Exercise Chapter 2 – Binarization

This exercise aims to show you different processes to binarize an image (with Matlab). Load the ISABE_LUMI.BMP image in your file and update the path browser.

1 – Read and display the image (functions ***imread*** and ***image***). Display the image histogram (function ***imhist***) to chose a binary threshold.

2 – With the function ***im2bw*** (*image processing toolbox*) you can convert this grayscale image to binary by thresholding. Use Matlab help and create and display the binary image of *ISABE_LUMI* with the function ***im2bw***.

3 – We want now to create the binary image of *ISABE_LUMI* without using the function ***im2bw***. Write a script file for thresholding and binarization. You can use classical Matlab statements (if, for, etc.) but as Matlab is an interpreter language, it is much more efficient to work with vectorial data (the function ***find*** can be useful).

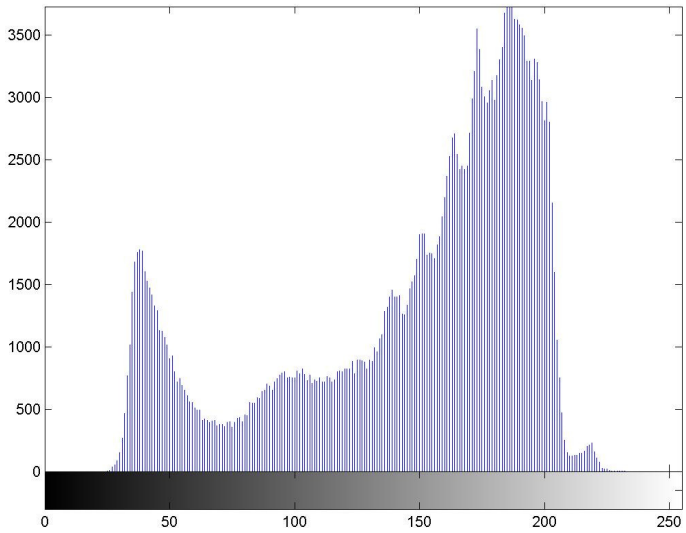
Exercise solution: Binarization

Binarization is based on a rough thresholding. The output binary image has values of 0 (black) for all pixels in the input image with luminance less than the threshold level and 1 (white) for all other pixels. The output image has only two intensity levels (value 0 and 1). The output image is a binary image.

1 – After loading the ISABE_LUMI image, enter the following script:

```
Im=imread('ISABE_LUMI.BMP');
r=0:1/255:1;
g=r;
b=r;
image(Im);
colormap([r' g' b']);
figure
imhist(Im);
```

and we will have the following histogram:

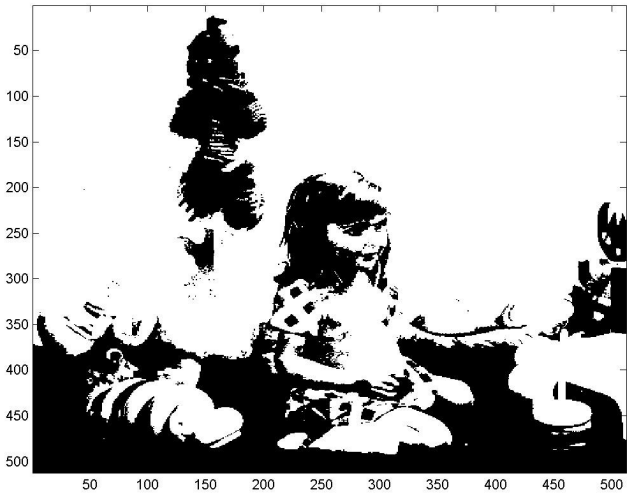


The threshold can be chosen according to different ways. In the case of ISABE_LUMI image, the histogram presents two peaks around the gray levels 75 and 190. We can then for example choose the median gray level between these two peaks as threshold. Nevertheless there are many other ways to choose the threshold: intensity mean value of the histogram, median value of the full gray level range [0, 255])…

2 – We choose the median value 128 of the full range [0, 255] as the threshold. This value must be normalized on the range [0, 1] to be used with the function ***im2bw***. The binary image of ISABE_LUMI is then obtained by the statements:

```
Im=imread('ISABE_LUMI.BMP');
ImBinary=im2bw(Im,128/255) ;
imshow(ImBinary);
```

We display the following binary image:



3 – We present 3 other script files to binarize this image (threshold=128).

a) the first script use " for " loops. The goal is to compare each pixel intensity value of the grayscale image to the threshold:

- if the pixel value (i, j) of the original image is lower than threshold, pixel (i, j) of binary image is black (value 0) ;

- if the pixel value (i, j) of the original image is higher than threshold, pixel (i, j) of binary image is white (value 1).

Here is the Matlab script file to perform this process:

```
threshold = 128;
Im = imread ('ISABE_LUMI.BMP');
n= size(Im,1);    % Number of rows of image
m= size(Im,2);    % Number of columns of image
      for i=1:n         % each row
            for j=1:m   % each column
                        if Im(i,j) < threshold
                              ImBinary (i,j) = 0;
                        else
                              ImBinary (i,j) = 1;
                        end
            end
      end
imshow(ImBinary);
```

b) The second script uses the Matlab function ***find*** (to avoid using “for” loops and to reduce run time). This function determines the indices of array elements that meet a given logical condition.

```
Im=imread('ISABE_LUMI.BMP');
threshold = 128 ;
ImBinary=Im;
ImBinary (find(ImBinary < threshold))=0;
ImBinary (find(ImBinary >= threshold))=1;
image(ImBinary)
% LUT to display in black and white
r=[0 1];
g=r;
b=r;
map=[r' g' b'];
colormap(map);
```

c) The third script also shows the effectiveness of vectorized code with Matlab (you can often speed up the execution of MATLAB code by replacing for and while loops with vectorized code):

```
Im = imread ('ISABE_LUMI.BMP');
threshold = 128 ;
ImBinary = Im > threshold;
imshow(ImBinary);
```

Statement « *ImBinary = Im > threshold* » creates a logical mask *Imbinary* which has the same size as *Im*. One pixel (i, j) of this mask is set to “1” when the condition « Im(i, j)> *threshold* » is true otherwise it is set to “0”. The mask created is thus the desired binary image.