# Assignment 3 – CV1 WS14/15 24/11/2014

Prof. Stefan Roth
Jochen Gast
Stephan Richter

**This assignment is due on December 15th, 2014 at 13:00.**

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

For this assignment please make use of the functions given in subfolder `common`. That is, you do not need to implement `harris` or `sift` by yourselves. Each problem already makes these functions available via `addpath`.

## Problem 1 - Image Stitching (27 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 `problem1.m` and should be completed with the necessary function calls. Your tasks are:

1. Function `make_masks.m` that computes two logical masks from overlapping regions of these two images, i.e. the right half of the first image and the left half of the second. However, you should mask out pixels within a 10-pixel wide boundary.

2. Function `detect_keypoints.m` that applies a Harris detector with $\sigma = 1.4$, a filter size of $15 \times 15$ and a threshold of $t = 10^{-7}$ to compute interest points for both images.

3. Continue with computing sift features for all detected interest points using $\sigma = 1.4$.

4. 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 this distance measure in `euclidean_square_dist.m`. Note that you are not allowed to use Matlab's `pdist` or `pdist2` in this task.
***Bonus points:*** Note that the Euclidean distance is often not very discriminative. A more discriminative distance measure is given by the Chi-square distance

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

where components are not equally weighted. You can get two bonus points if you (additionally) implement `chi_square_dist.m` in a vectorized way, i.e. do not use any for-loops and do not use `arrayfun` or `cellfun`, since these are not vectorized functions and consequently very slow).

5. Function `find_matches.m` that takes two sets of keypoints as well as the pairwise distance matrix and computes the nearest neighbors to each other, i.e. a one-to-one mapping from one set of keypoints to the other. Note that you should use the set with a smaller number of keypoints as reference to compute a one-to-one mapping.

6. Function `show_matches.m` that shows the estimated correspondences on top of the two given images.

You will know implement the RANSAC algorithm, however, you should first determine a reasonable number of iterations by implementing `compute_ransac_iterations.m`. 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 sample being an inlier. Continue to implement the RANSAC algorithm in function `ransac.m`. You must implement and reuse following subfunctions called from `ransac.m`:

1. Function `pick_samples.m` that randomly picks k points from given keypoints. For our problem we draw $k = 4$ correspondences per sample.

2. Function `condition.m` that conditions any set of given points to improve numeric stability. This function should be used in `compute_homography.m`.

3. Function `compute_homography.m` that estimates the homography from given correspondences.

4. Function `compute_homography_distance.m` that evaluates a homography w.r.t. the putative correspondences; i.e. compute the (squared) distance of one point to the transformed other point

$$d^2(H, \mathbf{x}_1, \mathbf{x}_2) = \|H\mathbf{x}_1 - \mathbf{x}_2\|^2 + \|\mathbf{x}_1 - H^{-1}\mathbf{x}_2\|^2.$$

5. Function `find_inliers.m` that determines the inliers 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 `show_matches.m`. Also show corresponding inlier matches on top of the two images.

### 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.mat`. To accomplish the task you have to transform the image `a4p3b.png` into the image plane of `a4p3a.png` using the homography. You should use bilinear interpolation to get the pixel values. (Matlab build-in function `interp2` might be useful.)

Implement the stitching in `show_stitch.m`. 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 `a4p3a.png`. The other part of the panorama image should be reconstructed from transformed `a4p3b.png`. Finally, display your panorama image.

## Problem 2 - Fundamental Matrix (10 points)

In this problem, you will use the 8-point algorithm to compute the fundamental matrix between two images `a4p2a.png` and `a4p2b.png` which are shown in the following Figure:

**Tasks:**

The main script `problem2.m` already loads the images and point correspondences. Plot these points into the images to verify that they correspond. Then write a function `eightpoint.m`, 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 should simply use the function `condition.m` that you already implemented for Problem 1.
- Using the conditioned image points, compute the actual fundamental matrix in `compute_fundamental.m` by first building the homogeneous linear equation system for the elements of the fundamental matrix, and then solving it using singular value decomposition.
- The preliminary fundamental matrix is not yet exactly of rank 2, due to noise and numerical errors. Enforce the rank constraint using SVD in `enforce_rank2.m`. This function should be called in `compute_fundamental.m`.
- 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.

