COLMAP 與 ORB-SLAM3 比較

壹、實驗目的：使用 COLMAP 作為 ground truth，試與 ORB-SLAM3 比較視覺定位的效果

貳、環境與測試影片

一、軟體環境：

COLMAP: COLMAP 3.6 GPU version(run on Windows10, GPU: GTX1650)

ORB-SLAM: ORB-SLAM3, OpenCV3.2.0 (run on Ubuntu 20.04)

二、拍攝設備：iphone 11 HD 30fps 橫拍(1080p, 原始影像：1920*1080)

三、拍攝環境與測試影片

選用家中客房，將房間整理成路線可成環狀，且四周在相似的牆面附近擺上一些物品來增大找到特徵點的機會。

由於 COLMAP 建立在他自己定義的世界坐標系上（與建模的照片有關），因此為了在分析時有良好的辨識度，將手機腳架放在辦公椅上推動，而移動幅度、轉彎盡量慢（與 ORB-SLAM 辨識有關），但整體不可拖太久（與 COLMAP 處理時間有關），所以可以造出一個在高度水平面幾乎沒有變動的平面軌道，同時房間有磁磚，便於測量總路長（一圈約 10.715 公尺）。

Fig.1, 2 上視圖（左）與斜視圖（右）

四、影片畫格處理方式

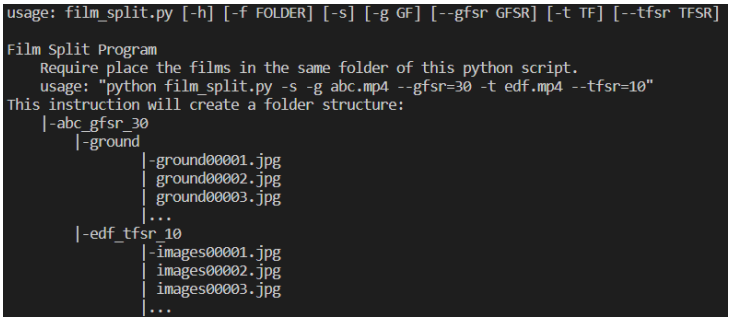
由於使用 iphone 進行拍攝，而產生的影片格式是.MOV 檔影片，windows 上的 OpenCV 不能處理該檔案(與.mov 不同)，因此考慮各式程式轉檔也可能壓縮內容，造成失真。最後選擇將所有影片接上傳到 youtube，待 youtube 處理後，再下載回來，此時即是.mp4 檔。而規格則是

(1280*720)，之後在影片分割時，再將影片壓縮至(640*360，長寬減半)。

環境影片長 2:17，由於環境影片僅是供後續跑測試影片時，能有更好的參考，而只需各角落有些特徵點即可，因此拍攝影片有繞到房間各位置、不同方向，而切畫格時，以每 20 幀存一格的方式(2/3fps)，切割後共有 205 張照片。

測試影片長約 78 秒，與環境影片相比，由於需要更細緻的相機軌道，以每 5 幀存一格的方式(6fps)，切割後共有 468 張照片。

此後在 windows 上使用 film_split.py，進行 colmap 的照片資料夾切割，當選定 image 資料夾命名（若沒設定則與環境影片名稱一樣），環境影片檔案、環境影片切割頻率(take 1 per 20 frame)、測試影片檔案、測試影片切割頻率(take 1 per 5 frame)後，即會開始建立 colmap 要設定 image 資料夾，裡面放 ground 資料夾（存放環境影片畫格，檔名規格：groundxxxxx.jpg）與測試影片資料夾（檔名規格：imagesxxxxx.jpg）。在建立模型前，先手動將測試影片資料夾移出 image 資料夾，待模型建好後，再拉進去 image 資料夾，即可繼續進行 feature extract, matching, reconstruction。



