3DCV Homework 2 Report

Problem 1:

Problem 2:

這次Problem 2 其實最花時間的部分應該還是RANSAC + P3P的實作，尤其是P3P內部有需多數學公式的推導跟簡化，下面會著重在這部分進行說明。

P3P 演算法實作原理的細節，主要參考了網上這篇文章的說明: [P3P相機姿態估計數學推導](https://blog.csdn.net/leonardohaig/article/details/120756834)，而RANSAC的部分則是以課堂上的講義為主。LLM的部分使用了GPT5和Gemini 2.5 (因為debug時GPT一直鬼打牆，我就又跑去問Gemini)，主要是用來做debug，也用來查詢o3d、scipy、cv2、numpy等一部分函數的功能跟範例。另外其實做到最後篩選出inliers後我不是很清楚再來要做什麼後，最後LSE的優化方式也是直接問LLM的scipy.optimize.least\_squares 的函式也是GPT推薦的。

那麼這之後會分成3個部分說明RANSAC + P3P的實作、error計算和visualization、以及2-2的AR渲染。

RANSAC + P3P 求解R、T矩陣的大致步驟如下sudo-code所示

Descriptors Matching and ratio test (這裡是直接搬助教作業1的code)

#RANSAC 實作

For iter = 100:

Sample 3 points

R, T = Solve p3p()

Calculate all project error by R, T #Project 3D point back to 2D point, and compare with ground truth 2D point

Determine inliers and num of inlier

Compare with max\_inliers, if more, replace max\_inliers

#LSE 最終優化

least\_squares(R\_max, T\_max) #Use the R, T matrix with maximun inliers as initial

Solve p3p():

c12, c13, c23, d1, d2, d3 = get cosine()

#Get cosine between 3 points (WCS) and their ray line (OP1, OP2, OP3)

Dis 12, Dis13, Dis23 = sqrt(np.sum((P – P)\*\*2)) #Get distance between 3 point

Solve quartic function by np.root() and remove complex and negative solutions

Solve a, b, c by np.root() as defined in lectures

By a, b, c, d1, d2, d3, we get 3D points in CCS, then we can solve and get RT by the transformation from CCS to WCS

Move the origin of both into the centroid of 3 points; thus, we remove the effect of translation, and calculate R by linear algebra

Use R to calculate T and return R, T

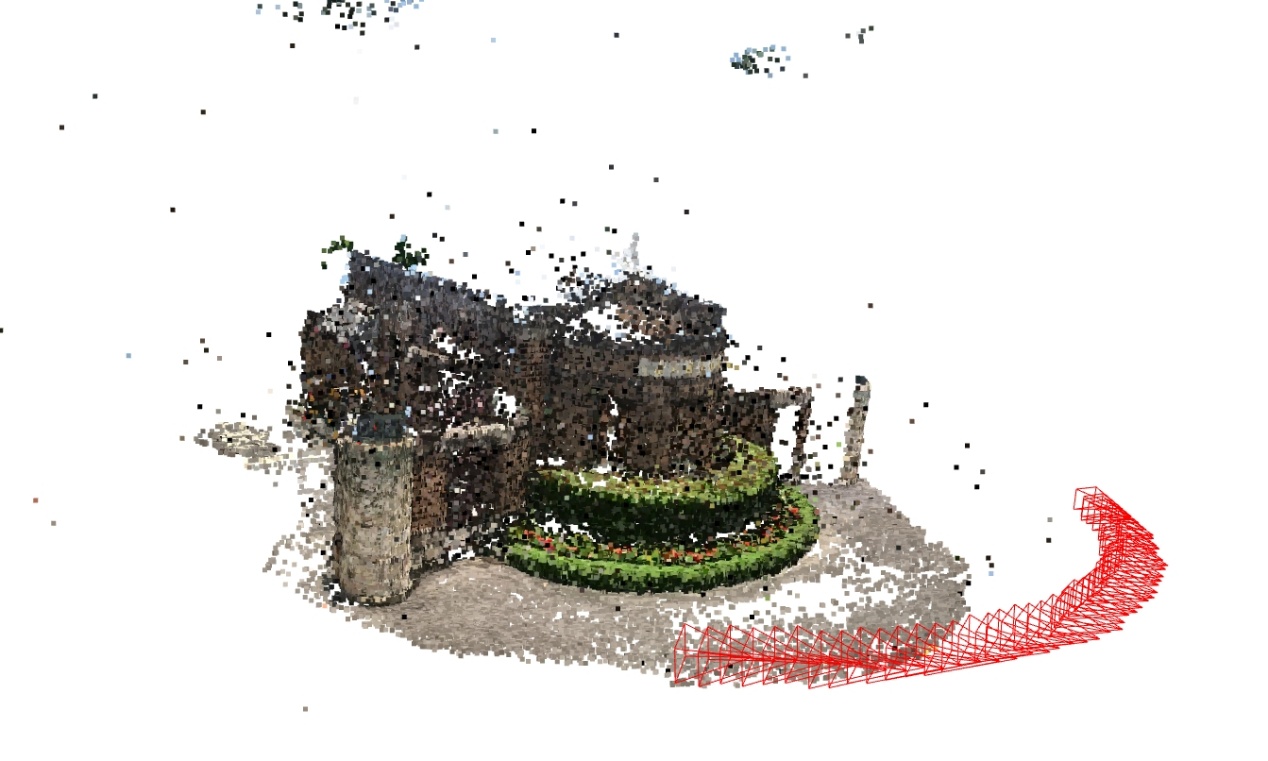
以上是大致流程，code內還有一點簡單的註解，大致上最複雜跟花時間的應該還是P3P的流程還有如何解中間的一大串方程式，另外我也查了不少資料確認camera matrix和distortion影響的時間點(運算是要先削去哪個還有project回去2D點時計算的順序)。另外我解參數y時 (算P3P內的a, b, c時)，我改用了sympy的package，這是LLM我用的，跟np.root()一樣是解方程用的function，主要是我debug時不知道為什麼那段一直出錯(可能我有什麼key錯了)，反正Gemini推薦我用後就跑過去了，所以我也就不改回np.root()了，另外就是他那個參數有點醜，比較難算。

