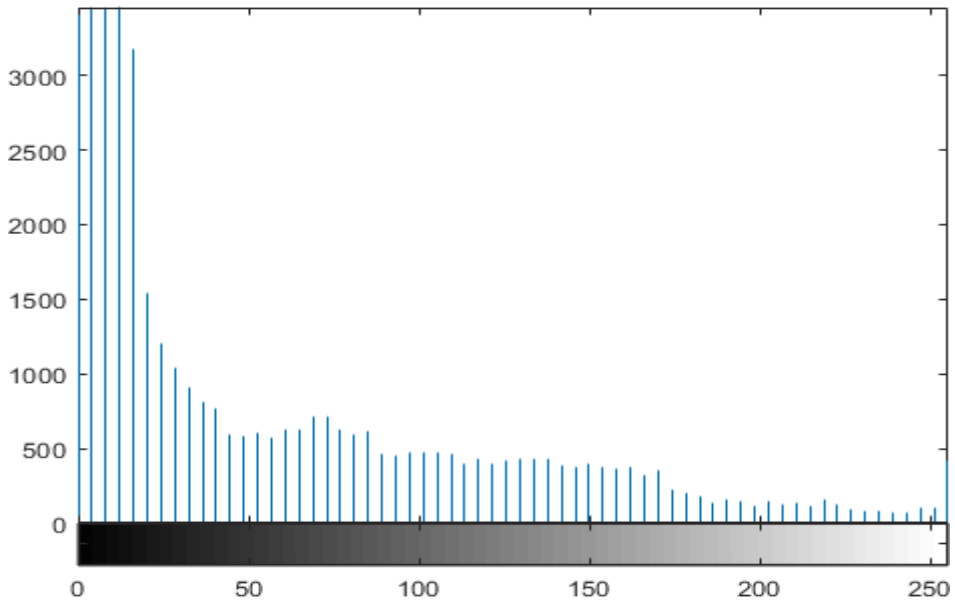
```
J = histeq(I);
```

Display the original image and the adjusted image.

```
imshowpair(I,J,'montage')
axis off
```

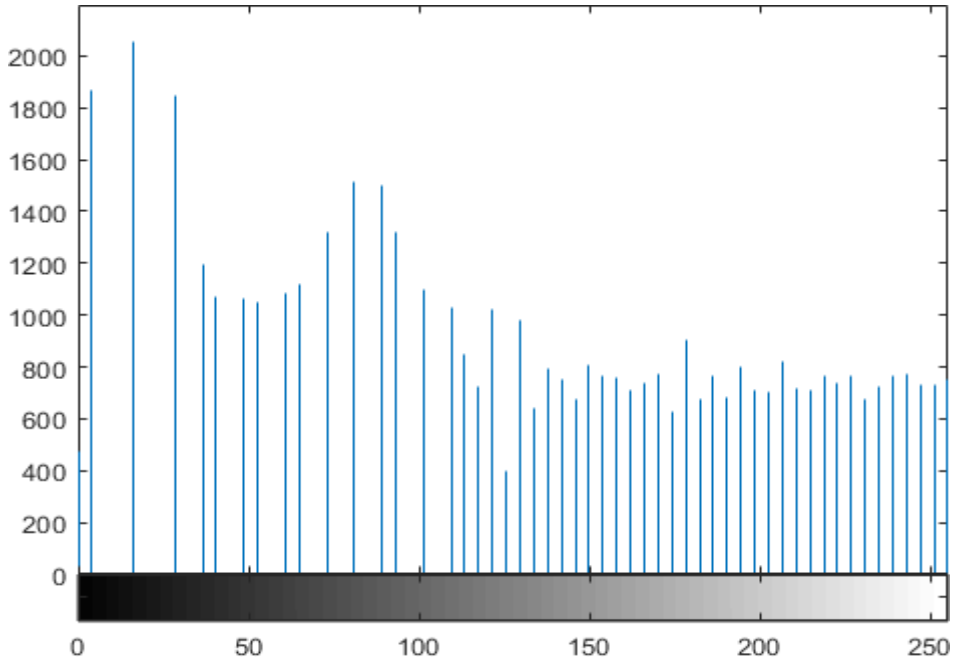
Display a histogram of the original image.

```
figure
imhist(I,64)
```



Display a histogram of the processed image.

```
figure
imhist(J,64)
```



Enhance Contrast of Volumetric Image Using Histogram Equalization

Try this Example

Load a 3-D dataset.

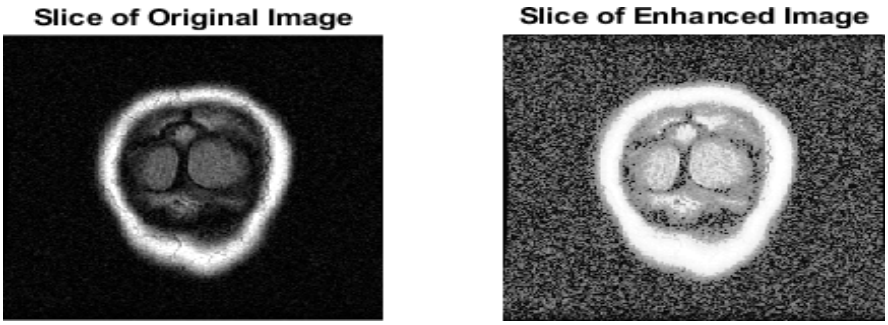
```
load mristack
```

Perform histogram equalization.

```
enhanced = histeq(mristack);
```

Display the first slice of data for the original image and the contrast-enhanced image.

```
figure
subplot(1,2,1)
imshow(mristack(:,:,1))
title('Slice of Original Image')
subplot(1,2,2)
imshow(enhanced(:,:,1))
title('Slice of Enhanced Image')
```



Enhance Contrast Using Histogram Equalization on a GPU

This example performs the same histogram equalization on the GPU.

```
I = gpuArray(imread('tire.tif'));
J = histeq(I);
figure
imshow(I)
figure
imshow(J)
```

Input Arguments

I — Input intensity image

numeric array

Input intensity image, specified as a numeric array. I can be 2-D, 3-D, or *N*-D.

Data Types: single | double | int16 | uint8 | uint16

hgram — Target histogram

numeric vector

Target histogram, specified as a numeric vector. hgram has equally spaced bins with intensity values in the appropriate range: [0, 1] for images of class double or single, [0,

255] for images of class uint8, [0, 65535] for images of class uint16, and [-32768, 32767] for images of class int16. histeq automatically scales hgram so that sum(hgram)=numel(I). The histogram of J will better match hgram when length(hgram) is much smaller than the number of discrete levels in I.

Data Types: single | double

n — Number of discrete gray levels

64 (default) | scalar

Number of discrete gray levels, specified as a scalar.

Data Types: single | double

gpuarrayI — Input image when run on a GPU

gpuArray

Input image when run on a GPU, specified as a gpuArray.

X — Indexed Image

array of real numeric values

Indexed image, specified as an array of real numeric values. The values in X are an index into map. X can be 2-D, 3-D, or *N*-D.

Data Types: single | double | uint8 | uint16

map — Colormap

n-by-3 array

Colormap, specified as an *n*-by-3 array. Each row specifies an RGB color value.

Data Types: double

Output Arguments

J — Output intensity image

numeric array

Output intensity image, returned as a numeric array of the same class as the input image I. J also has the same dimensions as I.

T — Grayscale transformation

numeric vector

Grayscale transformation, returned as a numeric vector. The transformation T maps gray levels in the image I to gray levels in J.

Data Types: double

gpuarrayJ — Output image when run on a GPU

gpuArray

Output image when run on a GPU, returned as a gpuArray.

gpuarrayT — Grayscale transformation when run on a GPU

gpuArray

Grayscale transformation when run on a GPU, returned as a gpuArray.

newmap — Colormap

n-by-3 array

Transformed colormap, specified as an *n*-by-3 array. Each row specifies an RGB color value.

Data Types: double

Algorithms

When you supply a desired histogram hgram, histeq chooses the grayscale transformation *T* to minimize

$$\downarrow c_1(T(k)) - c_0(k) \downarrow,$$

where c_0 is the cumulative histogram of A, c_1 is the cumulative sum of hgram for all intensities *k*. This minimization is subject to the constraints that *T* must be monotonic and $c_1(T(a))$cannot overshoot $c_0(a)$ by more than half the distance between the histogram counts at *a*. histeq uses the transformation $b = T(a)$ to map the gray levels in X (or the colormap) to their new values.