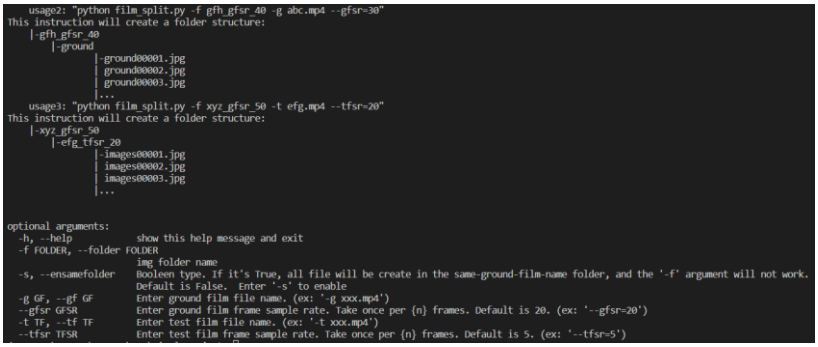


Fig.3, 4：film_split.py 使用方式與資料夾架構

而在處理 ORB-SLAM 資料夾時，使用 orb_file_split_v2.py，進行切割，並創建 rgb 資料夾（存測試影片畫格），rgb.txt（畫格資訊：timestamp、位置）。Timestamp 採用從影片開頭為 0 開始計數，因此第一個存到的畫格 timestamp 為 5*1/30fps=0.166667 秒（統一四捨五入取後 6 位）

Fig.5,6：orb_file_split_v2.py 使用方式與 rgb.txt 規格

五、相機校正

Colmap 內建數種相機模型，可以從照片中算出相機參數，因此不須先給校正參數。

使用 ORB-SLAM 則先對相機校正，因此使用 9*6 的棋盤格，拍攝影片後，一樣傳到 youtube，再下載分割，以每 20 幀方式取到不同角度(640*360)的相片。再使用之前 lab4 所作的相機內參提取程式 lab_4_1.py，並將取到的 fx, fy, cx ,cy, k1, k2, p1, p2, k3 更改在 TUMX5.yaml 裡。另外，該組態描述的 Camera.fps 調整至 6.0，Camera.width, Camera.height 調整到 640, 360。

六、匯入方式

COLMAP 使用預設的 SIMPLE RADIAL 相機模型。相機參數則設定 Shared per sub-folder，之後運算時，如果當初拍攝畫面平滑，則可以得到更好的結果，同時 reconstruction 的速度也會快很多。Matching 採用預設 Exhaustive 方式，雖然較耗時，但在數量數百張的情況下，可以有最完整的配對，且使用 GPU 版本時，共 205+468 張照片，feature extract, matching 也僅不到 1 分鐘就結束了，reconstruction 的時間也用不到 20 分鐘左右。

ORB-SLAM 則須將使用官方提供的 ORBvoc.txt，其餘則是自行準備的.yaml 組態敘述檔、照片集(with rgb.txt)。其中.yaml 除了相機參數、照片大小、影像分辨率(6fps)外，都使用預設數值(與 TUM1.yaml 那些檔案一樣)

參、測量方式

一、輸出格式

Colmap 輸出檔為.nvm 檔，會將所有能夠收斂到一個 model 的照片記錄下來，同時記錄格式包含照片檔名、從模型的世界坐標系轉至該照片視角坐標系的 4 元素、3 維平移向量。而 ORB-SLAM 的 KeyFrameTrajectory.txt 則是僅記錄在路徑過程中，變化足夠大的 KeyFrame，記下該 KeyFrame 在

rgb.txt 檔對應的 timestamp、以及以 KeyFrame 第一張（通常是照片集第一幀）的相機坐標系表示的四元素與 3 維位置。

讀檔時，因為 Colamp 幾乎每張照片都可以記錄到，但 ORB-SLAM 部會每張都記錄，所以統一轉以 timestamp 依序排列，因此之後比對某時刻的位置時，以 ORB-SLAM 的 keyframe 位置與同時刻的 colmap 位置做比較，如此設計是可以有助於解決 Colmap、ORB-SLAM 使用不同 fps 的照片集（由於 ORB-SLAM 具 real time 的能力，適合使用高 fps，有規劃不同 fps 的照片集時，利用在 Colmap 的路徑上進行內插值的方式，獲取沒有兩照片集沒有同一幀的問題，不過由於時間問題，程式這部分並沒有完成。）

