CS561 - HW 1

Amari Urquhart

A

Binocular stereo images taken of a shoe, with corresponding points labeled. I obtained the following image coordinates $(x_1, y_1, x_2, y_2, ...x_n, y_n)$

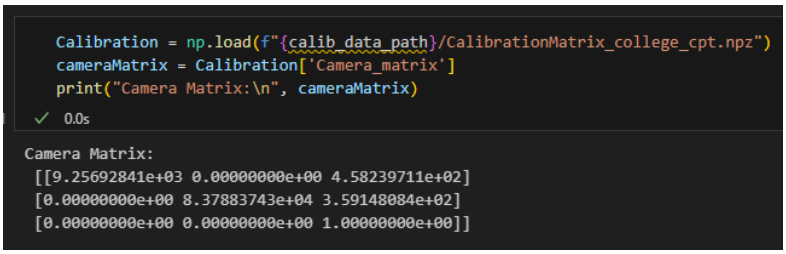
$img_1 coords =$ "$859, 2573, 1147, 2554, 1794, 2309, 2400, 1913, 2790, 1325, 2344, 1331, 2008, 1563, 1361, 2015$"

$img_2coords =$ "$536, 2431, 784, 2480, 1460, 2405, 2211, 2170, 2726, 1675, 2154, 1603, 1778, 1758, 1113, 2052$"

B

```
import cv2
import numpy as np

chessboardSquareMM = 13
checkerboardDimension = (10, 17)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

object3D = np.zeros((checkerboardDimension[0] * checkerboardDimension[1], 3), np.float32)
index = 0
for i in range(checkerboardDimension[0]):
    for j in range(checkerboardDimension[1]):
        object3D[index][0] = i * chessboardSquareMM
        object3D[index][1] = j * chessboardSquareMM
        index += 1

objPoints3D = []
imgPoints2D = []

checkerboards = [
    PATHS
]

for checkerboard in checkerboards:
    cbImage = cv2.imread(checkerboard)
    checkerboard = cv2.cvtColor(cbImage, cv2.COLOR_RGB2BGR)
    gray = cv2.cvtColor(checkerboard, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, checkerboardDimension, None)

    if ret:
        objPoints3D.append(object3D.copy())
        corners2 = cv2.cornerSubPix(gray, corners, (3, 3), (-1, -1), criteria)
        imgPoints2D.append(corners2)
        img = cv2.drawChessboardCorners(checkerboard, checkerboardDimension, corners2, ret)

        cv2.imshow('Chessboard Corners', checkerboard)
        cv2.waitKey(500)
        cv2.destroyAllWindows()

calibDataPath = PATH
```

```
ret, cameraMatrix, distCoeff, rotationalVecs, translationVecs = cv2.calibrateCamera(objPoint

np.savez(
    f"{calibDataPath}/CalibrationMatrix_college_cpt",
    Camera_matrix = cameraMatrix,
)
```



C

Assuming the world coordinate system is centered at 1st camera pinhole with Z along viewing direction, we use the following:

$$x'^T F x = 0$$

$$0 = u_1 u'_1 f_{11} + v_1 u'_1 f_{12} + u'_1 f_{13} + u_1 v'_1 f_{21} + v_1 v'_1 f_{22} + v'_1 f_{23} + u_1 f_{31} + v_1 f_{32} + f_{33}$$

with the following homogenous corresponding coordinate points:

$$x_i = (859, 2573, 1), (1147, 2554, 1), (1794, 2309, 1), (2400, 1913, 1), (2790, 1325, 1),$$
$$(2344, 1331, 1), (2008, 1563, 1), (1361, 2015, 1)$$

$$x'_i = (536, 2431, 1), (784, 2480, 1), (1460, 2405, 1), (2211, 2170, 1), (2726, 1675, 1),$$
$$(2154, 1603, 1), (1778, 1758, 1), (1113, 2052, 1)$$

Using the following

```
import numpy as np
from tabulate import tabulate

img1u = [859,1147,1794,2400,2790,2344,2008,1361]
img1v = [2573,2554,2309,1913,1325,1331,1563,2015]
img2u = [536,784,1460,2211,2726,2154,1778,1113]
img2v = [2431,2480,2405,2170,1675,1603,1758,2052]
n = 8
A = np.full((n, 9), int(0), dtype = object)
print(tabulate(A))
```

```
for i in range(n):
    A[i][0] = img1u[i] * img2u[i]
    A[i][1] = img1v[i] * img2u[i]
    A[i][2] = img2u[i]
    A[i][3] = img1u[i] * img2v[i]
    A[i][4] = img1v[i] * img2v[i]
    A[i][5] = img2v[i]
    A[i][6] = img1u[i]
    A[i][7] = img1v[i]
    A[i][8] = 1

print(tabulate(A))
```

