

LAB 3

Fall 2010, BESE- 13 & 14

Fundamentals of MATLAB

Objective

The aim of Today's lab is to introduce you to interactive I/O operations in MATLAB and creating and handling Graphical User Interfaces (GUIs). By the end of this lab, you should be able to 1) display information/instructions to user and accepting user inputs from keyboards and 2) deal with GUIs for performing basic functions in MATLAB Toolbox for Digital Image Processing.

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. Interactive I/O

It is sometimes required to display certain instructions or information to user and receiving user feedback or input through keyboard. For this purpose MATLAB offers a set of functions which provide the flexibility for interactive dealing of I/O operations. Following are the functions for interactive I/O:

- 1. disp(arg)** **displays information on a monitor screen depending upon the type of arguments passed.**
where, arg can either be a constant string e.g. 'Digital Image Processing' or it can be some variable or array e.g. disp(var).

Example:

```
>> disp('Digital Image Processing')
Digital Image Processing
>> a = 100;
>> disp(a);
100
>> mat = [1 2; 2 1];
>> disp(mat);
1  2
2  1
```

- 2. a = input('string')** **prompts for the user input from keyboard.**
where '**string**' is the information or instruction to be displayed on screen. And, **a** can be a vector or variable.

Example:

```
% to read a single character or number from a keyboard
>> val = input('Enter the value of n: ');
```

Enter the value of n: 100

```
>> disp(val);
```

```
>> 100;
```

% to read *multiple characters or numbers* from a keyboard

```
>> val = input('Enter value of x, y, and z: ', 's');
```

Enter the value of x, y, z: 100 200 300

```
>> disp(val);
```

```
100 200 300
```

Note: in above statement 's' allows the input command to read multiple elements from keyboard separated by space.

3. [a, b, c, ...] = strread('string', 'format', N, 'param', 'value')

Reads the data from string in a specified format.

where '**string**' is the input string which is to be read.

'**format**' of data can be %s, %c, %d, %u, %f. '**param**' can be '*delimiter*' or '*whitespace*' with its corresponding '**value**'.

Example:

```
>> str = '200.25 300.02 400.11';
```

```
>> res = strread(str, '%s', 2, 'delimiter', ' ');
```

```
>> disp(res)
```

```
'$200.25'
```

```
'$300.02'
```

Note: this we can also use this function to read string as unsigned int, float, double etc.

Exercise 1:

Write a function to apply resize operation (nearest neighbor replication) on a grayscale image:

INPUTS: 1) image and 2) name of resize operation i.e. 'nearest'.

USER INPUTS: Prompt user to enter resize factor from keyboard.

OUTPUTS: resized image. Use disp command to show operation name and resizing factor.

2. Cell Arrays and Structures

Cell arrays and structures are used to group together heterogeneous data (data of different data types). Cell arrays are kind of array of arrays. They are defined and used as follows:

```
Example:      % create a cell array
                >> cel_arr = { 3, 'DIP LAB', [13 14] };
                % access array elements
                >> cel_arr{3}
                13 14
                >> cel_arr{2}(5:7)
                LAB
```

Structures are analogous to the cell arrays, allow grouping of dissimilar data. But unlike cell array, where cells are accessed by a number, they are referred by specific field names.

