三維電腦視覺與深度學習應用

3D Computer Vision with Deep Learning Application

作業二 Homework2

R11921038 江讀晉

2022/10/25

Overview

a. Program execution

- Q1-1：`python q1_1.py`
- Q1-2：`python q1_2.py`
- Q1-3：`python q1_3.py`
- Q2-1：`python q2_1.py`

b. Modules

- cv2 (OpenCV) >= 4.5.1.48
- numpy >= 1.19.5
- scipy = 1.9.1
- open3d = 0.15.1

Problem 1. Camera Pose Estimation

Q1-1. For each validation image, compute its camera pose w.r.t to world coordinate.

a. Implement one algorithm that solves a PnP problem

作業的簡介投影片提及，預期演算法為 P3P + RANSAC。然而，根據課堂第 8 個講次所使用的投影片，我認為實作 P3P 的解法整體複雜，且所求答案非唯一解，需要搭配其他 constraint 才能得到好的結果，方可預期除錯時的困難度會大幅增加。

因此，我改採用 Normalized DLT (direct linear transform)的做法，亦即找出二維座標和三維座標之間的 3x4 camera projection matrix，如 Eq. (1)所示。順帶一提的是，我沒有將二維座標的 un-distortion 考慮進去，這是將來可以改進的部分。

$$\lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = K[R|t] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}, \text{for } i_{th} \text{ point} \quad (1)$$

b. Normalized DLT: implementation & the pseudo code

Normalized DLT 算法大參考作業一，計算內容如 Fig.1 所示：先分別算出二維座標和三圍座標的 similarity transform 和 normalized points，再算出 normalized points 之間的 projection，最後再轉換為所求二維座標和三圍座標間的 projection matrix。得到 projection matrix 後，再配合已知的 camera intrinsic matrix 及參考 Eq. (1)算出 R 和 t。Pseudo code 如 Fig.2 所示。

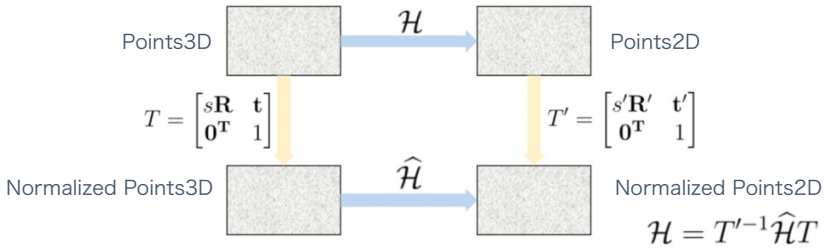

Fig.1 camera projection matrix 概念圖

1. Execute 2d-3d correspondence matching (referring to the sample code)
2. Randomly choose six index $k = random(range: 0\ to$ largeest index, $choose: 6)$
3. Sample six pairs of 2D-points and 3D-points $points2D,\ points3D \leftarrow k\ pairs$
4. Execute Normalized DLT $projection = NDLT(points3D, points2D)$
5. Compute pose $pose = dot(inverse\ of\ camera\ matrix,\ projection)$
6. Compute rotation $rotation_{matrix} = pose(column\ 1\ to\ 3\ )$
7. Compute translation $translation_{vector} = \frac{pose(column\ 4)}{norm(rotation_{matrix}\ first\ row)}$

Fig.2 使用 normalized DLT 求 pose 的描述

c. RANSAC: implementation & the pseudo code

課堂投影片僅介紹二維 line fitting 的 RANSAC 實作方法，因此我上網搜尋 pose estimation RANSAC 的實作建議。根據 Calibrated Camera Resectioning Using P3P in RANSAC Scheme（Matoušek, 2010, p.2）所述，通常 inlier 的判定由三維座標之 Euclidean projection error 去判定，在門檻值以內的判定為 inlier，反之為 outlier，並在每次迭代中更新最大迭代次數，以避免找不到 inlier。然而本題我僅有 2d-3d correspondence points，並沒有額外的 ground truth，因此我改用 translation error (Euclidean distance)大小去判斷該次算出的 pose 是否保留，並預先設定迭代次數上限。Pseudo code 如 Fig.3 所示。

