Computer Vision HW1 Group 16

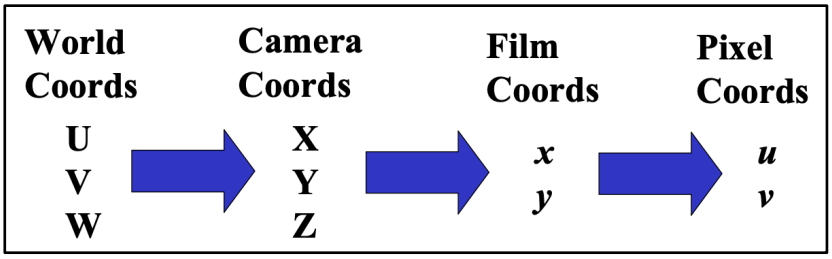
0856622 余家宏、309551067 吳子涵、309551122 曹芳驊

Introduction

Camera calibration is the process of estimating the intrinsic and extrinsic matrix parameters of a camera. Intrinsic parameters deal with the internal characteristics of cameras, such as focal length, skew, and image center. Extrinsic parameters describe the transformation between the camera coordinate and the real-world coordinate, including rotation and translation.

The purpose of this assignment is to implement a function to calibrate a camera. The input of the function should be chessboard images taken from different camera poses with the same camera. After extracting the chessboard corners and setting up the coordinate origin, it should be able to estimate the intrinsic parameters of the camera and the extrinsic parameters of every camera pose. Finally, we can use the intrinsic matrix and the extrinsic matrixes to plot them as 3D results.

Implementation Procedure



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f/s_x & s_k & o_x & 0 \\ 0 & f/s_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} U \\ V \\ W \\ 1 \end{pmatrix}$$

- Step1: calculate homography matrix H
  - A homography matrix is used to convert points from one coordinate space to another.
  - We treat K[R t] as the homography matrix H, and get the matrix multiplication as follows.

$$\mathbf{x} = \underbrace{\mathbf{K}}_{\text{intrinsic matrix}}\underbrace{\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}}_{\text{extrinsic matrix}}\mathbf{X} \qquad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

  - From the above equations, we can get the relation between (u, v) and (X, Y)

$$u = \frac{h_{00}.X + h_{01}.Y + h_{02}}{h_{20}.X + h_{21}.Y + h_{22}}$$

$$v = \frac{h_{10}.X + h_{11}.Y + h_{12}}{h_{20}.X + h_{21}.Y + h_{22}}$$

$$u.(h_{20}.X + h_{21}.Y + h_{22}) - (h_{00}.X + h_{01}.Y + h_{02}) = 0$$

$$v.(h_{20}.X + h_{21}.Y + h_{22}) - (h_{10}.X + h_{11}.Y + h_{12}) = 0$$

  - We can remodel the above equations in a simpler way.

$$\begin{pmatrix} -X & -Y & -1 & 0 & 0 & 0 & u.X & u.Y & u \\ 0 & 0 & 0 & -X & -Y & -1 & v.X & v.Y & v \end{pmatrix}\begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{pmatrix} = 0$$

  - The above equation is for one point in one image. For N points in an image, we can extend the matrix by vertically stacking.

$$\begin{pmatrix} -X_0 & -Y_0 & -1 & 0 & 0 & 0 & u_0.X_0 & u_0.Y_0 & u_0 \\ 0 & 0 & 0 & -X_0 & -Y_0 & -1 & v_0.X_0 & v_0.Y_0 & v_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -X_{N-1} & -Y_{N-1} & -1 & 0 & 0 & 0 & u_{N-1}.X_{N-1} & u_{N-1}.Y_{N-1} & u_{N-1} \\ 0 & 0 & 0 & -X_{N-1} & -Y_{N-1} & -1 & v_{N-1}.X_{N-1} & v_{N-1}.Y_{N-1} & v_{N-1} \end{pmatrix}_{(2\times N,9)} .\vec{h} = 0$$

