DATA.ML.300 Computer Vision Exercise 5

Miska Romppainen
H274426

February 2021

1 Task: Fundamental Matrix and Essential Matrix

In the task 1 we were given 4 questions to answer. The questions were:
a) For what purpose are these matrices used and what are their differences?
b) How can you derive essential matrix from fundamental matrix and what additional information you need in order to do that?
c) How many degrees of freedom does fundamental matrix have and why?
d) How many degrees of freedom does essential matrix have and why?
Here are my answers to the questions.

Question a: For what purpose are these matrices (Fundamental Matrix and Essential Matrix) used and what are their differences?
Fundamental Matrix and Essential Matrix are matrices used in stereo geometry where we want to describe geometric relations of image pairs. For example a location of cameras related to one and other between two images taken from a different perspective. The difference between the two matrices is that the essential matrix is used for calibrated camera and the fundamental matrix is used for non-calibrated camera. In other words essential matrix assumes you know the normalized coordinates whereas the fundamental matrix assumes you do not know the normalized coordinates.

Question b: How can you derive essential matrix from fundamental matrix and what additional information you need in order to do that?
The equation for the essential matrix is the following:

$$E = [t_x]R, \tag{1}$$

where t_x is the translation vector turned into 3x3 matrix and R is the rotation matrix. We can also show that knowing point x' from an image and knowing point x from another image, we can show that the relation between these points are:

$$x'^T E x = 0 \tag{2}$$

We can also show that Ex is the epipolar line associated with

$$x(l' = Ex) \tag{3}$$

If we do not know the epipolar constraint in terms of unknown normalized coordinates for point x' from an image and point x from another, we can write the equation (2) as follows:

$$\hat{x}'^T E \hat{x} = 0 \tag{4}$$

If we intepret that $\hat{x} = K^{-1}x$, where K is the calibration matrix. We can similarly write $\hat{x}' = K'^{-1}x'$. Using these we can write the equation 4 as follows:

$$(K'^{-1}x')^T E(K^{-1}x) = 0 \tag{5}$$

By rearranging the parameters we get

$$x'^T(K'^{-T}EK^{-1})x = 0 \tag{6}$$

Here we can write that $F = K'^{-T}EK^{-1}$, where F is the fundamental matrix and we get

$$x'^T Fx = 0 \tag{7}$$

This is how we can derive fundamental matrix from essential matrix.

Question c: How many degrees of freedom does fundamental matrix have and why?
Fundamental matrix has 7 degrees of freedom, since it describes the rotation, translation and the calibration of the internal parameters for non-normalized coordinates.

Question d: How many degrees of freedom does essential matrix have and why?
Essential matrix has 5 degrees of freedom, since it describes the rotation and translation of the normalized coordinates.

2 Task: Camera calibration

Here is my solution to task 2. Camera calibration. Idea of the task was to implement a code of direct linear transform (DLT) method for camera calibration. The calibration object is a bookshelf whose dimensions are known. That is, width of a shelf is 758 mm, depth is 295 mm, and height between shelves is 360 mm.

The function *camcalibDLT* takes the homogeneous coordinates (world coordinates and image coordinates) of the points as input and outputs camera projection matrix. Snippet of the code can be see below. DLT algorithm is described as

$$\begin{pmatrix} 0^T & X_1^T & -y_1 X_1^T \\ X_1^T & 0^T & -x_1 X_1^T \\ ... & ... & ... \\ 0^T & X_n^T & -y_n X_n^T \\ X_n^T & 0^T & -x_n X_n^T \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0 \tag{8}$$

Where X_n is the known 3D coordinate, and x_n and y_n are the known image projection coordinates. The task was to find P minimizing $||Ap||^2$ (homogeneous least squares). By perform homogeneous least squares fitting and finding the eigenvector of $A^T A$ with the smallest eigenvalue we can find the solution. The output images can be seen in figures 1 and 2.

