Homework 2

Introduction to Image Analysis

Handout: 5th April 2023,
Handin: 10th May 2023, 13:00

Instructions

Your hand-in will consist of a `.zip` archive named `hw2_firstName_lastName.zip` containing the following:

- Python source files named `hw2_exY_firstName_lastName.py`, where Y is the exercise number.
- All necessary files to run the above.
- Your report named `hw2_firstName_lastName.pdf`.

Your archive must be uploaded on ILIAS before the deadline.

Code [44 points]

Code templates are provided to help you get started in solving the exercises. In the typical case, you will fill in the missing function bodies and code blocks. Note that you may also make your own code from scratch.

IMPORTANT: In general, if you are not able to produce a functional code (i.e. your script crashes), comment out the concerned part, and if necessary, give a short analysis in your report. Scripts that do not run will be penalized.

Report [30 points]

Along with the code, you are asked to provide a short report. Comment on all the questions on the report (both theory and coding), show your results, and briefly explain what you did to obtain them. If you encountered problems and did not manage to finish the exercise, explain here what you did. On ILIAS you will find a LaTeX template to get started. Note that the use of LaTeX is NOT mandatory.

1 Linear Filtering [13 points]

1.1 [2 points]

Write a function that returns a box filter of size $[n \times n]$. Make sure the filter values sum up to 1. If you are using the provided template, you may fill in the `boxfilter(n)` function.

HINT: This can be done as a simple one-line function.

1.2 [5 points]

Implement a 2D convolution function between an image $[m \times n]$ and a filter $[k \times l]$. DO NOT USE any built-in convolution functions (such as from scipy or numpy). You should code your own version of the convolution, valid for both 2D and 1D filters. If you are using the provided template, you may fill in the `myconv2(image,filt)` function.

IMPORTANT: The function should return a full convolution, meaning the output should be of size $(m + k - 1) \times (n + l - 1)$. Use any border-filling approach you choose to ensure values at the border. For example, you could assume 0 for values outside the image (zero padding).

1.3 [2 points]

In your script, create a box filter of size 11 and convolve this filter with your image. Show your result.

1.4 [4 points]

Write a function that returns a 1D Gaussian filter for a given value of sigma. The filter should be a vector of length `filter_length` if `filter_length` is odd, `filter_length + 1` otherwise. Each value of the filter can be computed from the Gaussian function, $e^{-\frac{x^2}{2\sigma^2}}$, where x is the distance of an array value from the center. This formula for the Gaussian ignores a constant factor, so you should normalize the values in the filter so they sum to 1.

If you are using the provided template, you may fill in the `gauss1d(sigma, filter_length)` function.

HINTS: For efficiency and compactness, it is best to avoid "for" loops. One way to do this is to first generate a vector of values for x, for example, `[-3 -2 -1 0 1 2 3]` for a filter with length 7. These can then be used to calculate the Gaussian value corresponding to each element.

2 Finding edges - [6 points]

2.1 [2 points]

Begin by defining a derivative operator of your choosing. For example, `dx = [-1, 0, 1]`. Using your functions from Exercise 1, convolve `dx` with a Gaussian filter with sigma = 1. Repeat the same for `dy`.

2.2 [4 points]

Using `dx` and `dy`, construct an edge magnitude image. That is, for every pixel in the image, assign the magnitude of gradients. Also calculate for each pixel the gradient orientation. Display the magnitude image next to the original one. If you are using the provided template, you may fill in the `create_edge_magn_image(image, dx, dy)` function.

3 Corner detection - [25 points]

For this exercise, you can either use your functions from exercise 1, or `scipy.signal.convolve()`.

3.1 [15 points]

Write a function that computes the harris corner for each pixel in the image. The function should return the R response at each location of the image.

If you are using the provided template, you may fill in the `myharris(image, w_size, sigma, k)` function.

HINT: You may have to play with different parameters to have appropriate R maps. Try Gaussian smoothing with `sigma = 0.2`; Gradient summing over a 5x5 region around each pixel and `k = 0.1`.

3.2 [2 points]

Evaluate your function from the previous question on the chessboard image. Show the result.

3.3 [2 Points]

Repeat the same as 3.2, but this time rotate the chessboard image by 45 degrees (in either direction). Show the result.

You can use the built-in `scipy.ndimage.rotate()` function.

3.4 [2 Points]

Repeat the same as 3.2, but this time downscale the chessboard image by a factor of a half. Show the result.

You can use the built-in `scipy.misc.imresize()` function.

3.5 [4 Points]

Looking at the results from 3.2, 3.3, and 3.4, what can we say about the properties of Harris corners? What is maintained? What is it invariant to? Why is that the case?

Answer this question in your report.