3DCVDL HW2 Report

Q1

https://youtu.be/qB3hPdfpmEc

使用手機拍攝台大舊體育館前草皮上的 NTU 裝置藝術，長度約 19 秒的影片。

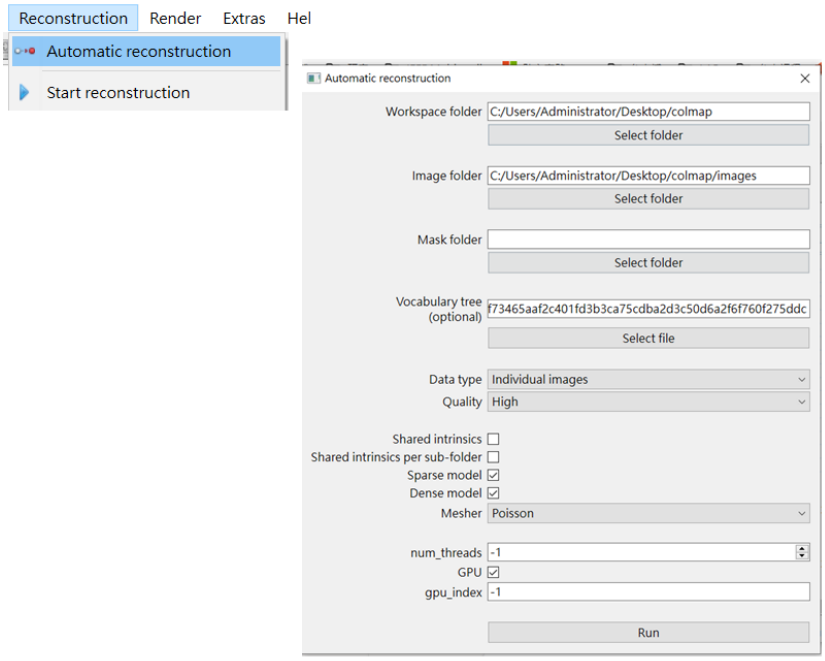
Q1-1: COLMAP Structure from Motion

步驟 1: 資料準備：從影片擷取圖像，使用線上工具 ezgif 並裁成 720x720 的影像

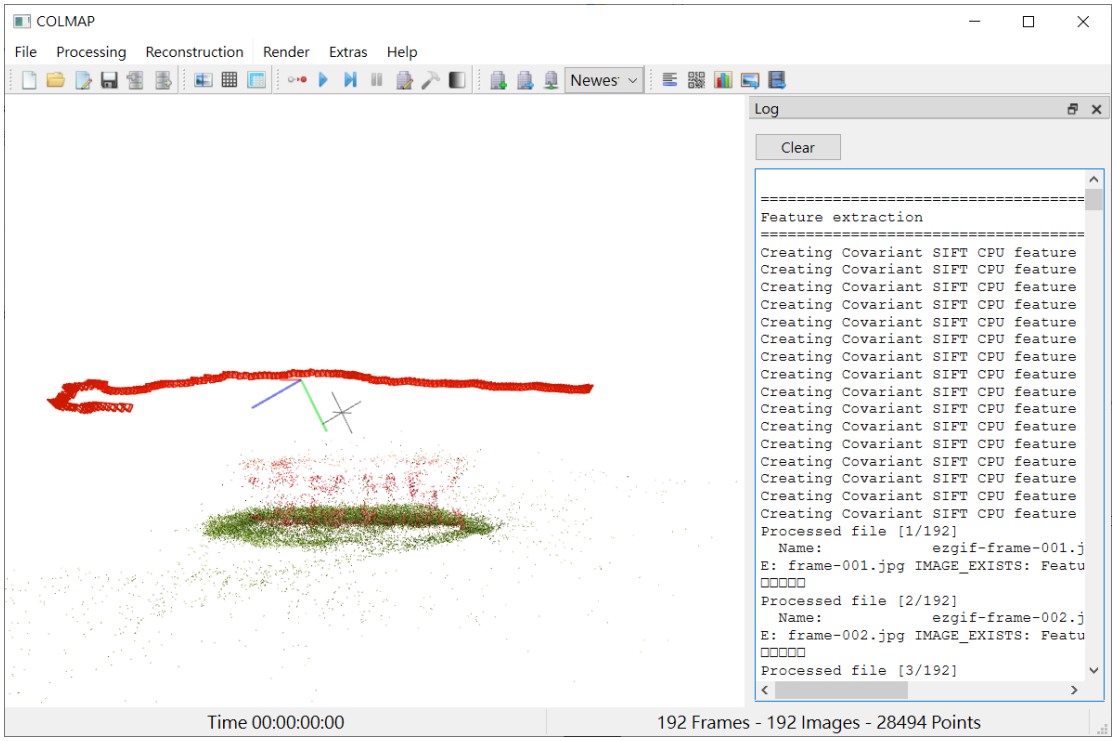
步驟 2: COLMAP 執行：

開啟 COLMAP 點選 Reconstruction → Automatic reconstruction

設定工作資料夾與圖片資料夾後點選 “Run”



結果截圖（Results）：



Q1-2: 點雲轉 3D 網格模型

將輸出 稠密點雲 (fused.ply) 使用 Python + Open3D 生成 mesh model

Poisson Surface Reconstruction

```
# 法向量估計 (Poisson reconstruction 需要法向量)
pcd.estimate_normals(search_param=o3d.geometry.KDTreeSearchParamHybrid(
    radius=0.05, max_nn=30))

# Poisson 表面重建
mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(
    pcd, depth=10)

# 移除低密度部分（去除浮動碎片）
vertices_to_remove = densities < np.quantile(densities, 0.05)
mesh.remove_vertices_by_mask(vertices_to_remove)

