Darius Rückert
AmirAbbas Davari
21.05.2018
LME
LGDV COMPUTER GRAPHICS • ERLANGEN

The deadline for this exercise is on Sunday 3.06.2018 at 23:59

Structure from Motion

1 Introduction

In this exercise, we will compute the relative transformation of two cameras, i.e. rotation and translation, and the 3D position of a set of feature matches. Consider Fig. 1 that shows a *memorial shield* captured from two different angles. Given a point x in image 1 and the corresponding match x' in image 2, the following constraint can be derived from epipolar geometry (see lecture):

$$\mathbf{x}'^T \mathbf{F} \mathbf{x} = 0, \tag{1}$$

The 3×3 matrix $\mathbf{F}$ is the fundamental matrix, mapping points in one image to epipolar lines in the other image.

Figure 1: A *memorial shield* captured from two different viewpoints at the Germanic National Museum.

2 Computation of F [5 Points]

Similar to the homography computation of the last exercise sheet, the fundamental matrix can be computed from 8 matching feature points. This procedure is wrapped in a RANSAC algorithm to account for incorrect matches.

2.1 Distance from Epipolar Line

Implement the function `epipolarConstraint` in `fundamentalMatrix.cpp` that checks, if a given match satisfies the epipolar constraint. This is the case, if the distance of x' to the epipolar line of x is smaller than a threshold t.

- Compute the epipolar line $l = (l_x, l_y, l_w) = Fx$
- Normalize l to get the scale independent line $l' = l/\sqrt{l_x^2 + l_y^2}$
- Compute the point line distance $d = x'^T l'$
- A match is an inlier if $|d| < t_e$

2.2 8-Point Algorithm

Given 8 correct feature matches, the fundamental matrix can be computed with the 8-point algorithm. The principle is similar to the homography computation from the last exercise sheet, for which only 4 correct matches are required.

Implement the function `computeF` in `fundamentalMatrix.cpp` with the following steps:

1. Construct the matrix A. (Note: 1 row per match = 8 rows in total)

$$A = \begin{bmatrix} p_{x1}q_{x1} & p_{x1}q_{y1} & p_{x1} & p_{y1}q_{x1} & p_{y1}q_{y1} & p_{y1} & q_{x1} & q_{y1} & 1 \\ p_{x2}q_{x2} & p_{x2}q_{y2} & p_{x2} & p_{y2}q_{x2} & p_{y2}q_{y2} & p_{y2} & q_{x2} & q_{y2} & 1 \\ \dots & & & & & & & & \\ \dots & & & & & & & & \\ p_{x8}q_{x8} & p_{x8}q_{y8} & p_{x8} & p_{y8}q_{x8} & p_{y8}q_{y8} & p_{y8} & q_{x8} & q_{y8} & 1 \end{bmatrix}$$

2. Solve $Af = 0$ and extract F (already implemented).
3. Make sure that $Rank(F) = 2$.
   - Compute the SVD $F = U\Sigma V^T$ with

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}$$

   - Set $\sigma_3 = 0$
   - Recompute F with the updated Σ.
4. Normalize F (already implemented).

At the beginning of `main` a test is executed to validate your 8-point implementation. This test should succeed before continuing with the next tasks.

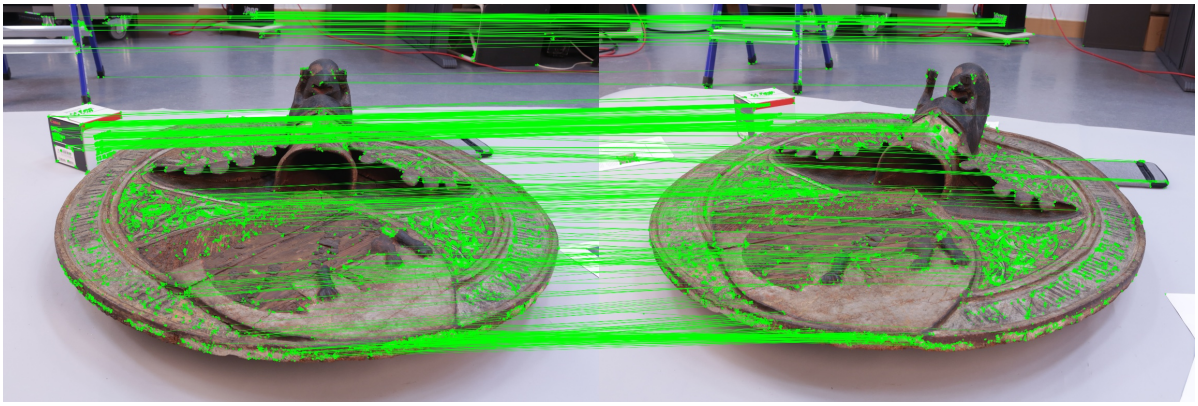

Figure 2: The *geometric consistent* matches. That are all matches passing the *ratio test* **and** satisfying the epipolar constraint.

3 Triangulation [5 Points]

Triangulation describes the method of finding the position of a point in 3D space given its image in two views and the camera parameters of both views (see Figure 3).

