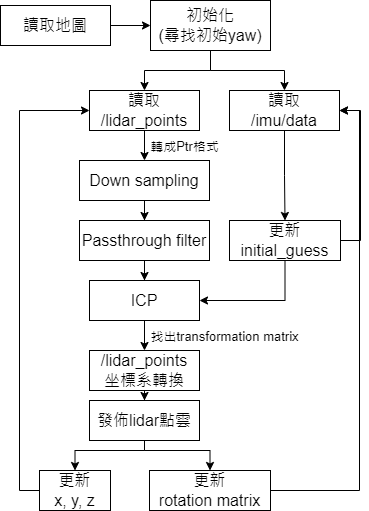
Report

Name: 謝元碩 Student ID: 311512015

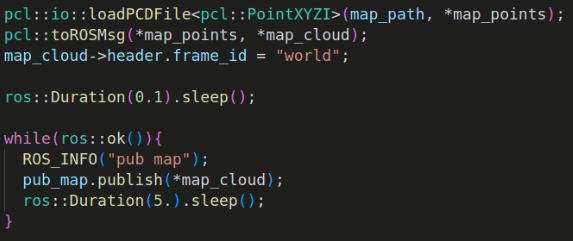
1. Code Pipeline



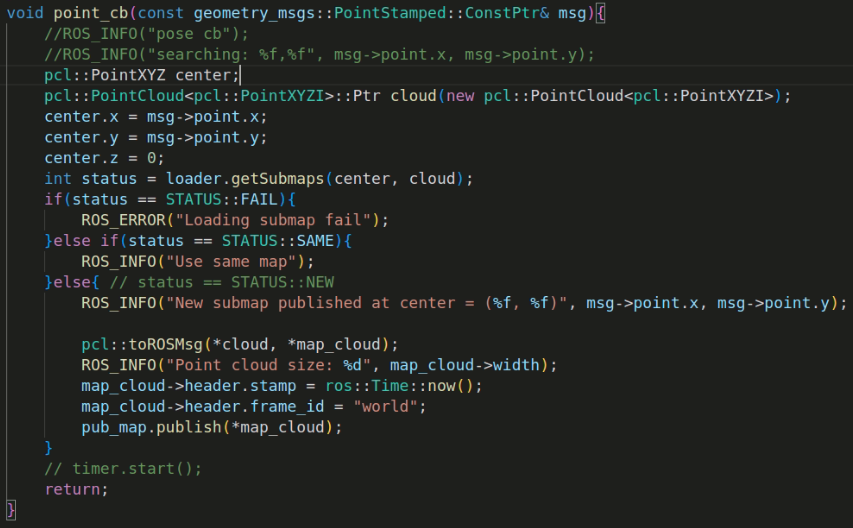
1. Code explanation

* **讀取地圖**

1. 競賽一使用pub\_map\_node.cpp首先讀取pcd檔，並轉成ROS的message格式後發佈至topic，程式碼如下：



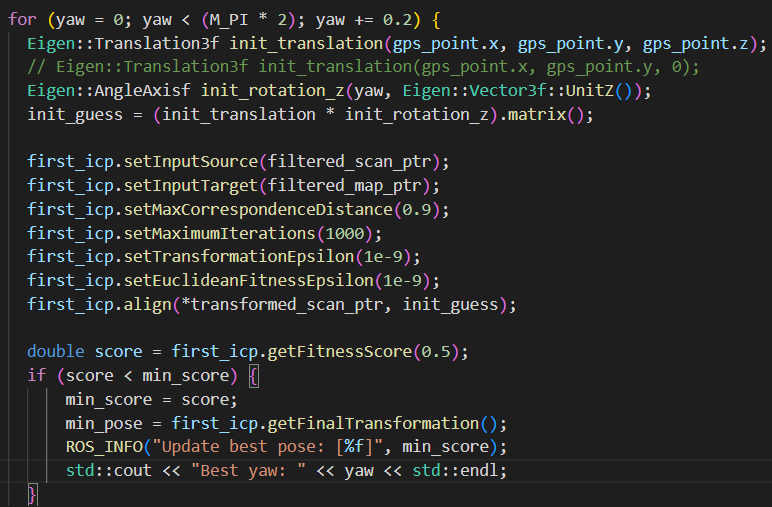
1. 競賽二、三則使用map\_publisher.cpp，由於地圖龐大且子地圖較多，透過取得GPS定位資訊，將map點雲修正到正確的座標位置並發佈出去，主要程式碼如下：



