Homework 2

Introduction to Signal and Image Processing

Handout: 20th, March 2019
Handin: April 03rd, 2019, 14:15

Instructions

The following homework is an opportunity for you to have hands on experience manipulating and working with images and the material covered in class.

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 [70 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, show your results, including figures, 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 https://www.tug.org/texlive/LaTeX template to get started. Note that the use of LaTeX is NOT mandatory.

1 Linear Filtering - [25 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]

Write a function which performs a 2D convolution between "image" $[m \times n]$ and "filt"$[k \times l]$. This function should return the result of a 2D convolution of these two images. 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.

1.3 [2 Points]

In your script, create a boxfilter of size 11 and convolve this filter with your image. Show this 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 sigma of 1. These can then be used to calculate the Gaussian value corresponding to each element.

1.5 [2 Points]

Write a function that returns a 2D Gaussian filter. Remember that a 2D Gaussian can be formed by convolution of a 1D Gaussian with its transpose. Use your function from 1.2 to perform the convolution.

Display a plot using sigma of 3.

If you are using the provided template you may fill in the `gauss2d(sigma, filter_size)` function.

1.6 [3 Points]

Define a function that applies Gaussian convolution to a 2D image for the given value of sigma. Do this by first generating a filter with gauss2d, and then applying it to the image with myconv2 (image, filter).

Run your gconv on the image with a sigma of 3 and display the result.

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

1.7 [3 Points]

Convolution with a 2D Gaussian filter is not the most efficient way to perform Gaussian convolution with an image. In a few sentences, explain how this could be implemented more efficiently and why this would be faster.

Answer this question in yours report.

HINT: How can we use 1D Gaussians?

1.8 [4 Points]

To show this fact create a plot of filter size vs. computation time. In this plot, show in one color (e.g. blue) what this size-time relation is when using 2D box filter of increasing size (3,100) with a step of 5. Repeat the same for your solution to (1.7) and show it in another color.

2 Finding edges - [20 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. Write a function that performs this operation.

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.

2.3 [6 Points]

You will now produce edge images of particular directions. An edge image is a binary image. In this case, you will produce 8 such images. For $i = 0, \ldots, 7$, an edge image e_i should have values 0 except if a pixel has an edge of magnitude greater than threshold, r = 3 and has an orientation in the interval $(\frac{2i\pi}{8} - \frac{\pi}{8}, \quad \frac{2i\pi}{8} + \frac{\pi}{8})$.In this case, the pixel should have value 255.

Please implement a function that performs this operation.If you are using the provided template you may fill in the `make_edge_map(image, dx, dy)` function.

Show the result of these edge images by concatenating all of them with the original and magnitude edge image.

HINT: To verify if you are doing this correctly, try with an image with 1 values everywhere except in a circle filled with 0 at the center of the image. You should generate edge images that cover the extent of the perimeter of the circle.

2.4 [8 Points]

Implement edge non-max suppression on the magnitude edge image, as in canny edge detection, in order to reduce the thickness of the edge response you observe.

Please implement a function that performs this operation. If you are using the provided template you may fill in the `edge_non_max_suppression(img_edge_mag, edge_maps)` function.

Display the result showing the original image, the magnitude edge image and the result of the non-maximum suppression.

2.5 [5 Points - BONUS]

Implement edge hysteresis with a lower and upper threshold to complete a full canny edge detector (e.g. combining steps from (2.2) and (2.4)).

Please implement a function that performs this operation. If you are using the provided template you may fill in the `canny_edge(image, sigma)` function.

Show the result.

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 which 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.