CS655000 Computer Vision Homework 2
Camera Calibration & Homography Transformation

108061613 黃雅琳

Part 1. Camera Calibration

A. Compute the projection matrix from a set of 2D-3D point correspondences by using the least-squares (eigenvector) method for each image.

(step 0): Normalization

在 1-A，為了降低 RMSE，首先我將 3D points 和 2D points 進行 normalization, 利用:

$$X_{normalized} = \frac{x-mean(x)}{std(x)} \quad \text{-> } x = x'^{*}std(x) + mean(x)$$

我首先找出 x = T*x’ , 再對 T 做反矩陣，得到 normalization matrix :

$$Tn = T^{-1}$$

Results: (以 Fig 1. 的 2D points 為例)

Normalization matrix , T2d_1:

```
[[ 0.01096945    0.             -1.63201092]
 [ 0.            0.01096945     -3.30363375]
 [ 0.            0.              1.        ]]
```

Normalized points , normpts_2d_1:

```
[[-0.57894338 -0.87938452]
 [-0.22792087 -0.91229288]
 [ 0.07922383 -0.94520124]
 [ 0.43024634 -0.9781096 ]
 [-0.58991283 -0.70387327]
 [-0.24985977 -0.73678163]
 [ 0.11213219 -0.76968999]
 [ 0.4631547    -0.78065944] ...]
```

(normalization function : hw201Func.py > normalization())

(step 1): compute projection matrix

利用 SVD 找到 projection matrix，並且同時做完 denomalization，

如此之後的輸入便能直接用沒有額外 normalize 的 points。

Denomalization:

Inputs: X ; Outputs:U

*Normalize procedure:

$$T_n * X = X_n \quad , \quad D_n * U = U_n$$

$$P * X_n = U_n \rightarrow P*(T_n * X) = D_n * U$$

$$P_{final} = D_n^{-1} * P * T_n$$

Fig1 . chessboard_1.jpg

Fig2. chessboard_2.jpg

Projection matrix from a set of 2D-3D points of Fig 1:

[[-2.4433e+01 4.9703e+00 7.6604e+00 -5.5956e+01]
 [1.1108e+00 -3.6226e+00 3.0335e+01 -1.8734e+02]
 [-4.2146e-03 4.0868e-02 3.7685e-02 -6.2181e-01]]

Projection matrix from a set of 2D-3D points of Fig 2:

[[-2.0341e+01 -7.2692e+00 5.5857e+00 -9.2582e+01]
 [2.0853e+00 -2.4498e+00 2.4288e+01 -1.5410e+02]
 [-1.9838e-02 2.6882e-02 2.5109e-02 -5.1460e-01]]

(step 2): 測試 RMSE，確保 P_{final} 能夠利用 3D points project 回 2D Points

rmse of P1 : 0.7566816148253328

rmse of P2 : 0.8286659577525896

(step 3) 對 P 的首三行首三列做 determinant, 若 $\det(P_{3x3}) < 0$ ，

將 P 乘上-1，詳細內容於 hw201Main 執行結果與程式碼。

B. Decompose the two computed projection matrices from **(A)** into the camera **intrinsic matrices K**, **rotation matrices R and translation vectors** t by using the Gram-Schmidt process. Any QR decomposition functions are allowed. The bottom right corner of intrinsic matrix K should be normalized to 1. Also, the focal length in K should be positive.

作法:

1. 將前面得到projection matrix 分成兩部分: $P=[M_{3x3}|t_{3x1}]$
2. 對 M_{3x3} 取norm，並將原本的M_{3x3} 除上norm值，得到 estP
3. 對estP做RQ分解，此時得到的R為R_head, Q是Q_head
4. 利用R_head 和 Q_head 找 D:

```
D = [[np.sign(Rhead[0,0]), 0, 0],
     [ 0,    np.sign(Rhead[1,1]),0],
     [ 0,    0,np.sign(Rhead[2,2])]]
```

