第十七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：同济大学

队伍名称：小车快快跑队

参赛队员：王子安

姜垒

黄锴康

朱捷

带队教师：张志明

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：2022. **. **

（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　摄像头激光雷达数据融合

实验目的：

1、了解激光雷达及其所得数据的处理方法；
2、掌握数据的坐标转化运算方法；
3、熟练掌握 rviz 功能包的使用

实验内容：

1、利用激光雷达获取目标点坐标数据；
2、利用摄像头获取目标点坐标数据；
3、结合激光雷达和目标点坐标数据找到对应的标定参数，即联合标定；

实验仪器：

ROS 智能车、单目摄像头、激光雷达等

实验原理：

激光雷达是一种遥感技术，它使用光并以脉冲形式发射光。相对于激光辐照测量散射光，并且分析到物体的长距离和物体的特性。这种技术雷达是通过替换无线电波雷达到光而得到相似的。到物体的距离由发光后直到接收到反射光的时间差决定。

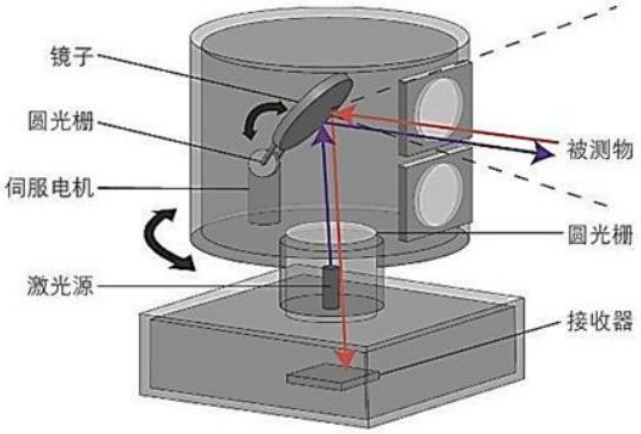

图 1.1 激光雷达内部构造示意图

理论上激光雷达点要先从世界坐标系经过旋转、平移、缩放等变换转换到相机坐标，再乘以相机内参矩阵转换到图像坐标，但这种方法显然处理数据时显得过于繁琐。

事实上，更简洁地，二维激光雷达坐标到图像坐标的变换可以看成是一个射影变换，二维激光的激光点就相当于俯视图，摄像头拍到的图像可以想象为主视图，而俯视图乘以一个单应矩阵变换到主视图的过程就是射影变换。

转换的一般形式如下：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (1.1)$$

其中，左式是[x y 1]是激光雷达的齐次坐标，右式[u v 1]则是激光转换到图像坐标系后的图像齐次坐标。将 h 矩阵中的 h_9提出并放到等式左边，作为缩放因子 λ 后，可将公式写成如下形式：

$$\lambda * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} n_1 & n_2 & n_3 \\ n_4 & n_5 & n_6 \\ n_7 & n_8 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (1.2)$$

上式得到的矩阵就是我们需要求出来的变换矩阵参数，为求该矩阵，需要消去缩放因子 $\lambda = n_7 * x + n_8 * y + 1$. 如下式所示：

$$\begin{cases} (n_7x + n_8y + 1)u = n_1x + n_2y + n_3 \\ (n_7x + n_8y + 1)v = n_4x + n_5y + n_6 \end{cases} => \begin{cases} u = n_1x + n_2y + n_3 - n_7ux - n_8uy \\ v = n_4x + n_5y + n_6 - n_7vx - n_8vy \end{cases} \quad (1.3)$$

相似地，假设收集到了 k 组对应点，可将上述方程均写成矩阵形式：

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -u_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -v_1x_1 & -v_1y_1 \\ & & & & & \cdots & & \\ & & & & & \cdots & & \\ x_k & y_k & 1 & 0 & 0 & 0 & -u_kx_k & -u_ky_k \\ 0 & 0 & 0 & x_k & y_k & 1 & -v_kx_k & -v_ky_k \end{bmatrix} * \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \\ n_7 \\ n_8 \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ \cdots \\ \cdots \\ u_k \\ v_k \end{bmatrix} \quad (1.4)$$

从式中可以看出，一组数据对应两个 n 参数，那么要求出 n 向量至少需要 4 组数据，组成超定方程求最优解至少需要 5 组数据，超定方程的求解则使用最小二乘法。实际应用中，可以适当地多收集几组数据。

实验步骤：

<1> 将锥桶的棱作为目标参照物，在锥棱上贴上白纸，并在和激光雷达等高的地方做上黑色的标记，减少其余无关因素干扰，调用 calibration 中的 opencv 程序可以很快得到黑点的坐标。

值得注意的是，为了更清晰地定位黑线标记，这里有黑色标记的白纸不贴在面上，而是选择贴在有一定角度的棱上，把白纸对折也可以达到预期的效果。这样就可以收集实验原理中提到的求解所需的几组数据。收集时尽量让点的位置交错开，不要三点共线，以免影响 n 的求解。

图 1.2 标定参照物示意图

<2> 在 rviz 中寻找该锥桶的棱所在的坐标，即得到标定物在 laser 坐标系下的 2d 坐标，使用 rplidar 中的 frame_grabber 进行数据采集。

