MATLAB – Introduction to IPT – *Image Processing Toolbox*

Introduction

When we are working with images in MATLAB we need to take into account several issues such as:

- read files containing images;
- check the class of the data and convert it to other suitable formats;
- correct visualization of an image;
- convert between different image formats;
- save in a new file the image that results from a processing sequence.

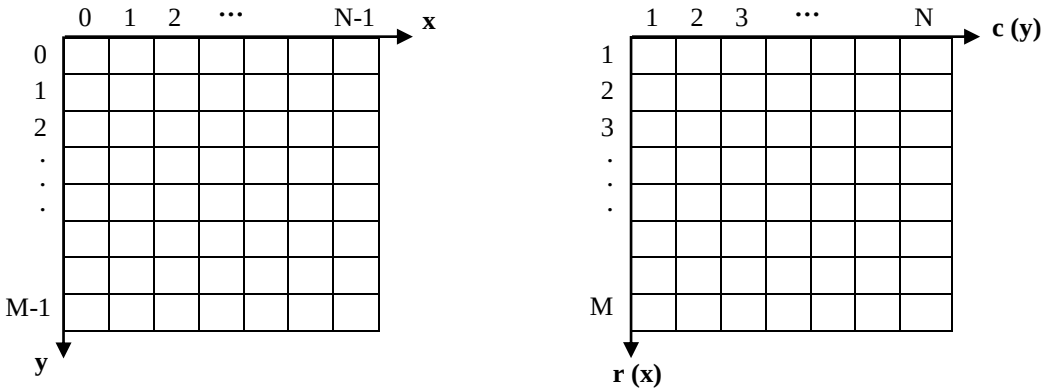
Before starting to use functions that allow performing the actions previously mentioned, it is important to consider some facts related to the internal representation of digital images in IPT as well as with the use of the functions provided by this *toolbox*.

1. Image coordinate system

Consider an image $f(x, y)$ with *M* rows and *N* columns. The values of the coordinates (x, y) are discrete quantities. In most image processing systems, the convention for coordinate representation is illustrated in the next figure (left side), where the image origin is defined to be at the point $(x, y) = (0, 0)$ and the *x coordinate values* are associated with the *columns* and the *y coordinate values* are associated with the *rows*.

The coordinate convention used in IPT is illustrated in the right side of the same figure. First, instead of using (x, y), the toolbox uses the notation (r, c) to indicate rows (*r*) and columns (c). Another difference is that the origin of the coordinate system is at $(r, c) = (1,1)$. The *size* of the image is M×N (M rows and N columns).

Image Processing Toolbox documentation refers to the coordinates (r, c) as *pixel coordinates.* In some circumstances, the toolbox also employs another coordinate convention, called *spatial coordinates*, which follows the convention used in the left side of the next figure where *x* denotes columns and *y* refers to rows.



2. Representation of digital images by matrices

A digital image can be naturally represented by a matrix, as illustrated below:

$$f=\begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix}$$

Matrices in MATLAB are stored in variables with names that must begin with a letter and contain only letters, numbers, and the underscore char "_".

3. Functions provided by IPT

The functions provided by IPT, as all the other MATLAB functions, can be used directly in the *Command Window* or inserted in special files, *M-files*. The call of these functions is integrated into instruction sets of the MATLAB programming language.
Whatever is the method for using an IPT function, it is always necessary to know its name, the set of allowed parameters and their types, as well as the expected results. MATLAB provides very detailed help information for each of its functions. The first use of a given function should always be preceded by a careful consultation of the information provided in MATLAB help (for instance, by placing the text cursor on the function name and pressing the F1 key).

Work proposal

1. **Reading files in image format (*imread* function)** (see details in Help – Image Processing Toolbox – Import, Export, and Conversion – Read and Write Image Data from Files).

   A. Assess which are the image formats supported in MATLAB;

   B. Write an instruction to read the file 'mona.tif' and save its content into the variable Img_1;

   C. Write an instruction to read the file 'mamografia.bmp' and save its content into the variable Img_2;

   D. Verify the creation of the above mentioned variables in the *Workspace* window and evaluate their content.

2. **Reading files in DICOM format (*dicomread*, *dicominfo* functions)** (see details in Help – Image Processing Toolbox – Import, Export, and Conversion – Read and Write Image Data from Dicom Files).

