Submission before: 09.11.2015
Discussion on: 10.11.2015
Submission on stud.ip, submission folder for sheet.
Please submit a zip file containing the .m files for Matlab programming tasks.

Organizational issues
All assignments can and should be prepared and submitted in groups of 3 persons.
Please sign up for a feedback session group on Stud.IP!
You will be admitted to the final exam for the computer vision lecture if you receive at least 50% in $N - 2$ of the sheets.

Exercise 1 *(Convolution – 8p)*

Develop a function that applies a local operator. Do not use any specialized Matlab functions like `conv2` or `imfilter` for this task. Proceed with the following steps:

(a) Implement a function that applies an $(2m + 1) \times (2n + 1)$-kernel to an image patch of the same size, using the convolution formula from the lecture.

(b) Implement a function that applies the kernel to the whole image, i.e. loop over all pixels of the image and apply step (a). You may pad the boundary with zeros to avoid problems. Demonstrate your function by applying a Gaussian filter kernel to the image `building.jpg` (you can get a Gaussian kernel from the Matlab function `fspecial('gaussian',SIZE,VAR)` or use the binomial approximation from the lecture).

(c) Now extend that function to cover the whole image, i.e. care for the boundary. You may implement whatever strategy you like (just ignoring the boundary is not an option).

(d) The Gaussian filter kernel is separable. As mentioned in the lecture, this allows for a more efficient computation. Try the effect for different kernel sizes using the Matlab functions `tic` and `toc`. You may use binomial filters as an approximation instead of the Gaussian kernel (the function `nchoosek` can be used to compute binomial coefficients).

Hints: (1) You may convert images to `double` to avoid range problems. (2) You may want to pass a function `myfunc` as an argument to another function. This can be done with the Matlab notation `@myfunc`.

Exercise 2 *(Edge detection – 4p)*

(a) Apply simple gradient filters, Prewitt operators and Sobel filters to the image `car.jpg`. Briefly describe in your comments, what these filter do.

(b) Apply the Laplace filter and calculate "zero-crossings" in the result. A zero-crossing is a point which differs in sign from its neighbors. It is sufficient to compare a pixel to its right and lower neighboring pixel. Apply the function to the image `street.png`. Then introduce a threshold to ignore zero-crossings with very small differences.

Hint: You may use your convolution function from exercise 1 or some Matlab function like `imfilter`, or `conv2` here.

Exercise 3 *(Nonlinear smoothing – 4p)*

Introduce Gaussian and salt-and-pepper noise to some image of your choice. You may use the Matlab function `imnoise` to do so. Then implement and apply the following nonlinear smoothing filters:

(a) Min and Max filters.

(b) Median filter (you may use the Matlab function `sort`).

(c) The symmetric-nearest-neighbor filter (SNN).

Hint: You may again reuse functionality of exercise 1 here.

Exercise 4 *(Morphological operators – 4p)*

(a) Implement dilation and erosion (you may again reuse code from exercise 1, but do not use Matlab functions like `imerode` or `imdilate` here). Apply your functions to `note.png`.

(b) Use erosion to implement a boundary function (you may use `imerode` now). Then iterate erosion to compute a distance transform. Display your result for `leaf.png`.

(c) Implement a function for morphing beteween two images based on a distance transform (you may use the Matlab function `bwdist` to compute the distance transform). Acquire two binary images A and B (by whatever means you like) and apply your morphing function to generate and display a stream of images that morphs from A to B.

Hint: the Matlab function `all` and `any` may be useful in this exercise.