CSC, KTH
DD2423 Image Analysis and Computer Vision:

General laboratory guidelines

This document provides general information about the labs and specific guidelines regarding image handling in Matlab. Read through the text carefully.

Using Matlab

If you are not familiar with Matlab, you may be interested in "Användarhandledning till Matlab", that can be bought at CSCs studentexpedition. You may also read through the first sections of chapter 2 in the "Matlab's User's Guide", while trying out some examples on your computer. Note that Matlab has a built-in help facility based on the command `help`.

Starting up Matlab

Before you start Matlab for the first time, you should check the search paths. On CSCs Unix machines you do this with

```
echo $MATLABPATH
```

and make sure that the paths include the subdirectories of the files used for the labs. Thereafter you may start Matlab with

```
matlab
```

Within a few seconds a window with the Matlab logo will be shown, and moments later a window with a welcoming message and a prompt. If not, make sure that all above mentioned instructions have been properly followed. If your environment is too slow, you may prefer to use terminal based interaction. You should then start up Matlab with

```
matlab -nojvm
```

Loading and displaying images

Images in Matlabs binary format. In order to load one of the binary images from the image database of this course, you may use

```
load canoe256
```

With the command `who` you can now see the following variables defined in Matlab

```
Canoe zmax zmin
```

Here the variable `Canoe` includes the image. You may then display the image using the function `showgrey`, which has specifically been written for this lab course

```
showgrey(Canoe)
```

Images in the ASCII-based format To load one of the images stored in the ASCII-based text format, you may e.g. write

```
phone = phonecalc256;
```

This operation assigns the value of the image to the variable `phone`, and the semi-colon suppresses an output of the result of this assignment. (As noted this format require considerable more time for an image to be read.) If you wish to see the result display, use

```
showgrey(phone)
```

You may test another similar image, using a shorter notation, with

```
showgrey(nallo128)
```

As you see the contrasts are somewhat low in the lower part of the image. In the first lab we will try to use different methods to improve the image quality in order to highlight information that can otherwise be hard to see.

Your own images

If you have other images that you would like to process, you may use MATLABs embedded function `imread`, that handles a number of different image formats, including tiff and jpeg – see "`help imread`". You read such an image with the sequence

```
rawpic = imread('image.tif');
floatpic = double(rawpic);
showgrey(floatpic);
```

Observe that you need to convert the image with the command `double`, before you can use the output of `imread` for lab related functions such as `showgrey`.

Conversions of image formats can be done with command `convert` as follows:

```
module add gnome
convert image.gif image.tiff
convert onepage.ps onepage.tiff
```

Cleaning up

If you have loaded a large number of images, you can clea them from the memory using

```
clear
```

and then use command `who` to see if they have been erased. You may close a MATLAB window with

```
close
```

or close down MATLAB itself with

```
exit
```

MATLAB was originally developed for processing of matrices and images are nothing else but matrices.

Embedded documentation

MATLABs embedded help functionality works as follows: Command

```
help
```

lists different topics, under which all embedded commands have been collected. Command

`help` *topic*

where *topic* is among the headlines of this comprehensive list, lists the commands associated to this topic. Command

`help` *command*

gives specific information regarding a particular command. Some of the commands/topics that will be most relevant to this course are `arith`, `axis`, `clear`, `close`, `colon`, `color`, `colormap`, `diary`, `figure`, `for`, `function`, `help`, `if`, `image`, `input`, `keyboard`, `linspace`, `load`, `ones`, `paren`, `pause`, `plot`, `print`, `punct`, `relop`, `save`, `script`, `showgrey`, `size`, `slash`, `title`, `while`, and `zeros`. Try them out!

