三維電腦視覺與深度學習應用

3D Computer Vision with Deep Learning Application

作業二 Homework3

電機所 R11921038 江讀晉

2022/11/22

Synopsis

a. Program execution
   - `python vo.py ./frames`

b. Modules
   - cv2 (OpenCV) >= 4.5.1.48
   - numpy >= 1.19.5
   - open3d >= 0.12.0

Camera Calibration

純粹使用 repository 提供的程式碼選取影像並計算出 camera calibration numpy file，argument 皆使用預設，並無特別更改。對於選取影像，則是看到影片中棋盤格沒有嚴重偏移出畫面時，頻繁按下 space 鍵選畫面。

Feature Matching

此部分參考投影片建議，使用 ORB (oriented fast and rotated brief) descriptor 加上 brute-force matching。此外，將兩張影像 matching 的點分別以 pts1 和 pts2 回傳，以方便後續步驟計算 essential matrix、camera pose 和 triangulate points，並在影像上畫出 matching 所使用的 points。大致流程如 Fig.1。

ORB_BFmatching: input - image1, image2 / output - points1, points2

1. Initiate ORB detector and BF matching object (via OpenCV)
2. Find the keypoints and descriptors with ORB
3. Match desciptors
4. Sort the matching results in the order of their distance.
5. Use query index and train index separately to find matching points for image1 and image2 $\rightarrow points1, points2$
6. Return points1 and points2

Fig.1 feature matching 流程概述

Pose from Epipolar Geometry

主要的計算皆使用 OpenCV 函式，pseudo code 和描述如 Fig.2 所示。

	Find_camera_pose:	
1	**for_loop** from path1 to pathN, **do**	
2	img ← imread(path), img_old ← imread(path_old)	# Capture new frame
3	pts1_ori, pts2_ori ← ORB_BFmatching(img, img_old)	# ORB descriptor and matching
4	pts1, pts2 ← undistort(pts1_ori, pts2_ori, camera_matrix, distortion_matrix)	# Undistort the matched points
5	E ← find_essential_matrix(pts1, pts2, camera_matrix)	# Find essential matrix
6	R_rel, t_rel, tri_pts ← recover_pose(E, pts1, pts2, camera_matrix)	# Get camera pose
7	P_rel ← [R_rel, t_rel]	
8	**If** idx = 0 **do**	# For first frame, just store the pose
9	P_cum ← P_rel	
10	**end**	
11	**Otherwise, do**	# For other frames, compute the
12	index ← coord_matching(pts1, buff(pts2))	scale by using the same points from
13	scale ← get_scale(tri_pts, buff(tri_pts))	previous loop and current loop
14	P_rel updates with scaled t_rel	# Use rational scale to update t_rel
15	P_cum ← dot(P_cum, P_rel)	# Compute cumulative pose
16	**end**	
17	buff ← pts2, tri_pts, t_rel	# Store information of current loop
18	queue ← R_cum, t_cum from P_cum	# Put the pose into the queue to
19	**end**	draw the pose pyramid

Fig.2 find camera pose pseudo code and comment

就 translation vector scale 的計算，投影片所述為使用兩相同的 scene points pairs。然而，前後迴圈有共同的部分為，上個迴圈的 train image 和當次迴圈的 query image 是相同的。由於都是使用相同 descriptor，可以預期會找出相同座標的點，僅是次序不大相同而已，因此，第 12 行先找出相同座標的 index，再計算 scale。

此外，根據影片可以看出每個 frame 為接近均勻的頻率由影片擷取，因此預期 translation vector 的大小相近。然而實際計算發現，仍會存在明顯的誤差，因此在第 13 行找 scale 時，將 scale 限縮至 0.5 至 2 倍的合理區間，以避免畫圖時，camera pyramid 在畫面中到處噴飛的情形。

Results Visualization

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。首先，從 queue 中 pop 已計算好的 rotation matrix 以及 translation vector，然而值得注意的是，OpenCV 函式庫所計算出的 pose 為 frame 座標固定 point 移動，而非 point 固定 frame 或相機移動。因此，

需要先將 pose 執行 inversion，即

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & -R^{-1}t \\ 0 & 1 \end{bmatrix}$$

才是得到正確的 camera pose。之後，計算相機中心，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大致和作業二相同，如 Fig.3 所示。

