Self-Driving Cars Radar Localization Report

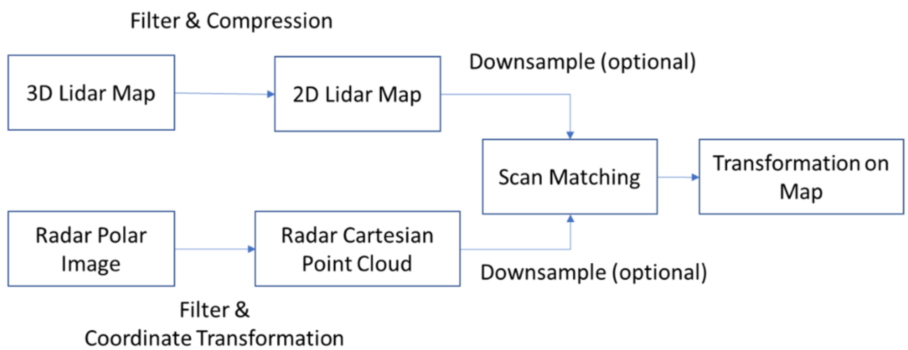
National Yang Ming Chiao Tung University

Graduate Degree Program in Robotics

Student ID: 312605001

Name: Ou, Ting-Wei

1. Pipeline
    a. Work flow

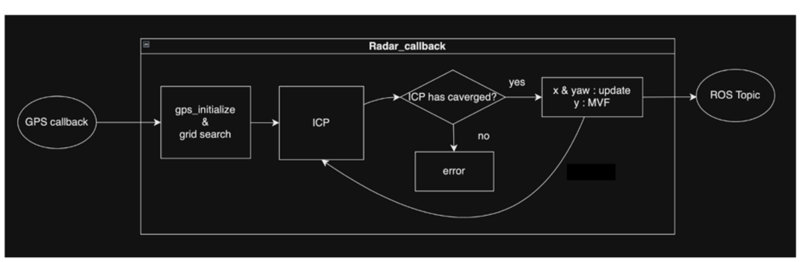


    b. Program Architecture

```
|-- catkin_ws
|--|-- src
|--|--|-- localization
|--|--|--|-- launch
|--|--|--|-- src
|--|--|--|--|-- localization.cpp
|--|--|--|--|-- map_modified.cpp
|--|--|--|--|-- radar.cpp
|--|--|--|--|-- MovingAverageFilter.cpp
|--|--|--|--|-- KF.cpp
```

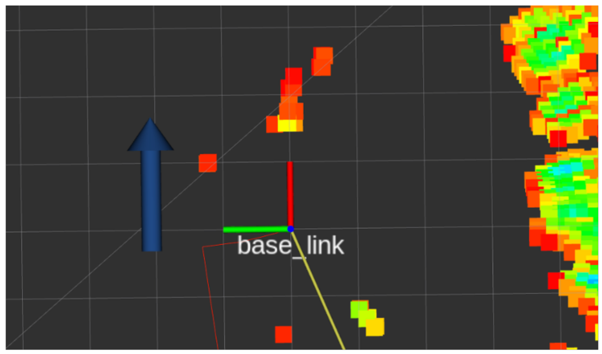
    c. Call back function and ICP

In this section, if gps_callback has been called, the radar_callback will execute gps_initialize and perform a grid search to find the best pose. Then, the ICP will iterate until convergence, and finally, the pose will be updated to the rostopic.