1. Execute 2d-3d correspondence matching (referring to the sample code)
2. Do normalized DLT find $rotation_{matrix}$, $translation_{vector}$
3. Compute error $error = norm_2(translation_{vector},\ translation_{groundtruth})$
4. If $error < threshold$, then finish the search process
5. Otherwise, repeat step 2 to 4 again

Fig.3 找尋較佳 pose 解的描述

為了方便觀察，我將（重複）計算 normalized DLT 的次數顯示出來，如 Fig.4 所示。值得注意的是，有幾張 valid image 的 error 值計算多次後仍無法找到足夠好的解，此結果也會在 Q1-3 顯示出數個明顯的 camera pose outlier。

當所有 valid image 的 pose 的計算完後，將 pose 還有 projection matrix 儲存起來，以供後面數題使用。

```
imgid 269
     IMAGE_ID             NAME       TX        TY        TZ        QW        QX        QY        QZ
268       269  valid_img570.jpg  4.60982 -0.072615  0.167618  0.843358 -0.017248 -0.534776 -0.049654
total counts of DLT computation: 131 times
imgid 270
     IMAGE_ID             NAME       TX        TY        TZ        QW        QX        QY        QZ
269       270  valid_img575.jpg  4.70341 -0.047515 -0.016797  0.849108 -0.014966 -0.525592 -0.050446
total counts of DLT computation: over 5000 times!
```

Fig.4 Q1-1 執行結果

Q1-2. Compute median pose error of rotation and translation.

本題利用 Q1-1 所儲存的 q1_pose.npy 計算 pose error。Rotation 的部分，我是將 matrix form 轉成 quaternion form，再由 q_w 算出角度，即$\theta = \cos^{-1}(q_w)$；Translation 則是計算 L2-norm。如 Fig. 5 所示。

Median rotation error	Median translation error
0.05765	0.00623

Fig.5 Q1-2 執行結果

此外，我也使用 opencv 的 solvePnP 計算 pose，並比較我自己計算的結果，rotation 大約差了 0 至 1 個數量級，translation 則是 1 至 2 個數量級。

Q1-3. Plot the trajectory and camera poses along with 3d point cloud model.

因為要繪製金字塔型，需要頂端座標（也就是相機中心座標）以及 base 的法向量。計算相機中心，使用 Q1-1 所計算的 rotation matrix 以及 translation vector，即$C_{camera} = R^{-1}T$。至於法向量和 base 平面的計算，根據課堂投影片圖示，可以得出

$$\begin{cases} normal\ vector = Z_W \\ base\ plane = E_{X_W Y_W} \end{cases}$$

因此我計算金字塔五個頂點的方式如 Fig.6 所示。

1. Find camera center $C_{camera} = R^{-1}T$
2. Find normal vector $Z_W = R^{-1}Z_C$
3. Compute projection of camera center on base plane $C_{proj} = C_{camera} + Z_W$
4. Use 4 vector $(X_W, Y_W), (-X_W, Y_W), (-X_W, -Y_W), (X_W, -Y_W)$ from C_{proj} to find the vertices of the base, where $X_W = R^{-1}X_C$ and $Y_W = R^{-1}Y_C$

Fig.6 計算金字塔五頂點的描述

而計算相機軌跡 camera trajectory，則是利用上述計算出的 camera center 組成 line set，將其連線起來即為軌跡。只是因為 image id 和軌跡順序不大一致，因此有手動調整 camera center 連接順序，而非全按照 image id 排序。

根據結果來看，大致相機中心以及軌跡皆為合理範圍，但可以注意到有少數幾個相機中心及法向量方向皆有極大誤差，這也對應了在 Q1-1 時有數張影像計算出的 error 都很大。目前推測為那幾張照片的 distortion 較為嚴重，因我忽略 un-distortion 而造成較大誤差。也可能為我找較佳解的方式並不適用這幾張照片的資料點，因此造成誤差。本題結果如 Fig.7 所示。

Fig.7 Q1-3 呈現結果，棕色為 camera pose，黃色為 camera trajectory

Problem 2. Virtual Cube Augmented Reality

Q2-1. Place a cube and draw voxels in validation images with Painter algorithm.

