Image Analysis and Spatial Statistics
Computer Exercise 1 : Getting Started

Mats Kvarnström
Department of Mathematical Statistics
Chalmers University of Technology

January 2005

1 Introduction

This computer exercise is intended to be an introduction to image processing using Matlab and the Image Processing Toolbox (IPT), covering the basic procedures like loading and saving images and how to display them. Also included are how images are represented and the different so called Image Types used in Matlab. In short, the basic things needed to get started with Matlab as image analysis tool. Some basic Matlab experience is assumed, but not much.

In Section 3 a very short introduction to the UNIX program *XV* is given. This program will be useful when converting an image from an image format not supported by Matlab, and the other way around.

The description is written in a 'hands-on-keyboard-and-try'-kind of way, by which I mean that you are supposed to investigate by your own with this document as a guide. You are encouraged to use the Matlab command `help` as soon as a new command or function is introduced. As we get in trouble, I'll try to explain what happened and why we got in trouble and how to solve it. For a thorough, but less trouble-oriented exposition of Matlab's Image Processing Toolbox you are referred to the IPT User's Guide by Mathwork, available in HTML format by writing `doc` in a Matlab command window.

Everything written in `Courier` refers to a Matlab variable, command or function and everything with a `>>` in front of it, is for you to type in the Matlab command window.

In Section 4 a list of the Matlab commands and functions used in this document is supplemented. The command you need to know about right away is the `help` command mentioned above.

1.1 Images

The images used in this laboration are

- `rice.png`: A standard image in Matlab.
- `fumitory-007.jpg`: A color photograph of a fumitory plant ('jordrök' in Swedish) standing out against a brown background. Can be downloaded from the course homepage.

1.2 Start Matlab

To open a Matlab command window on a UNIX station, type `matlab` in an X-term window (or equivalent).

2 Basic image operations in Matlab

This section is built up as an exercise. So, hands on keyboard and follow the instructions. As you always should do whenever you encounter a new Matlab command or function, type `help` followed by the name, to get full syntax information and a short description.

2.1 Loading and displaying of images in Matlab

In order to load an image to Matlab from your present directory, the command `imread` is used. To get the habit, start by writing

```
>>help imread
```

but don't read the entire description right away. The details and all options may be needed for later work, but not right now. Write

```
>>I=imread('rice.png');
```

to read the TIFF-file `rice.tif` and store it in the variable `I`. This is the working variable and you don't have to bother about the actual filename anymore (unless you have to reload it again for some reason).

Now you probably want to take a look at it. Type

```
>>figure(1)
>>imshow(I)
```

and Matlab will first open an empty figure (window) and then display the image stored in `I` in it.

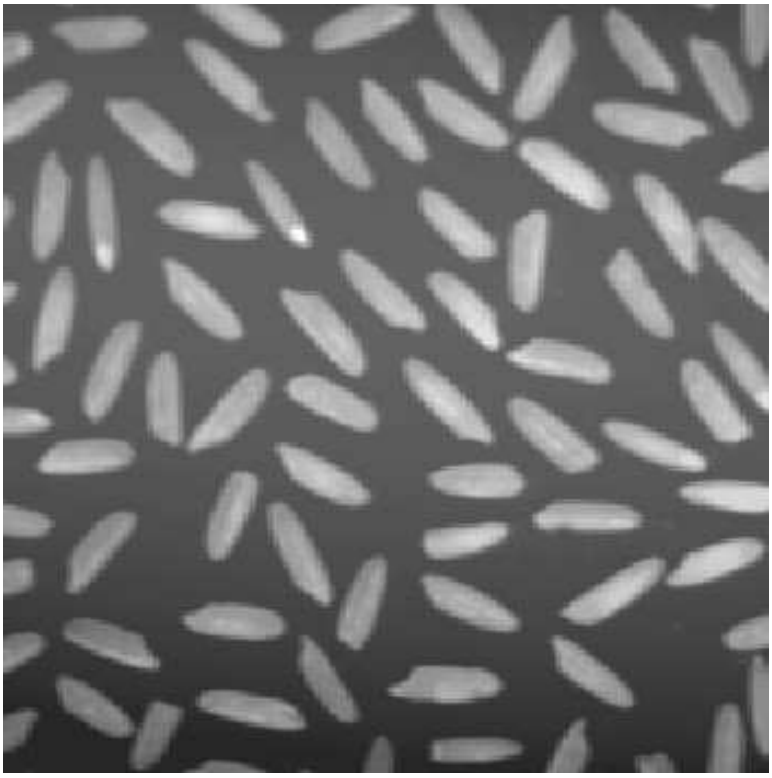

Figure 1: The image `rice`

2.1.1 Histogram

Also, look at the histogram to have a feeling for the distribution of the pixel values in the image.

```
>>figure(2)
>>imhist(I)
```

The histogram is of great importance when we for example *threshold*. See Section 2.3.2 for an example of this operation.

