**第十九届全国大学生智能汽车竞赛**

**室外ROS无人车竞速赛（高教组）**

**技 术 手 册**

学 校：广东工业大学先进制造学院

队伍名称：Moss

参赛队员：游蕙铭、陈子瑞、李嘉、廖卓远、盛童

带队教师：苏畅、曾莹

关于技术手册使用授权的说明

本人完全了解全国大学生智能汽车竞赛关于保留、使用技术手册和研究论文的规定，即：参赛技术手册内容著作权归参赛者本人，比赛组委会和赞助公司可以在相关主页上收录并公开参赛作品的设计方案、技术手册、源码以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：

带队教师签名：

日 期： 2024.\*\*.\*\*

（本页需要签字后，替换为扫描页）

**技术手册撰写注意事项：**

了解到实验室队伍老生带新生容易断带、知识不能很好的积累沉淀转化等痛点，本次技术手册的内容更加聚焦于技术总结，知识整理。力图将技术手册打造为老队员、新队员、指导老师之间共用的知识纽带。通过几年的打磨，便于带队老师开设自动化控制、自动驾驶、人工智能、软件编程等相关课程以及开展新队员的筛选工作，便于新队员更快的传承已有的知识储备，使得实验室队伍建设构成闭环。

技术手册内容主要是将智能车作为硬件载体，以自动化控制、自动驾驶、人工智能、软件编程等专业为导向，构建一套用于学习的实验体系。书写的内容包括：

(1)参赛队员在备赛期间，结合比赛要求，结合机械、电控、编程等课本上的知识对智能车进行量化测试、局部优化的内容;

(2)参赛队员在比赛规则以外的其他方面结合智能车做的一些技术探索;

(3)参赛队员在实现比赛功能过程中，做的技术探索。

技术手册书写注意事项如下：

（1）提交形式为**实验指导书+源码；**

(2) 所有内容，需要以智能车作为唯一硬件载体；

(3) 提供不少于5个实验，以**实验指导书+源码**的形式提交；

(4) 每一个实验都要保证可以在**原版智能车**(学校最初拿到车时候的硬件、系统)上进行复现测试；

(5) 实验指导书书写时需要体现如何调用提供的源码，便于复现验证；

(6) 实验内容不得照搬原有智能车使用手册和视频资料，若与已有知识储备内容完全重复，视为该实验未书写；

技术手册总分30分，评判标准如下：

(1)每个实验最高分6分，总体不少于5个实验；

(2)每个实验得分根据实验难易程度、书写清晰度、实验间是否有关联性、创新性等因素进行判定；

(3)若书写多于5个实验，逐个实验判分后，以分数最高的5个实验的成绩加和作为最终成绩。

### 目录

实验一 雷达数据分析，确定锥桶坐标实验 15

实验二 激光雷达循迹实验 22

实验三 激光 SLAM 建图 28

实验四 导航点获取 32

实验五 第一圈与第二圈衔接处理实验（循迹与导航的切换 ） 37

实验六 自主导航 41

实验七 停车点识别 46

[实验八 地图滤波和锥桶连线](#_bookmark0) 58

... ...

### 实验一 雷达数据分析与锥桶坐标获取实验

#### 一、实验目的：

1. 掌握激光雷达测量数据的获取和基本处理方法
2. 为后续基于激光雷达的路径跟踪（循迹）功能打下基础

#### 二、实验内容：

1. 激光雷达测量数据的理解：熟悉激光雷达传感器提供的数据类型和数据结构，掌握各字段的含义及其在路径规划和障碍物检测中的作用。
2. 雷达数据处理与锥桶坐标的获取：通过处理激光雷达扫描数据，识别并提取锥桶的坐标位置，为智能车的路径跟踪算法提供所需的环境信息 。

#### 三、实验仪器：

ROS智能车：实验所使用的硬件平台，配备ROS（机器人操作系统）用于控制和数据通信。

激光雷达：主要传感器，用于测量周围物体的距离和位置。

计算机：数据处理和程序开发的工作平台，运行ROS系统并执行算法。

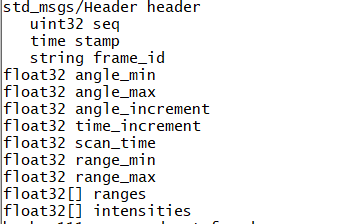
#### 四、实验原理：

激光雷达传感器将环境中的物体通过扫描方式转化为极坐标信息，并以 LaserScan 消息格式传入系统。实验中，通过读取并解析该消息内容，提取锥桶在环境中的坐标位置。利用这些数据，为路径规划及避障提供输入，最终实现智能车的循迹功能。

### 五、实验步骤：

**<1>** 了解激光雷达测量数据。

在激光雷达系统中，传入的测量数据类型为 sensor\_msgs 包中的 LaserScan 消息格式。可以通过在终端中运行命令 rosmsg show sensor\_msgs/LaserScan 来查看其结构定义。



（图 1-1）

如图 1-1 所示。该消息类型包含以下关键字段：

· Header header：消息的头部信息，包含三个子字段 uint32 seq（序列号）、time stamp（时间戳）、frame\_id（坐标系框架）。这些头部信息通常用于数据同步或标识数据来源，但对本实验的主要内容不涉及，因此这里不做详细说明。

· float32 angle\_min：表示激光束的最小角度（单位：弧度），即雷达扫描的起始角度。

· float32 angle\_max：表示激光束的最大角度（单位：弧度），即雷达扫描的终止角度。

· float32 angle\_increment：激光束之间的角度增量（单位：弧度），每两个连续扫描点的角度间隔，用于计算每个激光束相对于起始角度的位置。

· float32 time\_increment：每个激光束扫描之间的时间增量（单位：秒），表示相邻激光束测量的时间差。

· float32 scan\_time：完整扫描所需的时间（单位：秒），即雷达完成一轮扫描所用的时间。

· float32 range\_min：激光测量的最小距离（单位：米），通常用于过滤掉小于该距离的噪声数据。

· float32 range\_max：激光测量的最大距离（单位：米），用于过滤掉大于该距离的异常数据。

· float32[] ranges：按角度顺序排列的激光束距离值数组，表示每个激光束到检测物体的距离。

· float32[] intensities：按角度顺序排列的激光束强度值数组，表示每个激光束的反射强度。

在上述数据中，对于本实验的主要目标，我们重点使用的是以下三个字段：

· angle\_min：起始扫描角度，用于确定第一个测量点的角度位置。

· angle\_increment：每个激光束之间的角度增量，用于计算各测量点的相对角度。

· ranges：距离值数组，存储了每个激光束测量到的距离信息，为误差计算和锥桶识别的主要数据来源。

这些字段提供了激光雷达在极坐标系中的测量信息，借助 angle\_min 和 angle\_increment 可以确定每个测量点的角度位置，再结合 ranges 数组的距离值即可计算各测量点的精确坐标。

**<2>** 处理雷达数据，获取锥桶基于小车的直角坐标系坐标。

遍历雷达传入数据，找到距离小车 2 米范围内的所有锥桶。

当激光雷达扫描到锥桶时，通常会呈现出两个显著特征：一是距离出现突变，二是突变后的距离数据具有一定的连续性。基于这两个特征，我们可以遍历激光雷达传入的数据，识别符合这些特征的点来确定锥桶位置。

首先，根据距离突变的特性，我们遍历 ranges 数组，寻找满足 ranges[i-1] - ranges[i] >= 2.0 的数据（即距离突然缩短超过 2 米则认为是距离突变）。为保证突变数据的有效性，还需对 ranges[i] 进行筛选，要求 ranges[i] 小于设定的最大距离值 ranges\_maxvalue。由于户外环境复杂且雷达扫描范围较大，距离突变大于 2 米的情况可能频繁出现，设定 ranges\_maxvalue 可以有效减少后续筛选的计算量。实验表明，ranges\_maxvalue 设定在 2.0 米左右较为适宜。

接下来，我们根据连续性特征进一步筛选突变后的数据。定义距离连续性为相邻数据在一个小范围内波动。具体而言，我们认为当 ranges[i] 和 ranges[i+1] 的差值绝对值小于 0.2 时，即满足一次连续。在此基础上，从起始点 i 开始，遍历 ranges[i] 至 ranges[i+bucket\_threshold]，逐一计算 abs(ranges[i] - ranges[i+idx])，若该值小于 0.2，则视为一次连续。经过实验验证，bucket\_threshold 设置为 10 时效果最佳。

在遍历这 10 个数据的过程中，若要求所有数据都满足连续性标准，可能会因数据干扰而导致漏检。特别是激光雷达在强光或高温等干扰条件下，数据可能存在波动，因此，我们只需在 10 个数据中满足设定的连续次数 vaild\_bucket\_threshold，即可判定锥桶位置。经多次实验，vaild\_bucket\_threshold 设置在 5 到 7 之间效果较佳。

1.获取锥桶基于小车的极坐标系坐标。

假设激光雷达位于原点位置，从上方俯视，激光雷达以逆时针方向旋转，且将小车的正前方定义为极坐标系的极轴方向。基于此，我们建立了一个以小车为中心的极坐标系。

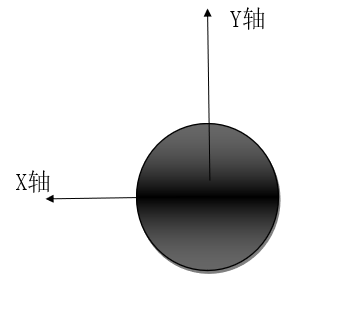
在这个极坐标系中，激光雷达扫描的每个点可由极径和极角来描述。极角 theta 可通过以下公式计算：

其中，anglemin 表示激光雷达的最小扫描角度，angleincrement 表示每个扫描点之间的角度增量，i 为扫描点的索引。

相应地，扫描点的极径 range 即为 ranges[i]，代表了从激光雷达到物体的距离。通过该坐标系，我们可以将每个扫描点的极坐标数据转换为直角坐标，更方便地对锥桶等障碍物的相对位置进行分析和处理。

2.将极坐标系坐标转化为直角坐标系。

假设激光雷达位于坐标原点，我们建立一个平面直角坐标系，其中小车的正左方定义为 x 轴的正方向，而正前方定义为 y 轴的正方向。



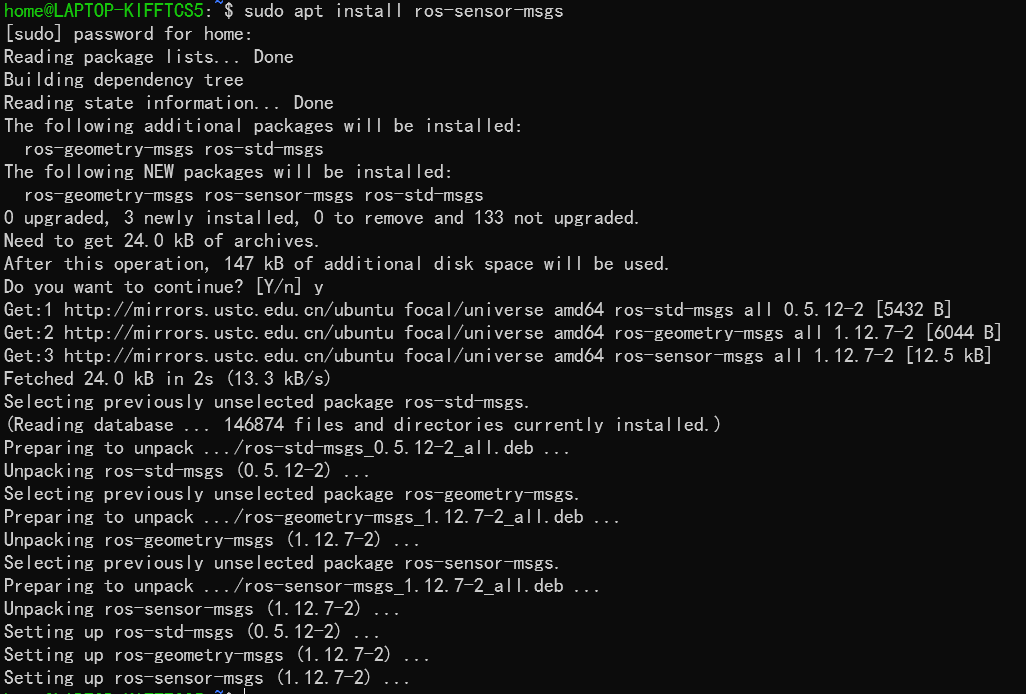
（图 1-4）

如图 1-4 所示，激光雷达扫描获得的每个点的坐标可以通过以下公式进行转换：

其中，range 表示从激光雷达到该点的距离，而 theta 为扫描点的极角。通过这一坐标转换，我们能够将激光雷达的极坐标数据转化为更直观的平面直角坐标系表示，以便进一步分析小车周围物体的相对位置。

