Lecture Automotive Vision 2019
Institut für Mess- und Regelungstechnik, Karlsruher Institut für Technologie
Dr. Martin Lauer, M.Sc. Annika Meyer, M.Sc. Danial Kamran

# Feature Point Methods/SIFT

This assignment deals with feature point methods and implements a simplified variant of SIFT (scale invariant feature transform) which we call SIFT-light. The implementation combines maxima search in scale space, the elimination of unstable detections, and the calculation of a simplified descriptor based on orientation histograms. The simplified method is neither rotation invariant nor does it use the technique of image pyramids.

Figure 1: Image from the input sequence.

The method is tested in a short image sequence with 34 images recorded with an onboard camera of a moving vehicle. You find the image sequence in folder *images* and some pieces of Matlab code in the main folder. The Matlab code is incomplete, the missing parts should be filled by you in the assignment. Those parts are marked with

```
% ----- Your code here ------
% --------------------------
```

# Detection of Feature Points

The detection of feature points in SIFT-light is done in multiple steps. Firstly, the input image is convolved with Laplacian of Gaussian (LoG) filters of varying scale ($\sigma$ parameter). The images resulting from the convolutions are then stacked into a three-dimensional tensor $L(v,u,\sigma) := \sigma^2 \cdot (g * LoG_\sigma)(v,u)$

Given the tensor $L$, feature point candidates are extracted from the tensor by searching for local maxima. A local maximum is represented by $p_m = (u_m, v_m, scale_m, L(v_m, u_m, \sigma_m), \sigma_m)$ where $\sigma_m = (\sqrt[3]{2})^{scale_m}$. A feature point candidate is considered

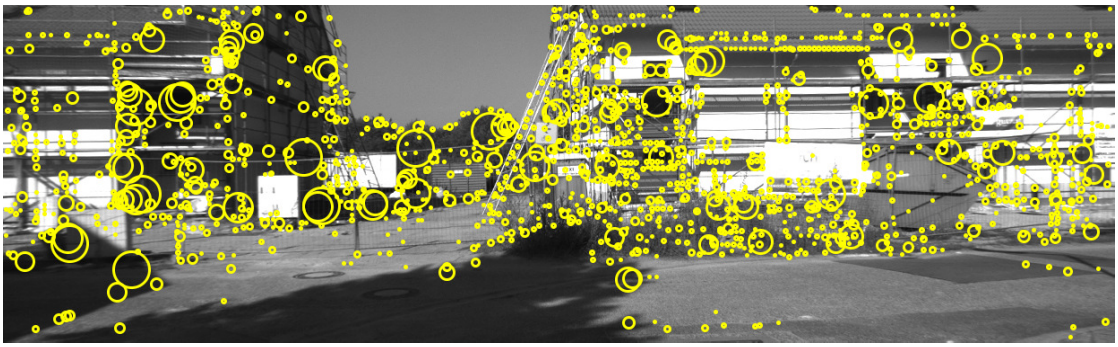

Figure 2: The result for input image *images/0000.png*.

as feature point if the following conditions are met:

- The filter response for the local maximum $L(v_m, u_m, \sigma_m)$, is larger than a threshold $\rho$.
- The Hessian matrix of the filter response at scale $\sigma_m$, $H(u_m, v_m, \sigma_m)$ fulfills the condition $\frac{Trace^2(H)}{\det H} < \frac{(r+1)^2}{r}$ for a threshold $r$.
- The feature point candidate is not located at the boundary of the image nor at the smalles or largest scale layer of $L$.

**Exercise (3.1):**

A skeleton file for the feature point detection is provided in *detect_sift_light_features.m* and can be tested with *run_detection.m*. The function returns the scale and image location of features (at multiple scales) as matrix. Implement the following tasks for the feature point detection:

(a) For each scale, convolve the input image `Idouble` with the LoG filter using the `fspecial` function of MATLAB and store it in the tensor. Note that the filter size must grow proportional to the value of $\sigma$.

(b) The feature point candidates, stored as

$$p_m = (u_m, v_m, scale_m, L(v_m, u_m, \sigma_m), \sigma_m)$$

, are filtered to extract good feature points. Implement the subfunction `threshold_features` that filters those interest points with an $L(u_m, v_m, m)$ value larger than a threshold $\rho$.