所得的坐标数据中，横坐标为 Y（左右相对偏移量），纵坐标为 X（远近相对距离）。在得到的图中，可以清晰地看到下半部分的三个折角，这是上一步中白纸贴在锥棱上的数据采集效果，如果贴在面上则效果不佳，需要有一定角度。黑色标记的位置就是激光雷达点的坐标（即 X,Y 的值）。

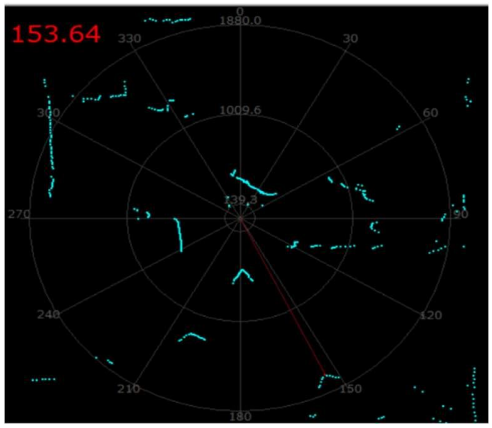

图 1.3 标定参照物激光雷达图像

<3>利用摄像头图像读取数据（take_photo），筛选所得数据以实现联合标定

在 ubuntu 系统下 vs 中运行 take_photo 程序，即可以实现按 q 截取摄像头画面这一功能。在截取完画面后，运行 mouse 程序读取之前截取的画面。

如下图所示，得到标定参照物摄像头坐标的图像。对于锥桶侧棱上与激光雷达等高的标记，图中将显示对应像素点所在的坐标（U,V），其中 U、V 分别为横纵坐标。

图 1.4 标定参照物摄像头坐标图像

在实际实验中，记录数据的组数应尽可能多以确定标定参数。我们从赛题发布后实际通过实验测量了大约上百组数据以保证标定参数的可靠性。最终，从中筛选出表现较好、较为可靠的二十多组数据，列表如下。其中，X，Y 单位为毫米，表示锥桶在 laser 坐标系下的对应二维坐标。而 U，V 分别表示在摄像头图像（640×480）中像素点的横坐标与纵坐标，即联合标定步骤。

X/mm	Y/mm	U	V
347	162	32	129
380	-27	348	158
401	-217	594	187
1227	586	100	222
1314	-38	324	235
1528	-666	497	250
1666	1104	21	225
1771	361	226	236
1902	-1390	616	261
1098	615	57	216
1895	-417	405	247
1442	-990	601	256
982	427	112	216
1517	67	292	237
1120	-822	627	251
920	573	23	209
1088	-223	402	235
919	-609	608	244
682	388	33	194
682	65	256	207
788	-544	625	239
492	245	49	172
598	-75	372	204
508	-303	605	212

<4> 利用 matlab，数据以 6 组为一个周期求解向量 n，得出标定参数

相关代码如下：

```
x=[x_1 x_2 … x_6]; % 雷达坐标 x 轴
y=[y_1 y_2 … y_6]; % 雷达坐标 y 轴
u=[u_1 u_2 … u_6]; % 图像坐标 u 轴
v=[v_1 v_2 … v_6]; % 图像坐标 v 轴
z=[u_1 v_1 u_2 v_2 … u_6 v_6]'; % 所有 u v 依次排下去，即（1.4）式等号右边的向量
% X 为公式（1.4）中的左边第一个矩阵
X=[ x(1) y(1) 1 0 0 0 -u(1).*x(1) -u(1).*y(1) ;
    0 0 0 x(1) y(1) 1 -v(1).*x(1) -v(1).*y(1) ;
    x(2) y(2) 1 0 0 0 -u(2).*x(2) -u(2).*y(2) ;
    0 0 0 x(2) y(2) 1 -v(2).*x(2) -v(2).*y(2) ;
    x(3) y(3) 1 0 0 0 -u(3).*x(3) -u(3).*y(3) ;
    0 0 0 x(3) y(3) 1 -v(3).*x(3) -v(3).*y(3) ;
    x(4) y(4) 1 0 0 0 -u(4).*x(4) -u(4).*y(4) ;
    0 0 0 x(4) y(4) 1 -v(4).*x(4) -v(4).*y(4) ;
    x(5) y(5) 1 0 0 0 -u(5).*x(5) -u(5).*y(5) ;
    0 0 0 x(5) y(5) 1 -v(5).*x(5) -v(5).*y(5) ;
    x(6) y(6) 1 0 0 0 -u(6).*x(6) -u(6).*y(6) ;
    0 0 0 x(6) y(6) 1 -v(6).*x(6) -v(6).*y(6) ];
    n=X\z; % 向量 n = X^(-1) * z 即 X 矩阵的逆乘以 z 向量
```

最终采取的标定参数为：

float a[9] = {−2.0189 , 2.6404 , 292.7613, − 1.6558 , 0.2168,380.2369, − 0.0065,0.0003 , 1}，这是一个 3*3 的矩阵。