   A. Write an instruction to read the file 'Chest_XRay.dcm' and save its content into the variable Img_dicom.

   B. Write an instruction to read the content of the file 'Chest_XRay.dcm' to the variable Info_dicom.

C. Verify the creation of the above mentioned variables in the Workspace window and evaluate their content.

D. Check the data in the header of the dicom file, namely the number of bits used for pixel representation.

3. Obtaining information relative to a variable (containing an image) (*size* function)

A. In the command window, run the command "help size" (or seek help about the *size* function in MATLAB help for obtaining information on how to use this function).

B. In the command window, run the command "help whos" (or seek help about the function *whos* in MATLAB help for obtaining information on how to use this function).

C. Use the *size* function for determining the size of the variables Img_1, Img_2 and Img_Dicom.

D. Use the functions *whos* for getting additional information about the variables Img_1, Img_2 and Img_Dicom.

4. Displaying images (*imshow* function) (see details in MATLAB Help – Image Processing Toolbox – Display and Exploration – Basic Display).

A. View the contents of the variable (image) Img_1.

```
>> imshow(Img_1)
```

B. View the contents of the variable (image) Img_2.

```
>> imshow(Img_2)
```

C. Use the command *figure* between the two previous instructions and evaluate the obtained results.

```
>> imshow(Img_1), figure, imshow(Img_2)
```

D. Execute the following instructions and comment the results:

```
>> figure, subplot(1,2,1), imshow(Img_1)
>> subplot(1,2,2), imshow(Img_2)
```

E. Write an instruction to display the content of the Img_Dicom variable; comment the obtained results.

F. Execute the following command sequence. Evaluate and comment the obtained results.

```
>> close all;        % close all the open windows
>> figure(1),  imshow(Img_Dicom), figure(2), imshow(Img_Dicom, [])
>> figure(3), imshow(Img_Dicom, [0 255])
>> figure(4), imshow(Img_Dicom, [255 1024])
```

5. Interactive tools for image visualization (*imtool* function) (see details in MATLAB Help – Image Processing Toolbox – Display and Exploration - I Interactive Exploration with the Image Viewer App).

A. View the image Img_1 previously used with the command:

```
>> imtool(Img_1)
```

Use the available tools on the *Tools* menu: Pixel Region tool, Image Information tool and Adjust Contrast tool.

6. **Writing data arrays in files (*imwrite* function)** (see details in MATLAB Help – Image Processing Toolbox – Import, Export, and Conversion – Read and Write Image Data from Files).

   A. Write an instruction to save the content of Img_2 in a file named 'mamografia', in TIFF format.
   B. Write an instruction to save the content of Img_1 in a file named 'mona', in JPEG format, using different values for the quality parameter; Read once again the saved image files and display them.
   C. Get information about the file saved to the disk using the *imfinfo* function.
   D. Write an instruction to save the content of Img_dicom in a file named 'micro_radiografia' but in TIFF format.
   E. Read the file 'micro_radiografia.tif' to a variable and check the class of saved data (do it by inspection of the *Workspace* or by using the command *whos*).

7. **Data classes** (see details in MATLAB Help – Image Processing Toolbox – Import, Export, and Conversion – Image Type Conversion)

   Each variable containing an image is associated with a data class (the class used to represent the pixel values). When we convert between data classes we should be aware of the ranges of values that can be represented using data from a particular class.

   A. The most common data classes to represent images are: *uint8*, *uint16*, *double* and *logical*. Use the information available in MATLAB help to check the meaning of the numerical data representation using each of the data types available.

   B. In MATLAB the numerical operations are performed using *double* values. Thus, the format conversion of *uint8* or *uint16* to *double* format is quite common (and also is advised!). Evaluate the results of the following instructions for converting data classes, checking changes in the Workspace window.

```
>> Img_2_double=double(Img_2);
>> Img_dicom_uint8=uint8(Img_dicom);
>> Img_4_double=Img_2_double.*4;
>> Img_4=uint8(Img_4_double);
```

   C. Display the images Img_2_double, Img_dicom_uint8, Img_4_double and Img_4 using the command *imshow*. Comment the obtained results.

8. **Converting between data classes** (see details in MATLAB Help – Image Processing Toolbox – Import, Export, and Conversion – Image Type Conversion)

   Each variable that contains an image is characterized by a data class and an image type. When we only convert between data classes using *typecast* (see 7), all the values that are outside the range of the new class representation are truncated to the nearest value. The range for data of class *uint8* is [0 .. 255], for *uint16* is [0 .. 65535], for *double* is the range for real numbers [0 .. 1].
   To avoid problems resulting from such operation, the IPT provides functions for converting between classes, which also perform the required scaling in order to convert the input data into values belonging to the range recognized by the IPT for the final image type. These operations depend on both the input and final classes.

