3DCV HW2

R10922134 賴奕善

Q1-1:

P3P:

```
#step1 transform 2D points from image coordinate sys to camera coordinate sys
ccs_coor = camera_matrix^-1 @ image_coor
#step2 calculate angle and distance for 3D points (Cab, Cac, Cbc, Rab, Rac, Rbc)


#step3 calculate distance ||x1 - T|| = a, ||x2 - T|| = b, ||x3 - T|| = c
```

$K_1 = (R_{bc}/R_{ac})^2$ and $K_2 = (R_{bc}/R_{ab})^2$.

$$
\begin{aligned}
0 &= (1-K_1)y^2 + 2(K_1 C_{ac} - x C_{bc})y + (x^2 - K_1) \\
0 &= y^2 + 2(-x C_{bc})y + [x^2(1-K_2) + 2x K_2 C_{ab} - K_2]
\end{aligned}
$$

Solve y

$$
\begin{aligned}
G_4 &= (K_1 K_2 - K_1 - K_2)^2 \\
&\quad - 4K_1 K_2 C_{bc}^2 \\
G_3 &= 4(K_1 K_2 - K_1 - K_2)K_2(1-K_1)C_{ab} \\
&\quad + 4K_1 C_{bc}[(K_1 K_2 - K_1 + K_2)C_{ac} + 2K_2 C_{ab} C_{bc}] \\
G_2 &= [2K_2(1-K_1)C_{ab}]^2 \\
&\quad + 2(K_1 K_2 - K_1 - K_2)(K_1 K_2 + K_1 - K_2) \\
&\quad + 4K_1[(K_1 - K_2)C_{bc}^2 + K_1(1-K_2)C_{ac}^2 - 2(1+K_1)K_2 C_{ab} C_{ac} C_{bc}] \\
G_1 &= 4(K_1 K_2 + K_1 - K_2)K_2(1-K_1)C_{ab} \\
&\quad + 4K_1[(K_1 K_2 - K_1 + K_2)C_{ac} C_{bc} + 2K_1 K_2 C_{ab} C_{ac}^2] \\
G_0 &= (K_1 K_2 + K_1 - K_2)^2 \\
&\quad - 4K_1^2 K_2 C_{ac}^2
\end{aligned}
$$

$$0 = G_4x^4 + G_3x^3 + G_2x^2 + G_1x + G_0$$

Solve x from the above quartic polynomial.

Get a, b, c with x,y.

#step4 get 2 possible camera center(T)

Compute T from a, b, c with trilateration.

#step5 calculate lambda and R

$$\lambda = \pm \mathrm{norm}(\mathrm{Points3D} - \mathrm{T})$$

$$\mathrm{R} = \lambda\,(\mathrm{Points2D}_{ccs})@\,(Points3D - T)^{-1}$$

#step6 use 4th point to choose best result

Choose minimum error solution.

RANSAC:

Determine N : $N = \frac{\log(1-p)}{log\,(1-(1-e)^s)}$

#step1: 2D points undistort.

Brown-Conrady Model:

$$\begin{aligned} x_u &= x_d + (x_d - x_c)(K_1r^2 + K_2r^4 + \cdots) + (P_1(r^2 + 2(x_d - x_c)^2) \\ &\quad + 2P_2(x_d - x_c)(y_d - y_c))(1 + P_3r^2 + P_4r^4 \cdots) \\ y_u &= y_d + (y_d - y_c)(K_1r^2 + K_2r^4 + \cdots) + (2P_1(x_d - x_c)(y_d - y_c) \\ &\quad + P_2(r^2 + 2(y_d - y_c)^2))(1 + P_3r^2 + P_4r^4 \cdots), \end{aligned}$$

where

- $(x_d,\ y_d)$ is the distorted image point as projected on image plane using specified lens;
- $(x_u,\ y_u)$ is the undistorted image point as projected by an ideal pinhole camera;
- $(x_c,\ y_c)$ is the distortion center;
- K_n is the n^{th} radial distortion coefficient;
- P_n is the n^{th} tangential distortion coefficient; and
- $r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$, the Euclidean distance between the distorted image point and the distortion center.[3]

#step2: Do P3P

#step3: calculate the number of outliers.

Choose R and T with minimum number of outliers.

Q1-2:

Self P3P with undistort

```
rotation error: 0.2858228199742912, pose error: 6.345801993809135
```

Self P3P without undistort

```
rotation error: 0.19981849308140898, pose error: 6.367368440729819
```

Opencv P3P

```
rotation error: 0.0, pose error: 0.00012192393677003609
```

Discussion:

做了一個簡單的小實驗，在其中一次不將點 undistort，出乎意料的是在 rotation 的 error 反而降低了，可能的解釋是 undistort 對 rotation 的估測並沒有幫助。

在執行程式時，P3P 不一定會有解，所以需要注意為他設 exception，防止程式直接結束。

Q1-3:

相機模型: 先畫出 image plane，原先的座標為 Pixel(image) coordinate system，乘上相機內在參數的反矩陣轉換到 Camera Coordinate System，再乘上外在參數的反矩陣加上 T 的到最後的相機原點。五個點連起來就變成金字塔型的相機模型。

Selfp3p with undistort

Selfp3p without undistort

Opencv:

Q2:

Points 的生成: 在每一面等距生成點，每一面的顏色和對面相同。

Points 顯示的順序是依照和相機的距離排序以後，從近到遠顯示。

需要注意的是照片的順序和原本算出的 R ,T 一起運算，否則方塊會沒辦法待在 3D 圖原本的位置。

Video link:

https://drive.google.com/file/d/16x9sBB_XgWyRCM416rIJUurZFGw8bmhn/view?usp=sharing

Or Download from github.

Usage:

Q1: python Q1.py

Q2: python Q2.py

Env: python 3.8 不然打不開 pkl