Lecture Automotive Vision 2019

Institut für Mess- und Regelungstechnik, Karlsruher Institut für Technologie
Dr. Martin Lauer, M.Sc. Annika Meyer, M.Sc. Danial Kamran

Block Matching

This excercise sheet covers block matching as one method to find corresponding points in pairs of images. The pairs of images referred to in the following tasks are shown in Figure 1 and are located in *images/left/0000.png* and *images/right/0000.png*, respectively. Additional images for further testing are available in the same directories.

(a) Left camera image

(b) Right camera image

Figure 1: Input images.

Exercise (2.1):

Answer and discuss the following tasks:

a. Describe shortly the working principle of block matching to find corresponding points in both input images.

b. Name three mathematical functions and their formulas that could be used to compare two image blocks. What are their advantages and disadvantages?

c. Discuss the influence of the window size used in block matching on the quality of the matching result.

Point Matching as Search Problem

The objective in this task is to implement block matching to match points in the left image to corresponding points[1] in the right image. Various image points (x, y) are given in the left image (Figure 2):

$(161, 297), (980, 350), (1046, 320), (600, 26), (597, 87), (680, 109), (356, 121), (840, 101), (1086, 86), (603, 262)$

Figure 2: Left input image with given points as indicated by blue circles.

A skeleton script for this task is available in the file `block_matching_points.m`. It reads both images, defines the set of points in left image as listed above and calls the function `match_point` for each of the points to return the best matching point in the right image. The signature of the function `match_point` is:

```
match_point(I1, I2, p, search_area, window_size)
```

The meaning of the function parameters is as follows:

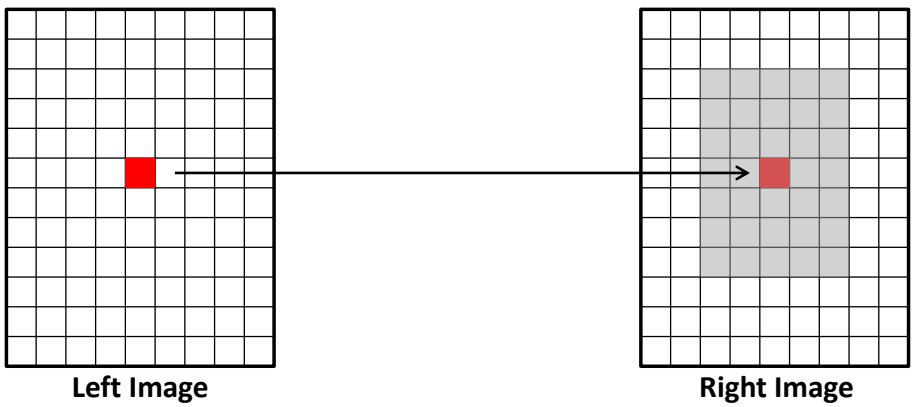

Figure 3: The meaning of the `search_area` parameter of the function `match_point`.

[1]The terms *matched (image) point* and *corresponding (image) point* are used interchangeably in this excercise sheet to refer to points in the right image that *match* points in the left image.

`I1`	The left input image.
`I2`	The right input image.
`p`	The image point coordinates (x, y) in the left image to consider for matching.
`search_area`	The size as $[height, width]$ of the area to search for in the right image. The search area is considered around the point `p` in the right image (see Figure 3). The search should be applied to every point in the search area, even if the matching window might be (partly) outside of it. Note: the width and height are expected to be odd.
`window_size`	The window size as $[height, width]$ to apply the comparison function from the lecture to. Note: The width and height are expected to be odd.

Exercise (2.2):

Implement and answer the following tasks. Make sure to convert the pixel values from uint to double if necessary.

a. Implement the function `match_point.m` to return the best matching point in the search window as $[u, v]$. Use the helper function mentioned below and use the sum of absolute gray-value difference as comparison function. In order to execute your function, run `block_matching_points.m`.

b. Extend the function `match_point` to return the best matching points in the search window as $[u_1, v_1; ...; u_N, v_N]$. Limit the result to only the best 10 of the matches.

c. Perform multiple experiments with varying search area and window sizes and discuss the requirements for an image point to result in good corresponding points when using block matching.

