Image processing basics using MATLAB

M. Thaler, H. Hochreutener, February 2008, ©ZHAW

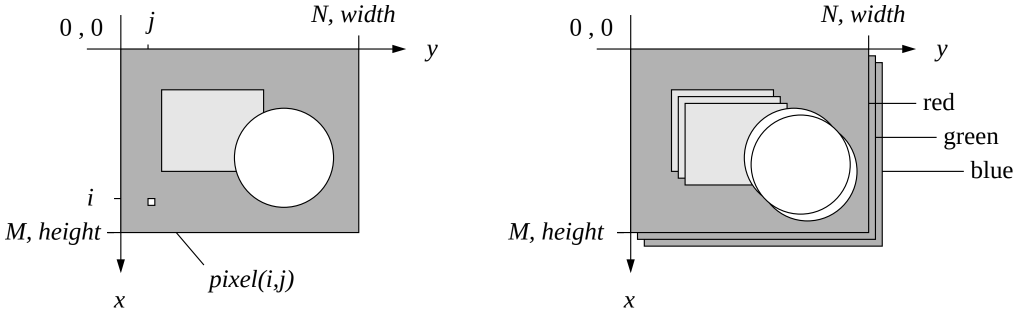
1 Introduction

In the following we give a short overview on a very limited set of basic image processing functions provided by MATLAB. The function descriptions only cover the basic usage, more detailed information can be found in the manual pages by typing `doc` *`functionName`* at the MATLAB command line.
For an overview of the image processing toolbox type: `doc images`
For an overview of the image acquisition toolbox type: `doc imaq`

MATLAB provides with it's image processing toolbox many powerful and very efficient image processing functions (see function list of the image processing toolbox). With this it is very simple to implement complex image processing applications, especially for fast prototyping. On the backside, a lot of understanding how image processing works is hidden within black boxes and temps to make things more complicate than they really are.

2 Image representation

In MATLAB a binary and gray-scale image is represented by one 2-dimensional array, whereas a color image are represented by a 3-dimensional array (one 2-dimensional array for each of the color planes or color channels `red`, `green` and `blue`):



The origin of the image is in the upper left and the size of the image is defined by the parameter `width` (number of columns of the array) and `height` (number of rows of the array). Note that the x- and y-coordinates are chosen such that the z-axis points to the front.

A single point within the image is called *pixel*. A gray-scale or binary pixel consists of one data value, a color pixel consists of 3 data values (each for one of the color channels). The most common data types of the individual pixels are:

`uint8`	unsigned integer: data range $0..255$
`double`	double precision float: data range 0.0 ... 1.0

Binary images have pixel values of 0's and 1's resp. 0.0 and 1.0. In the case of `uint8` images, the *logical* flag must be turned on, to be recognized as binary image (for details see below).

Be careful with data types in MATLAB. Many of the predefined functions, e.g. `imadd(img1, img2)` which adds two images, just truncates data values to 255 on `uint8`-arrays ... make sure if that is what you want.

Hints:
To avoid problems with data types, especially when working with predefined image processing functions, it is advisory to work with type `double` and make sure that data is scaled to the range 0 ... 1.0.

3 Basic MATLAB functions

3.1 MATLAB manual

`doc` *functionname*	displays the manual for the MATLAB function *functionname*
`doc images`	the manual for the image processing toolbox
`doc imaq`	the manual for the image acquisition toolbox

3.2 Image information

`imfinfo('foo.ext')`	displays information on image format etc. of the file `foo.ext`
`imformats`	displays an overview of all MATLAB image formats
`whos img`	displays information about the array `img`: size, data type, etc.

3.3 Reading, writing and displaying images

`myImg = imread('foo.ext')`	reads file `foo.ext` into array `myImg`, image format is determined by the file extension (jpg, tiff, tif, gif, bmp, png, ...)
`imwrite(anImg, 'foo.ext')`	writes the image `anImg` to the file `foo.ext`, where the image format is chosen according to the extension `ext`. Valid extensions are: `tif`, `tiff`, `jpg`, `jpeg`, `gif`, `png`
`imshow(myImg)`	displays the image `myImg` as gray-scale, binary or color image depending on the data type of `myImg`
`imshow(myImg,[])`	displays the image `myImg` as gray-scale, binary or color image depending on the data type of `myImg` and scales the image properly
`figure(n)`	opens a new window with number `n`, the next call to `imshow()` displays the image within this window

