Homework 1 Report

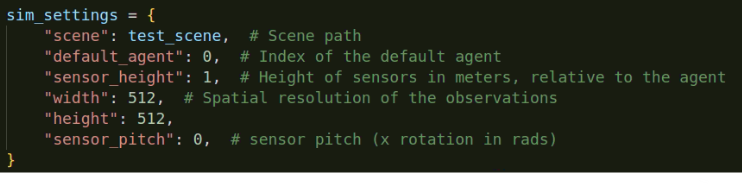
Name: 謝元碩

Student ID: 311512015

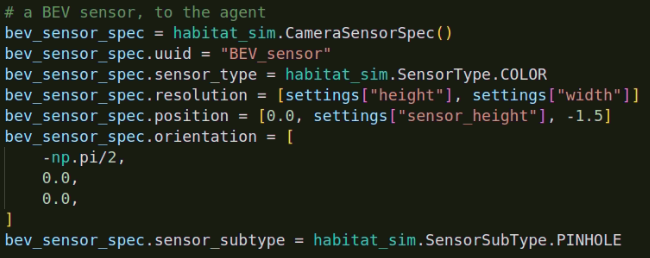
**Task1. BEV projection**

1. Code Design
2. load.py

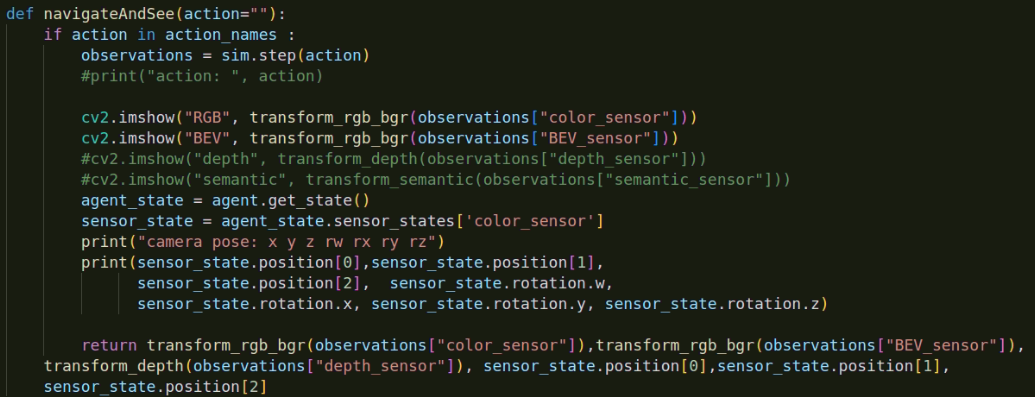
* **降低sensor高度以縮小誤差：**將sim\_settings的"sensor\_height"調整為1



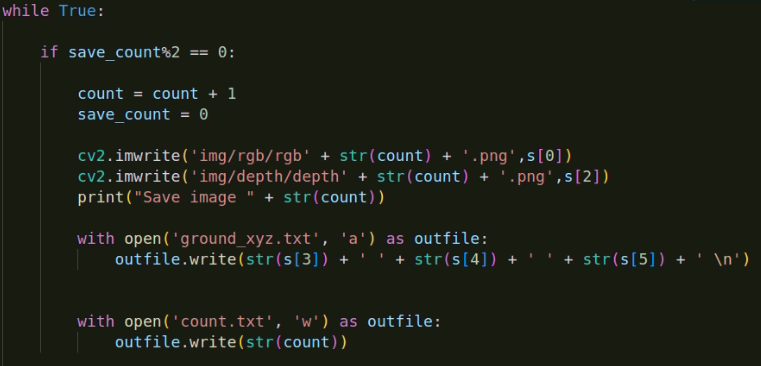
* **在make\_simple\_cfg函式中加入bev sensor：**將sensor orientation的roll項調整為順時針旋轉90度(俯視)，sensor position的最後一項改為-1.5(平移座標)，其餘sensor不變



* 在navigateAndSee函式中，加入顯示BEV視角的畫面。函式最後需**回傳front view image(color sensor)、BEV view image(BEV sensor)**，如此可在使用鍵盤的過程中儲存移動時觀測到的影像



* 加入鍵盤按鍵q和e的條件：按下q將儲存front view的圖像，按下e將儲存BEV view的圖像。s[0]即對應到navigateAndSee函式回傳的第1項變數(front view image)，s[1]同樣對應函式回傳的第2項變數(BEV view image)

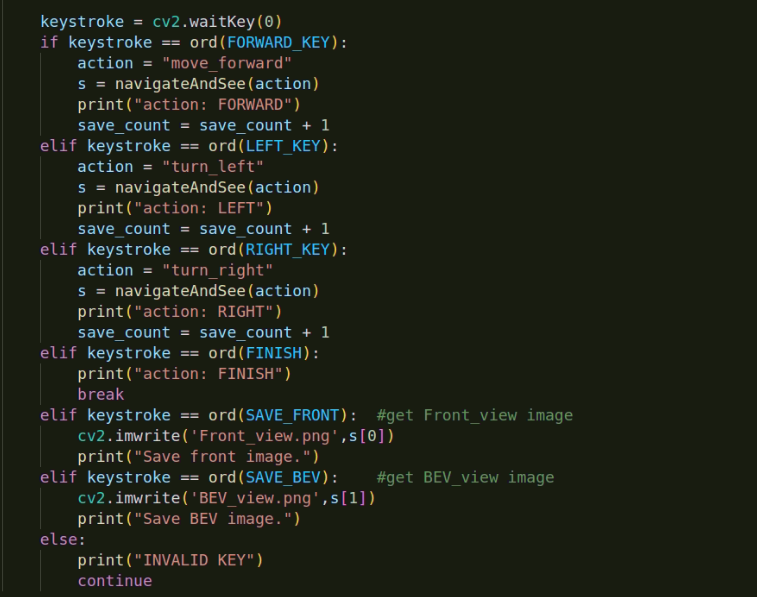


.

