Stefan Breuers <breuers@vision.rwth-aachen.de>
Wolfgang Mehner <mehner@vision.rwth-aachen.de>

Exercise 6: Fundamental Matrix, RANSAC, Triangulation, Lucas-Kanade Optical Flow

due **before** 2017-01-30

Important information regarding the exercises:

- The exercise is not mandatory.
- There will be no corrections.
- Nevertheless, we encourage you to work on the exercises and present your solutions in the exercise class. For this regard the submission rules.
- In the archive for this exercise you will find the functions `apply.m` that should be used for displaying your results. You should also use it to test your implementation and see if the results make sense. Answers are to be submitted within `answers.m` Do **not** modify the apply files in any way.
- Please do **not** include the data files in your submission!
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Submit your solutions via the L^{2}P system.

Question 1: Fundamental Matrix Estimation ..... $(\mathbf{\Sigma = 0})$

In this exercise, we will use the Eight-point algorithm presented in the lecture in order to estimate the fundamental matrix between a pair of images. We will use a slightly simpler version of the algorithm here than the one presented in the lecture. The exercise archive contains two pairs of stereo images on which we want to try out our implementation.

Let's first assume that we are given a list of perfect correspondences $\mathbf{x} = (u, v, 1)^T$ and $\mathbf{x}' = (u', v', 1)^T$, so that we don't have to deal with outliers. The fundamental matrix constraint states that each such correspondence must fulfill the equation

$$\mathbf{x'}^\top \mathbf{F} \mathbf{x} = 0. \tag{1}$$

We can reorder the entries of the matrix to transform this into the following equation

$$\begin{bmatrix} uu' & uv' & u & vu' & vv' & v & u' & v' & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{21} \\ F_{31} \\ F_{12} \\ F_{22} \\ F_{32} \\ F_{13} \\ F_{23} \\ F_{33} \end{bmatrix} = 0 \tag{2}$$

By stacking $N \geq 8$ of those equations in a matrix $\mathbf{A}$, we obtain the matrix equation

$$\mathbf{Af} = 0 \tag{3}$$

$$\begin{bmatrix} u_1u'_1 & u_1v'_1 & u_1 & v_1u'_1 & v_1v'_1 & v_1 & u'_1 & v'_1 & 1 \\ u_2u'_2 & u_2v'_2 & u_2 & v_2u'_2 & v_2v'_2 & v_2 & u'_2 & v'_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_Nu'_N & u_Nv'_N & u_N & v_Nu'_N & v_Nv'_N & v_N & u'_N & v'_N & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{21} \\ \vdots \\ F_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4}$$

which can be easily solved by Singular Value Decomposition (SVD), as shown in previous exercises. Applying SVD to $\mathbf{A}$ yields the decomposition $\mathbf{A} = \mathbf{UDV}^\top$. The homogeneous least-squares solution corresponds to the least singular vector, which is given by the last column of $\mathbf{V}$.

In the presence of noise, the matrix $\mathbf{F}$ estimated this way will however not satisfy the rank-2 constraint. This means that there will be no real epipoles through which all epipolar lines pass, but the intersection will be spread out over a small region. In order to enforce the rank-2 constraint, we therefore again apply SVD to $\mathbf{F}$ and set the smallest singular value D_{33} to zero.

The reconstructed matrix will now satisfy the rank-2 constraint, and we can obtain the epipoles as

$$\mathbf{Fe_1} = 0 \quad \text{and} \tag{5}$$

$$\mathbf{F}^\top\mathbf{e_2} = 0 \tag{6}$$

by again applying SVD and setting $\mathbf{e_1} = \frac{\begin{bmatrix} V_{13} & V_{23} & V_{33} \end{bmatrix}}{V_{33}}$ and $\mathbf{e_2} = \frac{\begin{bmatrix} U_{13} & U_{23} & U_{33} \end{bmatrix}}{U_{33}}$.