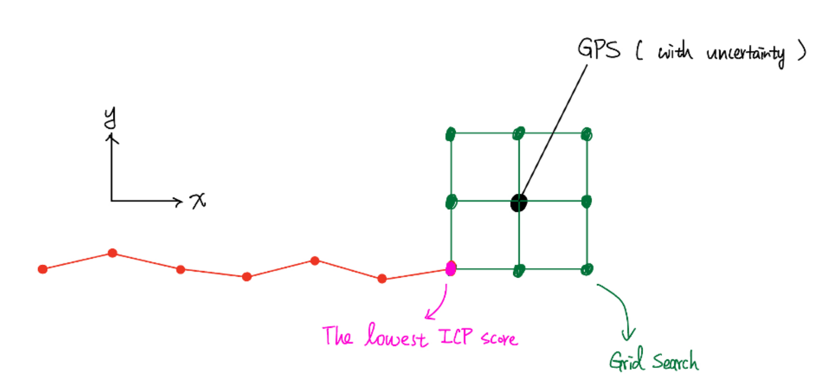


2. Contribution
    a. Optimizing GPS search range
    In this section, I'll introduce my contribution on how to use GPS data to prevent ICP failure. It is important to know that the GPS signal may not be entirely accurate, but we can utilize this inaccurate pose to estimate the best update pose.

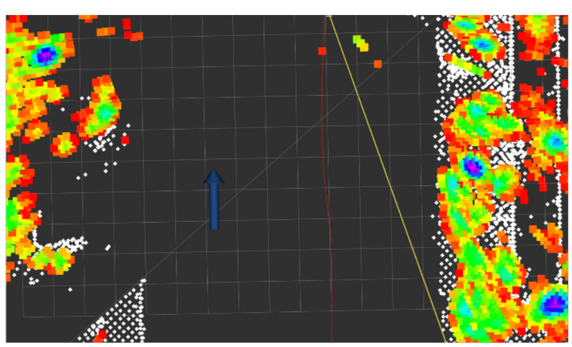
    As shown in the figure, when directly updating the pose using GPS, it results in an offset in the overall path.



    To address this issue, I have designed a pose updating strategy. When updating GPS pose, roughly align nearby positions using ICP, and choose the one that yields the lowest score for the GPS update.



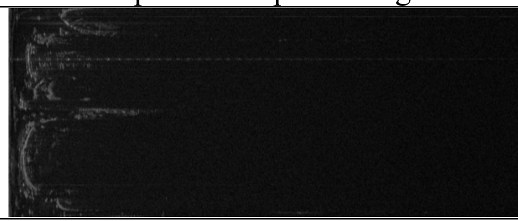
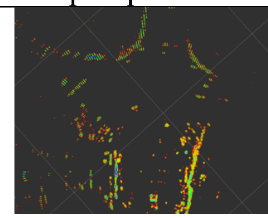
    The result are as follows:



3. Problem and solution

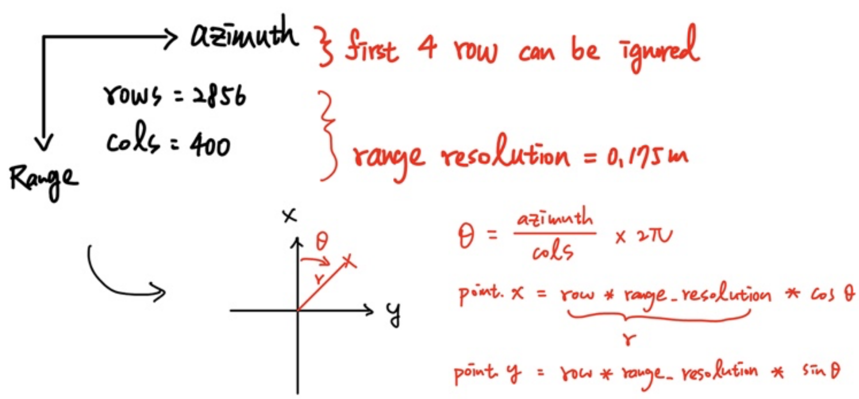
   a. Radar.cpp

      i. Sensor’s heading direction issue

Input : radar polar image	Output: point cloud
	

In the radar data processing section, we need to covert the poler image into a 2D point cloud and publish it on the rostopic.

First, we need to check the coordinate axis definition. During my initial attempt, I observed that the radar conversion result is inverted horizontally. This discrepancy arises from my assumption that the radar senser’s heading direction align with the car frame, whereas it is, in fact, oriented at 180 degree.