.

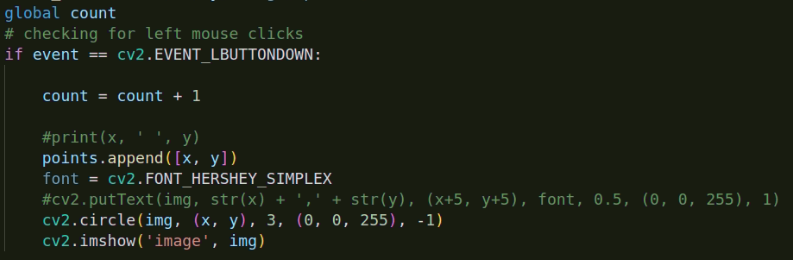
.

.



1. bev.py

* 在click\_event函式中，當滑鼠左鍵被觸發，將會累加按下的次數，同時記錄點座標



* 這裡將說明我如何完成top\_to\_front projection。在2D image與3D coordinate system的轉換中，使用到Pinhole Camera Model，公式及示意圖如下：

,

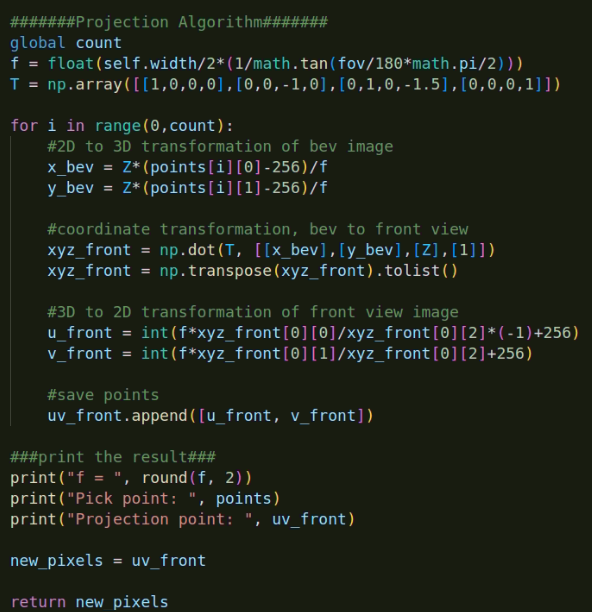


首先在2D轉3D，由於數值需配合上圖三角形比例關係，u、v需先扣掉width/2、height/2，始能換算x跟y，此步驟的z為BEV sensor的預設高度1；3D轉成2D的部分，同樣先把經轉移矩陣換算過的xyz帶入公式求出u、v，再將width/2、height/2加回來，成為front view image的座標，如此就能順利完成轉換。f為使用FOV公式求得，公式如下：

轉移矩陣的部分，我們需將bev sensor的坐標系轉換到front view sensor的坐標系。以自己定義的bev view及front view坐標系，平移只有z軸部分， 為-1.5，轉動為對x軸順時針轉90度，其所得轉移矩陣如下：

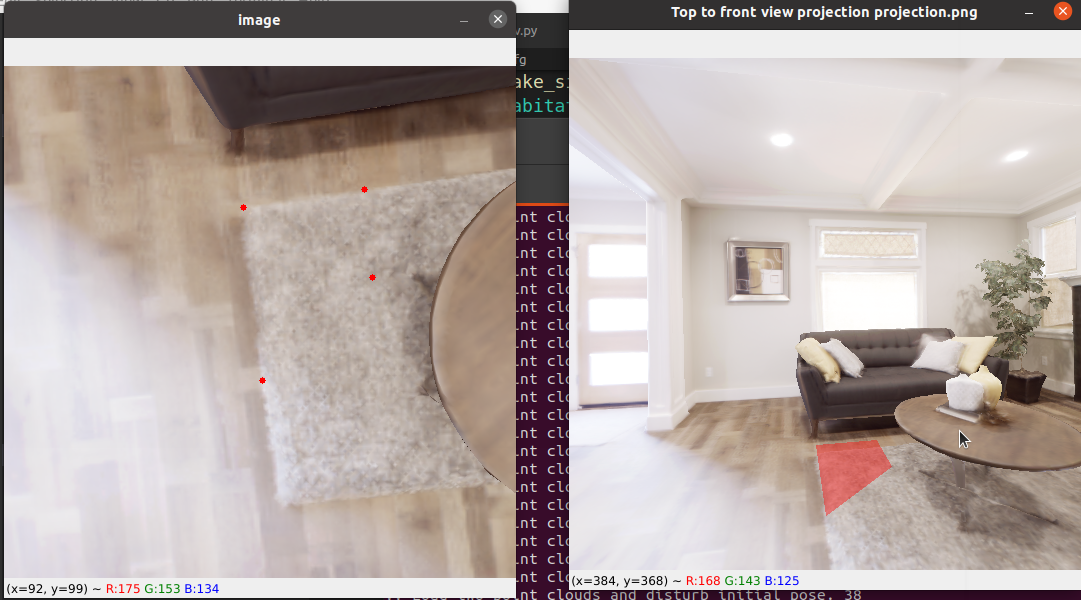
接著在程式中使用此轉移矩陣對bev座標做轉換，並轉成list型態以便取值做運算。