2.2 Representation of images

Now that we have an image variable we want to know how it is represented (i.e. how it is stored) by Matlab, and what kind of operations we are permitted to do on it. To begin this exploration, type

```
>>whos
```

and you'll get a list of all variables defined in this Matlab session [1] , including our image variable `I`. Here you can see for example that the size of the image `I` is 256×256 pixels and that the class is `uint8` array. This means that it has the data structure of an array with each element being stored as an unsigned integer of size 8-bit. The last thing is important. By default, most data in Matlab are stored as arrays of with elements stored in double precision (64-bit) as floating-point numbers called `double`, and therefore most of Matlab's functions are only defined on this class of numbers. Images however, are usually stored in arrays of 8-bit or 16-bit unsigned integers called `uint8` and `uint16`, in order to reduce memory consumption. So, problems can arise when one needs to use the majority of the standard functions in Matlab (several functions in IPT support `uint8` and `uint16`, though).

Let's say that you want a brightened version of the image. A way to do this is to convert the image variable to `double` and then multiplicate by, say, a factor 1.5:

```
>>Ib=1.5*double(I);
```

Now try to display `Ib` in figure number 2:

```
>>figure(2)
>>imshow(Ib)
```

The displayed image is all white. What is the problem? To answer that question thoroughly, we have to know pretty much about the different so called Image Types and how Matlab handles images in general.

2.3 Image Types

The problem in the last section is due to the fact that the IPT handles four different types of images

- Indexed Images
- Intensity Images
- Binary Images
- RGB Images

The most common image types are the Indexed and the Intensity type. Loosely speaking, one could say that the two types in the middle are special cases of the first one, whereas the RGB Image type is a generalization of the Intensity Image.

[1] In later versions of Matlab, this list of all variables are shown in the workspace window of the desktop.

2.3.1 Indexed and Intensity Images

The following two paragraphs might be a bit hard to digest. Don't bother to get all the technical details at your first reading. The important stuff is in the practical consequences it implies.

An Indexed Image is represented by a data matrix `X` with element values in the range $\{1, \ldots, m\}$ corresponding to indices, and a *colormap* `map`. The colormap is an m-by-3 array of class `double` containing values in the range $[0, 1]$, such that each row of `map` specifies the red, green, and blue (RGB) components of a color. The elements in the data matrix can be either of class `double`, `uint8`, or `uint16`. The reason for the name 'Indexed' is that each element in the data matrix (usually simply referred to as 'the image') specifies the color of the corresponding pixel by referring to the *index* (row) of the colormap `map`.

The Intensity Image consists solely of a data matrix `X`, the elements of which can be of class `double`, `uint8`, or `uint16`. Here, the elements represents intensities, usually gray levels, where 0 normally represents black and 1, 255, and 65535, respectively, represents white, according to class. The 'usually gray levels' in the last sentence is due to the fact that Matlab uses a colormap to display the image here as well (!). The standard colormap of the Intensity Image corresponds to the 256 gray levels, though.

The practical consequence of this is that the colormap in use always specifies which colors we use and that we only have to bother about these two different types when the data matrix is of class `double` (because of the different ranges of the two image types). Usually, with rare exceptions, images with elements of class `double` are interpreted as Intensity Images and therefore the gray scale 'lies in the range' $[0, 1]$ (with 0 and 1 representing black and white, respectively, and everything in between as different shades of gray).

Now we have reach a point where we can fix our problem when trying to display the brightened image `Ib`. When we use `double` to convert an image from `uint8` to `double`, Matlab interprets the new image as an Intensity Image and consequently, by the preceding three paragraphs, 0 represents black and 1 white. If you look at a slice of the data matrix `Ib` (don't use semi-colon now!):

```
>>Ib(1:10,1:10)
```

you see that the elements are all greater than 1. To see what the smallest element of `Ib` is, use `min` two times (first along the columns and then along the rows):

```
>>min(min(Ib))
```

No wonder the image turned up all white; all elements are larger than 1![2]

[2]You could also have used `imhist` to realize these facts immediately.

Therefore we need to normalize:

```
>>Ib2=(1.5*double(I))/255;
>>figure(2)
>>imshow(Ib2)
```

An alternative approach is to use the function `im2double`, which automatically normalizes to the range $[0, 1]$, i.e. to an Intensity image of class `double`.

```
>>Ib3=1.5*im2double(I);
>>imshow(Ib3)
```

Notice that the images `Ib2` and `Ib3` consists of elements of class `double` in the range $[0, 1.5]$. If you like to have the original $\{0, \ldots, 255\}$ integer range of the original image as a reference scale. Then `imshow` has an optional parameter specifying the data range you'd like to represent the gray scale with (maybe you already have noticed this when using `help imshow`!)

```
>>imshow(Ib,[0,255])
```

This parameter is very useful and will probably come to use in your later work in this course.

