Computer Vision HW1: Camera Calibration

資科工碩 309551178 秦紫頤

Introduction

The goal of this homework is to use the concept taught in class to implement camera calibration. This homework should not use `cv2.calibrateCamera()`. Camera calibration is also referred to as camera resectioning, it estimates the parameters of a lens and image sensor of an image or video camera. You can use these parameters to correct lens distortion, measure the size of an object in world coordinates, or determine the location of the camera in the scene. These tasks are used in applications such as machine vision to detect and measure objects. They are also used in robotics, navigation system, and 3-D scene reconstruction. In this homework, I use camera calibration to determine the camera in the scene, I will show the results later on in this report. Also, I will compare my result with OpenCV and shown that my results are close to OpenCV's results.

Implementation Procedure

1. Find chessboard corner

This is already provided in the sample code. Passing in the chessboard image shot by the camera without autofocus, `cv2.findChessboardCorners()` will return corners (corner points on image plane). The object points is defined by (0,0), (0,1)...(7,7). Now we have object points and image points, I can start to calibrate the camera.

2. Find homography

In order to find the homography matrix which is a 3x3 matrix (9 unknown), you will need at least 4 correspondence points (object points, image points). I use all the corresponding points in the image. We will define a P matrix which is our correspondence point matrix like this:

$$\begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x_i' & y_i x_i' & x_i' \\ 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' & y_i \end{bmatrix}$$

(x_i, y_i) are the coordinates of the object points and ares the coordinates of the image points. I am solving the nullspace of the $PH = 0$ (H is the homography matrix, vector in here) which should look like the following:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 & y_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y'_2 & y_2y'_2 & y_2 \\ \vdots \\ -x_n & -y_n & -1 & 0 & 0 & 0 & x_nx'_n & y_nx'_n & x'_n \\ 0 & 0 & 0 & -x_n & -y_n & -1 & x_ny'_n & y_ny'_n & y_n \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h8 \end{bmatrix} = 0$$

We solve this by using svd (singular value decomposition) to get the eigenvectors in decreasing order of the eigenvalues. We get the last column of the eigenvector matrix as our Homography matrix because the eigenvector of the smallest eigenvalue is the solution of the nullspace. We reshape the eigenvector to a 3x3 homography matrix.

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

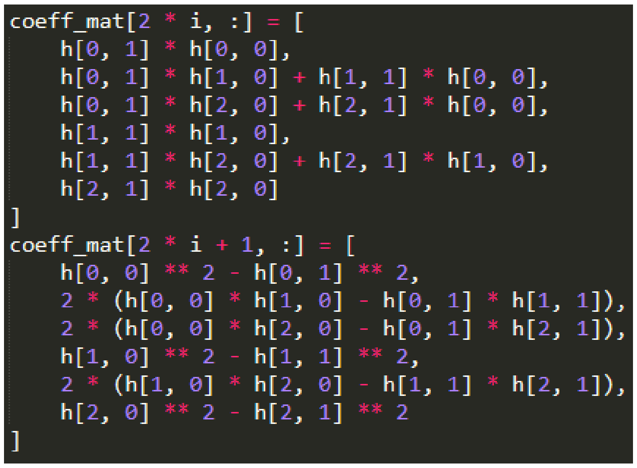
3. Find intrinsic parameters

I derive this step into multiple small steps. First, we define a B matrix as $K^{-T}K^{-1}$ and K is the intrinsic matrix. Also, B satisfied two equations:

$$h_1^T B h_2 = 0$$

$$h_1^T B h_1 = h_2^T B h_2$$

I organize the two equations into a coefficient matrix times B vector (6 parameters) and will equal to 0. The coefficient matrix is as follow:



```
coeff_mat[2 * i, :] = [
    h[0, 1] * h[0, 0],
    h[0, 1] * h[1, 0] + h[1, 1] * h[0, 0],
    h[0, 1] * h[2, 0] + h[2, 1] * h[0, 0],
    h[1, 1] * h[1, 0],
    h[1, 1] * h[2, 0] + h[2, 1] * h[1, 0],
    h[2, 1] * h[2, 0]
]
coeff_mat[2 * i + 1, :] = [
    h[0, 0] ** 2 - h[0, 1] ** 2,
    2 * (h[0, 0] * h[1, 0] - h[0, 1] * h[1, 1]),
    2 * (h[0, 0] * h[2, 0] - h[0, 1] * h[2, 1]),
    h[1, 0] ** 2 - h[1, 1] ** 2,
    2 * (h[1, 0] * h[2, 0] - h[1, 1] * h[2, 1]),
    h[2, 0] ** 2 - h[2, 1] ** 2
]
```

