

Computer Vision Exercises

Lecture 2: Image Acquisition and Processing

Many times you will want to process a specific image, other times you may just want to test a filter on an arbitrary matrix. In MATLAB you will need to load the image so you can begin processing. If the image that you have is in color, but color is not important for the current application, then you can change the image to grayscale. This makes processing much simpler since then there are only a third of the pixel values present in the new image.

A stored image must be imported into MATLAB to be processed. If color is not an important aspect then **rgb2gray** can be used to change a color image into a grayscale image. The **class of the new image** is the same as that of the color image. As you can see from the example M-file in Figure 4.1, MATLAB has the capability of loading many different image formats, two of which are shown.

The function **imread** is used to read an image file with a specified format. Consult **imread** in MATLAB's help to find which formats are supported.

The function **imshow** displays an image, while **figure** tells MATLAB which figure window the image should appear in. If figure does not have a number associated with it, then figures will appear chronologically as they appear in the M-file.

```
%load a color .bmp image and convert to grayscale
A = 'C:\MATLAB6p5\work\splash2.bmp';    %designate matrix A as the specified file
B = imread(A, 'bmp');                   %matrix B loaded with bitmap file specified by A
figure(1), imshow(B);                   %show image B in figure window 1
C = rgb2gray(B);                         %convert color image to grayscale image
figure(2), imshow(C);                   %show image C in figure window 2

%load a color .jpg image and convert to grayscale
D = 'C:\MATLAB6p5\work\b747.jpg';       %designate matrix D as the specified file
E = imread(D, 'jpg');                   %matrix E loaded with JPEG file specified by D
figure(3), imshow(E);                   %show image E in figure window 3
F = rgb2gray(E);                         %convert color image to grayscale
figure(4), imshow(F);                   %show image F in figure window 4
```

1. Write a program that reads the image **castle.png** file and stores its gray level data inside an array.

See function *read_img.m*

```
img=imread(path); % 512x512x3
img_gray=rgb2gray(img); % 512x512
imshow(img_gray);
```

Note: *img* é array 512x512x3 enquanto *img_gray* é 512x512 isto porque ficou apenas com 1 banda para descrever a intensidade (gray scale). Esta imagem tem 512 pixéis de largura e altura (matriz 512 por 512)

Writing an Image – Sometimes an image must be saved so that it can be transferred to a disk or opened with another program. In this case you will want to do the opposite of loading an image, reading it, and instead write it to a file. This can be accomplished in MATLAB using the **imwrite** function. This function allows you to save an image as any type of file supported by MATLAB, which are the same as supported by **imread**.

In order to save an image you must use the **imwrite** function in MATLAB. This M-file loads a bitmap file and saves the grayscale image created as a JPEG image.

```
%load a color .bmp image and convert to grayscale
A = 'C:\MATLAB6p5\work\splash2.bmp';    %designate matrix A as the specified file
B = imread(A, 'bmp');                  %matrix B loaded with bitmap file specified by A
figure(1), imshow(B);                  %show image B in figure window 1
C = rgb2gray(B);                       %convert color image to grayscale image
figure(2), imshow(C);                  %show image C in figure window 2
%save the image as a .jpg image
imwrite(C, 'C:\MATLAB6p5\work\splash2new.jpg', 'jpg') %save grayscale image C as jpeg file
```

Note: The “splash2” bitmap picture must be moved into MATLAB’s work folder in order for the **imread** function to find it. When you run this Mfile notice how the JPEG image that was created is saved into the work folder.

2. What is the average gray level on the image? Subtract that value to the whole image. What happens?

See function *average_filter.m*




```
avg=uint8 (mean (mean (img) ) )
```

(Para se obter um valor tem de se aplicar 2 vezes *mean* por ser matriz; *unit8* para arredondar)

```
img_ft=img-avg;
img_ft(img_ft<=0)=0;  Evitar valores negativos com subtração
imshow(img_ft)
```

Mean=113 (valor usando *unit8*)

A subtração do valor médio permite normalizar a imagem em torno de zero, ficando consequentemente mais escura, ou seja, remove componentes mais brilhantes da imagem.

		
Gray scale	Gray scale com valores negativos	Gray scale sem valores negativos com unit8

Histogram – A histogram is bar graph that shows a distribution of data. In image processing histograms are used to show how many of each pixel value are present in an image. Histograms can be very useful in determining which pixel values are important in an image. From this data you can manipulate an image to meet your specifications. Data from a histogram can aid you in contrast enhancement and thresholding.

In order to create a histogram from an image, use the **imhist** function. Contrast enhancement can be performed by the **histeq** function, while **thresholding** can be performed by using the **graythresh** function and the **im2bw** function.