5. 並利用R_head,Q_head,D找到K(intrinsic matrix)與R(rotation matrix)。
6. 對K除上其最右邊最底部的數字
7. 最後，用6.得到的K找translation vector: $T = K^{-1} * t_{3x1}$

結果:

```
================== fro P1 ===================
detR: 1.0
intrinsic K:
[[426.93070807  -1.66030467 191.36298232]
 [   0.         446.25643105 318.66439243]
 [   0.           0.           1.        ]]
Rotation R:
 [[-0.99224831 -0.12235778   0.02172232]
 [ 0.0986319   -0.66907203   0.73662363]
 [-0.07559784   0.73305607   0.67595397]]
translation t:
 [[   2.65004895]
 [   0.43433054]
 [-11.15339655]]
================== fro P2 ===================
detR: 1.0000000000000002
intrinsic K:
[[495.04892263  12.10980084 199.45298354]
 [   0.         510.73483003 287.77019527]
 [   0.           0.           1.        ]]
Rotation R:
 [[-0.80084402 -0.59881464   0.00835918]
 [ 0.36514726 -0.47718285   0.79935224]
 [-0.47467497   0.64320879   0.60080456]]
translation t:
 [[   0.49301034]
 [ -0.28184983]
 [-12.31304701]]
```

The focal length in K should be positive

The bottom right corner of intrinsic matrix K should be normalized to 1.

C. Re-project 2D points on each of the chessboard images by using the computed intrinsic matrix, rotation matrix and translation vector. Show the results (2 images) and compute the point re-projection root-mean-squared errors.

作法:

1. 先利用於B取得的(Intrinsic matrix)K,(Rotation matrix)R,(translation vector)T 組合回後面要用來做映射的projection matrix

   在此步驟，講義上提及兩種方法:

   (1) $$\begin{bmatrix}x\\y\\1\end{bmatrix} \sim \underbrace{\begin{bmatrix}f_x & s & c_x\\0&f_y&c_y\\0&0&1\end{bmatrix}}_{\text{Intrinsic K}} \underbrace{\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix}\mathbf{R}&\mathbf{t}\\0_3^T&1\end{bmatrix}}_{\text{extrinsic}} \begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$ 得到 proj_Mat

   (2) 得到 KRt

   $$\mathbf{m} \sim \mathbf{K}\,[\mathbf{R}\,|\,\mathbf{t}]\;\mathbf{M}$$

KRt - NumPy array

	0	1	2	3
0	-486.71	-173.931	133.65	-2215.22
1	49.8961	-58.6176	581.151	-3687.28
2	-0.474675	0.643209	0.600805	-12.313

proj_Mat - NumPy array

	0	1	2	3
0	-486.71	-173.931	133.65	-2215.22
1	49.8961	-58.6176	581.151	-3687.28
2	-0.474675	0.643209	0.600805	-12.313

   並比較兩者的差異，於數字上來看是一樣的。

   於後我使用KRt作為我的主要projection matrix。

2. 將每組3D points多加一個維度(加上1)，使其成為 $[X\ Y\ Z\ 1]^T$ homogenous coordinate
3. Predicted 2D_points = projection matrix*3D_points
4. Predicted 2D_points 此時也是homogeneous coordinate，

   將第一個維度(X)除上第三個維度(Z)，第二個維度(Y)除上第三個維度(Z)

   得到真正的projected 2D coordinate
5. 算MSE

Numerical results:

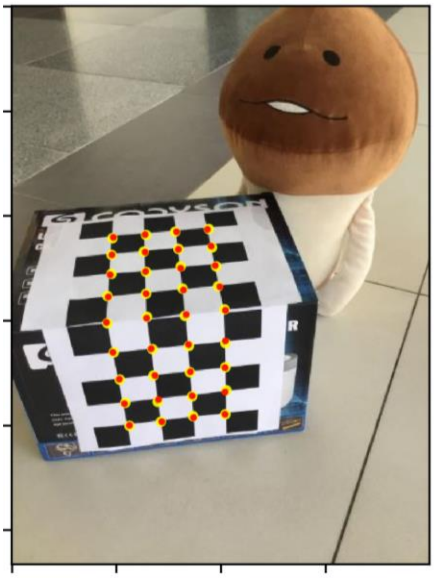
P1: RMSE : 0.756682

Real 2D points:	Reprojected 2D points:
[[96 221 1] [128 218 1] [156 215 1] [188 212 1] [95 237 1] …	[[96.5729 220.1147 1.] [127.0047 217.5326 1.] [157.1133 214.9780 1.] [186.9038 212.4503 1.] [95.1980 237.0653 1.] …

