Computer Vision and Photogrammetry
2nd Task

Nikolaos Theokritos Tsopanidis aivc24022 03/06/2025

Περιεχόμενα

Import 1
Request 1st 1
Projection Table P 1
Mean Square Remaining Pixels Error σ₀ 2
Request 2nd 3
K Camera Matrix 3
Translation vector 3
Python Code 3
Material 5

Import

There are 2 questions for the 2nd task:

1. Calculate the projection matrix P, as well as the mean square error of the reprojection s0.
2. Extract the camera matrix K and the elements of the external orientation (rotation table R and the shift vector).

Request 1st

Projection Table P

According to the instructions given for the implementation of the task, the projection matrix **P** will be calculated by applying the technique of analysis of **peculiar values SVD** to **A**. The coefficient table **A** is created by extracting the coefficients of the functions from the collinearity condition for each point with pixelated coordinates (x,y) and the corresponding coordinates in 3D space (X, Y,Z). According to the implementation of Professor David Kriegman of the University of California, San Diego, we calculate table A in Python code as follows:

```
A = np.zeros(shape=(2*object_points_m.shape[0], 12))
for i in range(object_points_m.shape[0]):
    x, y = image_points_pix[i, 0], image_points_pix[i, 1]
    X, Y, Z = object_points_m[i, 0], object_points_m[i, 1], object_points_m[i, 2]
    # coefficients of the collinearity condition equations
    A[2*i] = [-X, -Y, -Z, -1, 0, 0, 0, 0, x*X, x*Y, x*Z, x]
    A[2*i+1] = [0, 0, 0, 0, -X, -Y, -Z, -1, y*X, y*Y, y*Z, y]
```

Consequently, using the 'svd**()**' algorithm of 'scipy' we factor table A into tables **D**, **U** and **V**. The approximate elements of the projection table are obtained from the last line of the rectangular table V where we then form the P with dimensions of 3x4:

```
# approximating projection matrix
P_appr = Vh[-1, :]
P_appr = P_appr. Reshape(3, 4)
P_appr = P_appr / P_appr[2, 3]  # normalization
print(P_appr)
```

```
[[ 1.39840170e+02  1.58089836e+00 -2.63615643e+01  1.55108314e+02]
```

```
 [-9.12728974e-01  1.41950386e+02  2.14996981e+00 -2.88690618e+02]
 [-1.15080241e-02  3.62972440e-04 -5.63994371e-02  1.00000000e+00]]
```

Mean Square Remaining Pixels Error σ_o

To calculate the quadratic error, we first re-project the three-dimensional points to points with pictorial coordinates (x,y) and find the difference **v** with the coordinates given to us in the exercise.

```
# Re-projection of 3D points to 2D points
hom_object_points_pix = hom_object_points_m @ P_appr.T
hom_object_points_pix /= hom_object_points_pix[:, 2].reshape(-1, 1)
hom_object_points_pix = hom_object_points_pix[:, :2]
print(hom_object_points_pix)
```

```
[[-8.59716709e+02 -2.29284447e+02]
 [-8.30095698e+02 -6.52929122e+02]
 [-7.67254311e+02  3.73957138e+02]
 [-6.38594537e+02 -6.73686078e+02]
 [-5.06310089e+02 -1.65767266e+02]
 [-6.43047515e+02 -2.08091320e+02]
 [-6.96197878e+02  2.44894309e+02]
 [-6.57249176e+02  4.80978393e+02]
 [-3.43320727e+02  8.50024194e+02]
 [-2.41072453e+02  5.18153845e+01]
 [ 3.91146973e+01  5.03215768e+02]
 [ 4.33444708e+01  2.53399560e+02]
 [ 7.80893471e+00 -2.14391496e+02]
 [ 1.11795880e-02 -7.78701978e+02]
 [ 2.72195142e+02  1.86135133e+02]
 [ 6.15063309e+02  4.27412661e+02]
 [ 5.93830303e+02 -1.54618177e+02]
 [ 6.14436276e+02 -6.97166763e+02]]
```

```
v = image_points_pix - hom_object_points_pix
print(v)
```

```
[[ 0.58762208  0.54924015]
 [-0.35581227  0.26243325]
 [-0.56050628  0.07395377]
 [-0.47337644 -0.05963275]
 [ 0.36397179 -0.43384123]
 [ 0.36194998 -0.53561733]
 [ 0.10635283 -0.04779437]
 [-0.28071235 -0.04077907]
 [-0.00258417 -0.23669639]
 [ 0.33939525 -0.71056296]
 [-0.55915313  0.43670872]
 [ 0.57840927  0.39170199]
 [ 0.26890028 -0.06252757]
 [-0.34377115  0.22235376]
 [ 0.52698099  0.37288511]
 [-0.42259926 -0.23176097]
 [ 0.0132993   0.55003052]
 [-0.15886605 -0.48618146]]
```

