Medical Image Processing (SEE120)

Matlab Tutorial 2

Start Matlab. Remember those useful commands: - to close all figures:

```
>> close all;
```

- to clear every variables from your workspace:

```
>> clear all;
```

- to get some information on a Matlab function:

```
>> help function_name
```

- to look for Matlab functions by keywords:

```
>> lookfor keyword
```

1. A bit on colour images

A variable can be of class 'char'

```
>> image_name = 'ngc6543a.jpg'; % image_name = 'football.jpg';

>> class(image_name)
```

Question:

Try the following for the two images ngc6543a.jpg and football.jpg and understand what you see! How do you relate the intensity of a colour component (seen in gray levels) with the resulting color image?

1. Look at the color image

```
>> figure; imshow(image_name);
```

2. Load the image in rgb_im

```
>> rgb_im = imread(image_name);
```

3. let's get some info about this image

```
>> imfinfo(image_name) whos rgb_im size(rgb_im)
```

rgb_im is a 3D matrix of size 256x320x3, containing a true colour, meaning RGB (Red Green Blue), image. In fact, rgb_im contains three 256x320 matrices, one for each color, stored together. It is then possible to process each color component separately.

To get some info on RGB images, search for "rgb" in the Matlab documentation that you open by typing doc in the Command window.

4. Let's store each colour components in variables and plot them

```
>> rgb_im_red(:,:) = rgb_im(:,:,1); rgb_im_green(:,:) = rgb_im(:,:,2); rgb_im_blue(:,:) = rgb_im(:,:,3);

>> figure; imshow(rgb_im_red); figure; imshow(rgb_im_green); figure;

>> imshow(rgb_im_blue);
```

Now you can process each colour separately! And answer the question. The little exercise below will help you!

Exercise: create a new image (rgb_im_swapped) of the football or of the nebula ngc6543a where the red and the blue layers are swapped. Try to do that without looking at the answer at the bottom of the page!

Answer:

```
>> rgb_im_swapped(:,:,1) = rgb_im_blue; rgb_im_swapped(:,:,2) = rgb_im_green;

>> rgb_im_swapped(:,:,3) = rgb_im_red; figure; imshow(rgb_im_swapped);
```

2. Flow control

Like in any other programming language, it is possible to control the flow of operation with Matlab using some condition and loop statements.
Let's go through the most common ones.

a/ if, else and elseif

Syntax:

if condition 1 is true do this

elseif condition 2 is true do this

elseif ... ...

else
do this

end

condition1 = 2; condition2 = logical(0); condition3 = (1 == 1);

% condition2 = false % condition3 = true

Try the following examples, and understand: ex.1:

if condition1 == 3
display('2 equals 3 ? Really?!');

end

ex.2:

end

ex.3:

end

ex.4:

if condition1 == 2 statement = '2 equals 2!'; display(statement);

if condition2
display('This is not true');

else
display('This is true');

if condition1 == 3 display('That is false');

elseif condition2 display('That is false too');

elseif condition3 display('That is true');

elseif condition1 == 2 display('That is true as well');

else
display('That is always true');

end

b/ for loops

Syntax:

for index going from index_start to index_end by steps of index_increment do this

end

```
index_start = 1; index_increment = 0.5; index_end = 4;

for index = index_start:index_increment:index_end index

if mod(index, 1) == 0.5
display('The index is not a natural number');

else
display('The index is a natural number');

end end
```

c/ while

Syntax:

while this condition is true do this

end

```
a = 10;

while a > 0
display('a is still greater than zero') a
a = a - 1;

end
```

d/ switch

Imagine a switch, like on a railway for instance. The train follows only one path. The same way, the code executes only one case, defined after the keyword "switch".

case 1 ...

case X ...

case N otherwise

Syntax:

___ do this _____ >___ /... ...\

__code__ >__/____ do this _____ >______rest of code_ >__ \ ... ... /

___ do this _____ >___/ __ or do that instead _ >__/

switch to the case called case_X case case_1

do this ...

case case_X do this

...

case case_N do this

otherwise
if the case isn't define, do that instead

end

Load the 'cameraman.tif' image into the variable 'cam'

```
cam = imread('cameraman.tif');
```

Check the class of cam

```
class(cam)
```

Try several type of newclass:

```
%newclass = 'uint8'; %newclass = 'uint16'; newclass = 'single'; %newclass = 'double';

switch newclass case 'uint8'

cam_newclass = im2uint8(cam); case 'uint16'

cam_newclass = im2uint16(cam); case 'single'

cam_newclass = im2single(cam); case 'double'

cam_newclass = im2double(cam); otherwise

error('Unknown or improper image class.') end
```

Check the class of cam_newclass: class(cam_newclass)

3. Functions

```
clear all; close all;
```

In order to organize better your code, make it more readable and better re-usable, Matlab offers you the option to create functions that can be called from anywhere, accept any input and return any output.

Create a function in a new M-file and start the new M-file by the command: [outputs] = function name(inputs)
The parameters 'outputs' and 'inputs' are optional.

Let's call the function image_preprocessing that accepts the name of an image file (gray-level) as an input, as well as a threshold to get it converted in black and white, and returns the loaded image, its class, its size, its dynamic range, its compression ratio and its black and white equivalent.

```
[im_gray, im_class, im_size, im_dyn_range, im_comp, im_bw] = ...
image_preprocessing('cameraman.tif', 0.45);
```

Let's analyze the outputs:

```
figure; imshow(im_gray, []); figure; imshow(im_bw, []); im_class
im_size

im_dyn_range im_comp
```

You must first create the function image_preprocessing by copying the following code in a new M-file that you will call 'image_preprocessing.m':

image_preprocessing.m

```
function [out_gray,out_class,out_size,out_dyn_range,out_comp,out_bw] = ...
image_preprocessing(file_name, bw_threshold)

out_gray = imread(file_name);
out_class = class(out_gray);
out_size = size(out_gray);
out_dyn_range = [min(min(out_gray)) max(max(out_gray))];

K = imfinfo(file_name); % see p.23 of online book original_image_size = K.Width *
K.Height * K.BitDepth / 8; out_comp = original_image_size / K.FileSize;

out_bw = im2bw(mat2gray(out_gray),bw_threshold);
```

4. Exercise

Create a Matlab function (image_tuning) that takes as a first input a color image:

the first input can either be a file name (such as 'football.jpg') or a matrix containing the image. Both options must be accepted. Use an **IF conditional statement** to differentiate between the two options and process accordingly (use the function ischar).

The function should have 3 other input parameters for tuning each color intensity level, corresponding to 3 scaling factors tune_R, tune_G and tune_B between 0 and 1 for each of the Red, Green and Blue color layers, such that:
- for Red:

new_red_layer = red_layer / max(max(red_layer)) * tune_R - for Green:

new_green_layer = green_layer / max(max(green_layer)) * tune_G - for Blue:

new_blue_layer = blue_layer / max(max(blue_layer)) * tune_B The function should return the tuned image as an output array.

The function should make sure that the image has been converted to double with values between 0 and 1. If not, it must convert it using im2double (use an **IF conditional statement**).

Use a **FOR-loop** to process one layer at a time.

The tuned image can then be given as an input to the image_tuning function for further fine tuning. With an **IF conditional statement**, make sure not to re-tune a layer that is already tuned as requested in the input parameters of the function.

To test your function, you can choose your favorite image from the net or among Windows images and save it in your current folder (or add the path of the directory where you saved your image in Matlab search path, using the command addpath. Check the help.). You can also use directly one of Matlab standard color images: ngc6543a.jpg, football.jpg, board.tif, saturn.png, ...