<5> 对坐标进行射影变换，转换坐标系

求得向量 n 后需要实现对所有点进行坐标系转换。

具体操作步骤为，启动程序中激光雷达 ros 节点，发布/scan 话题，再运行 rosrun lidar_camera_fusion fusion（opencv_lidar）。

经过上述操作，能够求出向量 n，也是实现本实验联合标定的至关重要的数据。将向

量 n 放入数组 a[]中，显示在图上。经过坐标转换后所得的锥桶标定效果图如下所示。至于锥桶的红色和绿色边框，则需要转到锥桶识别和图像处理部分的实验。

图 1.5 转换点位后效果图

实验二　基于 gmapping 算法的虚拟墙应用实验

实验目的：

1、熟悉 gmapping 地图构建算法；
2、在 1 的基础上熟练掌握 gmapping 功能包的使用；
3、熟悉 rviz 功能包的使用以拓展虚拟墙的应用；

实验内容：

1、调用 gmapping 功能包并建图；
2、通过插值补出虚拟墙。

实验仪器：

ROS 智能车、激光雷达、编码器、惯导传感器等

实验原理：

Gmapping 是一个基于 2D 激光雷达使用 RBPF 算法完成二维栅格地图构建的 SLAM 算法，可以实时构建室内地图，构建小场景地图所需的计算量较小且精度较高。且地图精度较高，对激光雷达扫描频率要求较低。

Gmapping 以 Fast-SLAM 方案为基本原理。FastSLAM 算法中的机器人轨迹估计问题使

用的是粒子滤波方法（先建图，再定位）。由于使用的是粒子滤波，每一个粒子都包含了机器人的轨迹和对应的环境地图。将不可避免的带来两个问题。

第一个问题，当环境大或者机器人里程计误差大时，需要更多的粒子才能得到较好的估计，这是将造成内存爆炸；第二个问题，粒子滤波避免不了使用重采样，以确保当前粒子的有效性，然而重采样带来的问题就是粒子耗散和粒子多样性的丢失。

由于这两个问题出现，导致 FastSLAM 算法理论上可行，实际上却不能实现。针对以上问题 Gmapping 提出了两种针对性的解决方法：降低粒子数量和缓解粒子消散。

选择 Gmapping 是基于如下考虑：由于本次比赛要求的赛道元素少且锥桶之间距离较大（锥桶和锥桶间距离＞0.5m），整个赛道的精度要求不高，可将其缩为小地图，所以 gmmapping 在该场景下可以称得上是高效的算法。

相比 HectorSLAM，gmmapping 对激光雷达频率要求低、地图精度较高。此外，相比 Cartographer（把地图分成局部子图，最后拼起来成为完整的地图），在构建小场景地图时，gmapping 不需要太多的粒子，且没有回环检测因此计算量小于 Cartographer，精度相差不大。而且 gmapping 有效利用了里程计信息，通过里程计信息先定位后建图，这也是 gmapping 对激光雷达频率要求低的原因。本次智能车上的激光雷达频率最高为 10Hz，但建图效果非常好。

使用 gmmapping 算法，根据 costmap2d 中的三个插件，使用 setcost 方法可以编写一个虚拟墙插件，大大提高小车前进的效率。在调试中，我们使用 rviz 工具包中的"Publish Point" 发布的"/clicked _point”话题，在地图中绘制虚拟墙，仿真实验中，该方法得到一定的避障效果，于是在比赛过程中加以应用，效果较好。

为了使用 GMapping 在实验中尽可能提升建图的精度，有以下注意事项：GMapping 虽然没有优化，但依靠粒子的多样性，在回环时仍能消除累计误差。但需要注意的是，要尽量走小回环，回环越大，粒子耗尽的可能性就越高，越难在回环时修正回来。所以规划建图路径时，应先走一个小回环，当回环成功后，可以再多走几圈，消除粒子在这个回环的多样性。接下来走下一个回环，直到把整个地图连通成一个大的回环。机器人起始位置也很重要，应选在特征丰富的地方，这样在回环发生时更容易提高正确粒子的权重。场景越大，需要的粒子数越多。如果建图失败，可以适当提高粒子数量。

在测试中，具体在实际赛道上，我们获取锥桶点的方法为：将融合惯导的视觉里程计和激光雷达进行松耦合检测到的锥桶坐标信息，将该信息作为 topic 发布，使得绘制虚拟墙的节点订阅该消息，通过插值，在两个同侧相邻的锥桶间，补出连续虚拟墙，最终在 costmap 上用虚拟墙将同侧的离散锥桶连接成连续赛道边界，来避免第二圈 ros 导航过程中小车导出赛道。

实验步骤：

<1> 得到小车跑第一圈的数据（middle lane 规划为主）

小车根据锥桶图像处理识别后采用 middle lane 规划路径，按逆时针方向跑一圈赛道，其中赛道左侧为红色锥桶，右侧为蓝色锥桶。

<2> 打开 rviz 功能包，订阅 map 话题

小车在设置好的场地跑第一圈时，打开 rviz 功能包，订阅 map 话题以实时观测到 gmapping 建好的地图，跑完一圈后迅速建图以便第二圈的导航。所建好的图像（俯视图）如下所示，其中黑色的点状轨迹为红蓝锥桶围成的赛道。