其中getSubmaps的用意是將lidar點雲座標以及GPS當下xyz資訊為輸入，找出此點雲附近的pcd檔(檔案名稱就是xy座標，可透過此資訊尋找)，並發佈至topic。

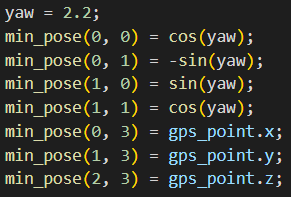
* **Yaw值初始化**

在最一開始，lidar的點雲與map的點雲並不會重合在一起，因此必須對lidar的初始點雲做座標轉換(即旋轉)，至於要旋轉多少度，我們必須去尋找最佳的yaw。



這個部分則是透過for迴圈來完成。迴圈中，我們不斷增加yaw值(initial\_guess)以做測試，在初始map與lidar點雲的icp做校正，並透過getFitnessScore()的分數不斷更新最佳yaw值以及轉移矩陣。

由於尋找yaw值的時間較為耗時，因此我在找到最佳yaw後，將程式碼修改如下：

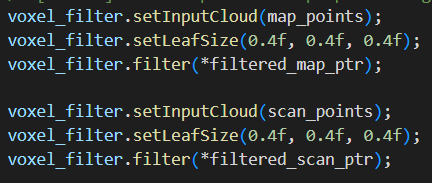


直接使用最佳yaw去對lidar座標做初始轉換，之後就不用每次執行都要等待尋找yaw值，時間將省下非常多。

* **Callback function: /lidar\_points**

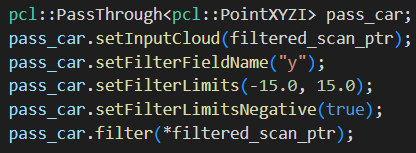
(i) 本次期中競賽的目的，就是如何讓lidar與map點雲疊合到最好，因此我們須對lidar點雲做前處理、應用。

一開始會先對lidar與map的點雲做降維，減少計算量：



0.4f為設定voxel的大小，決定了降維的程度多寡，降維後的點雲皆儲存在pcl::pointcloud的pointer格式filtered\_scan\_ptr中。

(ii) 接著視情況可使用passthrough filter。passthrough filter為直通濾波器，此濾波器能夠將特定範圍內的點雲做保留或濾除，甚至能用來去除outlier以增加icp準確度，或是對一些可能干擾定位的動態物件做濾除。