If you do not specify hgram, histeq creates a flat hgram,

```
hgram = ones(1,n)*prod(size(A))/n;
```

and then applies the previous algorithm.

Assignment No.4

Title : Write a program for image enhancement.

Aim: To implement image enhancement using MATLAB

Theory:

Adjusting Intensity Values to a Specified Range

You can adjust the intensity values in an image using the imadjust function, where you specify the range of intensity values in the output image.

For example, this code increases the contrast in a low-contrast grayscale image by remapping the data values to fill the entire intensity range [0, 255]. I = imread('pout.tif'); J = imadjust(I); imshow(J) figure, imhist(J,64)

This figure displays the adjusted image and its histogram. Notice the increased contrast in the image, and that the histogram now fills the entire range. Specifying the Adjustment Limits

You can optionally specify the range of the input values and the output values using imadjust. You specify these ranges in two vectors that you pass to imadjust as arguments. The first vector specifies the low- and high-intensity values that you want to map. The second vector specifies the scale over which you want to map them.

For example, you can decrease the contrast of an image by narrowing the range of the data. In the example below, the man's coat is too dark to reveal any detail. imadjust maps the range [0,51] in the uint8 input image to [128,255] in the output image. This brightens the image considerably, and also widens the dynamic range of the dark portions of the original image, making it much easier to see the details in the coat. Note, however, that because all values above 51 in the original image are mapped to 255 (white) in the adjusted image, the adjusted image appears washed out. I = imread('cameraman.tif'); J = imadjust(I,[0 0.2],[0.5 1]); imshow(I) figure, imshow(J)

Setting the Adjustment Limits Automatically

To use imadjust, you must typically perform two steps: View the histogram of the image to determine the intensity value limits. Specify these limits as a fraction between 0.0 and 1.0 so that you can pass them to imadjust in the [low_in high_in] vector.

For a more convenient way to specify these limits, use the stretchlim function. (The imadjust function uses stretchlim for its simplest syntax, imadjust(I).)

This function calculates the histogram of the image and determines the adjustment limits automatically. The stretchlim function returns these values as fractions in a vector that you can pass as the [low_in high_in] argument to imadjust; for example: I = imread('rice.png'); J = imadjust(I,stretchlim(I),[0 1]);

By default, stretchlim uses the intensity values that represent the bottom 1% (0.01) and the top 1% (0.99) of the range as the adjustment limits. By trimming the extremes at both ends of the intensity range, stretchlim makes more room in the adjusted dynamic range for the remaining intensities. But you can specify other range limits as an argument to stretchlim. See the stretchlim reference page for **more** information.

Gamma Correction

imadjust maps low to bottom, and high to top. By default, the values between low and high are mapped linearly to values between bottom and top. For example, the value halfway between low and high corresponds to the value halfway between bottom and top.

imadjust can accept an additional argument that specifies the gamma correction factor. Depending on the value of gamma, the mapping between values in the input and output images might be nonlinear. For example, the value halfway between low and high might map to a value either greater than or less than the value halfway between bottom and top.

Gamma can be any value between 0 and infinity. If gamma is 1 (the default), the mapping is linear. If gamma is less than 1, the mapping is weighted toward higher (brighter) output values. If gamma is greater than 1, the mapping is weighted toward lower (darker) output values.

The figure below illustrates this relationship. The three transformation curves show how values are mapped when gamma is less than, equal to, and greater than 1. (In each graph, the x-axis represents the intensity values in the input image, and the y-axis represents the intensity values in the output image.)

The example below illustrates gamma correction. Notice that in the call to imadjust, the data ranges of the input and output images are specified as empty matrices. When you specify an empty matrix, imadjust uses the default range of [0,1]. In the example, both ranges are left empty; this means that gamma correction is applied without any other adjustment of the data.

```
[X,map] = imread('forest.tif')

 I = ind2gray(X,map);

J = imadjust(I,[],[],0.5);

 imshow(I)

 figure, imshow(J)
```

Assignment No.5

Title : Write a program for image compression

Aim : To implement compression technique using MATLAB

Theory:

Aa standard matlab waveletcodepackage, WAVELAB802,toperformthe transforms. The wavelets wechoseto use werethe Deslauriers wavelets ofpolynomialsize 3.

Thecompressionscheme we used wasto setathreshold value thatwassome fraction of the norm of the entire wavelettransformmatrix. If the magnitude ofawaveletintherepresentation wasnot largerthan this value, itwas notincluded inthe compression. We thenrebuiltanimage which (to some degree thatdependedon howmanybaseswe included)resembed the original image by runningtheinversetransform.

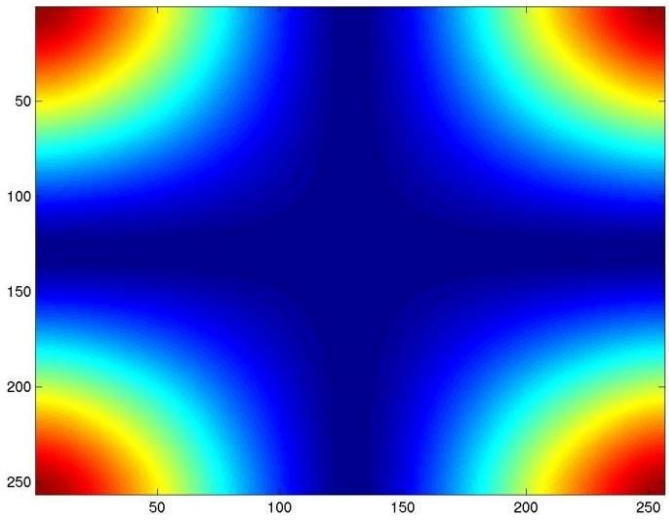
ORIGINALIMAG
E

Thefabulous Len

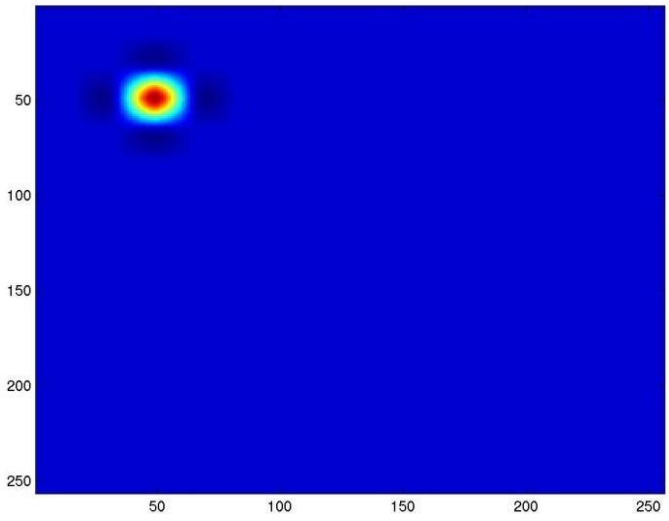
BASIS VECTORS

Thesearesome basis vectors obtained byrunningthe inversewavelettransformon a matrix with a singlenonzero value in it.