**<3>** C++代码实现锥桶坐标获取。

首先输入以下命令，并输入密码，安装 sensor\_msgs 依赖包：



（图 1-5）

安装完成后，编写 C++代码

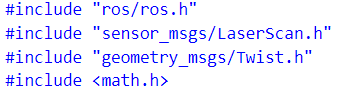
由于本实验中处理完激光雷达传入数据之后需实现在回调函数中发布数据， 因此需要定义一个类如下：

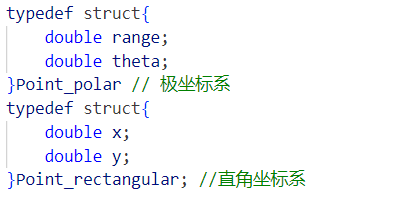


其中，/scan 为雷达话题，/car/cmd\_vel 为小车底盘控制话题。

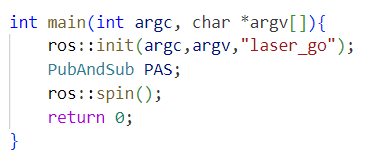
随后包含头文件、定义所需变量：

* 编写 main 函数如下：





编写 main 函数如下：



随后进行回调函数的内容编写，如下：



代码文件：laser\_go.cpp

使用方法：

1. 打开一个终端，先运行 roslaunch racecar Run\_car.launch 运行节点。
2. 再打开另一个终端，到代码文件夹下，运行 rosrun laser\_go.cpp，就能够正常输出 2 米内的锥桶在所建平面直角坐标系中的坐标了。

### 实验二 激光雷达循迹实验

#### 一、实验目的：

1、掌握使用激光雷达识别锥桶并进行循迹的方法。

2、ROS无人车完成比赛第一圈的循迹任务。

#### 二、实验内容：

1. 处理激光雷达采集到的锥桶坐标信息，并对锥桶进行分类分组。
2. 利用分类后的锥桶数据进行无人车循迹操作，确保无人车沿既定路线行驶。

#### 三、实验仪器：

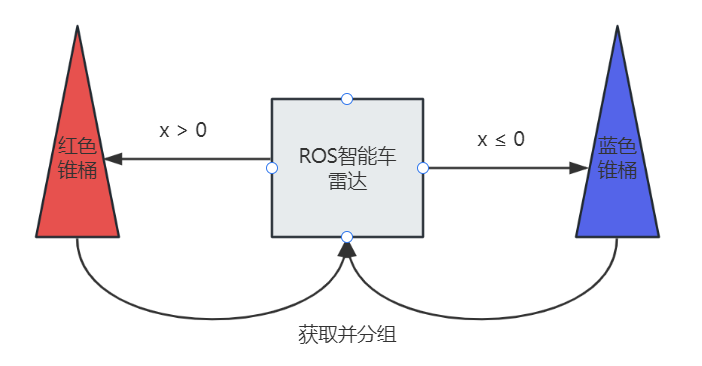
ROS智能车、激光雷达、电脑

#### 四、实验原理：

通过激光雷达采集锥桶的坐标信息，判断锥桶在空间中的位置，并以此作为循迹导航的依据。通过对不同锥桶位置的识别和追踪，可以确保无人车精准绕过障碍物，沿目标路径行驶。

#### 五、实验步骤：

<1> 将识别到的锥桶按照颜色进行分类，分为红色锥桶和蓝色锥桶两个组。

****

(图 2-1)

如图2-1所示，当小车沿直线路段移动时，我们可以对锥桶的初始位置进行筛选，以便实现初步的循迹导航。在直线路段上，假定红色锥桶位于小车的左侧，蓝色锥桶位于右侧。以小车的当前位置为基准，设x=0轴为小车所在的直线。将所有x>0的锥桶坐标归入红色锥桶数组red\_p[]中，而将x<0的锥桶坐标归入蓝色锥桶数组blue\_p[]中。

然而，当小车从直线路段进入弯道或S形弯道时，锥桶的分布可能偏离直线路段的初始假设，出现红、蓝锥桶相互交错的情况。例如，可能有一部分蓝色锥桶的x坐标大于0，或部分红色锥桶的x坐标小于0。为确保循迹任务的准确性，我们需要对初筛后的红、蓝锥桶数组进行进一步的验证和调整。然而在小车从直线路段进入弯道或S形弯道时，锥桶的分布可能会偏离上述假设，出现红蓝锥桶位置相互交错的情况。例如，可能会有部分蓝色锥桶的x坐标大于0，或部分红色锥桶的x坐标小于0。为了修正这种情况，我们需要对初筛后的红蓝锥桶数组进行进一步验证。

验证的核心原则是基于锥桶分布的区域来推测小车行驶的路径边界：

1. 蓝色锥桶偏离：若蓝色锥桶出现在x>0区域，则说明小车的正前方可能指向赛道的外边界；
2. 红色锥桶偏离：若红色锥桶出现在x<0区域，则表明小车正前方可能指向赛道的内边界。

需注意的是，这两种情况不会同时发生，因此我们只需对锥桶数量较多的数组进行验证，以确定是否存在误分组现象。

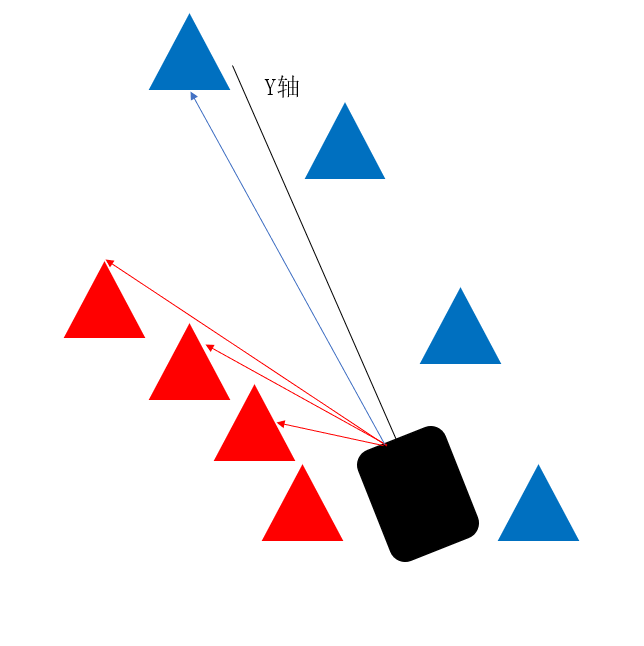
验证步骤的关键在于选择一个“标准锥桶”，即在该数组中位置最符合预期且误分组概率最低、可信度最高的锥桶。以“标准锥桶”为基准，基于其与其他锥桶的相对距离，进一步判断和校正误分组的锥桶，确保红蓝锥桶分布的准确性，为小车的循迹导航提供更可靠的数据支持。

因此为提高数据准确性，我们仅选取小车后方一定距离至车前方的锥桶数据进行分析。这一距离较短，确保采集的锥桶位于小车路径的直接影响范围内。根据激光雷达的扫描顺序，从近至远获取锥桶数据时，红色锥桶的下标从大到小排列，而蓝色锥桶的下标则从小到大排列。

在常规情况下，离小车更近的锥桶（此处指y坐标绝对值最小的锥桶，而非实际距离）数据误差更小，因而在选择“标准锥桶”时，我们优先选择与小车位置最近的锥桶作为参照。

1. 红色标准锥桶：选择下标为j-1（即红色锥桶数组中下标最大的锥桶）作为红色标准锥桶。
2. 蓝色标准锥桶：选择下标为0（即蓝色锥桶数组中下标最小的锥桶）作为蓝色标准锥桶。

通过这种标准锥桶的选择方法，我们能以相对稳定的锥桶位置数据为基准，进行进一步的循迹路径校正和分析，从而确保小车在实际行驶中依据准确的位置信息保持在指定赛道内。



（图 2-2)

如图2-2所示，直道进入弯道的情形是容易出现锥桶误分组的典型情况。在这种情况下，初步筛选出的红色锥桶数组可能包含蓝色锥桶，导致数据偏差。例如，在此图示情景下，红色锥桶数组red\_p[]包含4个元素，但我们实际知道其中下标为0的元素是一个蓝色锥桶。为修正此类误分组，需要基于标准锥桶选取原则对数据进行进一步验证。

1. 依据标准锥桶的选取规则，我们选择red\_p[3]（即下标最大的位置）作为红色标准锥桶，作为后续校正的参考点。
2. 为了检查数组中的锥桶是否被误分组，我们采用距离阈值判断的方法。首先计算相邻两个锥桶的距离：
   1. 计算red\_p[2]与red\_p[3]的距离，计算方式为：

若距离小于1.4米，则认为red\_p[2]是正确分组的红色锥桶。

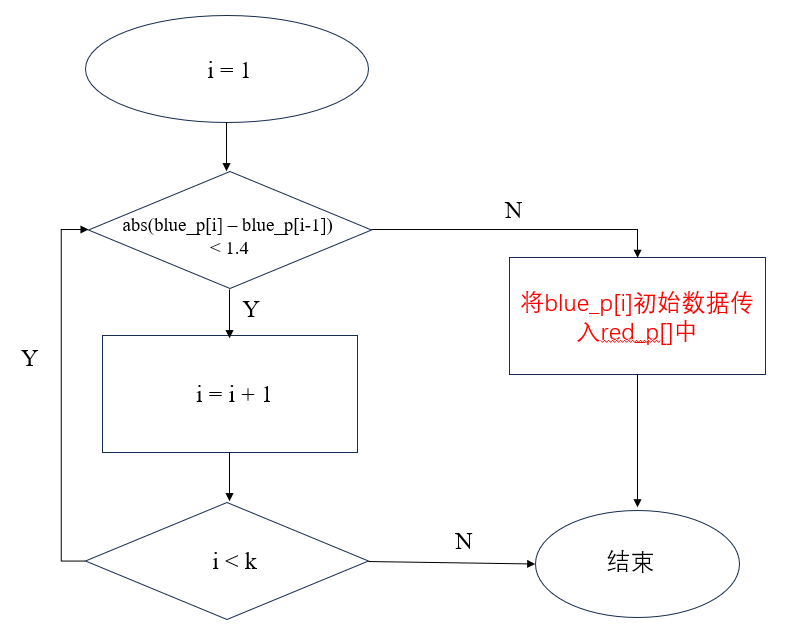
* 1. 继续递推计算red\_p[1]与red\_p[2]的距离，若此距离依然不超过1.4米，则red\_p[1]也是正确分组的锥桶。
  2. 最后计算red\_p[0]与red\_p[1]的距离。在该实例中，red\_p[0]与red\_p[1]的距离超过1.4米，表明red\_p[0]实际上并非红色锥桶，而是误分组的蓝色锥桶。

1. 由于激光雷达扫描方向不变，且赛道边界是连续的，一旦确认了red\_p[i]为误分组的锥桶，意味着red\_p[0]到red\_p[i]的所有锥桶均为误分组。同理，若蓝色锥桶数组blue\_p[]中的某个锥桶被确认误分组，那么从blue\_p[i]到blue\_p[k-1]的所有锥桶都应视为误分组。
2. 找到误分组的锥桶后，需要将它们重新归入正确的锥桶数组。为保持数据排列的特性（即“从近到远，红色锥桶下标从大到小，蓝色锥桶下标从小到大”），更新数组时应遵循以下规则：