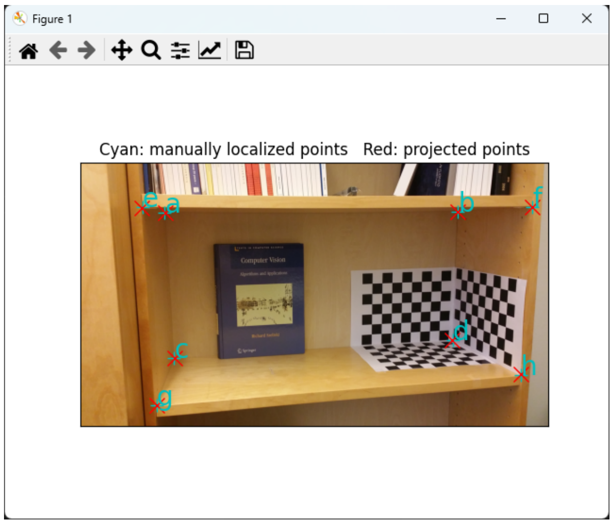


Figure 1: Output figure 1 of calibrated points

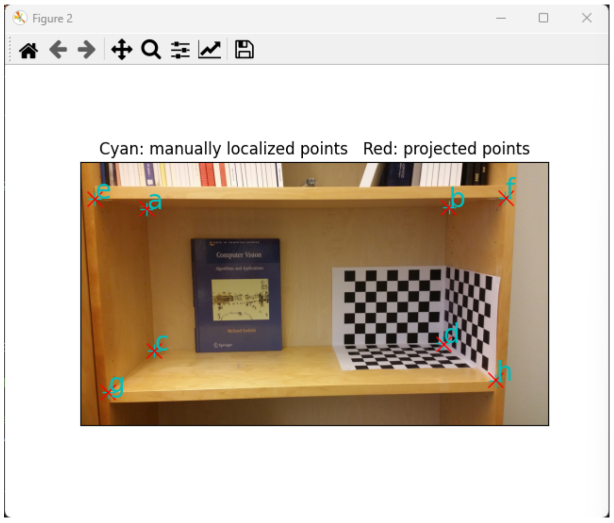


Figure 2: Output figure 2 of calibrated points

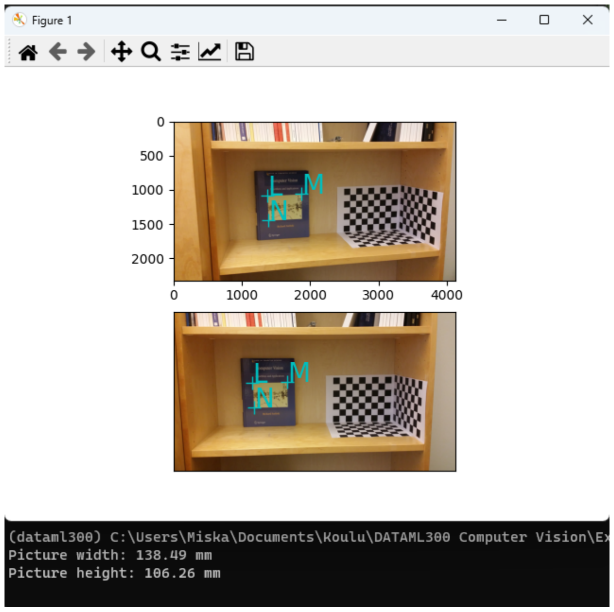
CODE SNIPPET FROM: *camcalibDLT.py*

```
def camcalibDLT(x_world, x_im):
    """
    :param x_world: World coordinatesm with shape (point_id, coordinates)
    :param x_im: Image coordinates with shape (point_id, coordinates)
    :return P: Camera projection matrix with shape (3,4)
    """

    # Create the matrix A
    ##-your-code-starts-here-##

    ##-your-code-ends-here-##

    # Create the matrix A
    n = x_world.shape[0]
    A = np.zeros((2 * n, 12))

    for i in range(n):
        X = x_world[i][0]
        Y = x_world[i][1]
        Z = x_world[i][2]
        u = x_im[i][0]
        v = x_im[i][1]
        A[2*i] = [X, Y, Z, 1, 0, 0, 0, 0, -u*X, -u*Y, -u*Z, -u]
        A[2*i+1] = [0, 0, 0, 0, X, Y, Z, 1, -v*X, -v*Y, -v*Z, -v]

    # Perform homogeneous least squares fitting.
    # The best solution is given by the eigenvector of
    # A.T*A with the smallest eigenvalue.
    U, s, Vt = np.linalg.svd(A)
    ev = Vt[-1]

    ##-your-code-ends-here-##

    # Reshape the eigenvector into a projection matrix P
    P = np.reshape(ev, (3, 4))  # here ev is the eigenvector from above
    #P = np.array([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1]], dtype=float)
# remove this and uncomment the line above

    return P
```

3 Task: Triangulation

Here is my solution to task 3. triangulation. The goal of the task was to calculate the size of the Richard Szeliski's books cover image using triangulation presented in the lecture slides and in the exercise PDF. Using the presented cross-product as a matrix multiplication we can calculate two independent equations in terms of X and using *np.stack*-function stack these vertically to acquire A and solve the homogeneous linear system $Ax = 0$. After which we can convert the triangulated points to Cartesian coordinates and compute the vectors corresponding to the book cover's horizontal and vertical sides.

