Image Processing and Pattern Recognition

Assignment 2

October 29, 2012

Submission: Send report and implementation (zipped) to `bvme@icg.tugraz.at`.
Deadline: November 12, 2012, 23:59

1 Harris Corner Detector (6 points)

In this task you have to implement the Harris corner detector, which is based on the structure tensor:

$$A = \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \tag{1}$$

where I_x, I_y denote the derivatives of the image in x- and y-direction. This matrix has to be computed for every pixel. Furthermore, you have to average the matrix values over a window w by a component-wise convolution with a Gaussian kernel (standard deviation σ; experiment with different values).
If both eigenvalues of this matrix are large, a corner is found. However, the computation of the eigenvalues is inefficient. Therefore, you have to think of another measurement for each pixel (hint: $det(A) = \prod \lambda_i$, $trace(A) = \sum \lambda_i$). Discuss your measurement function in your report!
Depending on your measurement function define a threshold to discard weak responses and perform a non-maximum suppression in a 5×5 neighborhood.

Implement the detector as a function with the following header
`function [keypoints] = harris(I, sigma, thresh, verbose)`
`keypoints` is a $4 \times M$ matrix, where the first and second row represent the keypoint location, the third row the magnitude of the smallest eigenvalue and the last row the orientation of the corresponding eigenvector. The eigenvalue and -vector are only used for visualization. With the parameter `verbose` you can either plot the keypoints in the image or suppress the plots. If `verbose` is true, the keypoints in the image are plotted, indicating the smallest eigenvalue and orientation of the corresponding eigenvector. Therefore, you are allowed to use the function `vl_plotframe` from the VLFeat toolbox[1].
Present in your report the result of the corner detector with different parameters.

[1] `http://www.vlfeat.org/`

2 SIFT Feature Descriptor (6 points)

Feature descriptors describe the local region around a keypoint and can be considered as a fingerprint for a keypoint. The *Scale-Invariant Feature Transform (SIFT)* descriptor has been introduced by David Lowe in 2004 [1]. In this task, you should implement a (simplified) SIFT descriptor which is invariant to rotation and illumination. As the keypoints are only detected at one scale, the scale can be neglected in your implementation. The function should look like:
`function [descriptors] = SIFTdescriptor(I, keypoints, verbose)`

2.1 Estimation of the main orientation

First, calculate a gradient magnitude image as well as an orientation image. Then do following steps for all keypoints: Extract a patch of size $2 \cdot W_{\text{patch}} + 1$ where

$$W_{\text{patch}} = \lceil \textit{region\_factor} \cdot \sigma_w \cdot 2 \rceil . \tag{2}$$

Assume that *region_factor* is 3 and σ_w is 1.5. Calculate a gradient orientation histogram with 36 bins $(0 - 360°)$ and assign the orientations of the patch which are weighted by a Gaussian function with σ_w to it. Make sure that the orientations contain values between 0 and 2π. Estimate the main orientation of the patch which is given by the highest peak of the histogram.

2.2 Basic concept of the SIFT descriptor

Figure 1 shows the basics of computing the SIFT descriptor. Each keypoint is described by a 4x4 spatial bin array of histograms with 8 orientations. This leads to a 4x4x8=128 dimensional feature vector. For each keypoint do the following steps:
Extract a patch of size $2 \cdot W_{\text{descriptor}} + 1$ around the keypoint where W is given by

$$W_{\text{descriptor}} = 2 \cdot \left\lceil \sqrt{2} \cdot \textit{region\_factor} \right\rceil + 1 \tag{3}$$

Now, you have to assign each sample value of the patch to the corresponding spatial bin. To achieve rotation invariance, rotate the sample point by the keypoint orientation and adjust also the orientation of the sample point such that it can be assigned to the different bins. A normalization of the rotated points by *region_factor* is also necessary. For each spatial bin you should calculate a gradient orientation histogram with 8 orientation bins $(0 - 360°)$. Weight the magnitude of the sample point by a Gaussian function with $\sigma_{w_2} = 2$ to decrease the impact of image gradient magnitudes which are located far away from the keypoint.
The feature vector has to be normalized to unit length to achieve illumination invariance. Affine changes in illumination can be avoided by reducing the influence of large image gradients. Therefore, the normalized feature vector is thresholded by 0.2. Finally, you have to normalize the feature vector again to unit length.