· 将被误分组的红色锥桶添加至红色锥桶数组的前端。

· 将被误分组的蓝色锥桶添加至蓝色锥桶数组的末端。

此方法确保了锥桶分组数据的准确性，有助于提升小车循迹系统的导航稳定性和路径精度，为小车后续的行驶提供可靠的数据支持。



(图 2-3)

如图2-3所示，展示了一个判断和调整误分组锥桶的基本流程。此处以蓝色锥桶数组元素较多的情况为例进行说明。当红色锥桶数组元素较多时，可采用类似的流程进行处理。

在该流程中，通过逐步验证锥桶的相对位置及距离，将误分组的锥桶从当前数组中移除并归入正确的组别。最终，确保每个数组仅包含正确分组的锥桶，红色锥桶数组red\_p[]和蓝色锥桶数组blue\_p[]分别对应赛道的内、外边界。

经过上述分组及调整流程，我们已成功将赛道内外的锥桶坐标数据分为红色和蓝色两组。红、蓝锥桶数组的划分代表了小车循迹路径的内外边界，为后续循迹操作提供了精确的参考数据。这一精细分组为小车的路径导航奠定了可靠的数据基础，有助于确保其沿赛道安全行驶。

**<2>** 熟悉 PID 控制和理解 PID 控制调参思路

PID控制是一种广泛应用的反馈控制算法，用于调节和稳定系统的输出。PID代表比例（Proportional）、积分（Integral）和微分（Derivative）三个控制项，旨在通过动态调整系统的控制量，使其达到并维持在预定目标状态。

常见的PID控制方法有位置式PID和增量式PID。本实验将采用位置式PID控制，这种方式通过直接计算比例、积分和微分项的总和来调整系统的输出。位置式PID控制特别适用于需要精确控制的位置调节任务。在舵机控制应用中，PID的输出是这三个部分输出量的加权总和，以实现对舵机精确的角度调整和稳定控制。

PID算法的公式：

将其离散化，得到

比例项：

积分项：

微分项：

（图 2-5）

如图2-5所示，离散化后的位置式PID控制公式为：

其中，ek表示当前时刻的误差，esum(k)为误差的累积和，Kp、Ki和Kd分别为比例、积分和微分控制的系数，u(k)是控制输出。

PID 参数的基本调节方法

1. Kp（比例系数）：
2. 作用：在PID算法中，比例系数的主要作用是决定系统响应的速度。增大 Kp ​ 可以提高系统对误差的响应速度，从而使小车能够更快地调整位置，达到目标状态。同时，适当增大 Kp 还可以消除静态误差。
3. 调节方法：当小车反应速度较慢时，应适当增大 Kp 以提高响应速度；当小车出现明显的左右摆动（即系统超调过大）时，应减小 Kp ，以避免过度反应。需要注意的是，Kp 过大会导致系统过度敏感，从而出现振荡或不稳定。
4. Ki（积分系数）：
5. 作用：积分项主要用于消除静态误差，尤其是在小车转弯过程中，能够帮助小车纠正长期积累的偏差。它对误差的积累进行加权求和，从而逐步减少偏差，改善小车的精度。
6. 调节方法：当小车转弯时，若发现容易碰撞到赛道的外边界，说明当前的积分系数可能过小，此时可以增大 Ki 来提高纠正能力；反之，如果小车在转弯后反应速度明显降低或偏离内圈，则应减小 Ki，以防止积分作用过强导致反应迟钝。实际应用中，为了避免积分项导致误差的无限累积，通常会对积分值进行限幅，防止过大的累积误差影响系统的响应。
7. Kd（微分系数）：
8. 作用：微分项用于预测误差的变化趋势，能够抑制系统的震荡，减少过度调整。微分系数 Kd 通过测量误差的变化速率，帮助系统做出更加平滑的调整。
9. 调节方法：通常，微分系数的调节对系统的稳定性有很大影响。若发现小车在行驶过程中存在过大的反应或震荡，可以适当增大 Kd​ 来平滑响应；相反，若系统反应迟缓且有滞后感，则可以减小 Kd 以提高响应的灵敏度。

PID控制算法的参数调节是一个需要不断试验和优化的过程。通过合理调节Kp、Ki和Kd，可以使小车在循迹过程中更加稳定、精确。调节过程中需要根据小车的具体表现，逐步优化参数，以实现平稳、高效的控制。

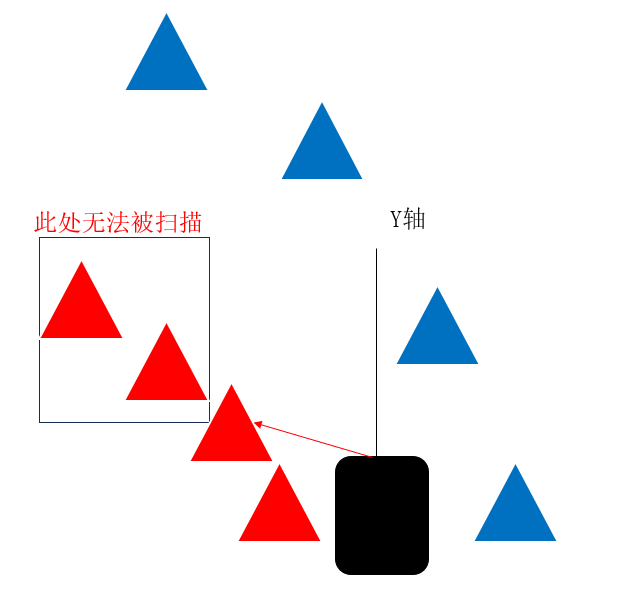
**<3>** 掌握获取位置偏差的方法。

在步骤<1>中，我们已将锥桶坐标分为红色和蓝色两组。通常情况下，扫描到的红色和蓝色锥桶的数量是不相等的，尤其是在非直道行驶时。因此，为了便于计算位置偏差，我们需要选择等量的红色和蓝色锥桶用于误差计算。

为了确保误差计算的准确性，选取等量的红蓝锥桶有两种直接可行的方案：

1. **舍弃多余的锥桶**： 在这种方案中，我们只取数量较少的锥桶数组中的元素，确保红、蓝锥桶的数量相等。具体做法是从两组中选择数量较少的锥桶数目，然后从数量较多的锥桶数组中去除一些锥桶。为了确保所选锥桶最具代表性，优先从离小车较近的锥桶中选取：红色锥桶优先选择下标较大的元素（即靠近小车的锥桶），蓝色锥桶优先选择下标较小的元素（即离小车较近的锥桶）。这种方法确保了所选锥桶数据的及时性与代表性。
2. **补充较少的锥桶数组**： 对于数量较少的锥桶数组，我们可以通过复制元素来补充，直到两组锥桶数量相等。补充的方式是将数量较少的锥桶数组中的元素复制，使其数量与较多的锥桶数组相匹配。需要注意的是，我们优先选择最远的锥桶数据进行复制，因为通常情况下，数量较少的锥桶数组之所以元素较少，往往是因为一些远距离的锥桶由于遮挡（例如近距离锥桶阻挡了远距离锥桶的激光雷达扫描）未能被扫描到。因此，补充的锥桶数据通常来自于离小车较远的部分，这种方式能够尽量减少因补充数据带来的误差。

通过这两种方案，我们可以确保在计算位置偏差时，使用的红色和蓝色锥桶的数量一致，从而获得更加准确的偏差计算结果。无论采用哪种方案，最终目标是保证计算过程中各组锥桶数据的均衡性和代表性，为后续的控制和调整提供可靠的依据。



（图 2-7）

在前述的两种锥桶选取方案中，方案1的优点在于锥桶坐标的准确性较高，计算结果更可靠，但其缺点是没有充分利用所有扫描到的锥桶数据。方案2的优点则在于充分利用了更多的锥桶数据，提升了数据的完整性，但可能在计算过程中引入更多误差。因此，为了平衡这两者的优缺点，结合它们的优点，我们提出了一个更为合理的锥桶选取方案，旨在最大化利用锥桶数据，同时最小化误差的引入。

1. **当某种锥桶的数量大于3个时，只保留3个距离小车较近的锥桶**： 如果扫描到的某种锥桶的数量超过3个，我们选择保留3个距离小车较近的锥桶进行计算。这样能够确保选取的数据足够准确，同时避免引入过多的远距离数据带来的误差。
2. **当某种锥桶的数量比另一种锥桶多出2个时，舍弃最远的锥桶**： 若一种锥桶的数量比另一种锥桶多出2个，个数较多的锥桶将舍弃最远的锥桶数据，以保持两个数组中的锥桶数量平衡。通过这种方式，我们避免了单侧过多数据的干扰，并且尽量减少了误差的引入。
3. **当某种锥桶的数量比另一种锥桶多出1个时，将最远的锥桶复制一份**： 如果一种锥桶的数量比另一种锥桶多出1个，则将数量较少的锥桶数组中的最远锥桶复制一份，进行数据补充。复制最远的锥桶数据有助于减少遮挡问题带来的影响，因为在遮挡情况下，两个锥桶的 x 坐标通常会非常接近，因此复制最远的锥桶不会引入太大误差。

误差计算采用加权求和的方式，具体步骤如下：

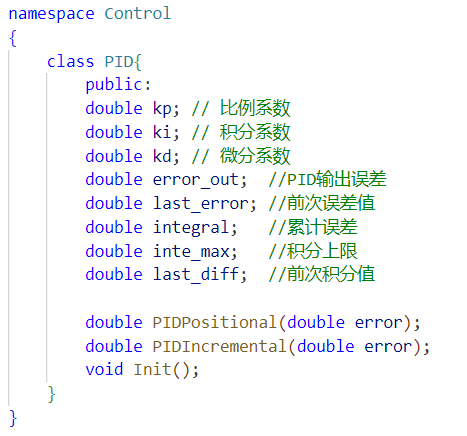
1. 确定参与误差计算的锥桶对数： 误差计算的基础是选取的等量红蓝锥桶。通过前述的锥桶选取方案，我们能够确保两组锥桶数量一致，这样就能够保证误差计算中的数据平衡。
2. 计算加权误差： 对于每一对参与误差计算的锥桶，我们首先从近到远按顺序处理，并计算每对锥桶的 x 坐标之和与相应权重的乘积。权重的设置依据具体的应用需求和系统特性来确定，通常会根据小车与锥桶的相对距离来设置权重，距离较近的锥桶权重较大，以便优先考虑小车当前路径附近的锥桶。
3. **将误差传递给PID控制器**： 最后，将计算得到的加权误差传递给PID控制器。PID控制器根据误差进行调整，从而控制舵机的动作，使小车能够准确地沿赛道行驶。

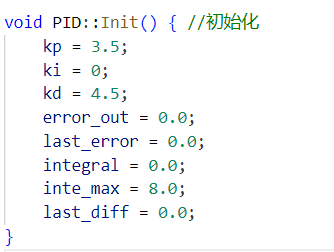
通过这种改进后的选取方案与误差计算方法，我们能够在保证数据准确性的同时，充分利用所有可用的锥桶数据，最大限度地减少误差的引入，从而提高小车循迹的稳定性与精度。

