Politechnika
Wrocławska

dr inż. Jacek Jarnicki, dr inż. Marek Woda
Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology
{jacek.jarnicki, marek.woda}@pwr.wroc.pl

Multimedia & Computer Visualization

Exercise #1

MATLAB® Environment + Image Processing Toolbox - Introduction

The purpose of this exercise is to gain insight on the elementary intuition of digital image as a collection of data stored in computer memory and knowledge of the simplest operations performed on digital images. MATLAB will be used as a tool for environmental engineering calculations supported by Image Processing Toolbox. As a preliminary point it will be demonstrated simple examples that illustrate the capabilities of MATLAB and how to write programs using the functions provided by the Image Processing Toolbox.

1. MATLAB - general description of the system

MATLAB is a programming environment designed to support scientific and engineering calculations. The name of the program comes from the words MAT*rix* LAB*oratory*, because at the beginning of its existence, MATLAB was designed primarily to perform numerical calculations in the field of matrix calculus. Subsequent editions have made it more versatile tool, and now it is a comprehensive system for various types of scientific computing, engineering, and computer simulations, allowing for easy and comprehensive presentation of the results of calculations in graphical form. MATLAB has its own programming language. It is a high-level language with syntax similar to C language. Following element can be used loops, conditional statements, functions, structures, and one can write object-oriented programs. About MATLAB special appeal and popularity decides so-called "toolboxes," or sets of functions written in MATLAB allowing to perform calculations using algorithms specific to the different branches of sciences: Mathematics, Engineering and many other fields, from Biology to Finance. Currently producer of MATLAB - MathWorks company offers dozens of such packages. In the field of image engineering there are three packages available: Image Processing Toolbox, Image Acquisition Toolbox and the Mapping Toolbox. The first, most extensive, contains hundreds of functions that reflect image processing algorithms known from literature, and various support functions with the help of which one can build and test their own algorithms running on digital images. The second package Acquisition Toolbox is designed to support various types of image acquisition devices (both static and dynamic), i.e. cameras, scanners, microscopes connected to a PC. The third package Mapping Toolbox is intended to analyze and visualize geographic maps. At the laboratory classes Image Processing Toolbox will be used only in the limited scope.

2. MATLAB + Image Processing Toolbox – basic capabilities demonstration

At the beginning, let's focus on a simple example in order to give you an idea about how powerful and the same time easy to use MATLAB is, assisted by Image Processing Toolbox package. Following example is to illustrate how you can improve the quality not very well lit photo. Steps you have to perform are as follows:

- Preparation to work and launching MATLAB enviroment

Before you start – it is highly advisable to create a folder (in the part of HDD where you have been permitted to write) with easy to remember name in which you will store your data and programs written during classes. Then copy to this folder the unpacked contents of **`Excercise_1.zip`** file.

Double-click the MATLAB icon (icon) and wait until a full initialization of the environment. This can take up to several to tens of seconds, because the software installed in the laboratory has a network license and to run the program it requires communication with the license server. After the initialization screen similar to given below should appear:

