Lab 1

Basic Image Processing Algorithms
Fall 2016

Lab practices - Thursdays

8:15-10:00 (room 219): Miklós Koller (koller.miklos@itk.ppke.hu)

10:15-12:00 (room 219): Márton Bese Naszlady (naszlady.marton.bese@hallgato.ppke.hu)

12:15-14:00 (room 222): Miklós Koller

Teaching assistants:
Csaba Farkas (farkas.csaba@hallgato.ppke.hu),
Dávid Kerekes (kerekes.david@hallgato.ppke.hu),
Ágnes Szabó (szabo.agnes@hallgato.ppke.hu)

General e-mail address to send your code (assignments, hws, except today!):
bipa2016fall@gmail.com

Lab 1: revising MATLAB

- 7 exercises will be given
- not necessary to complete all of them, nor to send them, but the usage of these matlab commands should be confident during the rest of the semester

Exercise 1

Create a function that:

- reads in a colored image (**`imread()`**, eg. *AlfredoBorba_TuscanLandscape.jpg*),
- displays it with **`imshow()`**,
- check if the input image has 3 channels (use **`size()`**),
- converts it to grayscale if necessary (use **`rgb2gray()`**),
- displays the grayscale image,
- save the gray image (with **`imwrite()`**),
- the parameters of this function should be as follows:
  - *input1*: path to the image file;
  - *input2*: output file path;
  - *output*: grayscale image (as an array).

Create a script that:

- calls the previously written function with the appropriate parameters,
- displays with **`imshow()`** the returned image.

Exercise 2

Create a function that:

- reads in a colored image,
- rotates it with 45 degrees (**`imrotate`**),
- flips the image vertically (**`flipud`**),
- flips the image horizontally (**`fliplr`**),
- saves all the images (name is the original name with postfix, use **`fileparts`**),
- before saving check the existence of the output dir (**`exist(some_path, 'dir')`**),
- parameters:
  - *input1*: path to the image file,
  - *input2*: absolute path of the output folder,
  - *output1*: array of the rotated image,
  - *output2*: array of the vertically flipped image,
  - *output3*: array of the horizontally flipped image.

Exercise 2

Create a script that:

- calls the previously written function with the appropriate parameters,
- displays the returned images in one figure, with multiple **`subplot`**-s (output parameters):

Exercise 3

Create a function that:

- reads in a colored image,
- mixes up channel R with channel B (do it pixel-by-pixel with a **`for`** loop),
- measures the runtime with **`tic/toc`**,
- saves the image (name is the original name with postfix, use **`fileparts`**),
- parameters:
  - *input1*: path to the image file,
  - *input2*: absolute path of the output folder,
  - *output1*: array of the mixed up image,
- implements the channel mix-up *without* **`for`** loop (think about the *colon operator* ’**`:`**‘ at the array-indexing), and measure the runtime again.

Exercise 3

Create a script that:

- calls the previously written function with the appropriate parameters,
- displays the returned image:

Exercise 4

Create a function that:

- reads in an image,
- converts it to grayscale if necessary,
- binarizes the image with a threshold (each pixel above the threshold will become white and the others black),
- saves the image with an appropriate postfix,
- parameters:
  - *input1*: path to the image file,
  - *input2*: absolute path of the output folder,
  - *input3*: threshold (if there is no 3rd input use the mean intensity level as threshold; to check the input parameters use **`varargin`**),
  - *output1*: binary image.

Create a script that:

- calls the previously written function with the appropriate parameters,
- displays the returned image.

Exercise 5

Create a function that:

- reads in an image,
- converts it to grayscale if necessary,
- creates a 10 pixel wide border for the image: create a matrix that is 20 pixel larger in both dimensions (use `ones()`), and place the image in the middle,
- saves the image with an appropriate postfix,
- parameters:
  - *input1*: path to the image file,
  - *input2*: absolute path of the output folder,
  - *output1*: array of the framed image.

Create a script that:

- calls the previously written function with the appropriate parameters,
- displays the returned image.

Exercise 6

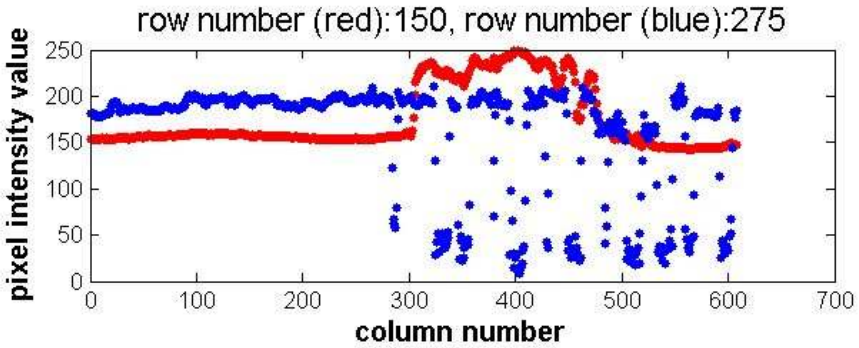
Create a function that:

- reads in an image,
- converts it to grayscale if necessary,
- displays the image in the upper **`subplot`** of a figure,
- plots the pixel intensity values of the *i-th row* of the image (use **`plot`** function) in the lower **`subplot`** of a figure,
- puts labels on the axis (**`xlabel`**, **`ylabel`**, **`title`**),
- saves the figure (**`saveas`**) with an appropriate postfix,
- parameters:
  - *input1*: path to the image file,
  - *input2*: absolute path of the output folder,
  - *input3*: value of *i* (which row to plot),
  - *input4*: optionally index for an other row --- plot this other row as well with a different color, use **`hold on/off`** and **`varargin`**,
  - no *output*.

Exercise 6

Create a script that:

- calls the previously written function with the appropriate parameters.



Exercise 7

Create a function that:

- reads in 2 images,
- converts them into the same size (**`imresize`**),
- creates a video of one image fading into the other --- use the following functions:

```
output_video = VideoWriter(out_video_path, 'Uncompressed AVI');
open(output_video);
…
writeVideo(output_video, out_frame);
…
close(output_video);
```

- parameters:
  - *input1*: path to the first image file,
  - *input2*: path to the second image file,
  - *input3*: output video path,
  - *input4*: number of frames to create.

At the generation of the individual frames, you have to think about the type-conversions (**`double`**, **`uint8`**).

Exercise 7

Create a script that:

- calls the previously written function with the appropriate parameters.