Computer Vision I
Assignment 5 (Bonus Assignment)

Prof. Stefan Roth
Krishnakant Singh
Shweta Mahajan

07/02/2022

This assignment is due on February 21st, 2022 at 23:59.

Please refer to the previous assignments for general instructions and follow the handin process described there.

Problem 1: Estimating Optical Flow (20 Points)

In this task, we will estimate optical flow (OF) using Lucas-Kanade method. We will also implement the iterative coarse-to-fine strategy, discussed in the lecture, and compare it to our first basic implementation.

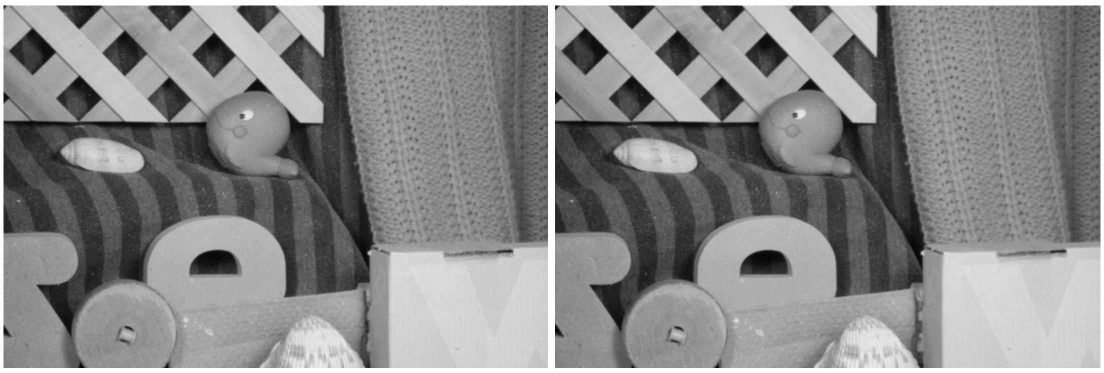

Figure 1: A pair of images for estimating the optical flow.

Recall from the lecture the following system of linear equations we are trying to solve:

$$\begin{aligned}\sum_R (u \cdot I_x + v \cdot I_y + I_t) I_x = 0,\\ \sum_R (u \cdot I_x + v \cdot I_y + I_t) I_y = 0,\end{aligned} \tag{1}$$

where (u, v) is the unknown flow vector for the centre pixel in region R; I_x, I_y and I_t are the image derivatives with respect to x, y and t; and the summation is over an image patch R of a pre-defined size. This implies that I_x, I_y and I_t should be read as scalar values per pixel in R. Observe that every summand in Eq. (1) is weighted equally. We will return to this observation in Task 8.

Tasks:

Our basic implementation of Lucas-Kanade will contain the following functions:

1. Implement function `compute_derivatives` that computes image derivatives I_x, I_y and I_t as defined in the lecture. Note that the returned values should have the same size as the image.

   (3 points)

2. Implement function `compute_motion` that computes (u, v) for each pixel given I_x, I_y and I_t as input. For now, ignore the extra function arguments `aggregate` and `sigma`.

   (3 points)

3. Implement function `warp` that warps the given (first) image with the provided flow estimates. *Hint:* You may find function `griddata` from package `scipy.interpolate` useful.

   (3 points)

4. We will need to verify that our flow estimates indeed minimise the objective. Recall the cost function minimised by Lucas-Kanade method and implement it in `compute_cost`.

   (2 points)

For coarse-to-fine estimation, we first need to build the Gaussian pyramid. You are welcome to re-use the functions below if you already implemented them in one of the previous assignments. Note, however, that you may need to re-adjust them according to the function specification below.

5. Implement function `downsample_x2` that downscales the input image by a factor of 2. To avoid aliasing, make sure to apply smoothing with a Gaussian kernel of the size and σ provided as the arguments to the function.

   (1 point)

6. Implement `gaussian_pyramid` that returns a list of images in the pyramid in the ascending order of resolution. The number of levels is specified by the argument `nlevels`.

   (2 points)

We are now ready to put things together for iterative coarse-to-fine estimation. Recall that every iteration comprises a sequence of OF estimation starting from the coarsest and proceeding to the finest resolution. To transition to the finer scale, upscale the flow field first and then warp the image. Keep in mind that scaling the flow grid will require corresponding scaling of the flow magnitudes.

7. Implement function `coarse_to_fine` that computes optical flow using the coarse-to-fine strategy. Note that you should use the number of iterations provided as the argument for the stopping criterion.

   (4 points)

8. Recall from Task 2 that function `compute_motion` has extra arguments that we have not used yet. The summation of pixel values in patch R can also be weighted, that is the pixels that are farther away from the centre pixel in the patch will have smaller contribution to the sum than the pixels closer to the centre pixel. This intuitive idea suggests a Gaussian weighting: the weights in the patch will correspond to the weights in the Gaussian kernel. Modify your implementation of `compute_motion` to accept `aggregate='gaussian'` and `sigma=[float]` that sums the patch values in this fashion. You are welcome to use the provided `gaussian_kernel` to construct the patch weights.

   (2 points)