**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**

|  |  |
| --- | --- |
| 1. Program execution  * python vo.py ./frames | 1. 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 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。

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **Find\_camera\_pose**:  **for\_loop** from path1 to pathN, **do**  img ← imread(path), img\_old ← imread(path\_old)  pts1\_ori, pts2\_ori ← ORB\_BFmatching(img, img\_old)  pts1, pts2 ← undistort(pts1\_ori, pts2\_ori, camera\_matrix, distortion\_matrix)  E ← find\_essential\_matrix(pts1, pts2, camera\_matrix)  R\_rel, t\_rel, tri\_pts ← recover\_pose(E, pts1, pts2, camera\_matrix)  P\_rel ← [R\_rel, t\_rel]  **If** idx = 0 **do**  P\_cum ← P\_rel  **end**  **Otherwise, do**  index ← coord\_matching(pts1, buff(pts2))  scale ← get\_scale(tri\_pts, buff(tri\_pts))  P\_rel updates with scaled t\_rel  P\_cum ← dot(P\_cum, P\_rel)  **end**  buff ← pts2, tri\_pts, t\_rel  queue ← R\_cum, t\_cum from P\_cum  **end** | # Capture new frame  # ORB descriptor and matching  # Undistort the matched points  # Find essential matrix  # Get camera pose  # For first frame, just store the pose  # For other frames, compute the scale by using the same points from previous loop and current loop  # Use rational scale to update t\_rel  # Compute cumulative pose  # Store information of current loop  # Put the pose into the queue to 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

才是得到正確的camera pose。之後，計算相機中心，即。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

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 2. Find camera center 3. Find normal vector 4. Compute projection of camera center on base plane 5. Use 4 vector from to find the vertices of the base, where and |

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。

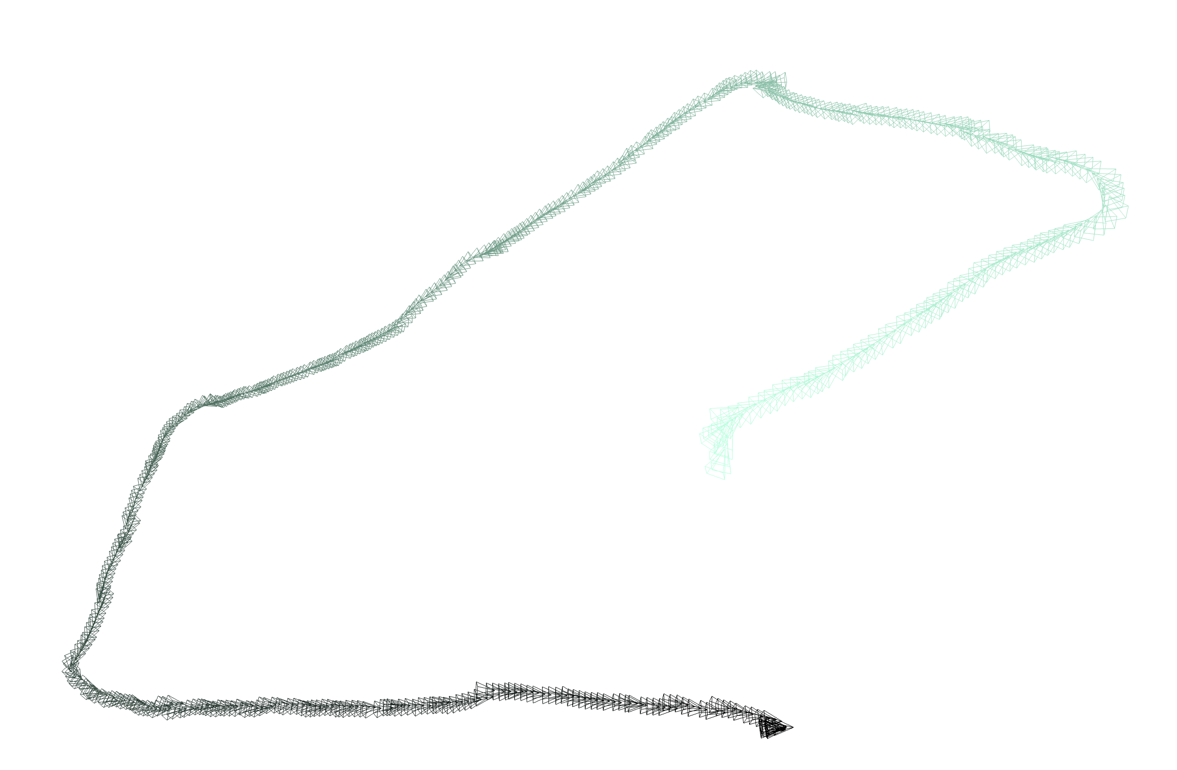


Fig.4 執行後的結果，可發現首尾高低差極大，且矩形路徑嚴重變形

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。

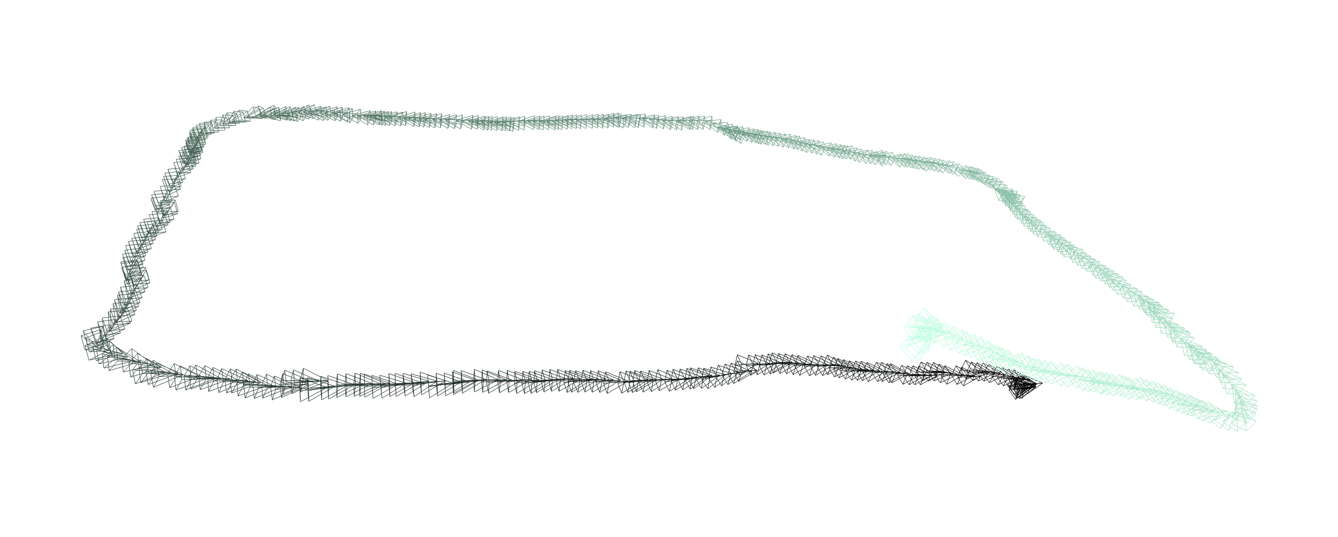


Fig.5 人工校正成果，可發現首尾高低差減小，且路徑形成較為明顯的長方形

**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/>