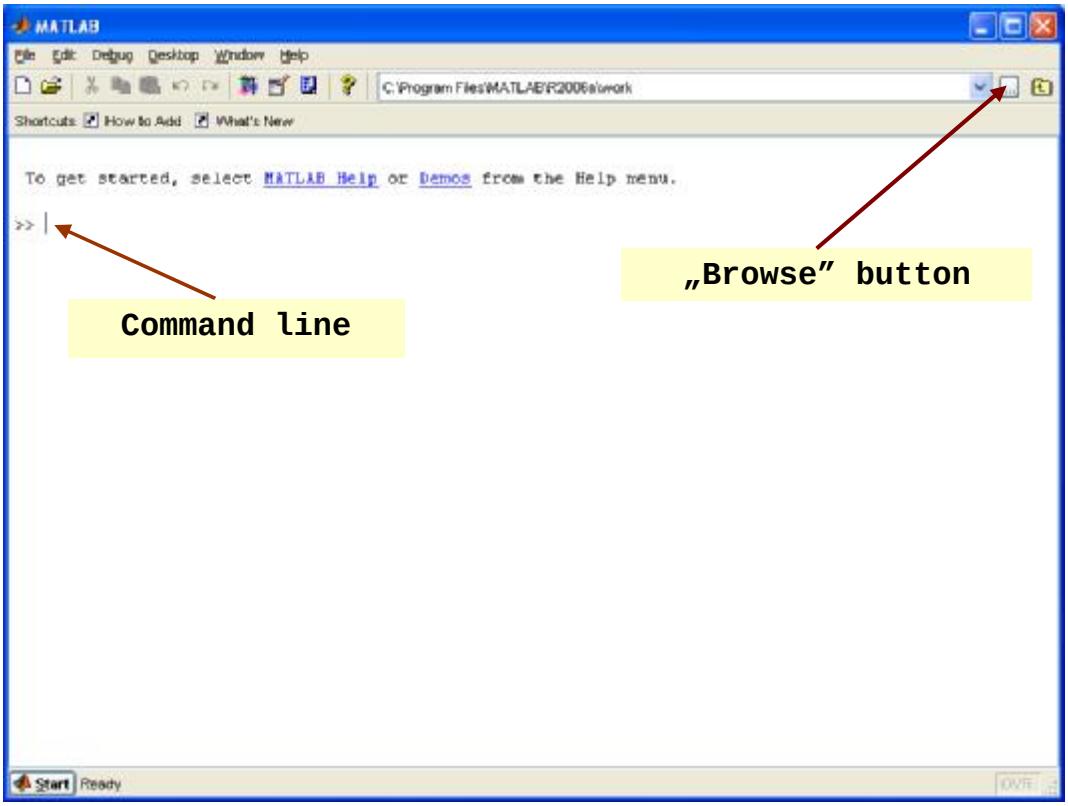

Fig. 1. Appearance of the MATLAB console and its environment

Next please button "Browse" in order to set the path to the folder created earlier. The environment is now ready for operation. After typing, in the command line (just after the symbol >>), any command that can be understood by MATLAB interpreter and pressing ENTER, it will be immediately executed.

- Reading an image, creation an array with the image

Please type in MATLAB command line:

```
>> I = imread('Lena_1.tif');
```

And pass typed in command for execution by pressing ENTER. There will be nothing new visible on the screen, but the execution of this command will read the image from the file and save ***Lena_1.tif*** in memory in the form of two-dimensional array, called I (of course, the array name can vary). This table contains data describing the image. To learn more about this table, type in the following command:

```
>> whos
```

Array **I** description text will pop up on the screen

```
Name        Size                    Bytes  Class

I         256x256                   65536  uint8 array

Grand total is 65536 elements using 65536 bytes
```

The interpretation of the displayed message is simple. Table **I** has a size of 256x256 and contains data as 8-bit unsigned integers (uint8), describing the individual elements of the image. The total number of bytes occupied by data describing the loaded image is 65536. Creation an array with image content enables direct access to each array element, meaning the ability to access each element of the image. For example, you can see what number describes the element (pixel) with coordinates (85, 35), by typing the command line:

```
>> I(85,35)
```

After the command is executed there will number 72 displayed on the screen. It is just a data, corresponding to the pixel with coordinates (85.35) (keep in mind that upper left corner of the image has following coordinates (0,0) expressed in pixels). What exactly this number describing the pixel means it will be explained later. It should be noted a very important convention occurring in the MATLAB programming language. If there is no comma at the end of the command line with the content MATLAB will display the result of its execution directly on the screen (as it was in the last example). If the semicolon is typed in, and then Enter is pressed, the command is executed, but without the effect of the displaying a message. The result of the semicolon at the end of the line can be seen by typing commands at the command line:

```
>> I(85,35);
```

- Display the image read

In order to see finally the image saved in table **I**, it is necessary to type in following command:

```
>> imshow(I)
```

A new window will be opened and the image, loaded from **Lena_1.tif** file, will be displayed. The result looks as follows:

Fig. 2. Loaded up image from Lena_1.tif file

Stored in the array **I** image turned to be a monochrome one. It should be noted that the new window called (figure) has its own interface allowing executing basic operations on the image such as enlarging, reducing, rotating, or writing to a file.