經上述三個步驟的轉換，即產生經projection後的新uv座標。只要將儲存坐標系的list回傳，bev projection即大功告成。程式碼如下：

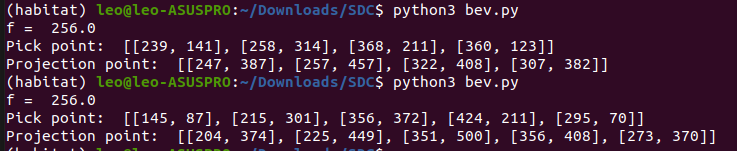


1. Result

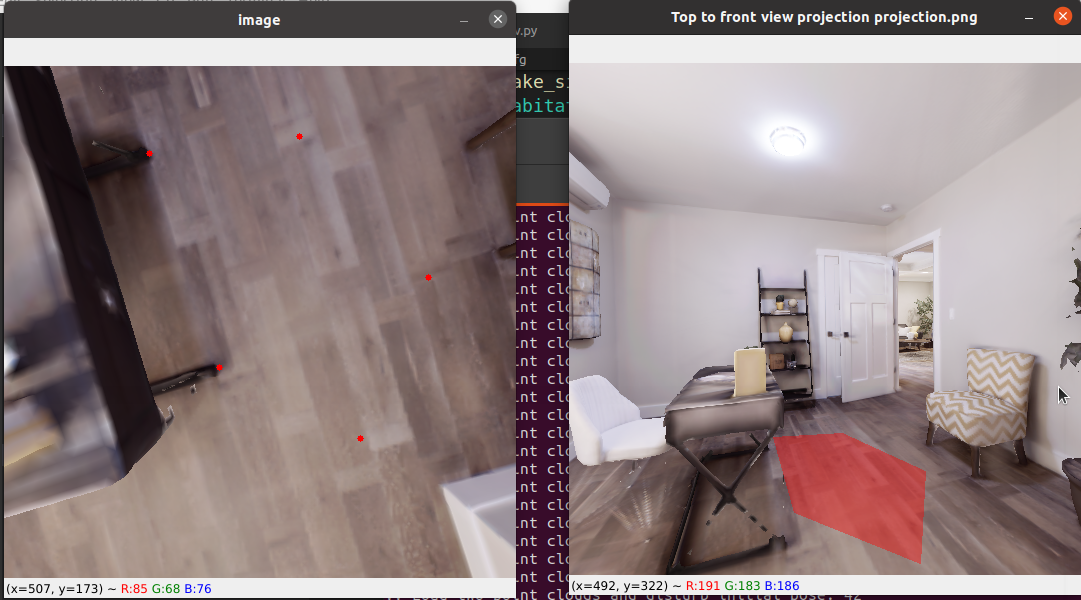
* Example 1:



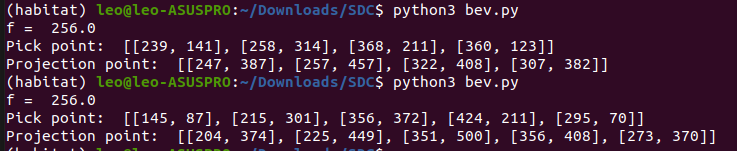
Output screen:



* Example 2:

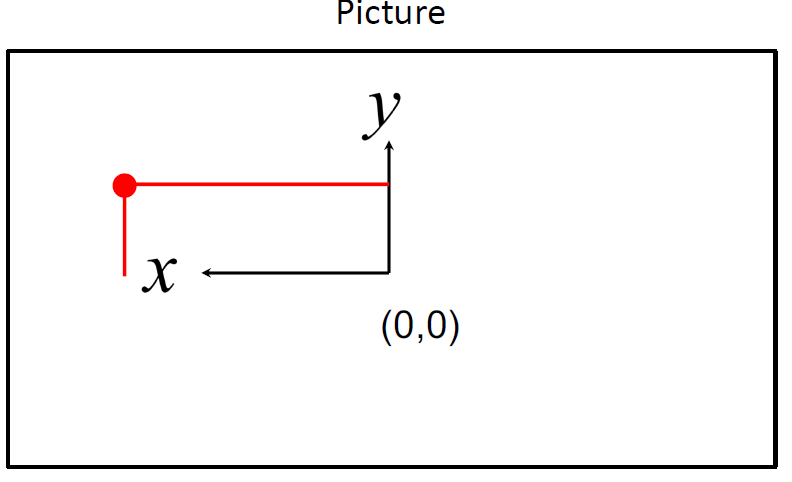


Output screen:



1. Discussion

思考演算法的過程中，我發現若相機中心與影像介面的坐標系設定不同(uv軸與xy軸的相對朝向)，2D與3D座標間的轉換容易導致xy軸方向有正負號的差別，如果沒注意到，將會使projection結果出錯。



+v

+u

在我定義的坐標系上做轉換時，當進行front view image的3D轉2D時，x軸換算中的Pinhole Camera Model部分需先多乘上一個-1，再去+256才會正確(如程式碼)，而這也是滿值得討論與深思的。