The following functions are provided as helper functions:

`extract_image_patch`	This function extracts a block from an image and returns this block as an NxM matrix. It accepts three parameters: the input image, the pixel location as $[u, v]$ and the dimensions of the block as $[height, width]$. The result has dimensions $N = height$ and $M = width$.
`visualize_matched_points`	This function takes the two images, the pixel location in the left image as $[u, v]$ and the list of corresponding points returned by `match_point` and visualizes the results for debugging purpose.

Feature Extraction

Given the results from the previous task, a very important prerequisite for good and useful block matching results is to find salient image points, in the following referred to as features, in the left image. This task therefore investigates how good features could be extracted by means of Matlab utility functions and then extends the Matlab code from the previous task.

Exercise (2.3):

Answer and implement the following tasks:

a. Which feature extraction algorithms are provided by Matlab? Name three algorithms.

b. Implement a Matlab function `detect_features` that returns a list of salient image locations in an image. The signature of the function contains the image `I` and `number` for the number of features to return. The extracted features are returned as an Nx2 matrix with each row holding the pixel locations of the features as $[u, v]$. Assume `number` = 10 in the following. Choose any of the feature detectors provided by MATLAB.

Given the function `match_point` from the previous task and the newly created function `detect_features` from this task, the two functions are next combined.

Exercise (2.4):

Implement the function `feature_matching` that computes corresponding image locations in a pair of images. The function takes two arguments as input: `I1` for the left image and `I2` for the right image. The function is supposed to return a Nx4 matrix with each row holding the locations of the corresponding image points in both images $[u_{left}, v_{left}, u_{right}, v_{right}]$. Note: use the function `detect_features` and `match_point` from the previous tasks that returns the best matching point in the right image and assume reasonable values for the parameters `search_area` and `window_size`. A skeleton script for executing `feature_matching` and visualizing the results is provided by *block_matching_feature.m*.

For debugging purposes, the visualization function `visualize_feature_matching` may be used. It requires the two images `I1` and `I2` as well as the list of corresponding points returned by `block_matching` as input. The result of this function is shown in Figure 4. Additionally, the debugging function `plot_patches` draws the patches of the matched points side-by-side to visually validate the quality of the results.

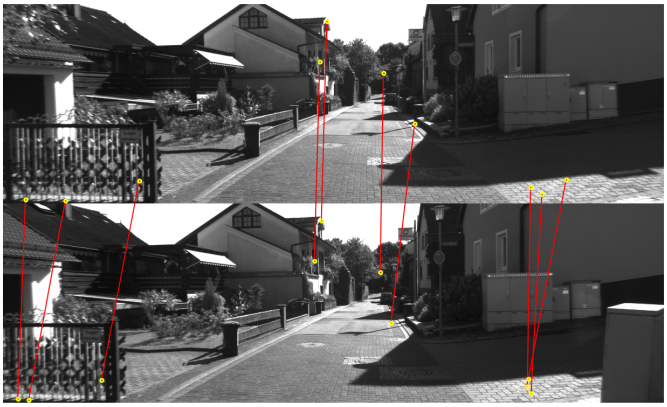

Figure 4: Example of the debugging image visualized by `visualize_matches`.

Rectification

In this task we assume that the image pairs provided as input are rectified.

Exercise (2.5):

Answer and implement the following tasks:

a. Explain shortly what rectification is and how cameras may be rectified.

b. Which essential property do rectified images have? How does this property help when applying block matching?

c. Modify the function `match_point` to include the assumption that the input images are rectified.

Experiments

Exercise (2.6):

Perform further block matching experiments with the additional input image pairs provided in the directories *images/left* and *images/right*.