LAB 4

Fall 2010, BESE- 13 & 14

Fundamentals of MATLAB

Objective

The purpose of Today's lab is to explore MATLAB functions for nearest neighbor interpolation and image labeling. Also, it gives introduction to new GUI controls and operations. By the end of this lab, you should be able to 1) write your own algorithm for resizing an image using 'nearest neighbor interpolation', 2) label a binary image using built-in MATLAB function and 3) manipulate GUI controls' properties at runtime.

Submission Requirements

You are expected to complete the assigned tasks within the lab session and show them to the lab engineer/instructor. Some of these tasks are for practice purposes only while others (marked as '*Exercise'* or '*Question*') have to be answered in the form of a lab report that you need to prepare. Following guidelines will be helpful to you in carrying out the tasks and preparing the lab report.

Guidelines

- In the exercises, when you are asked to display an image, you have to put the image displayed in your project report. You may either save the image as 'jpeg' (File->Save As) or add it to the report or use the 'Print Screen' command on your keyboard to get a snapshot of the displayed image. This point will become clear to you once you actually carry out the assigned tasks**.**

- Name your reports using the following convention:
  - ***Lab#_Rank_YourFullName***

  - '#' replaces the lab number
  - '*Rank'* replaces Maj/Capt/TC/NC/PC
  - '*YourFullName*' replaces your complete name.

- You need to submit the report even if you have demonstrated the exercises to the lab engineer/instructor or shown them the lab report during the lab session.

Tasks for Today

1. Nearest Neighbor Interpolation

Interpolation is the process of estimating an appropriate number between any two numbers. For instance, in a resized image we interpolate an intensity value between any two pixels locations with known intensities. MATLAB Image Processing Toolbox provides a built-in function to resize an image:

om = imresize(im, r, m)	**returns a resized image ‘om’.** Where, *im* is an input image and *r* is a resize factor. The size of output image is less than input image for all *r* ϵ (0, 1], except *r* > 1 when size of output image is greater than input image. *m* is a method name e.g. ‘nearest’, ‘bilinear’, or ‘bicubic’.

Example:	>> im = imread(‘cameraman.tif’);
	>> om = imresize(im, 2, ‘nearest’);

<u>ALGORITHM TO RESIZE AN IMAGE</u>:

This part discusses the **nearest neighbor replication** algorithm. For instance, given an image matrix ‘im’ of the size m1xn1 and we want to resize it to the order m2xn2, as shown below:

(1,1)	**(1,2)**	**…**	**(1,n1)**
(2,1)	**(2,2)**	**…**	**(2,n1)**
…	**…**	**…**	**…**
(m1,1)	**(m1,2)**	**…**	**(m1,n1)**

(1,1)	**(1,2)**	**…**	**…**	**(1,n2)**
(2,1)	**(2,2)**	**…**	**…**	**(2,n2)**
…	**…**	**…**	**…**	**…**
…	**…**	**…**	**…**	**…**
(m2,1)	**(m2,2)**	**…**	**…**	**(m2,n2)**

This implies that:

1) 1 pixel in an image of size m1xn1 is vertically resized to m2/m1 pixels in image of order m2xn2.
   i.e. row resize ratio (height) = m1/m2
2) 1 pixel in an image of size m1xn1 is horizontally resized to n2/n1 pixels in an image of order m2xn2.
   i.e. column resize ratio (width) = n1/n2

So an algorithm to resize image using **'nearest neighbor replication'** can be described as follows:

ri:	No. of rows in an input image
ci:	No. of columns in an input image
ro:	No. of rows in an output image
co:	No. of columns in an output image
r_ratio:	Row resize ratio
c_ratio:	Column resize ratio

```
r_ratio = ri / ro
c_ratio = ci / co
for each pixel index 'i' and 'j' in a resized image
    // calculate index of input image whose value is to be copied
    x = ceil(i*r_ratio)
    y = ceil(j*c_ratio)
    new_im(i, j) = org_im(x, y)
 [end loop]
```

Exercise 1:

Write a function named 'resize' to program nearest neighbor replication algorithm for a gray image/matrix:

INPUTS: 1) image and 2) resizing factor.

OUTPUTS: resized image. Show the original as well as resize image to prove the results of algorithm.

2. Image Labeling

Labeling is process of clustering the pixels into components on basis of pixel neighborhood *(e.g. 4-connectivity)* such that all connected pixels share similar intensity value and are in some manner connected with each other. Once all the groups have been identified, each pixel is labeled with a gray-level or a color (pseudo coloring).

Usually labeling is used to identify the number of objects and their respective area (i.e. no. of pixels). MATLAB's built-in function to perform image labeling is bwlabel, defined as follows:

[L, num] = bwlabel(im, n)	**labels image 'im' with 'n'-connectivity and returns a labeled image L with number of connected objects found 'num'.** *im* is a binary image and n is neighborhood connectivity (e.g. 4 or 8).
Example:	>> im = imread('particles.png'); >> [om, no] = bwlabel(im2bw(im), 4);

3. Dealing with Control Properties at Runtime

MATLAB uses *Tag* property as a way to get an access to the controls for reading and/or setting their status. This tag property is stored as a fieldname of a global structure named **handles**. A handle of a control can be accessed by handles.control_tag. Here, apart from handles, control_tag is will be replaced with the tag of a control.

