Digital Image processing Lab

Islamic University – Gaza
Engineering Faculty
Department of Computer Engineering
2013
EELE 5110: Digital Image processing Lab
Eng. Ahmed M. Ayash

Lab # 2

Basic Image Operations in Matlab

February 20, 2013

1. Introduction

This worksheet is an introduction on how to handle images in Matlab. When working with images in Matlab, there are many things to keep in mind such as loading an image, using the right format, saving the data as different data types, how to display an image, conversion between different image formats, etc. This worksheet presents some of the commands designed for these operations. Most of these commands require you to have the *Image processing tool box* installed with Matlab, to find out if it is installed, type ***ver images*** at the Matlab prompt.

2. Fundamentals

A digital image is composed of pixels which can be thought of as small dots on the screen. A digital image is an instruction of how to color/gray level each pixel. A typical size of an image is 512-by-512 pixels. It is convenient to let the dimensions of the image to be a power of 2. For example, $2^9 = 512$. In the general case we say that an image is of size *M*-by-*N* if it is composed of *M* pixels in the vertical direction and N pixels in the horizontal direction.

3. Image formats supported by Matlab

The following image formats are supported by Matlab:

Format	Description	Extension
TIFF	Tagged Image File Format	.tiff .tif
JPEG	Joint Photographic Expert Group	.jpg .jpeg
GIF	Graphic Interchange Format	.gif
BMP	Windows Bitmap	.bmp
PNG	Portable Network Graphics	.png
XWD	X Window Dump	.xwd

Most images you find on the Internet are JPEG-images which is the name for one of the most widely used compression standards for images. An image named **myimage.jpg** is stored in the JPEG format.

4. Working formats in Matlab

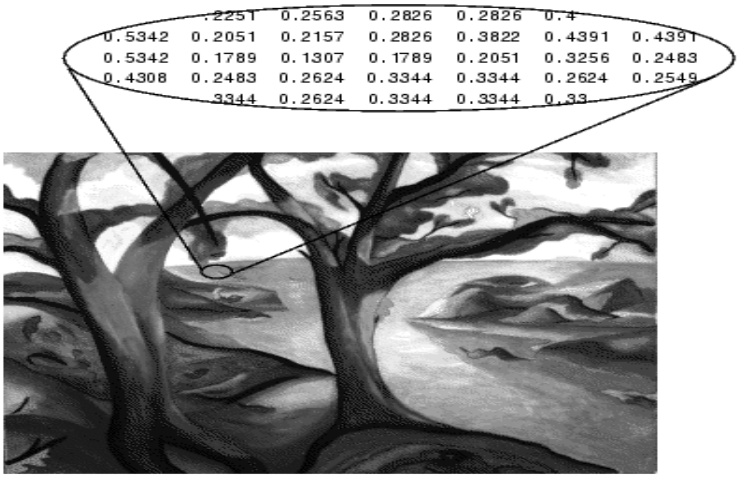
If an image is stored as a JPEG-image on your disc we first read it into Matlab. However, in order to start working with an image, for example perform a wavelet transform on the image, we must convert it into a different format. There are four common formats:

I. Intensity image (gray scale image)

This is the equivalent to a "gray scale image" and this is the image we will mostly work with in this course. It represents an image as a matrix where every element has a value corresponding to how bright/dark the pixel at the corresponding position should be colored. There are two ways to represent the number that represents the brightness of the pixel:

- The **`double`** data type: It assigns a floating number ("a number with the value decimals") between 0 and 1 to each pixel. The value 0 corresponds to black and 1 corresponds to white.
- The **`uint8`** which assigns an integer between 0 and 255 to represent the brightness of a pixel. The value 0 corresponds to black and 255 to white.