- Since we know $(X_0, Y_0) \cdots (X_{N-1}, Y_{N-1})$, (u_0, v_0), (u_{N-1}, v_{N-1}), we can use Singular Value Decomposition (SVD) method to find the h vector and reshape h to H.
- The following code shows how we compute the homography matrix H.

```
def homography(objpoints, imgpoints):
    H = np.zeros((len(objpoints), 9))

    imgpoints = np.squeeze(imgpoints)
    img_num = len(objpoints)
    for img_idx in range(img_num):
        pt_3d = objpoints[img_idx]
        pt_2d = imgpoints[img_idx]
        P = np.empty([0, 9])

        for pt_idx in range(len(pt_3d)):
            x = pt_3d[pt_idx, 0]
            y = pt_3d[pt_idx, 1]
            u = pt_2d[pt_idx, 0]
            v = pt_2d[pt_idx, 1]

            arr1 = np.array([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
            arr2 = np.array([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
            P = np.concatenate((P, arr1[np.newaxis,:], arr2[np.newaxis,:],), axis=0)

        U, S, Vt = np.linalg.svd(P)

        H[img_idx] = Vt[-1]
        H[img_idx] /= H[img_idx][-1]    # normalize

    H = np.reshape(H, (img_num, 3, 3))
    return H
```

- Step2: calculate intrinsic matrix K through B
  - After getting H, we need to decompose it into the intrinsic matrix K and the extrinsic matrix [R t].

$$\boldsymbol{H} = (\mathbf{h_1}, \mathbf{h_2}, \mathbf{h_3}) = \underbrace{\begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{(\mathbf{r_1}, \mathbf{r_2}, \mathbf{t})}$$

$$\mathbf{r_1} = \mathbf{K}^{-1}\mathbf{h_1} \text{ and } \mathbf{r_2} = \mathbf{K}^{-1}\mathbf{h_2}$$

- Due to the constraints of the rotation matrix(orthonormal), we can have
  $\mathbf{r_1}^T\mathbf{r_2} = \mathbf{0}$, $||\mathbf{r_1}||=||\mathbf{r_2}||=1$

- So that we have these two equations

$$\mathbf{h}_1^\top \mathbf{K}^{-\top}\mathbf{K}^{-1}\mathbf{h}_2 = 0$$
$$\mathbf{h}_1^\top \mathbf{K}^{-\top}\mathbf{K}^{-1}\mathbf{h}_1 = \mathbf{h}_2^\top \mathbf{K}^{-\top}\mathbf{K}^{-1}\mathbf{h}_2$$

- Let $B = K^{-T}K^{-1}$, so B is symmetric and positive definite.

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

- Because the above two constraints are with the same form (i.e. $h_i^TBh_j$), we try to compute $h_i^TBh_j$ first:

$$h_i^T B h_j = \begin{bmatrix} h_{i1} & h_{i2} & h_{i3} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{bmatrix} \begin{bmatrix} h_{j1} \\ h_{j2} \\ h_{j3} \end{bmatrix}$$

- Unfold the matrix multiplication, and then integrate the equation as follows.

  (B is a symmetric matrix, so we only have 6 elements actually.)

$$\begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}^T \begin{bmatrix} b_{11} \\ b_{12} \\ b_{22} \\ b_{13} \\ b_{23} \\ b_{33} \end{bmatrix}^T = V_{ij}^T b$$

V_{ij}

- We have a new matrix v_{ij} there, to represent $h_i^T B h_j$ with a new form $v_{ij}^T b$, and then apply thi s form back to the two constraints.

$$\begin{cases} h_1^T K^{-T} K^{-1} h_2 = 0 \\ h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \end{cases} \Rightarrow \begin{cases} \mathbf{v}_{12}^T b = 0 \\ \mathbf{v}_{11} b = \mathbf{v}_{22} b \end{cases}$$

- Reforming these two equations into matrix multiplication.