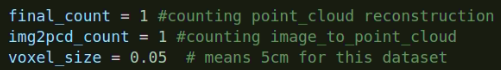
**Task2. 3D Scene Reconstruction**

1. Code Design
   1. load.py

* **儲存在模擬環境行走時所顯示的rgb image及depth image**：在這個task中，load.py最主要的功能，就是儲存所有行走過程看見的rgb及depth圖像，這些圖片檔將會用來進行3D重建。由於電腦性能限制，我設定為每走2步記錄一次，資料量才不會太過龐大。
* 另外也同時**記錄當下位置的卡式座標(xyz)**，以畫出ground truth trajectory
* 最後**更新儲存圖片的總次數**到count.txt，這個數值將會在reconstruct.py中讀入做使用



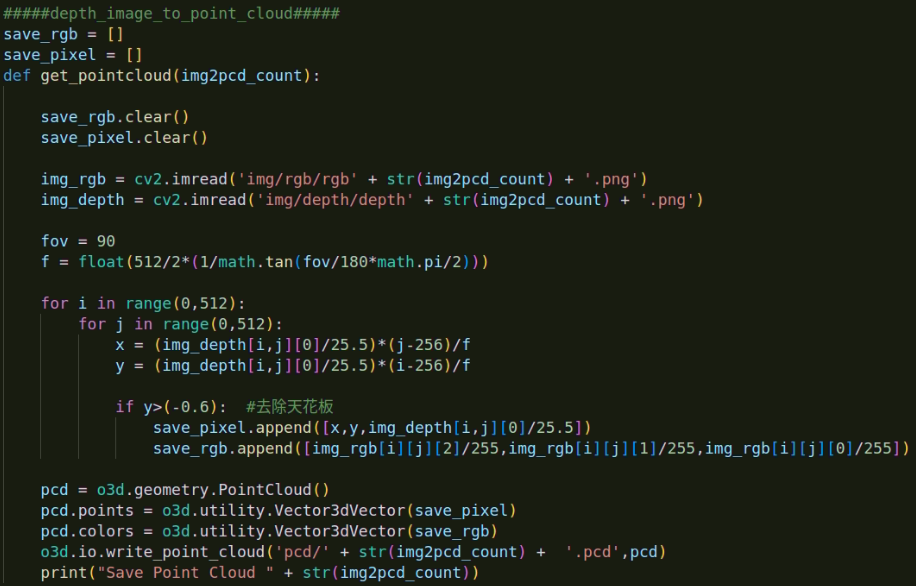
* 1. reconstruct.py
* 宣告將會用到的兩種count全域變數(註解如圖)



* 函式part 1：建立點雲

在get\_pointcloud函式中，會讀入兩張分別為rgb及depth資訊的圖片檔，利用圖檔的資訊，做2D轉3D的座標轉換，並將轉換後的卡式座標及rgb資料分別存到save\_pixel和save\_rgb。

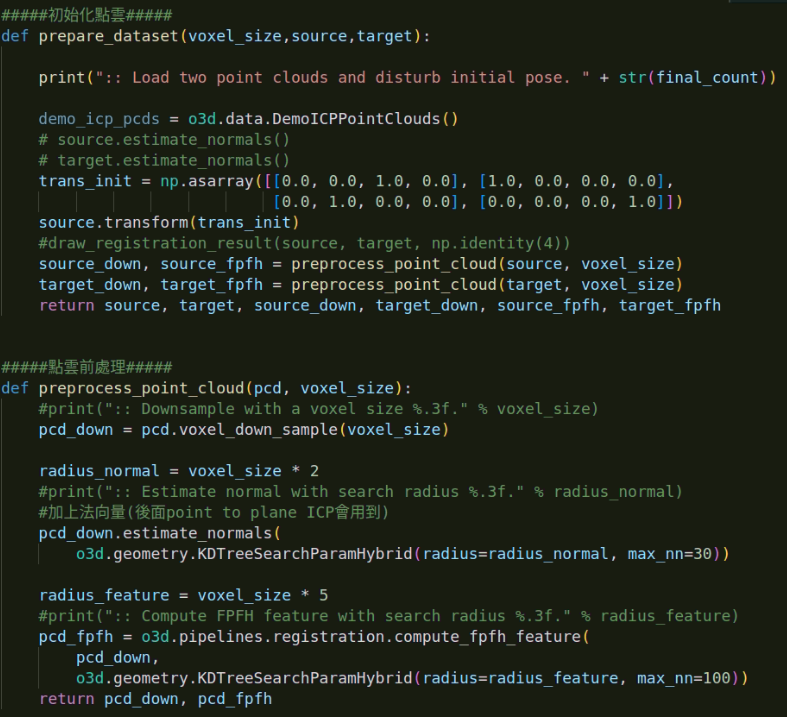
接著宣告一個PointCloud，把前述data存入點雲中，最後將點雲存成pcd檔。