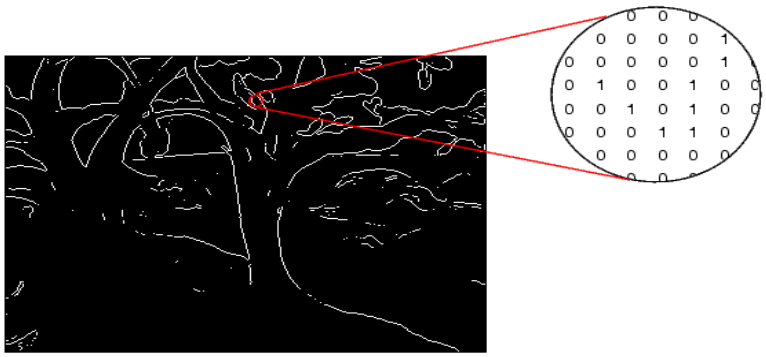
The figure below depicts an intensity image of class *double*.



Pixel Values in an Intensity Image Define Gray Levels

II. Binary image

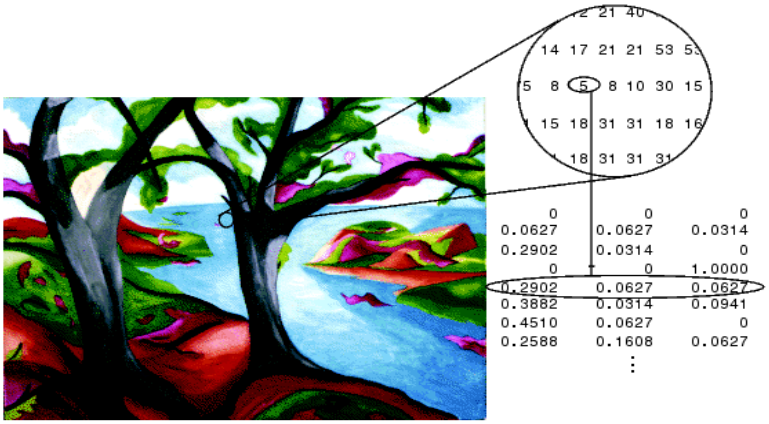
This image format also stores an image as a matrix but can only color a pixel black or white (and nothing in between). It assigns a 0 for black and a 1 for white.



Pixels in a Binary Image Have Two Possible Values: 0 or 1

III. Indexed image

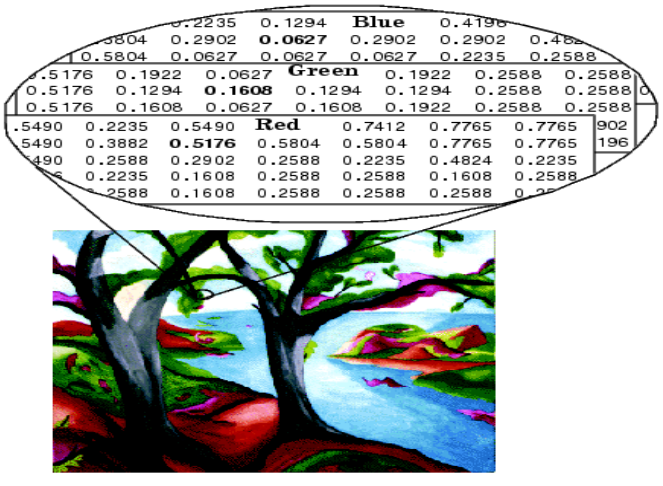
This is a practical way of representing color images. (In this course we will mostly work with gray scale images but once you have learned how to work with a gray scale image you will also know the principle how to work with color images.) An indexed image stores an image as two matrices. The first matrix has the same size as the image and one number for each pixel. The second matrix is called the color map and its size may be different from the image. The numbers in the first matrix is an instruction of what number to use in the color map matrix.
The figure below illustrates the structure of an indexed image. The pixels in the image are represented by integers, which are pointers (indices) to color values stored in the color map.



Relationship of Pixel Values to Colormap in Indexed Images

IV. RGB image

An RGB image, sometimes referred to as a true-color image, is another format for color images. It represents an image with three matrices of sizes matching the image format. Each matrix corresponds to one of the colors red, green or blue and gives an instruction of how much of each of these colors a certain pixel should use.
For example, the red, green, and blue color components of the pixel (10,5) are stored in RGB(10,5,1), RGB(10,5,2), and RGB(10,5,3), respectively.