P2: RMSE : 0.828666

Real 2D points:	Reprojected 2D points:
[[101 231 1] [131 222 1] [156 213 1] [184 203 1] [121 246 1] …	[[102.0764 231.9518 1.] [130.6077 221.5357 1.] [157.4286 211.7440 1.] [182.6883 202.5222 1.] [118.8982 246.5425 1.] …

結果圖示:



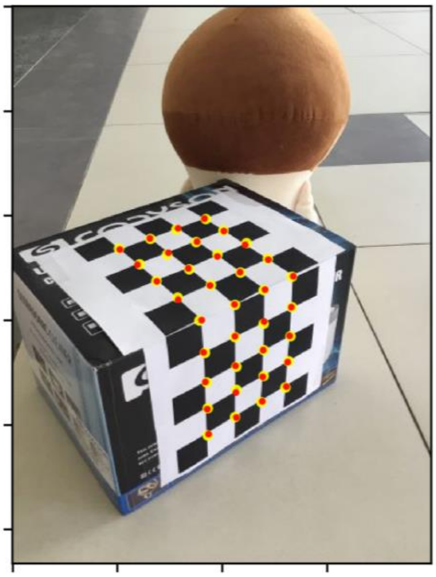


Fig 3. re-project to 2D points of (Fig1.) Fig 4. re-project to 2D points of (Fig2.)

(yellow circle:原始利用clicker.py標得到的2D點,

red circle: reproject後得到的2D點)

D. Plot camera poses for the computed extrinsic parameters (R, t) and then compute the angle between the two camera pose vectors.

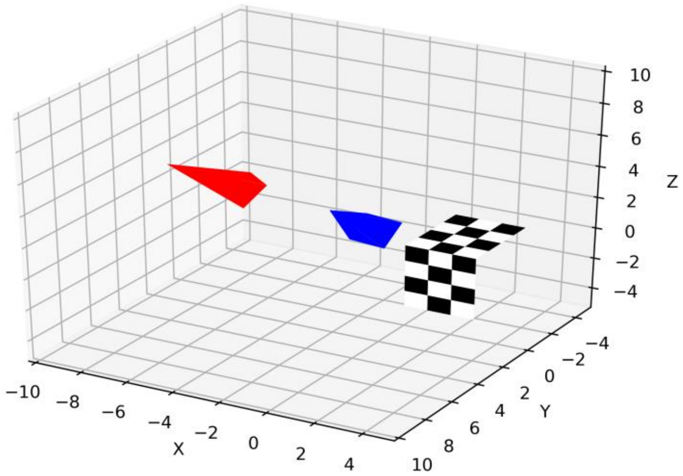


Fig 5. camera poses

Angle between two cameras: 24.00517203571544 (unit: degree)

E. **(Bonus) (10%)** Print out two “chessboard.png” in the attached file and paste them on a box. Take two pictures from different angles. For each image, perform the steps above (A ~ D).

F. **(Bonus) (10%)** Instead of mark the 2D points by hand, you can find the 2D points in your images automatically by using corner detection, hough transform, etc.