Finally, we want to check the correctness of the computed fundamental matrix. First, draw the epipolar lines by implementing the function `show_epipolar.m`. Second, verify that the epipolar constraints are satisfied (up to a small residual) for all pairs of corresponding points by computing the reprojection error.

## Problem 3 - Optical Flow (8 points)

For estimating the optical flow, the method of Lucas and Kanade minimizes the local energy

$$E\big(\mathbf{u}(x_0,y_0)\big) = \sum_{(x,y)\in\mathcal{N}_\rho(x_0,y_0)} \Big(f_x(x,y,t)u + f_y(x,y,t)v + f_t(x,y,t)\Big)^2,$$

where $u$ and $v$ are horizontal and vertical components of optical flow vector $\mathbf{u}$, $\mathcal{N}_\rho(x_0,y_0)$ is the spatial neighborhood window of size $\rho$ around $(x_0,y_0)^\mathrm{T}$. By setting the derivatives w.r.t. $u$ and $v$ to be zero, we can get following systems of equations:

$$\begin{bmatrix} \sum_{\mathcal{N}_\rho} f_x^2 & \sum_{\mathcal{N}_\rho} f_x f_y \\ \sum_{\mathcal{N}_\rho} f_x f_y & \sum_{\mathcal{N}_\rho} f_y^2 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum_{\mathcal{N}_\rho} f_x f_t \\ -\sum_{\mathcal{N}_\rho} f_y f_t \end{bmatrix}.$$

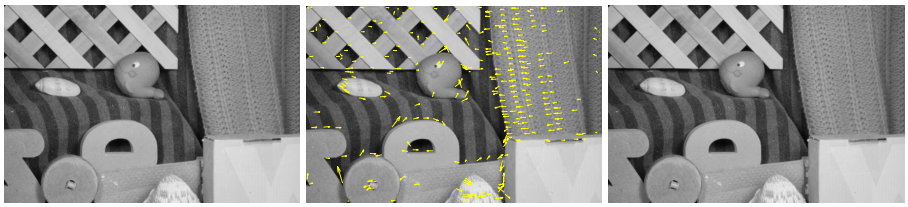
For simplicity, we can replace the hard window $\mathcal{N}_\rho$ by convolution with a Gaussian smoothing filter $G_\rho$ and get

$$\underbrace{\begin{bmatrix} G_\rho * (f_x^2) & G_\rho * (f_x f_y) \\ G_\rho * (f_x f_y) & G_\rho * (f_y^2) \end{bmatrix}}_{A} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -G_\rho * (f_x f_t) \\ -G_\rho * (f_y f_t) \end{bmatrix}.$$

Note that the matrix $A$ is just the structure tensor which we use for Harris points detection.

### Tasks:

Compute the optical flow from the 9th frame (`frame09.png`) to the 10th frame (`frame10.png`) of the rubber-whale sequence and show the estimated flow vectors as in the following figure.



The main script, from which you start is `problem3.m`. It already loads the images and sets the required parameters.

- Start by presmoothing the images with a Gaussian filter ($\sigma = 2$ and size $25 \times 25$) in `presmooth.m`.
- Now calculate the spatial and temporal derivatives from the smoothed images with `compute_derivatives.m`. Use central differences for estimating the spatial derivatives $f_x$ and $f_y$ and forward differences for the temporal derivative $f_t$.
- As full flow can only be estimated at the points for which A has full rank (i.e. corner points), you should use the Harris detector to find all feature (corner) points in the 9th frame (with parameters $\sigma = 1$, filter size $15 \times 15$ and threshold $t = 10^{-7}$). Apply the interest point detection to the original image (before presmoothing) in detect_interestpoints.m. Note that you do not need to implement the Harris detector again, as we have supplied the function in the `common` folder.
- Compute the coefficients of the linear system of equations in `compute_coefficients`. For the Gaussian window $G_\rho$, use $\sigma = 2$ and window size $11 \times 11$.
- Calculate the optical flow in `compute_flow.m` by solving for $u$ and $v$. *Bonus points:* If you successfully implement `compute_flow.m` without using any loop you can get another 2 points as a bonus.
- Show the flow vectors on top of the image by using the Matlab built-in `quiver()` in `show_flow.m`.