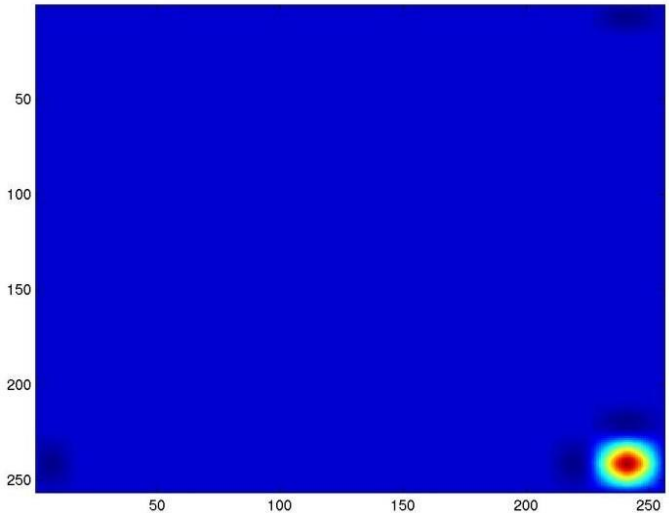
Deslaurier(1,1)



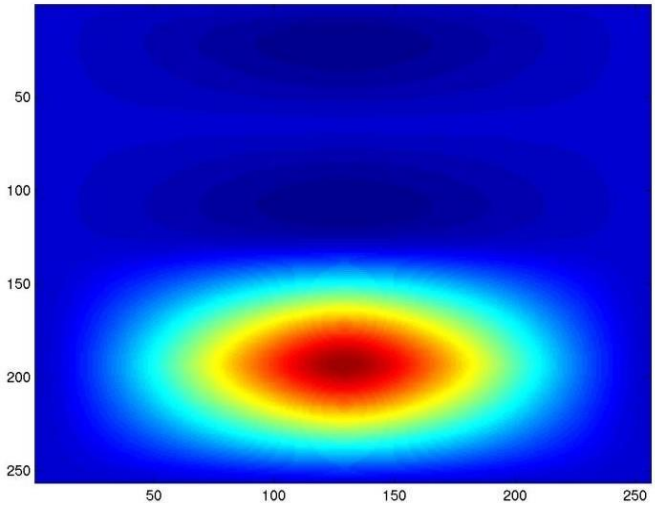
Deslaurier(10,10)



Deslaurier(16,16)

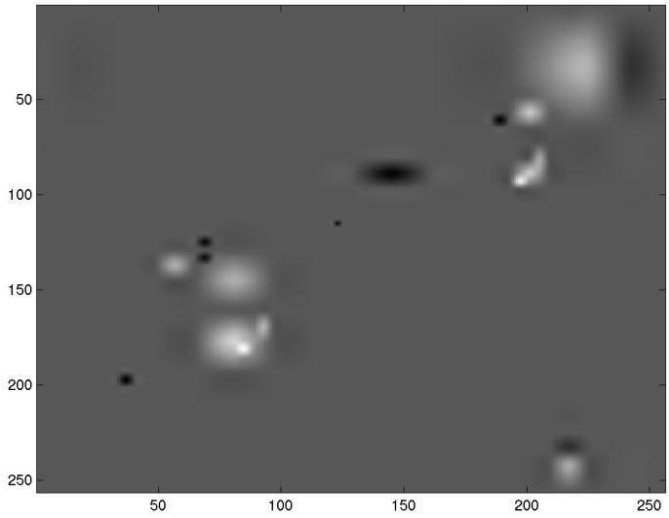


Deslaurier(4,2)

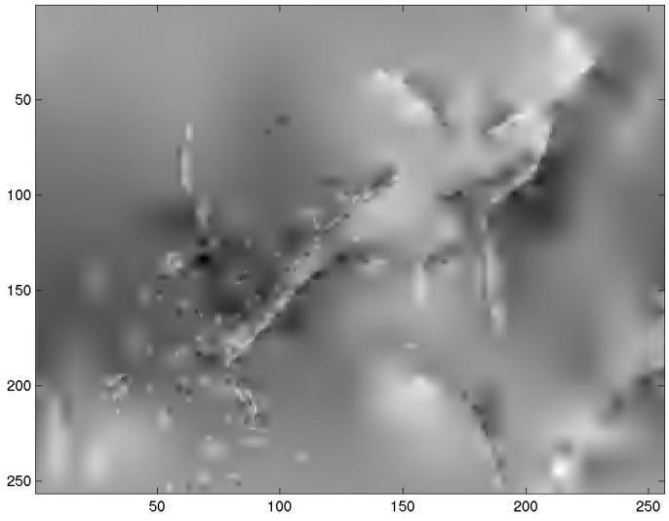


COMPRESSEDIMAGES

Threshold= .5, Basesincluded = 19, Compression ratio = 3400:1



Threshold= 1, Basesincluded = 470, Compression ratio = 140 :1



Threshold= 2, Basesincluded = 2383, Compression ratio = 27:1

Threshold= 4, Basesincluded = 6160, Compression ratio = 10:1

Threshold= 8, Basesincluded = 12378, Compressionratio =5 :1

MATLAB CODE

```
load/home/nirav/elec301/lena256.m
at; imagesc(lena256);
colormap(gray(256));


[qmf, dqmf]= MakeBSFilter('Deslauriers', 3);


%TheMakeBSFilterfunction createsbiorthonormalfilterpairs.The filter

%pairs thatwe're makingis an Interpolating(Deslauriers-Dubuc)filter

%ofpolynomialdegree 3


wc = FWT2_PB(lena256, 1, qmf, dqmf);


%wc correspond tothewaveletcoefficientsofthesample image

%FWT2_PBisa functionthattakesa 2dimensionalwavelettransform

%We specifytheimagematrix,thelevelofcoarseness (1), the quadrature

%mirrorfilter(qmf), and the dualquadrature mirrorfilter(dqmf)


%we take a tolerance which issome fraction

%ofthenormofthe sample image


nl= norm(lena256)/(4 * norm(size(lena256)));


%if the valueofthe waveletcoefficientmatrix ata particular

% rowand column islessthan the tolerance, we 'throw'itout

%andincrementthe zero count.


zerocount=
0; fori=
1:256
forj=1:256
```

```
    if(abs(wc(i,j))<nl)
wc(i,j)=0;
        zerocount= zerocount+ 1;
    end
 end
 end

 x = IWT2_PB(wc,1, qmf, dqmf);
 imagesc(x);

 %hereis some sample code to view howthesedeslauriers waveletslook
 [qmf, dqmf]= MakeBSFilter('Deslauriers', 3);
 fori=  1:256
  forj=1:256
  wc(i,j)= 0;
  end
 end

 % thisisthe
 Deslauriers(4,2)matrix wc(4, 2)=
 1000;
 x = IWT2_PB(wc,1, qmf, dqmf);
imagesc(x);
```

Assignment No.6

Title: Write a program for Edge detection

Aim: To implement edge detection using MATLAB

Theory:

Edge Detection

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 several derivative estimators, each of which implements one of these definitions. 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.

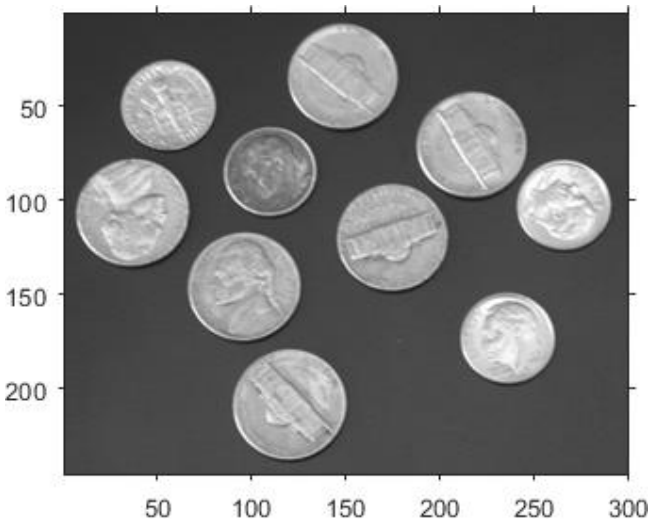
The most powerful edge-detection method that edge provides is the Canny method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be affected by noise, and more likely to detect true weak edges.

Detect Edges in Images

This example shows how to detect edges in an image using both the Canny edge detector and the Sobel edge detector.

