# CURRENT SUBJECTS IN COMPUTER SCIENCE
## PRACTICE SESSION 1: Introduction to the Matlab
### *Image Processing Toolbox*

The main purpose of this practice is:

- Understanding how the Matlab Image Processing Toolbox works.
- Carry out a very simple image processing.
- Setting the basis needed for the next seminar: Pattern recognition.

# Example 1: Basic image processing

## 1.1.- Reading and writing images

We must first clear the MATLAB workspace of any variables and close open figure windows.

```
>> clear, close all
```

To read an image, we can use the `imread` command. The example reads one of the sample images included with the Image Processing Toolbox, `pout.tif`, and stores it in an array named `I`.

```
>> I = imread('pout.tif');
```

`imread` infers from the file that the graphics file format is Tagged Image File Format (TIFF). For the list of supported graphics file formats, type 'help imread' in the Matlab command line.
Now, we would like to display the image. `imshow` is the toolbox's fundamental image display function.

```
>> imagesc(I)
>> colormap gray
>> axis equal
```

If we now type the following, we will find information related to the way Matlab stores the image in its Workspace.

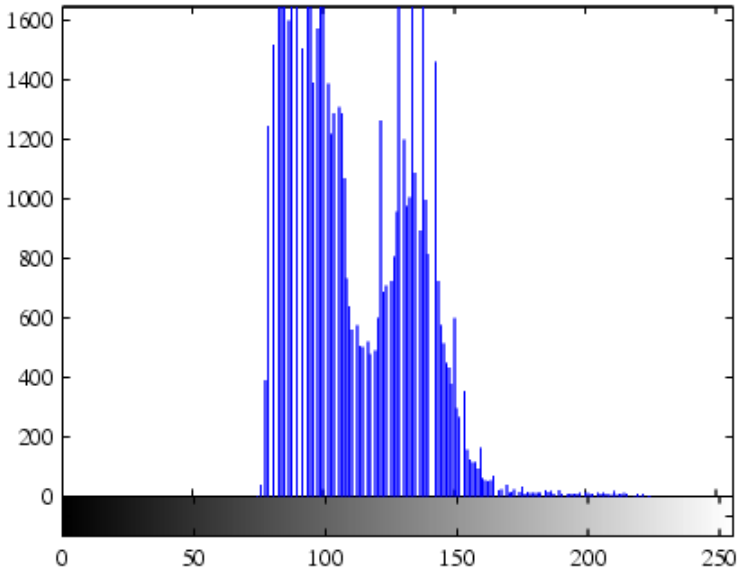
```
>> whos

Name        Size                    Bytes  Class
I         291x240                   69840  uint8 array

Grand total is 69840 elements using 69840 bytes
```

Download the file images.zip and extract the files to d:\cscs. Add the path to Matlab. You can try opening and showing other images, i.e. `I = imread('moon.tif');`

# 1.2.- Calculating image histogram

As you can see, the previous image is not well contrasted. We can improve this image by histogram equalization.To see the distribution of intensities in `pout.tif`, you can create a histogram by calling the `histo` function. (Precede the call to `histo` with the `figure` command so that the histogram does not overwrite the display of the image `I` in the current figure window.)

```
>> figure, histo(I)
```




Notice how the intensity range is rather narrow. It does not cover the potential range of [0, 255], and is missing the high and low values that would result in good contrast.

The toolbox provides several ways to improve the contrast in an image. One way is to call the `histeq` function to spread the intensity values over the full range of the image, a process called *histogram equalization*.

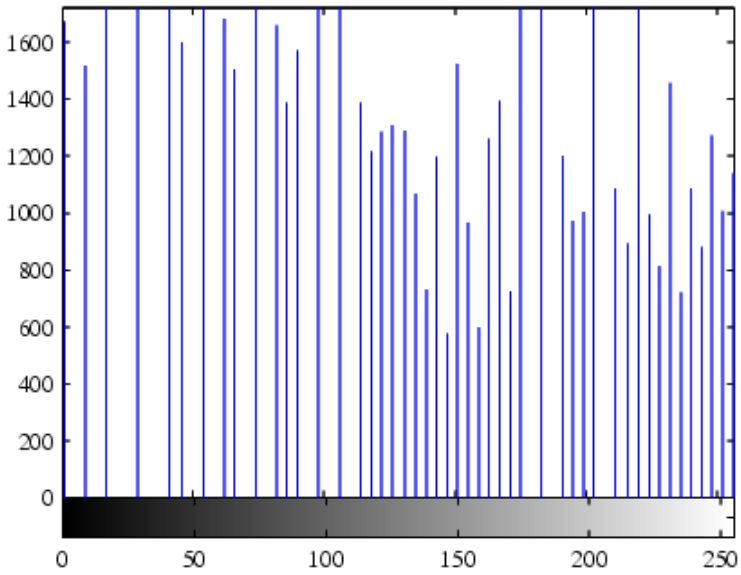
```
>> I2 = histeq(I);
```

Display the new equalized image, `I2`, in a new figure window.

```
>> figure, imagesc(I2), axis equal
```

Call `histo` again to create a histogram of the equalized image `I2`. If you compare the two histograms, the histogram of `I2` is wider than the histogram of `I1`.

```
>> figure, histo(I2)
```




Now, the distribution of pixel values covers the entire range [0 255].

Note: Depending of Matlab version, the `histeq` command may not be available.

