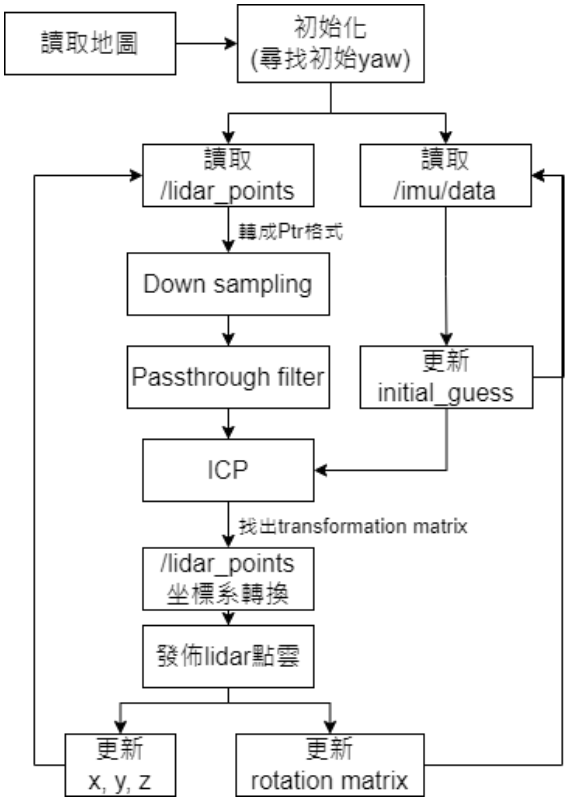
Report

Name: 謝元碩　Student ID: 311512015

1. Code Pipeline



2. Code explanation

- **讀取地圖**

  (i) 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：

```
pcl::io::loadPCDFile<pcl::PointXYZI>(map_path, *map_points);
pcl::toROSMsg(*map_points, *map_cloud);
map_cloud->header.frame_id = "world";

ros::Duration(0.1).sleep();

while(ros::ok()){
  ROS_INFO("pub map");
  pub_map.publish(*map_cloud);
  ros::Duration(5.).sleep();
}
```

(ii) 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：

```
void point_cb(const geometry_msgs::PointStamped::ConstPtr& msg){
    //ROS_INFO("pose cb");
    //ROS_INFO("searching: %f,%f", msg->point.x, msg->point.y);
    pcl::PointXYZ center;
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZI>);
    center.x = msg->point.x;
    center.y = msg->point.y;
    center.z = 0;
    int status = loader.getSubmaps(center, cloud);
    if(status == STATUS::FAIL){
        ROS_ERROR("Loading submap fail");
    }else if(status == STATUS::SAME){
        ROS_INFO("Use same map");
    }else{ // status == STATUS::NEW
        ROS_INFO("New submap published at center = (%f, %f)", msg->point.x, msg->point.y);

        pcl::toROSMsg(*cloud, *map_cloud);
        ROS_INFO("Point cloud size: %d", map_cloud->width);
        map_cloud->header.stamp = ros::Time::now();
        map_cloud->header.frame_id = "world";
        pub_map.publish(*map_cloud);
    }
    // timer.start();
    return;
}
```

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。

```
for (yaw = 0; yaw < (M_PI * 2); yaw += 0.2) {
  Eigen::Translation3f init_translation(gps_point.x, gps_point.y, gps_point.z);
  // Eigen::Translation3f init_translation(gps_point.x, gps_point.y, 0);
  Eigen::AngleAxisf init_rotation_z(yaw, Eigen::Vector3f::UnitZ());
  init_guess = (init_translation * init_rotation_z).matrix();

  first_icp.setInputSource(filtered_scan_ptr);
  first_icp.setInputTarget(filtered_map_ptr);
  first_icp.setMaxCorrespondenceDistance(0.9);
  first_icp.setMaximumIterations(1000);
  first_icp.setTransformationEpsilon(1e-9);
  first_icp.setEuclideanFitnessEpsilon(1e-9);
  first_icp.align(*transformed_scan_ptr, init_guess);

  double score = first_icp.getFitnessScore(0.5);
  if (score < min_score) {
      min_score = score;
      min_pose = first_icp.getFinalTransformation();
      ROS_INFO("Update best pose: [%f]", min_score);
      std::cout << "Best yaw: " << yaw << std::endl;
  }
```

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：

```
yaw = 2.2;
min_pose(0, 0) = cos(yaw);
min_pose(0, 1) = -sin(yaw);
min_pose(1, 0) = sin(yaw);
min_pose(1, 1) = cos(yaw);
min_pose(0, 3) = gps_point.x;
min_pose(1, 3) = gps_point.y;
min_pose(2, 3) = gps_point.z;
```