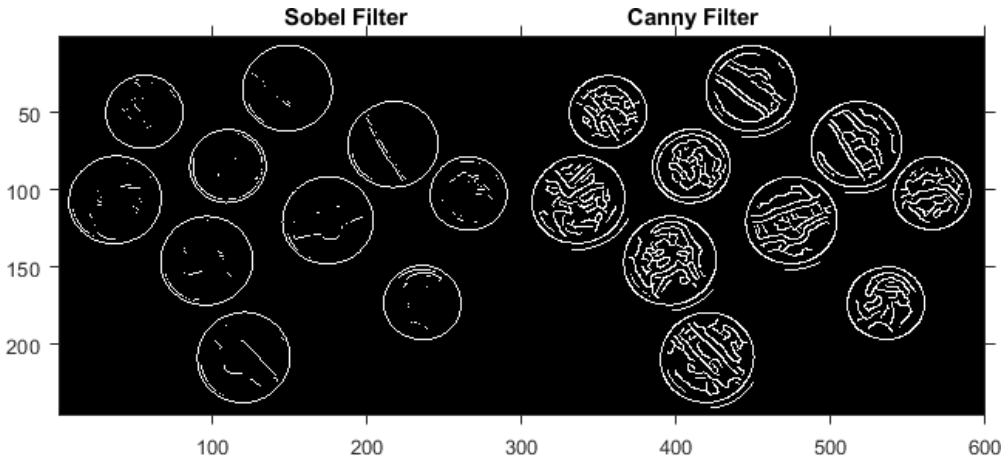
Read image and display it.

```
I = imread('coins.png');
imshow(I)
```



Apply both the Sobel and Canny edge detectors to the image and display them for comparison.

```
BW1 = edge(I,'sobel');
BW2 = edge(I,'canny');
figure;
imshowpair(BW1,BW2,'montage')
title('Sobel Filter Canny Filter');
```



Programfor Edge detection

```
%demonstrateedgedetection

%numbersofcolors sncols=128; ncols=32;

%getimagefromMATLABim
age load('trees');

%showoriginalimage
figure(1);
showgimg(real(X),sncol
s); drawnow;

%constructconvolutionfunctions

[m,n]=size(X);

gs=[1-1];ge=[];

hs=[1-1];he=[];

g=[gs,zeros(1,m-length(gs)-length(ge)),ge];

h=[hs,zeros(1,n-length(hs)-length(he)),he];

%constructconvolutionmatricesassparsematrices

Y=spcnvmat(
g);
Z=spcnvmat(
h);
Wg=Y*X;

Wh=X*Z';

%showtransformedimag
es figure(2);
showgimg(Wg,ncols);
drawnow;

figure(3)
showgimg(Wh,n
cols); drawnow;

figure(4) showgimg(abs(Wg)+abs(Wh),ncols); drawnow;
```

Assignment No.7

Title: Write a program for image segmentation

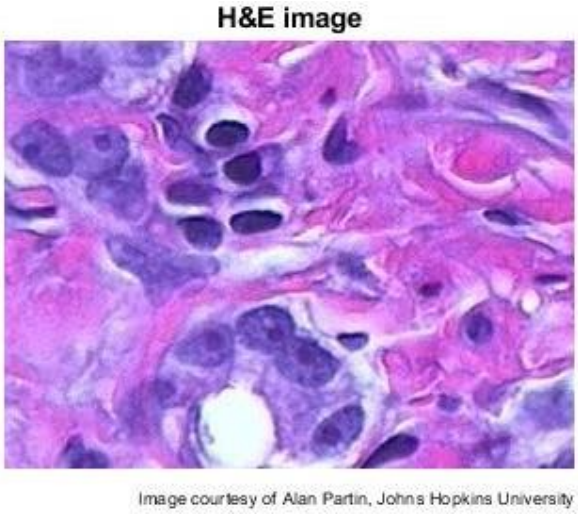
Aim : To implement image segmentation using MATLAB

Theory

Step 1: Read Image

Read in hestain.png, which is an image of tissue stained with hemotoxylin and eosin (H&E). This staining method helps pathologists distinguish different tissue types.

```
he = imread('hestain.png');
imshow(he), title('H&E image');
text(size(he,2),size(he,1)+15,...
    'Image courtesy of Alan Partin, Johns Hopkins University', ...
    'FontSize',7,'HorizontalAlignment','right');
```



Step 2: Convert Image from RGB Color Space to L*a*b* Color Space

How many colors do you see in the image if you ignore variations in brightness? There are three colors: white, blue, and pink. Notice how easily you can visually distinguish these colors from one another. The L*a*b* color space (also known as CIELAB or CIE L*a*b*) enables you to quantify these visual differences.

The L*a*b* color space is derived from the CIE XYZ tristimulus values. The L*a*b* space consists of a luminosity layer 'L*', chromaticity-layer 'a*' indicating where color falls along the red-green axis, and chromaticity-layer 'b*' indicating where the color falls along the blue-yellow axis. All of the color information is in the 'a*' and 'b*' layers. You can measure the difference between two colors using the Euclidean distance metric.

Convert the image to L*a*b* color space using makecform and applycform.

```
cform = makecform('srgb2lab');
lab_he = applycform(he,cform);
```

Step 3: Classify the Colors in 'a*b*' Space Using K-Means Clustering

Clustering is a way to separate groups of objects. K-means clustering treats each object as having a location in space. It finds partitions such that objects within each cluster are as close to each other as possible, and as far from objects in other clusters as possible. K-means clustering requires that you specify the number of clusters to be partitioned and a distance metric to quantify how close two objects are to each other.

Since the color information exists in the 'a*b*' space, your objects are pixels with 'a*' and 'b*' values. Use kmeans to cluster the objects into three clusters using the Euclidean distance metric.

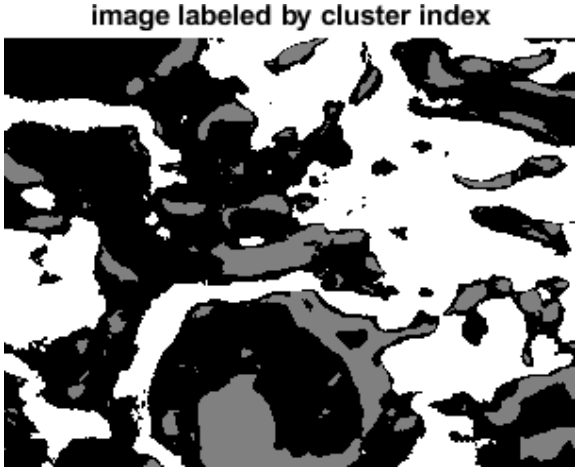
```
ab = double(lab_he(:,:,2:3));
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,2);

nColors = 3;
% repeat the clustering 3 times to avoid local minima
[cluster_idx, cluster_center] = kmeans(ab,nColors,'distance','sqEuclidean', ...
                                       'Replicates',3);
```

Step 4: Label Every Pixel in the Image Using the Results from KMEANS

For every object in your input, kmeans returns an index corresponding to a cluster. The cluster_center output from kmeans will be used later in the example. Label every pixel in the image with its cluster_index.

```
pixel_labels = reshape(cluster_idx,nrows,ncols);
imshow(pixel_labels,[]), title('image labeled by cluster index');
```



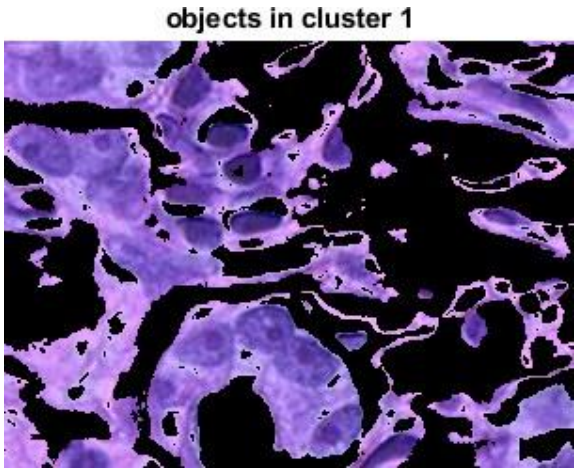
Step 5: Create Images that Segment the H&E Image by Color.

Using pixel_labels, you can separate objects in hestain.png by color, which will result in three images.

```
segmented_images = cell(1,3);
rgb_label = repmat(pixel_labels,[1 1 3]);
```

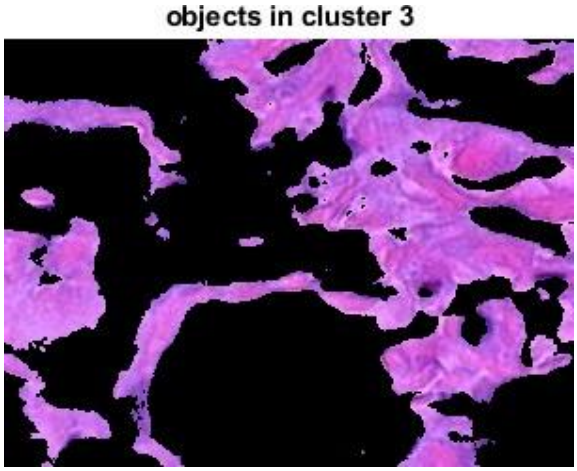
```
for k = 1:nColors
   color = he;
   color(rgb_label ~= k) = 0;
   segmented_images{k} = color;
end
```

```
imshow(segmented_images{1}), title('objects in cluster 1');
```




```
imshow(segmented_images{2}), title('objects in cluster 2');
```


```
imshow(segmented_images{3}), title('objects in cluster 3');
```



Step 6: Segment the Nuclei into a Separate Image

Notice that there are dark and light blue objects in one of the clusters. You can separate dark blue from light blue using the 'L*' layer in the L*a*b* color space. The cell nuclei are dark blue.