Storage in files

Variables in MATLAB can be stored on files using the command

`save` *filename* x y z

which leads to the binary representations of the variables `x`, `y` and `z` being written to the file *filename*`.mat`. This file can later be read in another MATLAB-session with

`load` *filnamn*

Note that already existing variables having the same names will be overwritten (without any warnings in advance). You can list all the variables with

```
who
```

For getting the information about the sizes of these, use

```
whos
```

Displaying images

Images can be displayed with MATLABs embedded command `image` or `showgrey`, for grey-level images. To display multiple images, use command

```
figure
```

All graphical contents will be drawn in the *current* figure window. This is usually the last opened figure window, or the window that was activated last, e.g. by clicking onto it, making it the window in the foreground. If you like a particular window, say window number *id*, to become the *current* window, just write

```
figure(id)
```

The command

```
close
```

removes the *current* window, while

```
close(id)
```

removes the window number *id*. Using the command `subplot` you may arrange multiple illustrations in a single figure, see

```
help subplot
```

Using

```
title(sprintf('Title: (arg1, arg2) = (%d, %f)', intarg, floatval))
```

you set a title for your figure, including the parameter setting used to generate it.

Printing

What is shown in a figure window can be printed using

```
print -Pprinter
```

You may also store the graphics in a POSTSCRIPT file named *filename* by typing

```
print -deps filename
```

Before you print out the image you might want to complement it with a title or change the coordinate axes. The command

```
title string
```

assigns the title *string* to the current figure. Coordinate axes can be drawn using

```
axis on
```

or be removed with

```
axis off
```

Keeping a diary

In some cases it might be appreciated to keep all input commands and results in a diary file, such that the MATLAB-session can later be recreated. This is done by

```
diary filename
```

which makes MATLAB copy all inputs and outputs (visible in the command window) to the file *filename*. Once a diary is opened it may be closed or reopened with `diary off` and `diary on`.

Scripts

A MATLAB-script is a file containing a series of MATLAB-commands. A script-file must have a name similar to *scriptname.m*. When you type

```
scriptname
```

in the MATLAB command window (or in another script or function) the series of commands in the script-file will be executed in the same way as if you wrote the same commands directly in the command window,

To speed of the presentation of your labs **you must create a script such that the lab can easily be reproduced**. For the purpose of presentation you can interrupt the script at certain points of execution, so that the intermediate results can be presented. This can be done using the commands `keyboard`, `pause` or `input`. In many cases the first command is preferable, since it allows MATLABs command window to be accessed.

In many cases, a practical way of creating a script-file is by first investigating the methods using the command window with a diary activated, manually edit the diary-file once it has been deactivated, and then successively update the script until it can be executed in its entirety. To prevent interference with some global variables you better reset these before every execution using `clear` *variable*.

Functions

A function in MATLAB very much works like a script. The difference is that a function

- uses local variables,
- accepts input arguments,
- returns (one or many) output variables

A function called *funtionname* has to be stored in a file named *functionsname.m*. Furthermore, the function must be declared on the first line of the file in order for the interpretor to understand the arguments.

A function that accepts two different input arguments and returns a single variable has a declaration similar to

`function outvar =` *functionsname*`(indata1, indata2)`

while a declaration of a function that returns three variables, without any input arguments, will be written as

`function [utvar1, utvar2, utvar3] =` *functionsname*()

In the function-file the arguments are treated like ordinary MATLAB-variables. Results are returned by assigning a value to the output variables (see below).

Search paths

For MATLAB to find a particular file, such as a file containing MATLAB-variables, a MATLAB-script or function, the file has to be placed either in MATLABs current working directory or among the directories included in the search paths.

You may output MATLABs current working directory with the command `pwd` and change it with `cd`. You may also, within MATLAB, see the current search paths of the system, as well as change these, using the command `path`.

If you create your own library of MATLAB-functions and scripts, you may want to include these to the search paths. On CSCs Unix computers this can be done by updating the environment variable `$MATLABPATH` in your `.login`-file or by creating your own module-file (see `/info/bildat12/module/bildat12` for an example of how to create module-files).