Direct access to the picture elements through the array **I** created in the previous step, , allows for relatively easy changes to the image content.

For example, after the following command execution:

```
>> I(85,35) = 236;
```

changes the value from 72 to 236 of the pixel with coordinates (85, 35). The result of this change can be observed executing following command again

```
>> imshow(I)
```

Using the zoom tool, you can now see what has changed in the content of the displayed image. It should be noted that changes made only applies to the contents of Table **I**, not to the image saved in **Lena_1.tif** file.

- Histogram drawing

At a closer look at this displayed image one can immediately see that it does not look good. The main drawback of the image is poor use of grayscale available. Practically points that are very bright and definitely dark are missing. For the numerical evaluation of

grayscale utilization, a method consisting in chart analysis called a histogram can be used. This graph shows the quantitative number of image pixels belonging to different gray levels. Calculation and drawing an image histogram in MATALAB environment with a package Image Processing Toolbox is an easy task. Just use the commands:

```
>> figure

>> imhist(I)
```

The task for **`figure`** command is to open a new window graphics, in which a histogram will be drawn. Without issuing this command histogram would be drawn in the already open window, causing blurring of the displayed picture. **`imhist`** command will calculate and draw a graphical histogram chart of image stored in TABLE **I** in the last open window. A histogram looks like this:

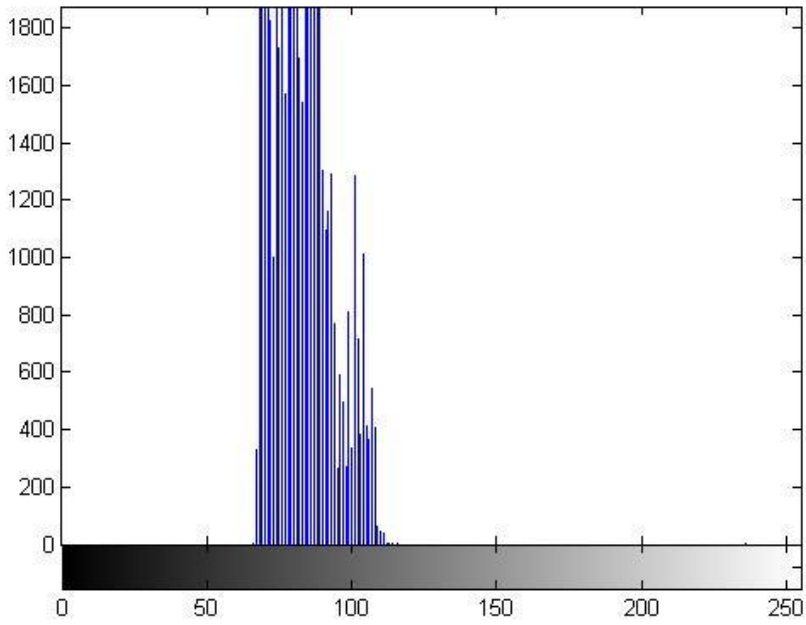

Fig. 3. Histogram of the image stored in the array I

The graph shows clearly that it was used practically only the middle part of the grayscale.

- Image quality improvement by histogram equalization

There are a number of algorithms, described in the literature of area called "Image Processing", used to convert one image into another. Such transformations, sometimes mathematically very complicated, are performed to achieve different purposes. One of them is image quality improvement. To improve the quality of the image that was loaded and displayed in the previous exercise section, an algorithm that changes the brightness of pixels in order that its histogram became aligned as much as possible will be applied. Histogram equalization will certainly make better use of grayscale available and thereby the appearance of the image will be improved.

