Two-view geometry

The goal of this laboratory session is to practice with several techniques used in two-view geometry such as: the epipolar constraint, estimation of the fundamental matrix via the 8-point algorithm, triangulation of 3-D points from their projections, extraction of relative camera motion from the essential matrix, robust fitting via RANSAC, etc.

1 Pinhole camera model and linear triangulation

Consider a camera with the following specifications:

- Horizontal Field of View (FOV): 60°
- Vertical FOV: 45°
- Image size: 640 x 480 pixels (horizontal $\times$ vertical)
- Zero skew and principal point in the center of the image.

Given this information, write a MATLAB script that:

- Calculates the focal length (in pixels) along the horizontal axis (X). (Recall that trigonometric functions such as $\sin, \cos, \tan$ work in radians).

$$\alpha_x =$$

- Calculates the focal length (in pixels) along the vertical axis (Y)

$$\alpha_y =$$

- Constructs the camera intrinsic matrix K

$$K = \begin{bmatrix} & & \\ 0 & & \\ 0 & 0 & 1 \end{bmatrix}$$

Two cameras with the above intrinsic parameters and in a side-by-side parallel configuration with a baseline of 0.2 meters along the horizontal axis, that is, with extrinsic parameters $(I, 0)$ and (I, T) where $T = (-0.2, 0, 0)^\top$, are viewing a 3-D point Q project at locations $(360, 302)$ and $(330, 302)$ in pixel coordinates, respectively. Note that the first camera acts as the world reference frame.

- You are asked to estimate the position of point Q in 3-D coordinates.
  For this, you are required to complete function `LinearTriangulation()` and call it with the correct arguments. This function is based on the linear approach explained at the end of the slides in the lecture corresponding to file "06_multiple_view_geometry_1.pdf", from the course website.

$$Q = (\quad , \quad , \quad)^\top.$$

2 Eight-point algorithm

In this exercise, we use several point correspondences in two images to compute the fundamental matrix F, which encapsulates the geometry of both views in an uncalibrated framework. Recall that the fundamental matrix relates entities only in the image planes, not in 3-D space.

Complete the function `FundamentalEightPoint()`, which receives a set of point correspondences $\{\mathbf{p}_1^i \leftrightarrow \mathbf{p}_2^i\}$ (in homogeneous coordinates) and returns the 3×3 fundamental matrix that minimizes the epipolar constraints $\mathbf{p}_2^{i\top} F \mathbf{p}_1^i = 0$ in a least-squares sense. The function uses the Singular Value Decomposition (SVD) to solve the linear homogeneous system of equations that is built from the point correspondences using the Kronecker product $(\mathbf{p}_2^{i\top} \otimes \mathbf{p}_1^{i\top})\text{vec}(F^\top) = \mathbf{p}_1^{i\top} F^\top \mathbf{p}_2^i = \mathbf{p}_2^{i\top} F \mathbf{p}_1^i$, as explained in the slides in the lecture notes, file "07_multiple_view_geometry_2.pdf". Observe that the estimated matrix F is corrected so that it is singular: $\det(F) = 0$, as enforced by function `F2SingularF()`. This is required for all the epipolar lines in an image to intersect at a single point, the epipole.

Use the MATLAB script provided `run_test_8point.m` to test the functions you have completed. You will also need to fill in the script to call to the functions you complete. The script generates a number of exact (i.e., noise-free) point correspondences that are used to validate your code. With such a data set you should get exact results (up to machine precision).

The quality of your estimated fundamental matrix F can be measured using different cost functions. For example, we provide code to compute the algebraic error given by the sum of squared epipolar constraints $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{p}_2^{i\top} F \mathbf{p}_1^i\right)^2}$. Note that, ideally, F and the points $\{\mathbf{p}_1^i \leftrightarrow \mathbf{p}_2^i\}$ should be normalized in scale/norm, otherwise the above error function is not invariant to scaling of F or the coordinates of the points.