**<4>** C++代码实现激光雷达循迹。

本实验代码建立在《实验一 雷达数据分析与锥桶坐标获取实验》的基础上， 部分相同代码不再赘述，只介绍本实验新增部分代码。

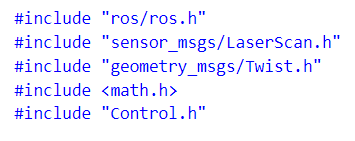
首先，编写 Control.h 头文件用于位置式 PID 控制：

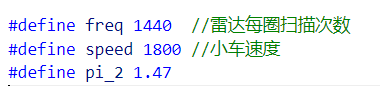


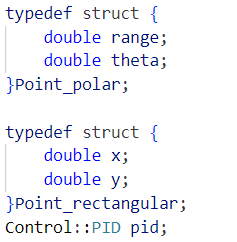


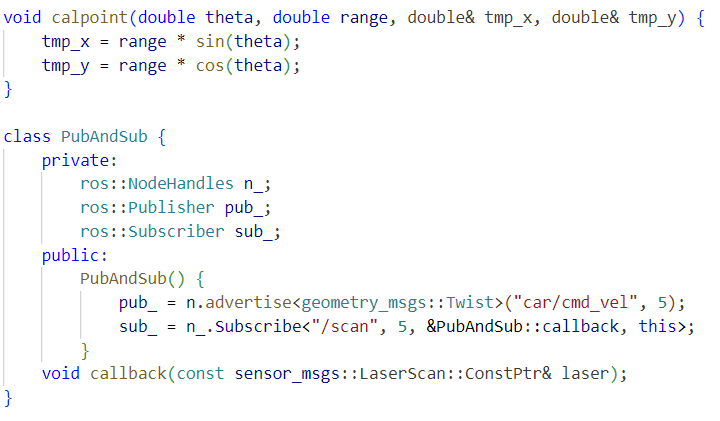


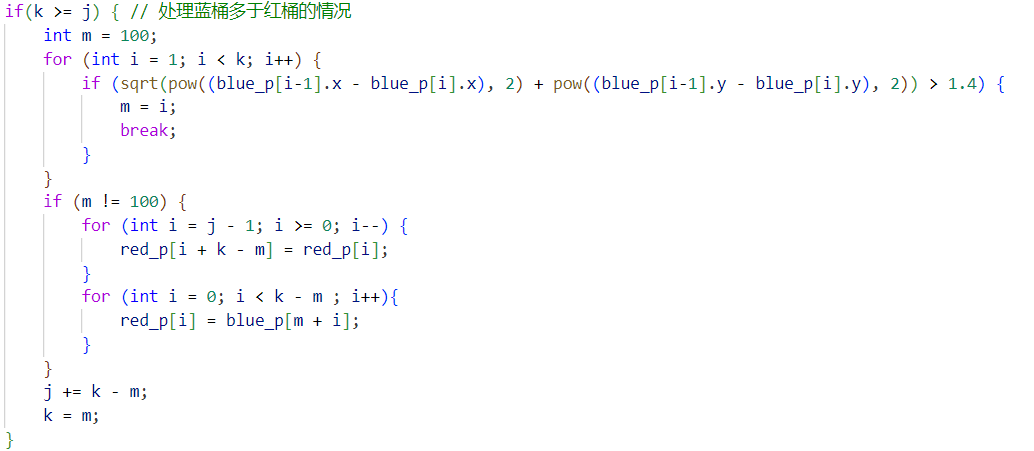
接着，补充回调函数：

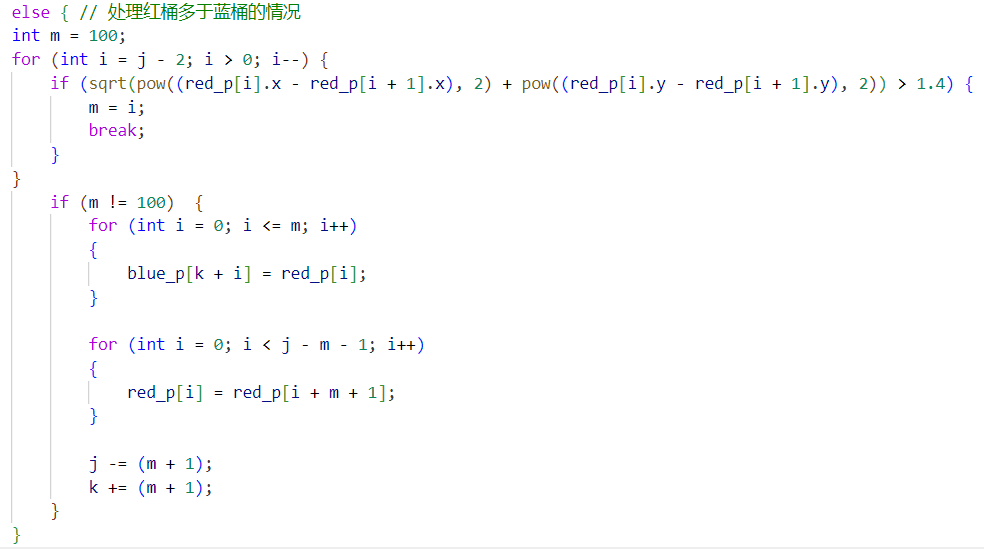


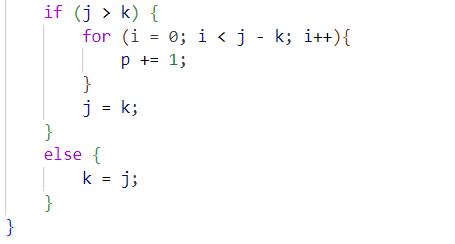












至此，代码编写完成，将小车放在赛道中，打开 Run\_car.launch 和本实验编写的节点，即可实现激光雷达自主循迹。

### 实验三 激光 SLAM 建图

#### 一、实验目的：

1、理解 gmapping 算法的原理

2、熟悉 ROS 环境下快速配置并使用 gmapping 算法

3、理解 gmapping 的关键参数

#### 二、实验内容：

1、安装 gmapping 功能包

2、运行 gmapping 节点并进行建图

3、理解 gmapping 算法各参数的作用

#### 三、实验器材：

ROS 智能车、激光雷达、电脑、IMU、编码器

#### 四、实验原理：

本实验采用 gmapping 算法进行建图，gmapping 是一种基于 2D 雷达使用 RBPF（Rao-Blackwellized Particle Filters）的激光SLAM（Simultaneous Localization and Mapping）算法，广泛用于机器人自主导航中。 gmapping 算法可以实时构建室内环境地图，可以处理复杂的动态环境，特别是在有障碍物、狭小空间或不规则环境中，表现出较好的鲁棒性。下面简要介绍一下 gmapping 算法的各方面。

①基本原理：gmapping 算法通过使用粒子滤波器（Particle Filter）来同时估计机器人的位姿和环境地图，结合了机器人的运动信息（如里程计数据）和传感器数据（通常是激光雷达数据）进行位姿估计与地图构建，以此改进位姿估计与地图的准确性。

②建图过程：在 gmapping 中，环境被表示为一个栅格地图，该地图将空间划分为多个栅格，每个栅格对应于一个空间单元，并标记为占用或空闲。 gmapping 通过结合激光雷达数据与当前地图进行逐步更新。激光雷达会扫描环境并返回一系列距离测量值，算法通过将这些测量数据与现有地图上的格子进行比对，调整相应格子的占用概率。这一过程不断进行，以逐步建立出一个完整的环境地图。

③定位过程：gmapping 利用粒子滤波器来估计机器人当前位置的位姿。初始化时，算法通过均匀采样生成一组粒子，每个粒子代表一个可能的机器人位姿（即位置和朝向的组合）。随着机器人在环境中的运动以及传感器观测数据的不断更新，粒子的权重会根据当前观测结果进行调整，以反映每个粒子代表的位姿的可信度。权重的调整基于激光雷达观测与地图匹配度的计算。随后，粒子滤波器会根据这些权重进行重新采样，将高权重的粒子保留下来，丢弃低权重的粒子，从而不断改进机器人的位姿估计。

④数据关联：gmapping使用一种称为**匹配度计算**的方法，通过比较激光雷达测量值与地图中对应位置的期望测量值之间的相似度，来评估粒子的可信度。这一过程使用**点云匹配**技术，将激光雷达获取的环境特征与地图上的预期特征进行对比，从而生成每个粒子的权重值。粒子的权重反映了其位姿与真实位置的匹配度，权重越高，表示该粒子代表的位姿越有可能是正确的。

⑤地图优化：在RBPF中，机器人的位姿由粒子滤波器估计，而地图更新则通过对每个粒子进行独立的地图优化来实现。这种方法有效地减小了计算复杂度，同时能够在不牺牲精度的情况下提高地图的质量。RBPF允许在每个粒子对应的机器人位置下单独优化和更新环境地图，从而确保在整个SLAM过程中，地图的精度随着位姿的不断优化而逐渐提高。

#### 五、实验步骤：

<1> 安装 gmapping 功能包：

通过从包安装安装 gmapping 功能包

连接互联网，在终端中输入以下命令：

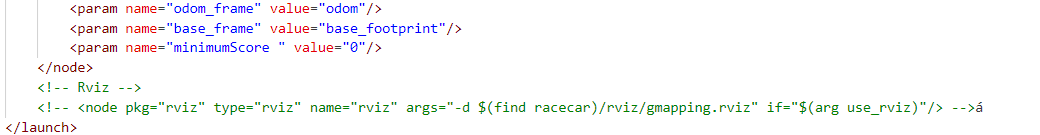
sudo apt install ros-noetic-gmapping

随后输入密码，安装 gmapping 功能包。

<2> 编写 launch 用于启动 gmapping 节点：

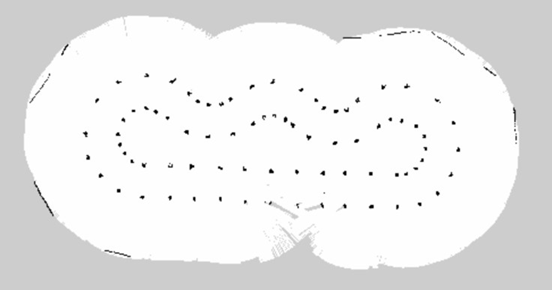
编写 launch 文件，内容如下：





该文件的作用为启动 gmapping 节点。

当配置好各传感器，获取雷达点云、里程计等信息后，启动 gmapping 节点即可开始建图，可以使用 rviz 查看建图情况。建图较好的情况如下：

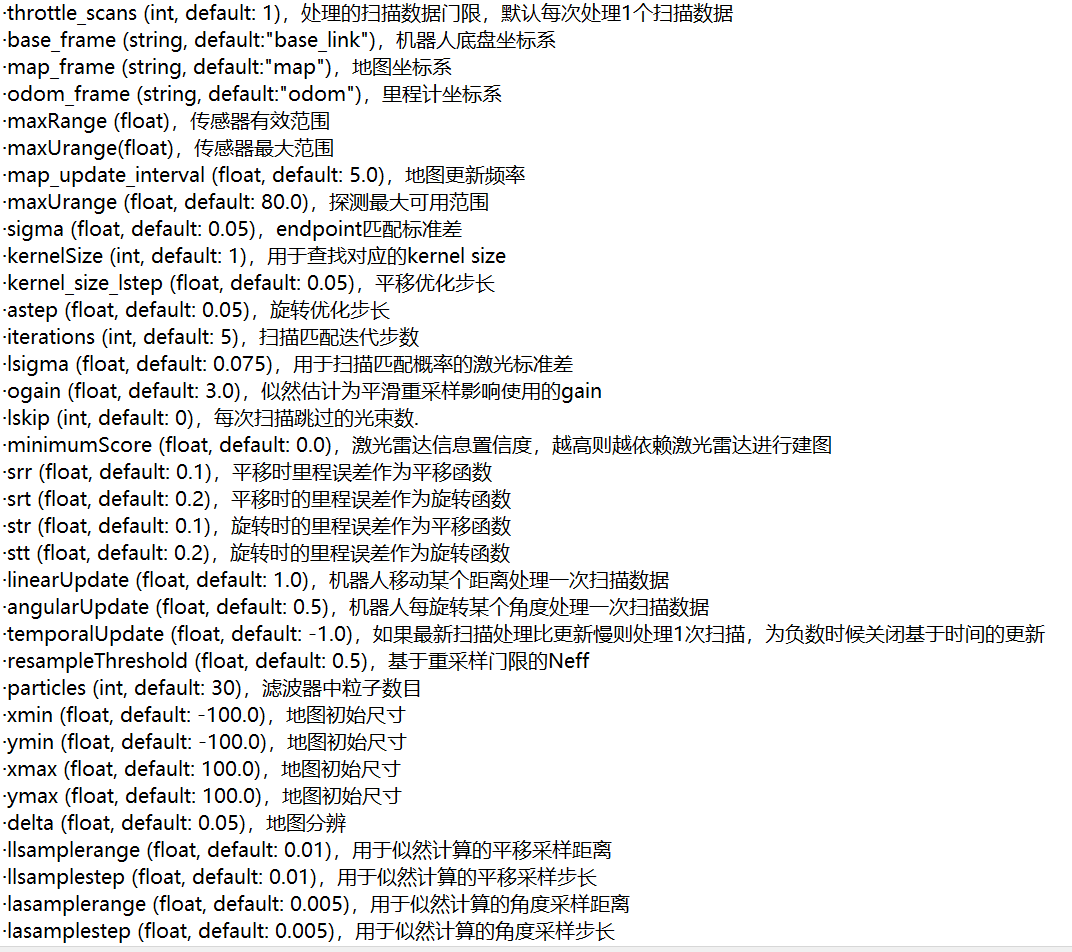


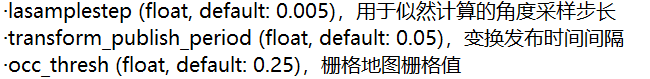
（图 4-2）

并使用rosrun map server map server -f 指定文件路径 将地图保存到指定文件路径中。

<3> 理解 gmapping 算法各参数的作用：

gmapping 具有以下参数：





其中，对建图效果影响较大的参数如下：

maxRange，该值的大小会影响建图处理的信息量和建图过程的流畅度，但越小时会越容易发生定位漂移；

particles，根据粒子滤波的原理，该值增大时会显著增加算法的运算量， 但可以提高建图的准确性。

通常，在算力有限的情况下，需要使 maxRange 和 particles 都尽量小，以获得较高的建图流畅性。但值得注意的是，这两个参数过小也会导致建图失败，需要依据实际情况进行调整。