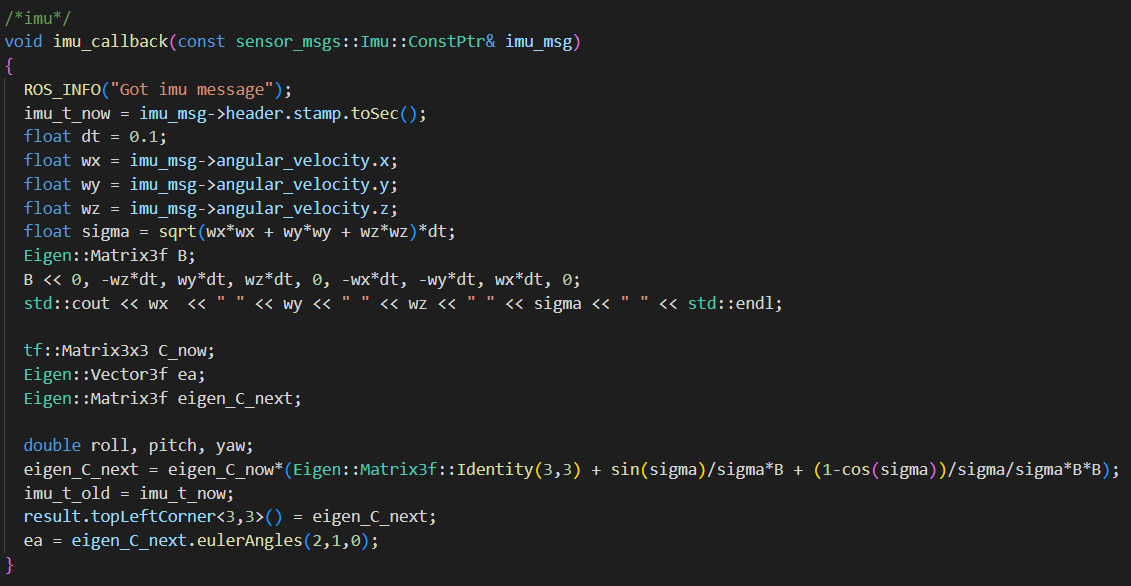


程式碼中僅須設定濾除的對象座標軸及範圍，

setFilterLimitsNegative(true)代表所選範圍內的點雲將被濾除；反之若是false，則是保留範圍內點雲，此用法將在競賽二、三中頻繁使用，以濾除道路上正在移動的汽車，以及地板，此部分會在problem and solution中做更多說明。

* **Callback function: /imu/data**

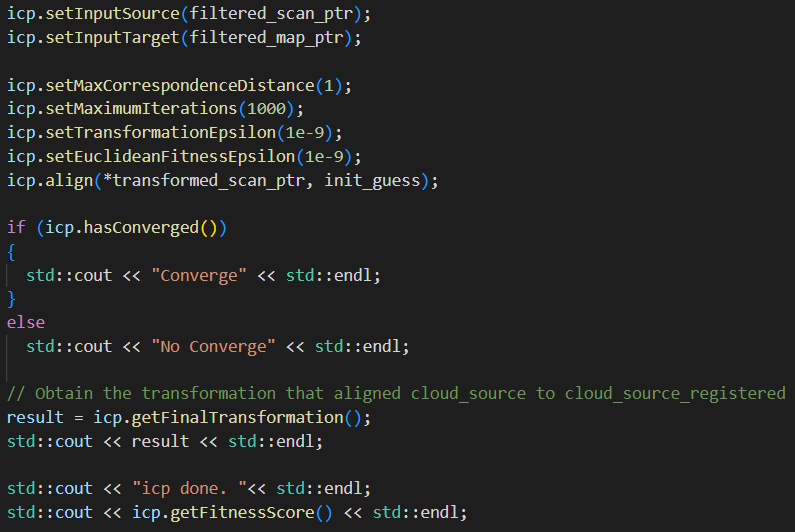
在競賽二、三中，由於為3D環境且場景複雜，且競賽二中汽車會先倒退幾秒再加速往前、競賽三中汽車有轉彎情形，僅使用單一lidar定位誤差容易隨時間增大。因此我加入imu慣性導航系統，將imu讀取到的角速度值做計算，去不斷修正icp產生的transformation matrix所需的initial\_guess，callback function如下：



這裡會先計算sigma以及B，使用公式為：

並使用前述兩值更新eigen C以做為下一次icp的新initial\_guess值，讓icp在align的時候降低誤差：

* **ICP**



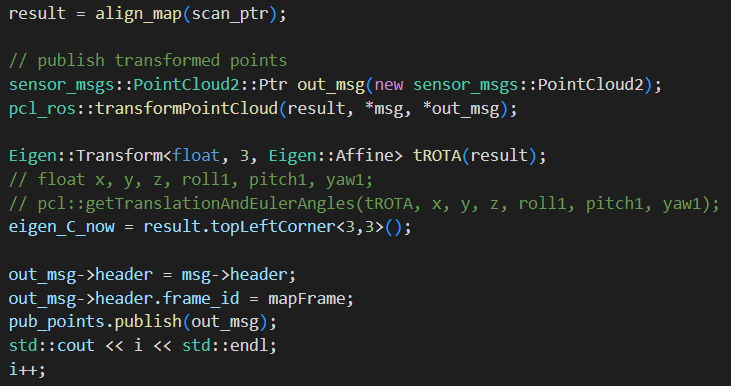
將降維後的lidar及map點雲做icp配準，其中參數設定包括點與點間的最大容許距離、迭代數、兩次icp矩陣間的誤差值等。