Without going into unnecessary detail, let's put it simple the algorithm deliberately lighten some pixels in the image and dimming the others. An example of histogram equalization algorithm can be found in [**1**]

In order to transform the image in another with a more balanced histogram pelase type in following command:

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

Running **`histeq()`** function will create a new array named **J**, in which the image is saved resulting from the algorithm execution. You can check this by issuing the given below command again.

```
>> whos
```

- Displaying improved image

What we have left is to see how the corrected image stored in an array **J** look like after the transformation. In order to do so type in two already had used commands:

```
>> figure
```

```
>> imshow(J)
```

The first of these is used again in order to create a new graphic window without destroying what has already been shown in previously open windows. The second command is to show you how new image obtained by histogram equalization looks like. Such effect is shown on Figure 4

Fig. 4. Image after histogram equalization

It could be easily concluded, by making comparison of the source image stored in **I** array, and transformed one stored in **J** array, that the histogram equalization operation gave good results. As you can see the histogram equalization greatly improved the general appearance of the image. One can observe the so-called increased image dynamics, i.e. the difference between the brightest and darkest image elements, which undoubtedly influences the image plasticity augmentation. Please be aware that the operation carried out is in some sense artificial manipulation, introducing information to the image, which does not come from reality, and only is shaped as a result of mathematical operations.

Now we can see the histogram of the processed image. You can do this by typing it already known commands:

```
>> figure

>> imhist(J)
```

Now the processed image histogram looks like this:

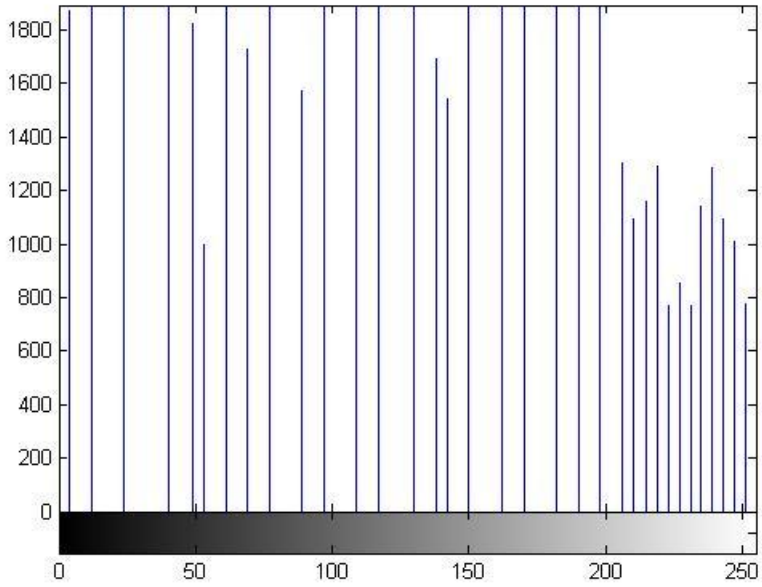

Fig. 5. Histogram of the processed image J

You can see that the histogram of the processed image is much more balanced. As a result of execution of the whole procedure described so far, it has been open four windows graphics. In the first two there are the source image and its histogram, while the next two contain the effect of processing - the resulting image and the corresponding histogram.

- Saving improved image to a file

The last activity in this example is saving the processed image from **J** array to a file. It not necessarily must be the same file type, which the source image was read from (it was a TIFF file in our case). For example, you can save processed image in compressed form using the JPEG algorithm. In order to do this, simply type in the following command:

```
>> imwrite(J,'Lena.jpg')
```

Execution of the command will save the data describing the image in compressed form to a file named **`Lena.jpg`**. It should be noted that the source image was saved in TIFF. This format is used to save images in uncompressed form.

