Tutorial 0:  
Introduction to Matlab Image Processing Toolbox 
2 August 2013

Lecturer: Dr. Thierry Peynot 
Tutor: Christopher Brunner 

This short tutorial aims at introducing some basic fundamentals of the Matlab Image Processing Toolbox. In particular, useful information and functions will be presented. You are invited to test all these functions on different images to get a good understanding of their purpose, syntax and effect on images or arrays. Throughout this document, you will also be asked to realize some additional tests or answer specific questions using the possibilities of the toolbox mentioned here.

1 Digital Image Representation

1.1 Coordinate Conventions

Consider a sampled image \( f(x, y) \), with \( M \) rows and \( N \) columns (size \( M \times N \)). In many image processing references, as in the lectures notes of this course, the image origin is set at \((x, y) = (0, 0)\), which means that \( x \) ranges from 0 to \( M - 1 \), and \( y \) from 0 to \( N - 1 \), in integer increments. The coordinate convention used in the toolbox is slightly different. The Image Processing Toolbox, uses the notation \((r, c)\) to indicate rows and columns, instead of \((x, y)\). Besides, the origin of the coordinate system is at \((r, c) = (0, 0)\) (since arrays in Matlab are always indexed starting at 1), which makes \( r \) range from 1 to \( M \) and \( c \) from 1 to \( N \), in integer increments as well.

1.2 Images as Matrices

Given the coordinate convention of the toolbox, a digital image in Matlab can be represented as the following matrix:

\[
\begin{bmatrix}
 f(1, 1) & f(1, 2) & \ldots & f(1, N) \\
 f(2, 1) & f(2, 2) & \ldots & f(2, N) \\
 \vdots & \vdots & \ddots & \vdots \\
 f(M, 1) & f(M, 2) & \ldots & f(M, N)
\end{bmatrix}
\]

2 Reading Images

Images are read into the Matlab environment using the function \texttt{imread}:

\texttt{imread('filename')}
filename being a string containing the complete file name of the image file (including extension). For example:

\[
    f = imread('myimage.jpg');
\]

reads the JPEG image myimage into an array \( f \). You can use the function \texttt{size} to see the row and columns dimensions of the image (the array):

\[
    \text{size}(f)
\]

or:

\[
    [M,N] = \text{size}(f)
\]

The \texttt{whos} function returns information about the array, in particular its size and \textit{data class} (see section 5):

\[
    \text{whos } f
\]

NB: Check the image formats supported by the function \texttt{imread}.

3 Displaying Images

Images are displayed in Matlab using the function \texttt{imshow}:

\[
    \text{imshow}(f)
\]

where \( f \) is an image array. You can specify a range of intensities by typing:

\[
    \text{imshow}(f, [\text{low} \ \text{high}])
\]

In that case, all values in the image which are less than or equal to \texttt{low} are displayed as black, and all values greater than or equal to \texttt{high} are displayed as white. Values in between are displayed as intermediate intensity values using the default number of levels.

You can also “expand” the dynamic range of an image automatically by:

\[
    \text{imshow}(f, [\ ])
\]

which sets the previous variable \texttt{low} to the minimum value of the array \( f \) and \texttt{high} to its maximum. This is useful for displaying images with a low dynamic range.

Test \texttt{imshow} on such an image.

Check the other options of \texttt{imshow} and try to use some of them.

The function \texttt{impixelinfo} (pixval in older versions of Matlab) enables to display the intensity values (or RGB values for a colour image) of individual pixels interactively, thanks to a cursor. Note also the function \texttt{imdistline} which shows Euclidean distances on the image (previously provided by the function \texttt{pixval} as well).

Note: \texttt{image(C)} displays a matrix \( C \) as an image (see also the function \texttt{imagesc}). What are the main differences between \texttt{image} (or \texttt{imagesc}) and \texttt{imshow}?
4 Writing Images

Images are written to disk using the function `imwrite`:

\[
\text{imwrite}(f, 'filename')
\]

where `filename` includes a recognized file format extension (check the accepted formats). You can also specify the desired format explicitly using a separate third argument:

\[
\text{imwrite}(f, 'basename', 'extension')
\]

The `imwrite` function can have other parameters, depending on the file format specified. For example, to save an image in the JPEG format, you can specify a `quality` parameter which will set the level of compression achieved:

\[
\text{imwrite}(f, 'filename.jpg', 'quality', q)
\]

where `q` is an integer between 0 and 100 (the lower the number, the higher the degradation due to JPEG compression, but the smaller the size of the file).

Realize some tests of quality on different kind of images by varying the `q` parameter.

The function `imfinfo` provides some details on the image:

\[
\text{imfinfo} \text{ filename}
\]

The information provided can be captured into a so-called structure variable. Let us name such a variable K. If we use:

\[
K = \text{imfinfo}( 'filename')
\]

to store the information, we can have access to specific fields. For example, the image width and height are stored in the fields `K.Width` and `K.Height`.

Use this `imfinfo` function to compute the compression ratio (number of bytes of the original image divided by number of bytes of the compressed image) of a couple of images that were saved in JPEG format previously.

NB: the function `imageinfo` can provide the same information than `imfinfo`.

Compare those two functions.

Check the other possible parameters of `imwrite`.

