Angelos Nikolaou
Prathmesh Madhu
03.05.2021

The deadline for this exercise is on Monday 24.05.2021, 23:59

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 from 1988.

When you are finished with this exercise,

- Compress the complete directory into a ZIP and upload it to StudOn. Make sure folder *results/ex1* exists in the project root. On Unix (and powershell) systems, the following command generates the submission archive

```
cd ComputerVision2021 (or CV2021);
# as a single line:
zip ../submission_ex1.zip ./data/ex1/input.jpg ./bin/run_ex1.py
./ex1/*py test/test_ex1.py ./results/ex1/*png
```

- For groups, one member uploads the submissions and adds their partner to the exercise. Don't forget to add all members of the team in *ex1/__init__.py*
- For the test-cases to work correctly you will also need to install line_profiler.

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

- **At the start of every section, if you see (x + y) %, then you will receive the x% for correct logic and y% for vectorized code implementation of the same. If and when you see this, there will always be test cases provided for you to test your vectorized code too!**

1 Harris Corner Response [(28 + 5)%]

At 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 `ex1/functions.py`.

1. Compute an approximation of first spatial derivative in x and y direction (I_x and I_y respectively) using filters and store the results in `Idx` and `Idy`. You can use the OpenCV function `Sobel`.

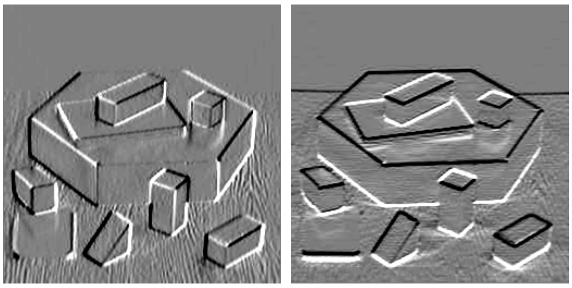

Figure 1: Gradients are displayed with zero as gray, black is negative, white is positive.

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 `Ixx`, `Iyy`, and `Ixy`. Make sure to use element-wise multiplication.

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

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

Use the OpenCV function `GaussianBlur` and store the result in `A`, `B`, and `C`.

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

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

$$T = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$
$$R = \mathrm{Det}(T) - k\ \mathrm{Trace}(T)^2$$

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

2 Corner Detection [(23 + 10)%]

Given the Harris response function R, stable corner points can be extracted by searching for local maximas and thresholding. Implement the function `detect_corners` of `ex1/functions.py` that detects a key-point for a pixel (x, y), if the following two conditions are met:

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

Use provided *ex1.draw_points* to render the points.

Depending on your exact implementation, you might have slightly different sensitivity. Large variations in sensitivity can be seen in the following figure.

This function can be performed in a vectorized manner without any loops or opencv functions, with simple numpy operations and functions. Please have a look at hints.

3 Edge Detection [(29 + 5)%]

Similar to the corner detection, implement the function *detect_edges* of `ex1/functions.py` to create boolean image marking edge points by checking the following conditions:

- $R(x, y) \leq t_e$, with $t_e = -0.01$.

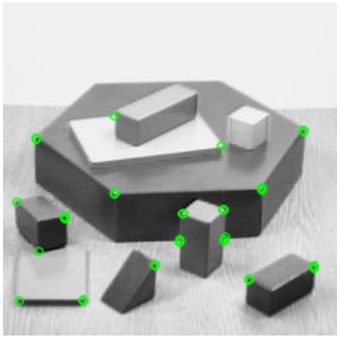

Figure 4: Desired Output

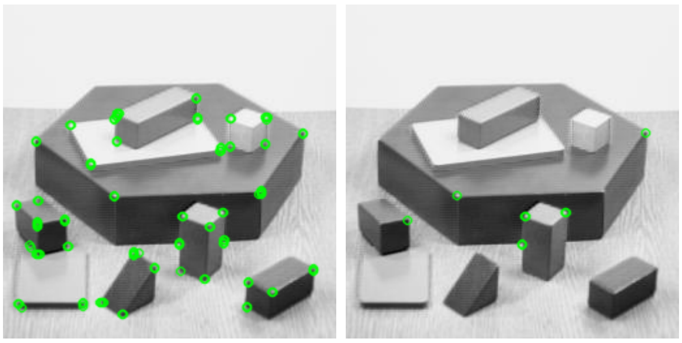

Figure 5: Under and Over estimations produced by a threshold of .01 and 0.3 respectively

- $R(x, y)$ is a local minimum in x **or** y direction.

Use the provided `ex1.draw_mask` to visualize the edges:

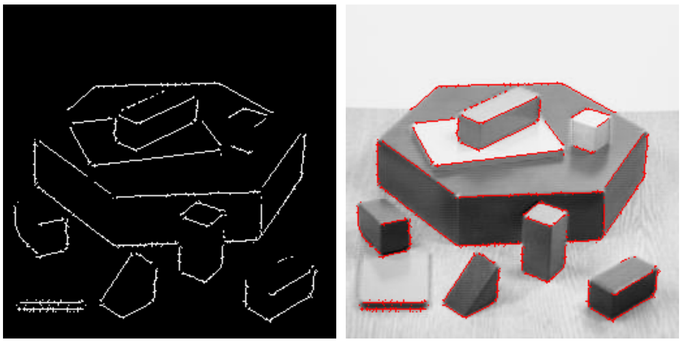

Figure 6: Detected edges.

This function can be performed in a vectorized manner without any loops or opencv functions, with simple numpy operations and functions. Please have a look at hints.

4 Vectorization Hints

Vectorization is crucial for a proper execution of the exercise. Vectorization in numpy, matlab, pytorch etc. can described as removing loops over large ranges, thus delegating massive computation to the underlying high performance implementations provided by algebra frameworks.

A few rules to remember:

- Compute all functions in a formula in to Images.
- Offset images instead of pixels.
- Use padding when using offset images.
- Compose stacks of images with neighbors as a third dimension.
- Useful numpy commands: np.pad, np.minimum, np.max, np.logical_or, np.logical_and, and np.concatenate