二、轉換座標空間

由於 ORB-SLAM 的座標是基於第一個 KeyFrame（第一幀）的相機坐標系，因此當要轉換成 Colmap 的世界坐標系時，需考慮下式：

$$X_c = R_{3\times3} M X_w + T$$

其中X_c為 ORB-SLAM 坐標系，X_w是 colmap 坐標系，$R_{3\times3}$為旋轉矩陣，M為拉伸矩陣，是T平移向量。

１、M拉伸矩陣

不同坐標系的軸單位會不一樣，因此 M 矩陣以 ORB-SLAM 記錄到的兩兩 KeyFrame 在三維空間的絕對距離，與對應到在 Colmap 的兩點絕對距離做比值分析。又三軸的縮放一致，因此會是一個單位矩陣乘一個常數。測試時，由於前段做為基準會更精確一點，中後段可能遇到失敗的 loop closing，因此距離會跑掉。最後選用前 15%的 frame 路徑進行推估平均，大概是 4.508433，std=0.544（而就算拉高至 100%，該常數測出來也落在 4.8 左右）

２、$R_{3\times3}$旋轉矩陣與T平移向量

以 Colmap 的第一幀畫面就可以找到從世界坐標系轉至相機坐標系的旋轉四元數，透過安裝以及 import scipy.spatial.transform 的 Rotation class 就可以透過該 class 進行(x,y,z,w)四元數轉成$R_{3\times3}$旋轉矩陣，T平移向量也是套用第一幀的平移向量。接著把所有 ORB-SLAM 的 Keyframe 位置 apply 上面的 M、R，再加上 T，即可將各點轉至世界座標上。

轉換軌跡時，讓 Colmap、ORB-SLAM 的第一幀從同一點開始，後續則透過透過上式進行三維座標變換。

三、分析方式

１、3D 繪圖

首先利用 matplotlib 套件，即可畫出 3D 視圖（如下 Fig.7 與其圖例），而透過引入 matplotlib.animation 類別可以讓該圖顯示出在兩條軌跡的對應畫格畫格位置（綠色菱形點）

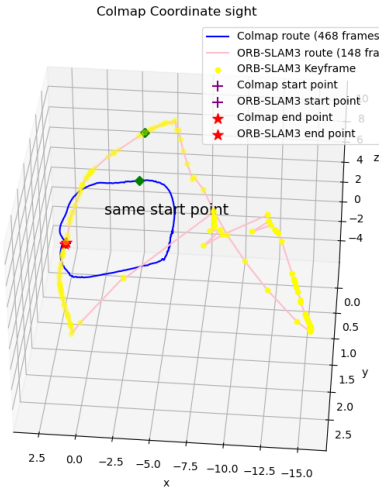


Fig.7 3D 視圖

２、各點位置誤差

由於當照片集是同樣 6fps 時，ORB-SLAM 的紀錄點必然少於 Colmap 的，因此比較誤差時僅比較 ORB-SLAM 紀錄的 KeyFrame 轉換位置與該時刻 Colmap 的相機位置，此部分也可以利用 matplot 套件來做圖表展示。

３、起點-終點誤差、真實距離尺寸

由於是環形軌跡，直接取第一幀與最後一幀的位置比較，而同樣是因為照片集切分方式一樣，ORB-SLAM 的最後紀錄點跟 Colmap 最後一點的 timestamp 是一樣的，因此可以直接找到對應的 Frame 紀錄比較位置，但真實情況下很難保持讓起點與終點位置相同，所以也計算 Colmap 的起點到終點誤差，與 ORB-SLAM 的起末誤差一起呈現。

另外比較距離時需要計算走多遠會產生多大的誤差，因為本次實驗假定 Colmap 是 ground truth，當量測真實軌跡路線長(10.175m)後，同時比較 Colmap 的路徑長，即可得到 Colmap 坐標系的軸單位與真實世界的軸單位轉換關係（約 0.572(meter/colmap unit)）。同理，可以將 Colmap 坐標系上的各距離誤差換算成真實世界的誤差。

