HW01 Report

1. Introduction

Camera calibration is critical in computer vision to understand how a camera projects 3D world points into 2D image points. It allows for the estimation of both intrinsic parameters and extrinsic parameters. Intrinsic parameters can describe the camera's internal characteristics, like, focal length and principal point. Extrinsic parameters describe the camera's position and rotation relative to the world.

Since camera calibration can estimate the relation of the 3D world coordinates and the 2D images. It can be applied to 3D construction and robotics. Camera calibration helps to accurately measure the real world object, correct the lens estimation.

In homework 01, we are going to implement the camera calibration. The camera calibration can be divided into three parts. First, we get the homography matrices from chessboard images. Next, calculate the intrinsic matrix. Then, we can obtain the extrinsic matrix of the images by using the homography matrices and the intrinsic matrix. Finally, we show the result of these matrices.

This step-by-step approach will help us better understand how calibration works and compare the results of manual calibration with automated methods, such as OpenCV's built-in function, providing insights into both approaches.

2. Implementation

a. Find the homography matrix.

The picture below shows the relation between the object plane and the image plane.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \underbrace{\begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{\text{Homography } H} \begin{bmatrix} U \\ V \\ 1 \end{bmatrix}$$

The computed homography matrix for each image is printed and stored in a list for further calculations.
The computed homography matrices, H, for each image encode the relationship between the 3D chessboard points and the corresponding 2D image points. These matrices are crucial for estimating the camera's intrinsic and extrinsic parameters in subsequent steps.

b. Find the intrinsic matrix.

After getting the homography matrices from the image and the object plane, we are going to calculate the intrinsic matrix, K.

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

Since (r_1, r_2, t) form an orthonormal basis, thus the equation below can be obtained.

$$h_1^T K^{-T} K^{-1} h_2 = 0$$

$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2$$

And $B \equiv K^{-T} K^{-1}$ is **symmetric and positive definite**. We can obtain the matrix B by using the function, **np.linalg.svd()**, where the image number should be **larger or equal to three**. After obtaining the matrix B, we can calculate the **intrinsic matrix**, **K**, by using **the Cholesky function** or **calculate the terms in the intrinsic matrix, K,**

directly.

c. Find the extrinsic matrix.

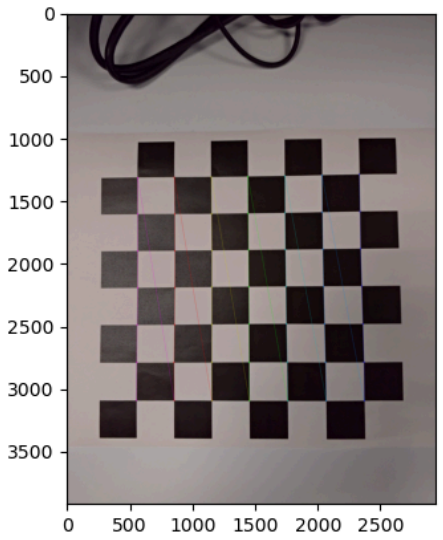
$$\mathbf{r}_1 = \lambda \mathbf{K}^{-1}\mathbf{h}_1$$
$$\mathbf{r}_2 = \lambda \mathbf{K}^{-1}\mathbf{h}_2$$
$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$
$$\mathbf{t} = \lambda \mathbf{K}^{-1}\mathbf{h}_3$$
$$\lambda = 1/||\mathbf{K}^{-1}\mathbf{h}_1||$$

For each homography matrix, follow the formula above to calculate **λ**, **r1**, **r2**, **r3**, and **t** respectively. Then, combine them together to get the extrinsic matrix. Finally, the extrinsic matrices of all the images can be obtained. With the functions that have been provided, the 3D plot visualizes the extrinsic parameters of the images.