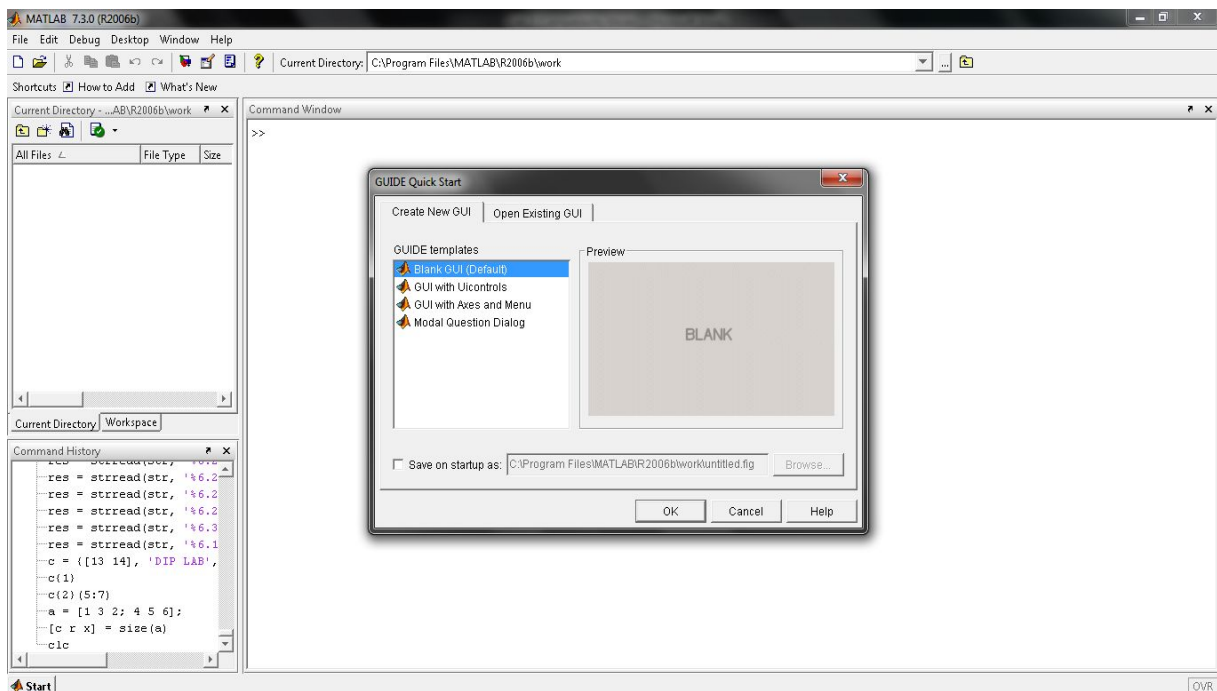
```
Example:      % create a structure
                >> mat = [1 2 3; 4 5 6];
                >> [rows cols d] = size(mat);
                >> matrix.rows = rows;
                >> matrix.cols = cols;
                >> matrix.depth = d;

                % access rows element of a structure
                >> matrix.rows
                2
```

3. Creating Graphical User Interface (GUI)

In MATLAB whenever Graphical User Interface (GUI) is created it possesses two files **.fig** and **.m**. The **.fig** file contains user interface design (figure layout and other controls) of MATLAB. While **.m** file holds the user functions and entire event handling code. To create a GUI in MATLAB, follow the steps in a sequence as listed below:

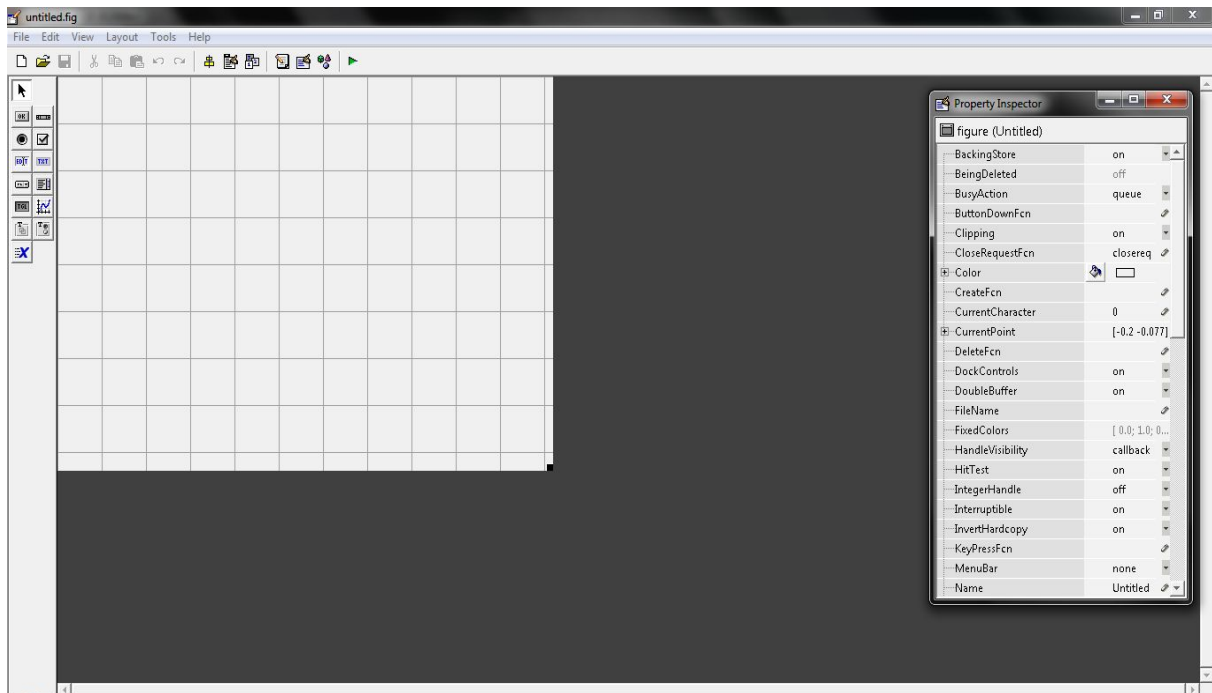
Step-1: Open the MATLAB, and goto File Menu → New → GUI. This will display a dialogue box titled 'GUIDE Quick Start' as follows:



Step-2: Select the 'Create New GUI' tab and then from the GUIDE templates select 'Blank GUI (Default)'.

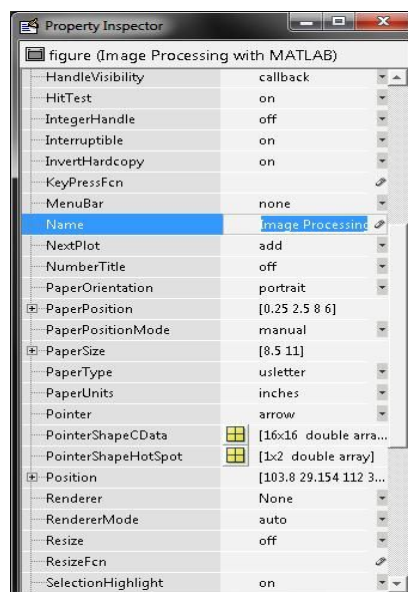
*Note: Since we've not yet created any GUI thus we didn't select 'Open Existing GUI'.
However, if you already have one then you may browse through that figure.*

Step-3: Finally, Click OK. This will display a window with a set of controls on its left side, as shown below:



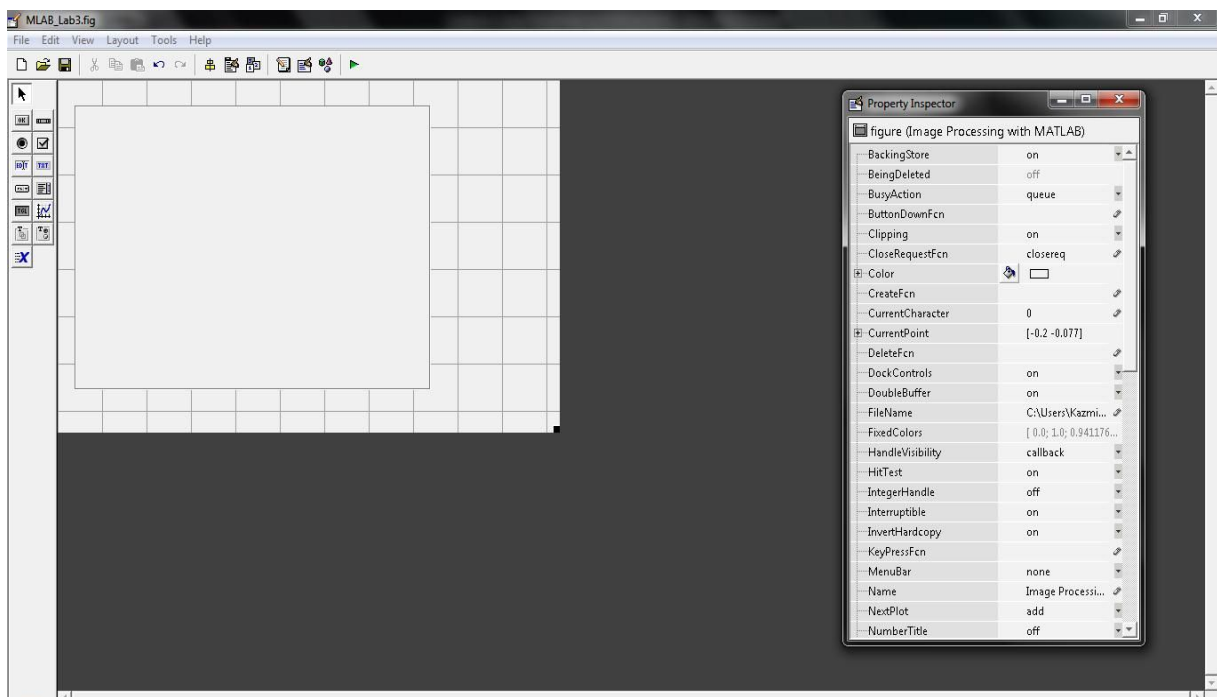
Step-4: Click on the File Menu and select 'Save as' to save the GUI. Browse the path where you want to save and provide the name GUI e.g. *MLAB_Lab3*.