Command line editor

MATLAB has a simple command line editor, that makes it possible for you to edit earlier input commands, instead of re-writing everything in the command window. These are some of the (emacs-like) commands of this editor:

<ctrl>-P	move to previous command line
<ctrl>-N	move to next command line
<ctrl>-B	one step backwards
<ctrl>-F	one step forwards
<ctrl>-A	to the beginning of a line
<ctrl>-E	to the end of a line
<ctrl>-D	delete to current line
<ctrl>-K	delete until the end of a line

Ongoing calculations and outputs can in general be interrupted with <ctrl>-C. If you are interested in finding a particular command line, such as the line on which the variable `thisvar` is assigned to value given by an expression, the easiest way is by first typing `thisvar` and then repeatably pressing <ctrl>-P. Then the command editor will go through only those lines starting with this combination of characters.

Comments

If a command line (e.g. in a script or a function) is preceded by a percent character `%` the rest of the information on this line will be ignored. In fact, MATLABs embedded `help`-function is based on such comment lines. The command

`help` *matlabfile*

where *matlabfile.m* is a script- or function-file, prints out the first unended sequence of comment lines found in this file. You may thus use the `help`-function also for your own functions and scripts. Just remember that the first line in the function-file needs to include the declaration of the function (see above).

Syntax and computational efficiency

The notation in MATLABs was primarily developed for treatment of matrices and vectors using a compact notation. There are a number of common constructions that allows you to avoid explicit loops and thus increase the computational speed. Since MATLAB is an interpreting system, you should try to use these constructions whenever it is possible. Some examples:

- Arithmetic sequences are generated using a colon `:` or `linspace`, such as

```
x = (1 : 2 : 97)
x = linspace(1, 97, 48)
```

- Colons can also be used to extract subsets of vectors and matrices

```
x(1 : 5 : 16)
```

- Constant matrices are easily generated with `zeros` and `ones`

```
17 * ones(3, 4)
```

- Multiplications of matrices are written with `*`, while `.*` denote element-wise multiplication

```
ones(2 : 3) * ones(3 : 4)
```

- Matrices and vectors are concatenated using parentheses `[ ]`. Spaces or commas separate elements within a line, while semi-colons work as separators between lines. An apostrophe `'` denotes a transpose. An example with a matrix created using these operators is

```
y = [[ones(1, 2), zeros(1, 2); linspace(1, 2, 4); 1 2 3, 4]' ...
     17*linspace(3, 81, 4)' * [0.1 0.2]]
```

If you still have to write an explicit loop, you should try to avoid making your function too slow, by allocating all necessary memory outside the loop, instead of inside it. For example, compare the following two constructions

```
clear x;
for n = 1:10000
  x(n) = n*n;
end

for n = 1:10000
  x(n) = n*n;
end
```

In the first loop considerably more CPU time will be needed than in the econd one, since the vector `x` is reallocated in every iteration in the first case, while it already is of sufficient length when the second loop is executed.

None of the above solutions are satisfactory, of course. Assignment like to those done above should in MATLAB be done with the embedded functionality:

```
tmp = (1 : 1 : 10000);
x = tmp .* tmp;
```

Coordinate systems

In image processing and computer vision there are a number of different conventions for coordinate systems: a common one is a Cartesian coordinate system with a horizontal right-sided x-axis and a y-axis vertically pointing straight up. However, the coordinate system in the book of Gonzalez & Woods and in Matlab is rotated 90 degrees in relation to the system we use, i.e. the first coordinate axis points downwards and the second one horizontally to the right.

How matrices are defined and accessed in MATLAB will be highlighted by a simple example. This example shows how the ASCII-based image format is defined as a function that returns an image matrix as output.

```
function pixels = matlabcoordimage()

pixels = [11 12 13 14; ...
          21 22 23 24; ...
          31 32 33 34];
```

To create a corresponding image in MATLAB where the values instead reflect the image points in Cartesian coordinates, use

```
function pixels = cartesiancoordimage()

pixels = [13 23 33 43; ...
          12 22 32 42; ...
          11 21 31 41];
```

Note that this increases the risk of confusion. Furthermore, in some programming environments vectors are accessed starting with the index zero, while the first value of a vector in MATLAB always has the index one. This is another degree of freedom that might complicate the situation.