Computer Vision I
Assignment 4

Prof. Stefan Roth
Jan-Martin Steitz
Dustin Carrión

TECHNISCHE UNIVERSITÄT DARMSTADT

08/01/2024

This assignment is due on Jan 21st, 2024 at 23:59.

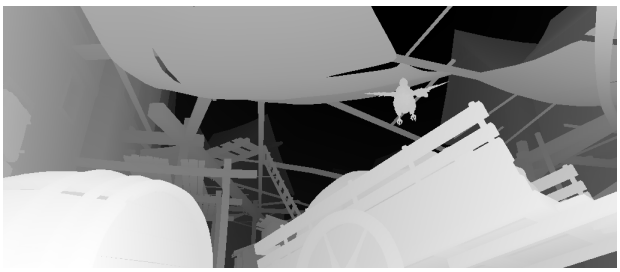
Please refer to the previous assignments for general instructions and follow the handin process described there.

Problem 1: 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 1: 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 are $m \times m$-sized image patches from the left and right image respectively, $\mathbf{w}_L$ and $\mathbf{w}_R$ are the vector versions of w_L and w_R of size of $m^2 \times 1$, and α is the weighting factor. The details of each cost function are described in the lecture slides.

Tasks:

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 window size for pixels near the image boundary, the boundary handling needs to be properly done by padding images. Thus, implement the function `pad_image` that inputs an image and outputs a padded image with the appropriate padding width based on a specified window size. An additional parameter of this function 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 the 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 α (`alpha`). To calculate the cost, call the cost calculation function (`cost_function`) inside the function `compute_disparity`.

   (4 points)

6. *Evaluate the result*: Implement the function `compute_aepe` that calculates the average end-point error (AEPE) 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 $\{-0.001, -0.01, -0.1, 0.1, 1, 10\}$ and return the value of alpha (from this set) with the minimum EPE in function `optimal_alpha`. For this question you have to use `max_disp` $= 15$, `window_size` $= 11$, and `padding_mode` $=$ `"symmetric"`.

   (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 `window_based_disparity_matching`.

   (4 points)

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

Problem 2: Estimating Optical Flow (10 Points)

In this task, we will estimate optical flow using the Lucas-Kanade method.

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

As explained in the lecture, we want to solve the following system of linear equations:

$$\begin{aligned} \sum_R (u \cdot I_x + v \cdot I_y + I_t) I_x = 0, \\ \sum_R (u \cdot I_x + v \cdot I_y + I_t) I_y = 0, \end{aligned} \tag{2}$$

where (u, v) is the unknown flow vector for the center pixel in the 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 values per pixel in R.

Tasks:

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

1. Implement the 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.

   (4 points)

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

   (3 points)

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