5 Data Classes

The following table lists the various data classes supported by Matlab and the Image Processing Toolbox for representing pixel values. The first eight are referred to as numeric data classes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>Double-precision, floating-point numbers, range (-10^{308}) to (10^{308}) (8 bytes per element)</td>
</tr>
<tr>
<td>uint8</td>
<td>Unsigned 8-bit integers in the range ([0,255]) (1 byte per element)</td>
</tr>
<tr>
<td>uint16</td>
<td>Unsigned 16-bit integers in the range ([0,65535]) (2 bytes per element)</td>
</tr>
<tr>
<td>uint32</td>
<td>Unsigned 32-bit integers in the range ([0,4294967295]) (4 bytes per element)</td>
</tr>
<tr>
<td>int8</td>
<td>Signed 8-bit integers in the range ([-128,127]) (1 byte per element)</td>
</tr>
<tr>
<td>int16</td>
<td>Signed 16-bit integers in the range ([-32768,32767]) (2 bytes per element)</td>
</tr>
<tr>
<td>int32</td>
<td>Signed 32-bit integers in the range ([-2147483648,2147483647]) (4 bytes per element)</td>
</tr>
<tr>
<td>single</td>
<td>Single-precision, floating-point numbers, range (-10^{38}) to (10^{38}) (4 bytes per element)</td>
</tr>
<tr>
<td>char</td>
<td>Characters (2 bytes per element)</td>
</tr>
<tr>
<td>logical</td>
<td>Values are 0 or 1 (1 byte per element)</td>
</tr>
</tbody>
</table>
6 Image Types

The toolbox supports four types of images:

- Intensity image: data matrix whose values have been scaled to represent intensities
- Binary image: \textit{logical} array of 0s and 1s
- Indexed image
- RGB image

NB: an array of 0s and 1s whose values are of any other data class than \textit{logical} is \textit{not} considered as a binary image in Matlab. A conversion function can be used in this case to ensure to obtain a logical array \( B \) from the numerical array \( A \):

\[
B = \text{logical}(A)
\]

NB: if \( A \) contains elements other than 0s and 1s, this will convert all nonzero quantities to logical 1s.

To test if an array is logical, one can use the \texttt{islogical} function.

Convert a regular gray-scale image into a binary image and observe the image obtained. Modify the dynamic range of the original image and convert the output image to a binary image. Compare the two binary images obtained.

7 Converting between Data Classes and Image Types

7.1 Converting between Data Classes

To convert between data classes one can do:

\[
B = \text{data\_class\_name}(A)
\]

where \texttt{data\_class\_name} is one of the names of the available data classes (cf. table).

NB: when converting between data classes, it is important to keep in mind the value ranges for each data class.

7.2 Converting between Image Classes and Types

The toolbox provides specific functions that perform the scaling necessary to convert between images classes and types (see table below).

<table>
<thead>
<tr>
<th>Name</th>
<th>Converts Inputs to:</th>
<th>Valid Input Data Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>im2uint8</td>
<td>uint8</td>
<td>logical,uint8,uint16,double</td>
</tr>
<tr>
<td>im2uint16</td>
<td>uint16</td>
<td>logical,uint8,uint16,double</td>
</tr>
<tr>
<td>mat2gray</td>
<td>double (in range ([0,1]))</td>
<td>double</td>
</tr>
<tr>
<td>im2double</td>
<td>double</td>
<td>logical,uint8,uint16,double</td>
</tr>
<tr>
<td>im2bw</td>
<td>logical</td>
<td>uint8,uint16,double</td>
</tr>
</tbody>
</table>
For example: create a simple $4 \times 4$ image $f$ of class double, with non integer values of intensity, and convert it to an image of class uint8 using the function im2uint8. This function will detect the data class of the input and performs all the necessary scaling. Observe the operations made by Matlab on the original intensity values.

Converting an arbitrary array $A$ of class double to an array of class double scaled to the range $[0, 1]$ can be made using the function mat2gray:

$$g = \text{mat2gray}(A)$$

where $g$ has values in the range $0$ (black) to $1$ (white). We can also specified a minimum and maximum intensity values in $A$ that will become respectively black and white in $g$:

$$g = \text{mat2gray}(A, [\text{Amin}, \text{Amax}])$$

Make several image class conversions on simple $4 \times 4$ arrays and observe the results obtained. Based on the range specified for each class in the table, interpret the mathematic operations realized by each function.

8 Array Indexing

Matlab supports a number of powerful indexing schemes that simplify array manipulation and improve the efficiency of programs. They are not specific to the toolbox, and will obviously be very important for programming efficient image processing applications in the future tutorials and assignments. Please make sure that you know these manipulations and syntaxes. If you do not, find out the information you need in the Matlab help or ask your tutor.

Using only array indexing, realize the following operations:

- Flip vertically an image and display the result.
- Subsample an image (e.g. take one pixel over two).
- Crop an image (e.g. extract the central part and display it).

9 Some Important Standard Arrays

The following functions can be useful to generate simple image arrays to try out ideas, test the syntax of functions during development, or test some image processing algorithms.

- zeros($M,N$) generates an $M \times N$ matrix of 0s of class double
- ones($M,N$) generates an $M \times N$ matrix of 1s of class double
- true($M,N$) generates an $M \times N$ logical matrix of 1s
- false($M,N$) generates an $M \times N$ logical matrix of 0s
- magic($M$) generates an $M \times M$ “magic square”: a square array in which the sum along any row, column or main diagonal, is the same. NB: numbers are integers.
• \texttt{rand(M,N)} generates an \( M \times N \) matrix whose entries are uniformly distributed random numbers in the interval \([0,1]\)

• \texttt{randn(M,N)} generates an \( M \times N \) matrix whose entries are normally distributed (i.e. Gaussian) random numbers with mean 0 and variance 1.

NB: if only one argument is included in any of the previous functions, the result will be a square array.

Use these function to create some test arrays. Manipulate them with the previous conversion functions.

References