實作時，是分別先跟cv2的solveP3P和solvePnPRansac兩個function對答案，先是確認P3P的演算法正確後，再確認RANSAC + LSE後的答案是否正確。另外因為有使用LSE做優化，這部分會跑得比較久一點，希望能加速的話可以試著拉高ration test(0.25)和RASANC error(0.35)的容許標標準，降低最終inliers的點對數減少LSE優化的計算時間

再來是error calculation和visualization，error比較麻煩的是旋轉誤差，位移誤差，直接算歐是距離就好，旋轉的話，因為四元數表示的(x, y, z, w)中w正好表示cosine，所以算出四元數後normalize再丟arccos()會比較方便，這邊我覺得角度值會直觀點，但因為cos(0) = 0，直接拿cos當誤差好像也行? 總之我的作法是先將兩個rvec轉成Rotation matrix，然後 ，轉成quaternion後如上所述計算誤差角度。

旋轉及位移誤差中位數

Visualization的部分是先以[0, 0, 0, 1], [0.15, 0.1, 0.4, 1], [-0.15, 0.1, 0.4, 1], [0.15, -0.1, 0.4, 1], [-0.15, -0.1, 0.4, 1]這五點構成的一個朝向+z軸的四角椎為目標(CCS)，反投影至WCS，而反投影矩陣就是，來計算這五點的WCS座標，最後用o3d畫出來。o3d function的使用主要是從助教transform cube的sample code搬，還有從作業slide提供o3d官網找方法。(也有問了下GPT點跟線的分別呈現方式)



相機位置可視化結果

最後是2-2的AR渲染，這部分和visualization也有點像，先準備好一個在WCS座標下的cube voxel，位置預設在 (0, 0, 0)到 (1, 1, 1)的立方區域，實際擺放位置、大小和旋轉，則是使用transform cube跑出來的(rotation, translation, scale)調整。放置完cube後計算到相機(-T)的距離，然後再進行排序。最後是每個voxel投影到相片上，實際的操作跟再RANSAC找inliers算誤差的動作是一樣的，需要額外注意的是在RT矩陣運算完後要先檢查CCS中Z座標是否大於0(位於相機前)，若沒有的話則不需在畫面中著色。再來是計算回2D座標，這邊則是檢查座標是否若相片的範圍內[(0, width), (0, height)]，若超出範圍的話也無需著色。

然後這邊提一個比較好笑的bug，一開始我是一個pixel對一個voxel的，結果因為點太小根本看不到。我找了老半天bug想說都怎麼cube沒畫出來，後來把image上畫的點調大(4\*4 pixel)就看得到了。