# Example 2: Dealing with pixels

As we have already seen, the toolbox provides us with lots of functions that deal with images and perform operations on them. That means that we need not operate with pixel values to perform a histogram equalization. However, for instance, if we want to make a correlation between two images, then, we should code it ourselves.

# 2.1.- Calculating brightness.

We are going to calculate image brightness as:

$$B = \frac{1}{R \cdot C} \sum_{i=1}^{R} \sum_{j=1}^{C} I(i, j)$$

Where $R$ and $C$ are the number of rows and columns respectively and $I(i, j)$ is the pixel value placed at row $i$ and column $j$.

We must open a new m-file and name it `brightness.m`. This file should be placed at our current directory.


We should write down the following commands in `brightness.m`. This would act as a batch file, reproducing commands when called from the Matlab command line.

```
%   brightness.m
%   Calculates image brightness
%

%reads image from file
I = imread('moon.tif');

%shows image in new window
figure, imagesc(I);
colormap gray

%finds image size
[R,C]=size(I);

%casts data to double type (cannot operate with unsigned int data type)
I = double(I);

%Calculates brightness
sum = 0;
for i = 1:R,
    for j = 1:C,
        sum = sum + I(i, j);
    end
end

'brightness is B'
```

```
    B = sum/(R*C)
```

We can call now `brightness.m` from the Matlab command line:

```
>> brightness
```

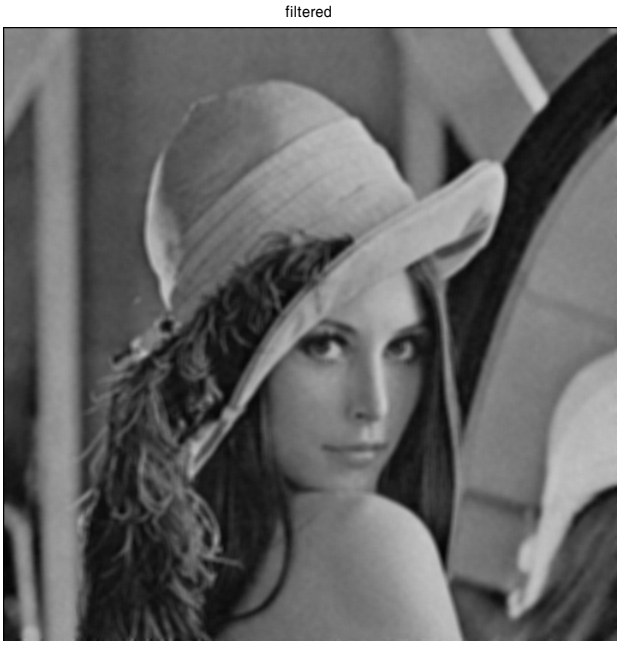
# EXERCISE 1: Calculating the mean filter.

The mean filter calculates an average of pixels in an NxM neighbourhood of pixel (i,j). The result is stored in pixel (i, j). It is thus a local operation.

Now, complete the following code to calculate the mean filter in a 5x5 neighbourhood of pixels. The mean filter is very useful to reduce noise in images. However, as you will see, it reduces image sharpness.

```
%  meanfilter.m
%   Calculates mean filter in a 5x5 window
%

%reads image from file
I = imread('lena.png');

%shows image in new window
figure, imagesc(I);
colormap gray
axis equal

title('original');

%finds image size
[R,C]=size(I);

%artificially add noise to image data
J = double(I);

for i=1:R,
    for j=1:C,
       J(i,j) = J(i,j) + 10*randn;;
    end
end

figure, imagesc(J)
colormap gray
axis equal

title('original + added noise')

%casts data to double type (cannot operate with unsigned int data type)
J = double(J);


%PLACE YOUR CODE HERE


K = uint8(K);
figure, imagesc(K)
colormap gray
axis equal
```

```
title('filtered');
```


# EXERCISE 2: Convolution

Convolve the image with the following mask. What can be observed in the result?

Mask:

| | | |
|---|---|---|
| 0 | 1 | 0 |

| | | |
|---|---|---|
| 1 | -4 | 1 |
| 0 | 1 | 0 |

```
%  convolution.m
%  Makes the convolution operation with a predefined mask
%

%reads image from file
I = imread('lena.png');

%shows image in new window
figure, imagesc(I);
colormap gray
axis equal

title('original');

%finds image size
[R,C]=size(I);

%casts data to double type (cannot operate with unsigned int data type)
I = double(I);

mask = [0 1 0;
        1 -4 1;
        0 1 0];

% PLACE YOUR CODE HERE


K = uint8(K);
figure, imagesc(K)
colormap gray
axis equal
title('filtered');
```

Repeat the convolution using the following masks. Compute the absolute value of the result. Compare the results of mask 1 and mask 2. Sum the result of both masks.

Mask 1:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Mask 2:

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

Repeat the convolution using the following masks. Compute the absolute value of the result. Compare the results of mask 1 and mask 2. Sum the result of both masks.

Mask 1:

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Mask 2:

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

# EXERCISE 3 (optional): Harris corner detector

Compute the 5 most significant responses using Harris Corner detector, as explained in the lecture slides.The matrix *M* should be computed on a 5x5 neighbourhood of the pixel. Use the following image to test the results.

# To be included in your reports:

- Printed code of your Matlab functions.
- Place your result figures in your report.