### 实验四 导航点获取

#### 一、实验目的：

1、学习 ROS 中 tf 包

2、掌握 gmapping 建图发布的坐标变换信息

3、获取 map 坐标系下 base\_link 所在位置

#### 二、实验内容：

1、查找 gmapping 的坐标变换信息

2、利用 tf 包动态观测 base\_link 坐标系与 map 坐标系变换

3、存储获取的 map 坐标系下 base\_link 所在位置

#### 三、实验器材：

ROS 智能车、激光雷达、电脑。

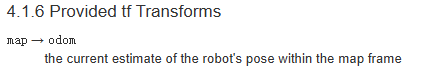
#### 四、实验原理：

在gmapping 建图时，会有五个坐标系，分别为雷达坐标系，base\_footprint 坐标系（智能车底盘为原点的坐标系），base\_link 坐标系（智能车中心为原点的坐标系），odom（里程计）坐标系和 map 坐标系（建图起始位置为原点）。但我们无法直接得到map坐标系的 base\_link的位置坐标，所以需要通过坐标变换的工具包tf包，坐标变换就能获得 map 坐标系下 base\_link 所在位置的坐标。

#### 实验步骤：

**<1>** 查找 gmapping 发布的坐标变换信息

通过 <http://wiki.ros.org/gmapping>了解 gmapping 节点的坐标变换。如图 5-1 所示，gmapping 发布 map 坐标系到 odom 坐标系的坐标变换，再加上 car\_tf.launch 文件里的 base\_footprint 和 base\_link 的坐标变换、base\_link 和雷达的坐标变换、里程计发布的 base\_footprint 和 odom 的坐标变换，就能获取 map 和 base\_link 的坐标变换。



（图 5-1）

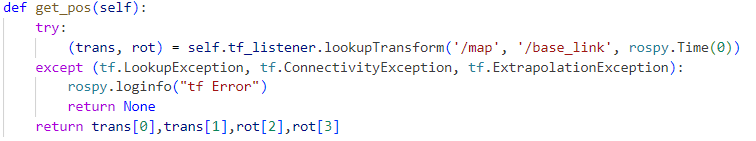
**<2>** 利用 tf 包动态观测 base\_link 坐标系与 map 坐标系变换

首先创建tf监视器，如图5-2。



（图5-2）

其中 tf 监听器监听 base\_link 到 map 坐标系的坐标变换，然后编写函数，通过 tf 监听器获取四元数与坐标，如图 5-3 所示。



（图5-3）

其中 trans为 t f监听器监测 base\_link变换 到 m a p坐标系返回的坐标，rot 为返回的四元数。智能车在赛道行驶，只有 x,y 坐标发生变化，四元数 z，w 发生变化，所以我们只用记录 x，y，z，w 就能确定智能车的具体位置。x对应函数中的trans[0]，y对应函数中的trans[1]，z对应函数中的rot[2]，w对应函数中的rot[3]。

**<3>** 存储获取的 map 坐标系下 base\_link 所在位置

在第一圈时，需要获取智能车在赛道中的位置作为第二圈的导航点，因此需要记录获取的位置。如图 5-4 所示，使用 csv 文件进行记录，并用欧式距离每隔 L 米记录一次。



（图5-4）

**<4>** 运行

以上代码编写在 get\_point.py 中。运行Run\_car.launch 文件和 Run\_gmapping.launch 文件后即可运行该脚本，首先运行 Run\_car.launch 文件和 Run\_gmapping.launch 文件，然后运行雷达寻路代码或者用键盘控制智能车运动，再运行该脚本便可以实现记录导航点功能。

### 实验五 第一圈与第二圈衔接处理实（循迹与导航的切换）

#### 一、实验目的：

1、掌握用代码运行 launch 文件和关闭节点

2、掌握线程的创建方法

3、实现循迹和导航的自动切换

#### 二、实验内容：

1、使用 roslaunch 模块和 rosnode 模块运行 launch 文件和关闭

2、使用 threading 模块开启线程

3、编写代码控制循迹和导航的自动切换

#### 三、实验器材：

ROS 智能车、激光雷达、电脑。

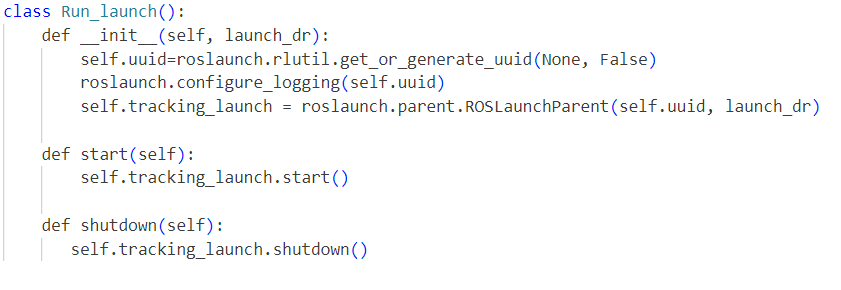
#### 四、实验原理：

第一圈主要需要运行 gmapping 和雷达循迹，而导航主要需要开启 amcl 和move base 等节点构成的 launch 文件。考虑到智能车算力资源，不能同时开启建图和导航，因此需要编写脚本实现循迹和导航切换。launch 文件中包含多个节点，一个节点运行相当开启一个进程，因此运行 launch 文件相当开多个进程。可以在脚本中使用 roslaunch 和 rosnode 模块实现对 launch 文件和节点的开启和关闭。同时，该脚本需要处理好雷达循迹和导航发布的控制信息。这要求在脚本进程中开一个子线程订阅雷达循迹还有导航发布控制信息的话题，再由该线程直接发最终的控制信息。

#### 五、实验步骤：

**<1>** 使用 roslaunch 模块和 rosnode 模块运行 launch 文件和关闭节点

roslaunch 模块可以实现launch 文件的运行和关闭，代码如下图所示， 为了方便代码复用，可以写成类的形式。



在实例化 Run\_launch 类时，需要传入 launch 文件的路径，使用 start()

方法运行 launch 文件，shutdown 方法关闭文件。

rosnode 模块可以用于关闭在运行的节点。导航开启后，为了减少资源开销，可以用 rosnode.killnode(["节点名"])关闭节点

**<2>** 协调雷达循迹和导航发布的控制信息

遇到红绿灯需要停车或者第二圈遇到停车点需要停车，因此需要协调好雷达循迹和导航发布的控制信息，需要停车时，要立刻发停车控制信息，在切换时，需要使用导航发布的控制信息。考虑到上述要求，可以在脚本进程订阅雷达循迹还有导航发布控制信息的话题，再发最终的控制信息。如下图所示， 可以创建类来实现



在 Judgement 类中，用有限状态机的思想，通过定义 state 来说明当前是使用雷达控制信息，导航控制信息还是停车，state 的变化通过红绿灯和停车点来决定。

**<3>** 创建线程

实验步骤二中的 state 的改变依赖于红绿灯和停车点，因此需要再脚本中增加 state 转移的代码，而订阅话题需要一直接收并用回调函数处理，这两段程序需要并发运行。为了便于信息共享，可以开子线程用于处理订阅话题的回调函数。首先，现在 Judgement 中增加一个实例方法，如下图所示



其中 rospy.spin()函数用于处理订阅话题的回调函数。 在 python 中，可以使用 threading 模块创建线程， 具体代码如下图所示



**<4>** 编写代码控制循迹和导航的自动切换





当识别到红绿灯时，就停下，保存建好的地图，运行导航 launch 文件，发布停下时车的位姿，最后切换到导航状态，开启导航。该脚本命名为auto\_transform.py,运行该脚本时需要打开红绿灯识别，雷达循迹。

### 实验六 自主导航

#### 一、实验目的：

1. 掌握 move\_base 算法导航的基本使用方法
2. 了解 DWA 和 TEB 局部运动规划的优缺点

3、熟悉导航中的各种参数及调参方法

#### 二、实验内容：

1. 切换为 Teb 局部运动规划算法

2、多点导航

#### 三、实验器材：

ROS 车、电脑

#### 四、实验原理：

move\_base 算法是 ROS（Robot Operating System）中常用的路径规划和导航算法之一。

它基于全局路径规划和局部避障两个核心模块。

①全局路径规划：move\_base 使用代价地图（costmap）来表示环境中的障碍物信息，通过算法（如 Dijkstra、A\*等）在这个代价地图上计算出全局路径， 以使机器人从起点移动到目标点。

②局部避障：move\_base 还包含一个局部避障模块，该模块使用代价地图中的障碍物信息，结合机器人当前状态（位置、姿态等）和传感器数据，计算出机器人在局部区域内的最优运动指令，以避开障碍物。

move\_base 算法的优点是可以处理复杂的环境和全局路径规划，同时提供局部避障能力。它适用于需要长时间导航和全局路径规划的场景，如移动机器人在大型室内环境中的导航任务。

**DWA (Dynamic Window Approach)**:动态窗口法是一种在机器人路径规划中

常用的方法。它基于机器人在速度和转向速度空间中的"动态窗口"，通过考虑机器人的运动动力学和环境的障碍物来选择最佳的速度和转向速度组合。DWA 旨在快速生成机器人的安全、平滑的运动轨迹。

优势：在速度和转向速度空间中进行搜索，适用于具有较低运动约束的情况。实时性较好，适用于快速移动的场景。

**TEB (Timed Elastic Band):**时间弹性带法是另一种用于路径规划的方法。TEB 方法将路径规划问题建模为一个在时间上具有弹性的带状结构。它考虑了机器人的动力学约束、避障和全局路径规划，以生成时间上连续和光滑的运动轨迹。TEB 的目标是兼顾路径的质量和执行的时间。

优势：能够生成时间上连续和平滑的运动轨迹。考虑了更复杂的动力学约束， 适用于具有较高运动约束和需要更精细控制的场景。

TEB 在运动过程中会调整自己的位姿朝向，当到达目标点时，通常机器人的朝向也是目标朝向而不需要旋转。DWA 则是先到达目标坐标点，然后原地旋转到目标朝向。在我们这个赛项上，车的底盘结构为阿克曼底盘，很显然使用 TEB 更合适。