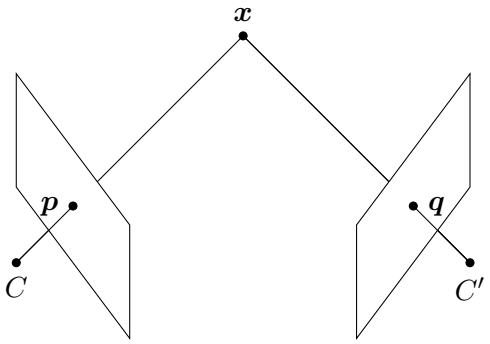

Figure 3: A point $\boldsymbol{x}$ is seen by two cameras. The rays starting from each camera center passing through the imaged points $\boldsymbol{p}$ and $\boldsymbol{q}$ intersect at the world point $\boldsymbol{x}$.

Given the pinhole camera model with the projection matrices $M_i = V_i K_i$ the projection equations

$$\boldsymbol{p} = \boldsymbol{M_1 x}$$

$$\boldsymbol{q} = \boldsymbol{M_2 x},$$

can be transformed to the homogeneous linear system of equations $Ax = 0$ with

$$\boldsymbol{A} = \begin{bmatrix} p_x \boldsymbol{m}_1^3 - \boldsymbol{m}_1^1 \\ p_y \boldsymbol{m}_1^3 - \boldsymbol{m}_1^2 \\ q_x \boldsymbol{m}_2^3 - \boldsymbol{m}_2^1 \\ q_y \boldsymbol{m}_2^3 - \boldsymbol{m}_2^2 \end{bmatrix},$$

where $\boldsymbol{m}_i^j$ is the j-th row-vector of M_i.

Implement the function `triangulate` in `triangulate.cpp` with the description from above. Check if the test case, which is executed in `main`, succeeds before you continue with the exercise.

4 Relative Transformation [5 Points]

To compute the relative transformation (rotation and translation) between the input images, first, the essential matrix E is computed from the fundamental matrix F and the camera intrinics K:

$$E = K^T F K$$

The essential matrix is then decomposed into two rotation matrices and translation vectors, giving a total number of 4 possible transformations per essential matrix. For each relative transformation, the feature matches are triangulated (Task 3) and projected to each image. Only one solution over all pose configurations is geometrically valid, which means that all 3D points lie in front of both cameras. The other configurations have at least one point that is behind one or both cameras.

4.1 Decomposition

Implement the function `decompose` in `decompose.cpp` with the following steps:

1. Compute the SVD $E = U\Sigma V^T$
2. The two possible rotations are

   $$R_1 = UWV^T$$
   $$R_2 = UW^TV^T$$

   with

   $$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. The two possible translations are

   $$t_1 = s$$
   $$t_2 = -s$$

   with

   $$s = u_3/|u_3|$$

   and u_3 being the third column vector of U.

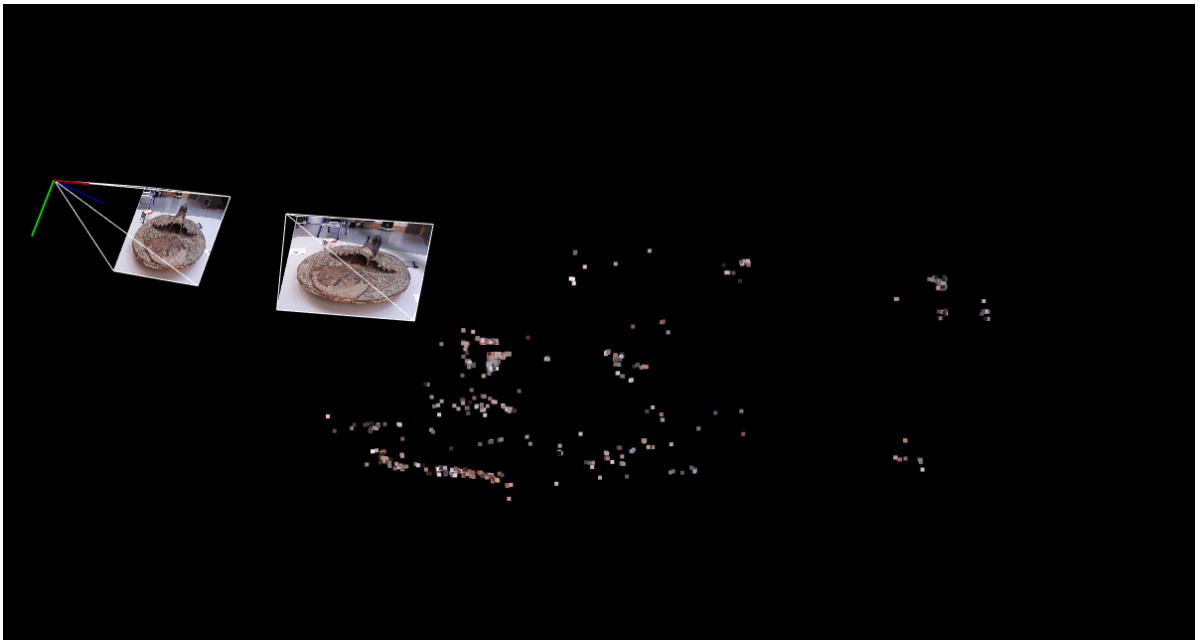

Figure 4: The two view reconstruction and final output after task 4. We have computed the relative transformation between both views and a set of 3D points that are seen by both cameras.