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：

```
voxel_filter.setInputCloud(map_points);
voxel_filter.setLeafSize(0.4f, 0.4f, 0.4f);
voxel_filter.filter(*filtered_map_ptr);

voxel_filter.setInputCloud(scan_points);
voxel_filter.setLeafSize(0.4f, 0.4f, 0.4f);
voxel_filter.filter(*filtered_scan_ptr);
```

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。

```
pcl::PassThrough<pcl::PointXYZI> pass_car;
pass_car.setInputCloud(filtered_scan_ptr);
pass_car.setFilterFieldName("y");
pass_car.setFilterLimits(-15.0, 15.0);
pass_car.setFilterLimitsNegative(true);
pass_car.filter(*filtered_scan_ptr);
```

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：

```
/*imu*/
void imu_callback(const sensor_msgs::Imu::ConstPtr& imu_msg)
{
  ROS_INFO("Got imu message");
  imu_t_now = imu_msg->header.stamp.toSec();
  float dt = 0.1;
  float wx = imu_msg->angular_velocity.x;
  float wy = imu_msg->angular_velocity.y;
  float wz = imu_msg->angular_velocity.z;
  float sigma = sqrt(wx*wx + wy*wy + wz*wz)*dt;
  Eigen::Matrix3f B;
  B << 0, -wz*dt, wy*dt, wz*dt, 0, -wx*dt, -wy*dt, wx*dt, 0;
  std::cout << wx  << " " << wy << " " << wz << " " << sigma << " " << std::endl;

  tf::Matrix3x3 C_now;
  Eigen::Vector3f ea;
  Eigen::Matrix3f eigen_C_next;

  double roll, pitch, yaw;
  eigen_C_next = eigen_C_now*(Eigen::Matrix3f::Identity(3,3) + sin(sigma)/sigma*B + (1-cos(sigma))/sigma/sigma*B*B);
  imu_t_old = imu_t_now;
  result.topLeftCorner<3,3>() = eigen_C_next;
  ea = eigen_C_next.eulerAngles(2,1,0);
}
```

這裡會先計算 sigma 以及 B，使用公式為：

$$\sigma = |wt|$$

$$B = \begin{vmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{vmatrix}$$

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：

$$C(t+dt) = C(t)(I + \frac{\sin\sigma}{\sigma}B + \frac{1-\cos\sigma}{\sigma^2}B^2)$$

- **ICP**

```
icp.setInputSource(filtered_scan_ptr);
icp.setInputTarget(filtered_map_ptr);

icp.setMaxCorrespondenceDistance(1);
icp.setMaximumIterations(1000);
icp.setTransformationEpsilon(1e-9);
icp.setEuclideanFitnessEpsilon(1e-9);
icp.align(*transformed_scan_ptr, init_guess);

if (icp.hasConverged())
{
  std::cout << "Converge" << std::endl;
}
else
  std::cout << "No Converge" << std::endl;

// Obtain the transformation that aligned cloud_source to cloud_source_registered
result = icp.getFinalTransformation();
std::cout << result << std::endl;

std::cout << "icp done. "<< std::endl;
std::cout << icp.getFitnessScore() << std::endl;
```

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。

```
result = align_map(scan_ptr);

// publish transformed points
sensor_msgs::PointCloud2::Ptr out_msg(new sensor_msgs::PointCloud2);
pcl_ros::transformPointCloud(result, *msg, *out_msg);

Eigen::Transform<float, 3, Eigen::Affine> tROTA(result);
// float x, y, z, roll1, pitch1, yaw1;
// pcl::getTranslationAndEulerAngles(tROTA, x, y, z, roll1, pitch1, yaw1);
eigen_C_now = result.topLeftCorner<3,3>();

out_msg->header = msg->header;
out_msg->header.frame_id = mapFrame;
pub_points.publish(out_msg);
std::cout << i << std::endl;
i++;
```

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。

3. 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。

```
yaw = 2.2;
min_pose(0, 0) = cos(yaw);
min_pose(0, 1) = -sin(yaw);
min_pose(1, 0) = sin(yaw);
min_pose(1, 1) = cos(yaw);
min_pose(0, 3) = gps_point.x;
min_pose(1, 3) = gps_point.y;
min_pose(2, 3) = gps_point.z;
```

- **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。

4. 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。

```
pcl::PassThrough<pcl::PointXYZI> pass_car;
pass_car.setInputCloud(filtered_scan_ptr);
pass_car.setFilterFieldName("y");
pass_car.setFilterLimits(-20.0, 20.0);
pass_car.setFilterLimitsNegative(true);
pass_car.filter(*filtered_scan_ptr);
```