# 儲存
o3d.io.write_triangle_mesh("mesh_poisson.ply", mesh)
o3d.visualization.draw_geometries([mesh])
```

結果截圖

Q2

使用提供的台大前門數據集，從 2D-3D 對應關係進行相機姿勢估計，以及擴增實境 (AR) 可視化。

https://youtu.be/YwsilzWkq7U

環境

- Python 3.10+
- 套件：pip install opencv-python scipy numpy pandas open3d tqdm

如何執行

- 將資料置於 ./data/（images.pkl, train.pkl, points3D.pkl, point_desc.pkl, frames/）。
- Q2-1：python 2d3dmatching.py --q1 --vis（估計姿勢、印出誤差中位數、儲存 est_poses.npy、執行 Open3D 可視化）。
- Q2-2：python 2d3dmatching.py --q2（生成 ar_video_est.mp4）。
- 調整立方體：先執行提供的 transform_cube.py 設定頂點並儲存 cube_vertices.npy。

Q2-1

步驟 1：相機姿勢估計

針對每張驗證圖像（依 NAME 中的 "valid" 篩選，並依數字 ID 排序），我們計算相對於世界坐標系的相機姿勢（旋轉 rvec 與平移 tvec）。

描述符匹配：

從 point_desc.pkl 載入圖像的查詢特徵點 (XY) 與描述符。針對模型，從 train.pkl 與 points3D.pkl 依每個 3D 點平均描述符，使用 average_desc（依 POINT_ID 分組、堆疊並取平均）。使用暴力匹配 (cv2.BFMatcher 搭配 L2 範數) 及 Lowe 比率測試（匹配距離 < 0.75 * 次佳距離）找出 2D-3D 對應關係。要求至少 6 個良好匹配。