2.3.2 Binary Images

A Binary Image can be said to be an Indexed Image with the exception that we only have two different colors available corresponding to on and off (white and black, respectively, if we use a gray scale colormap). A binary image can be either of class `double` or `uint8`. Create a Binary Image from `I` by *thresholding*

```
>>Ibin=I>150;
>>imshow(Ibin)
```

Now, the pixels in `I` larger than 150 is set to `1` and the pixels below to `0`, and as you probably have figured out once you'd displayed `Ibin`, this operation might come to use. It is one of the most primitive so called *image segmentation* method.

2.3.3 Colormap manipulation

When loading an image with `imread`, there is a optional output variable called `map` (see `help imread`). This is the variable for the colormap, mentioned above. This is returned empty however, if a colormap wasn't saved together with the image (data matrix), i.e. if the image is not an Indexed image. Even if this is the case, Matlab uses a colormap to display all of the three types above. Write

```
>>colormap
```

to see the colormap which is currently used. It should be a gray scale (there should be an equal amount of each of the three colors at each row). Notice also that it is of size 256-by-3.

If we want to use another colormap, we first create an array of appropiate size with elements of class `double` in the range $[0, 1]$. Then we use the function `colormap` (it is both a function and a variable) to tell Matlab to use this new array as colormap [3] . It is quite time-consuming to specify e.g. 256 colors with there RGB components, though. Fortunately Matlab has some built-in colormaps. One of them is called `jet`

```
>>map=jet(256);
>>colormap(map);
```

where the argument to `jet` of course specifies the number of rows. If you haven't used `help` for this function, do it now, and you'll see the other colormaps among the related functions at the bottom of the description. Try some of the other maps together with the images `I`, `Ib` and `Ibin`! Also: what happens if you specify to few colors. Except from using `gray`, you can construct a gray scale map by (of course; use `help` if you don't know how `linspace` works)

```
>>map_gray=[linspace(0,1,256);linspace(0,1,256);linspace(0,1,256)]';
>>colormap(map_gray)
```

Notice the transponate on `map_gray`! As an experiment try to change the range argument in `linspace` from `(0,1)` to e.g. `(1,0)` or `(.5,1)` and see what happens.

When you change from the original colormap to an artificial one, it is called pseudocoloring and it is not just something you use for your homepage or something like that. It can in fact help you to see things which might not be visible if we used the 'real' colors, e.g. small subtle changes in the texture. Try for example a random colormap by writing

```
>>map_rnd=rand(256,3);
>>colormap(map_rnd);
```

where `rand` creates uniformly distributed random numbers between $[0, 1]$.

2.3.4 RGB Images

An RGB image is stored in Matlab as an 3-dimensional array of class `double`, `uint8`, or `uint16`, where the first two dimensions specifies the pixel location,

[3] If you have several figures open, be sure to first activate the figure where you want to change the colormap. Either click on the figure window or use the function `figure`.

Figure 2: The RGB image `fumitory-007.jpg`

whereas the third dimension defines the red, green, and blue component of the pixel. (Informally speaking it consists of three Intensity images, one for each color, stored in a single array.) This means that an RGB image doesn't use a colormap, the color is handled by the data matrix. To demonstrate an image of this kind, load the image `fumitory-007.jpg`. (No, it has nothing to do with James Bond...)

```
>>Ifum=imread('fumitory-007.jpg');
>>imshow(Ifum)
```

Notice that the colors of the displayed image is independent of the colormap used.

Extraction of the three 'color planes' to separate images can be done with Matlab's standard matrix handling.

```
>>IfumR=Ifum(:,:,1);
>>IfumG=Ifum(:,:,2);
>>IfumB=Ifum(:,:,3);
```

Display these three images using `imshow`. Notice that they can be thought of as three Intensity images, each specifying the intensity of the corresponding

color. You might want to change colormap to gray scale if you haven't done that already. It may also be of interest to use `whos` to see the representation (and the memory usage!) of these 'slices'.

If we are looking for the green standing out against the dark background we display the green component and normalize with the sum of the intensities at each pixel location. Notice that we have to convert to `double` first. Create such an variable first and then display it:

```
>>Igreen=double(IfumG)./sum(double(Ifum),3);
>>imshow(Igreen)
```

where `sum(X,3)` means that the summation is carried out over the third dimension of the array `X` (the three different color planes in our case) and the dot before the / means, as usual in Matlab, that the division should be carried out component-wise.