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = \mathbf{V}\mathbf{b} = 0$$

- Matrix V is composed of the elements of the homography matrix H, so V is known. We ca n use SVD method to get the vector b, and then we can rearrange vector b to get the matri x B.
- Because $B = K^{-T} K^{-1}$, we can use Cholesky factorization to get K.
- Cholesky factorization

$$\begin{aligned} \mathbf{A} = \mathbf{L}\mathbf{L}^{\mathbf{T}} &= \begin{pmatrix} \mathbf{L}_{11} & 0 & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} & 0 \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{L}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{21} & \mathbf{L}_{31} \\ 0 & \mathbf{L}_{22} & \mathbf{L}_{32} \\ 0 & 0 & \mathbf{L}_{33} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}_{11}^2 & & \text{(symmetric)} \\ \mathbf{L}_{21}\mathbf{L}_{11} & \mathbf{L}_{21}^2 + \mathbf{L}_{22}^2 & \\ \mathbf{L}_{31}\mathbf{L}_{11} & \mathbf{L}_{31}\mathbf{L}_{21} + \mathbf{L}_{32}\mathbf{L}_{22} & \mathbf{L}_{31}^2 + \mathbf{L}_{32}^2 + \mathbf{L}_{33}^2 \end{pmatrix} \end{aligned}$$

  - we can get the decomposed matrix L as follows:

$$\begin{aligned} \mathbf{L}_{j,j} &= \sqrt{A_{j,j} - \sum_{k=1}^{j-1} \mathbf{L}_{j,k}^2} \\ \mathbf{L}_{i,j} &= \frac{1}{\mathbf{L}_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} \mathbf{L}_{i,k} \mathbf{L}_{j,k} \right), \qquad \text{for } i > j \end{aligned}$$

- Because the element K_{33} of the intrinsic matrix K should be 1, we need to normalize B before applying Cholesky factorization.

- Normalization term.

$$L_{11} = \sqrt{B_{11} - \sum_{k=1}^{0} L_{1k}^2} = \sqrt{B_{11}}$$

$$L_{21} = \frac{1}{L_{11}}\left(B_{21} - \sum_{k=1}^{0} L_{2k}L_{1k}\right) = \frac{1}{L_{11}}B_{21} = \frac{B_{21}}{\sqrt{B_{11}}}$$

$$L_{31} = \frac{1}{L_{11}}\left(B_{31} - \sum_{k=1}^{0} L_{3k}L_{1k}\right) = \frac{1}{L_{11}}B_{31} = \frac{B_{31}}{\sqrt{B_{11}}}$$

$$L_{22} = \sqrt{B_{22} - \sum_{k=1}^{1} L_{2k}^2} = \sqrt{B_{22} - L_{21}^2} = \sqrt{B_{22} - \frac{B_{21}^2}{B_{11}}}$$

$$L_{32} = \frac{1}{L_{22}}\left(B_{32} - \sum_{k=1}^{1} L_{3k}L_{2k}\right) = \frac{1}{\sqrt{B_{22} - \frac{B_{21}^2}{B_{11}}}}\left(B_{32} - L_{31}L_{21}\right)$$

$$= \frac{1}{\sqrt{B_{22} - \frac{B_{21}^2}{B_{11}}}}\left(B_{32} - \frac{B_{31}}{\sqrt{B_{11}}}\frac{B_{21}}{\sqrt{B_{11}}}\right) = \frac{1}{\sqrt{B_{22} - \frac{B_{21}^2}{B_{11}}}}\left(B_{32} - \frac{B_{31}B_{21}}{B_{11}}\right)$$

$$L_{33} = \sqrt{B_{33} - \sum_{k=1}^{2} L_{3k}^2} = \sqrt{B_{33} - (L_{31}^2 + L_{32}^2)} = \sqrt{B_{33} - \left(\frac{B_{31}^2}{B_{11}} + \frac{1}{B_{22} - \frac{B_{21}^2}{B_{11}}}\left(B_{32} - \frac{B_{31}B_{21}}{B_{11}}\right)^2\right)}$$

↳ We hope that this term should be 1

Let $\lambda = \sqrt{B_{33} - \left(\frac{B_{31}^2}{B_{11}} + \frac{1}{B_{22} - \frac{B_{21}^2}{B_{11}}}\left(B_{32} - \frac{B_{31}B_{21}}{B_{11}}\right)^2\right)}$

Thus, $B \to \frac{1}{\lambda}B$, then we get $L_{33} = 1$ after applying Cholesky factorization.