Step-5: Right-Click on grid, select 'Property Inspector' from context menu. This will display a property inspector window on the screen.



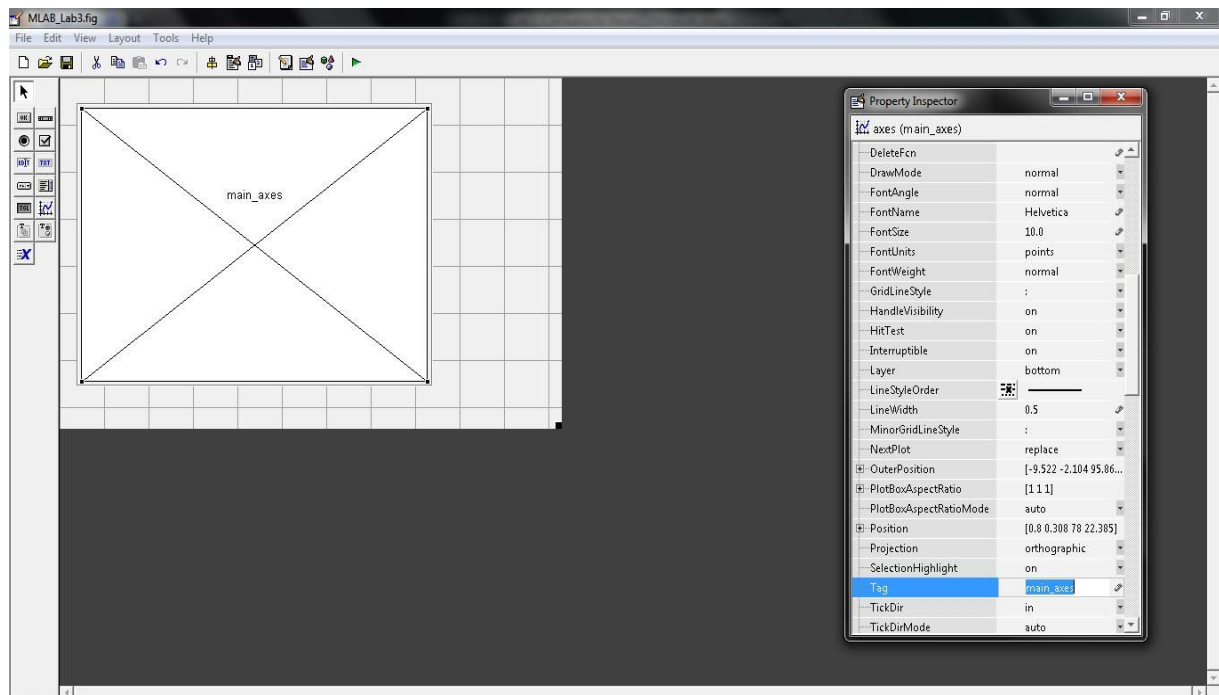
Scroll down to 'Name' property and change Name value to Image Processing with MATLAB or whatever Title you want. Next scroll to the 'Tag' property and change its value from figure1 (default) to main_fig. Finally, scroll down to the 'ToolBar' property and select 'none' by clicking it.

Step-6: Drag the control name 'Panel' from left panel and drop onto grid. Adjust its size with mouse according to need. Right-Click the 'Panel' control and then select 'Property Inspector' from context menu. Scroll down to the 'Tag' property and set its value to main_uipanel. Finally, clear the value of property 'Title'.