If you want to see the resulting histogram of a contrast enhanced image, simply perform the imhist operation on the image created with histeq.

Negative – The negative of an image means the output image is the reversal of the input image. In the case of an 8-bit image, the pixels with a value of 0 take on a new value of 255, while the pixels with a value of 255 take on a new value of 0. All the pixel values in between take on similarly reversed new values. The new image appears as the opposite of the original.

The **imadjust** function performs this operation.

Next is a demonstration of imhist, imadjust, graythresh, and im2bw.

Also, see an example of how to use imadjust to create the negative of the image. Another method for creating the negative of an image is to use **imcomplement**.

```
A = 'C:\MATLAB6p5\work\splash2new.jpg'; %designate matrix A as the specified file
B = im2double(imread(A, 'jpg')); %read and convert loaded image to class double
figure(1), imhist(B,256); %display histogram in figure window 1
D = imadjust(B,[0 1],[1 0]); %create negative of original image
figure(2), imshow(D); %display negative in figure window 2
E = histeq(B); %enhance contrast to equally spaced bins
figure(3), imshow(E); %display contrast enhancement in figure window 3
F = graythresh(B); %calculate threshold level
G = im2bw(B,F); %convert grayscale image to binary according to threshold
figure(4), imshow(G) %display binary image
```

In this example a JPEG image is used to create a histogram of the pixel value distribution and a negative of the original image. The contrast was then enhanced and finally the image was transformed into a binary image according to a certain threshold value.

As you can see the histogram gives a distribution between 0 and 1. In order to find the exact pixel value, you must scale the histogram by the number of bits representing each pixel value. In this case, this is an 8-bit image, so scale by 255. As you can see from the histogram, there is a lot of black and white in the image. In the negative of the image pixel values have been rotated about the midpoint in the histogram. A contrast enhanced version is then produced. As you can see, there is some blurring around the edges of the object in the center of the image. However, it is slightly easier to read the words in the image. This is an example of the trade-offs that are common in image processing. In this case, sacrificing fine edges allowed us to see the words better. Next, the binary image of the image as created according to the threshold level, **thresh**. The value for thresh was displayed in the MATLAB Command Window as:

```
>>
thresh =
0.5020
```

MATLAB chooses a value for thresh that minimizes the intraclass variance of black and white pixels. If this value does not meet your expectations, use a different value when using the `im2bw` function. Another function new to this example was **im2double**. This function converts the image from its current class to class double. Many MATLAB functions cannot perform operations on class `uint8` or `uint16`, so they must first be converted into class double. This is due to the unsigned nature of class `unit`. Certain mathematical functions must be able to output to a floating point array in order to operate. When writing an image, MATLAB converts the data back to class `unit`.

3. Write a function that calculates the histogram of the image, without resorting to the histogram function supplied with MATLAB;

See function `build_histogram.m`

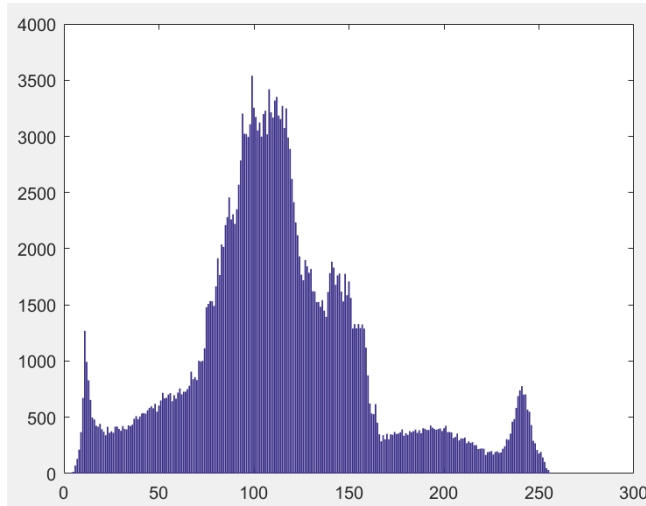
```
function gls = build_histogram(img_gray)
%UNTITLED5 Summary of this function goes here
%    Detailed explanation goes here

gls=linSPACE(0,255,256); % create an array with
256 values from 0 to 255

for i=0:255
    gls(i+1)=sum(sum(img_gray==i));
end

bar(gls)
```

end



4. Perform histogram equalization and plot the image. What has changed?

See function *build_histogram.m*

Usar função *histeq* para equalizar histograma



Contrast is increased

Do the histogram of the pollen figures tha were presented at the class

Some example code is given below

```
>> imshow(f)
>> figure, imhist(f)
>> ylim('auto')
>> g = histeq(f, 256);
>> figure, imshow(g)
>> figure, imhist(g)
>> ylim('auto')
```