Assignment 4 – Computer Vision 1 – WS19/20 16/12/2019

Prof. Stefan Roth
Junhwa Hur
Nikita Araslanov
Xiang Chen

This assignment is due on January 20th, 2020 at 13:00.

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

Problem 1 - Estimating Fundamental Matrix (15 points + 10 bonus points)

In this task, you will estimate a fundamental matrix from a pair of images. Recall that given corresponding coordinates $\mathbf{x}_1$ and $\mathbf{x}_2$, the fundamental matrix satisfies

$$\mathbf{x}_1^T F \mathbf{x}_2 = 0. \tag{1}$$

You will work with the image pair shown in Figure 1. For your reference, we also provide an image pair from the well-known *corridor* sequence along with select point matches.

Figure 1: A pair of images for estimating the fundamental matrix.

Tasks:

1. Implement `transform` that returns a transformation (3×3 matrix) that normalises the point coordinates as discussed in the lecture.

   [2 points]

2. Implement function `transform_pts` that transforms the input points with a given transformation matrix.

   [1 point]

3. As we have seen in the lecture, the fundamental matrix F can be found as a solution to a homogeneous least-squares problem, i.e. $A\mathbf{x} = 0$, subject to $\|x\| = 1$, where $\mathbf{x}$ is a vectorised representation of F. Implement function `create_A` that constructs this matrix A.

   [3 points]

4. Recall that the fundamental matrix F has rank 2. Implement function `enforce_rank2` that enforces this constraint.

   [1 point]

5. Implement function `compute_F` that computes the fundamental matrix from the given equation system specified by A. Remember to make use of `enforce_rank2` to ensure that F has rank 2.

   [2 points]

6. Implement function `denorm` that reverts the normalisation we performed previously in matrix F.

   [1 point]

7. To verify our solution, we can take a look at the residuals. Let us define $g = \frac{1}{N}\sum_i |\mathbf{x}_{1i}^T F \mathbf{x}_{2i}|$ that measures L1 error of our solution F. Implement `compute_residual` that computes this measure for the point correspondences and matrix F.

   [1 point]

8. Combine the functions you have just implemented in function `estimate_F`. The function takes two arrays of corresponding image coordinates, the normalisation function and returns the fundamental matrix and the residual g.

[2 points]

9. We can also visualise the epipolar lines to inspect how reasonable our estimate of F is. The epipolar line on image 2 corresponding to point $\mathbf{x}_1$ in the first image can be found as $l_2 = Fx_1$. We provide most of the code for visualisation in function `show_epipolar` included in `utils.py`. Your task is to implement function `line_y` that computes y-coordinates of the epipolar lines. Please, consult the `show_epipolar` source code to understand how `line_y` is used.

[2 points]

Bonus Tasks:

1. Recall from the lecture that normalisation of points is important for estimation of F. Implement an alternative normalisation scheme in function `condition_T_v2` proposed by Hartley [1]. In summary, the transformed points should satisfy the following criteria:
   - The centroid of the points is $(0, 0)$.
   - The average distance from the origin is $\sqrt{2}$.

[2 points]

2. *Miltiple Choice Question:* Please, complete method `answer` in class `MultiChoice` which discusses various aspects about the fundamental and essential matrices.

[3 points]

3. Implement function `compute_epipole` that computes eipoles from the fundamental matrix.

[2 points]

4. Implement function `intrinsics_K` that returns the intrinsic parameters. For this task, we will use *toys* image pair. The focal length, scaled by the largrst image size is $f_x = f_y = 1.05$. You can assume the image center as the coordinate of the principal point.

[1 point]

5. Implement function `compute_E` that computes the essential matrix from the fundamental matrix and the camera parameters returned by `intrinsics_K`.

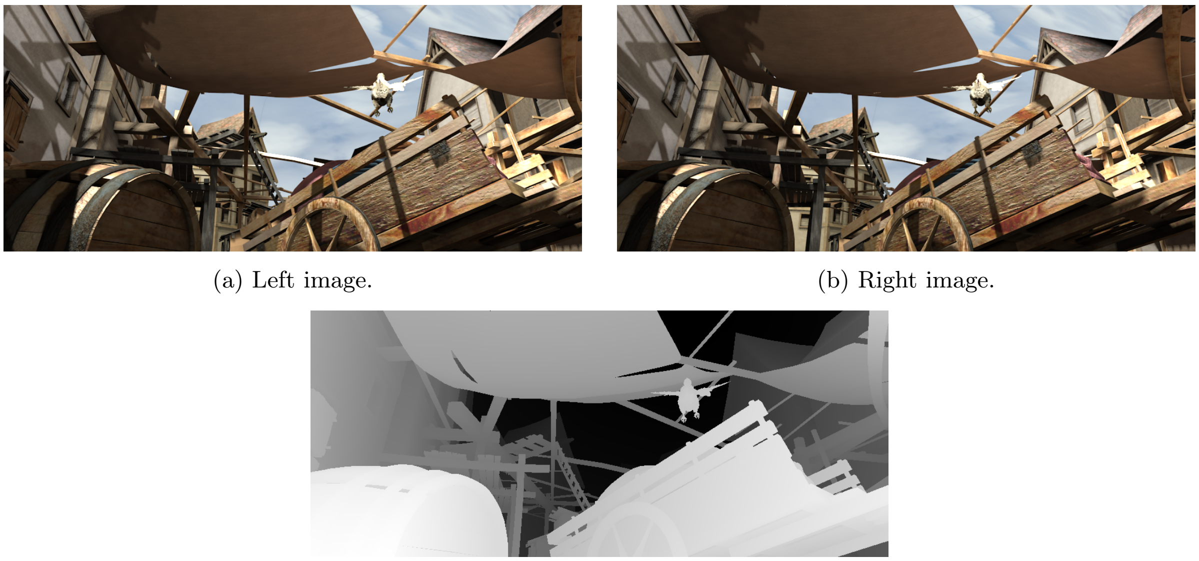
[2 points]

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

[1] Richard I. Hartley. "In Defense of the Eight-Point Algorith." TPAMI 1997.

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 are going to 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) Ground truth of the disparity map.

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):

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

with

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

$$\text{NC}(x, y, d) = \frac{(\mathbf{w}_L(x, y) - \bar{\mathbf{w}}_L(x, y))^T(\mathbf{w}_R(x - d, y) - \bar{\mathbf{w}}_R(x - d, y))}{|\mathbf{w}_L(x, y) - \bar{\mathbf{w}}_L(x, y)||\mathbf{w}_R(x - d, y) - \bar{\mathbf{w}}_R(x - d, y)|}, \tag{2c}$$

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 also 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 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. (2b).

   [1 point]

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

   [1 point]

3. `cost_function`: Implement the cost function (*i.e.*, Eq. (2a)) 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 (`pad_width`). An additional parameter is the name of the padding scheme, which can take one of the 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 of the function `compute_disparity`.

[4 points]

6. *Evaluate the result*: Implement function `compute_epe` that calculates an average end-point error (EPE) between the ground truth d_{gt} and the estimated disparity d, where the end-point error is $AEPE(d_{gt}, d) = \frac{1}{N} \sum \|d_{gt} - d\|_1$, 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 on how each setting affects on the results.

[4 points]

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