L2. Image processing in MATLAB

1. Introduction

MATLAB environment offers an easy way to prototype applications that are based on complex mathematical computations. This annex presents some basic image processing operations that can run in MATLAB. MATLAB has the great advantage that is a matrix oriented environment. All variables in MATLAB are actually arrays. Scalar values are 1x1 matrices. Common operations on matrices include addition, subtraction, logical operations, multiplication, division, matrix transpose, determinant, inverse matrix, eigenvalues and eigenvectors, etc. A very important and powerful aspect in MATLAB is that most operations can be performed either element by element or directly on the whole matrix. For example, multiplication can be applied in the mathematical sense of matrix multiplication (a non-commutative operation) and also in the sense of scalar multiplication, where the multiplication is performed element by element. The latter option is particularly useful in the image processing field where most operations are performed at pixel or pixel neighbor’s level.

A very important aspect is that, unlike C language, indexing starts at value 1 and matrix elements are organized on lines and then on columns. The element (i, j) of the matrix A, the element located at row *i* and column *j*.

In order to use the functions defined in the Image Processing Toolbox in Octave you should run the following command first:

```
pkg load image
```

2. Read an image

Imread MATLAB function reads images in different formats, the result being a matrix. The function call can be performed as following:

```
I = imread (‘nume.bmp’);
```

or

```
I = imread (‘nume.jpg’,’jpg’);
```

The first string can contain also the image path.

3. Display an image

Imshow function displays an image into a graphic handler. Function call:

```
imshow (I);
```

In case when an image is already displayed, then, by calling the ***imshow*** function, the image will be displayed in the opened figure. In case the user wants to display in a new figure, the function call must be preceded by the call of ***figure*** function:

```
figure, imshow (I);
```

To close all the opened figures, user must call function the ***close all*** function:

```
close all
```

4. Conversion from RGB to Grayscale

Most common image processing operations require to reduce the quantity of information in image. The representation that preserve the relevant image information while reducing the complexity is a grayscale representation, having values between 0 and 255. A color image will be converted to grayscale by calling the ***rgb2gray*** function:

```
Ig = rgb2gray (Ic);
```

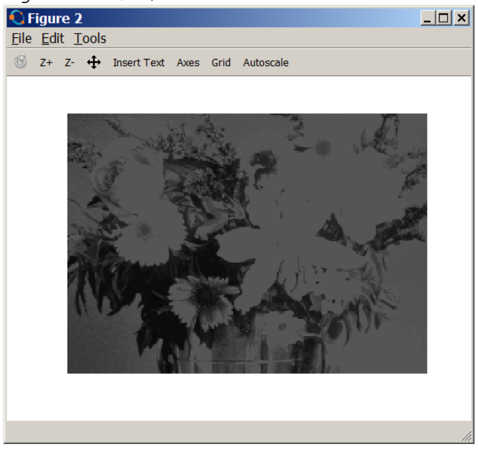
Reading each channel independently:

```
R=Ic(:,:,1);
G=Ic(:,:,2);
B=Ic(:,:,3);
```

Alternative ways to convert the color image to grayscale:

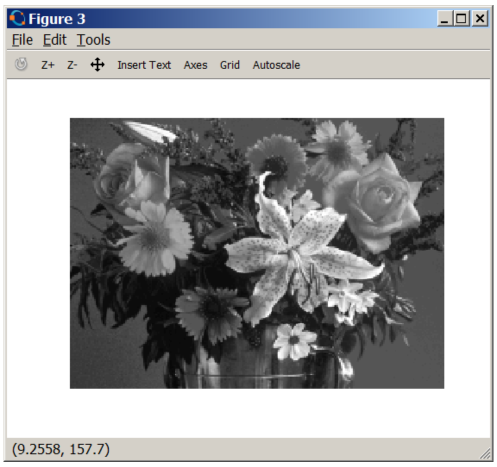
Wrong solution (R,G,B are of type uint 8, and the sum will be put in an uint8 matrix causing capacity overflow -> the obtained image is with artefacts)

```
It = R + G + B;
Ig1 = It/3;
```



Correct solution (cast to uint 16 in order to avoid overflow):

```
It = uint16(R) + uint16(G) + uint16(B);
Ig1=uint8(It/3);
```



5. Thresholding

Thresholding process is used to convert a greyscale image into a black and white image (also referred as binary image), where the pixels intensities are reduced to only two values: 0 and 1. The thresholding process require the definition of a threshold *P* with value between 0 and 255. Then each pixels is compared with the threshold and the one that are greater will be assigned to 1, while the lowest ones will be assigned to 0.

```
P = 100;
IB = (Ig > P);
```

An easy way to define the threshold (but not always the optimal way) is to choose the P value to be equal to the mean intensity value of the image.

```
P = mean (Ig(:));
IB = (Ig>P);
```

6. Saving image on disk

To write an image on disk, Matlab use the function ***imwrite***:

```
imwrite (I, 'file_name', 'type');
```

The parameters of this function are:
I – the image to be saved;
file_name – the name of the image including the path on disk where the user wants to save the image and the image extension;
type – specifies the type of the image (e.g. 'bmp', 'jpg', etc.).