Same as getting the homography matrix, I get the B vector by svd. And I reshape it to a 3x3 matrix. It is a symmetric matrix so 6 parameters are enough. The matrix form of B looks like this:

$$\begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

If the diagonal of the B matrix is negative we just transform it to positive so that I can do Cholesky factorization to solve K which is the intrinsic matrix. After Cholesky factorization, I get K^{-1} so I simply inverse it to get my intrinsic matrix K. The last item in this matrix should be 1, so I time $\frac{1}{B_{33}}$ to all the term in the B matrix. This will not change the result, cause this is just a scaling factor.

4. Get extrinsic parameters

Now, I have the intrinsic matrix, I can get the extrinsic matrix for every image. First, I calculate the extrinsic matrix's scaling factor λ which is simply equal to $1/||K^{-1}h_1||$. Then we get the first column of the rotation matrix by $r_1 = \lambda K^{-1}h_1$ (h_1 is the first column of homography matrix), and the second column of the rotation matrix by $r_2 = \lambda K^{-1}h_2$. I get the third column of the rotation matrix by r_1 cross r_2. For camera calibration, the result is a rotation vector instead of a rotation matrix, so I transform the rotation matrix to a rotation vector by `cv2.Rodrigues()`. Lastly, I get the translation vector by $t = \lambda K^{-1}h_3$. Then for each rotation vector and translation vector, I use a list to store like this $[r_1, r_2, r_3, t_1, t_2, t_3]$.

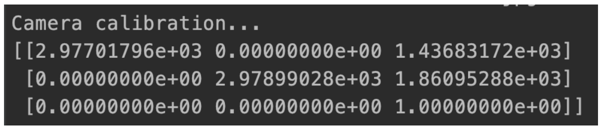
5. Post-processing

I drew out the location of the camera in the scene by passing the extrinsic parameters of all the images to the codes provided by TA. This completes the homework.

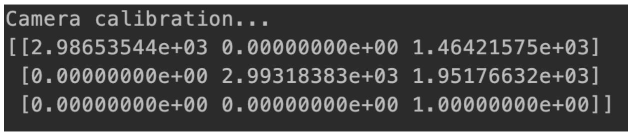
Experimental Results

I will show the comparison results of my implementation and `cv2.calibrateCamera()`. There are two datasets, TA's dataset and my dataset (image shot by my phone). All the results shown below will show the cv2's result on the left-hand side and my implementation results on the right-hand side.

TA's dataset (intrinsic matrix)