Similarly, for the points $\mathbf{x}, \mathbf{x}'$, we can obtain the epipolar lines $\mathbf{l}' = \mathbf{Fx}$, $\mathbf{l} = \mathbf{F}^\top\mathbf{x}'$ in the other image. Note that in projective geometry, a line is also defined by a single 3D vector. This can be easily seen by starting with the standard Euclidean formula for a line

$$ax + by + c = 0 \tag{7}$$

and using the fact that the equation is unaffected by scaling to apply it to the homogeneous point $\mathbf{x} = (X, Y, W)$. Thus, we arrive at

$$aX + bY + cW = 0 \tag{8}$$

$$\mathbf{l}^\top\mathbf{x} = \mathbf{x}^\top\mathbf{l} = 0. \tag{9}$$

The parameters of the line are easily interpreted: $-a/b$ is the slope, $-c/a$ is the x-intercept, and $-c/b$ is the y-intercept.

(a) Write a function that implements the above algorithm to compute the fundamental matrix and the epipoles from a set of (at least 8) perfect correspondences given in the vectors `x1` and `x2`.

```
function [F, e1, e2] = fundmatrix(x1, x2)
  % Input:
  %   x1,x2 : 3xN arrays of N homogenous points in 2D
  % Output:
  %   F     : The 3x3 fundamental matrix such that x2'*F*x1 = 0
  %   e1    : The epipole in image 1 such that F*e1  = 0
  %   e2    : The epipole in image 2 such that F'*e2 = 0
```

(b) As explained in the lecture, we need to take care to normalize the points in order to make sure the estimation problem is well conditioned. Write a function `normalize2dpts` that normalizes the given list of 2D points `pts` by first shifting their origin to the centroid and then scaling them such that their mean distance from the origin is $\sqrt{2}$. Since the input points are homogeneous, pay attention to divide them by their last component before processing them. The function should return both the transformed points `newpts` and the 3×3 transformation matrix `T`.

```
function [newpts, T] = normalize2dpts(pts)
  % Input:
  %   pts   : 3xN array of N homogenous points in 2D
  % Output:
  %   newpts: 3xN matrix of transformed points
  %   T     : the 3x3 transformation matrix, newpts = T*pts
```

(c) Now write an adapted function `normfundmatrix` that first normalizes the input points, computes the fundamental matrix based on the normalized points, and then undoes the transformation by applying $\mathbf{F} = \mathbf{T_2}^\top \mathbf{F} \mathbf{T_1}$ before computing the epipoles.

```
function [F, e1, e2] = normfundmatrix(x1, x2)
  % Input:
  %   x1,x2 : 3xN arrays of N homogenous points in 2D
  % Output:
  %   F     : The 3x3 fundamental matrix such that x2'*F*x1 = 0
  %   e1    : The epipole in image 1 such that F*e1  = 0
  %   e2    : The epipole in image 2 such that F'*e2 = 0
```

(d) Next we want to verify that our implementation is indeed correct. For this purpose, write a function

```
function [x1, x2] = inputcorresp(img1, img2, N)
```

that uses Matlab's **`ginput`** command to let the user specify a set of correspondences in the two input images. The return values `x1` and `x2` should be `3xN` matrices where each column corresponds to a point in homogeneous coordinates. Set the 3rd coordinate of each point to 1. Use **`save`** and **`load`** to save those correspondences for later reference and to comment on the respective results. (Note: at least 12 correspondences will typically be needed in order to get a robust estimate of the fundamental matrix.) Save two sets of correspondences: one without and one with outliers.

(e) Apply the above algorithms to the provided test image pairs `house1/2` and `library1/2` based on the manually selected correspondences as well as for the correspondences supplied in the files `house_matches.mat` and `library_matches.mat`. For each point $\mathbf{x}$ in one image, compute the corresponding epipolar line $\mathbf{l}'$ in the other image and visualize both to see if the correspondence point $\mathbf{x}'$ indeed lies on the correct line. For this, you can use the provided function `display_epipolar_lines(x, L, I)` (*hint*: the epipolar lines are displayed in green).

Compare the results of the normalized and the unnormalized Eight-point algorithm. What do you observe? What happens when your correspondence set contains outliers?

(f) In order to get a quantitative estimate for the accuracy of your results, write a function

```
function [x1, x2] = res_error(F, x1, x2)
```

that computes the residual error of your estimation (which in this case should be defined as the mean-squared distance between points in one image and their corresponding epipolar lines). *Hint*: Be sure to normalize both homogeneous points (divide $(wx, wy, w)^T$ by w) and lines (divide $(a, b, d)^T$ by $\sqrt{a^2 + b^2}$) before computing the distance. How big are the residual errors for the normalized and the unnormalized Eight-point algorithms? What is the effect of noise, outliers and the number of correspondences?

