Assignment 3 – Computer Vision 1 – WS18/19 26/11/2018

Prof. Stefan Roth
Junhwa Hur
Anne Wannenwetsch

This assignment is due on December 17th, 2018 at 13:00.

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

Common function module

For this assignment please make use of the functions given in `Common.jl`, which is imported at the top of each Julia problem file.

Problem 1 – Hessian Detector (10 points)

In this problem we will take a look at interest point detection, *e.g.* as shown below:

Figure 1: Interest points.

One of the earliest interest point detectors was the Hessian detector which identifies corner-like structures by searching for points $\mathbf{p} = (x, y)^T$ with a strong Hessian determinant $\det(\mathbf{H})$. We use the Hessian matrix $\mathbf{H}$ that is calculated on the image smoothed by a Gaussian filter with kernel width σ, *i.e.*

$$\mathbf{H}(\sigma) = \begin{bmatrix} I_{xx}(\sigma) & I_{xy}(\sigma) \\ I_{xy}(\sigma) & I_{yy}(\sigma) \end{bmatrix} \tag{1}$$

with $I_{xx}(\sigma)$, $I_{xy}(\sigma)$ and $I_{yy}(\sigma)$ denoting the partial (horizontal and vertical) second derivatives of the smoothed image I. The interest points are defined as those points whose Hessian determinant is larger than a certain threshold t, *i.e.*

$$\sigma^4 \cdot \det\left(\tilde{\mathbf{H}}\right) > t. \tag{2}$$

Note that we include an additional scale normalization factor σ^4 so that we can use the same threshold t independently of the value of σ. For the following tasks please use the function `gauss2d` from the `Common` module to generate a Gaussian smoothing filter with size 25×25 and use central differences to compute the derivatives. For color images you only need to detect the interest points in the gray-scale space (we included a function `rgb2gray` in `Common.jl`).

The code outline is given in `problem1.jl` which should be completed with the necessary functions:

1. Function `loadimage` that is used to load both the gray-scale and color version of `a3p1.png`.
2. Function `computehessian` to obtain the required components of the Hessian $\mathbf{H}$ with $\sigma = 4.5$. Use replicate boundary conditions for any filtering involved.
3. Function `computecriterion` that computes the scaled Hessian determinant given by the left-hand side of (2).
4. Function `nonmaxsupp` that applies non-maximum suppression to the computed criterion in order to extract local maxima, *i.e.*, points for which function values are the largest within their surrounding 5×5 windows, respectively. Allow multiple equal maxima in one window and throw away all interest points in a 5 pixel boundary at the image edges. After that find all local maxima with a function value that is larger than the threshold $t = 10^{-3}$. Note: To implement `nonmaxsupp`, the functions `nlfilter` and `findnonzero` of the `Common` module might be handy. Figure 1 shows an example of how a interest point visualization can look like.

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

Problem 2 - Image Stitching (25 points)

In this problem you will use the RANSAC principle to robustly estimate the homography between two images. Then you can register two partially overlapping images and build a panorama image, as shown below:

Figure 2: Image stitching.

Homography estimation and RANSAC

The outline is given in `problem2`. Complete the following tasks:

1. We have already precomputed a set of interest points using the Harris interest point detector. Implement the function `loadkeypoints` that loads two sets of interest points, *i.e.* one for each image, from the file `keypoints.jld2`.

2. As feature descriptor we will use the Scale-Invariant Feature Transform (SIFT) descriptor. In the problem script we already compute SIFT features for all detected interest points in each image with a standard deviation of $\sigma = 1.4$ for smoothing.

3. For finding putative matches between the keypoints in both images, we have to define a distance function for corresponding feature vectors. A simple distance measure is given by the (squared) Euclidean distance which is defined as
$$d^2(\mathbf{p}, \mathbf{q}) = \sum_i (q_i - p_i)^2 \quad i = 1 \dots D,$$
where $\mathbf{p}, \mathbf{q}$ are given feature vectors with dimension D. Please implement `euclideansquaredist` which returns the pairwise squared Euclidean distance for two sets of keypoints.