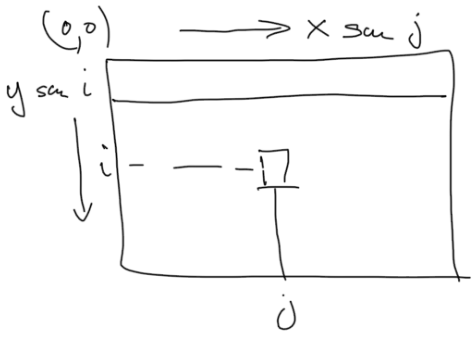
If I is the result of thresholding, it will contain only the values 0 and 1. In this case, the image should be multiplied with 255 before saving it, so the 1 value to correspond to white. Also, the type of output images provided by several processing is double, and sometimes it is required a conversion to byte before saving.

```
imwrite (255*uint8(IB), 'binary.bmp', 'bmp');
```

7. Image negative

```
I = imread('eight.bmp');
[height,width]=size(I);

% ver. 1 by sistematically scan the image pixels (slow)
tic;
for i=1:height,
  for j=1:width,
      Id(i,j) = 255-I(i,j);
  endfor
endfor
toc % displays the processing time
```



```
% ver.2 by performing matrix operations (fast)
tic;
Id1=255-I;
toc % displays the processing time

figure, imshow(I);
figure, imshow(Id);
figure, imshow(Id1);
```

Implement the negative of a color image using the acquired skills. Display the result.

8. Grayscale image histogram, mean and standard deviation

Implement the examples for Lecture 2 in a script saved as a *.m. file. Run the script.

Note: in order to use function 'medfilt1' you should load the 'signal' package: *pkg load signal*.

Computing the mean and the standard deviation of the image intensities can be done bay calling 2 equivalent functions:

```
% compute the mean
       medie = mean2(I)
%or:
       medie_= mean(I(:))
%compute the standard deviation
       std_dev = std2(I)
%or
       std_dev_ = std(I(:))
```

9. Brightness and contrast adjustment, histogram equalization

The MATLAB function ***imadjust*** automatically computes the optimal mean contrast and brightness for an image that is specified as a parameter.

```
IA = imadjust( Ig )
```

The image brightness can be changed by simply adding a constant value to all pixels in the image.

```
IBright = Ig + 50;  % brighter
IDark = Ig – 50;    % darker
```

Implement a function that performs the contrast adjustment using the formula given at the lecture (histogram schrnk/stretch.

Implement the histogram equalization of an image as described in Lecture2. Compute and show the histogram before and after the equalization.

For each example compute and show the histogram before and after the image adjustment !!

10. Image filtering

You can perform the filtering using several functions: *imfilter*, *filter2*, *conv2*.

First you have to generate the filter's kernel. For that you can use the *fspecial* function in order to generate some predefined filters:

Examples:

Low pass filters

```
hA = fspecial('average',size) % size = 3, 5, 7 ..
hG = fspecial('gaussian',size,sigma) % size = 3, 5, 7 .. ; sigma = size/6
```

High pass filters (edege detection)

```
hL = fspecial('laplacian',0.2)
hLoG = fspecial('log',size,sigma) % same as for the gaussian
hPy = fspecial('prewitt') % Prewitt kernel for computing the vertical component of the gradient
hPx=hPy' % transpose to compute the kernel for the horizontal gradient computation
hSy = fspecial('sobel') % Sobel kernel for computing the vertical component of the gradient
hSx=hSy' % transpose to compute the kernel for the horizontal gradient cumputation
```

You can also generate/use any custom filter by specifying explicitly its elements. For example a 3x3 average (mean) filter kernel can be specified as:

```
hA = [[1,1,1];[1,1,1];[1,1,1]] / 9
```

Perform the filtering using the following function: `Out = imfilter(I, h)` for the above mentioned filters. Show the sourse and destination images.

Perform noise reduction using the Gaussian filter for images corrupted by Gaussian noise and median filter for Salt&Paper noise using the median filter:

```
medfilt2 (img, [size1 size2])  % ex: for a 3x3 median filter size1 = size2 =
3
```

For edge detection you can use the *edge* function (https://www.mathworks.com/help/images/ref/edge.html). For example, to apply the Canny edge detection use the following syntax:

```
BW1 = edge(I,'Canny');
```

To do: Implement scripts (*.m file) that perform the following tasks:

1. **Read a grayscale image (ex. 'eight.bmp') and perform the following operations:**
   A. Image negative (using the 2 variations)
   B. Thresholding
   C. Display the results of each processing in separate figures
2. **Read a color (RGB24) image and convert it to a grayscale image.**
3. **Read a grayscale image (ex. 'baloons.bmp') and perform brightness and contrast adjustments:**
   D. Compute the mean and the standard deviation
   E. Compute and display the image histogram (see lectures Z2)
   F. Perform brightness adjustment (by adding a constant). Recompute the histogram and display its new shape
   G. Perform contrast adjustment (see lectures Z2). Recompute the histogram and display its new shape
   H. Perform histogram equalization using `histeq` function (see lectures Z2). Recompute the histogram and display its new shape
4. **Implement filtering operations using the `Out = imfilter(I, h)` function**
   A. Implement the low pass filters and display the Input and Output images in separate figures. Display also the filter/kernel values
   B. Implement the high pass filters and display the Input and Output images in separate figures. Display also the filter/kernel values
   C. Perform noise reduction for images corrupted with gaussian noise (*baloons_Gauss.bmp*) and with Salt&Pepper noise (*baloons_Salt&Pepper.bmp*). Display the images before and after filtering.
5. **Implement the Canny edge detection method**