CSCI 1320 Computer Science I: Engineering Applications - Fall 2017
Instructor: Zagrodzki
Assignment 7
Due October 15th, by 11:55pm

Image processing – cont.

For Lab 6 and Assignment 7, you will create a simple image-processing program, with functions similar to those found in Adobe Photoshop or The Gimp. Most of the functions you implement for this assignment will take an input image, process the image, and produce an output image.

Note: it would be possible to achieve some of the functionality required in this assignment by using built-in Matlab functions, especially from a couple of specialized toolboxes. You are NOT allowed to use these image-specific built-in functions. You will need to code your own implementation of these functions.

Provided files:
A shell script *lab6_hwk7.m* is provided to get you started. Also, a collection of images is provided for testing. You can use your own images as well.

What You Have to Do for the Lab
Implement a menu driven program, similar to the one in the textbook pages 209-215. Each button should trigger a call to a function, just like the textbook example. To start with, your program has the following three buttons:

1. Load Image – loads one image file. In order to select one image, the file needs to be in the same folder as the main script. The loaded image will become the current image and can be passed as an input to other functions.

2. Display Image – displays the current image.

3. Exit Program – closes the menu and terminates the script.

You have to add and test additional functionality for the program. For every button/functionality you add, you can use one of the images to test it.

In Lab 6, you implemented the functions `makeBright_L` and `makeBright_NL`, and modified the menu to add a button for each one.

For Assignment 7, you need to implement the following functions and add menu buttons for each of them. (note: Assignment 7 is graded out of 150 points, not 100).

Task 1 (10 points) Invert: inverts the colors of an image. **Use Loops!**

```
function [ outImg ] = invert_L( inImg )
```

Color inversion, also known as the negative effect, is one of the easiest effects to achieve in image processing. Subtract each RGB color value from the maximum possible value (**255**) and the result is an inverted image. You must use loops to modify each individual pixel values.

Choosing the *Invert_L* menu button should result in:
a. Calling the `invert_L` function with the current image as an input
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 2 (5 points) Invert: inverts the colors of an image. **NO Loops!**

```
function [ outImg ] = invert_NL( inImg )
```

This function will achieve the same thing as `invert_L`, without the use of loops.

Choosing the *Invert_NL* menu button should result in:
a. Calling the `invert_NL` function with the current image as an input
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 3 (15 points) AddRandomNoise: adds random noise to an image. **Use Loops!**

```
function [ outImg ] = addRandomNoise_L( inImg )
```

The input image `inImg` has pixel values between 0 and 255 in all three channels R,G and B. This function adds random noise to each pixel and writes out the new image

to `outImg`. The random noise added should be different for every pixel and it should be in the range [-255, 255].

Choosing the *AddRandomNoise_L* menu button should result in:
a. Calling the `addRandomNoise_L` function, with the current image as input.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 4 (5 points) AddRandomNoise: adds random noise to an image. **NO Loops!**
`function [ outImg ] = addRandomNoise_NL( inImg )`
This function will achieve the same thing as `addRandomNoise_L`, but without the use of loops.

Choosing the *AddRandomNoise_NL* menu button should result in:
a. Calling the `addRandomNoise_NL` function, with the current image as input.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 5 (15 points) Luminance: change a color image into a luminance (gray scale) image. **Use Loops!**
`function [ outImg ] = luminance_L( inImg )`
This method transforms a color image into a gray image and writes out the new image to `outImg`. **Note:** here, the variable `outImg` will be a 2-dimensional matrix. You must use loops to modify each individual pixel values for `outImg`.

In order to convert a particular color into gray scale we need to figure out what the intensity or brightness of that color is. A quick way is to calculate the mean value of the red, green and blue channel components:

$$I = \frac{R + G + B}{3}$$

Although it can produce reasonable results, this method is not perfect. This is because the human eye does not perceive reds, greens and blues at the same intensity level. The color green, for example, at maximum intensity looks brighter than blue at maximum intensity.
For this Assignment we will use the following weighting system:

$$I = 0.299R + 0.587G + 0.114B$$

Given this image:

Converting to grayscale using the mean method:

… and using the weighted method:

Choosing the *Luminance_L* menu button should result in:
a. Calling the `luminance_L` function, with the current image as input.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 6 (5 points) Luminance: change a color image into a luminance (gray scale) image. **NO Loops!**

```
function [ outImg ] = luminance_NL( inImg )
```

This function will achieve the same thing as `luminance_L`, but without the use of loops.