Question 2: RANSAC ..... $(\boldsymbol{\Sigma} = \mathbf{0})$

In practice, the correspondence set will always contain noise and outliers. We therefore apply RANSAC in order to get a robust estimate. As introduced in lecture 13, RANSAC proceeds along the following steps:

1. Randomly select a (minimal) seed group of point correspondences on which to base the estimate.
2. Compute the fundamental matrix from this seed group.
3. Find inliers to this transformation.
4. If the number of inliers is sufficiently large ($\geq m$), recompute the least-squares estimate of the fundamental matrix on all inliers.
5. Else, repeat for a maximum of k iterations.

The parameter k can be chosen automatically. Suppose w is the fraction of inlier correspondences and $n = 8$ correspondences are needed to define a hypothesis. Then the probability that a single sample of n correspondences is correct is w^n, and the probability that all samples fail is $(1 - w^n)^k$. The standard strategy is thus, given an estimate for w, to choose k high enough that this value is kept below our desired failure rate.

In the following, we will implement the different steps of the RANSAC procedure and apply it for robust estimation of the fundamental matrix.

(a) Suppose we have an inlier rate of 90% / 50% / 20% and we want to be 99.9% sure that we get the right fundamental matrix. How many RANSAC iterations do we need to perform at least?

(b) Write a function

```
function [x1, x2] = get_inliers(F, x1, x2, eps)
```

which takes as input an estimated fundamental matrix and the full set of correspondence candidates and which returns the subset of correspondences that are inliers to this transformation. A point pair $\mathbf{x}, \mathbf{x}'$ is defined to be an inlier if the distance of $\mathbf{x}$ to the epipolar line $\mathbf{l} = \mathbf{F}^\top \mathbf{x}'$, as well as the opposite distance, are both less than some threshold ε.

(c) Write a function

```
function [F e1 e2 xi1 xi2] = ransac_fundmatrix(x1, x2, eps, k)
  % Input:
  %   x1,x2 : 3xN arrays of N homogenous points in 2D
  %   eps   : inlier threshold
  %   k     : number of iterations
  % Output:
  %   F       : The 3x3 fundamental matrix such that x2'*F*x1 = 0
  %   e1      : The epipole in image 1 such that F*e1  = 0
  %   e2      : The epipole in image 2 such that F'*e2 = 0
```

```
10   %   xi1,xi2 : 3xNi arrays of Ni homogenous inlier points in 2D
```

which implements the RANSAC procedure to estimate a fundamental matrix using the normalized Eight-point algorithm. For randomly sampling matches, you can use the **randperm** function. Here, we want to use a simple version of the algorithm that just runs for a fixed number of k iterations and returns the solution with the largest inlier set.

(d) Apply your RANSAC implementation to the correspondence sets supplied in `house_matches.mat` and `library_matches.mat`. Experiment a bit with the parameter ε. What do you observe? Can you think of an explanation?

(e) Next, we want to automate the estimation procedure by using point correspondences obtained with the Harris detector we developed in a previous exercise. You can either use your own implementation or take the one provided in the archive. Extract Harris points together with e.g. `maglap` descriptors on fixed-size patches for both images, then match the extracted points to find correspondences. Experiment with different thresholds for accepting putative matches. Based on the obtained correspondences, apply your RANSAC fundamental matrix estimation algorithm. Is your implementation able to estimate the correct transformation between the image pairs? How many inliers do you get? Visualize your results.

Question 3: Triangulation ..... $(\mathbf{\Sigma = 0})$

As a final step, we want to reconstruct the observed points in 3D by triangulation. Note that just using two images, this is not possible without a calibration. You can therefore find camera matrices for each image provided in the archive `exercise6.zip`. They are stored as simple text files containing a single 3×4 matrix and can be read in with the **load** command, i.e.

```
1 P1 = load('house1_camera.txt');
```

For triangulation, we use the linear algebraic approach from the lecture. Given a 2D point correspondence $\mathbf{x_1}, \mathbf{x_2}$ in homogeneous coordinates, the 3D point location $\mathbf{X}$ is given as follows

$$\lambda_1 \mathbf{x_1} = \mathbf{P_1 X} \tag{10}$$
$$\lambda_2 \mathbf{x_2} = \mathbf{P_2 X}. \tag{11}$$