4. Function `findmatches` that takes two sets of keypoints as well as the pairwise distance matrix and computes for each point in the smaller set the nearest neighbor in the larger set.

5. Function `showmatches` that shows the estimated correspondences on top of the two given images.

You will now implement the RANSAC algorithm, however, you should first determine a reasonable number of iterations by implementing `computeransaciterations`. Estimate the amount of iterations needed to draw an uncontaminated sample of $k = 4$ corresponding point pairs with a probability of $z = 99\%$. Take a conservative guess

of $p = 50\%$ for the probability of any given pair being an inlier. Continue to implement the RANSAC algorithm in function `ransac`. You must implement and reuse following subfunctions called from `ransac`:

1. Function `picksamples` that randomly picks k points from given keypoints. For our problem we draw $k = 4$ correspondences per sample.
2. Function `condition` that conditions any set of given points to improve numeric stability. This function should be used in `computehomography`.
3. Function `computehomography` that estimates the homography from given correspondences. If you use the built-in function `svd`, please specify the option `full=true` to improve numerical stability (this is an issue of the implementation of SVD in Julia).
4. Function `computehomographydistance` that evaluates a homography w.r.t. the putative correspondences. It computes for each point the (squared) distance to the corresponding point transformed by the homography, i.e.
$$d^2(H, \mathbf{x}_1, \mathbf{x}_2) = \|H\mathbf{x}_1 - \mathbf{x}_2\|^2 + \|\mathbf{x}_1 - H^{-1}\mathbf{x}_2\|^2.$$
5. Function `findinliers` that determines the number of inliers and their indices for given homography distances and a distance threshold. Use a threshold of $t = 50$ within the RANSAC algorithm.

The RANSAC algorithm then proceeds as follows:

- Randomly draw a sample of four corresponding point pairs and estimate the corresponding homography using your homography function.
- Evaluate the homography by means of the homography distance specified above. For each iteration you have to determine the number of inliers for the estimated homography.

Repeat above two steps for the number of iterations you estimated beforehand. Remember the homography that has the highest number of inliers as well as the associated inliers and the correspondences used to compute the homography. Finally, show the 4 point correspondences w.r.t the obtained homography using `showmatches`. Also show corresponding inlier matches on top of the two images.

As a last step, refit the homography based on all inlier pairs in `refithomography`.

Panorama stitching

With the estimated homography, you can now stitch the two images into a panorama image. In case that you can not successfully estimate the homography from above tasks, use the provided homography matrix stored in `H.jld2` (note that this provided homography matrix may not be the answer but just one of the example cases). To accomplish the task you have to transform the image `a3p2b.png` into the image plane of `a3p2a.png` using the homography. You should use bi- linear interpolation to get the pixel values. You can use the Julia package `Interpolations` or the `bilinear_interpolation` function in the `Images` package.

Implement the stitching in `showstitch`. More precisely, initialize one larger image with 700-pixel width and the same height as the given images. Fill the left 300 pixel columns with pixels from image `a3p2a.png`. The other part of the panorama image should be reconstructed from transformed `a3p2b.png`. Finally, display your panorama image.

Discussion

Setting the threshold value t in a proper range for determining inliers is one of the most important factors to get reasonable outcomes when running RANSAC. Try to change the threshold value t between 0.005 and 500.0 and check the number of the best inliers and the quality of the reconstructed image with different thresholds. Also, try to run the source code multiple times with the same threshold and see whether the results are changing every time or consistent under the same threshold. Then, briefly explain in `answersproblem2.txt` which value of threshold (or which range of threshold) you want to choose and why you choose it in terms of *i)* the number of best inliers, *ii)* the quality of the output, and *iii)* the consistency with multiple runs.

Submission: Please include only `problem2.jl` and `answersproblem2.txt` in your submission.