Computer Vision I
Assignment 4

Prof. Stefan Roth
Jannik Schmitt
Jan-Martin Steitz

01/02/2021

This assignment is due on February 14th, 2021 at 23:59.

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

Problem 1: Eight-point algorithm (15 Points)

In this problem, you will use the Eight-point algorithm to compute the fundamental matrix between two images.

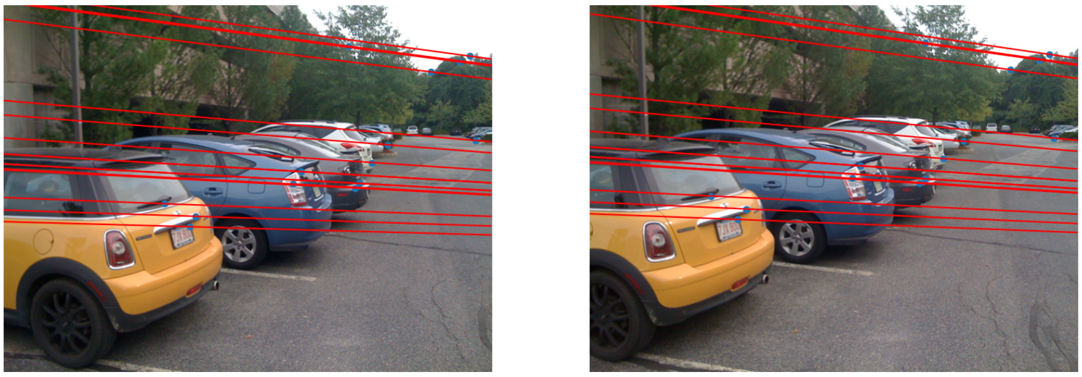

Figure 1: Epipolar lines for an image pair.

The main function `problem1` already loads the images and point correspondences. Write a function `eight_point`, which takes the correspondences in homogeneous coordinates as arguments, and outputs the fundamental matrix. The function performs the following steps:

- Condition the image coordinates numerically using the given function `condition_points` that also returns the conditioning matrix.
- From the conditioned image points, compute the actual fundamental matrix in `compute_fundamental` by first building the homogeneous linear equation system for the elements of the fundamental matrix, and then solving it using singular value decomposition.

  (3 points)
- The preliminary fundamental matrix is not yet exactly of rank 2, due to noise and numerical errors. Enforce the rank constraint using SVD in `enforce_rank2`. This function should be called in `compute_fundamental`.

  (2 points)
- These functions are called by `eight_point` to first condition the coordinates and then compute the fundamental matrix with rank 2. Transform it to obtain the one for the original pixel coordinates.

  (3 points)

We will now check, if the epipolar constraint is fulfilled by the fundamental matrix and find the epipolar lines and epipoles.

- First, draw the epipolar lines by implementing the function `draw_epipolars`. Given an array of coordinates in one image, it computes the coordinates where the corresponding epipolar lines intersect the left and right image border in the other image. This information is then used by `plot_epipolar` to generate an image similar to Fig. 1.

  (3 points)
- Second, verify that the epipolar constraints are (approximately) satisfied for all pairs of corresponding points by computing the maximum and average of the absolute value of remaining residuals $|p_1^\top F p_2|$ in `compute_residual`.

  (2 points)
- Finally, compute the cartesian coordinates of the epipoles in both images in `compute_epipoles`.

  (2 points)

Submission: Please include only `problem1.py` in your submission.

Problem 2: Estimating Optical Flow (17 Points)

In this task, we will estimate optical flow (OF) using the 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.

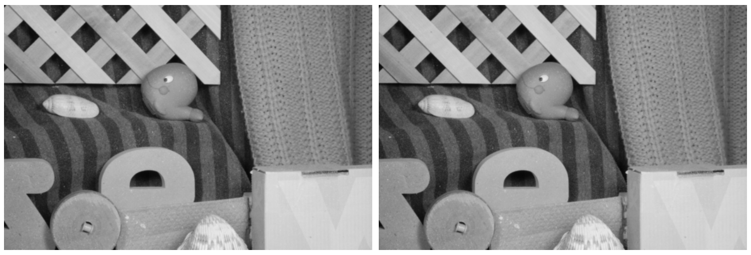
Figure 2: A pair of images for estimating the optical flow.

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 w.r.t. x, y and t; and the summation is over an image patch R of a pre-defined size, i.e. I_x, I_y and I_t should be read as scalar values per pixel in R.

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.

   (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 the Lucas-Kanade method and implement it in `compute_cost`.

   (2 points)

For coarse-to-fine estimation, we provide you with the functions `gaussian_pyramid` and `resize`. 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.

   (6 points)

Submission: Please include only `problem2.py` in your submission.