肆、測量結果

※下列結果、圖表、動畫皆使用 process.py 分析，該檔內有區塊的註解

真實總路徑長：10.175 meter

Colmap 總路徑長： 17.797090(colmap unit)

ORB-SLAM 總路徑長：40.362571(colmap unit)

Colmap 轉至真實世界的單位轉換：0.571722 (meter/colmap unit)

ORB-SLAM 最大誤差：14.319815(colmap unit)|| 8.186963(meter)

ORB-SLAM 最大誤差發生幀數：285

ORB-SLAM 最大誤差已走路徑長：10.018348(colmap unit)||5.722771(meter)

Colmap 頭尾誤差：0.032701(colmap unit)|| 0.018696(meter)

ORB-SLAM 頭尾誤差：0.057560(colmap unit)|| 0.195493(meter)

→ 由三角形第三邊(起點共點，但終點異點)可以推算

→ 每 10.175m 約略誤差 0.018696+0.195493m

→ 實走每 100m，誤差 2.107646m

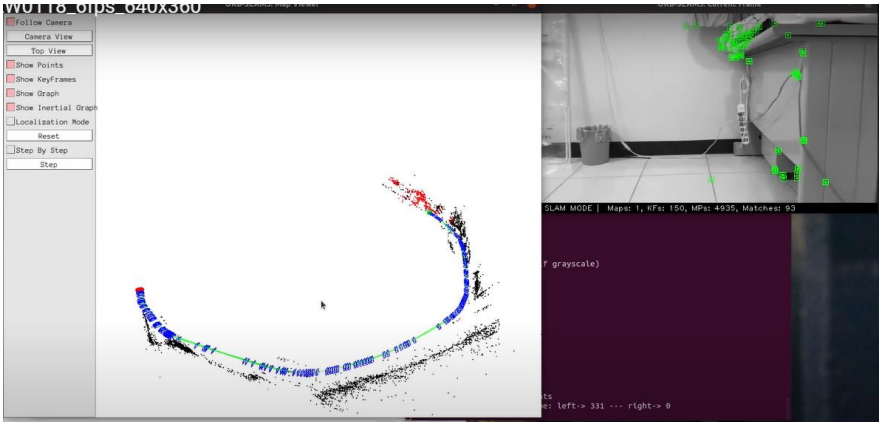


Fig.8: ORB-SLAM loopclosing 前的路徑

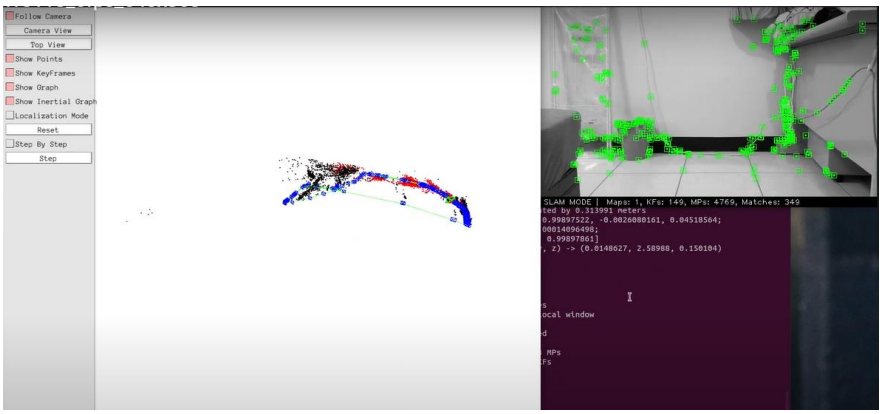


Fig.9: ORB-SLAM loopclosing 後的路徑

伍、分析評估與討論

一、COLMAP 對環境要求

由於 COLMAP 需作為 ground truth，因此 COLMAP 處理測試影片前，需要先對測試環境先建模，而 COLMAP 由於會將所有影像進行 Global BA，所以當影像數目增加，所花費的處理時間越久。經過嘗試多次後發現，環境尺度盡量不要太大，且光亮程度不可有太大差異（如有一面曝光，另一面陰暗；磁磚、玻璃、金屬反射強光），前者會使照片量增大而運算時間變長（曾給過 2000 多張照片，Global BA 的參數到後面都是幾十萬起跳，光是 reconstruction 的過程就花費超過 12 小時），後者則是會使 COLMAP 在擬合的時候，無法收斂，導致同一個場地被分割成不同 model，而且在 GUI 介面上也無法合併不同 model。