Icp.hasConverged()用來觀察是否配準成功；最後將轉移矩陣存至result以做使用。

在迭代做完icp後，可透過icp.getFitnessscore()回傳的數值觀察精準度，以做後續參數微調。

* **坐標系轉換、發佈最終點雲及更新姿態資訊**

這裡使用進行icp後產生的轉移矩陣，將lidar\_points轉到map坐標系，最後publish至topic。



Pcl\_ros::transformpointcloud將lidar點雲經座標轉換後存至out\_msg。

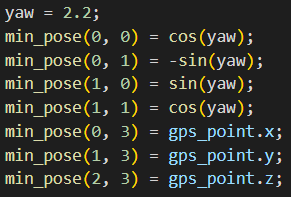
在上述程式碼中，有一行是將eigen\_C\_now做更新，此部分是為了儲存每一次icp後產生之轉移矩陣中的旋轉矩陣資訊，此旋轉矩陣將用在imu callback function中的eigen\_C\_now。

此部分最後即取得此汽車的x, y, z, roll, pitch, yaw資訊，並存進本作業需上傳之csv檔。

1. Contribution

* **節省每次尋找最佳initial pose的方法**

在每項競賽中，最佳initial pose的yaw值各自為定值，因此不需要每一次都尋找，這樣太耗時。我在這邊直接給予yaw一個數值代入求min\_pose，在透過迴圈找到合適的yaw後，將其輸入至此程式碼並註解原迴圈，此程式就會直接將lidar point轉換到最佳位置與map點雲重合，大幅縮短程式運作時間，並且不會影響到後續的bag運行。



* **新增imu優化定位誤差**

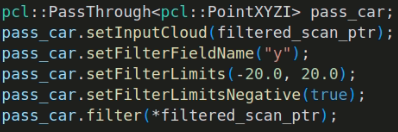
為提升競賽二、三的定位精準度，我加入imu去讀取角速度分量，並修正下一次的轉移矩陣初始值，此公式部分已在前面章節說明。由於imu的讀取頻率與lidar不同，兩者對initial\_guess的更新先後順序也會產生結果誤差變化。實測發現競賽二的imu更新速度慢，但效果卻不錯；競賽三imu更新速度正常，雖精準度有所提升，但提升量不大。

1. Problem & Solution

* **Passthrough filter設計理念**

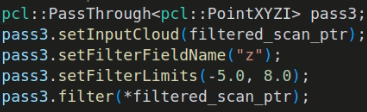
(i) 在競賽一，由於環境單純且為2D，故只嘗試調整icp參數達成定位效果。競賽二及競賽三的環境較複雜，而且動態物件多(汽車、樹葉)，因此必須將易影響到定位的這些動態物件濾除。

(ii) 競賽二中，對向車道會有汽車行經，靠近將會影響會定位；另外，我認為停在路旁的車子也會影響定位，因為lidar點雲上雖有汽車，但map上卻沒有，若做icp會導致點對點配準失敗。



因此，我選擇對y軸方向(汽車行進方向)進行濾除，濾除範圍為正負20。這樣做的話會**將非常靠近的動態汽車消除，同時也會移除兩旁的路邊停車，以及附近的地板資訊**，然而這會失去部分靜態牆壁資訊。