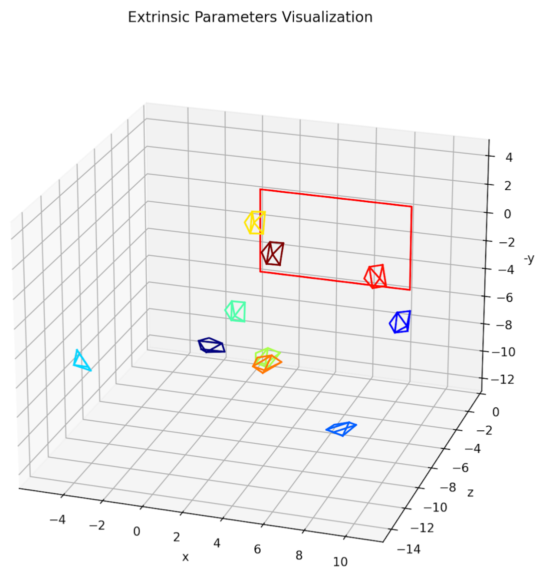


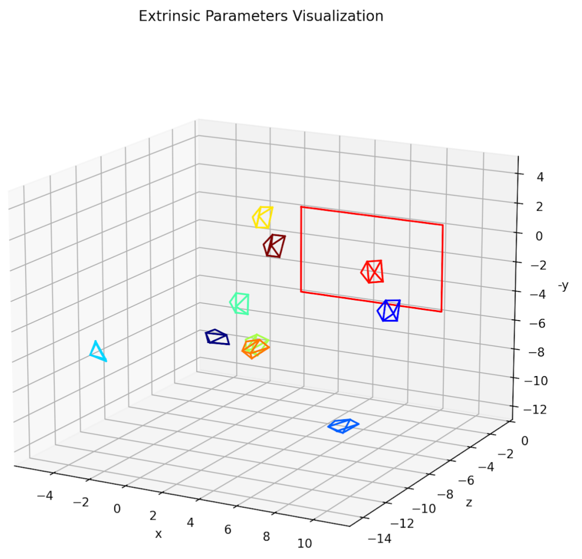
```
Camera calibration...
[[2.97701796e+03 0.00000000e+00 1.43683172e+03]
 [0.00000000e+00 2.97899028e+03 1.86095288e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```



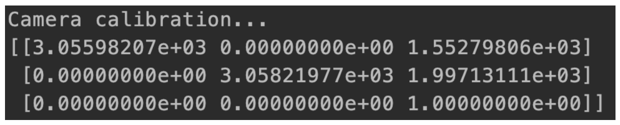
```
Camera calibration...
[[2.98653544e+03 0.00000000e+00 1.46421575e+03]
 [0.00000000e+00 2.99318383e+03 1.95176632e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

TA’s dataset (camera location in the 3D world)

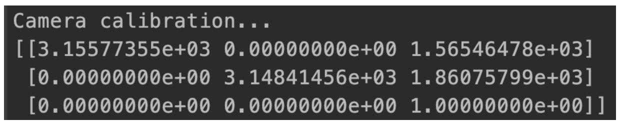




My dataset (intrinsic matrix)

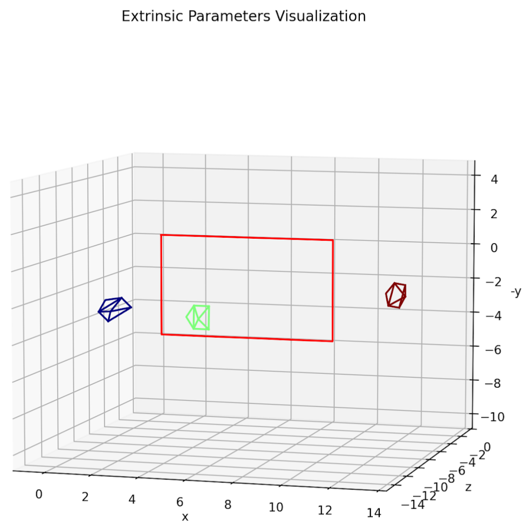


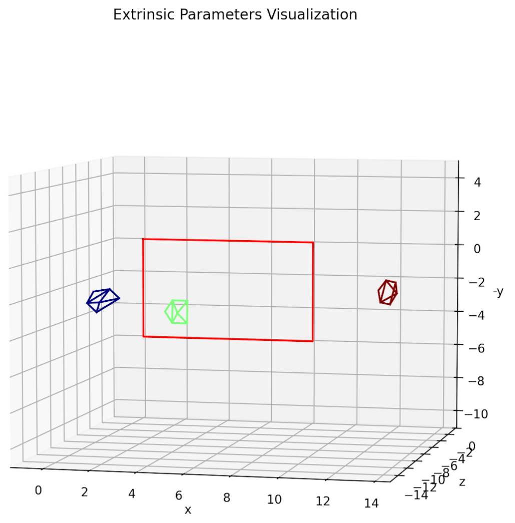
```
Camera calibration...
[[3.05598207e+03 0.00000000e+00 1.55279806e+03]
 [0.00000000e+00 3.05821977e+03 1.99713111e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```



```
Camera calibration...
[[3.15577355e+03 0.00000000e+00 1.56546478e+03]
 [0.00000000e+00 3.14841456e+03 1.86075799e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

My dataset (camera location in the 3D world)





Discussion

When finding intrinsic matrix, I can’t make `K[1][1]` this term to 0, instead, it is a small negative term. I don’t know how `cv2.calibrateCamera()` gets 0 in this term, and I couldn’t find any answer online, so I just forced this term to 0 after getting K. I think this won’t change the result since it is just a small negative number.

When using my dataset, I shot 6 images but from both cv2’s result and my implementation result, I can only get 3 camera locations on the 3D coordinate. I was wonder maybe for the six images there are only three kinds of locations that I shot the image.

Conclusion

In this homework, I have fallen into some trap when I tried to get the intrinsic matrix at first. I couldn’t get a close solution to the cv2’s solution. But after some debugging, I found out that I mix row and column in the homography matrix, which is kind of a stupid mistake. I think this homework isn’t really difficult, but it definitely needs to be more careful. Also, in lecture I can’t relate the math to the actual camera calibration, but after implementing myself I kind of can and be more understanding about this unit.

Work assignment plan