二、ORB-SLAM3 對環境要求

ORB-SLAM3 對環境的要求是相機校正需要特別準確，否則特別難找到特徵點，因此才會選擇同樣將.MOV 影片（環境、測試、校正 3 段影片）上傳到 youtube 再下載下來，保證過程是一樣的壓縮方式，避免特徵找不到。

此外影像需要特徵點夠多，原先拍攝環境有一面是單純白牆，並沒有任何參考物件，每當鏡頭轉過去的時候就會無法繼續定位，除非特徵點又夠多，才會開始記錄第 2 個 map。

最後，畫面需要十分穩定，也要盡量避免直接面向轉彎。前者是因為畫格變化太大的時候，找到的對應特徵點可能不夠使 ORB-SLAM 認知到是往哪邊移動、移動多少，因此很容易直接把地圖中段掉；後者則在下方過程分析會再提到。

三、過程分析

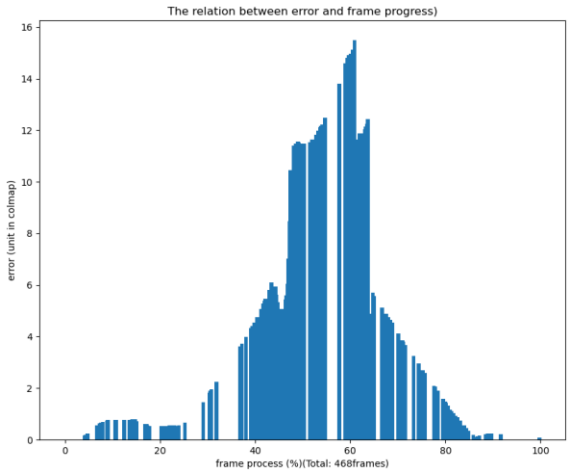


Chart.1 ORB-SLAM 誤差(colmap unit)對錄影過程(%)關係圖

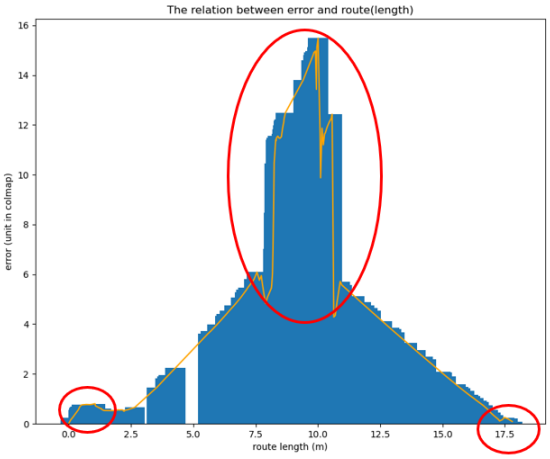


Chart.2 ORB-SLAM 誤差(colmap unit)對路徑長(colmap unit)關係圖
（長條圖與橘線描繪的是同一個資料點，但都畫出來才可好跟 chart1 評比）

上方兩圖表，原先是為了屏除行進速度快慢變換造成的誤差分析，而分開繪出，但其實兩圖表行為類似，因此可以知道大致直線行進時，前行速度約略一致。惟紅圈位置在誤差變化比較大，該區塊都是轉彎處所致。所以推測 ORB-SLAM 對畫面旋轉抗性極低，在跑 ORB-SLAM 時，看到旋轉的軌跡畫面也比較難轉相應的角度、距離，某些情況下甚至會將旋轉視為偏一側的直行路線。