(c) The second condition for good feature points is based on the Hessian matrix as described above and in the lecture. The Hessian matrix is computed at the feature point $p_m = [u_m, v_m, \sigma_m]$ as follows:

$$H = \begin{pmatrix} \frac{\partial^2 L}{(\partial v)^2} & \frac{\partial^L}{\partial u \partial v} \\ \\ \frac{\partial^2 L}{\partial u \partial v} & \frac{\partial^L}{(\partial u)^2} \end{pmatrix}$$

The partial derivatives can be approximated by differences as follows:

$$\begin{aligned} \frac{\partial^2 L}{(\partial v)^2} &\approx L(v_m + 1, u_m, \sigma_m) - 2 \cdot L(v_m, u_m, \sigma_m) + L(v_m - 1, u_m, \sigma_m) \\ \frac{\partial^2 L}{\partial u \partial v} &\approx \frac{1}{4}(L(v_m + 1, u_m + 1, \sigma_m) + L(v_m - 1, u_m - 1, \sigma_m) \\ &\quad -L(v_m + 1, u_m - 1, \sigma_m) - L(v_m - 1, u_m + 1, \sigma_m)) \end{aligned}$$

Implement the second condition of the list above ($\frac{Trace^2(H)}{\det H} < \frac{(r+1)^2}{r}$) in the subfunction `eliminate_edge_features` and test your implementation running the script *run_detection*. The result for input image *images/0000.png* should look similar to Figure 2.

## Descriptor of Feature Points

Various methods and algorithms exist to find correspondences between features points, e.g. block matching from assigment 02. In this section descriptors as more abstract representation of the local environment of a feature point are implemented. SIFT-light uses gradient lengths and orientations in a local environment around the feature point to compute a 32-dimensional descriptor vector following the basic ideas of SIFT. The geenral process to calculate a descriptor for a certain interest point is as follows:

1. Define a rectangular area around the interest point. The size of the area should grow proportional to $\sigma_m$, the scale at which the interest point was detected. Subdivide the rectangular area into four sectors (left upper, right upper, left lower, and right lower sector).
2. Calculate the graylevel gradient for all pixels within the rectangular area.
3. For each sector, calculate a histogram of oriented gradients with eight bins, one for the interval of angles between 0° and 45°, one for the interval of angles between 45° and 90°, and so on.
4. Concatenate the orientation histogram for the four sectors to a 32-dimensional vector.
5. Normalize the vector by dividing by its Euclidean norm.

**Exercise (3.2):**

A skeleton file for the descriptor computation is provided *extract_sift_light_descriptors.m*. Implement the subfunction `calculate_orientation_histogram` which calculates an orientation histogram from a block of pixels. The function takes two matrixes of the same size as input, `Gu` contains the first order partial derivatives of the graylevel function in horizontal direction $\frac{\partial g}{\partial u}$, `Gv` contains the first order partial derivatives in vertical direction $\frac{\partial g}{\partial v}$. The output of the function should be a vector of length 8. Each entry refers to one interval of gradient directions and it sums up the gradient lengths of those pixels with appriproiate gradient direction.
Hint: the Matlab function `atan2` might be useful for this task. It calculates the angle (in radiant) of a 2-dimensional vector.

## Correspondence of Feature Points

Two feature points are said to correspond[1] to each other if they both describe the same three-dimensional point in the scene (refer to assignment 02 for further details). To figure out which feature points from both input images correspond to each other, we want to use the Euclidean distance in order to compare two descriptors. The smaller the distance is, the more similar are the descriptors.