We have the matrix A as:

$$A = \begin{bmatrix} 460424 & 1379128 & 536 & 2088229 & 6254963 & 2431 & 859 & 2573 & 1 \\ 899248 & 2002336 & 784 & 2844560 & 6333920 & 2480 & 1147 & 2554 & 1 \\ 2619240 & 3371140 & 1460 & 4314570 & 5553145 & 2405 & 1794 & 2309 & 1 \\ 5306400 & 4229643 & 2211 & 5208000 & 4151210 & 2170 & 2400 & 1913 & 1 \\ 7605540 & 3611950 & 2726 & 4673250 & 2219375 & 1675 & 2790 & 1325 & 1 \\ 5048976 & 2866974 & 2154 & 3757432 & 2133593 & 1603 & 2344 & 1331 & 1 \\ 3570224 & 2779014 & 1778 & 3530064 & 2747754 & 1758 & 2008 & 1563 & 1 \\ 1514793 & 2242695 & 1113 & 2792772 & 4134780 & 2052 & 1361 & 2015 & 1 \end{bmatrix}$$

Use this to solve Af = 0, where f is the least eigenvector of $A^T A$.

```
U, Sigma, VT = np.linalg.svd(A)

f = VT[-1]
print("f:", f)
F = f.reshape(3, 3)
```

f is obtained from the SVD of A and reorganized to obtain the fundamental matrix F. Force Rank 2 F by setting $\sigma_2 = 0$ (The Forced Rank 2 F technically have all non-zero values but they are very small).

```
f: [-4.73747043e-08 -3.38729797e-07  7.13832145e-04  4.88489898e-07
  6.30134314e-08 -7.98249754e-04 -9.62059862e-04  1.78422208e-04
  9.99998948e-01]
Fundamental Matrix:
 [[-4.73747043e-08 -3.38729797e-07  7.13832145e-04]
 [ 4.88489898e-07  6.30134314e-08 -7.98249754e-04]
 [-9.62059862e-04  1.78422208e-04  9.99998948e-01]]
Rank 2 Forced Fundamental Matrix:
 [[-4.76737023e-08 -3.39139403e-07  7.13832144e-04]
 [ 4.87807843e-07  6.20790638e-08 -7.98249754e-04]
 [-9.62059862e-04  1.78422207e-04  9.99998948e-01]]
```

Use previously obtained intrinsic parameter K in $E = K^T F K$

```
K = np.array([
```

```
    [9.25692841e+03, 0, 4.58239711e+02],
    [0, 8.37883743e+04, 3.59148084e+02],
    [0, 0, 1]
])

KTK = np.matmul(K.T, K)
print("KT * K:\n", KTK)
E = np.matmul(KTK, F2)
print("Essential Matrix:\n", E)

F2Rank = np.linalg.matrix_rank(F2)
ERank = np.linalg.matrix_rank(E)
print(F2Rank)
print(ERank)
```

```
KT * K:
 [[8.56907236e+07 0.00000000e+00 4.24189220e+06]
 [0.00000000e+00 7.02049167e+09 3.00924341e+07]
 [4.24189220e+06 3.00924341e+07 3.38971979e+05]]
Essential Matrix:
 [[-4.08503942e+03  7.27786668e+02  4.30305653e+06]
 [-2.55260721e+04  5.80498406e+03  2.44882967e+07]
 [-3.11634237e+02  6.09096460e+01  3.17978343e+05]]
2
2
```

Given E, t is the null values that minimize $E^T t = 0$

```
U, Sigma, VT = np.linalg.svd(E)
t = U[:, 2]
print("Translation Vector:", t)
```