而中央誤差變大又縮小是因為 ORB-SLAM 具有 loopclosing 功能，當看到相同的 keyframe 時，就會想辦法把路徑繞回來，然而先前旋轉的累積誤差使路線已經跑篇很多了，硬要擬合時，反而讓中途過程變壞了，所以起點確實與終點靠近，但過程被旋變換到看不出是一個平面近矩形軌跡。

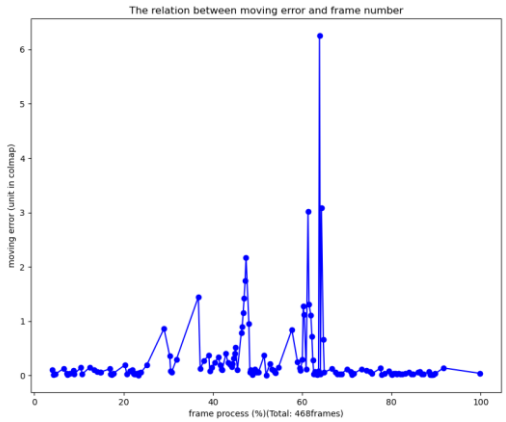


Chart.3 ORB-SLAM keyframe 移動誤差對 frame 進度關係

上圖是比較 ORB-SLAM 兩兩相鄰的 keyframe 距離與 ground truth 的距離，與 Chart1 一同比較，會發現大部分時間其實移動誤差接近 0。

可以看到：

1. 由於在跑 orb 時，就明顯看到最後結果被 loopclosing 搞壞，因此中段出現鋸齒高峰即是因為強制 loopclosing 後，中段的位置算不出好

的結果。

2. 因為 orb 僅有 local BA，因此在 Chat1 中可以看到前段的位置誤差基本上是上升狀，Loopclosing 則造成後段位置誤差逐漸收斂回起點。

綜合上兩點，可以知道相近 frame 的 local BA 基本上可以使移動時前行差小，但在遭遇轉彎時，以及錯誤的 loopclosing 收斂時，就可能讓位置突然誤差變大。

陸、補充實驗

一、OBR-SLAM 影片 fps

將測試影片切成原始 30fps 的相片集，在跑過 OBR-SLAM 後，一樣是壞掉的 Loopclosing。

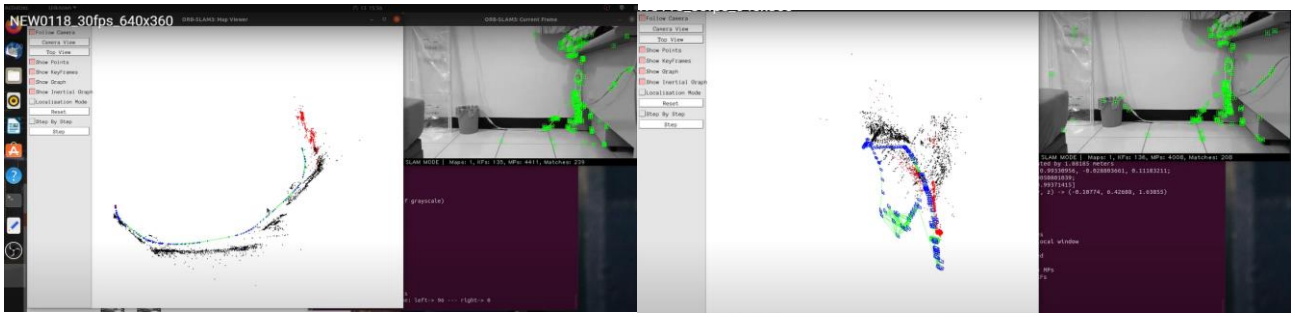


Fig.10, 11: ORB-SLAM loopclosing 前、後的路徑(相片集 30fps)

二、錯誤的 LoopClosing（此區塊使用另一個測試影片，檔名是 NEW0126.mp4）

下方案例是切在 6fps 的照片集。實測時，似乎在當 ORB-SLAM 在自己的坐標系跑太遠的時候，很可能最後無法做出 LoopClosing

Fig.12: ORB-SLAM no loopclosing (6fps)