* 函式part 2：點雲前置作業

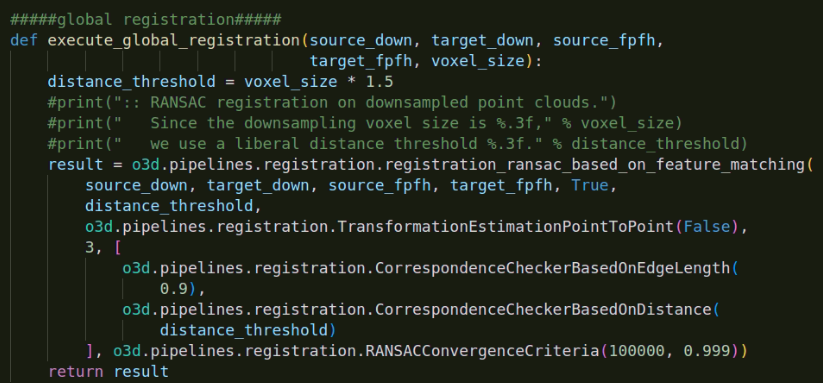
prepare\_dataset函式先對source點雲做初始座標轉換，接著將source PointCloud和target PoitCloud丟入preprocess\_point\_cloud函式，最後回傳初始點雲、降維後的點雲及經fpfh\_feature計算後的點雲

preprocess\_point\_cloud函式如前述所言，主要作點雲降維和fpfh特徵計算



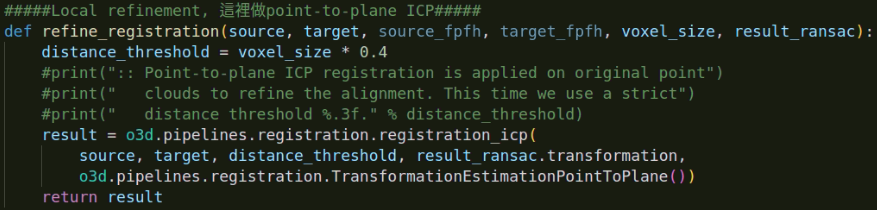
* 函式part3：執行ransac registration

此函式對source、target以feature matching的方法，進行點對點的ransac校正



* 函式part4：做open3d的ICP配準

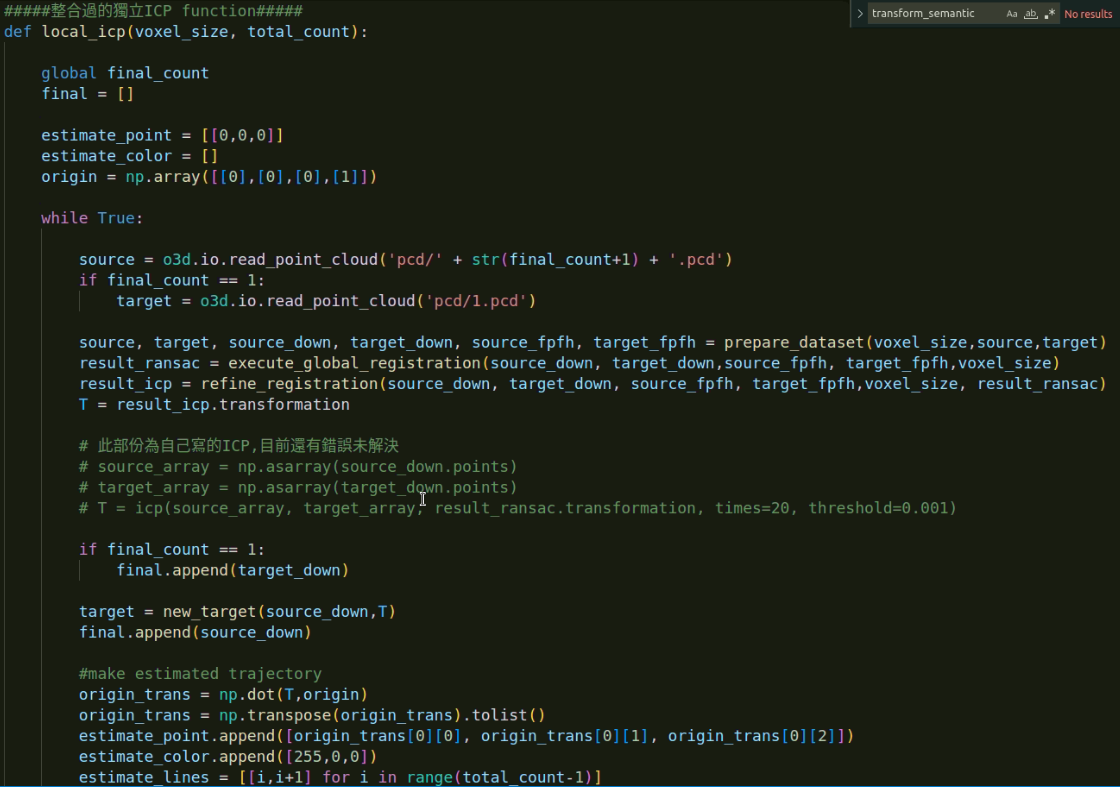
這裡使用open3d內建的ICP演算法，對兩份點雲做point to plane的配準