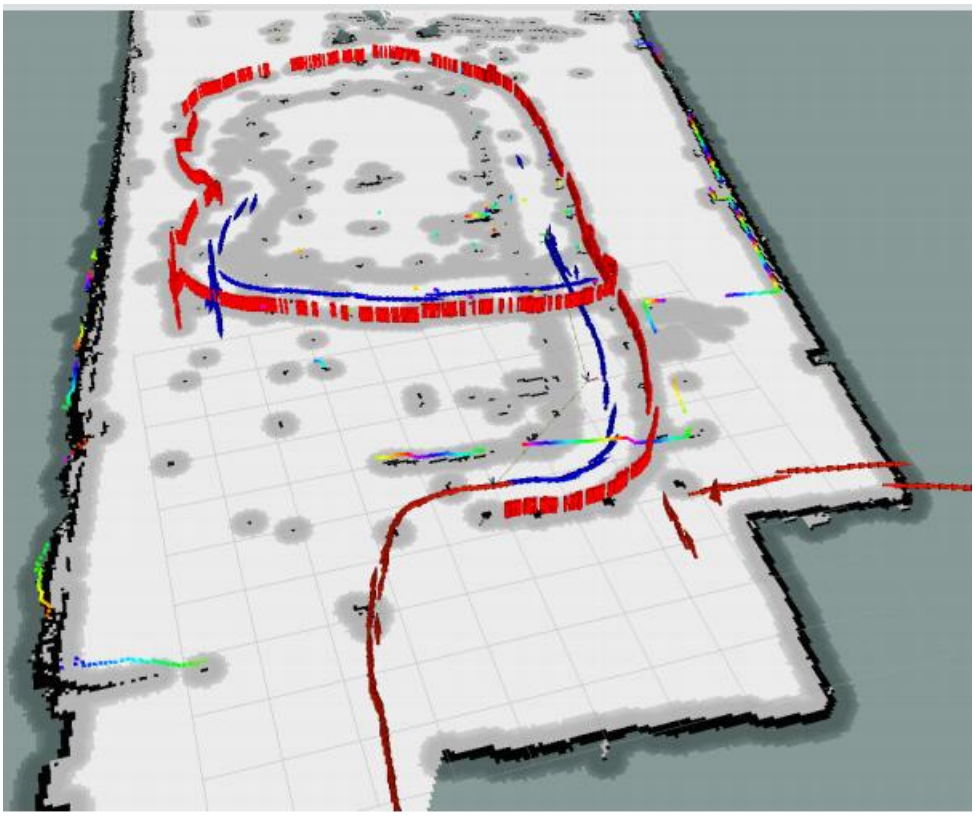

图 2.1 建图测试结果

<3> 订阅 virtual_wall 话题

小车跑第一圈时订阅 virtual_wall 话题，便可以看到虚拟墙的构建，如下图所示，可以看到点状轨迹间有直线连接，这样更便于小车高效完成第二圈。

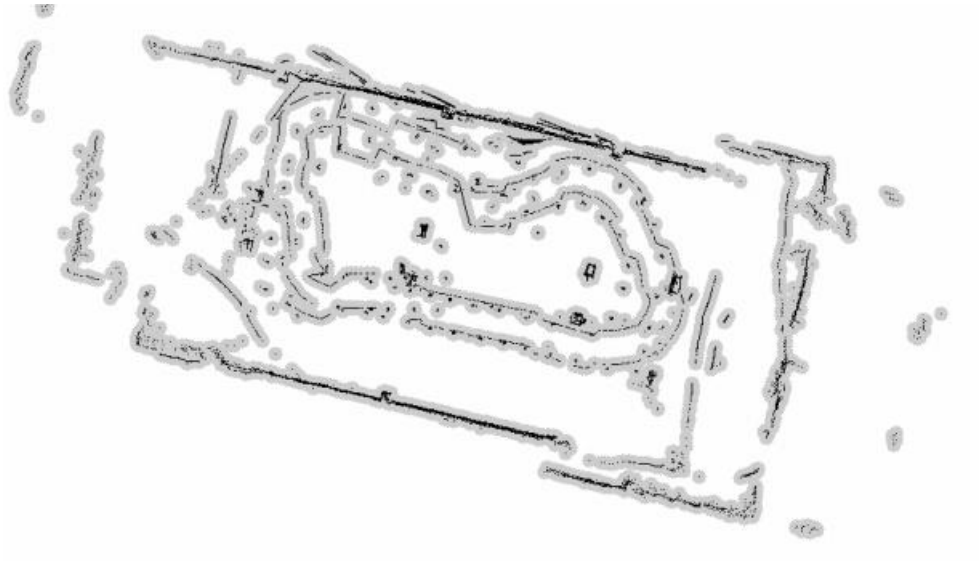

图 2.2 含虚拟墙的建图测试结果

<4> 查看 gmapping 所订阅和发布的话题

图中可以看到节点/slam _gmapping 订阅了/tf,/scan 两个话题,同时发布了/tf, /map 两个话题。从 tf 树中看到，map 跟 odom 的关系是由节点/slam _gmapping 发布的，即节点/slam_gmapping 给出了智能车里程计跟地图之间的关系，而这个关系对于建出正确的地图是不可缺失的。