∵ B is symmetric : $B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & b_3 \\ b_1 & b_2 & b_4 \\ b_3 & b_4 & b_5 \end{bmatrix}$

$$\therefore \lambda = B_{33} - \left(\frac{B_{31}^2}{B_{11}} + \frac{1}{B_{22} - \frac{B_{21}^2}{B_{11}}}\left(B_{32} - \frac{B_{31}B_{21}}{B_{11}}\right)^2\right)$$

$$= b_5 - \left(\frac{b_3^2}{b_0} + \frac{1}{b_2 - \frac{b_1^2}{b_0}}\left(b_4 - \frac{b_3 b_1}{b_0}\right)^2\right)$$

$$= b_5 - \left(\frac{b_3^2}{b_0} + \frac{1}{b_0 b_2 - b_1^2}\left(b_0 b_4 - b_1 b_3\right)^2 \cdot \frac{1}{b_0}\right)$$

$$= b_5 - \left(b_3^2 + \frac{1}{b_0 b_2 - b_1^2}\left(b_0 b_4 - b_1 b_3\right)^2\right) \cdot \frac{1}{b_0}$$

- The code of getting intrinsic matrix is as follows.

```
def get_V(h, i, j):
    return np.array([ h[0,i] * h[0,j],
                      h[0,i] * h[1,j] + h[1,i] * h[0,j],
                      h[1,i] * h[1,j],
                      h[2,i] * h[0,j] + h[0,i] * h[2,j],
                      h[2,i] * h[1,j] + h[1,i] * h[2,j],
                      h[2,i] * h[2,j]])
```

```python
def intrinsic(H):

    V = np.zeros([2 * len(H), 6])
    idx = 0
    for _h in H:
        V[idx] = get_V(_h, 0, 1)
        V[idx + 1] = get_V(_h, 0, 0) - get_V(_h, 1, 1)
        idx += 2

    U, S, Vt = np.linalg.svd(V)
    b = Vt[-1]
    B = np.array([[b[0], b[1], b[3]],
                  [b[1], b[2], b[4]],
                  [b[3], b[4], b[5]]])

    lambda_B = b[5] - ( b[3]*b[3] + pow((b[0]*b[4] - b[1]*b[3]), 2) / (b[0]*b[2]-b[1]*b[1])) / b[0]
    B = B / lambda_B    # normalize

    K = np.linalg.inv(np.linalg.cholesky(B).T)
    K_inv = np.linalg.inv(K)

    return K
```

- Step3: calculate extrinsic matrix [R t]
    - After getting H and K, we can simply compute the R and t from the homography constrain ts. $\mathbf{r_1}=\mathbf{K}^{-1}\mathbf{h_1}$ **and** $\mathbf{r_2}=\mathbf{K}^{-1}\mathbf{h_2}$

$$
\begin{aligned}
\mathbf{r}_1 &= \lambda \mathbf{K}^{-1}\mathbf{h}_1 \\
\mathbf{r}_2 &= \lambda \mathbf{K}^{-1}\mathbf{h}_2 \\
\mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2 \\
\mathbf{t} &= \lambda \mathbf{K}^{-1}\mathbf{h}_3 \\
\lambda &= 1/||\mathbf{K}^{-1}\mathbf{h}_1||
\end{aligned}
$$