然而當切在 30fps 時，因為中途中段找不到定位，重新建地圖時，又從起點開始建，由於離起點夠近，loopclosing 直接把 map2 路徑接回起點。

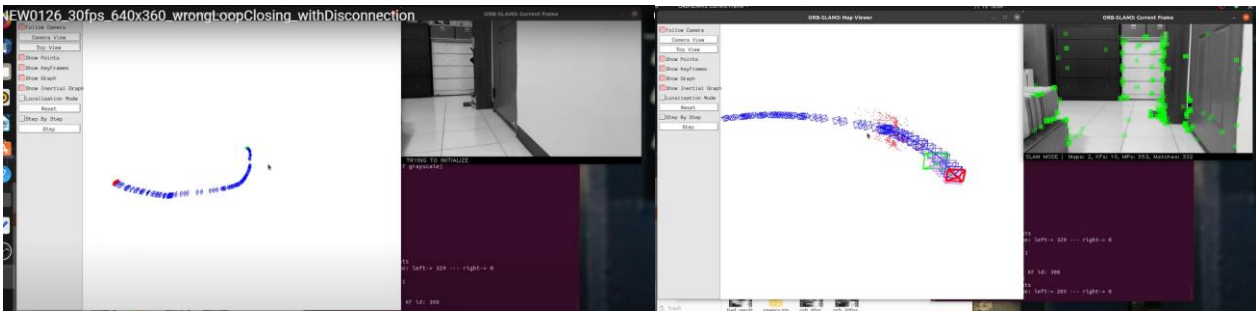


Fig.13, 14: 中段斷掉(left)、從起點開始建 map2(right)

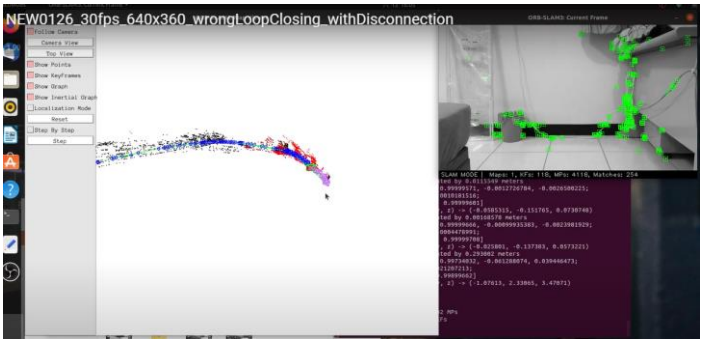


Fig.15: map2 在 LoopClosing 後接回起點

總結以上案例，大概可以歸納：當中途地圖中段以後之後，很容易創造出壞掉的 LoopClosing。另外劇烈的直接旋轉，會導致找不到定位，或是在 ORB-SLAM 的座標轉偏，而走越遠的話，則會可能無法順利 LoopClosing。然而上課看到的車子跑 ORB-SLAM 畫面 LoopClosing 效果不錯，推測可能是轉彎轉的幅度恰當，且座標移動尺度在還可以接受的範圍內。

柒、影片連結

1. 環境錄影：https://www.youtube.com/watch?v=5p-bXkvmFvo
2. 測試影片：https://www.youtube.com/watch?v=N02pnGMyZsE
3. COLAMP 環境建模結果：https://youtu.be/CyhykcG0l9U
4. COLAMP 測試影片結果(6fps)：https://youtu.be/54BYpND5cqE
5. ORB-SLAM 測試影片結果(6fps)（報告以此討論）：
   https://www.youtube.com/watch?v=fHBsFEefKu0
6. ORB-SLAM 補充測試
   Fig.10, 11 使用 COLAMP 測試影片結果(30fps)：
   https://www.youtube.com/watch?v=vKes7s5o3_s
   下列測試影片 NEW0126：https://www.youtube.com/watch?v=gRIjOvpyvAQ
   Fig.12: https://youtu.be/8EUxDqm4Fho?t=100
   Fig.13(1:13), 14(1:21), 15(1:42):
   https://www.youtube.com/watch?v=4pyvHYIfa-I
7. ORB-SLAM 旋轉壞掉的例子：https://youtu.be/lQZ1f0IJYH8?t=38