Recall that the 'L*' layer contains the brightness values of each color. Find the cluster that contains the blue objects. Extract the brightness values of the pixels in this cluster and threshold them with a global threshold using imbinarize.

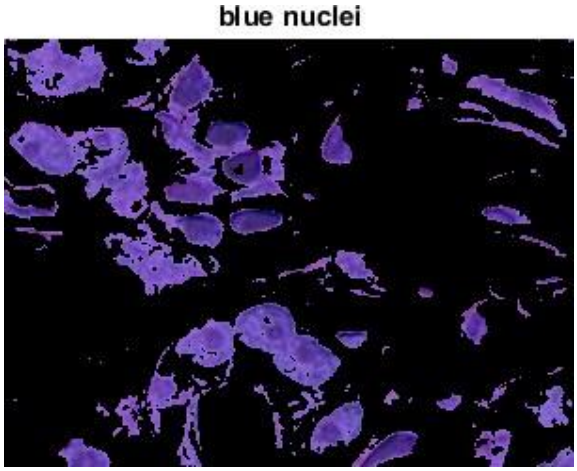
You must programmatically determine the index of the cluster containing the blue objects because kmeans will not return the same cluster_idx value every time. You can do this using the cluster_center value, which contains the mean 'a*' and 'b*' value for each cluster. The blue cluster has the smallest cluster_center value (determined experimentally).

```
mean_cluster_value = mean(cluster_center,2);
[tmp, idx] = sort(mean_cluster_value);
blue_cluster_num = idx(1);

L = lab_he(:,:,1);
blue_idx = find(pixel_labels == blue_cluster_num);
L_blue = L(blue_idx);
is_light_blue = imbinarize(L_blue);
```

Use the mask is_light_blue to label which pixels belong to the blue nuclei. Then display the blue nuclei in a separate image.

```
nuclei_labels = repmat(uint8(0),[nrows ncols]);
nuclei_labels(blue_idx(is_light_blue==false)) = 1;
nuclei_labels = repmat(nuclei_labels,[1 1 3]);
blue_nuclei = he;
blue_nuclei(nuclei_labels ~= 1) = 0;
imshow(blue_nuclei), title('blue nuclei');
```



Demonstration ofglobaland localthresholdingforsegmentation

```
% threshdemo.m

%Demonstration ofglobaland local thresholdoperations ofan image

clearall

[tmp,idx]=imread('lena.bmp');

a=ind2gray(tmp,idx);%grayscaleimage of lena, value between
0and 1 cleartmp idx

figure(1),clf,colormap('gray'),imshow(a),title('originalLena image')

[m,n]=size(a);%size ofimage a
b=reshape(a,m*n,1);%into a column
vector
figure(2),hist(b,50),title('histogramof
image')

%firstdo globalthresholding

mu=rand(2,1);%valuebetwen 0 and 1, twoclusters only

[W,iter,Sw,Sb,Cova]=kmeansf(b,mu);%W is the mean,

%Cova isthecovariancematrices

%member:membership ofeachX: Kby1 vectorofelements 1 toc

[d,member]=kmeantest(b,sort(W));
```

```
c=reshape(member-1,m,n);       cleard
memberb
figure(3),clf,colormap('gray'),imsho
w(c) title('global threshold')


%nextdolocalthreshold,partitiontheimage into 64x 64 blocks

%anddo    threshold    withineach
block c=zeros(512,512);trials=0;

fori=1:8,
  forj=1:8,

    trials=trials+1;

    disp([int2str(trials)'of64 iterations ...']);

    tmp=a(64*(i-1)+1:64*i,64*(j-1)+1:64*j);

    tmpi=reshape(tmp,64*64,1);

    mu=sort(rand(2,1));% value betwen 0 and 1,twoclusters only

    [W,iter,Sw,Sb,Cova]=kmeansf(tmpi,mu);%W is the mean,

    %Cova is the covariancematrices

    %member:membership ofeachX:Kby1 vectorofelements 1to c

    [d,member]=kmeantest(tmpi,sort(W));

    c(64*(i-1)+1:64*i,64*(j-1)+1:64*j)=reshape(member,64,64);

 end
end

figure(4),clf,colormap('gray'),imshow(c
-1), title('localthreshold, 64x64block');
```

Assignment No.8

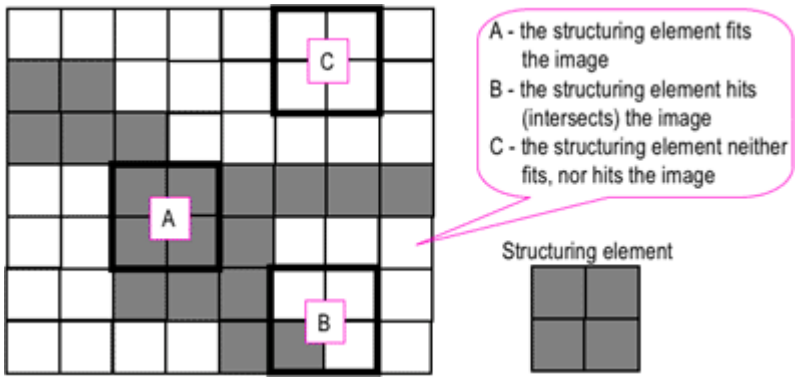
Title: Write a program for image morphology

Aim: To Implement Morphological Image Processing

Theory:

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. According to Wikipedia, morphological operations rely only on the relative ordering of pixel values, not on their numerical values, and therefore are especially suited to the processing of binary images. Morphological operations can also be applied to greyscale images such that their light transfer functions are unknown and therefore their absolute pixel values are of no or minor interest.

Morphological techniques probe an image with a small shape or template called a **structuring element**. The structuring element is positioned at all possible locations in the image and it is compared with the corresponding neighbourhood of pixels. Some operations test whether the element "fits" within the neighbourhood, while others test whether it "hits" or intersects the neighbourhood:

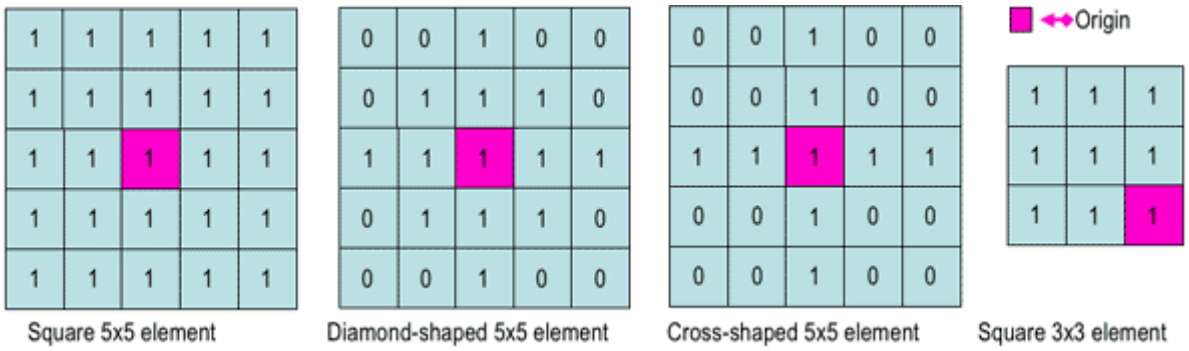


Probing of an image with a structuring element
(white and grey pixels have zero and non-zero values, respectively).

A morphological operation on a binary image creates a new binary image in which the pixel has a non-zero value only if the test is successful at that location in the input image.

