Assignment 3 – CV1 WS16/17 05/12/2016

Prof. Stefan Roth
Junhwa Hur
Tobias Plötz

This assignment is due on January 16th, 2017 at 12:00.

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

For this assignment please make use of the functions given in `Common.jl`, which gets already included at the top of each `problemX.jl`.

Problem 1 - Harris Detector (10 points)

The Harris detector identifies corner-like structures by searching for points $p = (x, y)$ for which the structure tensor $\mathbf{C}$ around p has two large eigenvalues. The matrix $\mathbf{C}$ can be computed from the first derivatives at p, spatially weighted by a Gaussian filter kernel $G(\tilde{\sigma})$:

$$\mathbf{C}(\sigma, \tilde{\sigma}) = G(\tilde{\sigma}) * \begin{bmatrix} D_x^2(\sigma) & D_x D_y(\sigma) \\ D_x D_y(\sigma) & D_y^2(\sigma) \end{bmatrix} = \begin{bmatrix} G(\tilde{\sigma}) * D_x^2(\sigma) & G(\tilde{\sigma}) * D_x D_y(\sigma) \\ G(\tilde{\sigma}) * D_x D_y(\sigma) & G(\tilde{\sigma}) * D_y^2(\sigma) \end{bmatrix}, \tag{1}$$

where $*$ is the convolution operator, and $D_x(\sigma), D_y(\sigma)$ denote the partial (horizontal and vertical) derivatives of the image that has been smoothed using a Gaussian smoothing filter with kernel width σ (Note the difference between σ and $\tilde{\sigma}$). The interest points are defined as those points whose Harris function values are larger than a certain threshold t

$$\det(\sigma^2 \mathbf{C}) - \alpha \cdot \left(\mathrm{trace}(\sigma^2 \mathbf{C})\right)^2 > t. \tag{2}$$

Note that we include an additional scale normalization factor σ^2 so that we can use the same threshold t independently of the value of σ. In practice, the parameters are usually set as: $\tilde{\sigma} = 1.6\sigma, \alpha = 0.06$. For the following tasks please use the built-in function `gaussian2d` to generate Gaussian smoothing filters 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 the `Common` module). The code outline is given in `problem1` which should be completed with the necessary function calls. Your tasks are:

1. Function `loadimage` that is used to load both the gray-scale and color version of `a2p3.png`.
2. Function `computetensor` to obtain the structure tensor $\mathbf{C}$ with $\sigma = 2.4$ as computed above. Use replicate boundary conditions for any filtering involved.
3. Function `computeharris` that computes Harris function values given by the left-hand side of (2).
4. Function `nonmaxsupp` that applies non-maximum suppression to the Harris function values in order to extract local maxima, *i.e.*, points for which Harris function values are the largest within their surrounding 5×5 windows, respectively. Allow multiple equal maxima in one window and throw away all Harris points in a 2 pixel boundary at the image edges. After that find all local maxima with a Harris function value that is larger than the threshold $t = 10^{-6}$. As an example, look at Figure 1. Note that these results are ***not necessarily*** correct.

Figure 1: Harris points.

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 in the following figure.

Homography estimation and RANSAC

The outline is given in `problem2`. Your tasks are:

1. Function `loadkeypoints` that loads two sets of interest points from the file `keypoints.jld`.
2. Continue with computing sift features for all detected interest points using $\sigma = 1.4$. This is already done in the function `problem2`.
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

$$d^2(\mathbf{p},\mathbf{q}) = \sum_i (q_i - p_i)^2 \quad i = 1 \ldots D,$$

   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 `thin=false` 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 $t = 50$ pixels 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.jld`. To accomplish the task you have to transform the image `a3p1b.png` into the image plane of `a3p1a.png` using the homography. You should use bilinear interpolation to get the pixel values. The Julia package `Grid` might be useful.

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 `a3p1a.png`. The other part of the panorama image should be reconstructed from transformed `a3p1b.png`. Finally, display your panorama image.

Bonus task (10 points)

You can earn bonus points by extending the above code for panorama stitching. For example, you could write a GUI for selecting multiple input images or implement color blending and other transformations that make the results look nicer. Be creative and show us your results! Briefly document what you did and how to use it, as well as the main problems that you encountered when solving this task.

Problem 3 - Fundamental Matrix (10 points)

In this problem, you will use the 8-point algorithm to compute the fundamental matrix between two images `a3p2a.png` and `a3p2b.png` which are shown in the following Figure:

Tasks:

The main function `problem3` already loads the images and point correspondences. Write a function `eightpoint`, which takes the 8 correspondences in homogeneous coordinates as arguments, and outputs the fundamental matrix. The function performs the following steps:

- Condition the image coordinates numerically, by moving their mean to $[0,0]^{\mathrm{T}}$ and scaling them such that coordinates are included in the interval between 1 and -1. Here, you can simply use the function `condition` that you already implemented for Problem 2.
- Using the conditioned image points, compute the actual fundamental matrix in `computefundamental` by first building the homogeneous linear equation system for the elements of the fundamental matrix, and then solving it using singular value decomposition. Again, use `thin=false` when calling `svd`.
- The preliminary fundamental matrix is not yet exactly of rank 2, due to noise and numerical errors. Enforce the rank constraint using SVD in `enforcerank2`. This function should be called in `computefundamental`.
- So far you still have the fundamental matrix for the conditioned image points. Transform it to obtain the one for the original pixel coordinates. This is done by matrix multiplication in function `eightpoint`.

Finally, we want to check the correctness of the computed fundamental matrix. First, draw the epipolar lines by implementing the function `showepipolar`. Second, verify that the epipolar constraints are satisfied for all pairs of corresponding points by computing the remaining residual in `computeresidual`.