**导航流程：**

①自动发布导航点：在先前建好地图的基础上，提前设置导航点，并在导航时通过一个节点进行发布。

②全局路径规划：基于地图和机器人当前位置，使用路径规划算法（如 A\* 算法）计算出一条从起点到目标点的全局路径。

③局部路径规划：基于机器人当前位置和局部感知信息，使用 DWA 算法、TEB 算法或其他规划算法，生成机器人的局部路径，并考虑避障。

④运动控制：根据当前位置和局部路径，通过简单的 PID 控制，向下位机发送指令，控制机器人的执行器（电机和舵机），实现机器人的自主导航。

#### 五、实验步骤：

**<1>** 安装 navigation 功能包

连接互联网，在终端输入以下命令：

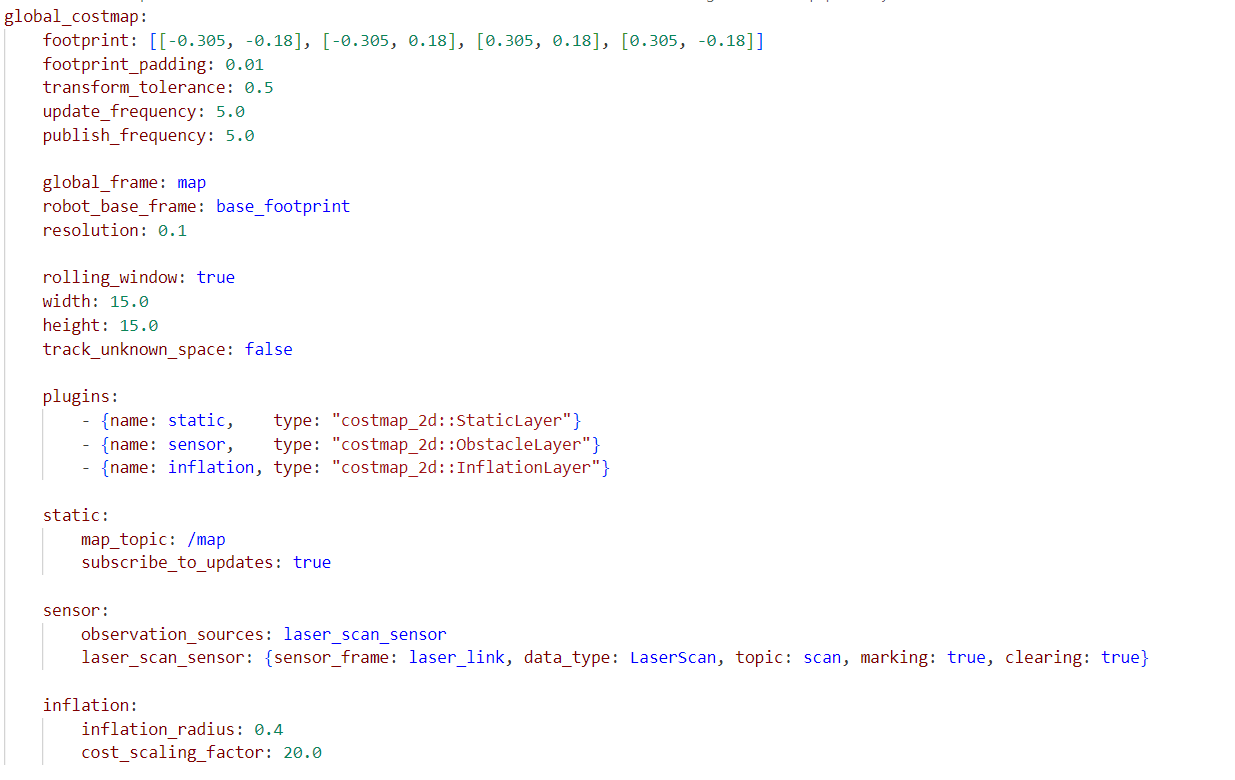
sudo apt install ros-noetic-navigation

输入密码，即可安装 navigation 功能包。

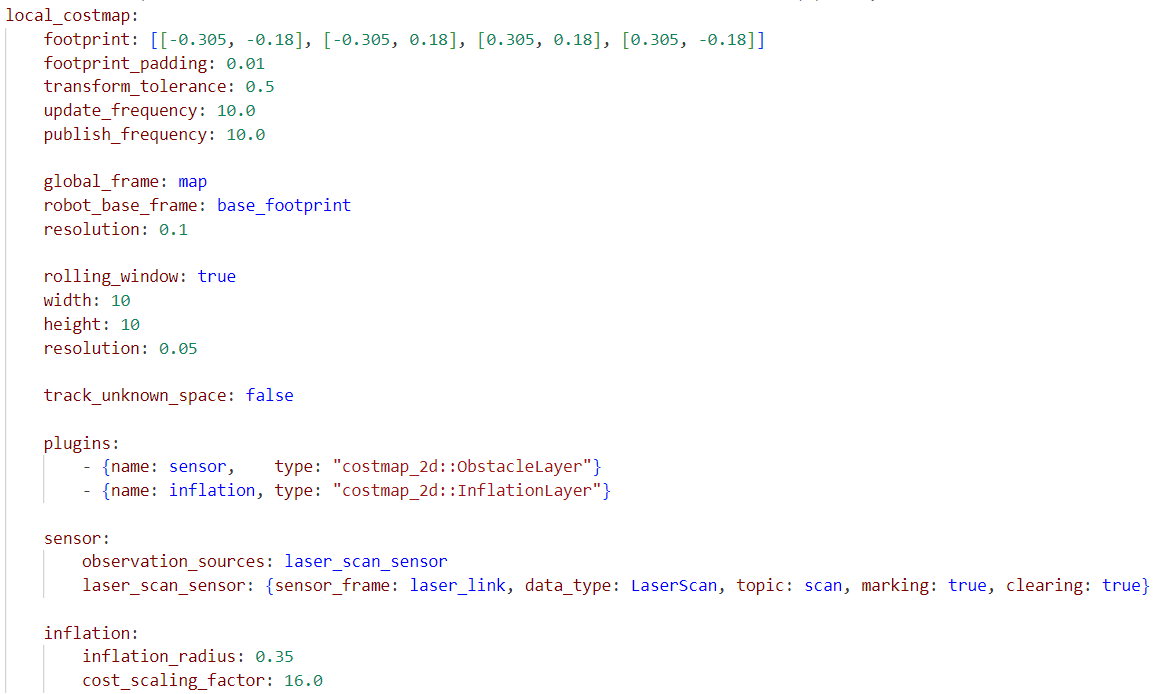
**<2>** 配置 move\_base 参数

此处我们使用 NavfnROS 作为全局路径规划算法，使用 DWAPlannerROS 作为局部路径规划算法，参数配置需要编写五个文件，其内容如下：

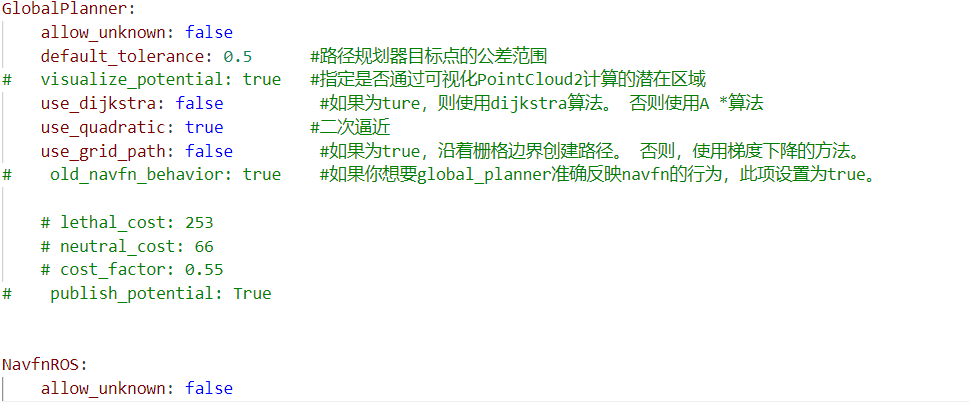
·global\_costmap\_params.yaml:



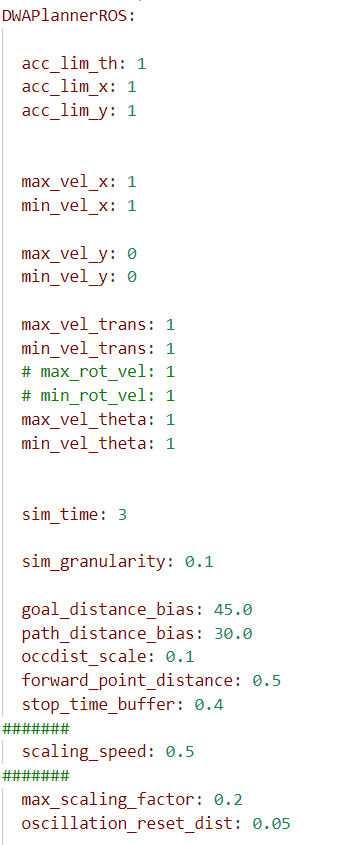
·local\_costmap\_params.yaml:

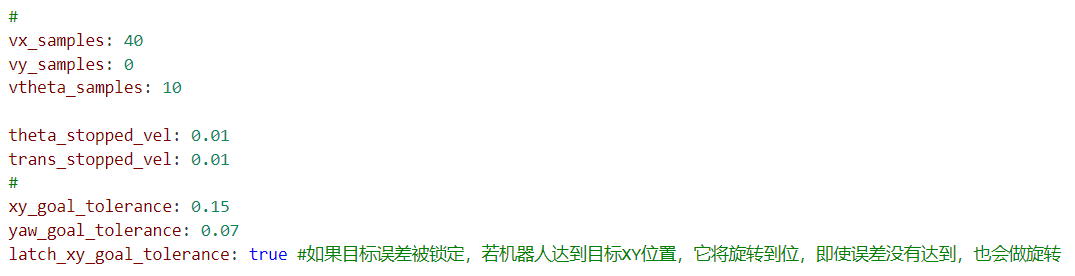


·base\_global\_planner\_params.yaml:

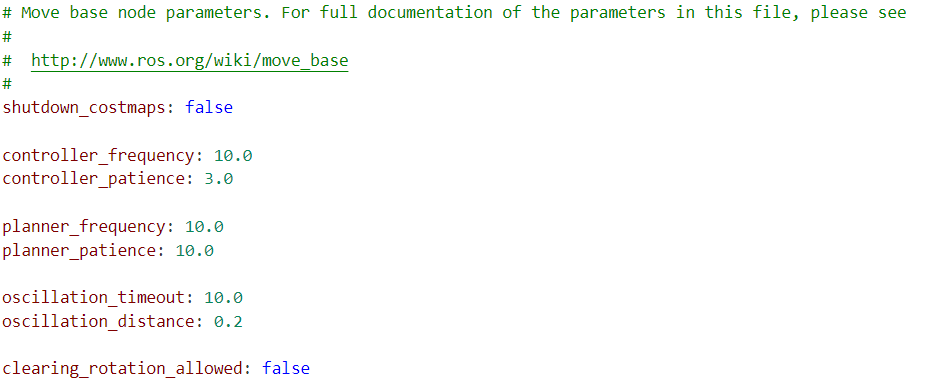


·dwa\_local\_planner\_params.yaml:

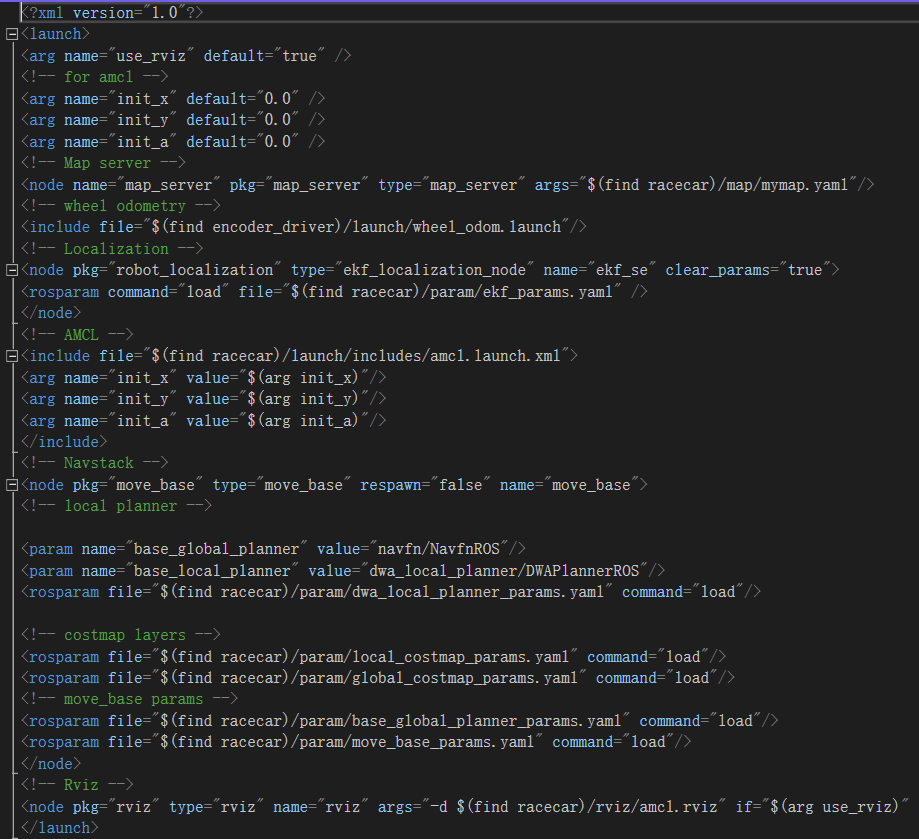




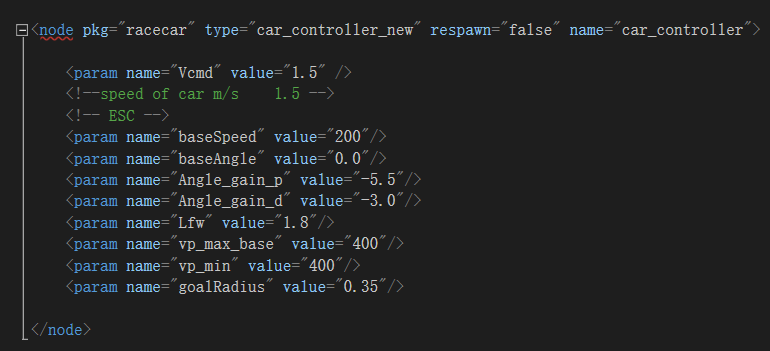
·move\_base\_params.yaml:



**<3>** 编写 launch 文件开启导航算法编写 launch 文件如下：



其中，需注意各文件的路径问题。运行该文件，在 rviz 界面修正小车的起始位置后发布导航点，即开始进行路径规划。我们可以选择使用 rviz 中的 2D goal point（紫色箭头）发布导航点，也可以选择使用程序进行自动发送。 在导航时，提前开启控制器，可以实现按照规划路径的导航。我们可以在编写的 launch 文件中加入以下代码：



随后，我们便可以实现自主导航。 值得注意的是，有许多参数会影响导航效果，需要读者自行调整。

**<4>** 局部路径规划更换为 TEB

上面所使用的 DWA 算法并不适用于我们所使用的阿克曼底盘 ROS 智能车， 故需要将其更换为可以适用于阿克曼底盘的 TEB 算法。 由于 ROS 的 navigation 不自带 TEB 算法，需要自己安装相关的功能包。连接互联网，在终端中输入下面这条命令：

sudo apt install ros-noetic-teb-local-planner

安装完后重新 source 一遍工作空间，通过下面命令检查是否安装成功：

rospack plugins --attrib=plugin nav\_core

要使用 TEB 算法，还需要配置参数、更改 launch 文件，流程如下：

·编写 TEB 相关参数文件内容

·更改 文件中 \_ 部分，需要注意文件路径

·运行 launch 文件，此时便可使用 TEB 算法进行局部路径规划。

### **实验七 停车点识别**

#### 一、实验目的：

1、理解图像处理和计算机视觉的基本概念。

2、熟悉 OpenCV 库，了解如何使用 OpenCV 进行图像处理、对象检测和特征提取。

3、熟悉图像处理技术，如直方图均衡化、滤波、形态学操作等，以提高图像质量和检测效果。

#### 二、实验内容：

1、读取智能车摄像头

2、识别停车点标志

三、实验器材：

ROS 智能车、电脑

#### 四、实验原理：

读取ROS智能车摄像头并转化为 HSV 色彩空间，进行二值化、腐蚀、膨胀等操作，根据颜色和停车点标志大小及长宽比筛选，进行停车点标志的识别。

#### 五、实验步骤：

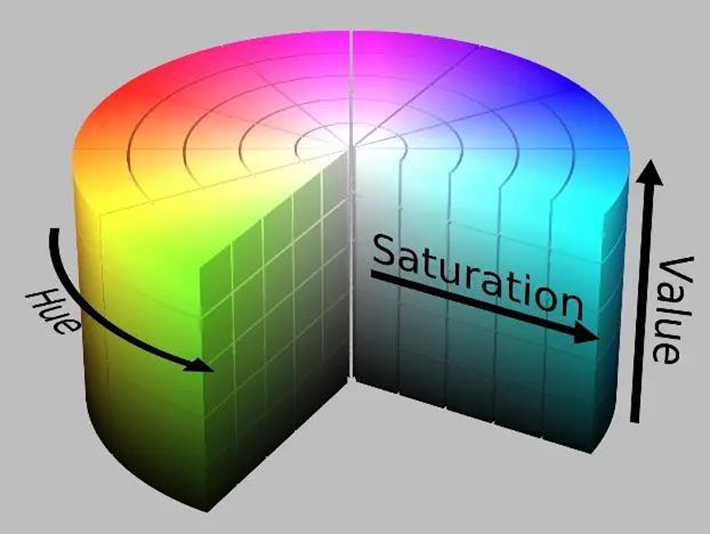
**<1>** 转化 HSV 颜色空间

摄像头所获取的图像通常为 RGB 图像。根据图 7-1 所示，停车标志最显著的特征是其大片的蓝色区域，因此其蓝色分量明显大于绿色和红色分量。如果我们提取蓝色通道并与其他颜色通道进行比较，若蓝色通道的值明显大于其他通道，则可以推测该物体在图像中主要呈现蓝色。另一方面，白色物体的三个颜色通道分量均较为均衡，因而其各通道值相减后的结果接近零。



（图 7-1）

然而，RGB 色彩空间的分离效果并不理想，且容易受到光照条件的影响。因此，为了获得更好的分离效果，通常会将图像转换到 HSV 颜色空间。图 7-2 中展示了 HSV 颜色模型，其中 H 表示色调或色相，其取值范围为 0 至 360°，H = 240 代表红色，H = 120 代表绿色，H = 240 代表蓝色，且色相变化连续平滑。S 表示饱和度，反映颜色的纯度，饱和度越高，颜色越深，值为 0 时表示纯白色，值越大则颜色越饱和。V 代表明度，取值范围为 0 至 100%，明度越高，颜色越亮，V = 0 时表示纯黑色。通过转换到 HSV 空间，可以更有效地进行颜色分离与处理，尤其是在光照变化较大的环境中。



（图7-2）

其代码如下:



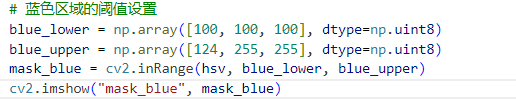
其识别结果如下:



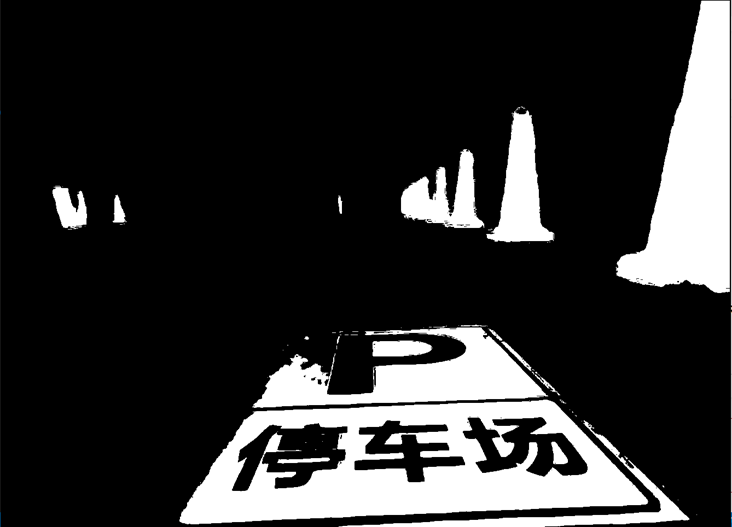
(图 7-3)

**<2>** 提取蓝色区域

提取蓝色区域需要使用 inRange() 函数，以此根据设定的阈值去除阈值之外的背景部分具体参数。



hsv 指的是 hsv 色彩空间的原图。blue\_lower 指的是图像中低于这个 blue\_lower 的值，图像值变为 0，即变为黑色。blue\_upper 指的是图像中高于这个 blue\_upper 的值，图像值变为 0。生成结果见图8-4。

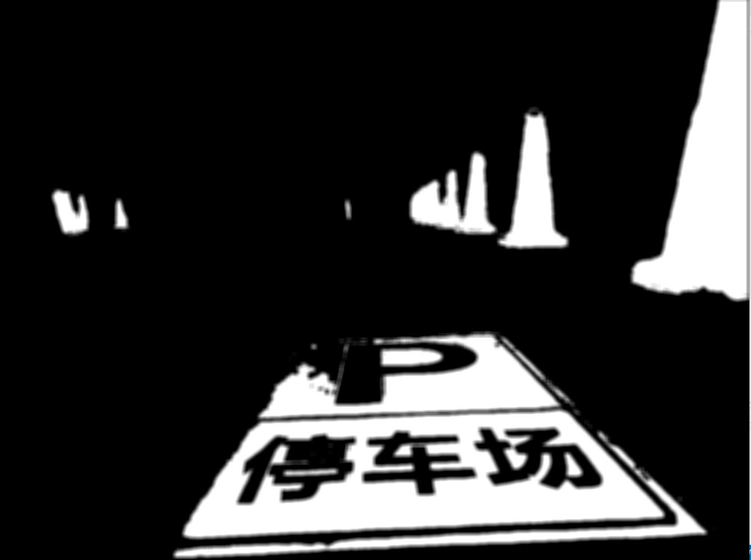


(图 7-4)

**<3>** 图像处理

cv2.blur(src, ksize): 采用均值滤波，它只取内核区域下所有像素的平均值并替换中心元素。这里使用 9x9 的盒式过滤器，详见图8-5。

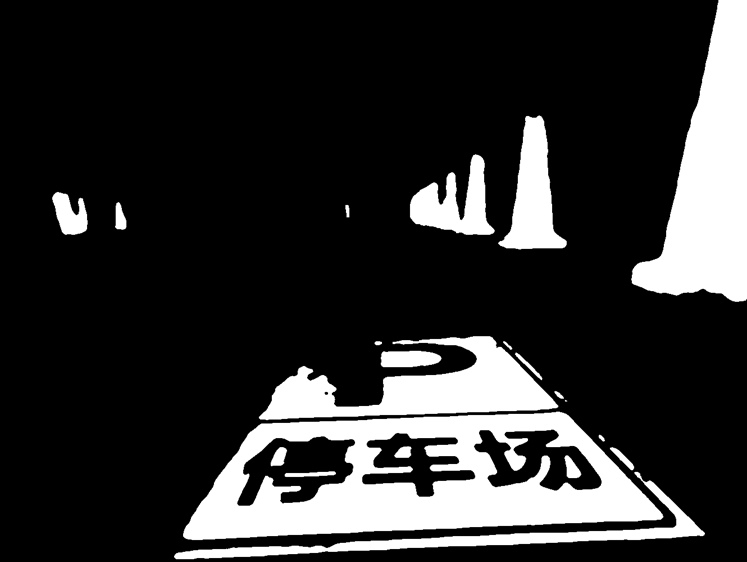




(图 7-5)

cv2.threshold(src, thresh, maxval, type): 对图像进行阈值化处理。 src 为图像，type为进制数。在这里用于将模糊后的图像转换为二值图像，详见图6。