The **structuring element** is a small binary image, i.e. a small matrix of pixels, each with a value of zero or one:

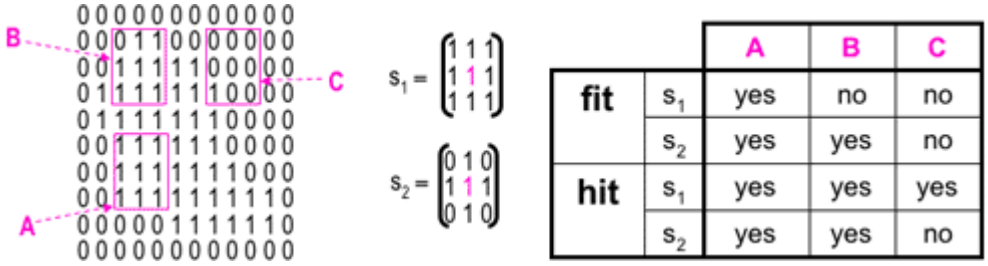
- The matrix dimensions specify the *size* of the structuring element.
- The pattern of ones and zeros specifies the *shape* of the structuring element.
- An *origin* of the structuring element is usually one of its pixels, although generally the origin can be outside the structuring element.



Examples of simple structuring elements.

A common practice is to have odd dimensions of the structuring matrix and the origin defined as the centre of the matrix. Stucturing elements play in moprphological image processing the same role as convolution kernels in linear image filtering.

When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighbourhood under the structuring element. The structuring element is said to **fit**the image if, for each of its pixels set to 1, the corresponding image pixel is also 1. Similarly, a structuring element is said to **hit**, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.



		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Fitting and hitting of a binary image with structuring elements s_1 and s_2.

Zero-valued pixels of the structuring element are ignored, i.e. indicate points where the corresponding image value is irrelevant.

Fundamental operations

More formal descriptions and examples of how basic morphological operations work are given in the Hypermedia Image Processing Reference (HIPR) developed by Dr. R. Fisher et al. at the Department of Artificial Intelligence in the University of Edinburgh, Scotland, UK.

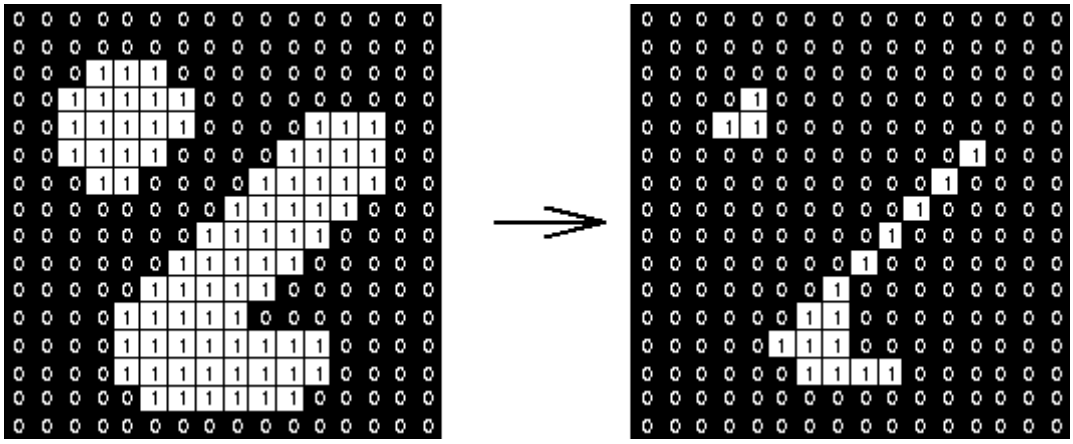
Erosion and dilation

The **erosion** of a binary image f by a structuring element s (denoted $f \ominus s$) produces a new binary image $g = f \ominus s$ with ones in all locations (x,y) of a structuring element's origin at which that structuring element s fits the input image f, i.e. $g(x,y) = 1$ is s fits f and 0 otherwise, repeating for all pixel coordinates (x,y).

Greyscale image | Binary image by thresholding | Erosion: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

Erosion with small (e.g. 2×2 - 5×5) square structuring elements shrinks an image by stripping away a layer of pixels from both the inner and outer boundaries of regions. The holes and gaps between different regions become larger, and small details are eliminated:



Erosion: a 3×3 square structuring element
(www.cs.princeton.edu/~pshilane/class/mosaic/).

Larger structuring elements have a more pronounced effect, the result of erosion with a large

structuring element being similar to the result obtained by iterated erosion using a smaller structuring element of the same shape. If s_1 and s_2 are a pair of structuring elements identical in shape, with s_2 twice the size of s_1, then

$f \ominus s_2 \approx (f \ominus s_1) \ominus s_1$.

Erosion removes small-scale details from a binary image but simultaneously reduces the size of regions of interest, too. By subtracting the eroded image from the original image, boundaries of each region can be found: $b = f - (f \ominus s)$ where f is an image of the regions, s is a 3×3 structuring element, and b is an image of the region boundaries.

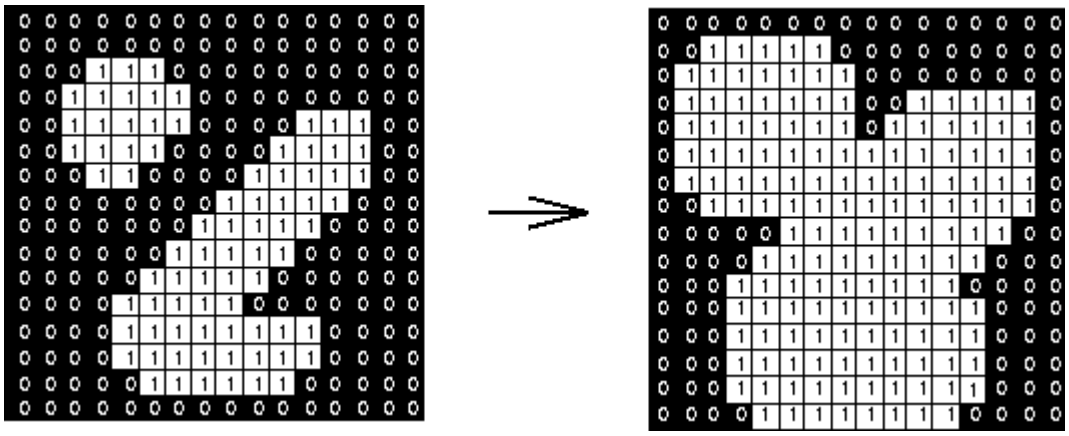
The **dilation** of an image f by a structuring element s (denoted $f \oplus s$) produces a new binary image $g = f \oplus s$ with ones in all locations (x,y) of a structuring element's orogin at which that structuring element $s$hits the the input image f, i.e. $g(x,y) = 1$ if s hits f and 0 otherwise, repeating for all pixel coordinates (x,y). Dilation has the opposite effect to erosion -- it adds a layer of pixels to both the inner and outer boundaries of regions.

Binary image

Dilation: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

The holes enclosed by a single region and gaps between different regions become smaller, and small intrusions into boundaries of a region are filled in:



Dilation: a 3×3 square structuring element
(www.cs.princeton.edu/~pshilane/class/mosaic/).

Results of dilation or erosion are influenced both by the size and shape of a structuring element. Dilation and erosion are *dual* operations in that they have opposite effects. Let f^c denote the complement of an image *f*, i.e., the image produced by replacing 1 with 0 and vice versa. Formally, the duality is written as

$$f \oplus s = f^c \ominus s_{rot}$$

where s_{rot} is the structuring element *s* rotated by 180o. If a structuring element is symmetrical with respect to rotation, then s_{rot} does not differ from *s*. If a binary image is considered to be a collection of connected regions of pixels set to 1 on a background of pixels set to 0, then erosion is the fitting of a structuring element to these regions and dilation is the fitting of a structuring element (rotated if necessary) into the background, followed by inversion of the result.