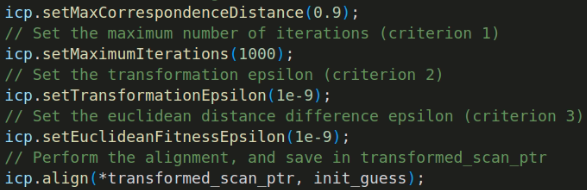
就定位原理而言，其實並非需要所有掃描資訊就能夠完成定位，就算濾除了一些近距離點雲(像牆壁)，剩餘的較遠點雲仍可實現定位功能，因此在做完y軸範圍濾除後，競賽二的定位誤差仍相當可觀。



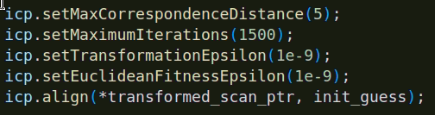
此外，我對z軸的-5~8範圍作點雲保留，此部分僅希望能夠去除在地圖以外的些許outlier，以確保每一個點雲都能正確配準。

(iii) 競賽三中，一開始我沿用競賽二的程式碼做參數微調後進行測試，但發現誤差較大。可能原因為此環境相較競賽二單純許多，道路旁只有牆壁、無樹木或停車，對向行經的汽車較少，沒必要濾除太多環lidar點雲資訊，因此我將passthrough filter的y軸濾除部分註解掉，如競賽一中用周圍所有的點雲資訊執行icp，經測試後準確度有明顯提升。

* **ICP參數設計理念**



(i) 競賽一的參數設置如上。這裡我將max correspondence設定較小、最後兩項設置為1e-9的理由，是因為競賽一是一個環境相當單純的2D地圖，且lidar移動速度穩定，使用相當高的配準條件是可以被容許的，因此設置這一組參數。



(ii) 對競賽二來說，我調整icp的主要根據，是lidar的移動情形。一開始max correspondence同樣使用1做定位，但很快就產生很高的誤差。仔細觀察後發現汽車的移動流程，是先緩慢後退一段距離後，接著加速往前，此部分的lidar更新可能沒辦法那麼快，導致點與點間的平均距離增大，icp就會移除這些長距離的配準資訊，間接影響定位結果。因此我將max correspondence更改為5去作配準，同時將bag播放速度放慢為0.01，成效非常好。

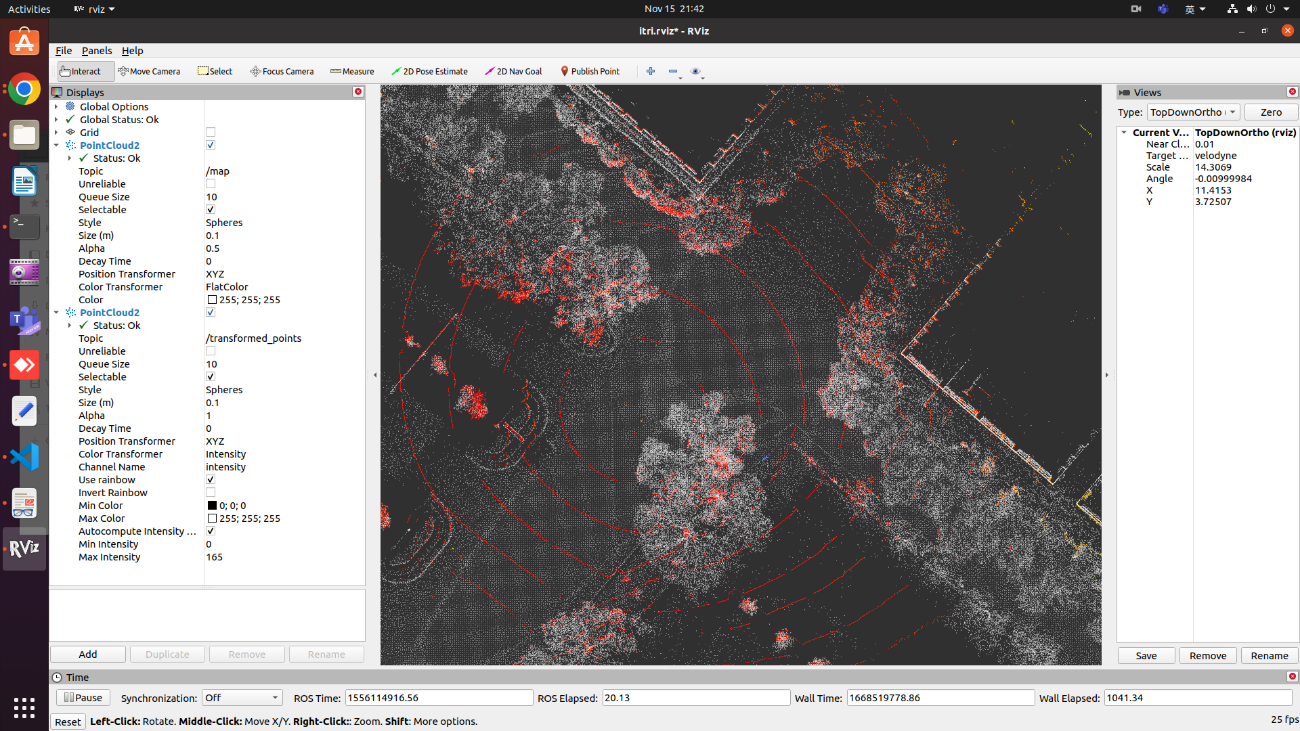
(iii) 競賽三的icp部分，由於場景單純，故沿用競賽一的參數去做icp，增加配準精度。