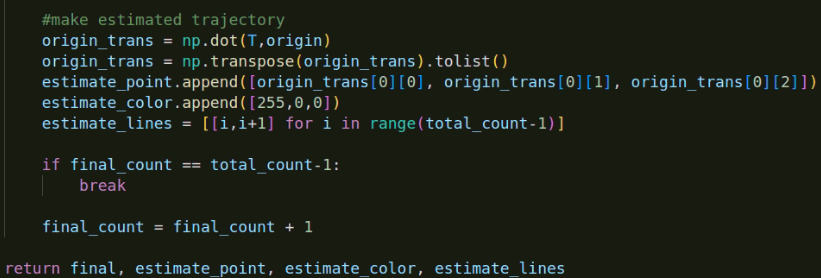


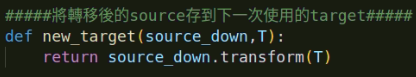
* 函式part5：大量點雲座標轉換

此函式整合前述part2~4的函式，以無窮迴圈不斷讀入新的點雲，每次皆使用ICP配準後得到轉移矩陣T，將source點雲轉移到target點雲的座標系，轉移後利用list存到final中，如此可得到完整的地圖點雲資料。每一次使用的target需為前一次轉移後的source資料，這樣地圖才會保持完整。

另外，我將estimated trajectory的計算也放在這裡。estimated trajectory的計算方式，同樣使用ICP配準產生的轉移矩陣，對每一個點雲的「原點」做轉換，並一一存到list中。透過ICP產生出來的座標，即代表估測的座標。

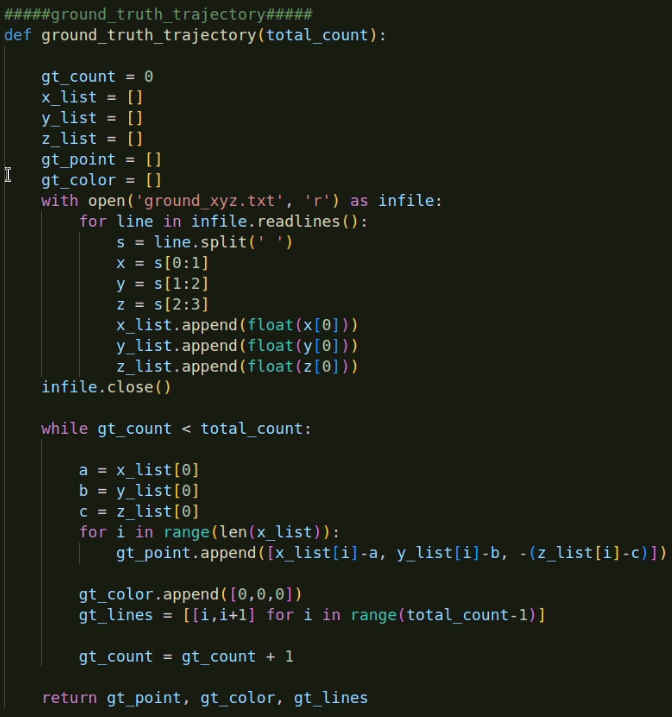






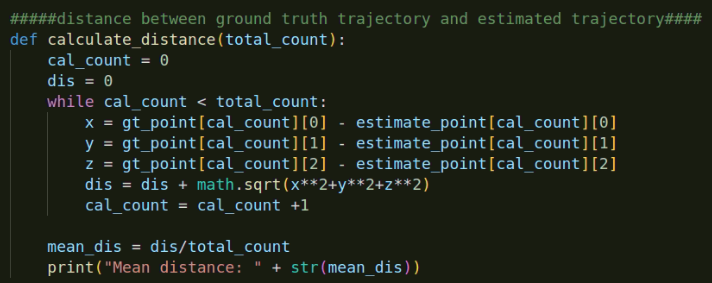
* 函式part6：ground truth trajectory

此函式直接將load.py中所存的每組xyz座標存入list，以用來生成路徑



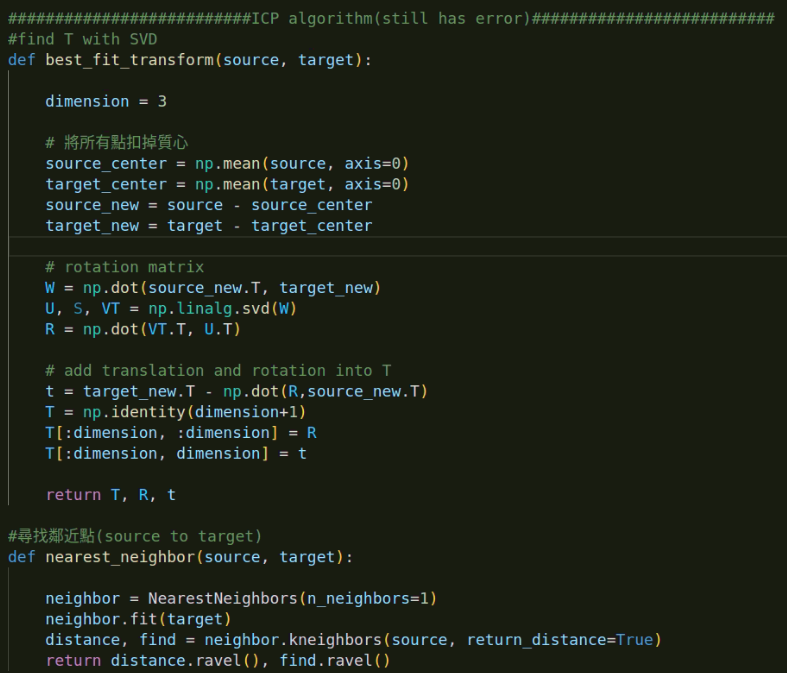
* 函式part7：計算estimated trajectory與ground truth trajectory路徑平均誤差

