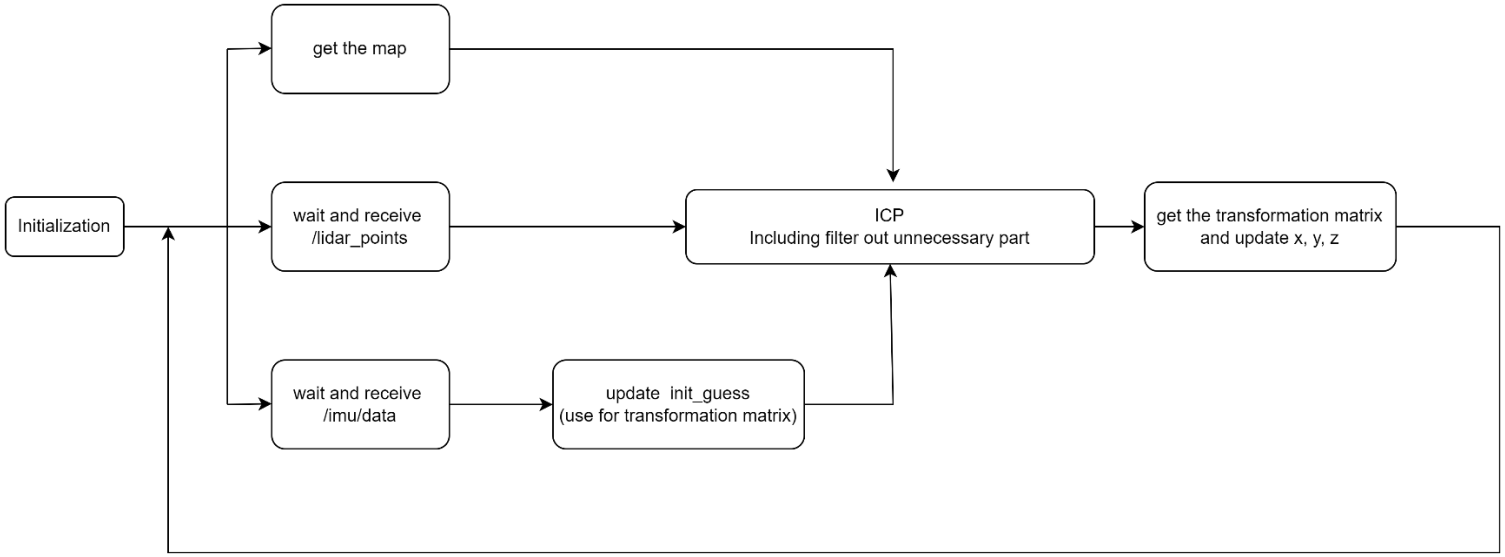
SDC Localization Competition

311512064 鄧書桓

1. Pipeline



基本 code 介紹:

i. Get the map

在競賽一使用 pub_map.cpp 讀取地圖(將 pcd 轉成 ROS 的資料形式再傳至 topic 供 localization.cpp 使用)，競賽二、三使用 map_publisher.cpp，先藉由 GPS 資訊將車子位置附近的幾個子地圖發布至 topic(由於地圖太大，一次讀取完會導致效能降低)。

ii. Wait and receive /lidar_points

藉由 pc_callback 這個回調函數對/lidar_points 這個 topic 做一些取值或些許調整，並藉由 aling_map 將吃到的

點雲做一些調整、疊圖，如 voxel down 或是濾除部分點雲，詳細的解釋會在附上的 code 中做說明，以下只放部分照片供參考:

Voxel down(將點雲數量縮小，可以順便去除雜訊):

```
//use for voxel down, can also remove noise
voxel_filter.setInputCloud (filtered_scan_ptr);
// voxel_filter.setInputCloud (scan_points);
//(1.0f)
voxel_filter.setLeafSize (0.3f, 0.3f, 0.3f); //0.6f
voxel_filter.filter (*filtered_scan_ptr);
std::cout << "filtered_scan_ptr: " << filtered_scan_ptr->size() << std::endl;

// if no use voxel down, map points will too much
voxel_filter.setInputCloud (map_points);
//(0.6f)
voxel_filter.setLeafSize (0.3f, 0.3f, 0.3f);
voxel_filter.filter (*filtered_map_ptr);
std::cout << "filtered_map_ptr: " << filtered_map_ptr->size() << std::endl;
```