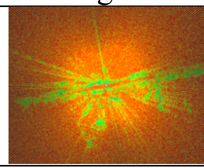
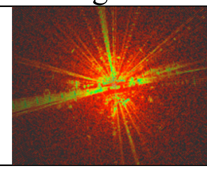
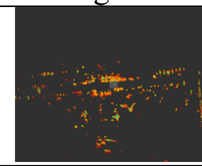
So I modified the conversion part to solve this issue

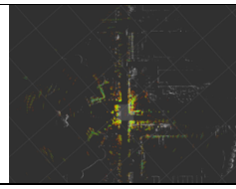
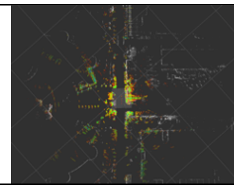
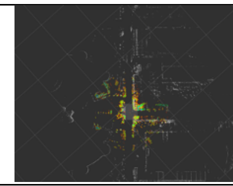
```
point.x = static_cast<float>(row) * range_resolution * cos(azimuth_rad);
point.y = static_cast<float>(row) * range_resolution * (-sin(azimuth_rad)); // Flip horizontally
point.z = 0;
```

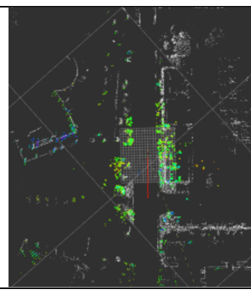
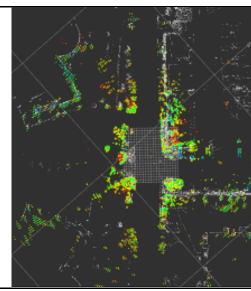
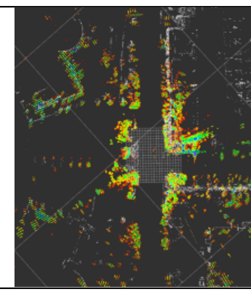
      ii. Noise issue

The radar signals contain a lot of noise, making it challenging for ICP perform accurate matching.

To address this problem, I design a filter that can control the sampling distance, radar signal strength and number of sampled points in each direction.

Strength = 0	Strength = 30	Strength = 70
		

Distance = 150 m	Distance = 100 m	Distance = 50 m
		

Points = 10	Points = 30	Points = 60
		

b. Localization.cpp

i. initialize issue

The initialization of pose is crucial for ICP, as poor initialization can easily lead to local minimum, GPS provides us with a good estimate for the initial pose guess. However, GPS may not be perfect in certain situations. Therefore, during GPS updates, I also perform a rough ICP matching (refer to the contribution) around the current pose to find the optimal GPS update pose.

ii. GPS update issue

ICP may fail when the map lacks distinctive features. To address this issue, I utilize GPS data. In instance where ICP fails, GPS will guide the pose correction.

iii. ICP matching issue

Setting ICP parameters properly can lead to better results in the iterative process. The parameters I have configured are as follows:

icp.setMaximumIterations	200
icp. setMaxCorrespondenceDistance	3.5
icp. setEuclideanFitnessEpsilon	1e-4
icp.setTransformationEpsilon	1e-4

c. Map_modified.cpp

i. Radar installation position

In order to approximate the installation position of the radar, I consulted several automotive-related websites and found that the height of an SUV is approximately around 1.6 meters.

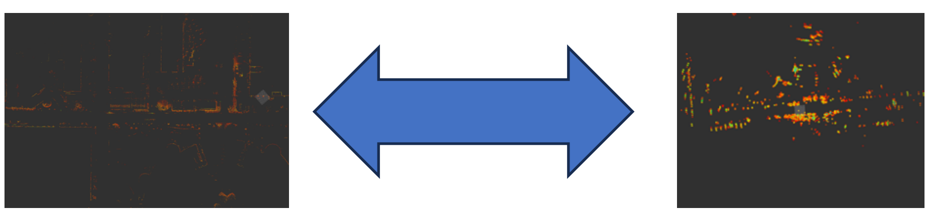
ii. Scan range

Next, I searched for information on the Navtech CTS350 radar and found that its elevation beamwidth is approximately 1.8 degrees. With this information, we can make some basic estimates about the radar installation position: a height of around 1.6 meters and a 3D scanning range.