3. Experimental result

a. The result from provided images

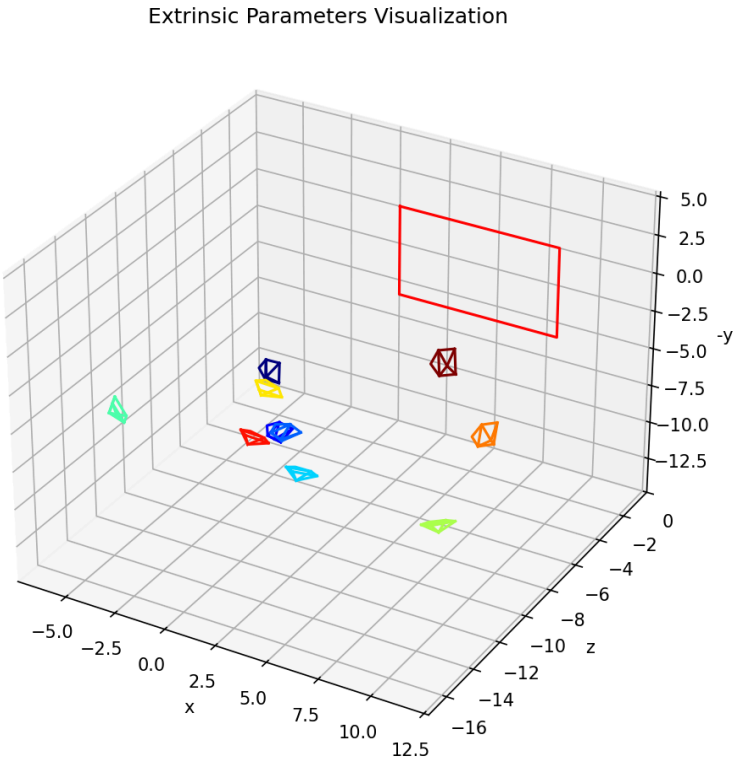
The picture below shows the corner of the chessboard has been caught.



(chessboardCorners_providedata.png)

The picture below shows the visualization of the extrinsic matrices in multiple cameras. Each frame shows the position

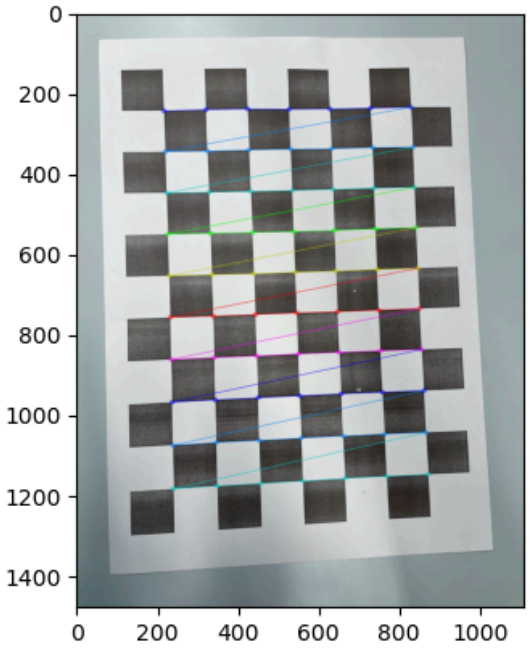
and the rotation of the camera.



(extrinsic_visualization_providedata.png)

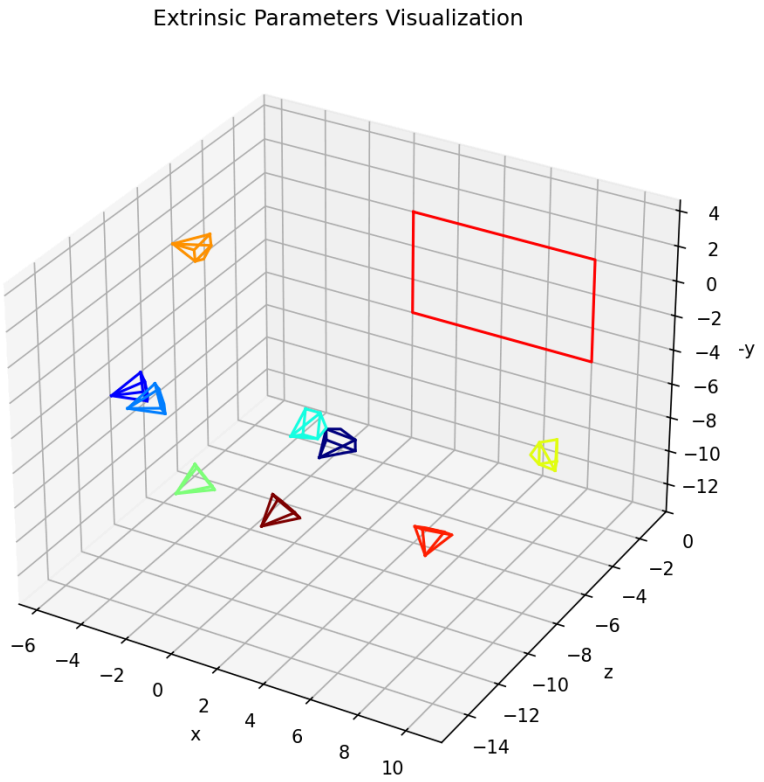
b. The result from our images

The picture below shows the corner of the chessboard has been caught.