Choosing the *Luminance_NL* menu button should result in:

a. Calling the `luminance_NL` function, with the current image as input.

b. Displaying the original image and the resulting image, side by side (use subplots)

c. Saving the resulting image.

Task 7 (20 points) Composite

```
function [ outImg ] = composite( inImg )
```

Create a composite image `outImg` that is two times as tall as the original image `inImg`, and two times as wide. Place the original image in the top left, the red layer in the top right, the green layer in the bottom left, and the blue layer in the bottom right parts of this composite image.

Choosing the *Composite* menu button should result in:

a. Calling the `composite` function, with the current image as input.

b. Displaying the original image and the resulting image, side by side (use subplots)

c. Saving the resulting image.

Task 8 (20 points) Color swap: replace pixels of one color with another color, with some leniency.

```
function [ outImg ] = color_swap( inImg, r1, g1, b1, r2, g2, b2, allowed)
```

This function will create a new image `outImg` by swapping pixels of one color (defined by `r1, g1, b1`) with another color (defined by `r2, g2, b2`). When we did this example in class, we noticed it is hard to find many pixels of purely one exact color. Use the input variable `allowed` for setting the leniency criteria. For example, for `r1=255, g1=0, b1=0` and `allowed` = 10, every pixel with a red value above 245 and green and blue values below 10 should be changed to the RGB values specified by `r2, g2, b2`).

Make sure the values entered by the user for `r1, g1, b1, r2, g2, b2, allowed` are within the boundaries for image intensities. Also, check that they are positive values. If they are not, display appropriate error messages and request the input of new values.

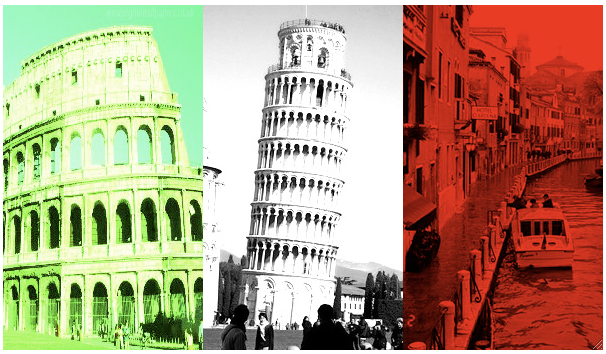
Choosing the *Color Swap* menu button should result in:
a. Asking the user to input values for `r1, g1, b1, r2, g2, b2, allowed`, followed by a call to the `color_swap` function, with the current image and the values of `r1, g1, b1, r2, g2, b2, allowed` as inputs.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.

Task 9 (25 points) Flag Collage: create a collage of pictures.

```
function [ outImg ] = flag( inImg1, inImg2,inImg3, ...)
```

Your collage will be made by copying at least 3 pictures onto a blank canvas. You must use at least 3 different pictures, as modified forms of original images. You can:
1) scale, crop, or rotate the image,
2) shift or alter colors in the image (invert, change contrast, extract just one layer),
3) make the image darker or lighter, or
4) combinations of any of these modifications.

Note: for changing the brightness or for getting a grayscale version of the original images, you can use the functions you developed for this assignments. For other things, like scaling and rotating, you can use built-in Matlab functions (**resize, rot90**). When finished, your collage should look like the flag of a country (examples below).



Note: You get to decide how you want to implement this task, how many inputs you want your function to have and how will you use other already developed functions. The function should have the original images as inputs, but whether you want to use any other input variable will be up to you, and you will need to modify the function header accordingly. (you might want to use an image of the flag as one of your inputs; this is allowed)

Choosing the *Flag Collage* menu button should result in at least the following:
a. Calling the `flag` function, with the original images as input values.
b. Displaying the resulting flag image.
c. Saving the resulting flag image.