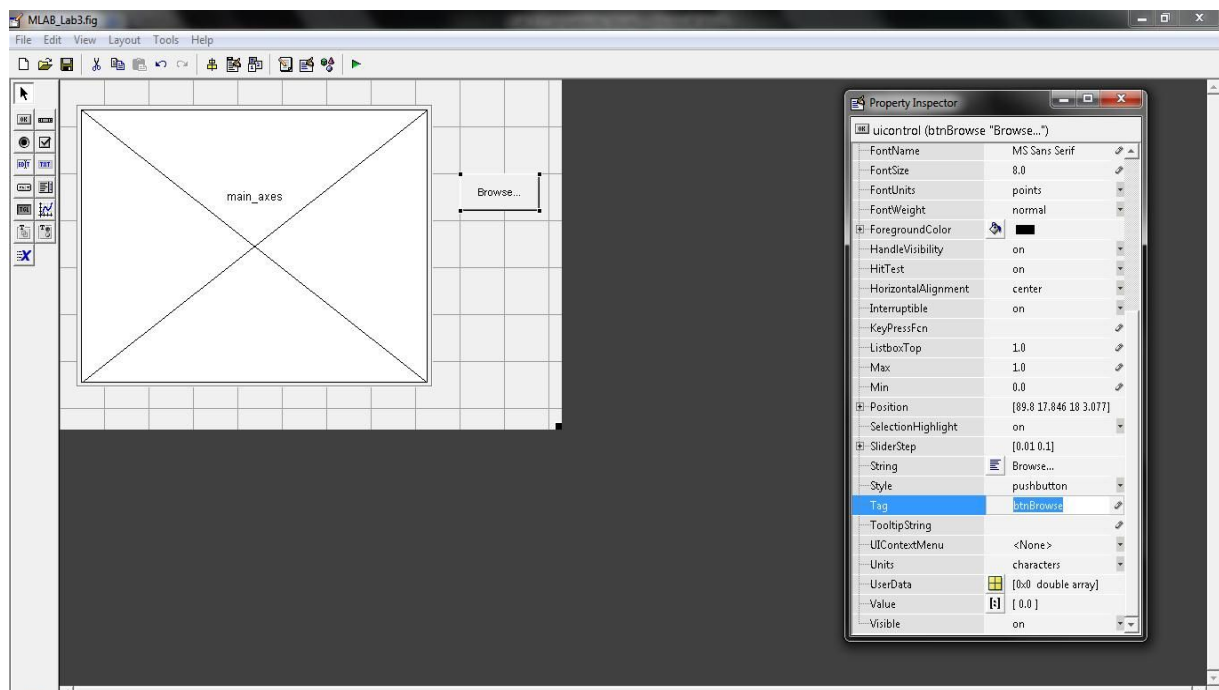


Step-7: Drag the control named 'Axes' from the left panel and drop it onto 'Panel'. Adjust its size with mouse according to need. Right-Click on the 'Axes' and select 'Property Inspector' from context menu.

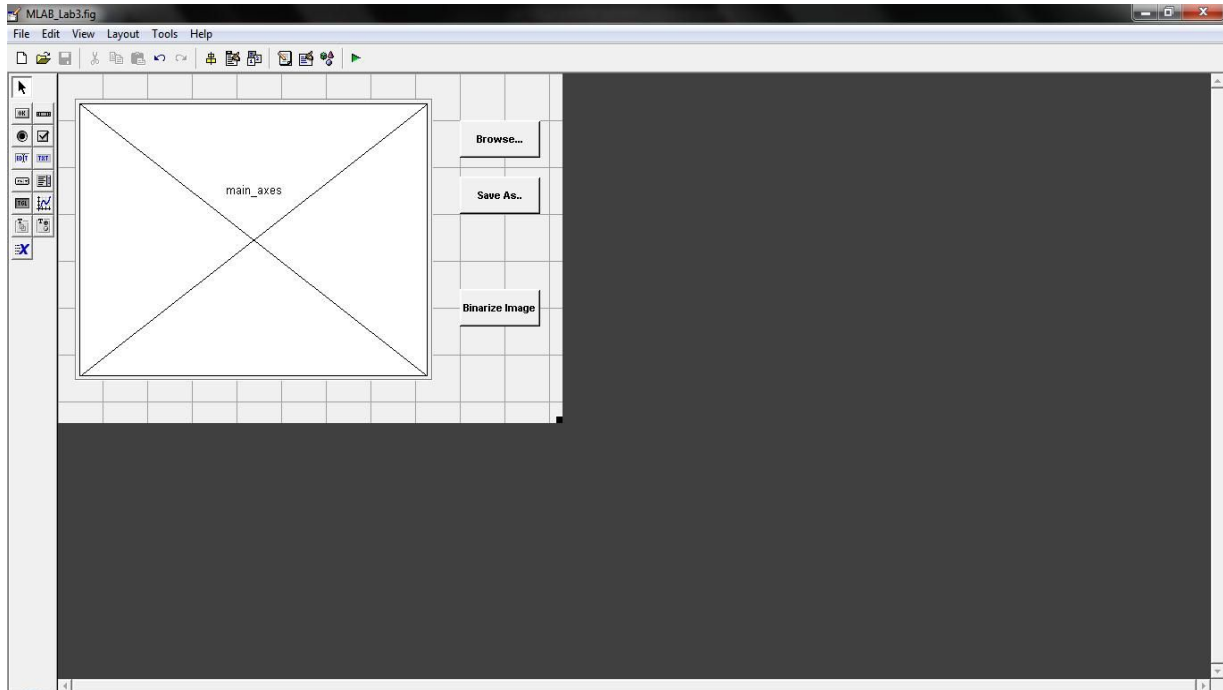
Scroll down to the 'Tag' property and change its value from axes1 (default) to main_axes. Finally, scroll to the 'Visible' property and set it off.



