CSCI 4830/5722 Computer Vision, Fall 2016
Instructor: Fleming
Assignment 1, Due Wednesday, September 14th, by 11:55pm

Early Vision – One Image

For Assignment 1, you will create a simple image-processing program, with functions similar to those found in Adobe Photoshop or The Gimp. 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 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. The convolution function (**conv**) is also not allowed. You will need to code your own implementation of these functions. Feel free however, to use these functions to check your results.

Provided files:
A shell script *vision_hwk1.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
Implement a menu driven program, where each button should trigger a call to a function. If you run the provided shell program, you will notice 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.

For Assignment 1, you need to implement the following functions and add menu buttons for each of them.

Task 1 (15 points) Mean Filter: also known as smoothing, averaging or box filter

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

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. This has the effect of eliminating pixel values that are unrepresentative of their surroundings.

The input argument `kernel_size` will determine the size of the smoothing kernel. For example, if `kernel_size` is 3, the smoothing kernel is of 3 x 3 size like in the picture below.

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

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 or pad the image with extra rows and columns (details in lecture, remember the NaN value!).

Choosing the *Mean Filter* menu button should result in:
a. Ask the user to input the size of the smoothing kernel (a positive integer)
b. Call the `meanFilter` function, with the current image as input, plus the value entered by the user for the size of the smoothing kernel.
c. Display the original image and the image returned by the function, side by side (use subplots)
d. Save the resulting image. Use a naming convention of your choice.

Task 2 (20 points) Gaussian Filter: smoothing filter, also known as Gaussian blur
`function [ outImg ] = gaussFilter( inImg, sigma )`
The Gaussian filter works in a similar fashion to the mean filter. The values of the weights in the Gaussian smoothing kernel will be different (as opposed to the mean filter where the weights were the same). They follow the Gaussian distribution:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

, where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. The size of the Gaussian kernel will be computed based on the value of `sigma` as follows:

```
2 * ceil ( 2 * sigma) + 1
```

Note: the origin is always at the (x,y) coordinates of the pixel you are trying to replace with a new, smoothed value. Since the size of the kernel will always be an odd number, for the calculation of the Gaussian weights the pixel in the middle of the kernel will be the (0,0) origin, and therefore have the highest weight.

Choosing the *Gaussian Filter* menu button should result in:
a. Ask the user to input a positive value for `sigma`

b. Call the `gaussFilter` function, with the current image as input, plus the value entered by the user for `sigma`.
c. Display the original image and the image returned by the function, side by side (use subplots)
d. Save the resulting image. Use a naming convention of your choice.

Task 3 (20 points) Scale Nearest

```
function [ outImg ] = scaleNearest( inImg, factor )
```

This method scales an image using nearest point sampling to obtain pixel values and returns the new image. The value of the input parameter `factor` is the factor by which the image is to be scaled.

Choosing the *Scale Nearest* menu button should result in:
a. Ask the user to input a positive value for `factor`
b. Call the `scaleNearest` function, with the current image as input, plus the value entered by the user for `factor`.
c. Display the original image and the image returned by the function, side by side (use subplots)
d. Save the resulting image. Use a naming convention of your choice.

Task 4 (20 points) Scale Bilinear

```
function [ outImg ] = scaleBilinear( inImg, factor )
```

This method scales an image using bilinear-interpolation to obtain pixel values and returns the new image. The value of the input parameter `factor` is the factor by which the image is to be scaled.

Choosing the *Scale Bilinear* menu button should result in:
a. Ask the user to input a positive value for `factor`
b. Call the `scaleBilinear` function, with the current image as input, plus the value entered by the user for `factor`.
c. Display the original image and the image returned by the function, side by side (use subplots)
d. Save the resulting image. Use a naming convention of your choice.

Task 5 (25 points) Fun Filter

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

This method applies a fun-filter to the current image and returns the new image. Warp an image using a non-linear mapping of your choice (examples are fisheye, sine, bulge, swirl). You can choose to have user input to specify which filter to apply, or just hard-code one filter in particular, indicating (in comments) which filter are you implementing and the mapping function used.

Choosing the *Fun Filter* menu button should result in:
a. Call the `funFilter` function, with the current image as input.
b. Displaying the original image and the image returned by the function, side by side (use subplots). *Here you can display the name of the filter used as well.*
c. Save the resulting image. Use a naming convention of your choice.

Note: You might want to implement two helper functions useful for tasks 3-5:

```
function [ value ] = sampleNearest( x, y)
```

This method returns the value of the image, sampled at position (x,y) using nearest-point sampling. https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation

```
function [ value ] = sampleBilinear( x, y)
```

This method returns the value of the image, sampled at position (x,y) using bilinear-weighted sampling. https://en.wikipedia.org/wiki/Bilinear_interpolation

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 1 by Wednesday, September 14th, by 11:55pm.