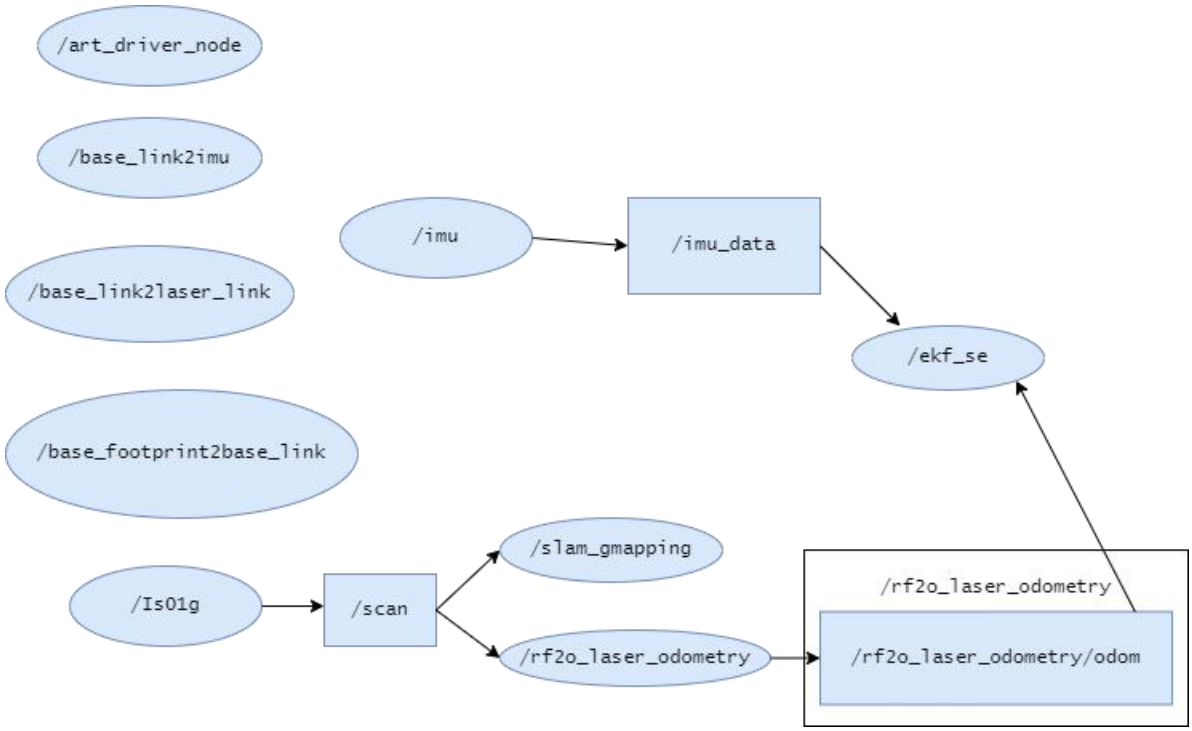

图 2.3 rqt_gragh-/slam_gmapping

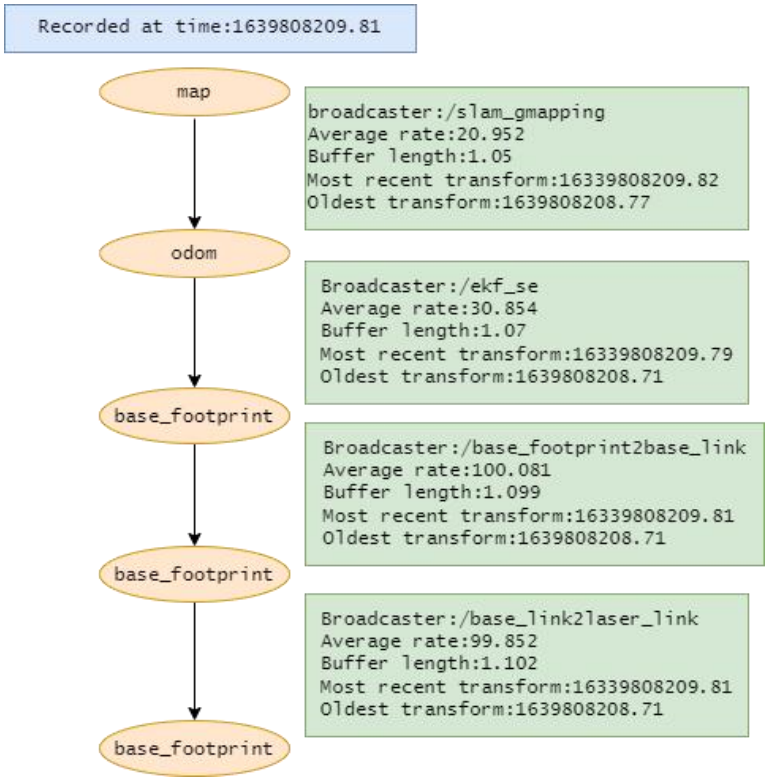

图 2.4 rqt_tf_tree

部分参数分析：

base_frame：base_footprint，机器人底盘坐标系基框架，附带在移动底盘的框架，也是原点。

odom_frame：odom，里程计坐标系里程计框架，附带在里程计的框架。

map_frame: map，地图坐标系地图框架，附带在地图上的框架。

map_update_interval: 0.15 地图更新速度，设置为 0.15 秒。

maxUrange: 16.0，激光最大可用距离，设置为 16 米。

<4> 代码分析

```
    //订阅锥桶坐标节点的回调函数
    void  VirtualWall:.AddWallCalback(const  geometry_msgs::PointStampedConstPtr&
msg)
    {
    geometry_msgs::Point32 point;
    point.x = msg->point.x;
    Point.y = msg->point.y;point.z= msg->point.z;
    wallmax_x = std::max(wallmax_x, msg->point.x);
    wallmax_y= std:max(wallmax_y, msg->point.y);
    wallmin_x = std::min(wallmin_x, msg->point.x);
    wallmin_y = std::min(wallmin_y, msg->point.y);
    if (v_wall.size() == 0)
    {
    virtual_wall::Wall wall;
    wall.id =0;
    wall.polygon.points.push_back(point);
    v_wall.push_back(wall);
    }
    else{
    if (v_wall.back().polygon.points.size() == 1)
    {
    if (v_wall.size() == l)
    {
    v_wall.back().id = 0;
    }else {
    v_wall.back().id =v_wall[v_wall.size() - 2].id + l;
    }
    v_wall.back().polygon.points.push_back(point);
    wallPoint.push_back(v_wall.back());
    //对虚拟墙插值
    WallInterpolation();
    }else if(v_wall.back().polygon.points.size()== 2)
    {
    virtual_wall::Wall wall;
    wall.id = v_wall.size();
    wall.polygon.points.push_back(point);
    v_wall.push_back(wall);
    }
  }
}
```

将融合惯导的视觉里程计和激光雷达进行松耦合检测到的锥桶坐标信息，将该信息作为 topic 发布，使得绘制虚拟墙的节点订阅该消息，通过插值，在两个同侧相邻的锥桶间，补出连续虚拟墙，最终在 costmap 上用虚拟墙将同侧的离散锥桶连接成连续赛道边界，来避免第二圈 ros 导航过程中小车导出赛道。

实验三　锥桶的识别及位置确定实验