Please compare now the size of both files to see what effect gave the compression.

3. The first program in MATLAB programming language

In the previous section demonstrated one of MATLAB capabilities to improve image quality. The procedure consisted of sequence of commands typed in the command line and passed to the execution by pressing ENTER key. For more complicated tasks, such procedure may be uncomfortable for obvious reasons. A good option is to save the commands in the form of a program. As mentioned at the beginning of MATLAB environment has its own programming language. This language is widely regarded as one of the high-level programming languages like C, Pascal, Basic and others. Programs written in this language can be executed but only in MATLAB environment, or on computers that MATLAB system was installed. Although there is possibility to compile such programs and to obtain executable version, but this requires additional treatments.

In order to write and run MATLAB program, follow these steps:

- Start MATLAB environment.
- Select working folder where the program will be saved and set the path to it using the tools available in the *Browse* window.
- Perform the sequence of commands: ***File, New, M-file*** (it will open the program editor window).
- Copy example program shown below into editor.
- Run ***SaveAs*** command to save the program under a unique name (it will be created a text file named **`your_program_name.m`**) in the folder previously selected
- Minimize the editor window
- Type in the MATLAB command line:

  ```
  >> your_program_name
  ```

  and press ENTER key.

Example of MATLAB program:

```
%*********************************************************************
%          First program written in MATLAB programming language
%*********************************************************************
I = imread('Lena_1.tif');    % Read source image
                             % and save in table I
J = histeq(I);               % Calculation of the output image J
                             % using histogram equalization algorithm
subplot(2,2,1);              % Upper left figure
imshow(I);                   %  Display output image
subplot(2,2,2);              % Upper right figure
imhist(I);                   % Display histogram of the source image
subplot(2,2,3);              % Lower left figure
imshow(J);                   %  Display output image
subplot(2,2,4);              % Lower right figure
imhist(J);                   % Display histogram of the output image
imwrite(J, 'Result.jpg')     % Save the output image to a file in
                             % compressed form
```

This is a program that does basically the same thing which had been done in the previous paragraph. Minor changes only apply to a way how images and graphs are displayed. In the place of several graphics windows, for an observer convenience, only one window is used in which two images are shown and two graphs. To fill the window graphic **`subplot(…)`** was function used, which allows you to draw in one graphics window a rectangular array of drawings.

Please refer now to MATLAB Help system to get more info about it.

4. Self-writing program in MATLAB

The task is to write a simple stand-alone program that converts the loaded source image in the negative one. Source image is located in the file **`Lena_2.tif`**. It is a monochrome image of size 256 x 256 points. The degrees of gray image points are coded using integers in the range [0, 225]. The number 0 means black, while the number 255, white.

Image transformation algorithm is given as follows:

- Read the source image file **`Lena_2.tif`** and save it in the array **`X`**
- Display the read source image.
- Determine the input image size
- Declare an array to record output image

- Calculate the gray value for the output image points (negative) **Y** using given relation

$$y(i,j) = 255 - x(i,j)$$

- Display the output and input images, and their histograms as it was presented in the previous example.

The result of the program should be such as shown on Figure 6.