Compound operations

Many morphological operations are represented as combinations of erosion, dilation, and simple set-theoretic operations such as the **complement** of a binary image:

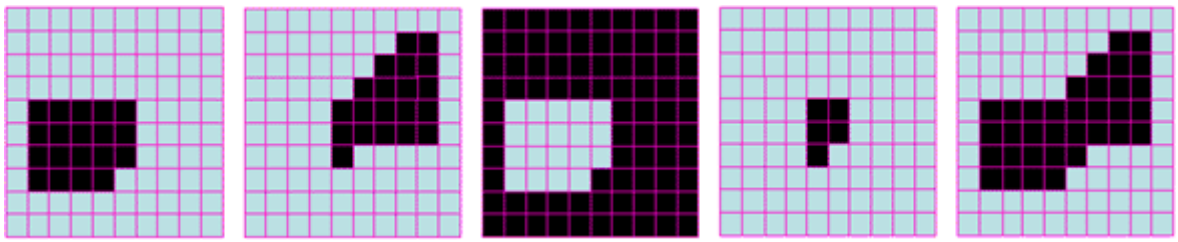
$f^c(x,y) = 1$ if $f(x,y) = 0$, and $f^c(x,y) = 0$ if $f(x,y) = 1$,

the **intersection** $h = f \cap g$ of two binary images *f* and *g*:

$h(x,y) = 1$ if $f(x,y) = 1$ and $g(x,y) = 1$, and $h(x,y) = 0$ otherwise,

and the **union** $h = f \cup g$ of two binary images f and g:

$h(x,y) = 1$ if $f(x,y) = 1$ or $g(x,y) = 1$, and $h(x,y) = 0$ otherwise:



Set operations on binary images: from left to right: a binary image f, a binary image g, the complement f^c of f, the intersection $f \cap g$, and the union $f \cup g$.

The **opening** of an image f by a structuring element s (denoted by $f \circ s$) is an erosion followed by a dilation:

$$f \circ s = (f \ominus s) \oplus s$$

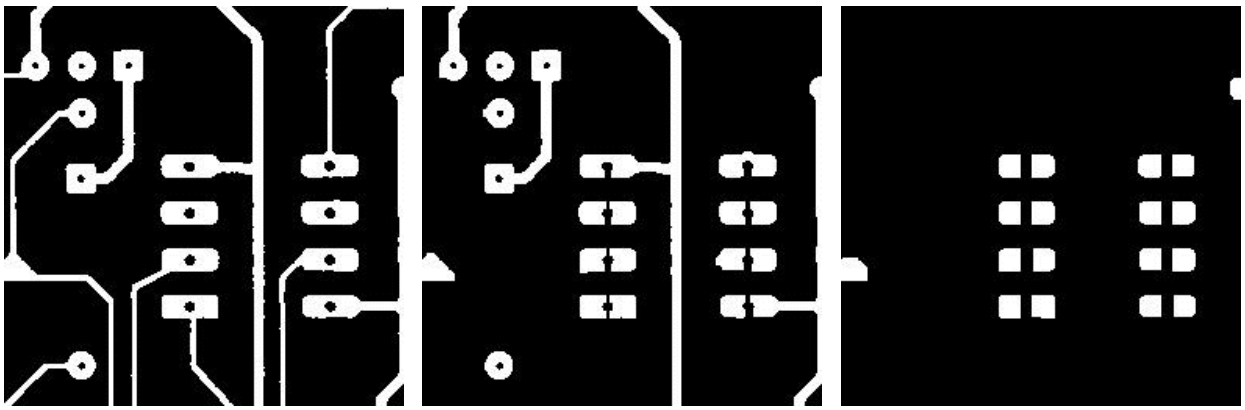
Binary image

Opening: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

Opening is so called because it can open up a gap between objects connected by a thin bridge of

pixels. Any regions that have survived the erosion are restored to their original size by the dilation:



Binary image f $f \circ s$ (5×5 square) $f \circ s$ (9×9 square)

Results of opening with a square structuring element (www.mmorph.com/html/morph/mmopen.html/).

Opening is an **idempotent** operation: once an image has been opened, subsequent openings with the same structuring element have no further effect on that image:

$$(f \circ s) \circ s) = f \circ s.$$

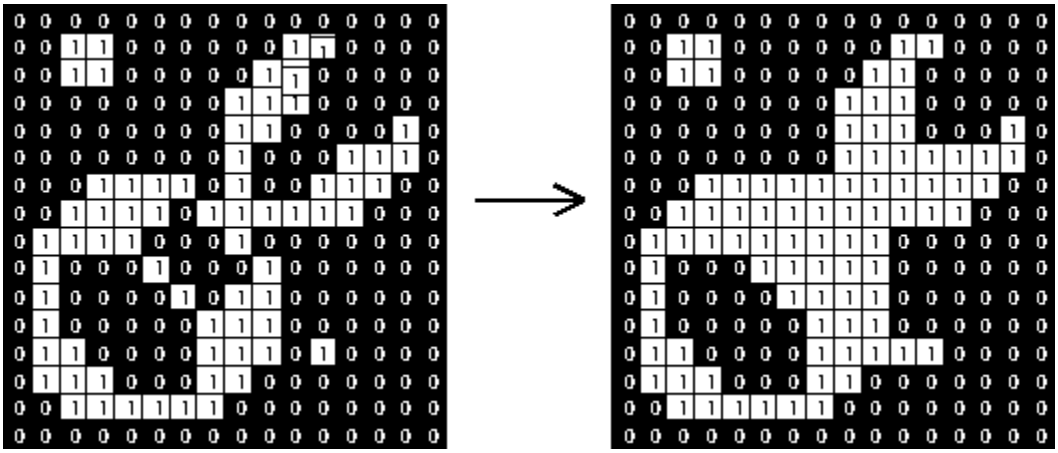
The **closing** of an image f by a structuring element s (denoted by $f \bullet s$) is a dilation followed by an erosion:

$$f \bullet s = (f \oplus s_{rot}) \ominus s_{rot}$$

Binary image Closing: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

In this case, the dilation and erosion should be performed with a rotated by 180° structuring element. Typically, the latter is symmetrical, so that the rotated and initial versions of it do not differ.



Closing with a 3×3 square structuring element
(www.cs.princeton.edu/~pshilane/class/mosaic/).

Closing is so called because it can fill holes in the regions while keeping the initial region sizes. Like opening, closing is idempotent: $(f \bullet s) \bullet s = f \bullet s$, and it is dual operation of opening (just as opening is the dual operation of closing):

$$f \bullet s = (f^c \circ s)^c; \quad f \circ s = (f^c \bullet s)^c.$$

In other words, closing (opening) of a binary image can be performed by taking the complement of that image, opening (closing) with the structuring element, and taking the complement of the result.

The **hit and miss transform** (see also HIPS2 web page) allows to derive information on how objects in a binary image are related to their surroundings. The operation requires a matched pair of structuring elements, $\{s_1, s_2\}$, that probe the inside and outside, respectively, of objects in the image:

$$f \circledast \{s_1, s_2\} = (f \ominus s_1) \cap (f^c \ominus s_2).$$

Binary image

Hit and miss transform:
an elongated 2×5 structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

A pixel belonging to an object is preserved by the hit and miss transform if and only if s_1 translated to that pixel fits inside the object AND s_2 translated to that pixel fits outside the object. It is assumed that $s_1$and s_2 do not intersect, otherwise it would be impossible for both fits to occur simultaneously.

It is easier to describe it by considering s_1 and s_2 as a single structuring element with 1s for pixels of s_1 and 0s for pixels of s_2; in this case the hit-and-miss transform assigns 1 to an output pixel only if the object (with the value of 1) and background (with the value of 0) pixels in the structuring element exactly match object (1) and background (0) pixels in the input image. Oterwise that pixel is set to the background value (0).

The hit and miss transform can be used for detecting specific shapes (spatial arrangements of object and background pixel values) if the two structuring elements present the desired shape, as well as for thinning or thickening of object linear elements.

Morphological filtering

of a binary image is conducted by considering compound operations like opening and closing as filters. They may act as filters of shape. For example, opening with a disc structuring element smooths corners from the inside, and closing with a disc smooths corners from the outside. But also these operations can filter out from an image any details that are smaller in size than the structuring element, e.g. opening is filtering the binary image at a scale defined by the size of the structuring element. Only those portions of the image that fit the structuring element are passed by the filter; smaller structures are blocked and excluded from the output image. The size of the structuring element is most important to eliminate noisy details but not to damage objects of interest.

Demonstrate boundaryextraction, interiorfilling

```
%demonstrate morphologicalboundaryextraction

%

clearall,closeall

A0=imread('myshap4.bmp');

imshow(A0);%a heartshape
handdrawing title('originalimage');

pause

%A0 contains mostly1s and thedrawingcontains 0s,uint8

A1=1-double(A0);%invertblackand white


B=ones(3);

A2=imopen(imclose(A1,B),B);%fill1 pixelhole andremove sticks

A3=imfill(A2,[100 100]);%fillthe
interior A=double(A2)+ double(A3);
imshow(A),title('afterinteriorfillingusingi
mfill');

pause

Ab=A-double(erode(A,B));


imshow(Ab), title('extracted
boundary'); clearA1 A2
A3;Ac=Ab;
vidx=[[1:20:280]280];

Ac(vidx,:)=1;Ac(:,vidx)=1;imshow(Ac)
```

Assignment No.9

Title: Illustrate and discuss use of various method of pattern recognition.