Operating Frequency	76 - 77 GHz
Range Resolution	0.175m
Azimuth Beamwidth	1.8°
Elevation Beamwidth	1.8°
Infill Elevation Beam	Standard
Measurement Rate	400 per rotation
Field of View	360°
Data Connection	1 Gbps Ethernet
Power Consumption	24 Watts

iii. Map modified (1.6 < z < 4.2)

With the above two estimates, we can now compare the transformed radar data with the map features to find the optimal point cloud of map for correcting the height.

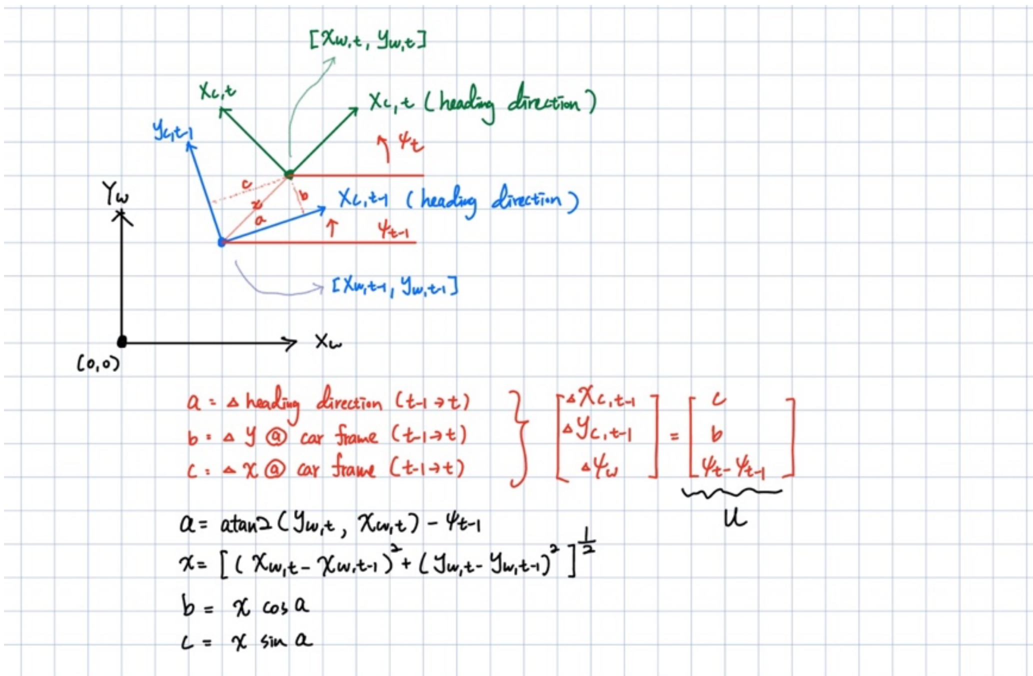


4. Others (Future works)

a. Kalman filter

Although the project deadline das passed, I am considering incorporating Kalman filter to enhance overall robustness.

Additionally, using grid search ICP places high computational demands on the computer, so I use the nonlinear motion model to predict the car’s motion and introduced GPS for update the pose, I hope this approach will improve performance.



