Exercise 1: Optical flow

Advanced Computer Vision Methods

2019/2020

In this exercise you will implement two of the most well known optical flow estimation methods, the Lucas-Kanade method and the Horn-Schunck method. Note that there are two Python functions attached to the material for the exercise that you can use: `gausssmooth` performs Gaussian smoothing of an image and `gaussderiv` computes first and second Gaussian derivatives of an image.

Submission instructions

The exercise should be submitted on-line on the course website. The submission should contain a report and the source code. Do not submit the data which was given as a part of the instructions, except you add some data that you collect from other sources. The report is the most important part of the submission, so make sure that you spend enough time on it, after you are done with coding and experiments. Note that a strict page limit of the report is **two pages maximum**. Detailed description of grading can be found at the end of the document.

The submissions should be done by the deadline. Late submissions are possible, however, a strict deadline is one week after the first one. A baseline for the late submission is 70%. After that you cannot submit the exercise anymore. The assistant will review the submissions and provide a feedback within a week after the deadline (unless stated otherwise). Each exercise must be done individually and all submissions will be checked for plagiarism. A student will be notified about the grade in the submission feedback. Passing all four exercises is required to pass this part of the course.

Assignment 1: Lucas-Kanade optical flow

As you have learned, Lucas-Kanade method assumes that the optical flow is very similar in the local neighborhood. This assumption is then used to define a system of equations that can be solved using the least squares method. The details of this process were explained at the lectures, here we only briefly mention the last stage of the process that defines the displacement vectors u and v for the given input images I_1 and I_2 as

$$u = -\frac{\sum_{\mathcal{N}} I_y^2 \sum_{\mathcal{N}} I_x I_t - \sum_{\mathcal{N}} I_x I_y \sum_{\mathcal{N}} I_y I_t}{D}, \quad v = -\frac{\sum_{\mathcal{N}} I_x^2 \sum_{\mathcal{N}} I_y I_t - \sum_{\mathcal{N}} I_x I_y \sum_{\mathcal{N}} I_x I_t}{D},$$

where $\mathcal{N}$ denotes the neighborhood of the pixel (usually a 3×3 pixel region), I_x, I_y denote the two spatial derivatives (the pixel-wise average image derivatives of the first and the

second image in x and y direction), and I_t denotes the temporal derivative $I_2 - I_1$. The D is a determinant of a covariance matrix that is defined as

$$D = \sum_{\mathcal{N}} I_x^2 \sum_{\mathcal{N}} I_y^2 - (\sum_{\mathcal{N}} I_x I_y)^2.$$

Implement a simple (non-pyramidal) Lucas-Kanade optical flow estimation method based on the equations above. Do it without any explicit loops. Hint: you can do this by replacing the summations by a convolution with an uniform kernel of the size of the neighborhood.

Implement function `lucaskanade` that accepts two images and the size of the neighborhood as an input and returns two matrices of the same size as the input image that contain u and v components of the optical flow displacement vectors.

```
def lucaskanade(im1, im2, N)
# im1 - first image matrix (grayscale)
# im2 - second image matrix (grayscale)
# n - size of the neighborhood (N x N)

# TODO : the algorithm
```

Assignment 2: Horn-Schunck optical flow

The basic idea of the Horn-Schunck algorithm is based on a global smoothness constraint. For given input images I_1 and I_2 the resulting formulas for displacement vectors u and v are defined iteratively as

$$u = u_a - I_x \frac{P}{D}, \qquad v = v_a - I_y \frac{P}{D},$$

where the u_a and v_a are the iterative corrections to the displacement estimate, defined by convolving the corresponding component with a "residual Laplacian kernel"

$$u_a = u * L_d, v_a = v * L_d, \quad L_d = \begin{bmatrix} 0 & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 \end{bmatrix}.$$

The I_x and I_y denote the pixel-wise average image derivatives for the first and the second image in x and y direction and the P and D terms are defined as

$$P = I_x u_a + I_y v_a + I_t, \qquad D = \lambda + I_x^2 + I_y^2,$$

where I_t denotes the time derivative $I_2 - I_1$. The initial estimates for u and v are typically set to 0 and are then iteratively improved.

Implement the Horn-Schunck algorithm in Python without explicit `for` loops (except the iteration loop). Write a function `hornschunck` that accepts two gray-scale images, a parameter λ and a number of iterations. The function then returns two matrices of the same size as the input image that contain u and v components of the optical flow displacement vectors.

```
def hornschunck(im1, im2, n_iters, lmbd)
# im1 - first image matrix (grayscale)
# im2 - second image matrix (grayscale)
# n_iters - number of iterations (try several hundred)
# lmbd - parameter

# TODO
```