The Color Planes of an RGB Image

Converting between different formats

The following table shows how to convert between the different formats given above. (Within the parenthesis you type the name of the image you wish to convert.)

Operation	**Matlab command**
Convert from intensity/indexed/RGB format to binary format.	`dither()`
Convert from intensity format to indexed format.	`gray2ind()`
Convert from indexed format to intensity format.	`ind2gray()`
Convert from indexed format to RGB format.	`ind2rgb()`
Convert a regular matrix to intensity format by scaling.	`mat2gray()`
Convert from RGB format to intensity format.	`rgb2gray()`
Convert from RGB format to indexed format.	`rgb2ind()`

The command **`mat2gray`** is useful if you have a matrix representing an image but the values representing the gray scale range between, let's say, 0 and 1000. The command **`mat2gray`** automatically rescales all entries so that they fall within 0 and 1.

Converting between `double` and `uint8`:

When you store an image, you should store it as a **`uint8`** image since this requires far less memory than **`double`**. When you are processing an image (that is performing mathematical operations on an image) you should convert it into a **`double`**. Converting back and forth between these classes is easy.

```
I=im2double(I); % Converts an image named I from uint8 to double.
I=im2uint8(I);  % Converts an image named I from double to uint8.
```

5. Reading and writing image files

Images are usually in form of a file (for example, if you download an image from the web, it is usually stored as a JPEG-file). Once processing has been done, the image may be written back to a JPEG-file. This can be done by ***imread*** and ***imwrite*** commands.

Operation	**Matlab command**
Read an image. (Within the parenthesis you type the name of the image file you wish to read. Put the file name within single quotes ' '.)	**`imread()`**

Write an image to a file. (As the first argument within the parenthesis you type the name of the image you have worked with. As a second argument within the parenthesis you type the name of the file and format that you want to write the image to. Put the file name within single quotes ' '.)	`imwrite(,)`

Make sure to use semi-colon after these commands, otherwise you will get LOTS OF number scrolling on your screen... The commands **`imread`** and **`imwrite`** support the formats given in the section "Image formats supported by Matlab" above.

6. Loading and saving variables in Matlab

This section explains how to load and save variables in Matlab. Once you have read a file, you probably convert it into an intensity image (a matrix) and work with this matrix. Once you are done you may want to save the matrix representing the image in order to continue to work with this matrix at another time. This is easily done using the commands **`save`** and **`load`**. Note that **`save`** and **`load`** are commonly used Matlab commands.

Operation	Matlab command
Save the variable x	`save X`
Load the variable x	`load X`

7. Displaying images in Matlab

Operation:	Matlab command:
Display an image represented as the matrix x.	`imagesc(X)`
Adjust the brightness, s is a parameter such that $-1 \leq s < 0$ gives a darker image, $0 < s \leq 1$ gives a brighter image.	`brighten(s)`
Change the colors to gray.	`colormap(gray)`
Display an image represented as the matrix x.	`imshow(X)`
Zoom in (using the left and right mouse button).	`zoom on`
Turn off the zoom function.	`zoom off`

8. Implementation Details with Results:

8.1 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.

Syntax

```
A = imread(filename,fmt)
```

Description

A = imread(filename,fmt) reads a grayscale or color image from the file specified by the string *filename*, where the string *fmt* specifies the format of the file. If the file is not in the current directory or in a directory in the MATLAB path, specify the full pathname of the location on your system.

