Image Analysis, Laboratory Session 1

Matlab

`Matlab` is a programming environment mostly for numerical matrix computations. An image can be regarded as a matrix where each element in the matrix corresponds to a *pixel* in the image.

Start matlab in the directory where all the lab files[1] are located by entering `matlab` at the command prompt. Alternatively, add a path to the directory using `addpath` at the matlab prompt.

Different image formats

In the same way as for video and music files there exist a number of different file formats for images. Even for digital images, there are a number of different formats. We are going to look at some common such formats.

Make sure you have an image file called `kalle.pgm` in your directory. You can type `dir` to check. The image is stored in ascii-format and is thus readable in an editor, even if it is not so easy to see what the image contains. Try for example `type kalle.pgm` or load it in a text editor.

To load the image into matlab one can use

```
>> im=imread('kalle.pgm')
```

By putting a semicolon after the command you will avoid seeing Matlab writing out the results, which in this case is a rather large matrix. However, the ascii-format is not so compact as a binary format, compare with the GIF-image `kalle.gif` which is stored in a binary format.

For the computations later on, it is better to convert the data format from UINT8 to DOUBLE by typing:

```
>> im=double(im);
```

Create also a somewhat smaller image:

```
>> tolv=[1 2 3;4 2 6;7 8 7;2 11 10]
```

You can find out which matrices you have available in Matlab and their sizes with the command

```
>> whos
```

For more information about matlab's imaging formats you can write

[1] The lab files can be downloaded from the course home page.

```
help imread
```

EXIF is a standard used by digital camera manufacturers to store information in the image file, such as, the make and model of a camera, the time the picture was taken and digitized, the resolution of the image, exposure time, and focal length. Try for example

```
>> exifread('domkyrkan.jpg')
```

To look at an image in Matlab

As mentioned in the lectures, computers are using color-tables instead of letting the pixel-values directly determine the intensity in each pixel. In Matlab, you can specify these tables yourself. Try the following commands:

```
>> map=(0:11)'/11*[1 1 1]
>> colormap(map)
>> image(tolv)
```

The first row creats a color palett with 12 different colors. Each row has three elements, that determine the amount of red, green and blue respectively. In this case, the first one was selected entirely black and the remaining ones in gray-scale levels up to the twelfth one, which is entirely white. The second row tells Matlab to use the palett map. Try

```
>> help colormap
```

The third row is a command for showing the matrix `tolv` as an image. Try

```
>> help image
```

to get more information. What will happen if you write

```
>> colormap((0:11)'/11*[1 0 0])
```

Substiture `[1 0 0]` with `[0 1 0]` and `[0 0 1]`. Try also

```
>> colormap(rand(12,3))
```

several times.
Look at the matrix `im` using the commands

```
>> image(im)
>> colormap(gray(255))
```

How is `gray` working? What other pre-defined color maps are there? Observe that the images are shown with the origin in the upper left corner. There are also other routines in Matlab to illustrate matrices. Show `im` using

```
>> mesh(im)
```

The matrix/image is now shown as a gray-level landscape, where each pixels gray-level value controls both hight and gray-level. You are now seeing the landscape somewhere from above. The command `view` changes the viewing angle. Try

```
>> view([-20 50])
>> view([-190 70])
>> view([0 90])
```

This way of changing the viewing angle is good if you want a special angle. If you want to try different angles it is easier to click on the rotation arrow in the control panel and then click and drag with the left mouse button in the image. Observe that `mesh` uses a coordinate system with the origin in the lower left corner. Turn the image right.

You can close a window using

```
>> close
```

Histograms and gray-level transformations in Matlab

In Matlab, it is easy to show a histogram of the gray-levels. Load `tire.tif`, convert to doubles and try

```
>> help hist
>> hist(tolv(:),1:12)
>> hist(im(:),20)
>> hist(tire(:),1:256)
```

If you don't use `(:)`, Matlab will make a separate histogram for each column in the image.

A gray-level transformation is not difficult to perform either. Try

```
>> trans=[24:-1:18 16:-2:8]
>> plot(trans)
>> c=trans(tolv)
```

How would you perform histogram equalization?
Task: Write your histogram equalization as a `function` in a file called `histeq_own.m` with arguments according to `imout=histeq_own(im)`. Show your program to the teacher!
Hint: Combine the commands `hist` and `cumsum` (and possibly `ceil`) in order to get a suitable gray-level transformation. If needed, look at the lecture notes from Lecture 1.