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。

```
pcl::PassThrough<pcl::PointXYZI> pass3;
pass3.setInputCloud(filtered_scan_ptr);
pass3.setFilterFieldName("z");
pass3.setFilterLimits(-5.0, 8.0);
pass3.filter(*filtered_scan_ptr);
```

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**

```
icp.setMaxCorrespondenceDistance(0.9);
// Set the maximum number of iterations (criterion 1)
icp.setMaximumIterations(1000);
// Set the transformation epsilon (criterion 2)
icp.setTransformationEpsilon(1e-9);
// Set the euclidean distance difference epsilon (criterion 3)
icp.setEuclideanFitnessEpsilon(1e-9);
// Perform the alignment, and save in transformed_scan_ptr
icp.align(*transformed_scan_ptr, init_guess);
```

(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。

```
icp.setMaxCorrespondenceDistance(5);
icp.setMaximumIterations(1500);
icp.setTransformationEpsilon(1e-9);
icp.setEuclideanFitnessEpsilon(1e-9);
icp.align(*transformed_scan_ptr, init_guess);
```

(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。

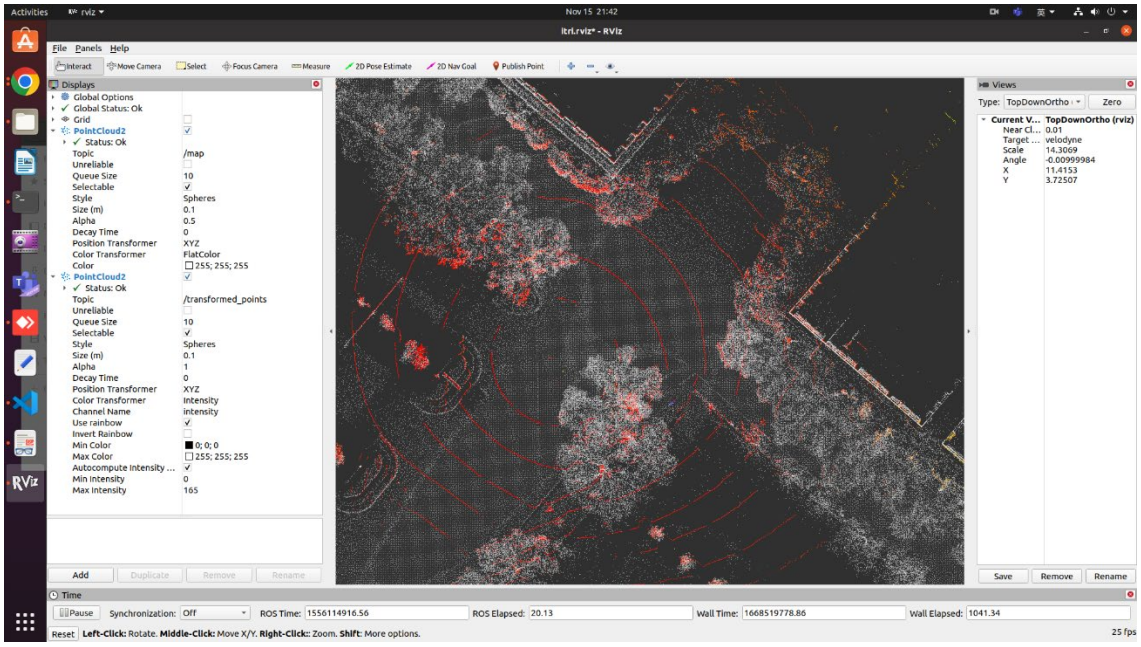
(iii) 競賽三的 icp 部分，由於場景單純，故沿用競賽一的參數去做 icp，增加配準精度。