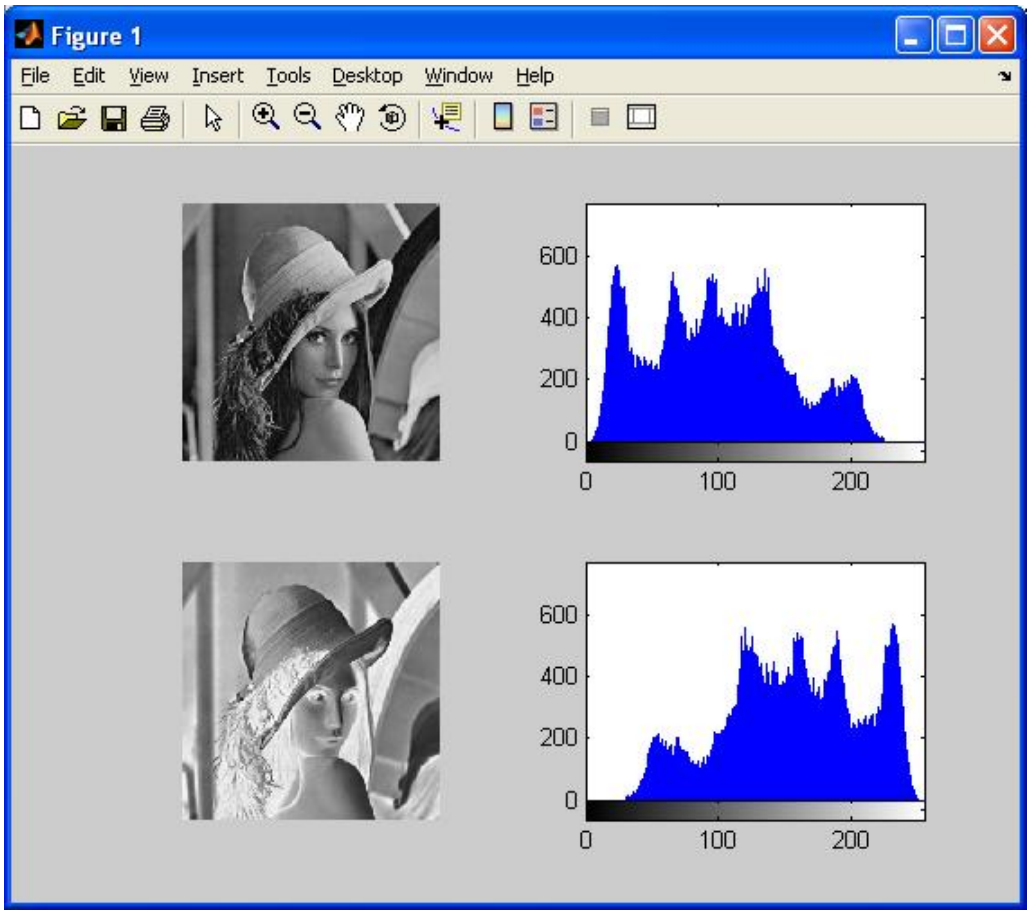

Fig. 6. Positive, negative, and their histograms

Auxiliary recommendations:

- To read the source image use **`imread()`** function (as in the previous example)
- To display the image use **`imshow()`** function.

§ Input image size can be determined using the function **size(…)** (please refer to MATLAB Help). For example, to specify the dimensions of a rectangular array A, you can use the **size(…)**(as follows):

```
[n, m ] = size(A);
```

Calling the function will result in the assignment **n** and **m** variables - the numbers that correspond number of rows and columns of array **A**.

§ An array where the output image is about to be saves MUST be previously declared. Declaration of an array allows you to reserve an appropriate amount of memory for data, which will be stored in an array. Array declaration can be done using the function **zeros()**,as a result an array filled with zeros will be created. Use **zeros()**function as follows:

```
B = zeros(n, m, 'uint8');
```

Given above command creates table **B** filled with zeros. Table **B** has **n** rows and **m** columns. Zeros will be stored in the array as unsigned integers (**uint8**).

§ Calculation of negative gray image pixels can be carried out in two ways

Use a software loop

Template loop is as follows:

```
for i = 1:n

   operations inside
   the loop

end
```

Take advantage of MATLAB array operations

Find out, referring to MATLAB help, how to write abovementioned program without loops.

It should be noted that writing high quality programs in MATLAB imposes minimization of loop usage.

LITERATURE

1. T. Pavlidis, Algorithms for Graphics and Image Processing, Computer Science Press, Rockville, Maryland, 1982, (416 pp). Translated into Russian (1986), Polish (1987), Chinese (1988), and German (1990).