8.2 Writing Image Data

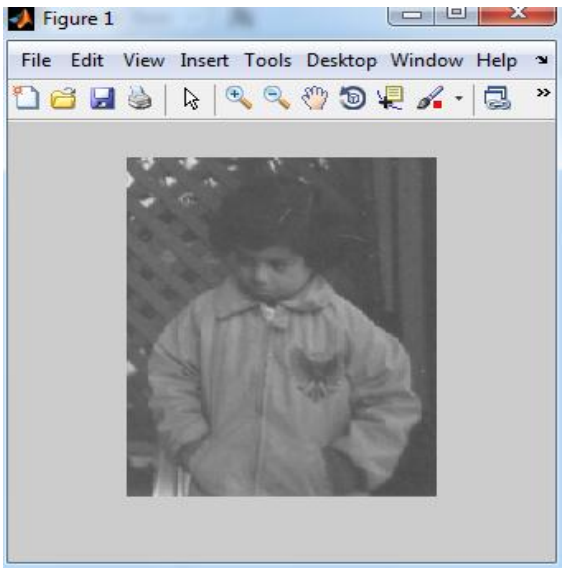
imwrite: Write image to graphics file

Syntax

```
imwrite(A,filename,fmt)
```

Example:

```
%Example1.m
a=imread('lab2_1.tif');
imwrite(a,gray(256),'b.bmp');
imshow('b.bmp')% imshow is used to display image
```



8.3 How to get no. of rows and columns of image

Function **size** gives the rows and columns dimension of image

```
>> [r,c]=size(a)
r =
   291
c =
   240
```

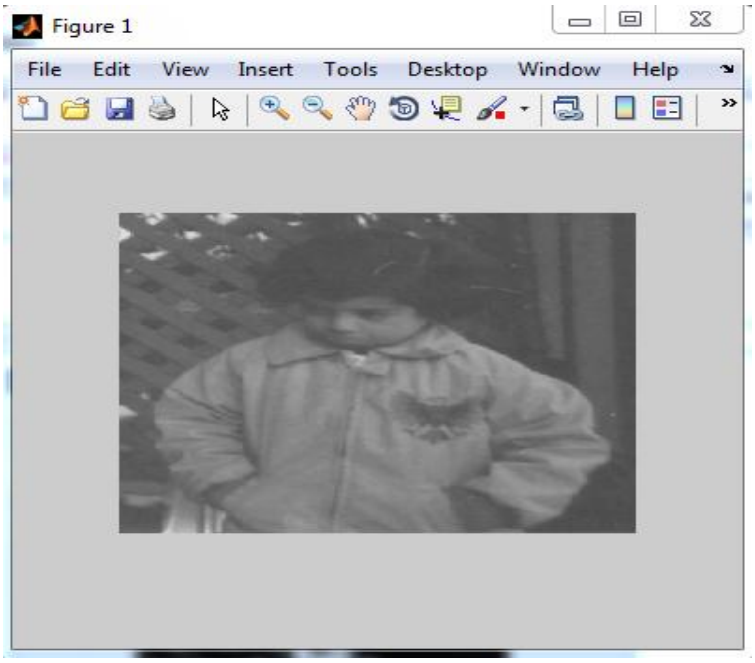
Function **imresize** used to resize image.

Example:

```
%Example2.m
a=imread('lab2_1.tif');
disp('Original Size:');
s=size(a)
a1 = imresize(a, [300 400]);
disp('New Size:');
s2=size(a1)
imshow(a1);
```

Output:

```
>> Example2
Original Size:
s =
   291   240
New Size:
s2 =
   300   400
```

