# Image Interpretation – Assignment 2

This assignment on **Features** covers the labs from the 26th of September. The exercises on this sheet are graded by a maximum of 10 points. You will be asked to implement several functions and write a brief report.

In order to submit the results, send a ZIP file with your implemented code (functions prefixed with `ii_` and any "helper" functions you wrote) to

`<mikhail.usvyatsov@geod.baug.ethz.ch>`

with subject

`Image Interpretation 2019 Assignment 2`

no later than on the

`2th of October, 2019.`

Your functions should work when called by the provided test code (functions prefixed with `test_`) which **must not** be modified. When run, they should produce a plausible output, no warnings, and no unnecessary output (remember to end your statements with semicolons). Eventual example output is given in the `ref_`-images. The functions that you write take images as arguments, not image filenames (i.e. do not use `imread` inside the functions). Your functions should not generate figures/plots themselves, the plots are generated by the test scripts.

In addition to the functions, include a brief report (max. 4 pages PDF) explaining the structure of the code and the MATLAB functions used. This includes the reasons for choosing particular functions as well as a short justification of their parameter setting. For the more complicated tasks, the choice of the underlying data structures and algorithms should be explained too. We encourage you to add also diagrams, illustrations, and figures into the report when appropriate, but it is not necessary to copy the related theory from the lecture slides. The report should not contain any code snippets (but the code should contain comments if appropriate).

**Team work is not allowed**. Everybody implements his/her own code and writes his/her own report. Discussing issues with others is fine, sharing code and/or report with others is **not**. If you use any code fragments found on the Internet, make sure you reference them properly.

**Exercise 1.** 2 P.

Gaussian smoothing is often needed when extracting multi-scale features. Implement function `ii_gaussian` that takes two arguments: an image $I$ and the size of the Gaussian kernel $s$ (which is guaranteed to be odd). The function returns the smoothed image of the same size as the input image.

**Explain what should the standard deviation of the Gaussian to fit nicely in the kernel of size $s$ and why.**

You can test your function with the help of `test_gaussian`, which produces 4 images, 3 of which are smoothed with Gaussian filters of different sizes.

Helpful MATLAB functions: `fspecial`, `imfilter`

**Exercise 2.** 4 P.

Edges inside images contains most of the information of the scenes being captured, that's why they are one of the most used features when interpreting images.

1. Implement function `ii_deriv` that compute the derivative of an image. It that takes 3 arguments: a (grayscale) image $I$, and the orders of derivatives in $x$ and $y$ directions. The returned derivative image has the same size as the input image. The first-order horizontal derivative filter should be a [-1 0 1] filter. Similarly, the first order vertical derivative filter corresponds to its transposed version.

2. Implement function `ii_sobel` that computes a better derivative operator on an image by first computing the derivative and then applying the gaussian filter of the specified dimention. It that takes 4 arguments: a (grayscale) image $I$, the order of derivative in $x$ and $y$ directions and the amount of gaussian blur. The returned edge image has the same size as the input image.

You can test your function with the help of `test_deriv1`, `test_deriv2`, and `test_sobel` which each produce images using different parameter values. If you wonder how to create the filters, recall that convolution is a commutative and associative operation.

**Check and explain if it is better to apply the gaussian filter on the derivative function to extract edges of an image**?

**Exercise 3.** 2 P.

Variance in pixel neighborhood is a feature useful for urban area detection. Implement function `ii_variance` that takes two arguments: a (grayscale) image $I$ and the (odd) size of the window $s$. The returned image has the same size as the input image: For each pixel, the variance in an $s \times s$ neighborhood should be computed. You can use `test_variance` to show an image and the result after

filtering. Can you see the urban area? Take special care here not to introduce boundary artifacts: No high variance should be detected at image boundary.

Helpful MATLAB functions: `padarray`, `colfilt`

**Can you see the urban area? Why? Is it better to compute the variance on the grayscale image or the color channels separately? Show the variance of different channels and motivate your answer.**

**Exercise 4.** 2 P.

Histograms can be used either as texture features or as robustifiers of more complicated (gradient-based) features. Implement function `ii_hist` that takes two arguments: a (grayscale) image $I$ and the number of levels $n$. The function then returns image histogram with $n$ bins that are evenly spaced over the whole theoretical value range (use e.g. `im2double` to get a standardized range of [0 1]). Test your function on input `(0:1/255:1,4)`. Did it correctly output `[64 64 64 64]'`? You can use `test_hist` to perform additional testing.

Helpful MATLAB functions: `linspace`, `imquantize` (do **not** use `hist`, `imhist`)