Visualize the results of the algorithm using the `showflow` function. First test your methods using the snippet following below that uses a synthetic image of a white noise that is rotated one degree around the center. The expected result of the algorithm is shown in Figure 1.

```
import numpy as np
import matplotlib.pyplot as plt

from ex1_utils import rotate_image, show_flow
from of_methods import lucas_kanade, horn_schunck


im1 = np.random.rand(200, 200).astype(np.float32)
im2 = im1.copy()
im2 = rotate_image(im2, -1)

U_lk, V_lk = lucas_kanade(im1, im2, 3)
U_hs, V_hs = horn_schunck(im1, im2, 1000, 0.5)

fig1, ((ax1_11, ax1_12), (ax1_21, ax1_22)) = plt.subplots(2, 2)
ax1_11.imshow(im1)
ax1_12.imshow(im2)
show_flow(U_lk, V_lk, ax1_21, type='angle')
show_flow(U_lk, V_lk, ax1_22, type='field', set_aspect=True)
fig1.suptitle('Lucas-Kanade Optical Flow')

fig2, ((ax2_11, ax2_12), (ax2_21, ax2_22)) = plt.subplots(2, 2)
ax2_11.imshow(im1)
ax2_12.imshow(im2)
show_flow(U_hs, V_hs, ax2_21, type='angle')
show_flow(U_hs, V_hs, ax2_22, type='field', set_aspect=True)
fig2.suptitle('Horn-Schunck Optical Flow')

plt.show()
```

Grading

The tasks marked with *Req.* are required to successfully complete the exercise. The number in the brackets represents number of points of other tasks while *Add.* stands for additional tasks which can bring you more than 100 points.

- (Req.) Implement both, Lucas-Kanade and Horn-Schunck optical flow methods and compare the results on the rotated random noise images.
- (15) Test both methods on other images (at least 3 more pairs of images), include results in the report and comment them. You can use images included in the project material or (even better) add your own examples.
- (10) How can we determine where the Lucas-Kanade optical flow can not be estimated reliably? Incorporate this idea into your function `lucaskanade`.
- (15) Which parameters need to be determined for Lucas-Kanade and Horn-Schunck? How do these parameters impact on optical-flow performance? Show examples and discuss your observations which parameters are optimal in which cases.
- (10) Measure time for Lucas-Kanade and Horn-Schunck optical flow methods and report measurements. Can you speed-up Horn-Schunck method, e.g., by initializing it with output of Lucas-Kanade? What is the speed and performance of the improved Horn-Schunck?
- (Add.) Implement pyramidal Lucas-Kanade. Show examples where pyramidal implementation performs better than non-pyramidal. Does running Lucas-Kanade iteration multiple times on the same scale improve performance?

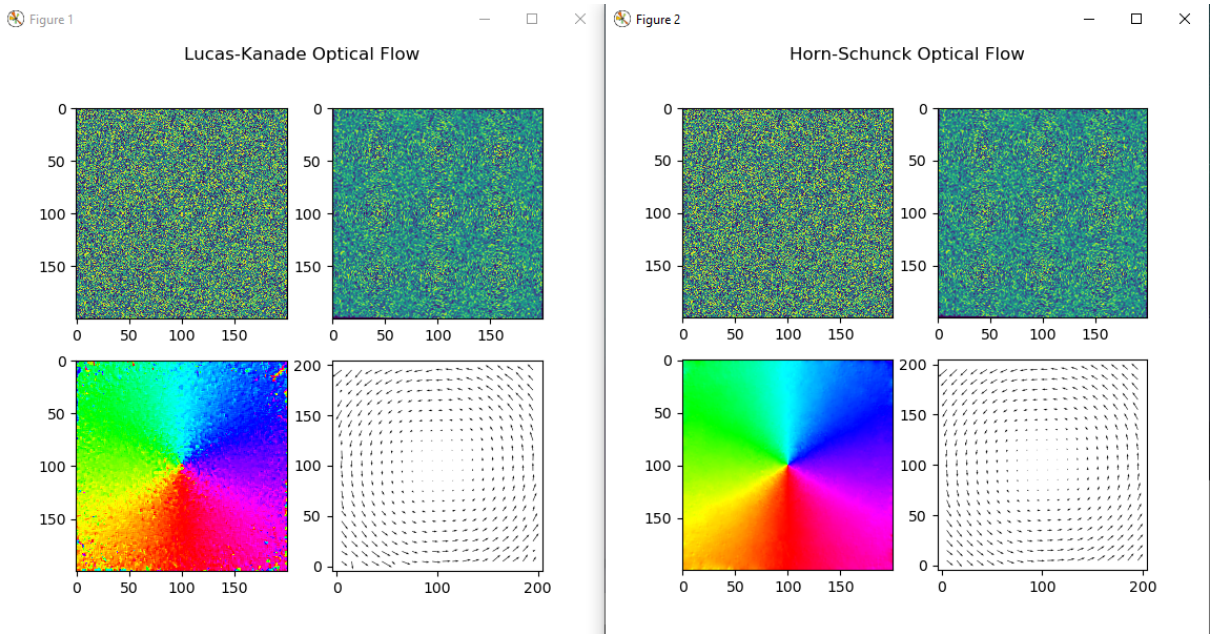

Figure 1: An example of expected result for the Lucas-Kanade and Horn-Schunck method.