实验目的：

1、了解常用的图像处理的方法
2、掌握图像处理各步骤的原理
3、通过 cone 程序加深对 opencv 的学习

实验内容：

运用传统的图像处理方式以实现对红蓝锥桶的识别，位置的确定，进而规划小车前进的路线。在程序中主要运用了 opencv，在 lidar_camera 包里的 cone.cpp 中，具体步骤如下：
1、先将 RBG 转 HSV
2、根据目标颜色进行图像二值化
3、对图像进行腐蚀、膨胀和中值滤波
4、找出目标颜色红色、蓝色的轮廓
5、对轮廓进行多边形拟合
6、找出顶点数在 3-10 之间的多边形
7、找出上边沿比下边沿小的多边形

实验仪器：

ROS 智能车、激光雷达、编码器、惯导传感器

实验原理：

1、RBG 转 HSV

在图像处理中，为了更高效地识别锥桶，需要将摄像头拍得的场地环境照片由 RBG 转为 HSV。RGB 是我们接触最多的颜色空间，也是原始得到的照片颜色，分别为红色(R)，绿色(G)和蓝色(B)。而 HSV 使用色调 H，饱和度 S，明亮度 V 来描述颜色的变化，H 取值范围为 0°～360°，从红色开始按逆时针方向计算，红色为 0°，绿色为 120°,蓝色为 240°。在 cone 中可看到相关定义和数值。

对于 HSV 而言，饱和度 S 越高，颜色对应深而艳。当光谱色的白光成分为 0 时，饱和度达到最高。通常取值范围为 0%～100%，值越大，颜色越饱和。H 表示颜色明亮的程度，对于光源色，明度值与发光体的光亮度有关；对于物体色，此值和物体的透射比或反射比有关。通常取值范围为 0%（黑）到 100%（白）。

从 RGB 到 HSV 的转换公式如下：

设 (r, g, b) 分别是一个颜色的红、绿和蓝坐标，它们的值是 0 到 1 之间的实数

设 max 等于 r，g，b 中的最大者

设 min 等于 r，g，b 中的最小者

$$h = \begin{cases} 0^\circ & \text{if } max = min \\ 60^\circ \times \frac{g-b}{max-min} + 0^\circ, & \text{if } max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{max-min} + 360^\circ, & \text{if } max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{max-min} + 120^\circ, & \text{if } max = g \\ 60^\circ \times \frac{r-g}{max-min} + 240^\circ, & \text{if } max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } max = 0 \\ \frac{max-min}{max} = 1 - \frac{min}{max}, & \text{otherwise} \end{cases}$$

$$v = max$$

2、图像二值化

图像的二值化处理就是将图像上的点的灰度值置为 0 或 255，最终效果即整个图像将呈现出明显的黑白效果。即将 256 个亮度等级的灰度图像通过适当的阈值选取而获得仍然可以反映图像整体和局部特征的二值化图像。在对图像做进一步处理时，二值化操作使得图像的集合性质只与像素值为 0 或 255 的点的位置有关，不再涉及像素的多级值，从而使处理变得简单，而且数据的处理和压缩量较小。为了得到理想的二值图像，一般采用封闭、连通的边界定义不交叠的区域。所有灰度大于或等于阈值的像素被判定为属于特定物体，其灰度值为 255 表示，否则这些像素点被排除在物体区域以外，灰度值为 0，表示背景或者例外的物体区域。

3、膨胀和腐蚀

图像膨胀的过程类似于一个卷积的过程，假设有图像矩阵 A 以及结构元素 B（注意，B 的形状、尺寸没有限制），B 在 A 矩阵上依次移动，每个位置上 B 所覆盖元素的最大值替换 B 的中心位置值（即锚点处），即为膨胀的过程。

图像腐蚀的过程同理，每个位置上 B 所覆盖元素的最小值替换 B 的中心位置值（即锚点处），即为腐蚀的过程。

4、拟合和目标值的选取

对处理后的图像进行红蓝锥桶轮廓的识别，随后进行多边形拟合。由锥桶的形状，找出所得多边形中轮廓点数在 3 和 10 之间的，且需要满足上边沿比下边沿小这一条件，可利用长和宽的计算比较得出。最终实现锥桶的识别和图像的处理这一功能。

实验步骤：

<1> 得到摄像头返回的图片，进行 RGB 到 HSV 的转换，可以使用 tracebar 程序以找出环境合适的 HSV 参数，放入 cone 的 inRange 函数中。对 HSV 图像进行二值化、腐蚀、膨胀和中值滤波等操作。

```
int left1LowH = 156,left1HighH = 180,left1LowS = 100,left1HighS = 255; //红色
int left1LowV = 100, left1HighV = 255;

int left2LowH = 0,left2HighH = 15,left2LowS = 100,left2HighS = 255; //红色
int left2LowV = 100, left2HighV = 255;

int rightLowH = 70,rightHighH = 134,rightLowS = 50,rightHighS = 255; //蓝色
int rightLowV = 10,rightHighV = 250;
int endLowH = 11,endHighH = 32,endLowS = 0,endHighS = 0;  //黄色
  int endLowV = 50,endHighV =255;}

cvtColor(img_src, img_rgb, CV_BGR2RGB);
cvtColor(img_rgb, img_HSV, CV_RGB2HSV);    //rgb 转 hsv 图像
//imshow("img_src",img_src);
//waitKey(0);
inRange(img_HSV, Scalar(left1LowH, left1LowS, left1LowV), Scalar(left1HighH,
left1HighS, left1HighV), left1);
//对目标颜色进行二值化处理结果保存在 left，right 和 end 矩阵中
inRange(img_HSV, Scalar(left2LowH, left2LowS, left2LowV), Scalar(left2HighH,
left2HighS, left2HighV), left2);
//对目标颜色进行二值化处理结果保存在 left，right 和 end 矩阵中
```

