Exercise 1 - Canny Edge Detector

The goal of this exercise is to familiarise yourself with edge detection. You will implement the *Canny edge detector*, one of the most common edge detectors. A code construct and example images are provided in a zip file which can be downloaded here:

Download Files For Exercise 1

Use these files as templates for your program and refer to the comments in the files for details on expected inputs and outputs.

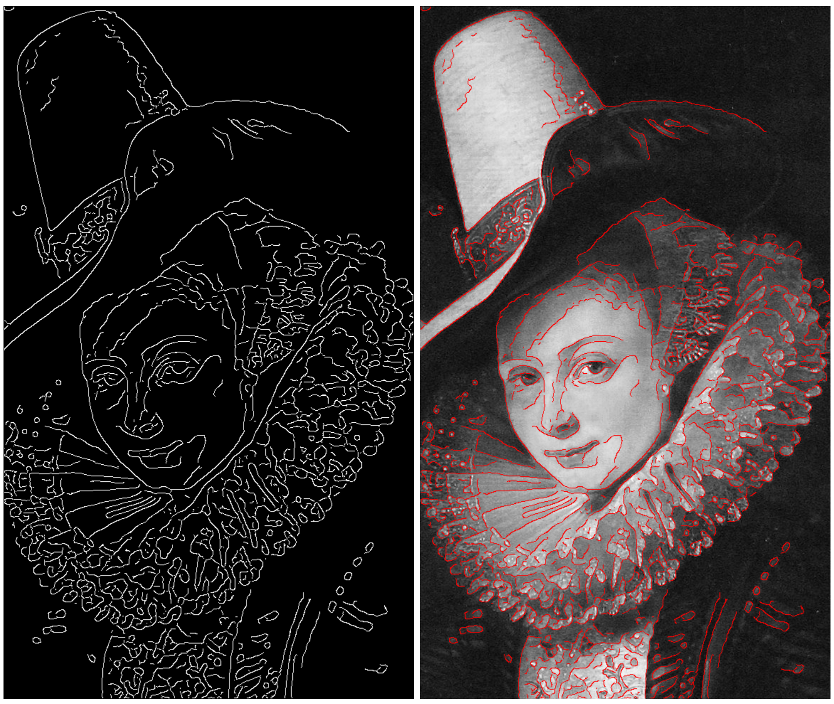

Figure 1: Resulting edge image (left), original image overlaid with detected edges (right)

Part 1: Implementation (10 Points)

You should write code so that it can be understood by someone else (the tutors, for example). Comment sections and lines which might not be easy to understand. If you've thought about a line or section for more than a couple of minutes, it probably warrants a comment. Use descriptive variable names. Try to follow the Python style guide.

Your code should not take too long to run. For the first few exercises, anything more than a few seconds likely means you are doing something very inefficient. Points may be deducted for slow code, even if your solution is correct. You should not need to import any additional libraries. If you import an additional library for some reason, please write an explanation. It might lead to point deduction. It is difficult to name each function you are not allowed to use because especially OpenCV is very powerful and has a lot of the algorithms we will implement in this course already integrated. As a general guideline, if a task can be solved with one function call, you probably should not use it. If you are unsure, please ask in the TUWEL forum.

Complete the function skeletons provided, implementing edge detection on a grayscale image using the Canny edge detector.

The algorithm consists of 4 main parts:

1. Image blurring (2 point)

Blur the image with a Gauss-filter. First, create a 2-dimensional NumPy array which has the values of the Gaussian kernel as defined in the lecture slides. The dimensions of the kernel should be variable depending on the sigma. The width of the square filter kernel should be calculated as follows:

```
kernel_width = 2 * round(3 * smooth_sigma) + 1
```

For the actual filtering use the OpenCV function `cv2.filter2D`. Look at the arguments of the `cv2.filter2D` function and think especially about a reasonable border type.

Complete the code in **blur_gauss.py**. For details on the function and the expected return values refer to the docstring in the file.

2. Edge detection (2 points)

Calculate the horizontal and vertical gradient of the image using a Sobel filter, transposing it as appropriate. Again, define the filter kernel as a `np.array` and apply the function `cv2.filter2D`. Make sure that before you do the Image has the data type `np.float32` and is in the range [0,1].

You may find `circle.jpg` useful for testing whether your gradient and orientation results are what you expect.

Complete the code in **sobel.py**.

3. Non-maximum suppression (4 points)

Implement the non-maximum suppression algorithm along the orientation of the gradients to thin out the edges. You may wish to quantise the gradient orientations to discrete levels like "horizontal", "vertical", "diagonal-left", "diagonal-right" first and then treat each case separately. E.g. if a pixel has a "horizontal" gradient orientation, its gradient value should only be kept if its immediate left and right neighbor pixels have a lower gradient value than the center pixel. If there are multiple pixels with the same maximum value next to each other, choose one arbitrarily.

Complete the code in **non_max.py**. Make sure you explain your approach in comments.

4. Hysteresis thresholding (1 points)

Implement hysteresis thresholding to receive connected edge segments. We recommend you to use the functions `np.where` and `cv2.connectedComponents` to reduce programming effort. After identifying connected edges, you can use `np.isin` to check each edge if it includes a pixel with `value > high`.