Aim: To illustrate various methods of pattern recognition

Theory:

Pattern recognition is a branch of machine learning that focuses on the recognition of patterns and regularities in data, although it is in some cases considered to be nearly synonymous with machine learning.[1] Pattern recognition systems are in many cases trained from labeled "training" data (supervised learning), but when no labeled data are available other algorithms can be used to discover previously unknown patterns (unsupervised learning).

The terms pattern recognition, machine learning, data mining and knowledge discovery in databases (KDD) are hard to separate, as they largely overlap in their scope. Machine learning is the common term for supervised learning methods[*dubious – discuss*] and originates from artificial intelligence, whereas KDD and data mining have a larger focus on unsupervised methods and stronger connection to business use. Pattern recognition has its origins in engineering, and the term is popular in the context of computer vision: a leading computer vision conference is named Conference on Computer Vision and Pattern Recognition. In pattern recognition, there may be a higher interest to formalize, explain and visualize the pattern, while machine learning traditionally focuses on maximizing the recognition rates. Yet, all of these domains have evolved substantially from their roots in artificial intelligence, engineering and statistics, and they've become increasingly similar by integrating developments and ideas from each other.

In machine learning, pattern recognition is the assignment of a label to a given input value. In statistics, discriminant analysis was introduced for this same purpose in 1936. An example of pattern recognition is classification, which attempts to assign each input value to one of a given set of *classes* (for example, determine whether a given email is "spam" or "non-spam"). However, pattern recognition is a more general problem that encompasses other types of output as well. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); and parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence.[*citation needed*]

Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to perform "most likely" matching of the inputs, taking into account their statistical variation. This is opposed to *pattern matching* algorithms, which look for exact matches in the input with pre-existing patterns. A common example of a pattern-matching algorithm is regular expression matching, which looks for patterns of a given sort in textual data and is included in the search capabilities of many text editors and word processors. In contrast to pattern recognition, pattern matching is generally not considered a type of machine learning, although pattern-matching algorithms (especially with fairly general, carefully tailored patterns) can sometimes succeed in providing similar-quality output of the sort provided by pattern-recognition algorithms.

k-means clustering

***k*-means clustering** is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the *k*-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with *k*-means because of the *k* in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by *k*-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

Assignment No.10

Title: Write a program for face detection in MATLAB.

Aim: To implement face detection

Theory:

Definition and related algorithms

Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars.

Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will invalidate the matching process.

A reliable face-detection approach based on the genetic algorithm and the eigen-face[3] technique:[4]

Firstly, the possible human eye regions are detected by testing all the valley regions in the gray-level image. Then the genetic algorithm is used to generate all the possible face regions which include the eyebrows, the iris, the nostril and the mouth corners.

Each possible face candidate is normalized to reduce both the lightning effect , which is caused by uneven illumination; and the shirring effect, which is due to head movement. The fitness value of each candidate is measured based on its projection on the eigen-faces. After a number of iterations, all the face candidates with a high fitness value are selected for further verification. At this stage, the face symmetry is measured and the existence of the different facial features is verified for each face candidate.

Code:

```
% usage:

%  I=double(imread('c:\Data\girl1.jpg'));

%  detect_face(I);

% The function will display the bounding box if a face is found.

function []=detect_face(I)

close all;

% No faces at the beginning
```

```
Faces=[];
numFaceFound=0;
I=double(I);
H=size(I,1);
W=size(I,2);
R=I(:,:,1);
G=I(:,:,2);
B=I(:,:,3);
```

LIGHTING COMPENSATION

```
YCbCr=rgb2ycbcr(I);
Y=YCbCr(:,:,1);
%normalize Y
minY=min(min(Y));
maxY=max(max(Y));
Y=255.0*(Y-minY)./(maxY-minY);
YEye=Y;
Yavg=sum(sum(Y))/(W*H);
T=1;
if (Yavg<64)
   T=1.4;
elseif (Yavg>192)
   T=0.6;
end
if (T~=1)
```

```
    RI=R.^T;
    GI=G.^T;
else
    RI=R;
    GI=G;
end
C=zeros(H,W,3);
C(:,:,1)=RI;
C(:,:,2)=GI;
C(:,:,3)=B;
```

EXTRACT SKIN

```
YCbCr=rgb2ycbcr(C);
Cr=YCbCr(:,:,3);
S=zeros(H,W);
[SkinIndexRow,SkinIndexCol] =find(10<Cr & Cr<45);
for i=1:length(SkinIndexRow)
    S(SkinIndexRow(i),SkinIndexCol(i))=1;
end
```

REMOVE NOISE

```
SN=zeros(H,W);
for i=1:H-5
    for j=1:W-5
        localSum=sum(sum(S(i:i+4, j:j+4)));
```

```
            SN(i:i+5, j:j+5)=(localSum>12);
        end
    end
```

FIND SKIN COLOR BLOCKS

```
    L = bwlabel(SN,8);
    BB  = regionprops(L, 'BoundingBox');
    bboxes= cat(1, BB.BoundingBox);
    widths=bboxes(:,3);
    heights=bboxes(:,4);
    hByW=heights./widths;

    lenRegions=size(bboxes,1);
    foundFaces=zeros(1,lenRegions);

    rgb=label2rgb(L);
    figure,imshow(rgb);
    title('face candidates');
```

CHECK FACE CRITERIONS

```
for i=1:lenRegions
```

% 1st criteria: height to width ratio, computed above.

```
  if (hByW(i)>1.75 || hByW(i)<0.75)
    % this cannot be a mouth region. discard
    continue; end
```

```
% implemented by me: Impose a min face dimension constraint
if (heights(i)<20 && widths(i)<20)
    continue;
end
% get current region's bounding box
CurBB=bboxes(i,:);
XStart=CurBB(1);
YStart=CurBB(2);
WCur=CurBB(3);
HCur=CurBB(4);
% crop current region
rangeY=int32(YStart):int32(YStart+HCur-1);
rangeX= int32(XStart):int32(XStart+WCur-1);
RIC=RI(rangeY, rangeX);
GIC=GI(rangeY, rangeX);
BC=B(rangeY, rangeX);
figure, imshow(RIC/255);
title('Possible face R channel');
```

% 2nd criteria: existance & localisation of mouth

```
M=zeros(HCur, WCur);
theta=acos( 0.5.*(2.*RIC-GIC-BC) ./ sqrt( (RIC-GIC).*(RIC-GIC) + (RIC-BC).*(GIC-BC) )
);
theta(isnan(theta))=0;
thetaMean=mean2(theta);
[MouthIndexRow,MouthIndexCol] =find(theta<thetaMean/4);
```

```
for j=1:length(MouthIndexRow)
   M(MouthIndexRow(j),MouthIndexCol(j))=1;
end
% now compute vertical mouth histogram
Hist=zeros(1, HCur);
for j=1:HCur
   Hist(j)=length(find(M(j,:)==1));
end
wMax=find(Hist==max(Hist));
wMax=wMax(1); % just take one of them.
if (wMax < WCur/6)
   %reject due to not existing mouth
   continue;
end
```

% 3rd criteria: existance & localisation of eyes

```
eyeH=HCur-wMax;
eyeW=WCur;
YC=YEye(YStart:YStart+eyeH-1, XStart:XStart+eyeW-1);
E=zeros(eyeH,eyeW);
[EyeIndexRow,EyeIndexCol] =find(65<YC & YC<80);
for j=1:length(EyeIndexRow)
   E(EyeIndexRow(j),EyeIndexCol(j))=1;
end
% check if eyes are acceptable.
EyeExist=find(Hist>0.3*wMax);
```

```
    if (~(length(EyeExist)>0))
        continue;
    end
    foundFaces(i)=1;
    numFaceFound=numFaceFound+1;
end
disp('Number of faces found');
numFaceFound;
if (numFaceFound>0)
    disp('Indices of faces found: ');
    ind=find(foundFaces==1);
    CurBB=bboxes(ind,:);
    CurBB
else
    close all;
end
end
```