A better quality criterion is given by the function `DistPoint2EpipolarLine()` provided, since it measures a geometric quantity in the image plane: the Euclidean distance from points to their epipolar lines given by the estimated fundamental matrix F. Specifically, this function computes the Root-Mean-Square error

$$\Big(\frac{1}{N}\sum_{i=1}^{N}\left(d_\perp^2(\mathbf{p}_1^i, \boldsymbol{\ell}_1^i) + d_\perp^2(\mathbf{p}_2^i, \boldsymbol{\ell}_2^i)\right)\Big)^{\frac{1}{2}},$$

where $\boldsymbol{\ell}_1^i = F^\top \mathbf{p}_2^i$ and $\boldsymbol{\ell}_2^i = F \mathbf{p}_1^i$ are the epipolar lines in images 1 and 2, respectively, and $d_\perp(\mathbf{p}, \boldsymbol{\ell})$ measures the point-to-line distance in the image planes.

The script also provides code to test in case of additive noise in the coordinates of the point correspondences.

2.1 Normalized eight-point algorithm

As seen in class, if there is a significant difference between the orders of magnitude of the individual coordinates of the points $\mathbf{p}_1^i, \mathbf{p}_2^i$, or if there are significant offests, the numerical conditioning of the system of equations in the eight-point algorithm is poor. This can be fixed using a normalized eight-point algorithm, which estimates the fundamental matrix on a set of normalized correspondences (with better numerical properties) and then unnormalizes the result to obtain the fundamental matrix for the given (unnormalized) correspondences. The algorithm has the following steps:

1. Normalize point correspondences: $\{\mathbf{p}_1^i \leftrightarrow \mathbf{p}_2^i\} \longrightarrow \{\tilde{\mathbf{p}}_1^i \leftrightarrow \tilde{\mathbf{p}}_2^i\}$, where $\tilde{\mathbf{p}}_j^i = T_j \mathbf{p}_j^i$ for $j = 1, 2$.
2. Estimate fundamental matrix using eight-point algorithm: $\{\tilde{\mathbf{p}}_1^i \leftrightarrow \tilde{\mathbf{p}}_2^i\} \longrightarrow \tilde{F}$
3. Unnormalize on the fundamental matrix: $\tilde{F} \longrightarrow F = T_2^\top \tilde{F} T_1$.

Next, you are asked to use the function `normalise2dpts()` provided to fill in the gaps in function `FundamentalEightPoint_Normalized()` that implements the previous three steps. Use the script provided `run_test_8point.m` to validate the functions you have completed.

3 Line fitting using RANSAC

The script `run_RANSAC_line.m` implements a line fitting algorithm based on RANdom SAmple Consensus (RANSAC). The algorithm is robust since it can tolerate the presence of outliers in the data, as shown in Fig. 1. Given a set of M points with coordinates $S = \{(x_i, y_i)\}_{i=1}^{M}$, the algorithm returns the homogeneous coordinates of a line $\boldsymbol{\ell} = (a, b, c)^\top$ that fits the points (it tries to satisfy the equations $ax_i + by_i + c = 0$).

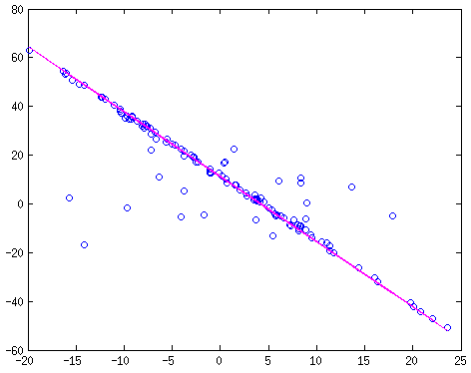

Figure 1: Robust line fitting using RANSAC.

Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Algorithm 4.4. *The RANSAC robust estimation algorithm, adapted from [Fischler-81]. A minimum of s data points are required to instantiate the free parameters of the model. The three algorithm thresholds t, T, and N are discussed in the text.*

Figure 2: RANSAC algorithm.

The RANSAC algorithm is summarized in Fig. 2, for a generic problem. We will use the RANSAC implementation provided by Dr. P. Kovesi[1] in the file `ransac.m`. Review the code in this file and try to understand it by identifying those statements that implement the steps in Fig. 2. The function `ransac` requires the input of several parameters and three functions, which we provide for the case of line estimation. In this case, we have that $s = 2$ points suffice to instantiate a model, that is, a line $\boldsymbol{\ell}$.

- `line_isdegenerate.m` : to validate a set of data points as not being in a degenerate configuration (Step i in Fig. 2). A degenerate configuration gives an undetermined model; this can only happen if the $s = 2$ points randomly picked points coincide.

[1] http://www.csse.uwa.edu.au/∼pk/Research/MatlabFns/

- `line_fitting.m` : to estimate/instantiate a model from a (minimal) set of data points (Step i in Fig. 2): $\boldsymbol{\ell} = \mathbf{p}_1 \times \mathbf{p}_2$, where $\mathbf{p}_1, \mathbf{p}_2$ are the homogeneous coordinates of the two selected points.
- `line_dist.m` : to classify data points as inliers or outliers to the model (Step ii in Fig. 2), according to some distance, for example the Euclidean distance (perpendicular distance from each point to the line).

The classification threshold (t in Fig. 2) is chosen according to the expected noise of the non-outlier data points.

4 Fundamental matrix estimation using RANSAC

In this section RANSAC is applied to the fundamental matrix problem. For that, we use the function in `ransac.m`, but with different parameters. In this problem, the data consists of a set of point correspondences $S = \{\mathbf{p}_1^i \leftrightarrow \mathbf{p}_2^i\}_{i=1}^M$, and the model is given by a 3×3 (fundamental) matrix F. The RANSAC algorithm returns a model instance F that fits the point correspondences by trying to satisfy the epipolar constraints: $\mathbf{p}_2^{i\top} F \mathbf{p}_1^i = 0$. Although a minimal sample to estimate F just requires 7 points, we use the 8-point algorithm already coded in section 2, so $s = 8$ points is the size of the sample used to instantiate a model F. The classification threshold t is chosen according to the expected noise of the data points. In practice, t is given values between 1-3 pixels, depending on the image size.

Look at the code provided in file `run_RANSAC_8point.m` (which is very similar to that seen before in `run_test_8point.m`) and complete it along with the three functions required by the function `ransac`. We have partially given them in files `fund_isdegenerate.m`, `fund_fitting.m` and `fund_dist.m`. You are also required to fill in function `SqrdErrVec_fundmat_Sampson.m`, which is called from `fund_dist.m` and specifies the distance used in step ii of Fig. 2 for inlier/outlier classification.

5 Structure from Motion

To integrate the concepts seen in this lab session and in the previous ones, we implement a Structure from Motion pipeline, as seen in the lecture slides in file "07_multiple_view_geometry_2.pdf" and summarized in Fig. 3. To this end, we provide script `run_SFM.m`, which you are required to complete by integrating the MATLAB code written in previous sections.

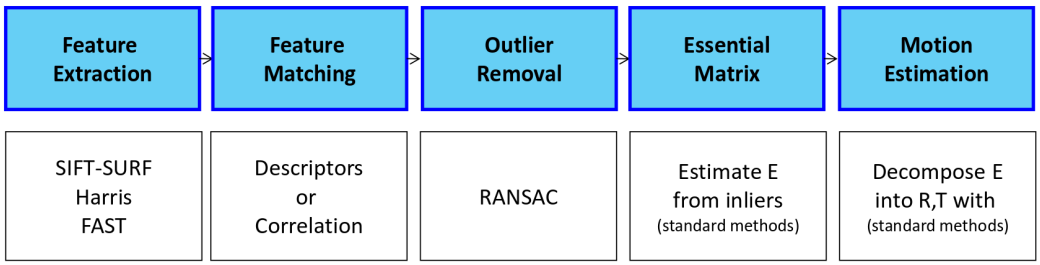

Figure 3: Structure from motion work flow.