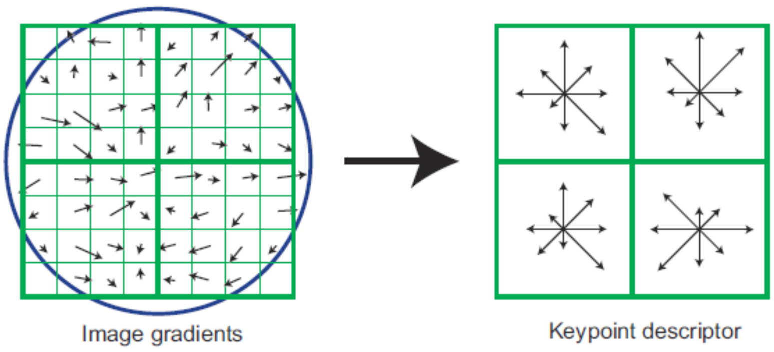

Figure 1: This figure depicts a pixel grid of 2x2 which is centered on the keypoint. The gradient magnitude and orientations are determined in a local neighborhood of the keypoint. Additionally, these image gradients are weighted by a Gaussian function which is indicated by the blue circle. For each of the 2x2 bin, a orientation histogram over the 4x4 subregions (8 orientations) is estimated. We use a pixel grid of 4x4 (spatial bin array) and 16x16 subregions.

Take an arbitrary image, calculate the Harris corners and estimate the corresponding descriptors. Show the orientation histograms for some keypoints and report your observations. To visualize the feature descriptors you can use the function `vl_plotsiftdescriptor` from the VLFeat toolbox.

3 Feature Matching (3 points)

Given the input images `match1.jpg` and `match2.jpg`, compute the keypoints and descriptors and implement simple matching of the descriptors using Euclidean distances. The second best match should be considered as well. If the distances of the best and second best match are similar (e.g. *ratio* of the two matches greater than 0.7), the matched keypoints should be rejected. Visualize your result as depicted in Figure 2.

4 Bonus: Stitching

For the bonus task you have to stitch two images together. Your implementation should be stable against outliers. Therefore, implement the RANSAC algorithm on your own to compute the homography.
The homography between two points can be derived as follows. Given two corresponding points $\mathbf{x}'_i, \mathbf{x}_i$ the homography $\mathbf{H}$ must fullfill $\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = 0$ (note, that you have to use projective coordinates; $\mathbf{x}_i = \begin{bmatrix} x_i & y_i & w_i \end{bmatrix}^T$). Rewriting the cross product, we get the linear equations $\mathbf{A}\mathbf{h} = \mathbf{0}$:

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{x}_i^T & y_i'\mathbf{x}_i^T \\ w_i'\mathbf{x}_i^T & \mathbf{0}^T & -x_i'\mathbf{x}_i^T \\ -y_i'\mathbf{x}_i^T & x_i'\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} = \mathbf{0} \tag{4}$$

Because the matrix $\mathbf{A}$ has rank 2, we only get two equations for each point pair. Therefore, you have to use at least 3 point pairs for affine, or 4 point pairs for projective transformation to compute the homography $\mathbf{H}$.

Now you can apply $\mathbf{H}$ to map one image to the other images (hint: `maketform` and `imtransform` in MATLAB). In the overlapping area use simple alpha blending $I(x, y)_{\text{blend}} = \alpha I_{\text{left}} + (1 - \alpha)I_{\text{right}}$ to get a smoother result.

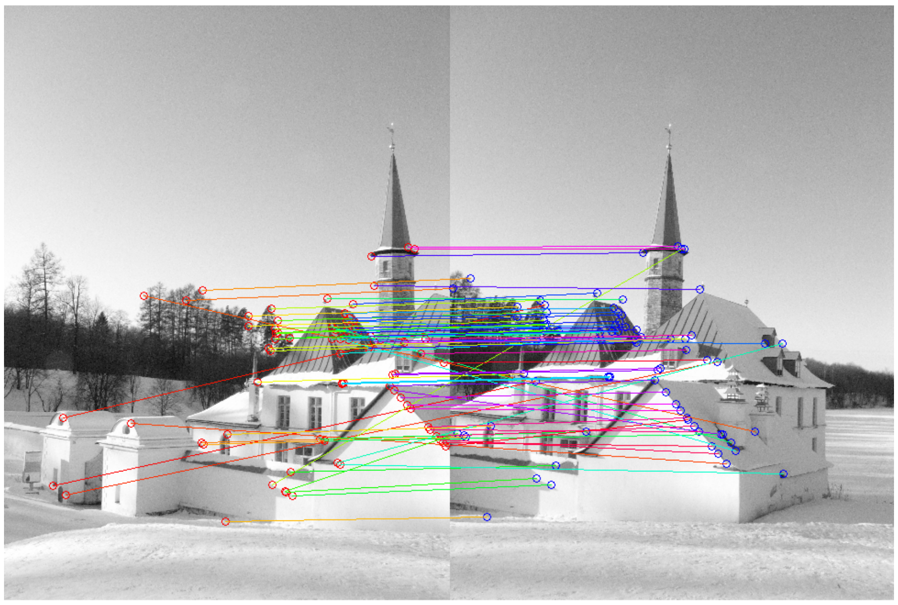

Figure 2: Matching of the two images

References

[1] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.