CV_HW2

郑琮珉 522030910137

Programming Assignment

First we need to get the pattern size. The pattern size of the chessboard is the number of inner corners per a chessboard row and column. And we get pattern size=(31,23).

Then we need to obtain the world points and the image points. For the world points, we initialize the points in world plane as (0,0), (1,0), (2,0)...(30,22). For the image points, we use the openCV's `findChessboardCorners` function. Before use this function, we need first convert the origin image to grey image. The ret value means whether we find the corners in the given image. After apply the function find_image_points, we find that not all image can find corners in it. And we only have 4 image that satisfy.

Next, we need to to solve the projection matrix **H** for one view. Like what we do in class, we can construct the relation as:

$$Ah = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 & -u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 & -v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -u_2y_2 & -u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -v_2x_2 & -v_2y_2 & -v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -u_nx_n & -u_ny_n & -u_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -v_nx_n & -v_ny_n & -v_n \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

Then use the linear least squares problem we can figure out h, where h is the eigenvector corresponding to the smallest eigenvalue of A^TA. There one thing should be stated that we can not use all points since the impact of the distortion parameters, so we only use those points that close to the central region of the images. Specifically, we choose the points according to the coordinate of the world point. The Eligible points have a coordinate $(x, y), x >= 8, x <= 24, y >= 4, y <= 20$.

The following content refers to：https://zhuanlan.zhihu.com/p/87334006

Afterwards，We need to construct B. Since B is symmetric, we can have

$$\mathbf{B} = \mathbf{K}^{-\mathrm{T}}\mathbf{K}^{-1} \equiv \begin{bmatrix} \mathrm{B}_{11} & \mathrm{B}_{12} & \mathrm{B}_{13} \\ \mathrm{B}_{12} & \mathrm{B}_{22} & \mathrm{B}_{23} \\ \mathrm{B}_{13} & \mathrm{B}_{23} & \mathrm{B}_{33} \end{bmatrix}$$

chang to

$$\mathbf{b} = \left[\mathrm{B}_{11}, \mathrm{B}_{12}, \mathrm{B}_{22}, \mathrm{B}_{13}, \mathrm{B}_{23}, \mathrm{B}_{33}\right]^{\mathrm{T}}$$

according to the derivation in the class, we have

$$\mathbf{h}_{\mathrm{i}}^{\mathrm{T}}\mathbf{B}\mathbf{h}_{\mathrm{j}} = \mathbf{v}_{\mathrm{ij}}^{\mathrm{T}}\mathbf{b} = 0$$

where $i \neq j$:

$$\mathbf{v}_{\mathrm{ij}} = \left[\mathrm{h}_{\mathrm{i1}}\mathrm{h}_{\mathrm{j1}}, \mathrm{h}_{\mathrm{i1}}\mathrm{h}_{\mathrm{j2}} + \mathrm{h}_{\mathrm{i2}}\mathrm{h}_{\mathrm{j1}}, \mathrm{h}_{\mathrm{i2}}\mathrm{h}_{\mathrm{j2}}, \mathrm{h}_{\mathrm{i3}}\mathrm{h}_{\mathrm{j1}} + \mathrm{h}_{\mathrm{i1}}\mathrm{h}_{\mathrm{j3}}, \mathrm{h}_{\mathrm{i3}}\mathrm{h}_{\mathrm{j2}} + \mathrm{h}_{\mathrm{i2}}\mathrm{h}_{\mathrm{j3}}, \mathrm{h}_{\mathrm{i3}}\mathrm{h}_{\mathrm{j3}}\right]^{\mathrm{T}}$$

then we have:

$$\begin{bmatrix} \mathbf{v}_{12}^{\mathrm{T}} \\ \left(\mathbf{v}_{11} - \mathbf{v}_{22}\right)^{\mathrm{T}} \end{bmatrix} \mathbf{b} = \mathbf{0}$$

for every view, we can get the above relation, and we finally get a **V** where $Vb = 0$. Then use the linear least squares problem we can figure out h, where h is the eigenvector corresponding to the smallest eigenvalue of $V^T V$. Then we can figure out the **B**.