Matlab has also implemented its own version of histogram equalization in a routine called `histeq` available. Write `help histeq` and compare with the results of your function on the image `tire`.

Make also some illustrations of reducing the size of an image.
Hints:

```
>> stepsize = 2;
>> y=rgb2gray(imread('domkyrkan.jpg'));
>> y=y(2:stepsize:end,2:stepsize:end);
>> image(y);
```

```
>> faktor=2;
>> y=rgb2gray(imread('domkyrkan.jpg'));
>> y=ceil(y/faktor)*faktor;
>> image(y);
```

At what level can you distinguish the original image from the reduced one?

Re-scaling images

Sometimes, you may want to enlarge or reduce the size of an image. Think about what problems you might encounter if you would like to make an image 3.72 times wider and 4.1 times higher. How would you determine the new pixel-values? There are several different ways to accomplish this.

Several methods are implemented in matlab with the command `imresize`. Read more in

```
>> help imresize
```

The options `p` (which is "default"), `b` and `c` selects between pixel replication, bilinear interpolation and cubic spline interpolation. For the `c`-option you may also provide a parameter. Somewhere between -0.5 and -1.0 usually provides good results. Enlarge the image `im` using different methods and compare them using for instance

```
>> im2=imresize(im,[exp(1) pi].*size(im),'nearest');
>> im2=imresize(im,[exp(1) pi].*size(im),'bilinear');
>> im2=imresize(im,[exp(1) pi].*size(im),'bicubic');
```

Generate the following image in Matlab

```
>> prickig=zeros(30,40);
>> prickig(2:2:end,2:2:end)=256;
>> image(prickig);colormap(gray(256));
```

The results might not be what you expected. The reason is interference between pixels on the computer screen and the pixels in the image file. Enlarge the image window with the mouse and see what happens.

Rescale the image `prickig` with a factor 1.6 with the three different methods. What happens with sharp edges in the different cases? Is any of the methods clearly worst of clearly best, or can they be good in different cases? How should an algorithm that enlarge an image really nicely work?

Convolution

Several operations on images can be described as *convolutions*. Some examples are low- and high-pass filtering and edge detection. Write down how some simple filter of different types and sizes can look like. Convolutions are very simple to perform in Matlab. A filter can be given as a matrix, e.g.

```
>> medel=[1 1 1;1 1 1;1 1 1]/9
```

Then convolutions is performed using the routine `conv2` , tex

```
utbild=conv2(inbild,medel)
```

The gray-levels of the resulting image is not always between 0 and 255 and therefore it is advisable to use the command `imagesc` to look at the resulting image. This command transforms the minimum gray-level in the image to 0 and the maximum gray-level to 255, based on a linear gray-level transformation.

When convolving image, there is often problems close to the boundaries of the image, since it is of finite size. It is not obvious how to define the convolution at the image boundary. In the routine `conv2` the image is padded with zeros outside the boundary and the resulting image, therefore becomes a little bit larger than the initial image. If you don't want a bigger image, you may use the option 'same' or 'valid'. 'same' removes the boundary so that the image gets the same size. 'valid' includes only pixels in the resulting image that hasn't used the zeros around the input image and thus the resulting image becomes smaller.

A specific usage of low-pass filtering is noise reduction. Load the image `pkalle.pgm` and look at it before and after filtering with the mean value filter `medel`. As you can see some of the noise disappears, but so does also the finer details in the image.

Some other common filters are

- Differentiation in the x-direction: `derx = [-1 1]`
- Differentiation in the y-direction: `dery = [-1;1]`
- The Laplacian filter: `laplace = [0 1 0;1 -4 1;0 1 0]`

Try differentiation in the x- and y-direction and discrete Laplace filtering on the image `im`. By using these filter you can obtain a simple edge detector. Unfortunately, it is rather noise sensitive, see extra Exercise 1.

(Extra exercises if you have time)

1 Try to develop a filter that removes noise but preserves fine details better than low-pass filtering.
Hints: If a pixel is significantly different from the mean value of the surrounding pixels, it is probably wrong and should be replaced with the mean value, otherwise, it should remain the same.

α Investigate the noise sensivity for a simple edge detector.