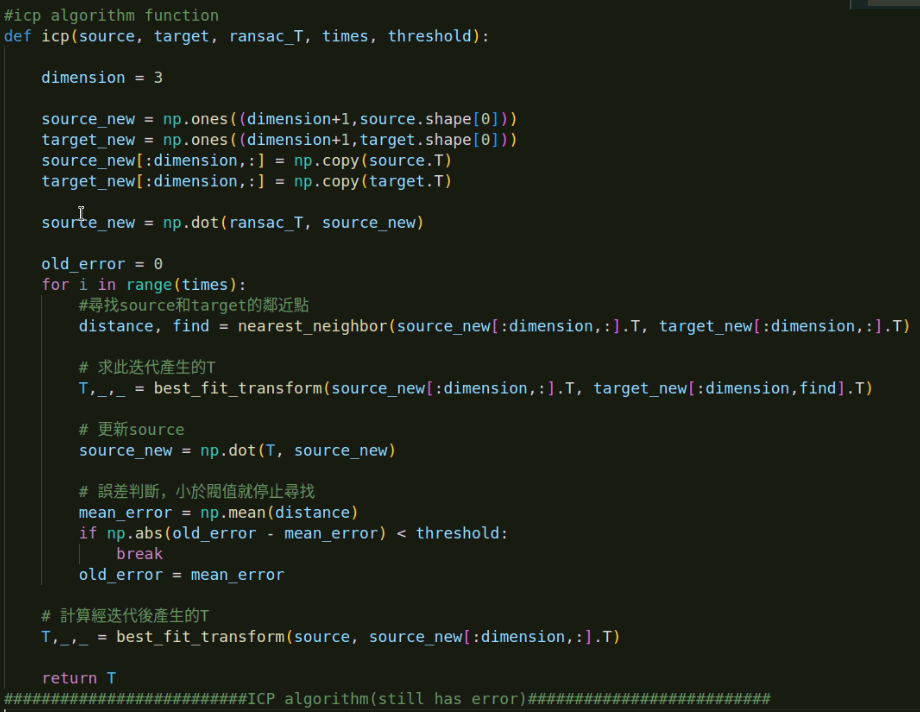
此函式將上述兩種路徑的每個對應點取歐式距離後，取平均值



* 函式part8：自己寫的ICP(未完成)