Step-8: Drag and drop the control named 'Button' onto grid. Adjust its size with mouse as per need. Next goto the 'Tag' property of Button and set its value to btnBrowse. Finally, scroll down to the 'String' property and set its value to Browse.



Setp-9: Repeat **Step-8** to add 2 more buttons. Set their 'Tag' property to btnBinary and btnSave. And set the value of 'Name' property to Binarize Image and Save respectively. Set the 'Style' propoerty of btnBinary to 'togglebutton'.

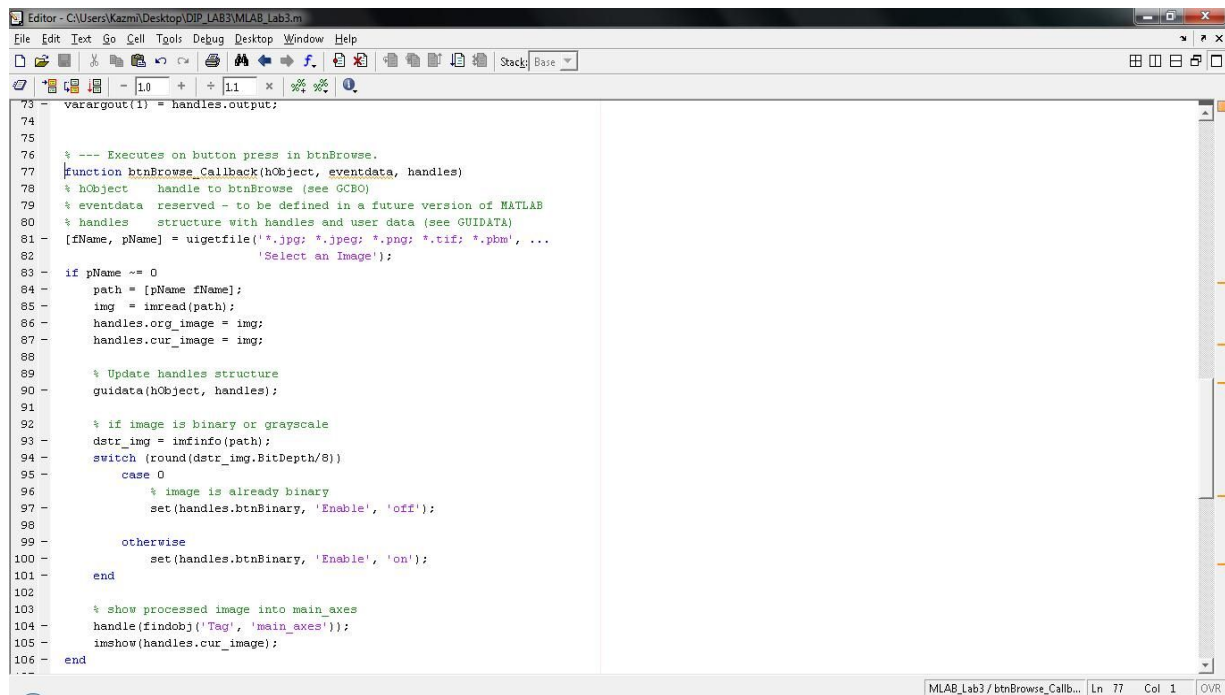


Step-10: In the current figure window, go to View Menu → M-file Editor which would display the code of this GUI. Scroll down to **function** btnBrowse_Callback(hObject, eventdata, handles) and type the following code after comments:

```
[fName, pName] = uigetfile('*.jpg; *.jpeg; *.png; *.tif; *.pbm', ...
    'Select an Image');
if pName ~= 0
    path = [pName fName];
    img = imread(path);
    handles.image = img;

    % if image is binary or grayscale
    dstr_img = imfinfo(path);
    switch (round(dstr_img.BitDepth/8))
        case 0
            % image is already binary
            set(handles.btnBinary, 'Enable', 'off');
        otherwise
            set(handles.btnBinary, 'Enable', 'on');
    end

    % show processed image into main_axes
    handle(findobj('Tag', 'main_axes'));
    imshow(handles.image);
end
```



```

73 - varargin{1} = handles.output;
74
75
76 % --- Executes on button press in btnBrowse.
77 function btnBrowse_Callback(hObject, eventdata, handles)
78 % hObject    handle to btnBrowse (see GCBO)
79 % eventdata  reserved - to be defined in a future version of MATLAB
80 % handles    structure with handles and user data (see GUIDATA)
81 [fileName, pName] = uigetfile('*.jpg; *.jpeg; *.png; *.tif; *.pnm', ...
82     'Select an Image');
83
84 if pName ~= 0
85     path = [pName fileName];
86     img = imread(path);
87     handles.org_image = img;
88     handles.cur_image = img;
89
90     % Update handles structure
91     guidata(hObject, handles);
92
93     % if image is binary or grayscale
94     dstr_img = imfinfo(path);