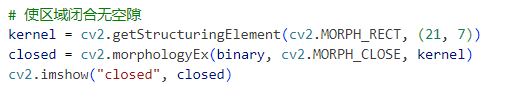


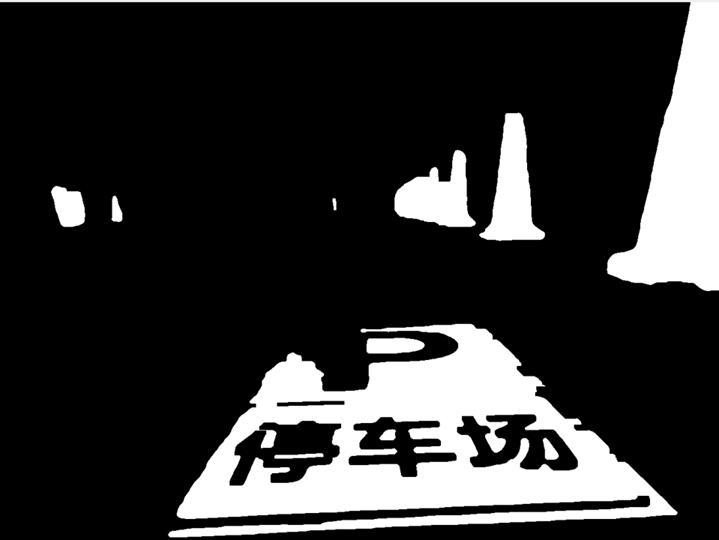


(图 7-6)

cv2.getStructuringElement(shape, ksize): 创建一个形状为 shape，大小为 ksize 的结构元素。

cv2.morphologyEx(src, op, kernel): 对图像进行形态学操作。这里是闭合操作。处理后的图像见图7。

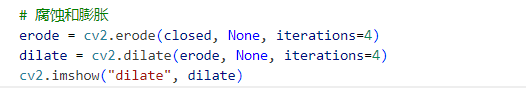


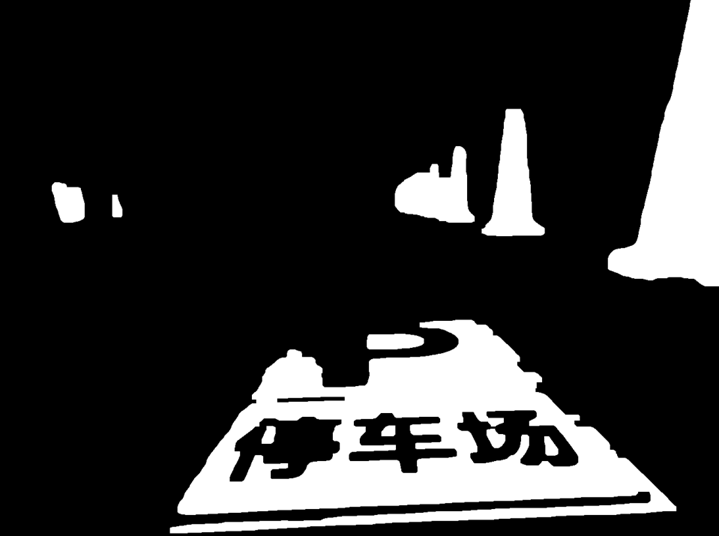


(图 7-7)

cv2.erode(src, kernel, iterations): 对图像进行腐蚀操作。这里用于对闭合后的图像进行腐蚀。

cv2.dilate(src, kernel, iterations): 对图像进行膨胀操作。这里用于对腐蚀后的图像进行膨胀。处理后的图像见8-8。





(图 7-8)

<4>查找轮廓

cv2.findContours(image, mode, method): 查找图像中的轮廓。这里用于找到二值图像中的外部轮廓。



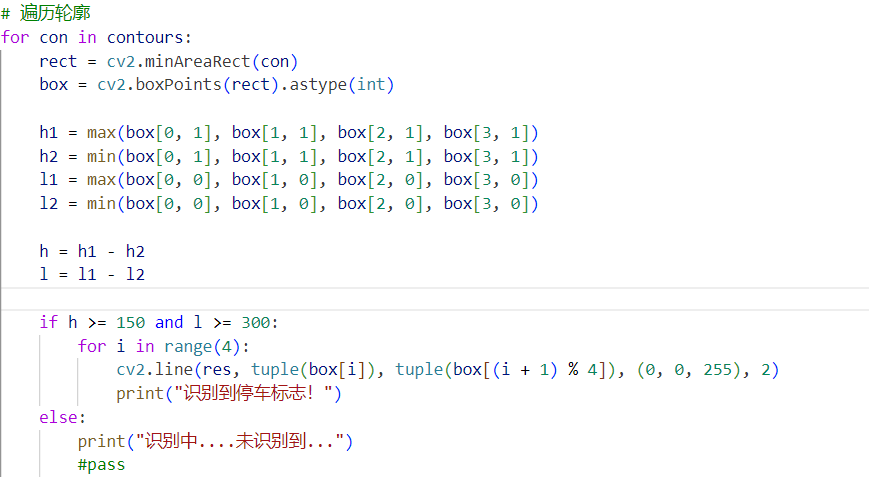
<5>遍历并筛选轮廓

由于右边的锥桶与停车场标识同色，所以我们需要根据停车点标志的尺寸，筛选出蓝色部分中真正的标志,并在原图中绘制出来。

cv2.minAreaRect(points): 根据一组点拟合最小外接矩形。这里用于找到外接矩形。

cv2.boxPoints(rect): 获取最小外接矩形的四个顶点坐标。

cv2.drawContours(image, contours, contourIdx, color, thickness): 在图像上绘制轮廓。这里用于在原图像上绘制找到的外接矩形。最后成功识别，见图8-9。





(图 7-9)

### 实验八：地图滤波与锥桶连线

#### 一、实验目的：

1. 掌握基础的图像处理方法，熟悉ROS智能小车的地图数据处理。  
2. 通过图像处理技术提高地图的可读性与清晰度。  
3. 通过对地图中锥桶的连线，改善导航效果，提升智能小车的路径规划准确性。

#### 二、实验内容：

1. 地图滤波实验：对地图图像进行噪点去除，提高图像清晰度。  
2. 锥桶连线实验：识别地图上的锥桶区域，通过连线算法将锥桶连接，形成连续的路径参考，便于导航系统识别和应用。

#### 三、实验器材：

ROS智能小车、电脑

#### 四、实验原理：

地图生成过程中，由于环境复杂等原因，地图数据可能会产生以下问题：  
  
- 噪点干扰：地图图像中可能包含噪声点，影响图像的识别和导航的准确性。  
- 锥桶信息模糊：作为导航标识的锥桶信息在地图上不够明显，影响路径识别。  
  
为了解决这些问题，可以采用图像滤波的方法对地图图像进行处理。首先保存gmapping生成的地图数据，格式为.pgm，然后对图像数据进行滤波操作，去除噪点；接着，利用凸包等图像处理算法，对锥桶进行连线，以增强锥桶的可识别性和连贯性。

#### 五、实验步骤：

### 1. 选择滤波方法：

选择合适的滤波方法是关键步骤。通过查阅相关资料和技术支持，学习了中值滤波、均值滤波、高斯滤波、双边滤波等用于去除图像噪声和增强图像质量的方法。最终我们选择了面积滤波法，主要因为它能显著增强锥桶轮廓的清晰度，并有效去除噪点。

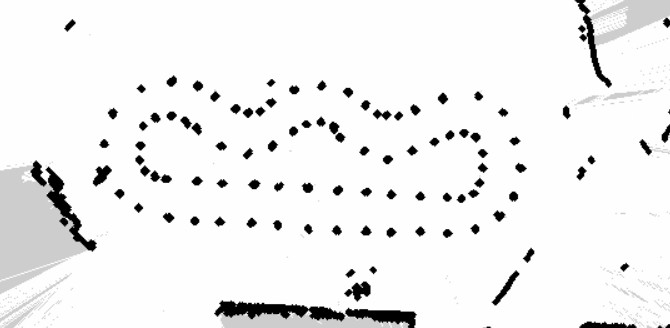
### 2. 滤波的实现：

安装图像处理库OpenCV和NumPy后，通过以下Python代码实现滤波步骤：

import cv2  
import numpy as np  
  
class MapBlur:  
 def mapBlur(self):  
 # 读取图像  
 img = cv2.imread('originMap.pgm')  
 # 转换为灰度图像  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 # 二值化处理  
 \_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY\_INV)  
 # 提取轮廓  
 contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 # 定义存储保留轮廓和过滤轮廓的列表  
 blk\_contour, filtered = [], []  
 # 根据面积筛选轮廓  
 for contour in contours:  
 area = cv2.contourArea(contour)  
 if area >= 0.3: # 设定面积阈值  
 blk\_contour.append(contour)  
 else:  
 filtered.append(contour)  
 # 复制图像并绘制轮廓  
 blured\_img = np.copy(img)  
 # 用白色填充过滤的区域  
 cv2.drawContours(blured\_img, filtered, -1, (255, 255, 255), cv2.FILLED)  
 # 用黑色标记保留的轮廓  
 cv2.drawContours(blured\_img, blk\_contour, -1, (0, 0, 0), 3)  
 # 显示对比效果  
 result = cv2.hconcat([img, blured\_img])  
 cv2.imshow('result', result)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

滤波效果：

对比原图可以看到，图中的噪点基本去除了，并且锥桶经过增强后清晰可见。

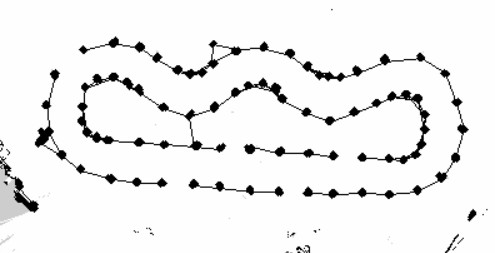


### 3. 锥桶连线的思路和实现：

观察地图图像，可以发现锥桶在多个同心圆上均匀分布，相邻锥桶间距离相似。基于这一特点，我们可以设定不同的圆心和半径，将符合条件的锥桶相连。  
  
实现步骤如下：  
- 读取地图图像，将其裁剪为包含锥桶的区域。  
- 将裁剪后的图像转为灰度图，再进行二值化处理。  
- 提取轮廓，计算每个轮廓的中心点，并将符合位置条件的点添加到不同的同心圆上。  
- 通过计算两点间的距离，实现锥桶的连线，以生成连贯的路径。

import cv2  
import numpy as np  
  
def distance\_2(p1, p2):  
 # 计算两点之间距离的平方  
 return (p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2  
  
class MapBlur:  
 def mapBlur(self):  
 img = cv2.imread('originMap.pgm')  
 # 灰度处理  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 \_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY\_INV)  
 contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 points = [contour[int(contour.shape[0]/2)][0] for contour in contours]  
 line\_points = []  
 # 定义同心圆的圆心和半径  
 cir\_centers = [(467, 357), (582, 357), (692, 357)]  
 for center in cir\_centers:  
 for p1 in points:  
 for p2 in points:  
 if p1 != p2 and distance\_2(p1, p2) < 34\*\*2:  
 line\_points.append([p1, p2])  
 # 连线并显示  
 for p1, p2 in line\_points:  
 cv2.line(img, tuple(p1), tuple(p2), (0, 0, 255), 2)  
 cv2.imshow('Connected Map', img)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()

滤波及连线效果：

可以看到，锥桶基本被连接起来，足够用于导航。

### 4. 实验复现步骤：

为确保实验的可复现性，可以在终端运行以下命令：

```bash  
python3 mapBlur.py # 运行滤波处理  
python3 bucketConnection.py # 运行锥桶连线  
```

## 六、实验结果：

经过滤波处理，地图图像中的噪点已被有效去除，锥桶区域也被增强。锥桶连线实验中，符合条件的锥桶被连接成线，形成连续的路径。这种处理效果可以提升智能小车的导航识别能力，为导航路径规划提供了更可靠的数据。