Computer Vision I
Assignment 4

Prof. Stefan Roth
Krishnakant Singh
Shweta Mahajan

TECHNISCHE UNIVERSITÄT DARMSTADT

10/01/2022

This assignment is due on Jan 24th, 2022 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 in `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 returns the conditioned image points and 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 the resulting fundamental matrix back to the original pixel coordinates.

(3 points)

We will now check if the fundamental matrix fulfils the epipolar constraint 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)

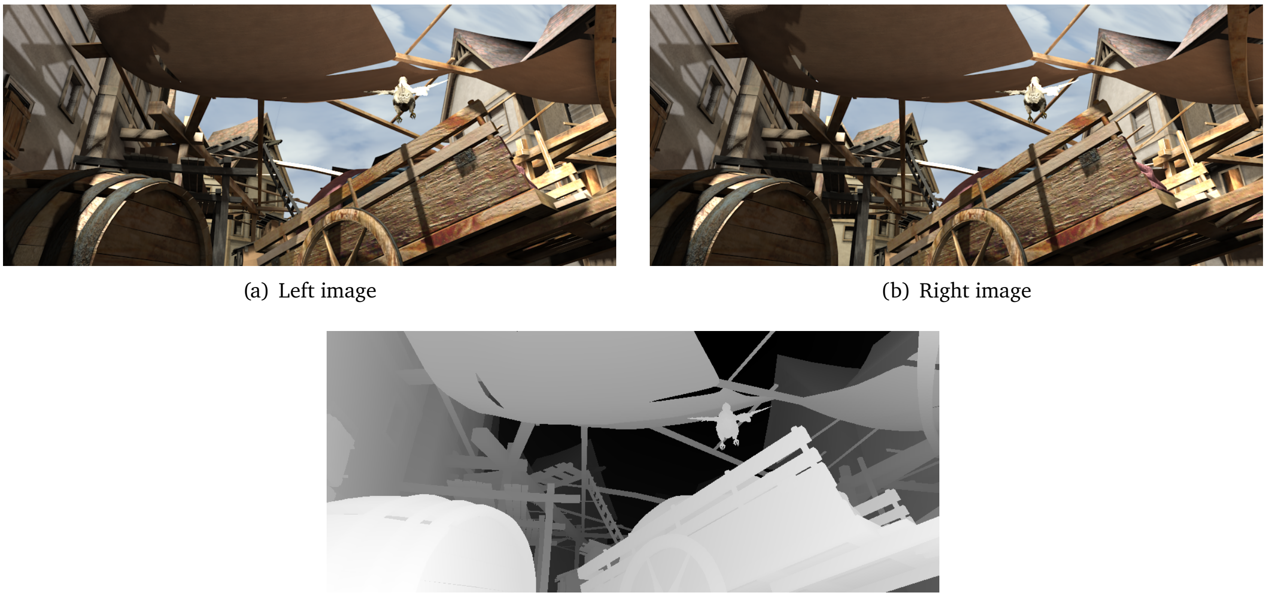
- Finally, compute the cartesian coordinates of the epipoles in both images in `compute_epipoles`.

(2 points)

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

Problem 2: Window-based Stereo Matching (15 Points)

In this problem, we will perform stereo matching by estimating a disparity map between two front-parallel images. As described in the lecture slides, we will try the window-based stereo matching method. Given the two rectified images, we estimate the disparity of each pixel along the horizontal scan-line by comparing the cost between two window patches.



(a) Left image

(b) Right image

(c) Disparity map (Ground Truth)

Figure 2: Estimating the disparity map between the two front-parallel images

As a cost function, we will use a weighted sum of two cost functions, SSD (Sum of Squared Differences) and NC (Normalized Correlation):

$$f_{\text{cost}}(x, y, d) = \frac{1}{m^2} * \text{SSD}(x, y, d) + \alpha * \text{NC}(x, y, d), \tag{1a}$$

with

$$\text{SSD}(x, y, d) = \sum_{(x',y') \in w_L(x,y)} (I_L(x', y') - I_R(x' - d, y'))^2 \tag{1b}$$

$$\text{NC}(x, y, d) = \frac{(\mathbf{w}_L(x, y) - \bar{\mathbf{w}}_L(x, y))^T(\mathbf{w}_R(x - d, y) - \bar{\mathbf{w}}_R(x - d, y))}{|\mathbf{w}_L(x, y) - \bar{\mathbf{w}}_L(x, y)||\mathbf{w}_R(x - d, y) - \bar{\mathbf{w}}_R(x - d, y)|}, \tag{1c}$$

where w_L and w_R is a $m \times m$-sized image patch from the left and right image respective, $\mathbf{w}_L$ the reshaped vector of w_L with the size of $m^2 \times 1$, and α is the weighting factor. The details of each cost function are described in the lecture slides.

Implement the two cost functions. Your first task is to implement the two cost functions, SSD (Sum of Squared Differences) and NC (Normalized Correlation). The input of each function is two $m \times m$ image patches from the left and right image, respectively, and the output is the scalar value of the calculated cost.

1. `cost_ssd`: Implement the SSD cost function in Eq. (1b).

   (1 point)

2. `cost_nc`: Implement the NC cost function in Eq. (1c).

   (1 point)

3. `cost_function`: Implement the cost function (*i.e.*, Eq. (1a)) that calls the two functions, `cost_ssd` and `cost_nc`, and returns their weighted sum specified by α.

   (1 point)

Compute per-pixel disparity. Compute the disparity map by using the window-based matching method.

4. *Boundary handling*: To have the same size of window for pixels near the image boundary, the boundary handling needs to be properly done by padding images. Implement the function `pad_image` that inputs an image and outputs a padded image with given the input padding width. An additional parameter here is the name of the padding scheme, which can take one of three values: `"symmetric"`, `"reflect"`, or `"constant"`. In the case of `"constant"` assume zero padding.

   (2 points)

5. *Compute disparity*: Implement function `compute_disparity` that calculates per-pixel disparity map between two input images after padding (*i.e.*, `padded_img_l` and `padded_img_r`), given the maximum disparity range (`max_disp`), the window size (`window_size`), and the alpha (α). To calculate the cost, call the cost calculation function(`cost_function`) inside the function `compute_disparity`.

   (4 points)

6. *Evaluate the result*: Implement function `compute_epe` that calculates the average end-point error (EPE) between the ground truth d_{gt} and the estimated disparity d, where the end-point error is defined as $AEPE(d_{gt}, d) = \frac{1}{N}\sum \|d_{gt} - d\|_1$, where N is the number of pixels.

   (1 point)

7. Experiments with different settings of α. Try values $\{-0.06, -0.01, 0.04, 0.1\}$ and return the value of alpha (from this set) with the minimum EPE in function `optimal_alpha`.

   (1 point)

8. *Multiple choice questions*: By changing the input window size and the padding schemes, have a close look at the estimated disparity map and its average end-point error. Then, answer the multiple-choice questions by returning the appropriate option in the function WindowBasedDisparityMatching

   (4 points)

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