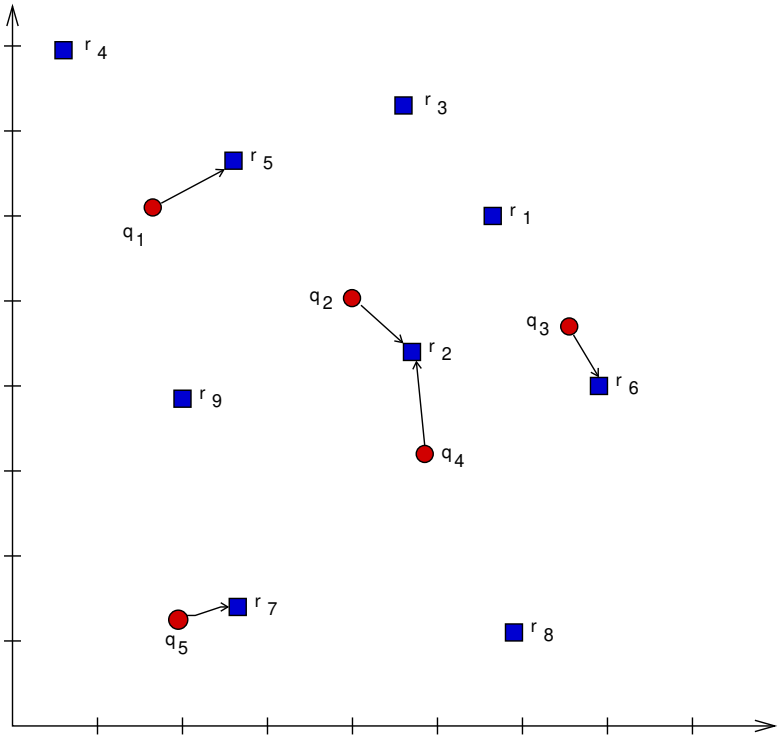


Figure 3: Visualization of the nearest neighbors function. For each of the five points $q_1, \ldots, q_5$ from set $Q$ the nearest neighbor in point set $R$ is assigned. The output of the function would be $idx = (5, 2, 6, 2, 7)$ and $D = (1.1, 0.9, 0.8, 1.2, 0.7)$

[1] An alternative expression to correspondence is feature point matching.

**Exercise (3.3):**

The file *nearest_neighbor.m* provides a skeleton code for the function that calculates the nearest neighbor of each point in one set of points $Q$ to points in another set $R$. The sets of points are arranged as matrices with one point per row. It should return two vectors. The first vector `idx` provides in its $i$-th entry the row number $j$ of the point in set $R$ that is the nearest one to the $i$-th point in set $Q$. The second vector `D` contains for each correspondence of nearest points the Euclidean distance between them. Figure 3 provides a visualization.
Once you have implemented the function nearest_neighbors run the test script *run_matching.m* in order to calculate feature matching between consecutive images of the example video. The result should look similar to Figure 4

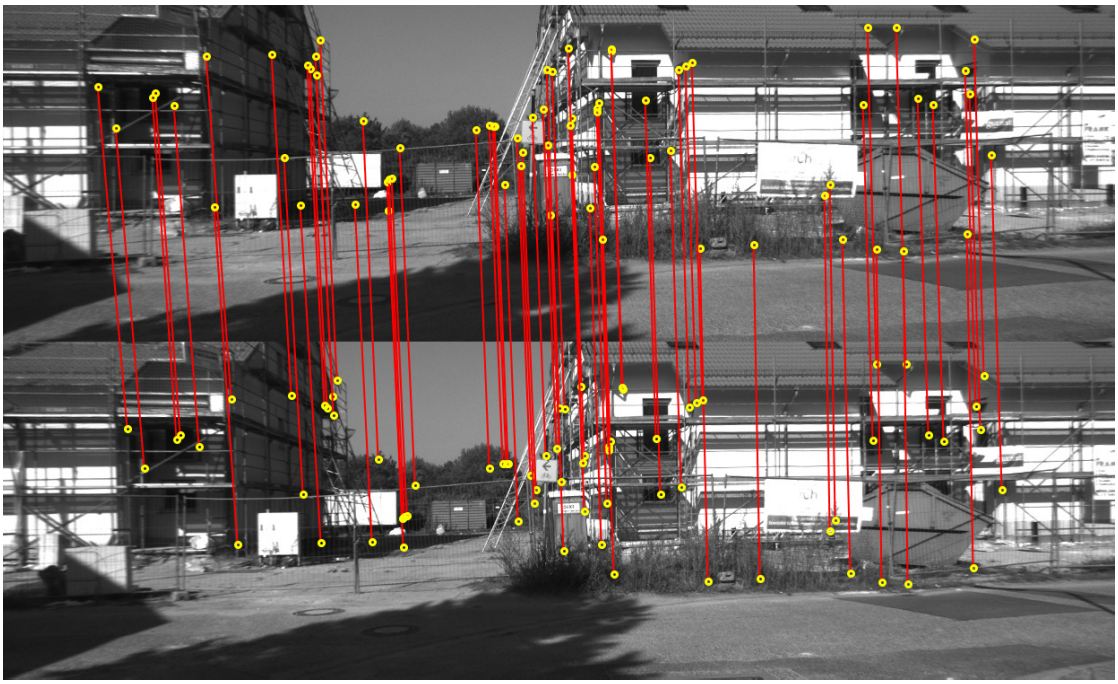

Figure 4: The result of feature matching for images *image/0001.png* (above) and *images/0000.png* (below)