依照講義的說明及步驟，試著寫出ICP演算法的雛型，目前有三組函式：使用SVD計算轉移矩陣、尋找鄰近點和ICP演算法。但目前執行結果仍是失敗的。



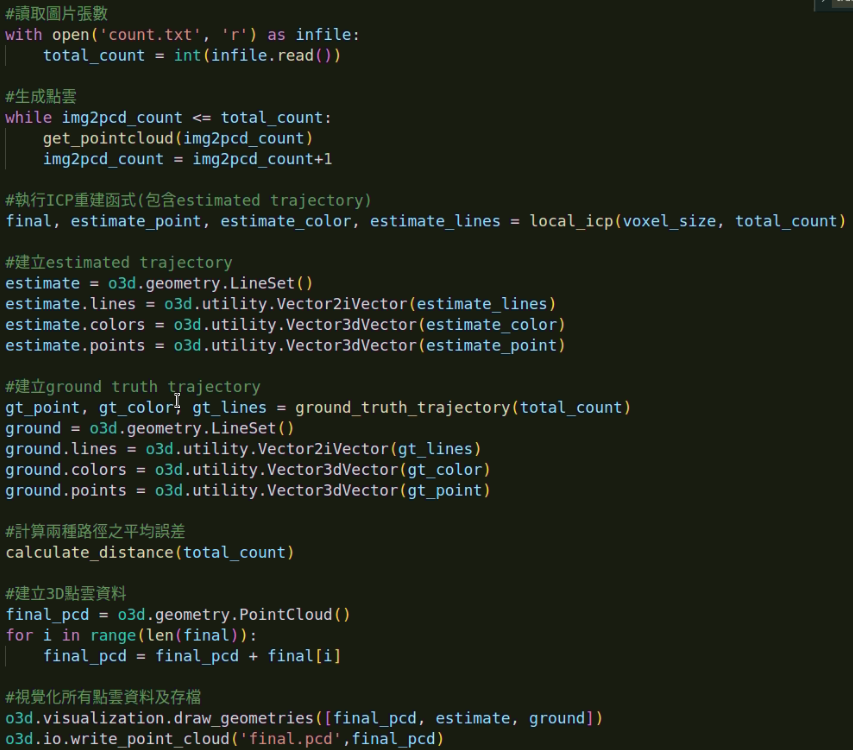


* 主函式

在主函式中，首先將所有圖檔轉成點雲資料，接著進行ICP配準及重建，此時會回傳轉移過後的所有點雲資料final，以及estimated point。

後續會執行ground truth trajectory的函式以取得移動座標，並將兩種路徑宣告成LineSet格式，再計算平均誤差並輸出在終端機中。

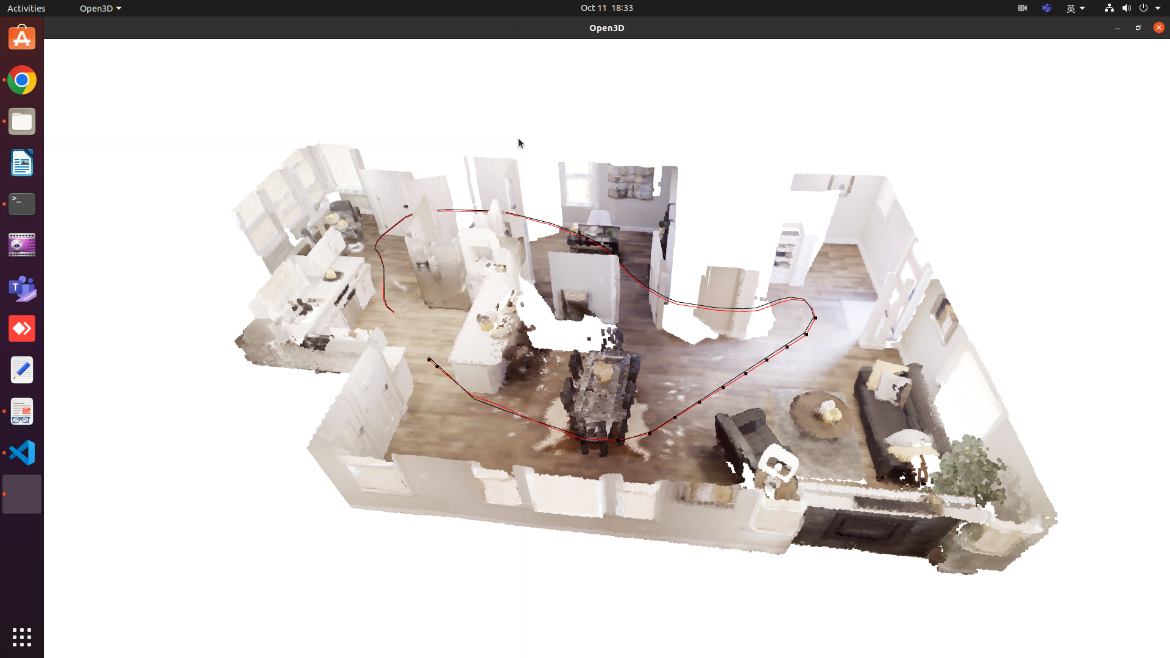
最後宣告點雲、把資料丟入並視覺化，完整的3D地圖、兩種路徑即大功告成。



1. Result

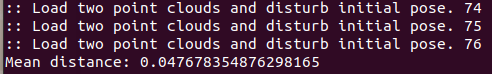
* Floor 1





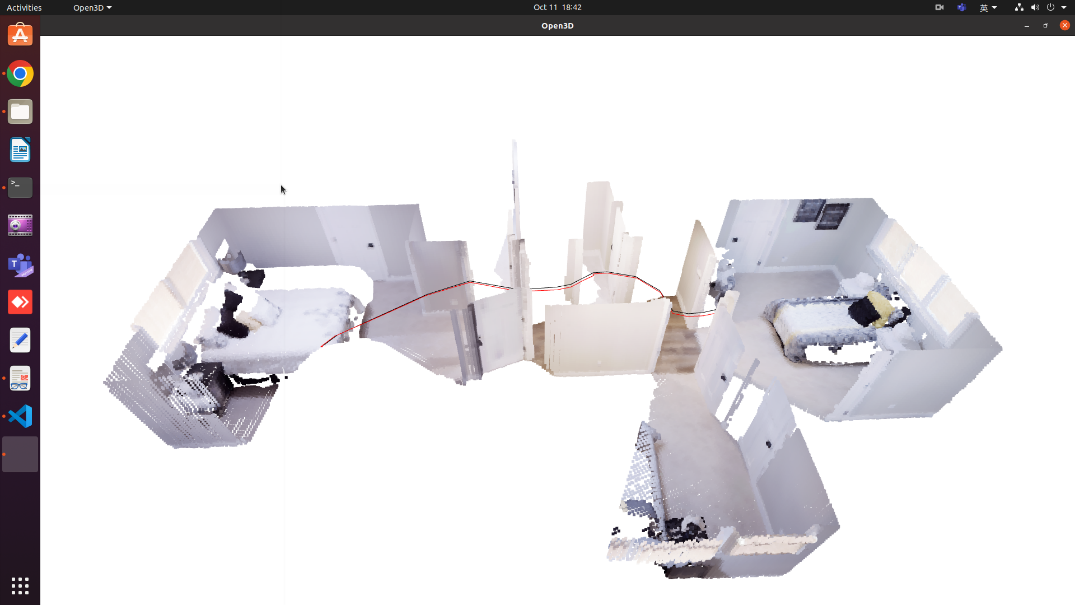
Output screen:

Include Mean L2 distance between ground truth and estimated trajectory



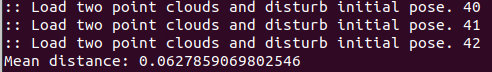
* Floor 2(因旋轉太多易重建失敗，故data size獲取較少)





Output screen:

Include Mean L2 distance between ground truth and estimated trajectory



1. Discussion

重建二樓時，由於二樓空間皆為單一出入口，行走時皆須旋轉較大的角度才能離開房間，多次且大量的旋轉，導致建圖失敗率極高，代表經過ICP得到的轉移矩陣在此狀況易出錯，如何去改善針對此狀況的ICP是一個很好的議題。

ground truth and estimated trajectory兩種路徑事實上並非重疊，不同的配準演算法會影響路徑的正確率。行走過程若產生誤差，似乎會發生誤差越來越大的狀況，無法再次收斂，導致路徑平均誤差增大。如何讓規畫路徑具強健性，值得討論。