CVHW1

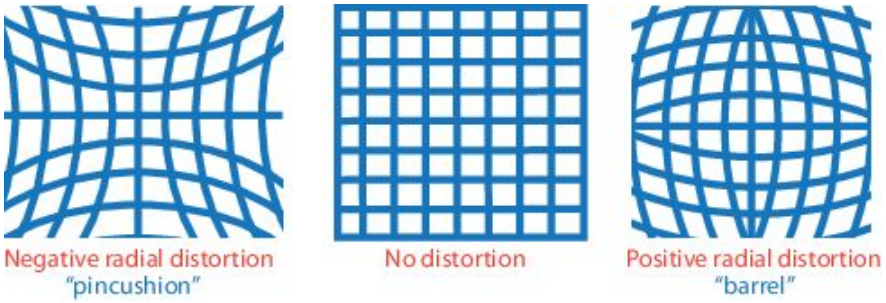
team members:0510134 王泓仁、0516006 賴奕善、0516218 軒轅照雯

1. Introduction

What Is Camera Calibration?

Geometric camera calibration, also referred to as camera resectioning, estimates the parameters of a lens and image sensor of an image or video camera.

You can use these parameters to correct for lens distortion, measure the size of an object in world units, 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, for navigation systems, and 3-D scene reconstruction.

Camera Calibration Parameters

The calibration algorithm calculates the camera matrix using the extrinsic and intrinsic parameters. The **extrinsic parameters** represent a rigid transformation from a 3D world coordinate system to the 3D camera's coordinate system. The **intrinsic parameters** represent a projective transformation from the 3D camera's coordinates into the 2D image coordinates.

2D Calibration Chessboard

Since the chessboard is a 2D plane in the 3D world, the corner points on the chessboard are on the same Z coordinate. We can cancel out Z, which is W, and the third column of the intrinsic matrix and refer to the multiplication of extrinsic matrix and intrinsic matrix as homography matrix H. H is a 3x3 matrix with 3 columns(h1, h2, h3). **In conclusion, the transformation of the two planes in 3D and 2D coordinate systems can be represented by**

homography matrix H. The process is shown below.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

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

(1) Find homography H

We can find matrix H by using a similar process of solving Pm=0 with SVD in 3D calibration rig :

$$p_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = MP_i = \begin{bmatrix} m_1 P_i \\ m_3 P_i \\ m_2 P_i \\ m_3 P_i \end{bmatrix}$$

known from calibration rig

(m_1, m_2, m_3 are rows of M)

$$u_i(m_3 P_i) - m_1 P_i = 0$$
$$v_i(m_3 P_i) - m_2 P_i = 0$$

each correspondence gives 2 equations, while M has 11 DoF

$$\begin{bmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ & \vdots & \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{bmatrix} \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} = \mathbf{P}m = 0$$

$$\underset{m}{\text{minimize}} \quad \|\mathbf{P}m\|^2$$
$$\text{subject to} \quad \|m\|^2 = 1$$

to avoid trivial solution $m=0$ $\forall k \in \mathbb{R}$, km is also a solution

can be solved by singular value decomposition (SVD)

$$P = UDV^T$$

set m equal to the last column of V

(2) Find intrinsic matrix K

After we find H, we can find the intrinsic matrix K. Since the rotation matrix R is an orthonormal matrix, we can get the following equation:

$$\mathbf{r_1}^\top \mathbf{r_2} = 0, \; ||\mathbf{r_1}|| = ||\mathbf{r_2}|| = 1$$

$$\rightarrow \mathbf{h_1}^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h_2} = 0$$
$$\mathbf{h_1}^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h_1} = \mathbf{h_2}^\top \mathbf{K}^{-\top} \mathbf{K}^{-1} \mathbf{h_2}$$

To solve K, we let **B =** $K^{-T}K^{-1}$

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

since B is symmetric and positive definite, we only need to find the values of the upper triangle or lower triangle in B. Define b=(b_{11}, b_{12}, b_{13}, b_{22}, b_{23}, b_{33}) leads to the system of Vb=0. Because real measurements might be corrupted with noise, we also use SVD to find b that minimize Vb.

Due to the same camera we use, no matter how the chessboard is transposed, the K is unique, that is, the intrinsic matrix stays the same. Each plane gives 2 equations and there are 6 degrees of freedom in B, we need at least 3 planes from different views.

After solving B, we can use Cholesky factorization to find K.

(3) Find extrinsic matrix

After we find H and K, we can calculate the extrinsic matrix as below:

$$H = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix} = \rho \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix}$$

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

2. Implementation procedure

(1) Create homogeneous coordinates

```
world_coor = np.copy(objpoints)
camera_coor = np.zeros((img_num, p_per_img, 3))
#homo coor
for i in range(img_num):
    for j in range(p_per_img):
        world_coor[i][j][2] = 1
        camera_coor[i][j][0] = imgpoints[i][j][0][0]
        camera_coor[i][j][1] = imgpoints[i][j][0][1]
        camera_coor[i][j][2] = 1
```

(2) Find the homography matrix H by solving Pm=0 with SVD

- let m be a matrix of size 12x1 which contains the transpose of rows in H
- let Pm=0 be the matrix representation of the equations

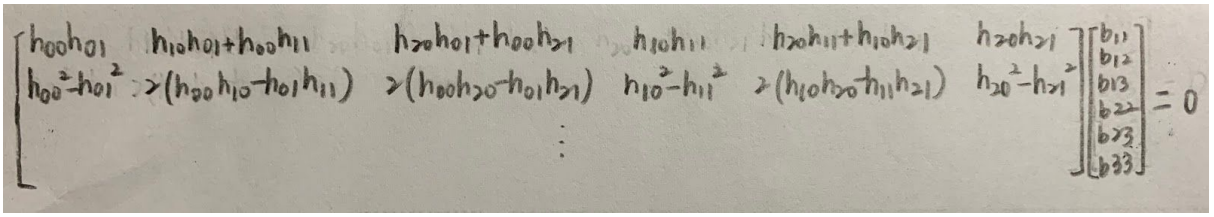
$$\begin{array}{l} u_1(m_3 P_1) - m_1 P_1 = 0 \\ v_1(m_3 P_1) - m_2 P_1 = 0 \\ \vdots \\ u_n(m_3 P_n) - m_1 P_n = 0 \\ v_n(m_3 P_n) - m_2 P_n = 0 \end{array} \quad \begin{bmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ & \vdots & \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{bmatrix} \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} = \mathbf{P}m = 0$$

```
#Pm=0 --> find H
H = []
for i in range(img_num):
    P = []
    for j in range(p_per_img):
        u, v = camera_coor[i][j][0], camera_coor[i][j][1]
        x, y = world_coor[i][j][0], world_coor[i][j][1]
        P.append([-x, -y, -1, 0, 0, 0, u*x, u*y, u])
        P.append([0, 0, 0, -x, -y, -1, v*x, v*y, v])
    P = np.asarray(P)
    u, s, vh = np.linalg.svd(P)
    m = vh[-1,:] / vh[-1,-1]
    H.append(m.reshape(3, 3))
```

(3) Find B by solving Vb=0 with SVD and then find the inverse of the intrinsic matrix K by doing Cholesky factorization on B

- let B = $K^{-T}K^{-1}$
- let b be a matrix of size 6x1 which contains the distinct elements in B
- let Vb=0 be the matrix representation of the equations

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




```
#Vb=0 --> find K
V=np.zeros((2*img_num ,6))
for i in range(img_num):
    V[2*i][0]=H[i][0][0]*H[i][0][1]
    V[2*i][1]=H[i][1][0]*H[i][0][1]+H[i][0][0]*H[i][1][1]
    V[2*i][2]=H[i][2][0]*H[i][0][1]+H[i][0][0]*H[i][2][1]
    V[2*i][3]=H[i][1][0]*H[i][1][1]
    V[2*i][4]=H[i][2][0]*H[i][1][1]+H[i][1][0]*H[i][2][1]
    V[2*i][5]=H[i][2][0]*H[i][2][1]
    V[2*i+1][0]=H[i][0][0]*H[i][0][0]-H[i][0][1]*H[i][0][1]
    V[2*i+1][1]=2*(H[i][0][0]*H[i][1][0]-H[i][0][1]*H[i][1][1])
    V[2*i+1][2]=2*(H[i][0][0]*H[i][2][0]-H[i][0][1]*H[i][2][1])
    V[2*i+1][3]=H[i][1][0]*H[i][1][0]-H[i][1][1]*H[i][1][1]
    V[2*i+1][4]=2*(H[i][1][0]*H[i][2][0]-H[i][1][1]*H[i][2][1])
    V[2*i+1][5]=H[i][2][0]*H[i][2][0]-H[i][2][1]*H[i][2][1]

u, s, vt = np.linalg.svd(V, full_matrices=True)
b=vt[-1,:] / vt[-1,-1]
B=np.zeros((3, 3))
B[0][0]=b[0]
B[0][1]=b[1]
B[0][2]=b[2]
B[1][1]=b[3]
B[1][2]=b[4]
B[2][2]=b[5]
B[1][0]=B[0][1]
B[2][0]=B[0][2]
B[2][1]=B[1][2]

#K
K=np.linalg.cholesky(B).T #actually it's K^(-1)
```

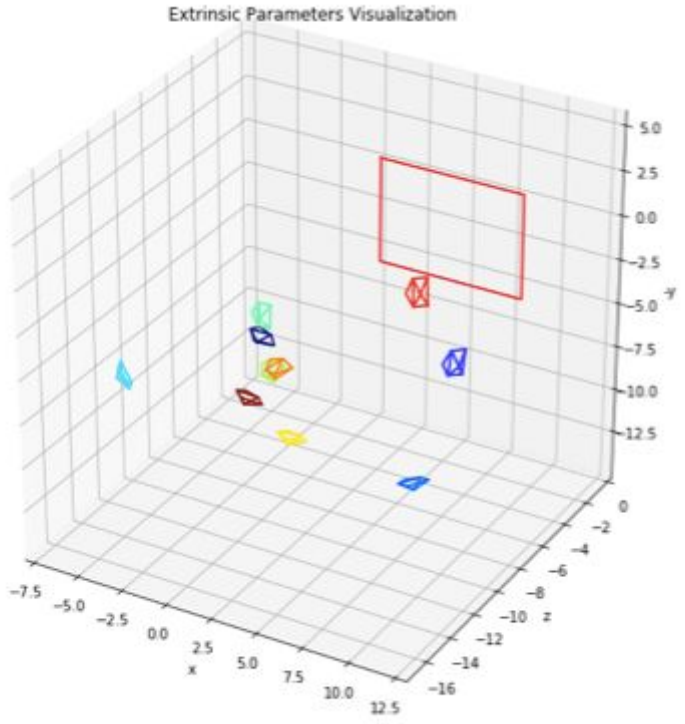
(4) Calculate the extrinsic matrix from the intrinsic matrix K

```
#extrinsic
extrinsics_S=np.zeros((img_num, 3, 4))
for x in range(img_num):
    l=1/(np.sum(np.power(np.matmul(K, H[x][:, 0]), 2))**0.5)
    extrinsics_S[x][:, 0]=l*np.matmul(K, H[x][:, 0])
    extrinsics_S[x][:, 1]=l*np.matmul(K, H[x][:, 1])
    extrinsics_S[x][:, 2]=np.cross(extrinsics_S[x][:, 0], extrinsics_S[x][:, 1])
    extrinsics_S[x][:, 3]=l*np.matmul(K, H[x][:, 2])
```

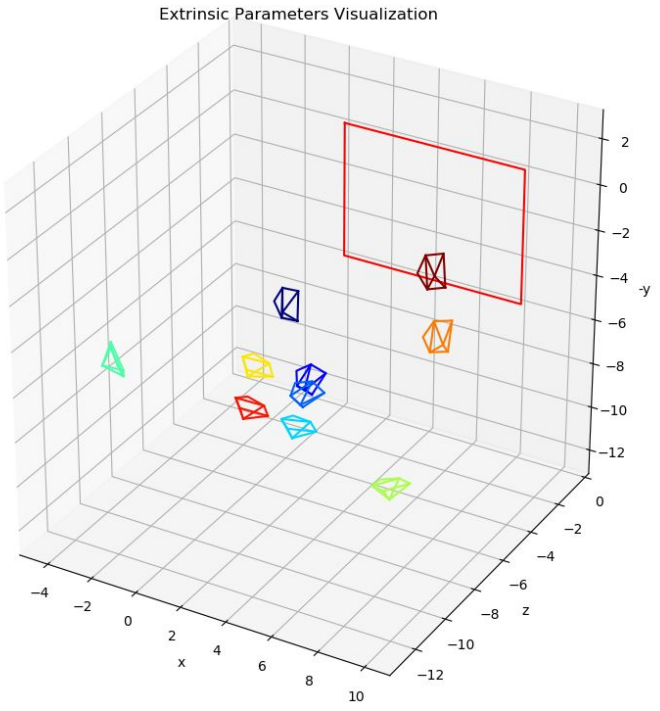
3. Experimental result

(1) Provided data

- using our own calibration function:

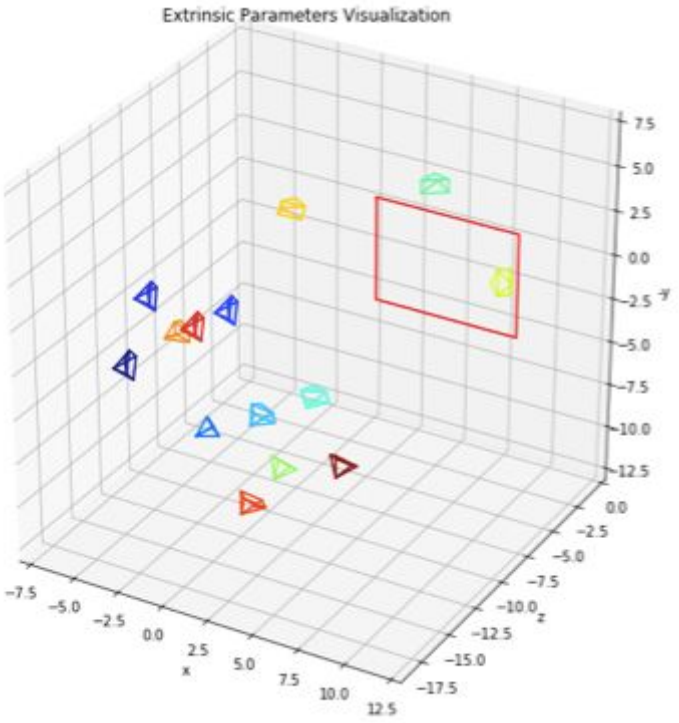


- using cv2.calibrateCamera:

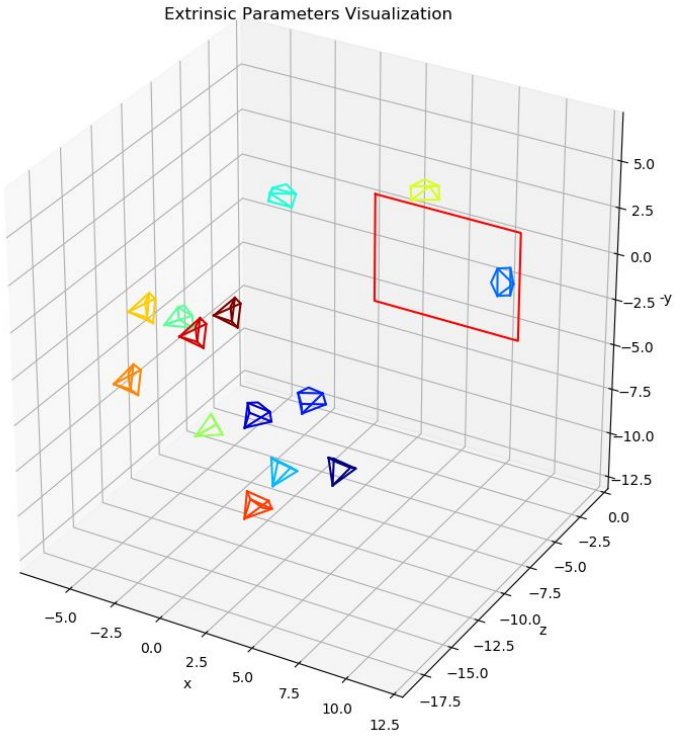


(2) data2 (unfixed camera position and image positions)

- using our own calibration function:

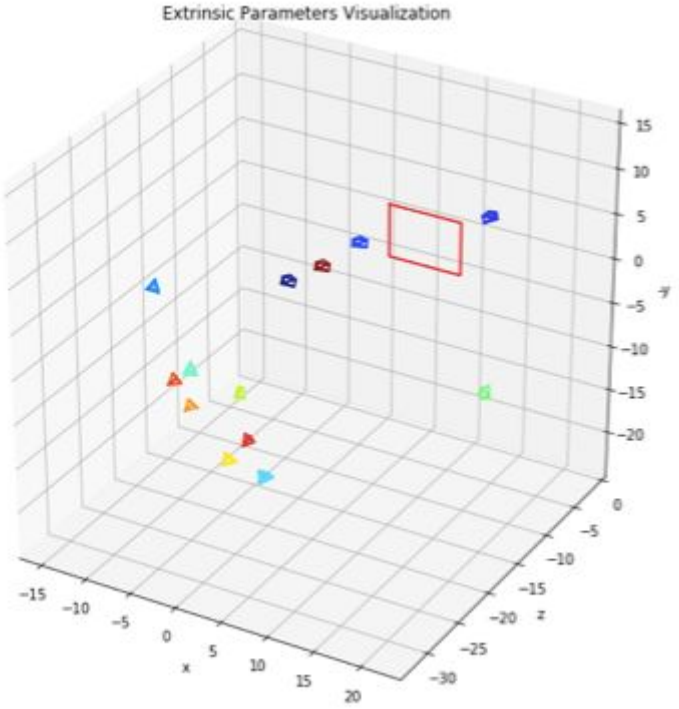


- using cv2.calibrateCamera:

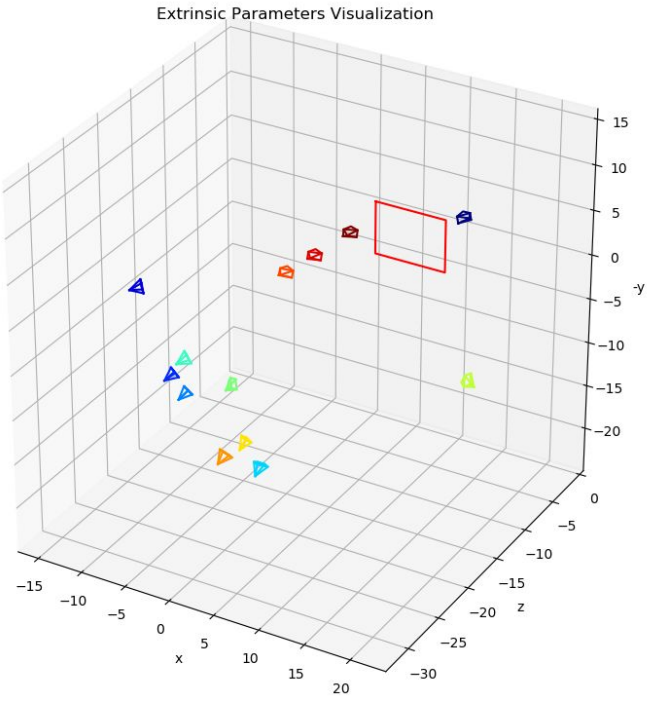


(3) data3 (fixed camera position and unfixed image positions)

- using our own calibration function:



- using cv2.calibrateCamera:



4. Discussion

(1) Correctness

From experiments 1 to 3, It is clear that the values of extrinsic parameters obtained by our own calibration function nearly equal to the output values of the cv2 inbuilt function, which proves the correctness of our own function.

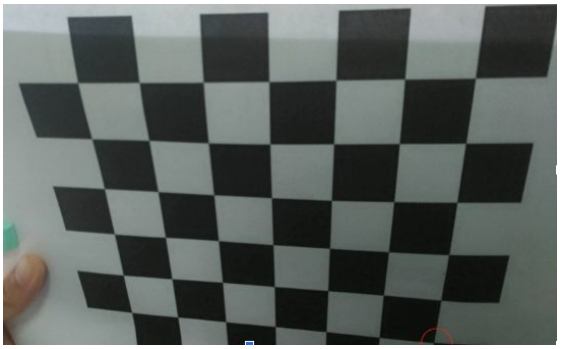
(2) Fixed/unfixed camera position

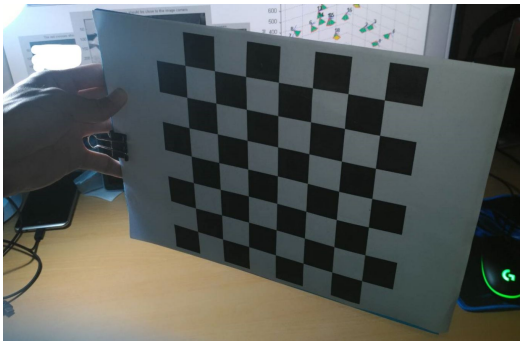
Experiments 2 and 3 show that whether the position of the camera is fixed or not, we can always get the relative position between the camera and the chessboard.

Since each graph already has **object points**, which are fixed coordinate, and **image points** detected by function cv2.findChessboardCorners. With these reference points of the chessboard, we can find the value of the homography matrix H in each graph. As long as three matrices H are acquired, matrix K can be found, then we can find the extrinsic matrix and the camera's relative position to the chessboard.

(3) The affection of image quality

In experiments 2 and 3, we found that some images with unclear chessboard due to the brightness, distance or incomplete size of view of the chessboard may affect the detection of corners. Without the image points obtained by corner detection, it cannot calculate the parameters in the homography matrix, causing the output error.





5. Conclusion

With the 2D calibration (which is more convenient than 3D calibration), we can obtain the quantized relation between the object in the world coordinate system and the camera coordinate system. We can consider how to compute 3D scene points from projections in several cameras. This task is easy if image points and camera matrices are given.

6. Work assignment

(1) Coding: 賴奕善、軒轅照雯
(2) Data preparation: 王泓仁
(3) Report: 王泓仁、賴奕善、軒轅照雯