5. Others

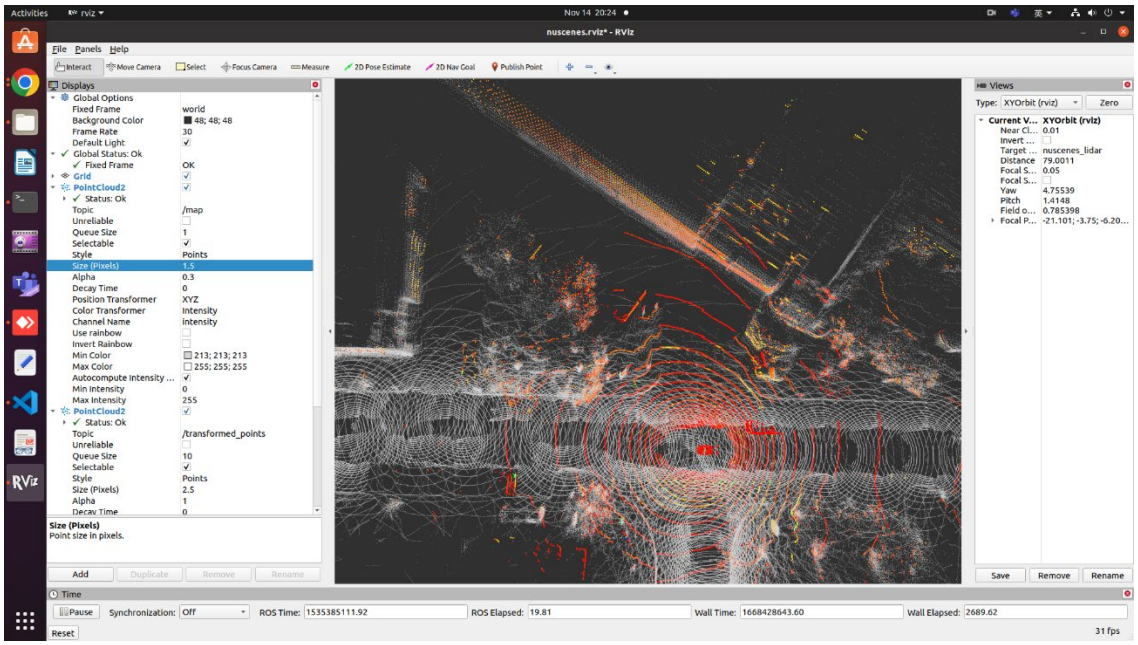
- **定位結果示意圖：**

  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。

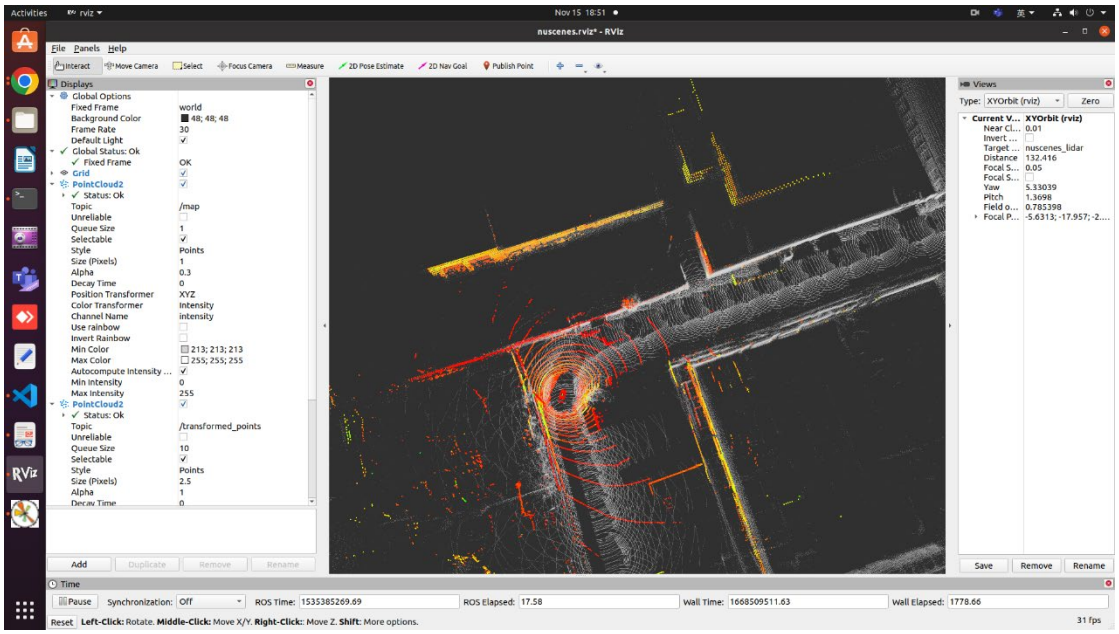
  (i) 競賽一



(ii) 競賽二



(iii) 競賽三



- **terminal 示意圖**

```
[ INFO] [1668509746.092717722, 1535385271.655024476]: Got lidar message
[ INFO] [1668509746.094665891, 1535385271.655024476]: point size: 34721
Converge
  -0.639596    0.781074   0.0431351     1679.91
  -0.780815   -0.641513 -0.00796573     924.002
  0.0215482   -0.038519     1.00753     1.83479
          0           0           0           1
icp done.
0.46264
388
[ INFO] [1668509753.990458596, 1535385271.655024476]: Got lidar message
[ INFO] [1668509753.993401657, 1535385271.655024476]: point size: 34721
Converge
   -0.63587    0.784028   0.0446095      1680.2
  -0.783818   -0.637818 -0.00948935     923.841
  0.0211147  -0.0407281     1.00738      1.8118
          0           0           0           1
icp done.
0.428842
389
```