We can now build the cross-product of each point with both sides of the equation and obtain

$$\begin{array}{rcccl} \mathbf{x_1} \times \mathbf{P_1 X} & = & [\mathbf{x_{1\times}}]\mathbf{P_1 X} & = & 0 \\ \mathbf{x_2} \times \mathbf{P_2 X} & = & [\mathbf{x_{2\times}}]\mathbf{P_2 X} & = & 0, \end{array} \tag{12}$$

where we used the skew-symmetrix matrices $[\mathbf{x_{i\times}}]$ to replace the cross products

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}_\times]\mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \mathbf{b}. \tag{13}$$

Each 2D point provides 2 independent equations for a total of 3 unknowns. We can therefore solve the overconstrained system by stacking the first two equations for each point in a matrix $\mathbf{A}$ and computing the least-squares solution for $\mathbf{AX} = 0$.

(a) Why is it not possible to obtain a metric reconstruction in this case without a calibration?

(b) Find the centers of both cameras. Recall from the lecture that the camera centers are given by the null space of the camera matrices. They can thus be found by taking the SVD of the camera matrix and taking the last column of $\mathbf{V}$ normalized by the 4th entry.

(c) Write a function `triangulate` that uses linear least-squares to triangulate the position of a matching point pair in 3D, as described above.

```
function X = triangulate(P1, x1, P2, x2)
```

A well suited helper function is

```
function skew_sym = skew_symmetric3(v)
```

which returns a skew symmetric matrix from the vector `v` with 3 elements.

(d) Apply the triangulation to the matches given in the dataset and collect the resulting 3D points. Display the two camera centers and the scene matches in 3D using Matlab's **`plot3`** command. Use the **`axis`** `equal` option to avoid automatic nonuniform scaling of the 3D space. Also compute the residual errors (average distance) between the observed 2D points and the projected 3D points in the two images. How does the result look?

Question 4: Lucas-Kanade Optical Flow (Bonus Question) . . . . . $(\mathbf{\Sigma = 0})$

In this exercise, you will implement the Lucas-Kanade method that is able to estimate optical flow, i.e. the motion between two images.

Recall from the lecture that optical flow is based on three assumptions: The brightness of pixels is constant, there is only small motion between frames, and points tend to move like their neighbors. Consider a pixel at the coordinates $\mathbf{p} = (x, y)$ in an image I. If the pixel moves by u, v pixels between time $t-1$ and time t, the brightness constancy assumption can be formulated as:

$$I(\mathbf{p}, t-1) = I(x+u, y+v, t) \tag{14}$$

Since we assume we are dealing with very small motion (usually less than a pixel), we can apply a first-order Taylor approximation and get

$$\begin{aligned} I(\mathbf{p}, t-1) &\approx I(\mathbf{p}, t) + I_x(\mathbf{p}) \cdot u + I_y(\mathbf{p}) \cdot v && (15)\\ \Leftrightarrow I(\mathbf{p}, t-1) - I(\mathbf{p}, t) &\approx I_x(\mathbf{p}) \cdot u + I_y(\mathbf{p}) \cdot v && (16)\\ \Leftrightarrow -I_t(\mathbf{p}) &\approx I_x(\mathbf{p}) \cdot u + I_y(\mathbf{p}) \cdot v && (17) \end{aligned}$$

where I_x and I_y denote the derivative of I in x- and y-direction respectively, and I_t is the derivative over time, i.e. the pixel-wise difference between the two frames.

In order to uniquely determine the unknowns u and v, we need more than one equation. We can get these by incorporating the spatial coherence constraint that states that neighboring pixels move in the same direction. By using neighboring pixels $\mathbf{p_1}, \ldots, \mathbf{p_{25}}$ from a quadratic (e.g. 5x5) window around our pixel we can create the linear system:

$$\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix} \tag{18}$$

$$Ad = b \tag{19}$$

This overdetermined system can be solved using least squares. In Matlab, this is very easy: `d = A\b;`