Painter algorithm 主要是比較個物體間的深度，去決定在二維影像上的繪製順序，越靠近相機的部分會越晚繪製，才能蓋住較深的部分。由於 transform_cube.py 所輸出之 cube 僅有八個頂點，但直觀來想，同一個面應該會一起同時繪製，最後看到的情況為前三面疊在後三面之上。因此，我先計算出六個面的中心點，再去比較六個中心點在三維座標和相機中心的距離，並加以排序。Fig.8 為輸出影面其中一段畫面的截圖，可以看出酒紅、青綠、靛藍三面疊在草綠、青色、湖水綠三面之上。

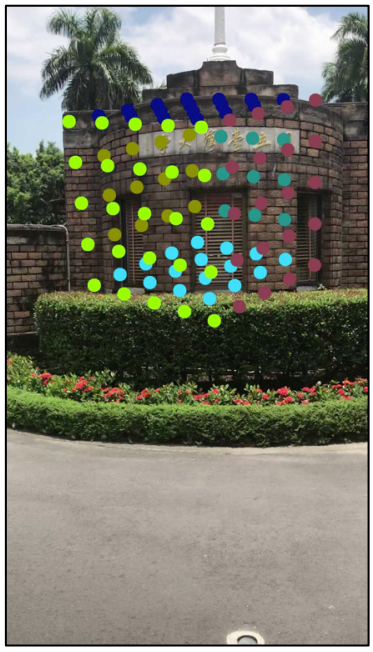

Fig.8 Q2-1 實作結果

Acknowledgement

由於本次作業數學計算較為複雜，感謝資工李昱廷同學給的建議。主要由以下幾部分：

1. PnP 的解決方向：由於 P3P 按照投影片實作起來複雜，因此向李同學詢問是否知道其他的 P3P 實作演算法可以參考。和他討論後，我權衡個演算法的複雜與否，改採用較直觀的 DLT。
2. 相機中心座標點：由於課堂投影片的 notation 和其他資料並不完全相同，我誤會相機座標的公式。因幾次嘗試及找答案未果，遂詢問他人正確公式。
3. 相機 pose 金字塔實作：我原本的作法為分別求出在 CCS 的五個座標點再轉換至 WCS，但效果不彰。和他討論後，他建議我改先找出法向量的方式。

整體而言，我和他討論主要著重在實作方向，以及核對數學公式是否有帶入錯誤，並沒有在程式碼的撰寫上有所討論，但他的建議對於我進度的推動影響顯著，故在此說明其貢獻。

Reference

Perspective-n-Point (PnP) pose computation

1. Pose computation overview
   OpenCV: Perspective-n-Point (PnP) pose computation
2. Solve PnP
   OpenCV: Camera Calibration and 3D Reconstruction

Direct linear transform

1. Estimating the Homography Matrix with the Direct Linear Transform (DLT)
   Estimating the Homography Matrix with the Direct Linear Transform (DLT)
2. Github: DLT
   DLT/DLT.py at master · acvictor/DLT

RANSAC on PnP

1. Calibrated Camera Resectioning Using P3P in RANSAC Scheme
   http://cmp.felk.cvut.cz/cmp/courses/TDV/labs/p3pransac.pdf

Camera position

1. How to find camera position and rotation from a 4x4 matrix?
   How to find camera position and rotation from a 4x4 matrix?

Open3d

1. Official documents
   Visualization - Open3D 0.7.0 documentation