Feature detection, extraction and matching. First, a pair of images are loaded in memory[2]. Then, BRISK features are extracted in both images (previously converted to grayscale) and matched across them. We use MATLAB's Computer Vision toolbox (CVT) implementation of BRISK. At a high level, these steps of feature detection, description and matching have been seen in the previous exercise session (Harris corners). If you are interested in more details, look at the help of functions `detectBRISKFeatures`, `extractFeatures` and `matchFeatures`. For example,

[2]We provide a dataset for which we know the calibration of the camera (i.e., the intrinsic parameters), so that we can compute the essential matrix E from the fundamental matrix F.

Figure 4: BRISK Feature detection in two images.

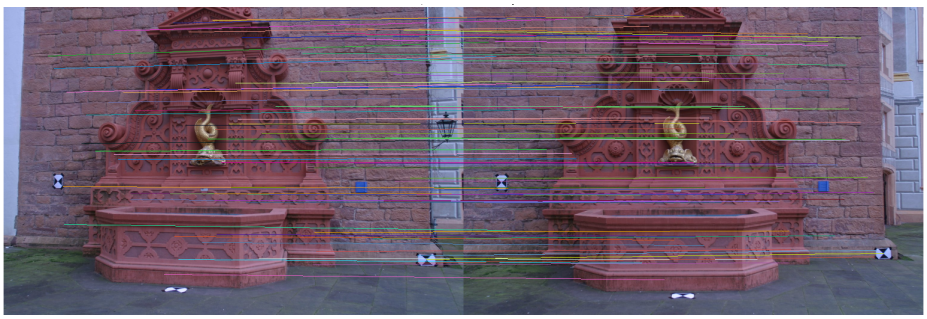

Figure 5: Candidate feature matching. (Before fundamental matrix estimation via RANSAC).

```
>> doc detectBRISKFeatures
```

One of the main parameters controlling the amount of features detected is the minimum intensity difference between a corner and its surrounding region. We decrease this value from its default one to get a large number of corners since the images provided do not have a good contrast. If you do not have the CVT, the matches produced by these steps can be loaded from the file `raw_matches.mat`. The output you should get would be similar to that shown in Figs. 4 and 5.

Estimation of Fundamental Matrix and outlier rejection. In the third block of Fig. 3, the RANSAC algorithm coded in section 4 is applied to the putative matches to estimate the fundamental matrix and the inliers consistent with the model. Then, the fundamental matrix is re-estimated using the 8-point algorithm on all the inliers. We introduce a similarity transformation in the image plane to improve the numerical conditioning of the estimation problem in this and subsequent steps, in the same spirit of the normalized 8-point algorithm. Fill in the missing lines of code to call RANSAC and to display the inliers. The results you obtain should be similar to those in Fig. 6.

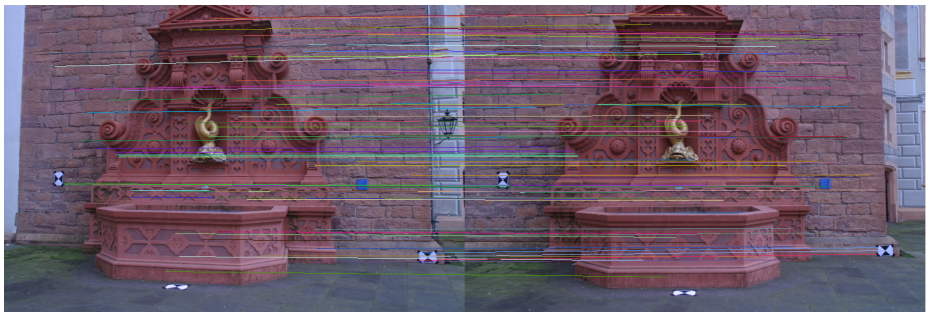

Figure 6: Inliers (matched features) using the fundamental matrix RANSAC algorithm.