1. Others

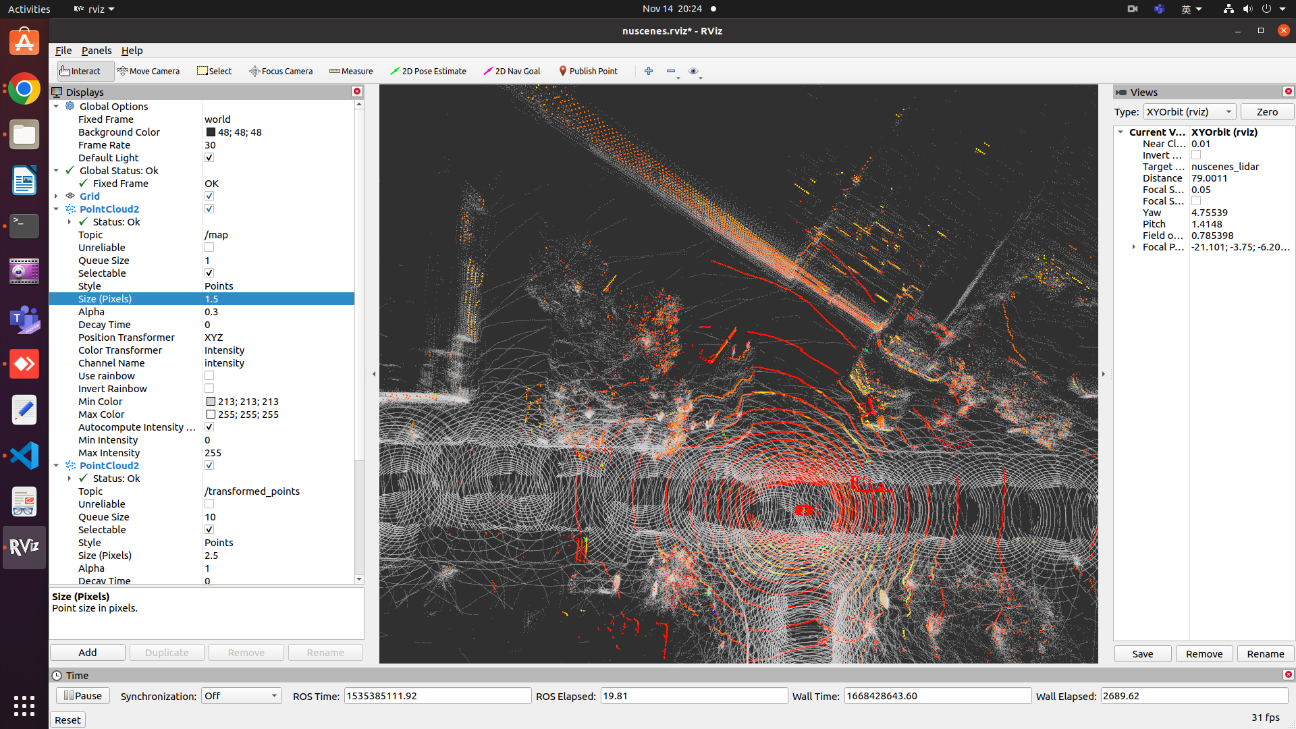
* **定位結果示意圖：**

下面結果皆顯示，在本次期中三項競賽中，除了成功將lidar點雲與map點雲配準，針對不同環境，同樣能夠實