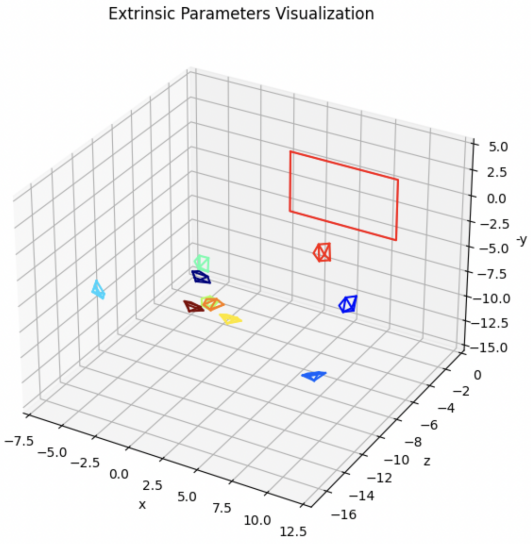
    - Thus, we get the extrinsic matrix for each image.

```python
def extrisic(H, K):

    extrinsics = np.empty([0, 3, 4])
    K_inv = np.linalg.inv(K)
    for img_idx in range(len(H)):
        h1 = H[img_idx][:, 0]
        h2 = H[img_idx][:, 1]
        h3 = H[img_idx][:, 2]

        lambda1 = 1 / np.linalg.norm(np.dot(K_inv, h1))
        lambda2 = 1 / np.linalg.norm(np.dot(K_inv, h2))
        lambda3 = (lambda1 + lambda2) / 2

        r1 = lambda1 * np.dot(K_inv, h1)
        r2 = lambda2 * np.dot(K_inv, h2)
        r3 = np.cross(r1, r2)
        t = lambda3 * np.dot(K_inv, h3)

        Rt = np.array([r1.T, r2.T, r3.T, t.T]).T
        extrinsics = np.concatenate((extrinsics, Rt[np.newaxis,:]), axis=0)

    return extrinsics
```

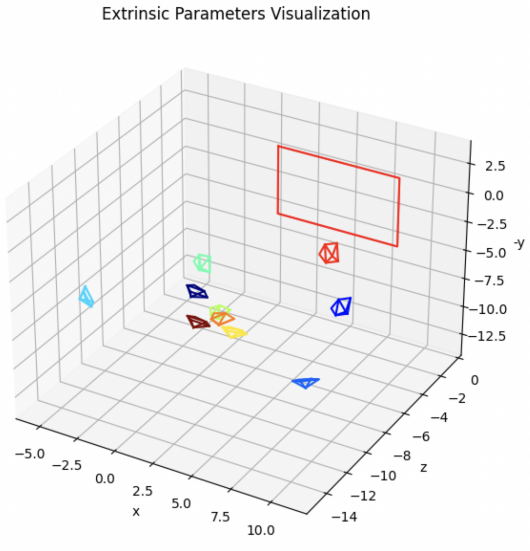
Experimental result

We first run the camera calibration function written by us with the images provided by TA and compare t he result with the Python Opencv build-in calibration function. We noticed that there are only a few offsets i n absolute coordinates, but the relative position of camera poses are roughly the same.