尋找最佳的 initial yaw(initial pose):

```
int iter = 1;
float temp = 0.0;
for (yaw = 0; yaw < (M_PI * 2); yaw += 0.6) {
  Eigen::Translation3f init_translation(gps_point.x, gps_point.y, gps_point.z);
  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(500); //5000
  first_icp.setTransformationEpsilon(1e-9);//-8
  first_icp.setEuclideanFitnessEpsilon(1e-9);//-8
  first_icp.align(*transformed_scan_ptr, init_guess);
  ROS_INFO("Updating");
  temp = std::floor((iter/((M_PI*2 -1)/0.6) * 100)*100)/100;
  std::cout<<"Updating min_yaw ....."<<temp<<"%"<<std::endl;
  double score = first_icp.getFitnessScore(0.5);
  if (score < min_score) {
      min_score = score;
      min_pose = first_icp.getFinalTransformation();
      ROS_INFO("Update best pose");
      min_yaw = yaw;
  }
  iter++;

}
```

ICP 部分參數調整與流程:

```
/* [Part 2] Perform ICP here or any other scan-matching algorithm */
/* Refer to https://pointclouds.org/documentation/classpcl_1_1_iterative_closest_point.html#details */
  //要疊的部份
  icp.setInputSource(filtered_scan_ptr);
  //疊過去的部份
  icp.setInputTarget(filtered_map_ptr);
  //Set the max correspondence distance to 1cm, remove outliner 1.(0.9) 2.(20)
  icp.setMaxCorrespondenceDistance(6); //6
  // Set the maximum number of iterations (5000)
  icp.setMaximumIterations(5000);
  //前一个变换矩阵和当前变换矩阵的差异小于阈值时，就认为已经收敛了，是一条收敛条件 (1e-09)
  icp.setTransformationEpsilon(1e-9);
  //还有一条收敛条件是均方误差和小于阈值， 停止迭代 (1e-9)
  icp.setEuclideanFitnessEpsilon(1e-9);
  //transformed_scan_ptr 為新的點雲，將source 先根據init_guess轉換後在根據icp與target 疊合所求得
  icp.align(*transformed_scan_ptr, init_guess);
  if(!icp.hasConverged ())
  {
    std::cout << "not converge" << std::endl;
  }
  std::cout << "initial_guess: " << init_guess << std::endl;
  //result is type of Eigen::Matrix4f
  result = icp.getFinalTransformation();
  std::cout << "result: " << result << std::endl;
  std::cout << "====================================" << std::endl;
  //icp.getFitnessScore() goar of fitness
  std::cout << "The MSE is : "<< icp.getFitnessScore() << std::endl;
  // pub filtered point
  pcl::toROSMsg(*transformed_scan_ptr, *test_msg);
  test_msg->header.frame_id = "world";
  pub_test.publish(test_msg);
  std::cout << "====================================" << std::endl;
/* Use result as next initial guess */
  init_guess = result;
  return result;
}
```

2. Contribution

由於三個 bag 檔中的場景每個都差異很大，因此我這邊對於每個場景都有做 case by case 的調整。但是三個都有一個共同的部分，就是一開始找尋最佳 yaw 的方式(用於 initial pose)，這邊使用助教提示的方式來完成，入下圖所示，透過嘗試不同大小的 yaw，並以更低的 getFitnessScore 來更新最佳 yaw。順帶一提，在這邊有使用過更低的間隔去做尋找(如:0.1、0.2 與 0.005)，但是它們既耗時且表現的也不會差到很多，因此後來就

被我捨棄掉了。

```
int iter = 1;
float temp = 0.0;
for (yaw = 0; yaw < (M_PI * 2); yaw += 0.6) {
  Eigen::Translation3f init_translation(gps_point.x, gps_point.y, gps_point.z);
  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(500); //5000
  first_icp.setTransformationEpsilon(1e-9);//-8
  first_icp.setEuclideanFitnessEpsilon(1e-9);//-8
  first_icp.align(*transformed_scan_ptr, init_guess);
  ROS_INFO("Updating");
  temp = std::floor((iter/((M_PI*2 -1)/0.6) * 100)*100)/100;
  std::cout<<"Updating min_yaw ....."<<temp<<"%"<<std::endl;
  double score = first_icp.getFitnessScore(0.5);
  if (score < min_score) {
      min_score = score;
      min_pose = first_icp.getFinalTransformation();
      ROS_INFO("Update best pose");
      min_yaw = yaw;
  }
  iter++;

}
```

接著是各場景 case by case 調整的部分。首先，先從第一個場景說起:

場景一:

這邊的 ICP 並沒有做多餘的變化，僅使用 open source 教的去做應用，本來考慮到新竹的風較大，會導致樹葉造成點雲疊圖的失誤率，因此想將地圖中上面包含到樹葉部分的點雲濾除，但是經過多次嘗試，儘管只有將樹葉的點雲部分濾除(也就是根據 Z 軸上去做限制)，效果也比全部點雲都吃進去做比較來的差，因此這邊就沒多做濾除。

另外，還有加入上網找到他人使用 imu 的方法，來根據慣性導航系統去更新 transformation matrix(更新 initial pose)。

以下是使用到的 imu code:

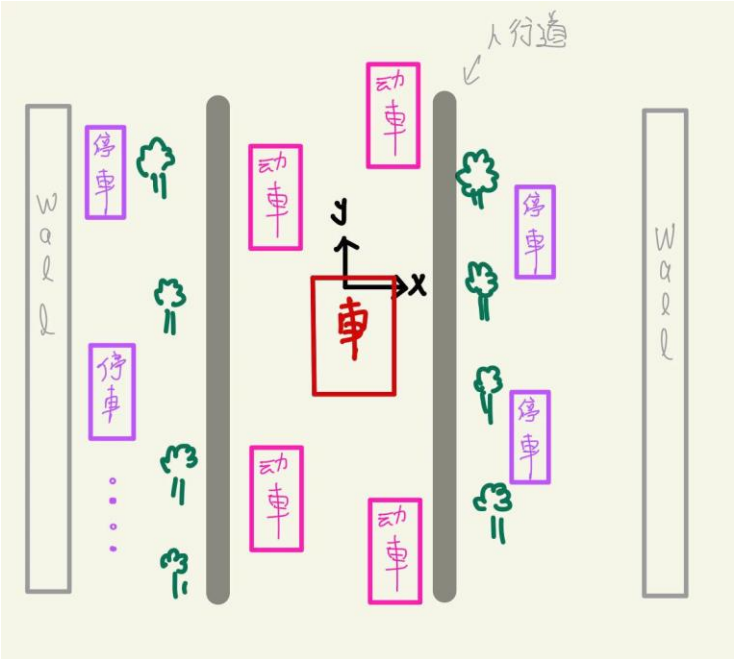
```
void callback_imu(const sensor_msgs::Imu::ConstPtr& imu_msg)
{
  imu_t_now = imu_msg->header.stamp.toSec();
  float dt = 0.1;
  float wx = imu_msg->angular_velocity.x;
  float wy = imu_msg->angular_velocity.y;
  float wz = imu_msg->angular_velocity.z;
  float sigma = sqrt(wx*wx + wy*wy + wz*wz)*dt;
  Eigen::Matrix3f B;
  B << 0, -wz*dt, wy*dt, wz*dt, 0, -wx*dt, -wy*dt, wx*dt, 0;

  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);
}
```

這邊的 B 是根據左下方這個式子去做計算，其中的各軸角速度是由 imu 根據 body frame 獲得的，而 eigen_C_next 是根據右下方的式子去做計算，以此來更新下一次做 icp 的初始值。這邊值得注意的是，/imu/data topic 發布的頻率比 lidar_points 發布的頻率低許多，因此主要還是根據 lidar_point 中的點雲資料去做點雲疊合定位，imu 的功能偏向輔助修正偏差。

$$B = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \qquad C(t+dt) = C(t)\left(I + \frac{\sin(\sigma)}{\sigma}B + \frac{1-\cos(\sigma)}{\sigma^2}B^2\right)$$

場景二:

由於本場地在車子附近多了許多高精地圖上沒有的會移動的障礙物(汽車與巴士)，還有新的障礙物，如停止的汽車，因此這邊有將點雲資料許多做切割，示意圖如下:



因此我的作法為先做出左右兩邊僅有牆壁資訊的點雲、所有高於車子高度的點雲、左右由人行道至樹木的點雲(不包含車道，會有移動的車子)，最後將他們全部相加得到分割完的點雲後對 y 軸進行裁減，以避免吃到離車子太遠的資訊。而上述這樣分很多部分去做雲的原因為這樣相較於直接使用切割的方式，能夠較好的去觀察各部分點雲切割的狀態。以下為這部分的 code:

```
//////////left wall of car
pass.setInputCloud(scan_points);
//According to x axis to do filtering
pass.setFilterFieldName("x");
pass.setFilterLimits(-40.0, -9.0);
// pass.setFilterLimits(-40.0, -10.0);
pass.filter(*filtered_scan_ptr);

//According to y axis to do filtering
pass.setInputCloud(filtered_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-50.0, 50.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*filtered_scan_ptr);

//According to z axis to do filtering
pass.setInputCloud(filtered_scan_ptr);
pass.setFilterFieldName("z");
pass.setFilterLimits(-1.0, 3.0); //(-1.0, 3.0)
pass.filter(*filtered_scan_ptr);
```

```
//////////right wall of car
pass.setInputCloud(scan_points);
//According to x axis to do filtering
pass.setFilterFieldName("x");
pass.setFilterLimits(10.0, 35.0);
//pass.setFilterLimits(10.0, 35.0); ////this has better
pass.filter(*temp_scan_ptr);

//According to y axis to do filtering
pass.setInputCloud(temp_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-50.0, 50.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*temp_scan_ptr);

//According to z axis to do filtering
pass.setInputCloud(temp_scan_ptr);
pass.setFilterFieldName("z");
pass.setFilterLimits(-1.0, 3.0); //(-1.0, 3.0)
pass.filter(*temp_scan_ptr);


*filtered_scan_ptr += *temp_scan_ptr;
// filtered_scan_ptr = temp_scan_ptr;
```

```
// z of higher than car
pass.setInputCloud(scan_points);
//According to x axis to do filtering
pass.setFilterFieldName("x");
pass.setFilterLimits(-35.0, 35.0);
//pass.setFilterLimits(-20.0, 35.0); ////this has better
pass.filter(*z_scan_ptr);

//According to y axis to do filtering
pass.setInputCloud(z_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-50.0, 50.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*z_scan_ptr);

//According to z axis to do filtering
pass.setInputCloud(z_scan_ptr);
pass.setFilterFieldName("z");
// pass.setFilterLimits(1.0, 10.0);
pass.setFilterLimits(1.0, 10.0);
pass.filter(*z_scan_ptr);


*filtered_scan_ptr += *z_scan_ptr;
// filtered_scan_ptr = z_scan_ptr;
```

```
// x near left car
pass.setInputCloud(scan_points);
//According to x axis to do filtering
pass.setFilterFieldName("x");
pass.setFilterLimits(-6.5, 0.0); //(-5.0, 0.0)
//pass.setFilterLimits(10.0, 35.0); ////this has better
pass.filter(*x_near_scan_ptr);

//According to y axis to do filtering
pass.setInputCloud(x_near_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-50.0, 50.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*x_near_scan_ptr);

//According to z axis to do filtering
pass.setInputCloud(x_near_scan_ptr);
pass.setFilterFieldName("z");
// pass.setFilterLimits(1.0, 10.0);
pass.setFilterLimits(-2.0, 1.0);
pass.filter(*x_near_scan_ptr);


*filtered_scan_ptr += *x_near_scan_ptr;
// filtered_scan_ptr = x_near_scan_ptr;
```

```
// x near right car
pass.setInputCloud(scan_points);
//According to x axis to do filtering
pass.setFilterFieldName("x");
pass.setFilterLimits(0.0, 5.3); //(0.0, 5.5)
//pass.setFilterLimits(10.0, 35.0); ////this has better
pass.filter(*x_right_near_scan_ptr);

//According to y axis to do filtering
pass.setInputCloud(x_right_near_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-50.0, 50.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*x_right_near_scan_ptr);

//According to z axis to do filtering
pass.setInputCloud(x_right_near_scan_ptr);
pass.setFilterFieldName("z");
// pass.setFilterLimits(1.0, 10.0);
pass.setFilterLimits(-2.0, 1.0);
pass.filter(*x_right_near_scan_ptr);


*filtered_scan_ptr += *x_right_near_scan_ptr;
// filtered_scan_ptr = x_near_scan_ptr;
```

```
//filter in y axis

//According to y axis to do filtering
pass.setInputCloud(filtered_scan_ptr);
pass.setFilterFieldName("y");
pass.setFilterLimits(-25.0, 25.0);
// pass.setFilterLimits(-25.0, 25.0);
pass.filter(*filtered_scan_ptr);
```

而這個場景中的/imu/data topic 傳遞的頻率遠低於 lidar_points，導致有加跟沒加一樣，因此這邊就不使用 imu 資訊。

場景三:

在這個場景一樣會有許多移動的障礙物車子，但是經過多次測試，有沒有濾除出來的效果很不明顯，因此最後僅對 z 軸做限制，將部分地板濾除而已。

而 imu 部分雖然其發布頻率比 lidar_points 的低一些，但是仍能做到修正的效果，因此有加入使用，這部分與場景一使用相同的 code，在此就不再做過多贅述。

3. Problem and Solution

在做本次期中競賽時主要遇到三個很麻煩的問題，第一個是場景二與場景三的地圖無法顯示的問題，由於我對於 Rviz 沒有很熟，所於在 debug 時花了很多時間，最後跟朋友一起找出來 ppt 上面指令給錯的問題(localization 少一個 a)，與 Rviz 的 config 檔中的 topic 給錯名子(/map 給成/world)。另外，第二個問題為 Rviz 有時候在 docker 無法使用的問題，單是這個一直找不到答案，因此後來都改成在本地端開啟 Rviz。第三個問題為 ICP 超參數的選擇方式，由於第二章圖的行進方式為後退一段距離後往前暴衝，這段因為速度太快會導致疊圖疊得很差，後來是將 icp.setMaxiimumCorrespondenceDistance(x)調高才能成功疊到圖，這個函式是用來將大於 x cm 的點雲當成 outliner 濾掉，調高可以將較遠的點雲圖疊在一起。