Setting value of a Control's property at runtime:

In MATLAB the value of a specific property of a control can be modified at runtime by using *set* function, as defined below:

set(handles.control_tag, 'property', 'value');

Where *handles.control_tag* is the handle of a control we want to access. *'property'* is the property of a control we are accessing (as seen by property inspector). *'value'* is the new value of this property that is to be set.

For Example:

```
% to Disable the Browse button
>> set(handles.btnBrowse, 'Enable', 'off');

% set the value of String to 512, as displayed on StaticText control
>> set(handles.txtWidth, 'String', ‘512’);
```

Reading status/value of a Control's property at runtime:

In interactive systems, it is quite desirable to take actions on the basis of event(s). For this purpose we need to read the status or value of a control e.g. toggles button, textbox, slidebar etc. MATLAB offers a *get* function to accomplish this task:

get(handles.control_tag, 'property');

Here *handles.control_tag* is the handle of a control we want to access. *'property'* is the property of a control whose value or status is to be read.

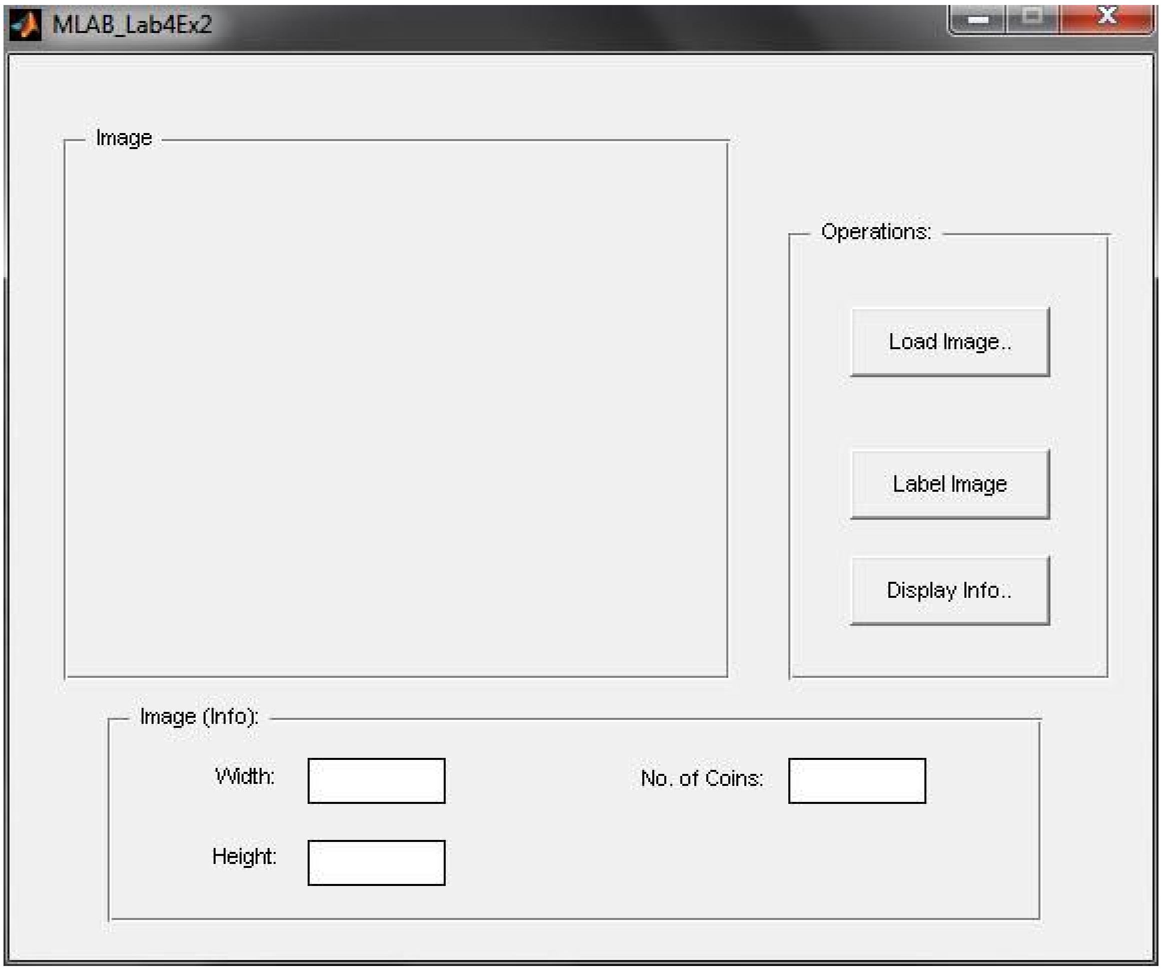
For Example:

```
% this will get the current Status of a toggle button
% return 1 when button is pressed, else return 0
>> val = get(handles.tbtnDisplay, 'Value');
```

Exercise 2

Design a MATLAB GUI, as shown below, to implement following operations:

1) Load Image: read image *‘coins.png’* from MATLAB repository and display into axes.
2) Label Image: Label the above read image using *‘bwlabel’* function. Populate information about image width, height and no. of coins in ‘static text’ controls. And display the labeled image into axes.
3) Display Info: This button should be a toggle button which is if pressed shows the information panel otherwise it will hide the panel.


MLAB_Lab4Ex2
Image
Operations:
Load Image..
Label Image
Display Info..
Image (Info):
Width:
Height:
No. of Coins: