CV HW1 Group 19 Report

0756079 陳冠聞, 0756138 黃伯凱

March 25, 2019

1 Introduction

Camera parameters include *Intrinsic matrix* and *Extrinsic matrix*. Intrinsic matrix indicates the camera's focal length, image center and distortion coefficient. On the other hand, extrinsic matrix represents the orientation and position of the camera in the world coordinate.

The main purpose of camera calibration is to estimate both **intrinsic** and **extrinsic** parameters. Since we have to remove the distortion if we want to estimate any structure in Euclidean space, intrinsic parameters are necessary. That is, camera calibration is an important technique in computer vision.

After we calibrate a camera, we can further calculate the distortion coefficient and restore distorted images, measure the size of an object in world unit, determine the position of the camera in the scene and even reconstruct 3D scene by several 2D pictures.

In this report, we will implement camera calibration algorithm. We will show both the intrinsic matrix and extrinsic matrix compared to the OpenCV result. Finally, plot the position and the orientation of the camera according to the chessboard image. The algorithm will be explained in detail in the next section.

2 Implementation

To estimate the intrinsic and extrinsic parameters of camera, we must have correspondences between object points (3D world) and image points (2D

image), so we use the chessboard image and codes provided by TA to find those correspondences (Unfortunately, the data provided by TA have the problem of inconsistent orientation, and we will show why this is a problem and how to fix it in the next section). The relationship between object point and image point is a transformation, which composed of a rigid transform from world coordinate to camera coordinate and a projective transform from camera coordinate to image coordinate. We can write the transform in matrix form

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} R & \vec{t} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1}$$

where $(u \quad v \quad 1)^T$ is the homogeneous coordinate in image plane, $(X, Y, Z, 1)^T$ is the is the homogeneous coordinate in world, and K, $(R \quad t)^T$ is the intrinsic matrix and extrinsic matrix respectively.

Since we know that the chessboard is just a 2D plane in 3D world, so the transformation is actually a homography transformation, and we can ignore the Z because it's zero for all point. Therefore, we only need to estimate the homography matrix $H \in R^{3x3}$.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} \vec{r1} & \vec{r2} & \vec{t} \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = H \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{2}$$

2.1 Estimation of homography matrix

By Equation 2, we know the relationship between object points and image points. If we rewrite the formula in matrix form with N correspondences

$$\begin{pmatrix} -X_0 & -Y_0 & -1 & 0 & 0 & 0 & u_0 * X_0 & u_0 * Y_0 & u_0 \\ 0 & 0 & 0 & -X_0 & -Y_0 & -1 & v_0 * X_0 & v_0 * Y_0 & v_0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ -X_N & -Y_N & -1 & 0 & 0 & 0 & u_N * X_N & u_N * Y_N & u_N \\ 0 & 0 & 0 & -X_N & -Y_N & -1 & v_N * X_N & v_N * Y_N & v_N \end{pmatrix} \begin{pmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{pmatrix} = \vec{0} \tag{3}$$

The Equation 3 is actually a homogeneous system $A\vec{h} = 0$, and we want don't want a trivial solution where $\vec{h} = 0$. Because the correspondences are the noisy observation, we can solve $min\|A\vec{h}\|$ to get our best estimation of H. By linear algebra, the solution can be computed using SVD, that is $U\Sigma V^T = A$, and the solution of $\vec{h}$ is the eigenvector corresponding to the minimum singular value in Σ.

Once get the $\vec{h}$, it's seem that we can get the H simply by reshape the $\vec{h}$. Unfortunately, since we get $\vec{h}$ by selecting the eigenvector, it's normal vector, so we can only get the H' which is normalized version of H. Thus, we need to denormalize H' to get back H. We first normalize the image Points p_i and object Points p_o by normalization matrix N_i and N_o respectively. Then we know $N_i p_i = H' N_o p_o$ or $p_i = (N_i^{-1} H' N_o) p_o$, and we know $p_i = (H) p_o$ by Equation 2, so we can get denormalized homography by $H = N_i^{-1} H' N_o$.

```python
# return the homography matrix H, where [u, v, 1] = H [X, Y, 1]
def get_homography(obj_points, img_points):
    obj_points = np.array([ [p[0], p[1], 1] for p in obj_points]).reshape(-1, 3, 1)
    img_points = np.array([ [p[0][0], p[0][1], 1] for p in img_points]).reshape(-1, 3, 1)

    N = img_points.shape[0]
    M = np.zeros((2*N, 9), dtype=np.float64)

    NormMat_obj = get_normalized_matrix(obj_points)
    NormMat_img = get_normalized_matrix(img_points)
    for i in range(N):
        obj_point_norm = np.matmul(NormMat_obj ,obj_points[i])
        img_point_norm = np.matmul(NormMat_img ,img_points[i])

        X, Y = obj_point_norm[0], obj_point_norm[1]
        u, v = img_point_norm[0], img_point_norm[1]

        M[2*i] = np.array([-X, -Y, -1, 0, 0, 0, X*u, Y*u, u])
        M[2*i+1] = np.array([0, 0, 0, -X, -Y, -1, X*v, Y*v, v])
    U, S, V_t = np.linalg.svd(M)
    H_norm_vec = V_t[np.argmin(S)]
    H_norm_mat = H_norm_vec.reshape(3, 3)
    H_mat = np.linalg.inv(NormMat_img).dot(H_norm_mat).dot(NormMat_obj)
    return H_mat
```

Figure 1: Code snippet for computing homography matrix

2.2 Estimation of Intrinsic matrix

After we get multiple homography matrix H_i from different viewing angles, we can use the fact that $R = (\vec{r1} \quad \vec{r2} \quad \vec{r3})$ is a rotation matrix, whose columns are orthnormal. Therefore, we can get two formula for each homography matrix

$$\vec{r_1}^T \vec{r_2} = (K^{-1}\vec{h_1})^T (K^{-1}\vec{h_2}) = 0 \tag{4}$$

$$1 = \|\vec{r_1}\| = (K^{-1}\vec{h_1})^T (K^{-1}\vec{h_1}) = (K^{-1}\vec{h_2})^T (K^{-1}\vec{h_2}) = \|\vec{r_2}\| \tag{5}$$

Let $B = (K^{-1})^T K^{-1}$, we can rewrite the formula similar to what we have done in Equation 3 to get another homogeneous system $V\vec{b} = \vec{0}$, and use SVD to get estimation of B. Then, by applying Cholesky decomposition, we can get our estimation of intrinsic matrix K.

```python
# retrurn the intrinsic matrix K
def get_intrinsic_parameters(H_r):
    M = len(H_r)
    V = np.zeros((2*M, 6), np.float64)

    for i in range(M):
        H = H_r[i]
        V[2*i] = v_pq(p=0, q=1, H=H)
        V[2*i + 1] = np.subtract(v_pq(p=0, q=0, H=H), v_pq(p=1, q=1, H=H))

    # solve V.b = 0
    U, S, V_t = np.linalg.svd(V)
    b = V_t[np.argmin(S)]
    B = np.array([
        [b[0], b[1], b[3]],
        [b[1], b[2], b[4]],
        [b[3], b[4], b[5]]
    ])

    L = np.linalg.cholesky(B)
    K = np.linalg.inv(L).T * L[2,2]
    return K
```

Figure 2: Code snippet for computing intrinsic parameters

2.3 Estimation of Extrinsic matrix

So far we have computed each homography matrix H_i and intrinsic matrix K, by Equation 2 we know that $H = K(\vec{r1} \quad \vec{r2} \quad \vec{t})$, so it's straightforward to compute extrinsic matrix by $\vec{r_1} = \lambda K^{-1}\vec{h_1}$, $\vec{r_2} = \lambda K^{-1}\vec{h_2}$ and $\vec{t} = \lambda K^{-1}\vec{h_3}$ (λ is normalized term where $\lambda = 1/\|K^{-1}\vec{h_1}\|$). For the $\vec{r_3}$, due to the orthonormality of the column vector in $R = (\vec{r_1} \quad \vec{r_2} \quad \vec{r_3})$, we can get $\vec{r_3}$ by the cross product of the other two vectors $\vec{r_3} = \vec{r_1} \times \vec{r_2}$.

```python
def get_extrinsic_parameters(K, Hs):
    K_inv = np.linalg.inv(K)
    extrinsics_pred = []

    for H in Hs:
        h0 = H[:,0].reshape(3,1)
        h1 = H[:,1].reshape(3,1)
        h2 = H[:,2].reshape(3,1)

        lambda_ = 1/(np.linalg.norm(np.matmul(K_inv, h0)))
        r0 = lambda_ * np.matmul(K_inv, h0)
        r1 = lambda_ * np.matmul(K_inv, h1)
        r2 = np.cross(r0, r1, axis=0)
        r2 /= np.linalg.norm(r2)
        tvec =  lambda_ * np.matmul(K_inv, h2)

        Rt = np.hstack((r0,r1,r2,tvec)) # world to model
        extrinsics_pred.append(Rt)

    extrinsics_pred = np.array(extrinsics_pred).reshape(-1, 3, 4)
    return extrinsics_pred
```

Figure 3: Code snippet for computing extrinsic parameters

3 Result

In this section, we will show results of the given data and our own experiment. The result contains intrinsic matrix and the extrinsic. Finally, compare our implementation to the OpenCV result.

3.1 Given Data

We found out that the result of the given data is always different from the OpenCV result. The reason is that the given images are not in the same orientation. Since cv2.calibrateCamera will take image size as parameter, all the given images have to be in the same orientation. For instance, some of the given image shapes are $3916 \times 2936 \times 3$ while others are $2936 \times 3916 \times 3$. That is, the estimation is definitely wrong. We will show both the wrong estimation and the correct result in this section.

3.1.1 Original Data

In this section, we will show the result of the given images that are not in the same orientation.

We can see the difference of the intrinsic matrix between our implementation and the OpenCV result from Figure 4. Our intrinsic has significant difference from the OpenCV result.

```
Our Intrinsic :
[ 3408.663 −36.830   1468.368 ]
[   0.000  3357.882  1415.735 ]
[ −0.000    0.000     1.000   ]
```

(a) Our Intrinsic Matrix

```
OpenCV Intrinsic :
[ 2701.930    0.000    1538.207 ]
[   0.000  2738.092    1960.135 ]
[   0.000    0.000       1.000  ]
```

(b) OpenCV Intrinsic Matrix

Figure 4: Intrinsic Matrix Comparison

Due to the difference between the intrinsic, it is obvious that the extrinsic is also different (Figure 5).

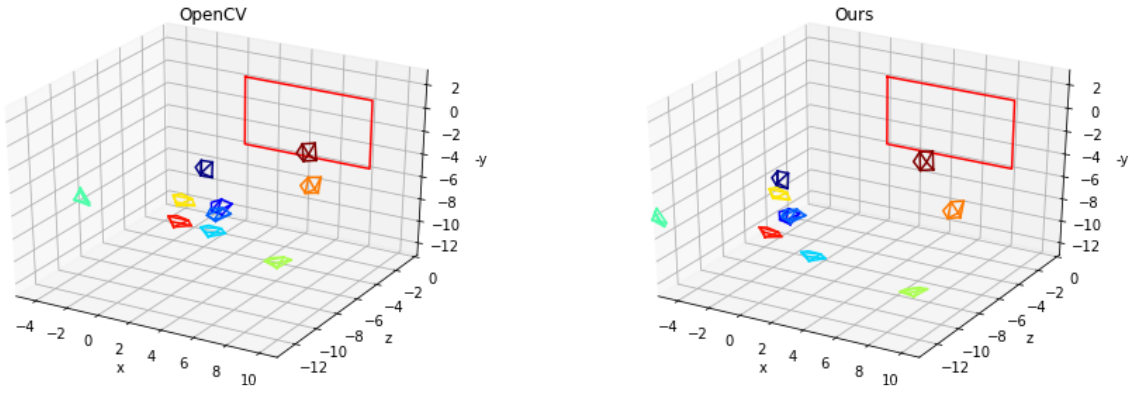


Figure 5: OpenCV Extrinsic v.s Our Extrinsic

The solution of this situation is to rotate all the images to the same orientation. We will explain it in detail in the next section.

3.1.2 Rotated Data

Since the given images are not in the same orientation, it is impossible for us to get the right coordinate of the chessboard. That is, we obtained the mistake result from the original images. The solution is to rotate all the images to the same orientation, and we will show the result in this section.

After rotating the images, the intrinsic is almost the same as the OpenCV result :

Our Intrinsic :

$$\begin{bmatrix} 2975.883 & -6.446 & 1454.546 \\ -0.000 & 2981.170 & 1961.061 \\ -0.000 & -0.000 & 1.000 \end{bmatrix}$$

(a) Our Intrinsic Matrix

OpenCV Intrinsic :

$$\begin{bmatrix} 2976.322 & 0.000 & 1481.721 \\ 0.000 & 2979.563 & 1875.997 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$

(b) OpenCV Intrinsic Matrix

Figure 6: Intrinsic Matrix Comparison

That is, the extrinsic is almost the same as the OpenCV result :

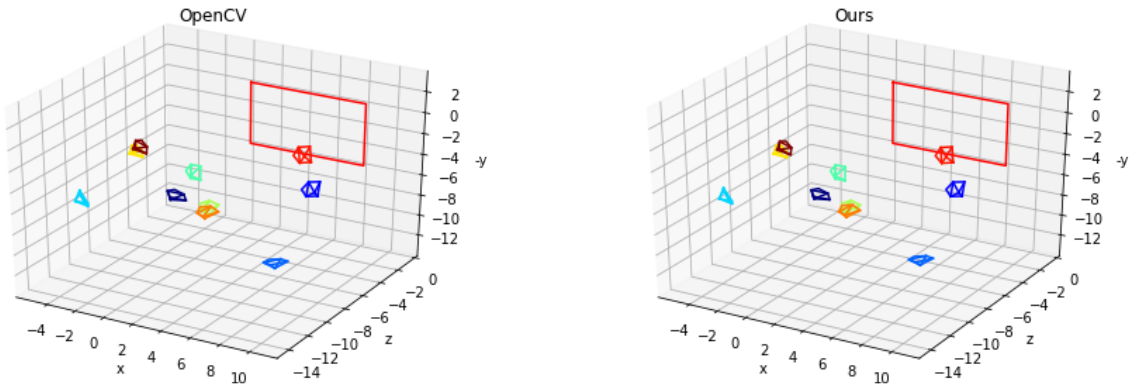


Figure 7: OpenCV Extrinsic v.s Our Extrinsic

After figuring out this problem, we notice that we have to make sure the images are in the same orientation before calibrating our cameras. So, we will do our experiment with the same-orientation images in next section.

3.2 Our Data

In our experiment, we took 10 images in the same orientation with iPhone 6S. We will show the result in the following figure.

Comparison of intrinsic :

Our Intrinsic :

$$\begin{bmatrix} 3585.622 & -6.819 & 1557.818 \\ -0.000 & 3600.746 & 1997.852 \\ -0.000 & -0.000 & 1.000 \end{bmatrix}$$

(a) Our Intrinsic Matrix

OpenCV Intrinsic :

$$\begin{bmatrix} 3518.612 & 0.000 & 1716.144 \\ 0.000 & 3503.302 & 1976.925 \\ 0.000 & 0.000 & 1.000 \end{bmatrix}$$

(b) OpenCV Intrinsic Matrix

Figure 8: Intrinsic Matrix Comparison

Comparison of extrinsic :

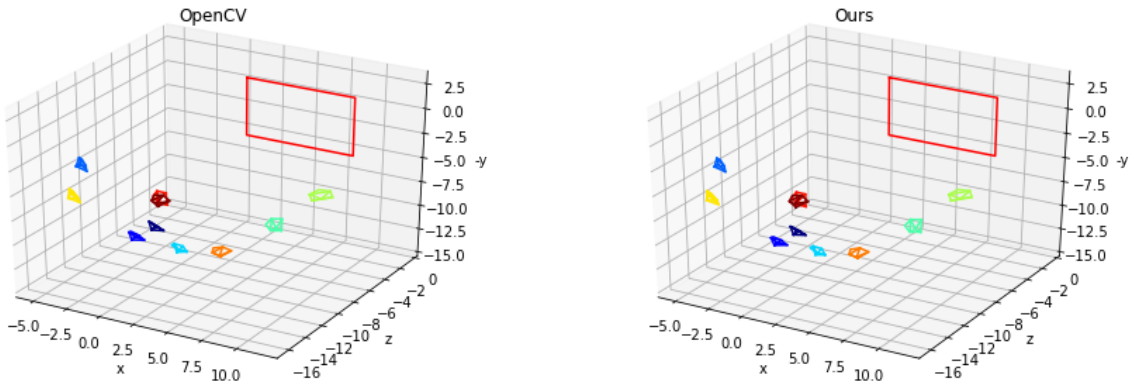


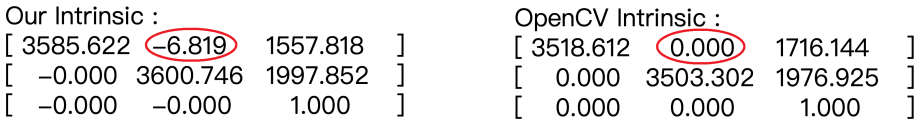
Figure 9: OpenCV Extrinsic v.s Our Extrinsic

Our implementation result is very close to the OpenCV result. That is, we have calibrated our camera successfully.

4 Discussion

4.1 Distortion Coefficient

At the beginning, when we obtained the wrong result in section 3.1.1, we thought that the error of the extrinsic may be caused by the distortion coefficient since our distortion coefficient is always a negative value (Figure 10(a)) while the OpenCV result always shows zero (Figure 10(b). However,



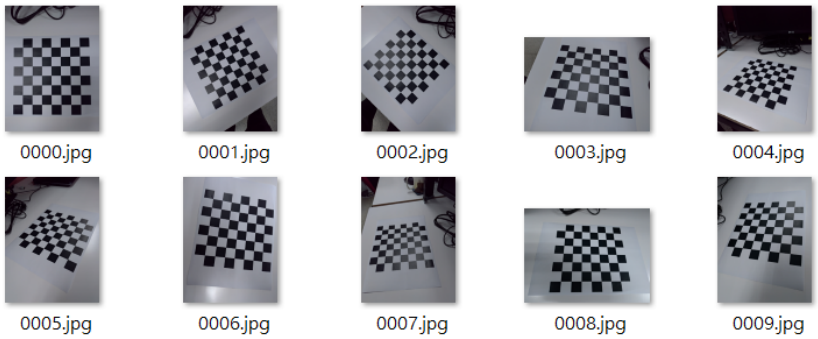
(a) Our Intrinsic (b) OpenCV Intrinsic

Figure 10: Distortion Coefficient

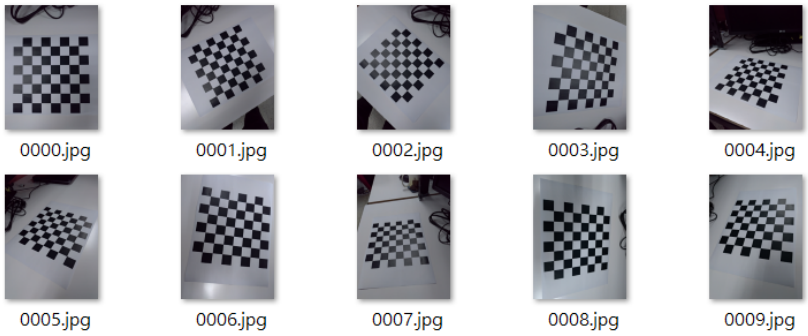
the result has nearly no difference even if we force our distortion coefficient to be zero. To prove that we were wrong, we search for the meaning of the distortion coefficient in the intrinsic matrix. We found out that the distortion coefficient indicates the skew of the plane we took by our camera. Since the chessboard is a flat plane, the skew must be zero. That is, the negative value in our intrinsic matrix may be caused by the complicated math computing or the slightly error while taking picture. In addition, distortion coefficient has nothing to do with the error since the value is not big enough to influence the result at all.

4.2 Image Orientation

After our misunderstanding to the distortion coefficient has vanished, we assume the error of the intrinsic matrix is caused by the complicated computing during calibrating since we can not find any mistake from our equation. However, we inadvertent discovered that the images are not in the same orientation in our folder (Figure 11(a)). Once we rotate them to the same orientation (Figure 11(b)), the error finally disappear.



(a) Different Orientation



(b) Same Orientation

Figure 11: Image Orientation

5 Conclusion

As we mentioned in Section 1, camera calibration enables us to achieve lots of goals when we have a number of photos. In this report, we calculate our camera intrinsic and extrinsic matrices of each image we have taken. Finally, we use the extrinsic matrices to determine the relative position of our camera to the chessboard. Notice that the orientation of images have to be the same, so that we can calibrate our cameras correctly.

6 Work Assignment Plan

0756079 陳冠聞：Code, Part of the report (Section 2)

0756138 黃伯凱：The rest of the report