Therefore, we apply the formula:

$$\sigma_o = \pm\sqrt{\frac{\mathbf{v}^T\mathbf{v}}{r}} = \pm\sqrt{\frac{\Sigma(v_x^2 + v_y^2)}{r}}$$

to calculate the error σ_o :

```
Vx = v[:, 0]
Vy = v[:, 1]
V_er = np.sum(Vx**2 + Vy**2)
sigma_0 = np.sqrt(V_er / 25)
print("Reprojection Error:", sigma_0)
```

```
Reprojection Error: 0.46247376635448867
```

Request 2nd

K Camera Matrix

To extract the K Camera Matrix, the decomposecamera.m algorithm was used by doing QR analysis on the P panel.

```
K, Rc_w = decomposecamera(P_appr)
K /= K[2, 2]
print("\nNormalized Camera Matrix K:\n", K)
print("\nRotation Matrix R:\n", Rc_w)
```

```
Warning: Note that rotation matrix is left handed

Normalized Camera Matrix K:
 [[ 1.17382366e-02  3.95379238e-01 -5.27242820e+00]
 [ 2.06616859e-18  5.35401454e+00  3.76686727e-01]
 [-3.08842219e-19 -8.90077925e-18  1.00000000e+00]]

Rotation Matrix R:
 [[ 2.13771911e-03 -2.12506989e-05 -9.99997715e-01]
 [ 8.12867695e-02  9.96690743e-01  1.52588254e-04]
 [-9.96688462e-01  8.12869100e-02 -2.13237225e-03]]
```

By using the algorithm, the rotation table is additionally extracted, where we can calculate the angles of rotation ω, ϕ, κ.

```
r_, p_, y_ = get_euler_from_rotation_matrix(Rc_w)
print("\nRoll:", r_)
print("Pitch:", p_)
print("Yaw:", y_)
```

```
Roll: 91.50267634318419
Pitch: 85.33585126608806
Yaw: 88.49355487276647
```

Translation vector

Having calculated the view table P, we can find the values of the translation vector (xo,yo) by taking the first two elements of the fourth column of P.

```
T_appr = P_appr_norm[:2, 3]
print("\nTranslation Vector:\n", T_appr)
```

```
Translation Vector:
 [ 155.10831372 -288.69061788]
```

Python Code

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from scipy.linalg import svd

def get_rotation_matrix_from_euler(roll, pitch, yaw):
    """
    by inserting roll, pitch and yaw angles in degrees, this function returns the rotation matrix

    :param roll: rotation around x-axis in degrees
    :param pitch: rotation around y-axis in degrees
    :param yaw: rotation around z-axis in degrees
    :return: rotation matrix
    """
    roll = np.radians(roll)
    pitch = np.radians(pitch)
    yaw = np.radians(yaw)
    # Rotation matrix for roll (rotation around x-axis)
    R_x = np.array([
        [1, 0, 0],
        [0, np.cos(roll), -np.sin(roll)],
        [0, np.sin(roll), np.cos(roll)]
    ])
    # Rotation matrix for pitch (rotation around y-axis)
    R_y = np.array([
        [np.cos(pitch), 0, np.sin(pitch)],
        [0, 1, 0],
        [-np.sin(pitch), 0, np.cos(pitch)]
    ])
    # Rotation matrix for yaw (rotation around z-axis)
    R_z = np.array([
        [np.cos(yaw), -np.sin(yaw), 0],
        [np.sin(yaw), np.cos(yaw), 0],
        [0, 0, 1]
    ])
    # Combined rotation matrix: R = R_z * R_y * R_x
    R = R_z @ R_y @ R_x
    return R