predict

```cpp
Eigen::Vector3d predict(const Eigen::Vector3d& u) {
    // Base on the Kalman Filter design in Assignment 3
    // Implement a linear or nonlinear motion model for the control input
    // Calculate Jacobian matrix of the model as A
    // u = [del_x, del_y, del_yaw]

    //setting the random noise R
    //setting the random noise R
    R(0, 0) = 0;
    R(1, 1) = 0;
    R(2, 2) = 0;

    R = R * 1;

    B << std::cos(pose[2]), -std::sin(pose[2]), 0,
         std::sin(pose[2]), std::cos(pose[2]), 0,
         0, 0, 1;  // setting the motion transition matrix

    A << 1, 0, -std::sin(pose[2]) * u[0] - std::cos(pose[2]) * u[1],
         0, 1, std::cos(pose[2]) * u[0] - std::sin(pose[2]) * u[1],
         0, 0, 1;  // setting the jacobian matrix

    pose += B * u; // motion model
    S = A * S * A.transpose() + R;    // state

    return pose;
}
```

update

```cpp
Eigen::Vector3d update(const Eigen::Vector3d& z) {
    // Base on the Kalman Filter design in Assignment 3
    // Implement a linear or nonlinear observation matrix for the measurement input
    // Calculate Jacobian matrix of the matrix as C
    // z = [x, y, yaw]

    //setting the random noise S
    Q(0, 0) = 2.25;
    Q(1, 1) = 2.25;
    Q(2, 2) = 0.44;

    Q = Q*10;

    // I choose the linear model to update the pose & state

    Eigen::Matrix3d K = S * C.transpose() * (C * S * C.transpose() + Q).inverse();
    pose = pose + K * (z - C * pose);
    S = (Eigen::Matrix3d::Identity() - K * C) * S;

    return pose;
}
```

b. Computational power issue

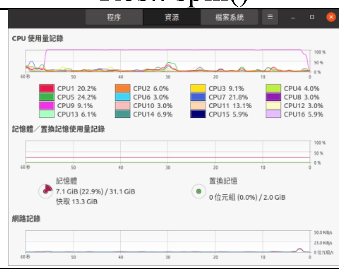
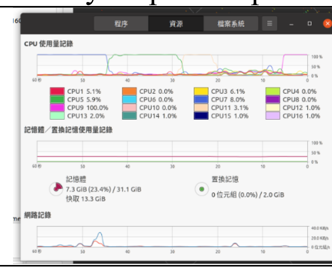
When running my code on my own computer, I noticed that the update speed was causing ICP to fall behind. In attempt to address this issue, I experimented with reducing the playback speed to at least 0.5 times to the normal speed for ICP to keep up. This limitation poses a significant drawback in real-time system. Consequently, I explored various methods to optimize the overall computational speed.

(a). multi-threads

“ros::spin()” is a traditional way of spinning the ROS message processing loop, it’s suitable for single-threaded applications where the main thread is dedicated to ROS processing.

“ros::AsyncSpinner” allows us to specify the number of threads to use for spinning, enable concurrent processing of callbacks. It’s useful in scenarios where you want to handle multiple callbacks simultaneously.

But I’ve noticed that the high computational demands are primarily concentrated in the ‘radar_callback’. Therefore, this modification doesn’t prove to be very effective.

Ros:: spin()	Ros::AsyncSpinner spinner(4)
	

(b) GPU

And I found through online research that ICP can be supported by GPU parallel computing. With this capability, it becomes feasible to perform real-time computations directly on the vehicle.

ICP算法加速优化--多线程和GPU

原创 给算法爸爸上香 已于 2023-05-01 18:26:26 修改 阅读量2.1k 收藏 26 点赞数 6 版权

分类专栏： CUDA PCL 6D pose estimation 文章标签： 点云 ICP算法 OpenMP CUDA

CUDA 同时被 3 个专栏收录 4 订阅 24 篇文章 订阅专栏

LZ之前的文章ICP算法实现（C++） 用C++实现了基础的ICP算法，由于该算法是一种迭代的优化算法，里面含有大量循环操作以及矩阵运算，可以通过使用多线程或者GPU硬件来进行加速，具体分别可以通过OpenMP和CUDA编程实现。这里给出的代码是根据github地址：https://github.com/alex-van-vliet/icp的代码改写的。原作者的代码质量还是不错的，有许多值得借签和学习的地方。但是考虑到使用的第三方库太多不便于配置和使用，LZ把这份代码重构了一下。原作者在代码里造了很多轮子，比如自己实现了Point3D、matrix以及vp-tree（也是一种搜索树，比PCL库中的kd-tree出现时间略晚）的数据结构，但是SVD分解还是调用了Eigen库（可能他也觉得底层实现太麻烦了吧~），LZ把这里面的矩阵结构统一用Eigen库实现了。另外去掉了一些不方便编译的第三方库，并简化了CmakeLists的内容以及程序的结构。如果只编译libcpu库只需依赖Eigen，编译libgpu库的话需要CUDA。