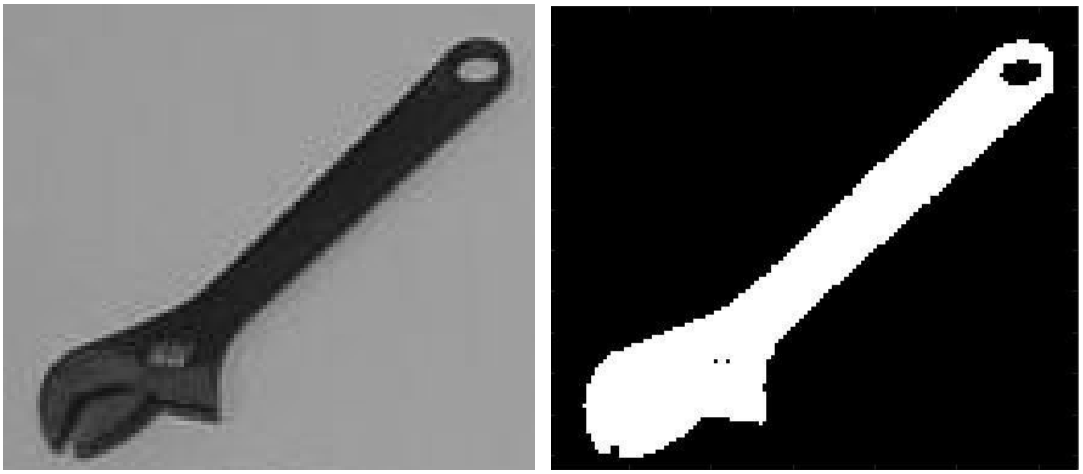
Task 10 (25 points) Binary image: similar to the blue-screening technique on TV and in movies

```
function [ outImg ] = binaryMask( inImg )
```

Create one binary image `outImg` (also known as *the mask*), which will represent the boundaries of the object of interest in the input image `inImg`. A binary image has only values of 0 and 1, so only black and white pixels, no grey shades (this means `outImg` is a 2-dimensional matrix).

To make this task easier, you only need to make this function work for one input image: *wrench1.jpg*. Your algorithm should be able to figure out the threshold value that separates the background from the "object of interest" in the image. The pixels in the binary image will have the value of 0 if they belong to the background, and 1 otherwise (if they are part of the wrench).

Choosing the *Binary Image* menu button should result in:
a. Calling the `binaryMask` function, with the current image as input.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Saving the resulting image.



Task 11 (35 points) Mean Filter: also known as smoothing, averaging or box filter

```
function [ outImg ] = meanFilter( inImg )
```

Mean filtering is a method of smoothing images, *i.e.* reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of mean filtering is simply to replace each pixel value in an image with the mean (average) value of its neighbors, including itself.

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

This has the effect of eliminating pixel values that are unrepresentative of their surroundings.
Note: Pay close attention to the pixels on the edge (first row, last row, first column, last column). How many neighboring pixels do they have? Use selection statements to address these special cases. (more details in lecture)

Choosing the *Mean Filter* menu button should result in:
a. Calling the `meanFilter` function, with the current image as input.
b. Displaying the original image and the resulting image, side by side (use subplots)
c. Save the resulting image.


Extra credit:
1. Notice that the sum of the tasks 1 through 11 is 180 points. Feel free to tackle any tasks you want and in any order you want. Anything above 150 points will be considered extra credit.
2. Flag Collage Competition. This task does not require a menu button. You can submit your flag collage image to be entered in the competition by submitting with your assignment an image entitled *flagCollage_<firstName>_<lastName>.jpg*. The top ten flag collages will receive **10 points**.

Submitting the assignment:
Make sure each script or function file is well commented and it includes a block comment with your name, course number, assignment number and instructor name. Save one resulting image for each of the buttons/functionality implemented. Zip all the .m files and the image files together and submit the resulting *.zip* file through Moodle as Assignment 7 by October 15th, 11:55pm.