(chessboardCorners_mydata.png)

The picture below shows the visualization of the extrinsic matrices in multiple cameras. Each frame shows the position and the rotation of the camera.



(extrinsic_visualization_mydata.png)

4. Discussion

a. Compare the accuracy of the number of images:

Ideally, the intrinsic matrix should always be the same in every image. However, we observed that the value of the intrinsic matrix might be different when the number of the input image changes.

When the number of the images is **three**, the intrinsic matrix shows as in the picture below.

```
K
array([[3.56137998e+03, 0.00000000e+00, 2.14468793e+03],
       [0.00000000e+00, 3.51579332e+03, 1.05868421e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_n=3.jpg)

When the number of the images is **six**, the intrinsic matrix shows as in the picture below.

```
K
array([[4.05216321e+03, 0.00000000e+00, 1.66049585e+03],
       [0.00000000e+00, 3.84070221e+03, 6.20223678e+02],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_n=6.jpg)

When the number of the images is **eight**, the intrinsic matrix shows as in the picture below.

```
K
array([[3.43595345e+03, 0.00000000e+00, 1.45906690e+03],
       [0.00000000e+00, 3.35567300e+03, 1.33494665e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_n=8.jpg)

When all the images are the inputs(**10 images**), the intrinsic matrix shows as in the picture below.

```
K
array([[3.40033701e+03, 0.00000000e+00, 1.47568441e+03],
       [0.00000000e+00, 3.34935946e+03, 1.40822869e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_n=10.jpg)

The intrinsic matrix by the function, cv2.calibrateCamera()

```
mtx
array([[3.18176422e+03, 0.00000000e+00, 1.64119466e+03],
       [0.00000000e+00, 3.20188958e+03, 1.42540602e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_compare.jpg)

It seems like the intrinsic matrix we obtain converges to the real intrinsic matrix when the number of the images gets larger. We think that using a larger number of calibration images resulted in minor errors in the final projection of 3D points onto 2D image points. On the other hand, fewer images lead to higher variability in the results, as each image's error has a more significant impact on the overall calibration process. With more images, the overall effect of noise or inaccuracies in individual images is reduced, leading to a more reliable estimation of the parameters.

b. Comparison to Built-in Calibration Methods:

The manual calculation of the homography matrix, while offering more profound control and understanding of the process, can be improved to achieve results closer to OpenCV's `cv2.findHomography()`. One of the critical advantages of manual calculation is its flexibility, allowing for customization of each step, such as fine-tuning the algorithm, adapting it for more complex transformations, or implementing custom weighting and smoothing techniques. However, specific improvements can be made to match the accuracy and robustness of `cv2.findHomography()`. Implementing RANSAC can help handle outliers and noise, improving the robustness of the homography estimation. Normalizing the points before solving for the homography and using Singular Value Decomposition (SVD) ensures better numerical stability. Additionally, iterative refinement methods like bundle adjustment can further minimize reprojection errors, leading to more precise results. By incorporating these techniques, the manual method can achieve results comparable to OpenCV's built-in function while maintaining the flexibility and insights gained from manual implementation.

The picture below shows the intrinsic matrices obtained by the built-in function, cv2.calibrateCamera() and the function we build. "mtx" is obtained by cv2.calibrateCamera(), and "K" is obtained by our codes.

```
mtx
array([[3.18176422e+03, 0.00000000e+00, 1.64119466e+03],
       [0.00000000e+00, 3.20188958e+03, 1.42540602e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
K
array([[3.40033701e+03, 0.00000000e+00, 1.47568441e+03],
       [0.00000000e+00, 3.34935946e+03, 1.40822869e+03],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

(intrinsic_matrix_compare.jpg)

5. Conclusion

In homework 01, we achieved the goal of manually implementing camera calibration from scratch, allowing us to compute the intrinsic and extrinsic parameters of the camera based on a series of images depicting a chessboard pattern. Several challenges were encountered throughout the project, including ensuring numerical stability during matrix operations and handling imperfect corner detection in some images. These challenges highlighted the complexity of manual calibration and provided valuable learning experiences in understanding projective geometry and matrix computations in computer vision.

6. Work assignment plan between team members.

彭宇楨: 33%, 洪義翔: 33%, 張宥楨: 34%