1. Inverse the pose popped from the queue to get correct camera pose

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & -R^{-1}t \\ 0 & 1 \end{bmatrix}$$

2. Find camera center $C_{camera} = R^{-1}T$
3. Find normal vector $Z_W = R^{-1}Z_C$
4. Compute projection of camera center on base plane $C_{proj} = C_{camera} + Z_W$
5. 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.3 計算金字塔五頂點的描述

Results

事實上，依照上述流程計算出的 visual odometry，可以看出連貫的軌跡且 base 方向正確，但是和實際軌跡相距不小，如 Fig.4 所示，黑色為起點，粉湖水綠為終點。

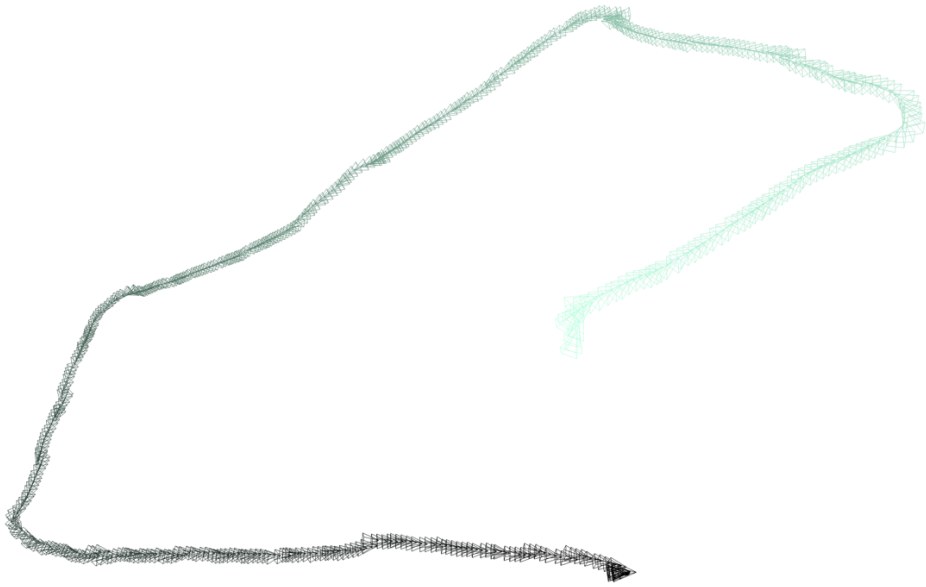

Fig.4 執行後的結果，可發現首尾高低差極大，且矩形路徑嚴重變形

明顯地，Fig.4 有幾處不連貫的 camera trajectory，即為造成圖形變形的原因。因此，我試著將幾個 frame 的 camera pose 人工加上些微的旋轉，其校正結果如 Fig.5 所示。當然，這並非 visual odometry 的本質，純粹紀錄實驗歷程於此。

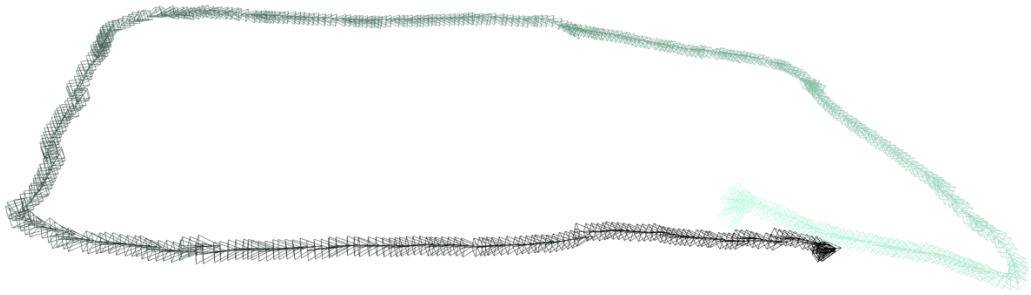

Fig.5 人工校正成果，可發現首尾高低差減小，且路徑形成較為明顯的長方形

YouTube Link

[2022 Fall] 3D Computer Vision Homework 3 -- Visual Odometry Demo
https://youtu.be/kn1ns2YohVU

Reference

[1] jaystab. opencv 3 essentialmatrix and recoverpose. 2018
https://answers.opencv.org/question/31421/opencv-3-essentialmatrix-and-recoverpose/