Part 2. Homography transformation

A. Shoot three images A, B and C. Image A has to contain two objects. Image B and C should contain one object separately. Like the images shown above. (3 images)

Fig 6. ImageA

Fig 7. ImageB

Fig 8. ImageC

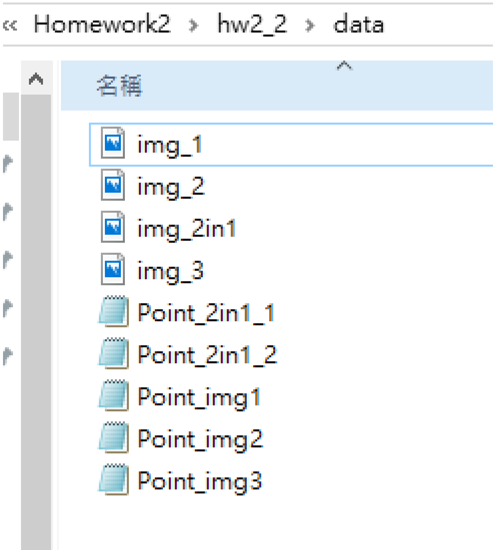


Fig 9. 檔案路徑示意圖

說明:

ImageA → img_2in1.png
ImageB → img_1.png
ImageC → img_2.png

利用助教給予的cliker.py，得到圖中欲交換區域的四個點。

於ImageA中，左邊螢幕的2D points 存於Points_2in1_1，右邊螢幕的2D points 存於Points_2in1_2; 於ImageB中，螢幕的2D points存Point_img1; 於ImageC中，螢幕的2D points存於Point_img2。

B. Compute homography transformation between the two objects in image A. Use both **backward** and **forward** warping to switch them, like what example 1 shows. (2 images)

(i) Homography:

在hw202Func.homography()，輸入points set 1 和 points set 2，就能得到 points set 1 = H*points set 2 的homography matrix。

Left to Right:

[[1.22059202e-01 -2.66595075e-01 2.48690183e+03]
[-1.59232173e-01 4.23911657e-01 1.10116550e+03]
[-1.35735045e-04 -5.01421739e-05 1.00000000e+00]]

Right to Left:

[[5.15644125e+01 1.52711346e+01 -1.45051678e+05]
[1.05097158e+00 4.94649761e+01 -5.70827883e+04]
[7.05179588e-03 4.55310958e-03 1.00000000e+00]]

(ii) Forward Warping

作法:

1. 先找好**輸入A區域**中的X軸最大及最小值，和Y軸的最大最小值，已確定有哪些像素需要做後續的工作。
2. 訂好工作範圍後，將每個像素做homography，此時得到的是homogeneous coordinate，故要將第一個維度X除以第三個維度Z，第二個維度Y除以第三個維度Z，**得到輸出B區域中 輸入A所對應的位置**。
3. 將B區域中A所對應的位置以A原本的像素取代。
4. 重複之。

Image Results:

Left to Right:

Fig 10.

Right to Left

Fig 11.

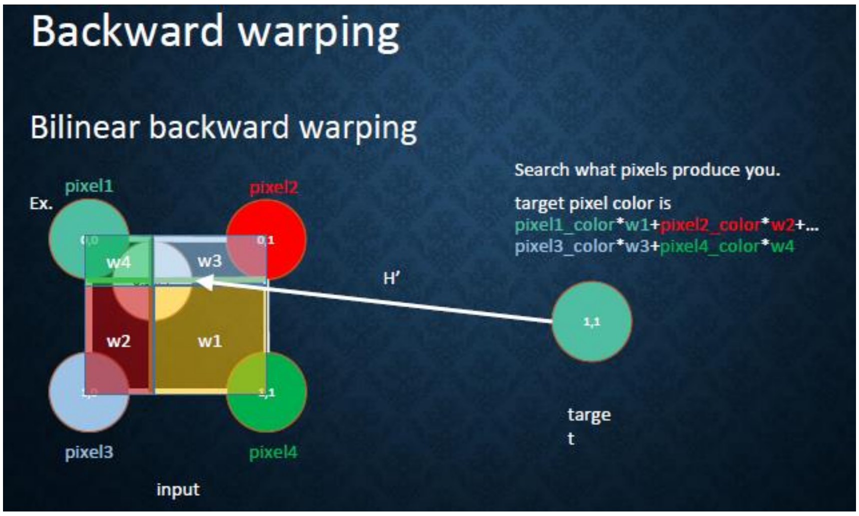
Forward Switch

(iii) Backward Warping

作法:

和Forward warping相反，Backward warping 是先知道欲輸出圖的位置，在回去找輸入圖所對應的像素。

1. 先找好**輸出B區域中**的X軸最大及最小值，和Y軸的最大最小值，已確定有哪些像素需要做後續的工作。
2. 在B區域的點中，利用homography找到**對應輸入A區域中該點的位置**。
3. 將此位置代入hw202Func的interpolation函式中，以講義的方式，對四周的像素作加權，以得到該點欲給的像素值(target pixel color)。



4. 將在**輸出B區域**的**目標位置的像素**替換成target pixel color。
5. 重複之。

Image Results:

Left to Right:

Right to Left:

Fig 13.

Fig 14.

Backward Switch

Fig 15

C. Compute homography transformation between the object in image B and the object in image C. Use both **backward** and **forward** warping to switch them. Example 2 gives some illustration. (4 images)

作法:

(同 B)

結果:

(i) Homography results:

ImageB to ImageC:

```
[[ 8.60249069e-01  -4.14499506e-02   2.17761285e+02]
 [ 5.26969514e-02   7.37318003e-01   9.78663081e+01]
 [ 8.75173611e-05  -6.22154960e-05   1.00000000e+00]]
```

ImageC to ImageB:

```
[[ 1.16803123e+00   4.38389880e-02  -2.58642341e+02]
[-6.93395539e-02   1.32166878e+00  -1.14247374e+02]
[-1.06537005e-04   7.83916061e-05   1.00000000e+00]]
```

(ii) Fordward Warping:

ImageB to ImageC:

Fig 16.

ImageC to ImageB:

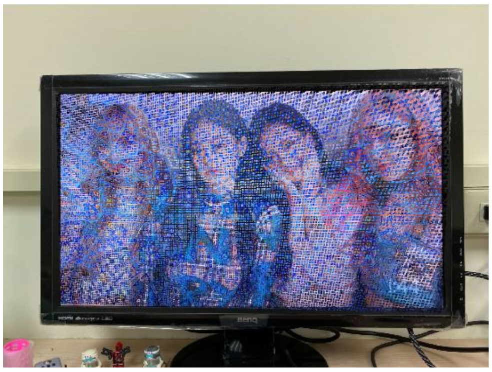


Fig 17.

(iii) Backward Warping:

ImageB to ImageC:

Fig 18

ImageC to ImageB:

Fig 19

D. Discuss the difference between forward and backward warping based on your results.

從 Fig 10,11,14,15 中，能夠觀察出只要是 object 影像區域比 target 影像區域還小的，都會出現 object pixel color 沒辦法完全填滿 target 影像區域的情形。而此種情形都是出現在使用 Forward warping 的前提下才會出現的。這是因為，Forward warping 的宗旨就是利用 object 的單顆像素直接找到他在 target 區的位置並填上該像素，故當 target 區域比 object 大時，一定會有部分像素是填不到的，會維持原本的像素。這就像是只有 10 顆糖，要分給 100 個人，一定會有人分不到。

而在 Backward warping 卻沒有此問題，因為 Backward warping 的宗旨就是，先知道 target 區域有哪些點，投影回 objet 區域，此時投影到的位置可能不會是整數，也就是沒有完全屬於其中一個像素，此時會在經由與周圍的像素(4 個)進行權重內插，藉此決定像素的值(target pixel color)，再回傳給 target point，使該 point 的像素值等於 target pixel color。如此一來，每個 target pixel 都一定保證會有一個新的 pixel color，且是和 objet pixel 相關的。

雖然利用 backward warping 能夠將較小的 object region 投影到較大的 target region，但我想他應該會有其使用極限，也就是當 object pixel 數少於一個值時，warping 出的結果也會非常差，雖然還是會將 target region，但影像猜測應該會很模糊。