def rq3(A):
    """https://www.peterkovesi.com/matlabfns/Projective/rq3.m"""
    if not np.all(np.array(A.shape) == [3, 3]):
        raise ValueError('A must be 3x3')
    eps = 1e-10
    A[2, 2] += eps
    c = -A[2, 2] / np.sqrt(A[2, 2]**2 + A[2, 1]**2)
    s = A[2, 1] / np.sqrt(A[2, 2]**2 + A[2, 1]**2)
    Qx = np.array([[1, 0, 0], [0, c, -s], [0, s, c]])
    R = np.dot(A, Qx)
    R[2, 2] += eps
    c = R[2, 2] / np.sqrt(R[2, 2]**2 + R[2, 0]**2)
    s = R[2, 0] / np.sqrt(R[2, 2]**2 + R[2, 0]**2)
    Qy = np.array([[c, 0, s], [0, 1, 0], [-s, 0, c]])
    R = np.dot(R, Qy)
    R[1, 1] += eps
    c = -R[1, 1] / np.sqrt(R[1, 1]**2 + R[1, 0]**2)
    s = R[1, 0] / np.sqrt(R[1, 1]**2 + R[1, 0]**2)
    Qz = np.array([[c, -s, 0], [s, c, 0], [0, 0, 1]])
    R = np.dot(R, Qz)
    Q = Qz.T @ Qy.T @ Qx.T
    # Adjust R and Q so that the diagonal elements of R are +ve
    for n in range(3):
        if R[n, n] < 0:
            R[:, n] = -R[:, n]
            Q[n, :] = -Q[n, :]
    return R, Q

def decomposecamera(P):
    """https://www.peterkovesi.com/matlabfns/Projective/decomposecamera.m"""
    p1 = P[:, 0]
    p2 = P[:, 1]
    p3 = P[:, 2]
    M = np.array([p1, p2, p3])
    # Perform RQ decomposition of M matrix
    K, Rc_w = rq3(M)
    # Check that R is right handed, if not give warning
    if np.dot(np.cross(Rc_w[:, 0], Rc_w[:, 1]), Rc_w[:, 2]) < 0:
        print('Warning: Note that rotation matrix is left handed')
    return K, Rc_w
#------------------------------------------------------------------------------------------------------------------------------
with open('object_points_m.txt', 'r') as f:
    object_points_m = np.loadtxt(f, delimiter=',', unpack=True)

with open('image_points_pix.txt', 'r') as f:
    image_points_pix = np.loadtxt(f, delimiter=',', unpack=True)

hom_object_points_m = np.hstack((object_points_m, np.ones(shape=(object_points_m.shape[0], 1))))
hom_image_points_pix = np.hstack((image_points_pix, np.ones(shape=(image_points_pix.shape[0], 1))))

# https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf

A = np.zeros(shape=(2*object_points_m.shape[0], 12))
```

```
for i in range(object_points_m.shape[0]):
    x, y = image_points_pix[i, 0], image_points_pix[i, 1]
    X, Y, Z = object_points_m[i, 0], object_points_m[i, 1], object_points_m[i, 2]

    # coefficients of the collinearity condition equations
    A[2*i] = [-X, -Y, -Z, -1, 0, 0, 0, 0, x*X, x*Y, x*Z, x]
    A[2*i+1] = [0, 0, 0, 0, -X, -Y, -Z, -1, y*X, y*Y, y*Z, y]

U, S, Vh = svd(A)

# approximating projection matrix
P_appr = Vh[-1, :]
P_appr = P_appr.reshape(3, 4)
P_appr = P_appr / P_appr[2, 3]  # normalization
print(P_appr)

# Re-projection of 3D points to 2D points
hom_object_points_pix = hom_object_points_m @ P_appr.T
hom_object_points_pix /= hom_object_points_pix[:, 2].reshape(-1, 1)
hom_object_points_pix = hom_object_points_pix[:, :2]
print(hom_object_points_pix)

# calculating the reprojection mean squared error
v = image_points_pix - hom_object_points_pix
Vx = v[:, 0]
Vy = v[:, 1]
V_er = np.sum(Vx**2 + Vy**2)

sigma_0 = np.sqrt(V_er / 25)
print("Reprojection Error:", sigma_0)

K, Rc_w = decomposecamera(P_appr)
K /= K[2, 2]
print("\nNormalized Camera Matrix K:\n", K)
print("\nRotation Matrix R:\n", Rc_w)

r_, p_, y_ = get_euler_from_rotation_matrix(Rc_w)
print("\nRoll:", r_)
print("Pitch:", p_)
print("Yaw:", y_)

# we can extract the translation vector directly from the projection matrix
# as it is the last column of the projection matrix without the scale factor
# but first we have to normalize the projection matrix
P_appr_norm = P_appr / P_appr[2, 3]
T_appr = P_appr_norm[:2, 3]
print("\nTranslation Vector:\n", T_appr)
```

Material

- https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_estimation.pdf
- https://www.peterkovesi.com/matlabfns/Projective/rq3.m
- https://www.peterkovesi.com/matlabfns/Projective/decomposecamera.m