Calculate $[t]_x$ from t. Use the formula $V'^T = V^T R \rightarrow R = VV'T$ to get the following $t, [t]_x, R$.

```
tX = np.full((3, 3), int(0), dtype = np.float64)
tX[0][0] = 0
tX[1][0] = t[2]
tX[2][0] = -t[1]
tX[0][1] = -t[2]
tX[1][1] = 0
tX[2][1] = t[0]
tX[0][2] = t[1]
tX[1][2] = -t[0]
tX[2][2] = 0

print("[t]x:\n", tX)
```

```
U, Sigma, VTt = np.linalg.svd(tX)
R = np.matmul(VTt, VT.T)
print("Rotation Matrix:\n", R)
print(np.linalg.det(R))
print(np.matmul(R, R.T))
```

```
Translation Vector: [-0.04944156 -0.0042811   0.99876784]
[t]x:
 [[ 0.         -0.99876784 -0.0042811 ]
 [ 0.99876784  0.          0.04944156]
 [ 0.0042811  -0.04944156  0.        ]]
Rotation Matrix:
 [[ 0.04840271 -0.8068855  -0.58872147]
 [ 0.00452134  0.58958328 -0.80769494]
 [ 0.99881767  0.03643282  0.03218564]]
1.0
[[1.00000000e+00 1.17913401e-16 3.39895325e-17]
 [1.17913401e-16 1.00000000e+00 5.94204107e-16]
 [3.39895325e-17 5.94204107e-16 1.00000000e+00]]
```

We now have the essential matrix, the translation vector, and the rotation matrix recovered, verifying their ranks and determinants are correct.

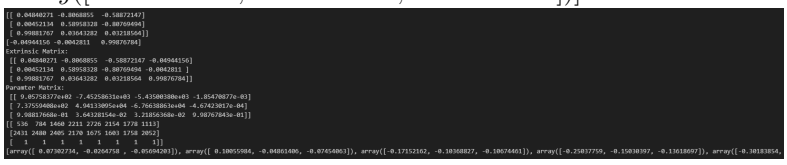
D

To calculate a new 3-D cloud using the intrinsic parameters I found earlier, I adjust the rotation matrix/translation vector to create a cloud that is not the same as my 8-point algorithm. It uses the image coordinates for 2D and its projection to determine a new set of 3D coordinates.

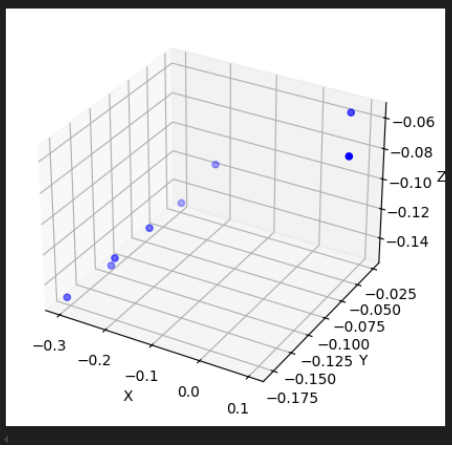
```
Xlist = []
Rnew = R
tnew = (t + np.array([0, 0, 0])).reshape(3, 1)
print(R)
print(t)
Rt = np.hstack((Rnew, tnew))
print("Extrinsic Matrix:\n",Rt)

P = np.matmul(K, Rt)
print("Paramter Matrix:\n",P)

imagepoints = np.vstack((img2u, img2v, np.ones_like(img2u)))
print(imagepoints)

for i in range(imagepoints.shape[1]):
    row1 = img2u[i] * P[2, :] - P[0, :]
    row2 = img2v[i] * P[2, :] - P[1, :]
```

```
    A = np.array([row1, row2])
    U, Sigma, VT = np.linalg.svd(A)
    X = VT[-1]
    X = X / X[-1]

    Xlist.append(X[:-1])

print(Xlist)
```

combining the rotation matrix/translation vector, forming the matrices A for each $X = x_{img}^2 P$, and using the null V^T from each A to obtain each 3D projection of each image coordinate, I obtain the 3D coordinates
$[array([0.07302734, -0.0264758, -0.05694203]),$
$array([0.10055984, -0.04861406, -0.07454063]),$
$array([-0.17152162, -0.10368827, -0.10674461]),$
$array([-0.25037759, -0.15030397, -0.13618697]),$
$array([-0.30183854, -0.17800952, -0.15026707]),$
$array([-0.24163788, -0.15049563, -0.13025173]),$
$array([-0.20228886, -0.12796365, -0.11613539]),$
$array([-0.13225196, -0.08127148, -0.08690061])].$



```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

Xlist2array = np.array(Xlist)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xlist2array[:, 0], Xlist2array[:, 1], Xlist2array[:, 2], c='b', marker='o')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```



Some of the that would have been closer to the front of the shoe are extended out and more distant from other points than they should be, but the general shape of the point cloud as it was shown on the shoe is preserved.