Essential matrix and camera motion estimation. In the next step, the essential matrix is computed from the fundamental matrix and the intrinsic parameters of both cameras, according to formula $E = K_2^\top F K_1$. Then, E is decomposed into the relative camera motion $(R, \mathbf{t})$ between both cameras, taking the first of them as the origin of the world coordinates. There are four possible solutions $(R_i, \mathbf{t}_i)$ that are consistent with the essential matrix E, however, only one of those solutions is physically feasible in the sense that the triangulated 3-D points using the camera motion are in front of both cameras. Complete the code to call the triangulation function and therefore disambiguate the relative camera motion. A function to compute the depth of triangulated 3-D points with respect to a camera is given.

Finally, complete the code to visualize the reconstructed 3-D points and camera poses. Recall that the world origin is in the camera corresponding to the first image. Your results should be similar to those in Figs. 7 and 8.

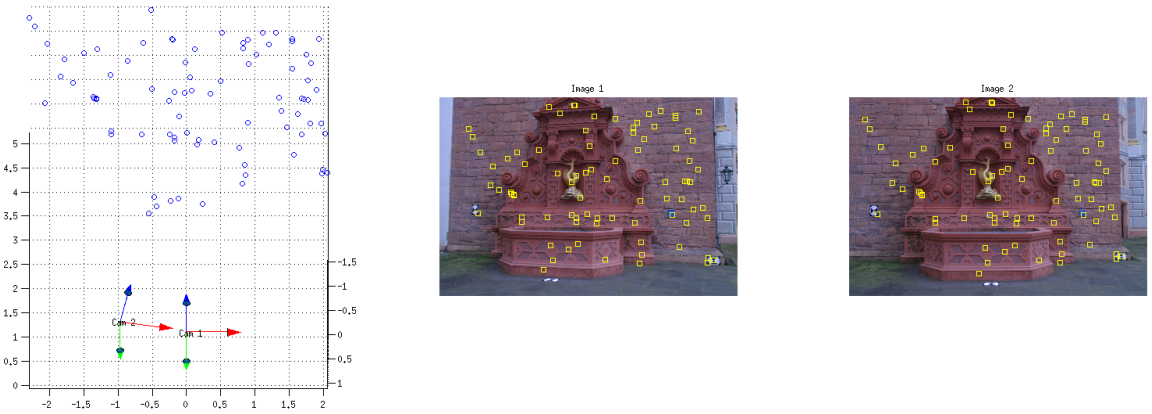

Figure 7: 3-D scene (sparse points and camera poses) and matched image features.

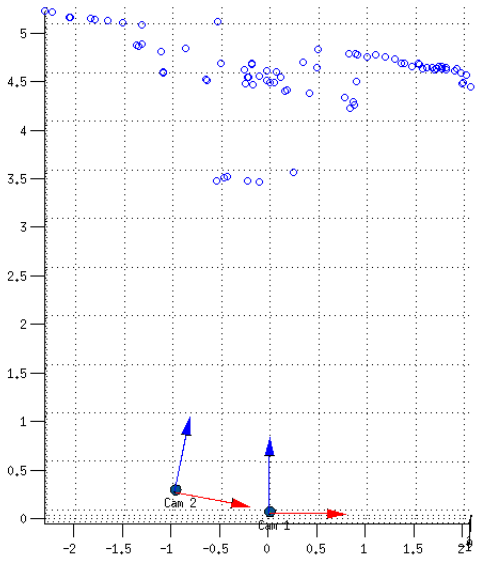

Figure 8: Top view of the reconstructed scene.