K can be calculated from B using Cholesky factorization. Use the `np.linalg.cholesky` , we can get the camera matrix K. There one thing should be stated that we should normalize K by divide K_{33} .

Result

```
Camera Calibration Matrix:
[[731.17512     -4.50279686 469.43716645]
 [  0.         675.97924364 290.90471958]
 [  0.           0.           1.        ]]
Camera Calibration Matrix by OpenCV:
Camera Matrix:
 [[730.46122482   0.         456.97841613]
 [  0.         681.51960987 341.21083089]
 [  0.           0.           1.        ]]
Reprojection Error:
[np.float64(0.6668900853521109), np.float64(2.9400621468354156), np.float64(2.7132952481349646), np.float64(7.646843370955509)]
```

From the above result, we can see that $f_x = 731, f_y = 676, o_x = 469, o_y = 291$, the f_x, f_y, o_x is close to the Camera Calibration Matrix by OpenCV while o_y have a big error. This might due to the selection points of the world points and image points.

And the reprojection error is not too big, reflecting the decent accuracy.

Written Assignment

a. rewrite the the sum of the squared errors

$$E(A,T) = \sum_{i=1}^{N}(Y_i - AX_i - T)^T(Y_i - AX_i - T) = \sum_{i=1}^{N} Y_i^T Y_i - 2Y_i^T AX_i - 2Y_i^T T - 2X_i^T A^T T - X_i^T A^T AX_i - T^T T$$

In order to find the transformation (A,T) that minimizes the sum of the squared errors, we need derivatives to be zero. Then we have:

$$\frac{\partial E}{\partial A} = \sum_{i=1}^{N} 2(AX_iX_i^T + TX_i^T - Y_iX_i^T) = 0$$

$$\frac{\partial E}{\partial T} = \sum_{i=1}^{N} 2(AX_i + T - Y_i) = 0$$

so we can deduce that

$$T = \frac{1}{N}\sum_{i=1}^{N}(-AX_i + Y_i) = \hat{Y} - A\hat{X}$$

Then for the first equation:

$$\sum_{i=1}^{N} 2(AX_i + T - Y_i)X_i^T = \sum_{i=1}^{N} 2(AX_i + \hat{Y} - A\hat{X} - Y_i)X_i^T = 0$$

$$\sum_{i=1}^{N} (AX_i - A\hat{X})X_i^T = \sum_{i=1}^{N} (-\hat{Y} + Y_i)X_i^T$$

since

$$XX^T = \sum_{i=1}^{N} (X_i - \hat{X})(X_i - \hat{X})^T$$

$$YX^T = \sum_{i=1}^{N} (Y_i - \hat{Y})(X_i - \hat{X})^T$$

for

$$\sum_{i=1}^{N} (X_i - \hat{X}) = 0 \longrightarrow \sum_{i=1}^{N} (X_i - \hat{X})\hat{X} = 0 \longrightarrow \sum_{i=1}^{N} (X_i - \hat{X})(\hat{X} - X_i^T + X_i^T) = 0$$

we have

$$\sum_{i=1}^{N} (X_i - \hat{X})X_i^T = \sum_{i=1}^{N} (X_i - \hat{X})(-\hat{X} + X_i^T) = XX^T$$

Similarly, it can be concluded that

$$\sum_{i=1}^{N} (Y_i - \hat{Y})X_i^T = \sum_{i=1}^{N} (Y_i - \hat{Y})(-\hat{X} + X_i^T) = YX^T$$

so that

$$\sum_{i=1}^{N} (AX_i - A\hat{X})X_i^T = \sum_{i=1}^{N} (-\hat{Y} + Y_i)X_i^T \longrightarrow AXX^T = YX^T \longrightarrow A^* = YX^T(XX^T)^{-1}$$

also

$$T^* = \hat{Y} - A^*\hat{X}$$

Proved!

---

b. A has 3*3=9 unknown numbers and T has 3 unknown numbers. Since one $Y = AX + T$ can bring 3 equations, we need 12 equations to figure out all the unknown numbers. So 12/3=4. 4 is the minimum number of correspondences needed to estimate the transformation.