   The IPT functions that perform conversion between data classes and image types are: *im2uint8; im2uint16; im2double; im2bw; mat2gray*.

   A. Execute and comment the results of the following command sequence.

```
>> a=[ -0.5 0.5; 0.75 1.5]
>> b=uint8(a)
>> c=im2uint8(a)
```

B. Execute and comment the results of the following command sequence.

```
>> d=uint8([ 25  50; 128 200])
>> e=double(d)
>> f=im2double(d)
>> g=mat2gray(d)
>> g0=mat2gray(d, [0 255])
```

C. Compare the results of the following command sequence with those obtained in B.

```
>> d = [ 25  50; 128  200]
>> e1=double(d)
>> f1=im2double(d)
>> g1=mat2gray(d)
```

D. Execute and comment the results of the following command sequence.

```
>> h=uint8( [ 25  50; 128 200])
>> i=im2bw(h)
>> j=im2bw(h, 0.7)
```

E. Execute and comment the results of the following command sequence.

```
>> close all;
>> figure(1); imshow(Img_1);
>> Img_1_d= double(Img_1);
>> figure(2); imshow(Img_1_d);
>> Img_1_i2d= im2double(Img_1);
>> figure(3); imshow(Img_1_i2d);
>> Img_1_gray=mat2gray(Img_1)
>> figure(4); imshow(Img_1_gray);
>> Img_1_bw= im2bw(Img_1)
>> figure(5); imshow(Img_1_bw);
```

9. Basic techniques for matrix data manipulation (*Array indexing*).

Execute and comment the results of the following command sequence:

```
>> figure, imshow(Img_1)
>> Img_1_flipped = Img_1(end:-1:1, :);
>> figure, imshow(Img_1_flipped)
>> Img_1_crop = Img_1(257:768, 257:768);
>> figure, imshow(Img_1_crop)
>> Img_1_subsampled = Img_1(1:2:end, 1:2:end);
>> figure, imshow(Img_1_subsampled)
>> figure, plot(Img_1(512, :))
```

A. Get the maximum and the minimum in a matrix (functions *max* and *min*). Execute the following command sequence:

```
>> A=[10 50 3; 70 2 125]
>> V=A(:)
>> maximum_A=max(A)
>> maximum=max(A(:))
>> minimum_A=min(A)
>> minimum=min(A(:))
```

10. Image types (see details in MatLab Help – Image Processing Toolbox – Import, Export, and Conversion – Image Type Conversion)

Each variable containing an image is associated with an image type. The image types supported by the IPT are: *grayscale* or *intensity*, *binary*, *indexed* and *truecolor* or *RGB* (color images).

A. Use MATLAB help for assessing the meaning of each image type.

B. Verify the range of values recognized by the IPT for each image type.

11. Conversion between image types (see details in MATLAB Help – Image Processing Toolbox – Import, Export, and Conversion – Image Type Conversion)

Conversion of a color image (RGB format) to a binary image (*im2bw* function) – see topic 8.E.

Conversion of a color image (RGB format) to the corresponding monochrome image (*rgb2gray* function)

A. Run and comment the results of the following command sequence:

```
>> img_rgb=imread('retinografia.tif');
>> img_gray=rgb2gray(img_rgb);
>> imshow(img_rgb); figure, imshow(img_gray);
```

B. Decomposition of an RGB image into its 3 components.
Run and comment the results of the following command sequence:

```
>> red=img_rgb(:, :, 1); figure, imshow(red), title('RED');
>> green=img_rgb(:, :, 2); figure, imshow(green), title('GREEN');
>> blue=img_rgb(:, :, 3); figure, imshow(blue), title('BLUE');
```

C. Reconstruction of an RGB image from 3 individual color components (*cat*).

Run and comment the results of the following command sequence.

```
>> red_modificada=imadjust(red, [0.5 1]); figure, imshow(red_modificada);
>> img_rgb_modificada=cat(3,red_modificada, green,blue);
>> figure, imshow(img_rgb_modificada);
```

[HOMEWORK (not to be delivered)] Create a .m script that finds all .png images on a given folder, reads and coverts each image to double and stores the result on a cell of a cell array.