(a) Take a look at the two pairs of images (`(a|b)(1|2).png`) to get an impression of the motion between the frames.
Implement a function

```
function [u, v] = lucas_kanade(im1, im2, window_size)
  % Input:
  %   im1, im2:    The pair of images that the optical flow is
     computed on
  %   window_size: The size of the optical flow window
  % Output:
  %   u, v:        The optical flow windows in u and v direction
```

that takes a pair of images `im1` and `im2` and a window size and calculates the optical flow matrices `u` and `v` that contain for each pixel the motion estimated using the method described above.
You can use the function `gaussderiv.m` from exercise sheet 2 to calculate I_x and I_y. When calculating I_t, apply a Gaussian filter with a small σ value (e.g. $\sigma = 1$) to both images to eliminate high frequency noise before calculating the pixel-wise image difference. To avoid special cases, you can skip the estimation for the border pixels of the image where part of the sliding window would lie outside the image.

(b) Try out your algorithm on the provided test images. You can use the function `flowToColor(u, v)` to visualize the optical flow. In this visualization, a darker color corresponds to a stronger motion. The motion directions are encoded as colors. Take a look at `wheel.png` in the exercise archive to see what color corresponds to what movement direction.

(c) In order to get a better impression of the accuracy of the computed flow field, we can warp the first image with the computed flow field and compare the result to the second image. For this, implement a function

```
function im_w = warp_image(im, u, v)
  % Input:
  %   im:  the input image
  %   u,v: the optical flow
  % Output:
  %   im_w: the warped image
```

that takes as arguments an image `im` and a flow field given by `u` and `v` and returns the warped image `im_w`. *Hint:* Use Matlab's functions **`interp2`** and **`meshgrid`**.

(d) Visualize the result when warping the first image of each of the image pairs with the computed flow field and compare it to the second image. Play around with the window size and explain what effect this parameter has on the result. Is the method able to produce a satisfactory warping? Where does it fail and why?

(e) To improve on the result of the standard Lucas-Kanade algorithm, we will now implement the iterative Lucas-Kanade approach described in the lecture.

```
function [u, v] = iterative_lucas_kanade(im1, im2, window_size,
   n_iters)
  % Input:
```

```
%   im1, im2:    The pair of images that the optical flow is
    computed on
%   window_size: The size of the optical flow window
% Output:
%   u, v:        The optical flow windows in u and v direction
```

The idea is to alternately estimate the flow between the two images and warp the first image with the estimated flow field. The flow should be accumulated in each iteration and returned in the end.

The function should have an additional parameter `n_iters` that specifies the number of iterations to be performed.

Hint: When alternatingly warping and estimating the flow, small errors in either of these steps will propagate through the rest of the process. In order to prevent warping errors from propagating, you should not update your warped version of the image in each step by iteratively warping it with each flow field you compute. Instead, you should first add your flow field to the accumulated flow, and then apply the accumulated flow to the original image. This way, flow estimation will be more stable.

(f) Visualize the result of the iterative Lucas-Kanade algorithm on the provided test images. What effect does the number of iterations have? Why is this method able to produce better results? Where does it still give inaccurate results and why?

(g) For even better precision, implement the coarse-to-fine version of Lucas-Kanade.

```
function [u, v] = ctf_lucas_kanade(im1, im2, window_size)
  % Input:
  %   im1, im2:    The pair of images that the optical flow is
      computed on
  %   window_size: The size of the optical flow window
  % Output:
  %   u, v:        The optical flow windows in u and v direction
```

First, build up a Gaussian pyramid of both images by iteratively scaling the images with a factor of 0.5.

```
function pyramid = gausspyramid(im)
```

You can use Matlab's `imresize` function for this. This function takes care of both filtering and resampling, so you do not need to apply a gaussian filter yourself. Then, starting at a coarse level (that still allows for some movement of the sliding window), perform the following steps iteratively:

1. Estimate the optical flow using Lucas-Kanade.
2. Add the resulting flow to the flow calculated in previous steps and scale this accumulated flow field by a factor of 2 (Note: Also scale the flow vectors accordingly by multiplying each element of the flow field by a factor of 2.)
3. Warp the next finer level of the Gaussian pyramid of the first image with the accumulated and scaled flow field.

For debugging purposes, you should visualize the flow field in each iteration to see if the accumulation works as expected. How does the result compare to iterative Lucas-Kanade?

Hint: To avoid rounding problems, pad your images such that their dimensions are powers of 2.