该工程的运行效果如下所示：
lib_icp.exe line1.pcd line2.pcd --cpu 100 1e-6 32 （CPU单核，0ms）
lib_icp.exe line1.pcd line2.pcd --cpu 100 1e-6 32 （CPU多核，2ms）
lib_icp.exe line1.pcd line2.pcd --gpu 100 1e-6 1024 （GPU，81ms）
lib_icp.exe bunny1.pcd bunny2.pcd --cpu 100 1e-6 32 （CPU单核，1407ms）
lib_icp.exe bunny1.pcd bunny2.pcd --cpu 100 1e-6 32 （CPU多核，246ms）
lib_icp.exe bunny1.pcd bunny2.pcd --gpu 100 1e-6 1024 （GPU，156ms）
lib_icp.exe horse1.pcd horse2.pcd --cpu 100 1e-6 32 （CPU单核，12585ms）
lib_icp.exe horse1.pcd horse2.pcd --cpu 100 1e-6 32 （CPU多核，1603ms）
lib_icp.exe horse1.pcd horse2.pcd --gpu 100 1e-6 1024 （GPU，363ms）
Release模式下编译，测试平台为Windows10系统，内存32G，CPU是i7-12700（20线程），GPU是NVIDIA GeForce RTX 3070 Laptop GPU（显存8G，每个block最大线程数为1024），计算的运行时间为三次求平均值。修改后的代码在linux系统也能编译，本人在WSL（Ubuntu20.04）的docker中测试过和Windows本地差别不大，但是调用CUDA版本代码计算结果有误。line1.pcd和line2.pcd点数为10，bunny1.pcd和bunny2.pcd点数为35947，horse1.pcd和horse2.pcd点数为193940；另外LZ调用PCL库计算这三组点云ICP配准耗时分别为0ms，758ms，30550ms。可以看出多核和GPU的加速作用随着点云点数增加优势还是非常明显的，但是点数非常少的话，GPU由于需要和CPU进行数据传输，此时运算速度会不太理想。

c. Moving Average Filter

In order to address the slight deviation in the y-direction caused by errors in ICP matching for the vehicle, I once applied a Moving Average Filter to average out the variations in the y-direction with past values. I believed this approach was reasonable when the vehicle was moving in a straight line since there is typically no y-directional movement during straight-line motion. However, I discovered issues when the vehicle encountered turns. Consequently, I ultimately decided not to proceed with this solution.

```cpp
class MovingAverageFilter {
private:
    std::vector<double> bufferX;
    size_t windowSize;
    double sumX;

public:
    MovingAverageFilter(size_t initialWindowSize = 2) : windowSize(initialWindowSize), sumX(0.0) {
        bufferX.reserve(windowSize);
    }

    double update(double newValue) {
    // Add the new value to the buffer
    bufferX.push_back(newValue);
    sumX += newValue;

    // If the buffer size is below a certain threshold, return the raw value
    if (bufferX.size() < windowSize) {
        return newValue;
    }

    // If the buffer size exceeds the window size, remove the oldest value
    sumX -= bufferX.front();
    bufferX.erase(bufferX.begin());

    // Calculate and return the moving average
    return sumX / bufferX.size();
    }
};
```

5. Reference

- https://github.com/ori-mrg/robotcar-dataset-sdk
- https://github.com/ori-mrg/robotcar-dataset-sdk
- https://arxiv.org/pdf/2211.02445.pdf