95     switch (round(dstr_img.BitDepth/8))
96     case 0
97         % image is already binary
98         set(handles.btnBinary, 'Enable', 'off');
99
100     otherwise
101         set(handles.btnBinary, 'Enable', 'on');
102     end
103
104     % show processed image into main axes
105     handle(findobj('Tag', 'main_axes'));
106     imshow(handles.cur_image);
107 end

```

Setp-11: Scroll down to function btnSaveAs_Callback(hObject, eventdata, handles) and write down the following code:

```

[fileName, pathName, filterIndex] = uinputfile(
    {'Supported Formats (*.jpg; *.jpeg; *.png; *.tif; *.pnm)'; ...
    '*.jpg'; '*.jpeg'; '*.png'; '*.tif'; '*.pnm'}, 'Save image as');
if pathName ~= 0
    if filterIndex <= 1
        errordlg('Image format {e.g. *.jpg etc} NOT selected!', ...
            'Illegal Operation!');
        return;
    end

    path = [pathName, fileName];

    % write current_image file onto the disk path
    imwrite(handles.cur_image, path);
end

```

Setp-12: Move to the function btnBinary_Callback(hObject, eventdata, handles) and do as directed below:

```

if (get(handles.btnBinary, 'Value'))
    handles.cur_image = im2bw(handles.org_image);

    %display image
    handle(gca);
    imshow(handles.cur_image);

else
    handles.cur_image = handles.org_image;
end

```

```

    %display image
    handle(gca);
    imshow(handles.cur_image);
end

% Update handles structure
guidata(hObject, handles);

```

Finally, in M-file Editor go to Debug Menu → Run. Alternatively, press F5 to execute the code.

Exercise 2

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

- 1) Browse: load image e.g. 'lena.png'.
- 2) Quantize: prompt user to enter n-bits per pixel (1 – 8) and quantize image to the specified gray level.

(Hint: Normalize the intensity 'i' of org_image(x, y) as follows:

$$new_img(x, y) = \text{round}((i - \min) / (\max - \min) * L)$$

where, max and min are maximum and minimum intensity values in an org_image and $L = 2^{n\text{-bits}}$ gray-levels).

- 3) Binarize: convert image into black & white image such that for all $img(x, y) > 127 = 1$ else 0. **Don't use im2bw**

