Feature Detection

In this exercise, you will implement the corner and edge detection algorithm based on the paper `A Combined Corner and Edge Detector` by Harris et al. from 1988.

If you are working on your own computer, install `line_profiler` for the test-cases:

```
pip3 install line_profiler # --user if you have no root privileges
```

1 Harris Corner Response

First, the Harris Corner Response is computed for every pixel of the input image. The value of this function is then used in task 2 and 3 to extract corners and edges. Everything for this task has to be implemented in the function `compute_harris_response` of `utils/functions.py`.

1. Compute an approximation of the first spatial derivatives I_x and I_y in x and y direction respectively, by using filters and store the results in two variables `Idx` and `Idy`. You can use the OpenCV function `Sobel`.

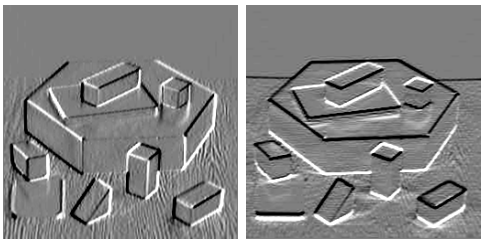

Figure 1: I_x, I_y, displayed with zero as gray; black is negative, white is positive.

2. Compute the mixed products I_{xx}, I_{yy}, I_{xy} for auto-correlation with

$$I_{xx} = I_x^2$$
$$I_{yy} = I_y^2$$
$$I_{xy} = I_x I_y$$

   and store the results in variables `Ixx`, `Iyy`, and `Ixy`. Make sure to use element-wise multiplication.

3. Convolve the mixed products with a zero mean Gaussian G with $\sigma = 1$:

$$A = I_{xx} \circledast G$$
$$B = I_{yy} \circledast G$$
$$C = I_{xy} \circledast G$$

   Use the OpenCV function `GaussianBlur` and save the result to variables `A`, `B`, `C`.

Figure 2: A and B are rendered with zero as black, C is rendered with zero as gray.

4. For each pixel, construct the structure tensor T and compute the Harris response R with

$$T = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

$$R = \mathrm{Det}(T) - k \cdot \mathrm{Trace}(T)^2$$

with $k = 0.06$.

Figure 3: The response is displayed with zero as gray; black is negative, white is positive.

2 Corner Detection

Given the Harris response function R, stable corner points can be extracted by searching for local maxima and thresholding. Implement the function `detect_corners` that detects a key-point for a pixel (x, y), if the following two conditions are met:

- $R(x, y) > t_h$, with $t_h = 0.1$
- $R(x, y)$ is a local maximum of R in the 1-neighborhood of (x, y) (8 neighbors in total).

Depending on your exact implementation, you might have a slightly different sensitivity. The effect of large variations in sensitivity can be seen in the following figure.

This function can be written using vectorization (see the tutorial from the previous exercise sheet) without any loops or OpenCV functions, with simple numpy operations and functions instead. Try to adjust your code if you haven't already done so.

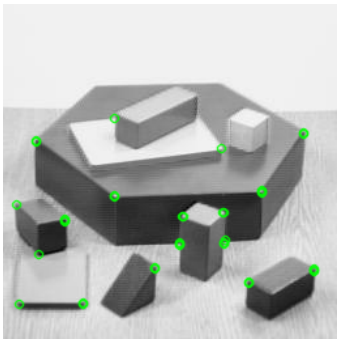

Figure 4: Desired output for the corner detection.

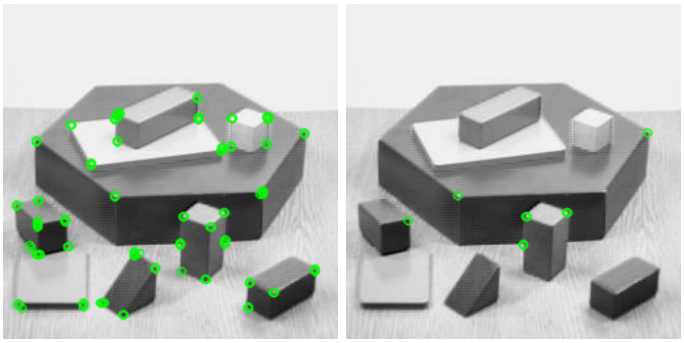

Figure 5: Over- and underestimations produced by a threshold of 0.01 and 0.3.

3 Edge Detection

Similar to the corner detection, implement the function `detect_edges` to create a boolean image, marking the edge pixels which meet the following conditions:

- $R(x, y) \leq t_e$, with $t_e = -0.01$.
- $R(x, y)$ is a local minimum in x **or** y direction.

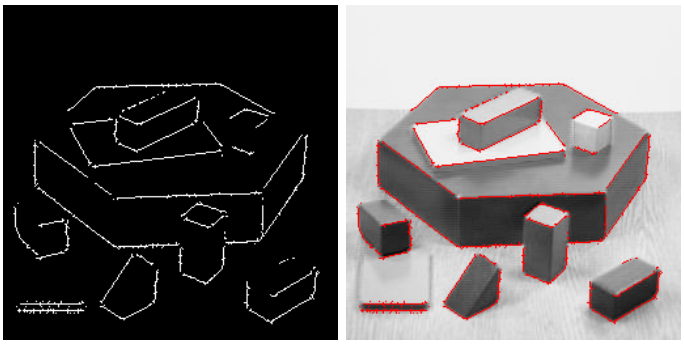

Figure 6: Detected edges.

Again, this function can be written using vectorization without any loops or OpenCV functions, with simple numpy operations and functions instead. Try to adjust your code if you haven't already done so.