2.4 Saving images

To save the image `Igreen` as a TIFF-file (we'll need it in forthcoming Computer Exercises), type

```
>>imwrite(Igreen,'fum_green.tif','tif');
```

In order to save memory, the image (which was of class `double`) is saved in 8-bit format `uint8`. To illustrate this load the previously created image file to the variable `Igreen2`:

```
>>Igreen2=imread('fum_green.tif');
```

and use `whos` to see the representation.

It should be noted that it is also possible to save the actual image *variable*, instead of saving it in an image file. This can be done with the standard Matlab command `save`. You should notice however that this means that you can only open the saved image (variable) in Matlab.

3 Converting images using the XV program

Sometimes you stumble across images that are not in one of the formats which Matlab can handle (see `help imread` for a list of the formats supported by Matlab). If that is the case, then you have to convert the image to a Matlab-supported format before you can do anything else. A standard image handling program on UNIX capable of this (and more) is *XV*.

Start *XV* by typing `xv` in an X-term window and a window like the one in Figure 3 will pop up. Right click on the `XV`-window to open the control window.

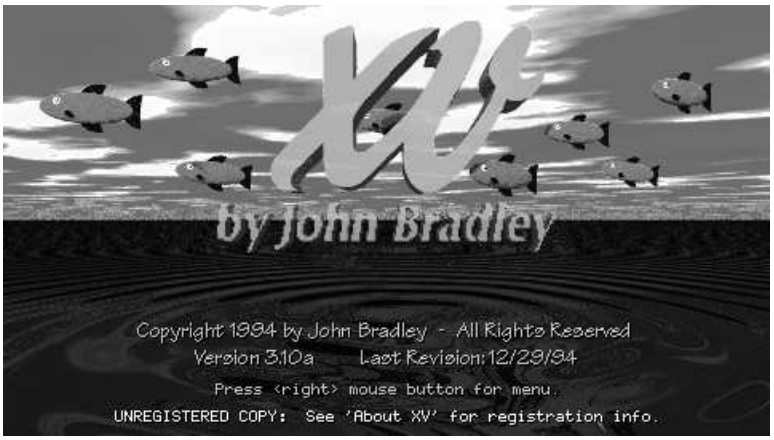

Figure 3: The *XV*-program can be used e.g. for convertion of images in formats not supported by Matlab.

3.1 Example of a convertion

To be concrete, let's say we need to convert the previously saved TIFF-image `fum_green.tif` to JPEG-format.

1. Click on 'Load' and browse through your files. Double-click on `fum_green.tif` and your normalized green-component image will be loaded and displayed.

2. Click on 'Save'.

3. Choose format to 'JPEG'. Notice that the filename at the bottom of the window changes file extension suffix to `.jpg`. Make sure that 'Colors:' is set to 'Greyscale' and click on 'OK'.

4. Now another (format depending) window pops up. Here you can make adjustments according to the format you're saving to. For JPEG the options are different levels of smoothing and quality (compression). Set for example Quality to 75% and Smoothing to 0% and click 'OK'.

4 Useful Matlab commands and functions

- `clear`: This command clears all variables currently in Matlab's working memory. Can also be used to clear specific variables. This may be useful especially when working with images.

- `colormap`: This is (usually) both a variable and a function. As a function it specifies which colormap to be used when handling indexed images. When using `imread` without a `map` output variable, the colormap is stored in the implicit (not visible) variable `colormap`.

- `dir`: Lists the files in the present directory. Faster than the Unix command `ls`.
- `double`: Converts an element to class `double`.
- `gray`: This function creates a (linear) gray-scale color map.
- `help`: This is probably the most frequently used command when using Matlab. It is used not only when you need a description for a function but, more often, when you need to check the settings for the in and out parameters and variables, at function calls. It is also used when exploring toolboxes. When you, for example, type `help images` you get a complete list of all the functions and demos in the image processing toolbox.
- `im2double`: Converts an image to class `double`.
- `imhist`: Displays the histogram of the image.
- `imread`: Reads an image file to a Matlab variable.
- `imshow`: This function is used to display an image, which is one of types Indexed, Intensity, Binary and RGB image. For the first three types the colors are specified by the colormap used. A closely related function is `image` (standard Matlab).
- `imwrite`: Writes a Matlab image variable to an image file.
- `min`: Finds the minimum element value and (optional) the index of this, along one dimension at a time, of an array. So, for an image use this twice, first along the columns and then along the rows.
- `sum`: Sums the elements of an array, along one dimension at a time. See also `min`.
- `whos`: A command used to list all the variable names currently used by Matlab, together with information such as size and class.