現不錯的定位，不會發生任何兩點雲間距離的偏移。

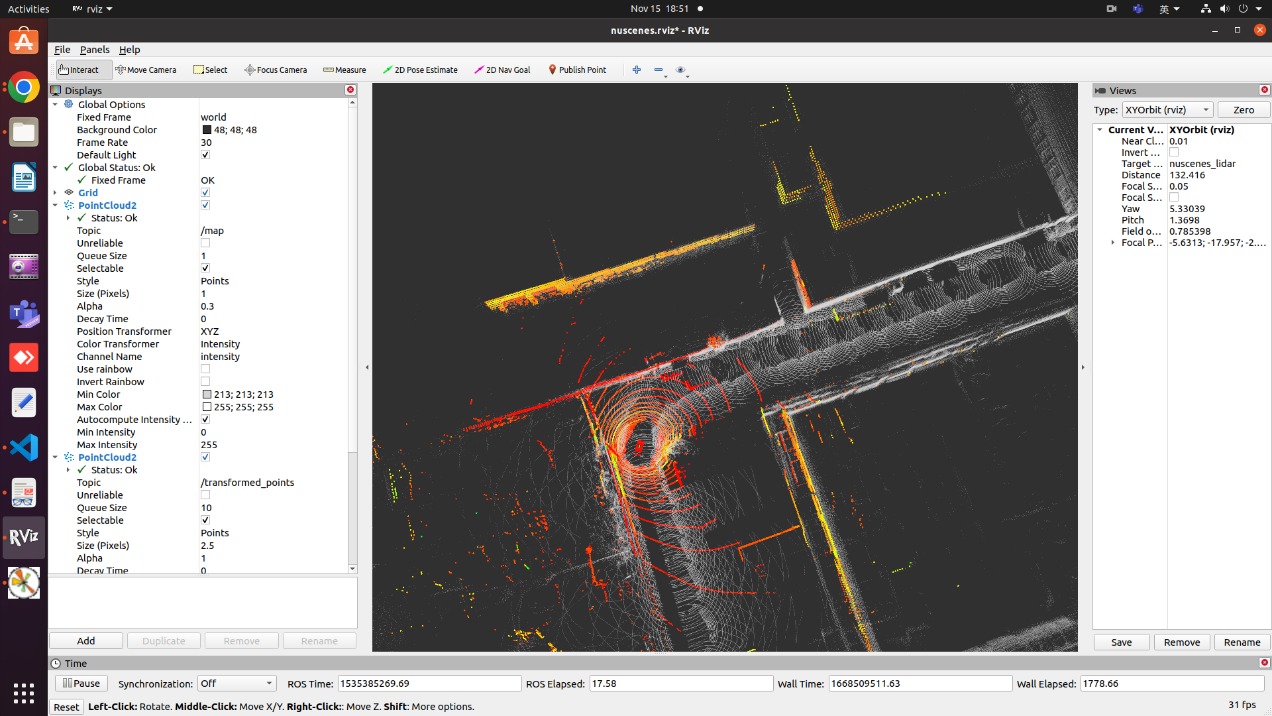
(i) 競賽一



(ii) 競賽二



(iii) 競賽三



* **terminal示意圖**