PnP 求解：實作 RANSAC + P3P 而非使用 OpenCV 的 solvePnPRansac。

- 去畸變：預先將 2D 點去畸變至正規化坐標，使用迭代 Brown-

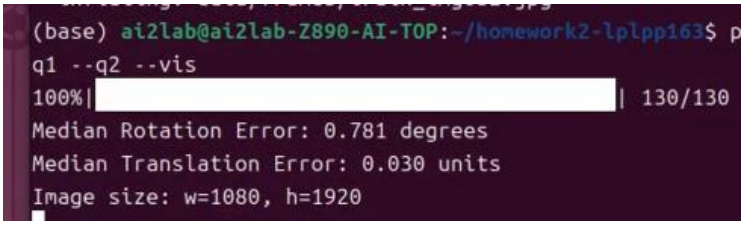
Conrady 模型（最多 5 次迭代）搭配給定的內參 K 與畸變 distCoeffs。

- RANSAC 迴圈（2000 次迭代、99% 信心水準）：抽樣 4 個點（3 個用於 P3P、1 個用於驗證）。於正規化坐標使用 P3P。
- P3P 實作（Grunert 方法）：
  1. 計算視射向量（正規化去畸變點）。
  2. 預計算 3D 三角形的邊長 a,b,c。
  3. 求解四次方程式以得尺度 v（實數正根）。
  4. 針對每個有效 v，計算尺度 s1,s2,s3，投影至相機框架，透過 Procrustes 對齊（對中心化點進行 SVD）得 R,T。
  5. 驗證第 4 點的重投影誤差（>2 倍閾值則跳過）。
  6. 使用 project_points 投影所有點（處理 K、透過前向投影處理畸變），計數內點（<2px 重投影閾值）。
  7. 選擇最佳姿勢（最大內點數）；如需則精煉。
- 將最佳 R,T 轉換為 rvec,tvec。若 <6 匹配或無有效姿勢則跳過。

步驟 2：姿勢誤差計算

針對每個估計姿勢，從 images.pkl 比較真值（四元數 QW,QX,QY,QZ；平移 TX,TY,TZ）。姿勢為世界至相機 (W2C)。

- 平移誤差：歐幾里德距離 ||t_est - t_gt||_2。
- 旋轉誤差：兩者轉換為矩陣 R_est, R_gt。相對 R_rel = R_est @ R_gt.T。角度 = acos(clip((trace(R_rel)-1)/2, -1,1)) * 180/π（軸角幅度）。



步驟 3：可視化

將估計 W2C rvec,tvec 轉換為相機至世界 (C2W) 矩陣：R_cam = from_rotvec(rvec).as_matrix(), T_cam = tvec，然後 R_world = R_cam.T, T_world = -R_world @ T_cam，堆疊成 4x4 c2w。

- 點雲：載入 points3D.pkl 的 XYZ/RGB，創建 Open3D PointCloud。
- 相機視錐：針對每個姿勢，定義金字塔頂點（頂點 [0,0,0]，底面於 z=1 縮放 0.1）。透過 c2w[:3,:3] @ verts.T + c2w[:3,3] 轉換至世界。繪製為紅色 LineSet（邊線：頂點至底面、底面四邊形）。
- 軌跡：綠色 LineSet 依序連接 C2W 平移。

使用 draw_geometries 顯示。

Q2-2

從 est_poses.npy 載入估計姿勢。篩選/排序測試圖像（驗證集）。使用 K 與 distCoeffs。立方體頂點來自 cube_vertices.npy（單位立方體於手動位置/方向/縮放，例如置中於大門基部）。

- 立方體面：定義 6 個面（四邊形索引），指派 BGR 顏色（紅/綠等）。
- 取樣：每面雙線性取樣 16x16=256 點（密度=15）於四邊形。
- 投影：cv2.projectPoints 搭配姿勢、K、畸變。篩選邊界內 (0<w,h) 及視野 (arctan 檢查 vs. 焦距)。
- 畫家演算法：每幀，將面中心轉換至相機 Z (R_w2c @ center + tvec)。依降序 Z 排序面（最遠先）。依序繪製每面點 (cv2.circle, 半徑=10, 填充顏色)。
- 影片：透過 cv2.VideoWriter 寫入 10 FPS MP4 (1080x1920)。