3.4 Basic image processing functions

`islogical(binImg)`	checks whether array `binImg` has the logical flag set or not (returns value 1 or 0)
`img = uint8(zeros(512,1024))`	creates a *black* image with `width` 1024 and `height` 512 of type `uint8`
`img = uint8(255*ones(512,1024))`	creates a *white* image with `width` 1024 and `height` 512 of type `uint8`
`img = double(zeros(512,1024))`	creates a *black* image with `width` 1024 and `height` 512 of type `double`
`img = double(ones(512,1024))`	creates a *white* image with `width` 1024 and `height` 512 of type `double`
`[height width d] = size(myImg)`	retrieves height and width and stores the values in variables `height` and `width`, `d` ist set to the array dimension.
`red   = myImg(:,:,1)`	stores the red component of `myImg` (rgb-image) in array `red`
`green = myImg(:,:,2)`	stores the red component of `myImg` in array `green`
`blue  = myImg(:,:,3)`	stores the red component of `myImg` in array `blue`
`mx = max(myImg(:)))`	computes the maximum value of an 2-d array
`mi = min(myImg(:)))`	computes the minimum value of an 2-d array
`img = double(myImg)/255`	converts an `uint8` array to a `double` array (no scaling)
`img = double(myImg)/double(mx)`	converts `uint8` to `double` and scales maximum to 1.0

`img = uint8(anImg*255)`	converts a `double` array to an `unit8` array and rescales the array to the proper data range
`bw  = logical(binImg)`	sets the logical flag on the `unit8` array `binImg` (data values 0 and 1), array `bw` is then interpreted as a black and white image and logical operations can be applied
`gray = (+bw)*255`	turns the logical flag off and rescales the array `bw` to be displayed as `unit8` array

3.5 Examples

3.5.1 Scaling images

The following two statements scale a double type image to the range $0.0 \ldots 1.0$. This is important, when the image contains negative pixel values, as e.g. after applying edge detection algorithms.

```
f = f - min(f(:));
f = f / max(f(:));
```

3.5.2 Color planes

The green and red color plane of image `rgbimage.jpg` are swapped:

```
f   = imread('rgbimage.jpg');
red = f(:,:,1);
g(:,:,1) = f(:,:,2);
g(:,:,2) = red;
g(:,:,3) = f(:,:,3);
imshow(g);
```

3.5.3 Individual pixel processing

The intensity of the red color channel of `rgbImage.jpg` is divided by 2. Note that the resulting image `rImg` is allocated prior to iterating through the pixels which makes the computation much faster.

```
f = imread('rgbImage.jpg');
[M N d] = size(f);
g = unint8(zeros(M,N,3));
for x = 1:M
    for y = 1:N
        g(x,y,1) = f(x,y,1) / 2;
        g(x,y,2) = f(x,y,2);
        g(x,y,3) = f(x,y,3);
    end;
end;
imshow(g);
```

Using the MATLAB array notation, this may be written as:

```
f = imread('rgbImage.jpg');
g = f;
g(:,:,1) = g(:,:,1) / 2;
imshow(g);
```

The image color image rgbImage.jpg is converted to a grayscale image, by simply computing the mean of the three color channels (one possible method) and then stored in file grayImage.jpg. Note that the quality of the resulting image is set to 100 (no data loss):

```
f = imread('rgbImage.jpg');
[M N d] = size(f);
g = unint8(zeros(M,N));
for x = 1:M
    for y = 1:N
        g(x,y) = (f(x,y,1) + f(x,y,2) + f(x,y,3)) / 3;
        % The line above doesn't work.
        % Overflow occurs, while processing uint8, because
        % the value range in the intermediate results are limited to 255
        g(x,y) = (f(x,y,1)/3 + f(x,y,2)/3 + f(x,y,3)/3);
    end;
end;
imshow(g);
imwrite(g, 'grayImage.jpg', 'Quality', 100);
```

Using the MATLAB array notation, this may be written as:

```
f = imread('rgbImage.jpg');
g = uint8(mean(f,3));
imshow(g);
imwrite(g, 'grayImage.jpg', 'Quality', 100);
```

The image grayImage.jpg is slightly blurred by computing the mean of a 3x3 pixel environment and by setting the resulting center pixel to this mean value:

```
f = imread('grayImage.jpg');
[M N d] = size(f);
g = unint8(zeros(M,N));
for x = 2:M-1
    for y = 2:N-1
        sum = f(x-1,j-1) + f(x-1,j) + f(x-1,y+1);
        sum = sum + f(x,y-1) + f(x,y) + f(x,y+1);
        sum = sum + f(x+1,y-1) + f(x+1,y) f(x+1,y+1);
        rImg(x,y) = unint8(sum/9);
    end;
end;
imshow(g);
```

The same functionality could be achieved, by using MATLAB's powerful image processing functions and in addition avoids the *boundary* problem:

```
f = imread('grayImage.jpg');
h = fspecial('average',3)
g = imfilter(f, h, 'replicate');
imshow(g);
```