Dataset provided by TA



our calibration result



built-in calibration result

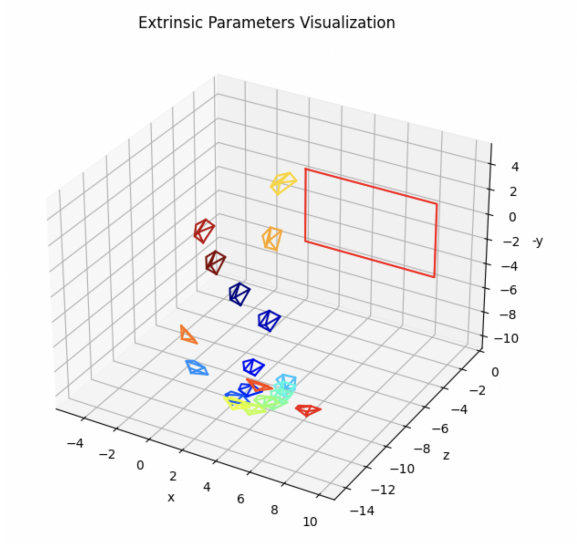
- intrinsic matrix comparison

```
our intrisic matrix:
 [[ 3.40033701e+03 -3.52543658e+01  1.47568441e+03]
 [ 0.00000000e+00  3.34935946e+03  1.40822869e+03]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
built-in intrisic:
 [[3.17677579e+03 0.00000000e+00 1.64148698e+03]
 [0.00000000e+00 3.19707029e+03 1.43116619e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

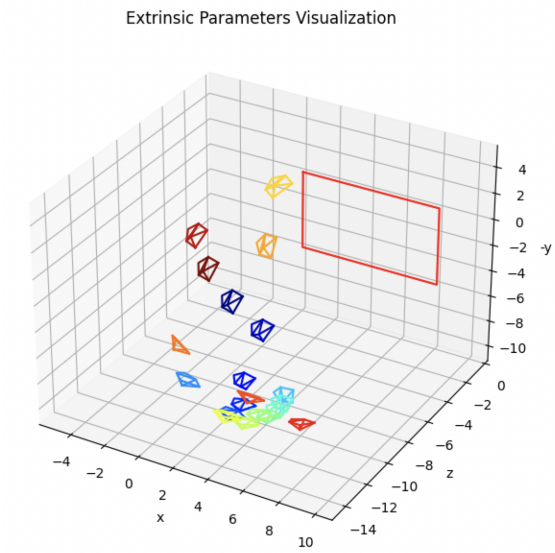
(In OpenCV the camera intrinsic matrix does not have the skew parameter.)

Dataset provided by ourselves

We also run the camera calibration function with our own images.



our calibration result



built-in calibration result

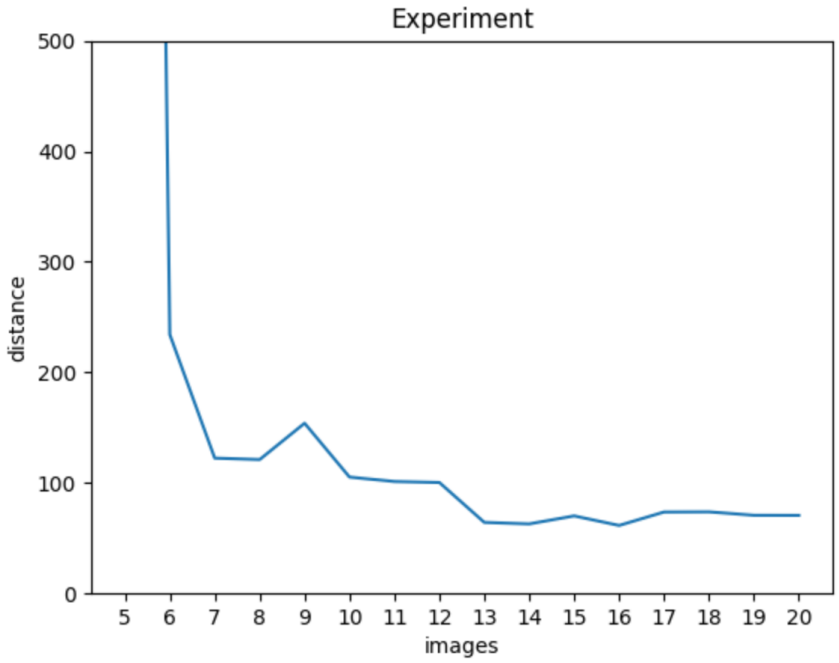
- intrinsic matrix comparison

```
our intrisic matrix:
 [[3.49257296e+03 1.67555870e+01 1.55443504e+03]
 [0.00000000e+00 3.48744880e+03 2.02684661e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
built-in intrisic:
 [[3.50250672e+03 0.00000000e+00 1.49723745e+03]
 [0.00000000e+00 3.50258207e+03 1.99346561e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Discussion

In our experiment, we noticed that the more images we use to calibrate the camera, the more accurate the result we will obtain.

We used different numbers of images to calibrate the camera and analyzed the intrinsic matrix we got in different situations. In our assumption, we treated the intrinsic matrix computed by using the 20 images wit h the built-in function as the ground truth and calculated the distance between the ground truth matrix and t he intrinsic matrices we got using our calibration function with different numbers of images.



From the chart above, we can simply discover that the intrinsic matrix is getting similar to the ground tru th intrinsic matrix while the number of images is increasing, which is in line with our intuition.

Conclusion

In this project, we successfully implement a camera calibration function in python. Though the result is s lightly different from the build-in calibration function provided by Opencv, it still shows the rough relative position between each camera poses correctly. In our experiment, we noticed that by taking more chessboar d images from different poses, we will obtain more accurate results of calibration.