To make parametrisation easier you should normalise the gradient values to lie within the interval [0, 1] before applying the hysteresis thresholding. Before applying `cv2.connectedComponents` you will need to convert it to [0,255] and `np.uint8`.

Complete the code in **hyst_thresh.py**.

5. Automatic hysteresis threshold selection (1 points)

The hysteresis thresholds can be difficult to select manually, as you will probably discover if you run the code on various images. It's not clear from looking at an image what thresholds you should use. Implement an automatic threshold selection function which calculates these thresholds and then calls the `hyst_thresh` function. The function expects two arguments `low_prop` which is the proportion of pixels that should lie above the low threshold, and `high_prop` which is the proportion of pixels above the high threshold. This means `low_prop > high_prop`.

Complete the code in **hyst_thresh_auto.py**

Forbidden and allowed commands

Python commands `cv2.Sobel`, `cv2.Canny`, `cv2.GaussianBlur`, `cv2.getGaussianKernel` , `np.gradient` must not be used. However, the commands `cv2.filter2D` and `np.arctan2` , along with other block-processing and sliding-window functions may be used.

Unless explicitly stated, you should not use functions which solve an exercise, or a significant part of it, in a single line of code. If you are unsure of whether you are allowed to use a function, please contact the tutors.

Advice

Since NumPy is a matrix-oriented library, "vectorised" processing is much more efficient than loops. Pictures should be converted to floating point format before further processing. Also make sure that your starting image only contains RGB values in the uint8 range from 0 to 255. Keep your programs as simple as possible. Catching wrong user inputs and printing error messages is not necessary. The file `helper_functions.py` contains a few functions which might be helpful to plot intermediate results.

If your code is slow, you can measure which function takes up how much time of the total runtime. Pycharm offers a Profile functionality to do so.

Useful commands

`cv2.filter2d` - Filtering with linear filter kernel

`cv2.imread` - Load an image

`cv2.imshow` - Display an image (for RGB image, value format is uint8 with range [0, 255], for grayscale, value format is double with range [0, 1])

`cv2.waitKey` - Needs to be called after imshow.

`cv2.cvtColor` - Image color conversion (e.g. to grayscale)

`cv2.connectedComponents` - Label connected components with a specific label

`.astype(np.uint8)` - Can be called on np.arrays to convert them to 8-bit unsigned integer format

`.shape` - Shape of an NumPy array

`np.max` - Find maximum

`np.sort` - Sort an array

`np.isin` - Return a boolean array of the same shape as the input array with True for elements which are the same in both arrays.

`np.square` - Elementwise square

`np.sqrt` - Elementwise square-root

`np.exp` - Elementwise exponential function with Euler's number as base

Part 2: Documentation (6 Points)

Perform the following experiments and answer the questions in a few sentences.

- Answer each numbered question separately, with the answer labelled according to the experiment and question number. For example, experiment 1 question 2 should be numbered 1.2. You do not need to include the question text.
- Include images to support your answers. These should be referenced in the answers. The document should include information about what parameter settings were used to generate images so that they can be reproduced. Feel free to use your own images but if you do so, upload the original image as well so we can reproduce your results.
- Your answers should attempt an explanation of why an effect occurs, not just describe what you see in the supporting images.
- Be precise. "It is better" or similarly vague statements without any additional information are not sufficient.

Points may be deducted if your report does not satisfy the above points.

Only change the parameters in the file main_ex1.m in the marked places. We recommend creating a separate Python file for your experiments (e.g. `experiments.py`) to not mess up the main file. If you do so, also upload this file because it makes reproducing your experiment results easier for the tutors.

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

Experiment 1 (2 points)

Investigate the effect of `sigma` of the Gaussian blurring.

1. Why is the kernel width defined as a function of sigma? Try various width and sigma inputs for the `cv2.filter2d` function and examine the results. Does an incorrect width have an effect in practice?
2. What effect does blurring have on pixel intensities in the image? You might find the function `plot_row_intensities` from the `helper_functions.py` file useful to document your findings.
3. What is its effect on the edges you receive after non-max suppression?

4. How does it affect the edges you are left with after the hysteresis thresholding, assuming thresholds are kept constant?

Experiment 2 (2 points)

Investigate the effect of thresholds used during hysteresis thresholding.

1. Change the parameters `low` and `high` in the `hyst_thresh` function. What effects do these values have on the result?
2. You might notice that keeping thresholds the same over different images can give inconsistent results (try it out). Why is this? Can you think of situations in which you might wish to keep the thresholds the same?
3. Does the automatic selection help with getting reasonable edges across various images? Why?
4. Can you think of a situation where the automatic selection method implemented may not work well? How might one mitigate this issue?

Experiment 3 (2 points)

Investigate the effect of noise on the results. You can add noise to an image with the function `add_gaussian_noise` included provided in the `helper_functions.py` file. Add gaussian noise with zero mean and try standard deviations in the range [0.01, 0.5].

1. What effect does the noise have on the resulting edges?
2. Is it possible to set the parameters to get reasonable edges? How does changing `sigma` and the hysteresis threshold values affect the results?

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL discussion forum to discuss your problems.
2. Email for questions or in-person help requests: machinevision@acin.tuwien.ac.at

Upload

Please upload all Python files and your documentation (pdf-format) **as one ZIP-file** via TUWEL.