The output of my code can be seen in the figure 3, where the estimated picture width is 138.49mm and estimated picture height 106.26mm. According to google Richard Szeliski's book *Computer Vision: Algorithms and Applications* dimensions are 22.6 x 28.7 cm, where the image size of 13.8 x 10.6 cm sounds extremely plausible.



```
(dataml300) C:\Users\Miska\Documents\Koulu\DATAML300 Computer Vision\Ex
Picture width: 138.49 mm
Picture height: 106.26 mm
```

Figure 3: Output of *triangulation_task.py*

CODE SNIPPET FROM: *trianglin.py*

```
def trianglin(P1, P2, x1, x2):
    """
    :param P1: Projection matrix for image 1 with shape (3,4)
    :param P2: Projection matrix for image 2 with shape (3,4)
    :param x1: Image coordinates for a point in image 1
    :param x2: Image coordinates for a point in image 2
    :return X: Triangulated world coordinates
    """

    # Form A and get the least squares solution from the eigenvector
    # corresponding to the smallest eigenvalue
    ##-your-code-starts-here-##
    # Construct the matrix A
    lmn1 = x1
    lmn2 = x2
    A = np.vstack((lmn1[0] * P1[2] - P1[0],
                   lmn1[1] * P1[2] - P1[1],
                   lmn2[0] * P2[2] - P2[0],
                   lmn2[1] * P2[2] - P2[1]))

    # Solve the homogeneous linear system Ax = 0
    _, _, V = np.linalg.svd(A)
    X = V[-1, :]
    ##-your-code-ends-here-##

    return X
```

CODE SNIPPET FROM: *triangulation_task.py*

```
import numpy as np
import matplotlib.pyplot as plt
from trianglin import trianglin


# Visualization of three point correspondences in both images
im1 = plt.imread('im1.jpg')
im2 = plt.imread('im2.jpg')

# Points L, M, N (corners of the book) in image 1
lmn1 = 1.0e+03 * np.array([[1.3715, 1.0775],
                            [1.8675, 1.0575],
                            [1.3835, 1.4415]])

# Points L, M, N (corners of the book) in image 2
lmn2 = 1.0e+03 * np.array([[1.1555, 1.0335],
                            [1.6595, 1.0255],
                            [1.1755, 1.3975]])

# Annotate and show images
labels = ['L', 'M', 'N']
plt.figure()
plt.subplot(2, 1, 1)
plt.imshow(im1)
for i in range(len(labels)):
    plt.plot(lmn1[i, 0], lmn1[i, 1], 'c+', markersize=10)
    plt.annotate(labels[i], (lmn1[i, 0], lmn1[i, 1]), color='c', fontsize=20)
plt.subplot(2,1,2)
plt.imshow(im2)
for i in range(len(labels)):
    plt.plot(lmn2[i, 0], lmn2[i, 1], 'c+', markersize=10)
    plt.annotate(labels[i], (lmn2[i, 0], lmn2[i, 1]), color='c', fontsize=20)
plt.xticks([])
plt.yticks([])


# Your task is to implement the missing function 'trianglin.py'
# The algorithm is described in the lecture slides and exercise sheet.
# Output should be the homogeneous coordinates of the triangulated point.

# Load the pre-calculated projection matrices
P1 = np.load('P1.npy')
P2 = np.load('P2.npy')

# Triangulate each corner
L = trianglin(P1, P2,
              np.hstack((lmn1[0, :].T, [1])),
              np.hstack((lmn2[0, :].T, [1])))
M = trianglin(P1, P2,
              np.hstack((lmn1[1, :].T, [1])),
              np.hstack((lmn2[1, :].T, [1])))
N = trianglin(P1, P2,
              np.hstack((lmn1[2, :].T, [1])),
              np.hstack((lmn2[2, :].T, [1])))
```

```
# We can then compute the width and height of the picture on the book cover
# Convert the above points to cartesian, form vectors corresponding to
# book covers horizontal and vertical sides using the points and calculate
# the norm of these to acquire the height and width (mm).
##-your-code-starts-here-##
# Convert the triangulated points to cartesian coordinates
L = L[:3] / L[-1]
M = M[:3] / M[-1]
N = N[:3] / N[-1]

# Compute the vectors corresponding to the book cover's horizontal and vertical sides
u = M - L
v = N - L

picture_w_mm = np.linalg.norm(u)
picture_h_mm = np.linalg.norm(v)
##-your-code-ends-here-##
print("Picture width: %.2f mm" % picture_w_mm)
print("Picture height: %.2f mm" % picture_h_mm)
plt.show()
```