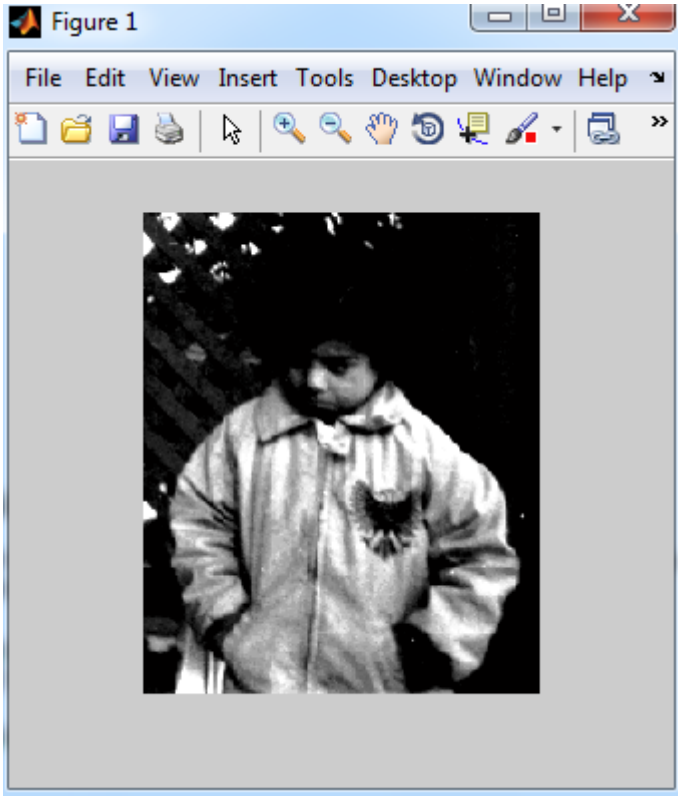


8.4 Image Display

`imshow('lab2_1.tif')`	here by default intensity level is 256
`imshow(f,[low high])`	Displays as black all values less than or equal to low and as white all values greater than or equal to high

Example:

```
imshow(a,[100 150])
```



9. Exercises

• Type in the following commands and see what each command does.

```
%ex1.m
I=imread('D:\MATLAB\toolbox\images\imdemos\liftingbody.png');
 % Load the image file and store it as the variable I.
whos
% Type "whos" in order to find out the size and class of all stored
variables.
save I
% Save the variable I.
ls
% List the files in your directory.
% There should now be a file named "I.mat" in you directory
% containing your variable I.
```

• Next we will see that we can display an image using the command **imshow**

```
%ex2.m
clear
% Clear Matlab's memory.
load I
% Load the variable I that we saved above.
whos
% Check that it was indeed loaded.
imshow(I)
% Display the image
figure(1)
I=im2double(I);
% Convert the variable into double.
[r c]=size(I);
whos
% Check that the variable indeed was converted into double
% The next procedure cuts out the upper left corner of the image
% and stores the reduced image as Ired.
for i=1:r/2
  for j=1:c/2
     Ired(i,j)=I(i,j);
  end
end
whos
% Check what variables you now have stored.
figure(2)
imshow(Ired)
% Display the reduced image.
```

Output:

Figure1

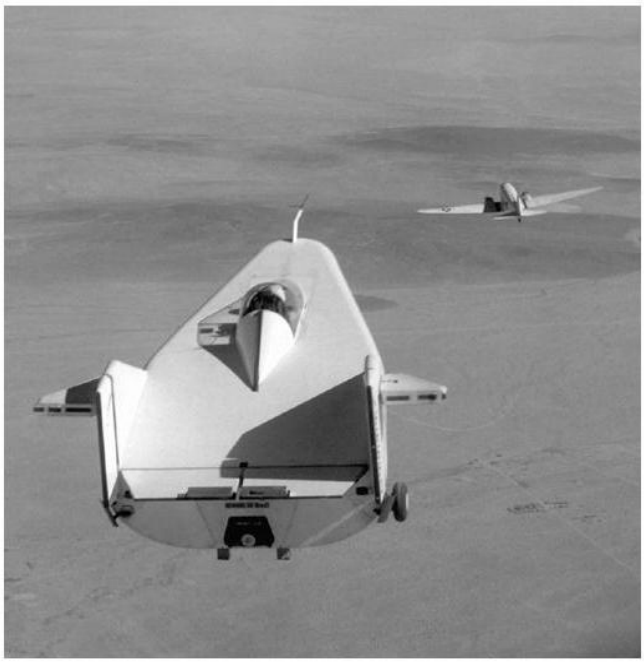


Figure2

Homework:

Write a program to produce mirror image of the image passed to it; also display both the original and mirror image as follows