第一步处理完后得到的图像如下如所示：

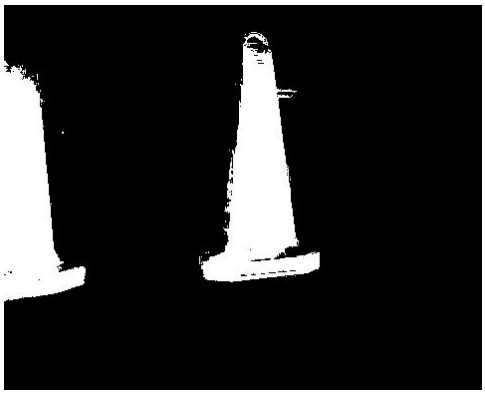

图 3.1　初步处理后的锥桶图像

〈2〉对处理完的图片进行识别，检测出红蓝锥桶的边缘，找出其轮廓，所得如图所示，此后进行多边形拟合，找出其中顶点数在 3 和 18 之间的多边形。

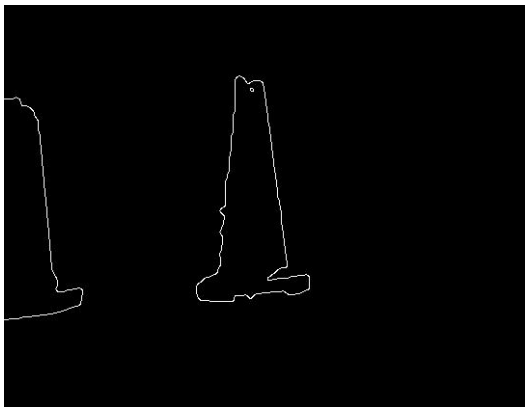

图 3.2　检测提取的锥桶轮廓

<3>对所找出的符合条件的多边形，获取其 ROI 信息（感兴趣区，在图像上的坐标和宽高），并存入数组，检验得到红蓝锥桶后提取在该圆锥面里的激光雷达数据点，中值滤波后求这些点的平均值，作为该圆锥的位置信息。

```
ConeInfo cone_judge(Mat image)
{
    Mat img_all_convex_hulls;
    vector<vector<Point> > contours;
    vector<Vec4i> hierarchy;
    ConeInfo temp_cone;
    temp_cone.amount = 0;
    Mat img_thresh_opened,img_thresh_blurred,img_edges;

    morphologyEx(image, img_thresh_opened, 2, element);  //腐蚀膨胀
    medianBlur(img_thresh_opened, img_thresh_blurred, 5);  //中值滤波
    //imshow("img_thresh_opened",img_thresh_opened);
    //waitKey(0);
    //imshow("img_thresh_blurred",img_thresh_blurred);
    //waitKey(0);
    Canny(img_thresh_blurred, img_edges, 20, 80, 3);
    findContours(img_edges, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));  //提取出轮廓
    //imshow("img_edges",img_edges);
    //waitKey(0);
    vector<vector<Point> >approx_contours(contours.size());
    vector<vector<Point> >all_convex_hulls(contours.size());
    vector<vector<Point> >convex_hulls_3to10(contours.size());
    vector<vector<Point> >convex_hulls_cone(contours.size());

    //ROS_INFO("129");

    for( int i = 0; i < contours.size(); i++ )
    {
        approxPolyDP( Mat(contours[i]), approx_contours[i], 3, true );  //
对轮廓进行多边形拟合
    }
    for( int i = 0; i < approx_contours.size(); i++ )
    {
        convexHull( Mat(approx_contours[i]), all_convex_hulls[i], false );
//对拟合后的曲线提取出凸包
    }

    for( int i = 0; i< all_convex_hulls.size(); i++ )
    {
       /*修改这里*/
        if(all_convex_hulls[i].capacity() >= 3)  //提取出顶点大于 3 小于 15 的
多边形
        {
          convexHull( Mat(all_convex_hulls[i]), convex_hulls_3to10[i],
false );
        }
     }}
```

坐位置数据按（横向距离，纵向距离）的格式得到记录，单位为毫米，程序在 lidar_camera 程序包中的 opencv_lidar 里。效果如下图所示，数据标定在图中：

图 3.3　标定坐标后的带标红色锥桶

图 3.4　标定坐标后的红蓝锥桶

实验四　middle lane、补帧算法下的路径规划实验

实验目的：

1、了解 bresenham 算法及应用；
2、熟练掌握 middlelane 规划；
3、了解并掌握补帧的应用场合；

实验内容：

1、识别定位锥桶，确定边界线；
2、以中线作为路径；
3、若丢失两侧锥桶则补帧；

实验仪器：

ROS 智能车、激光雷达、编码器、惯导传感器等

实验原理：

bresenham 算法是一种光栅化的直线生成算法，是计算机图形学目前使用广泛的直线扫描转换算法，具体逻辑很简单，即通过各行、各列像素中心构造一组虚拟网格线，按照直线起点到终点的顺序，计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列像素

中与此交点最近的像素，所以 bresenham 的算法研究实际上是研究目标点的选择。

其原理为：设直线方程为 y=kx+b，误差项为 d 并初始化为 0。则每当 x 每增加 1 时，y 的值是否增加 1 取决于 d 与 0.5 比较的大小，由 y=k(x+1)+b 得 y 的增量为斜率 k，故 d=d+k。当误差项 d≥0.5 时，下一像素点取值为右上方的点（x+1，y+1），当 d＜0.5 时，则取下一像素点为（x+1，y)。设 e=d-0.5，上述比较等同于 e 与 0 的比较。当 e＞0 时，置新的 e=e-1，当 e＜0，e=e+k。

为实现小车的最优路线规划，在逆时针跑圈条件下，设红色圆锥为左边界，蓝色圆锥为右边界。为了确定左边界，必须先找出所有红色圆锥，在这些圆锥中找出离小车最近的那一个，并将它设为第一个圆锥，再找出离第一个圆锥最近的圆锥，并将它设为第二个，以此类推。右边界同理。伪代码如下:

```
Input: allConeU,allConeV
Output: sortConeU,sortConeV
u[0] ← mapWidth/2
v[0] ← mapHeight
u ← allConeU
v ← allConeV
for i = 1 → coneAmount − 1 do
    for j = 1 → coneAmount do
        coneDistance ← (u[0] − u[j]) * (u[0] − u[j]) + (v[0] − v[j]) * (v[0] − v[j])
    end for
    for out = 0 → coneAmount − 2 do
        for in = 0 → coneAmount − 2 − out do
            if coneDistance[in] < coneDistance[in+1] then
                swap(u[in + 1], u[in + 2])
                swap(v[in + 1], v[in + 2])
                swap(coneDistance[in], coneDistance[in + 1])
            end if
        end for
    end for
    sortConeU ← u[1]
    sortConeV ← v[1]
end for
```

图 4.1　middle_lane 算法伪代码

找到圆锥顺序后，按上述规则排好的顺序，从第一个到最后一个锥桶，依次在圆锥间用 bresenham 算法画线，这样就确定了小车跑圈的边界线，并将边界线在另一个实时图像中显示(online_map），达到俯视图的效果。

找到赛道边界线后转入通过路径规划算法寻找合适的路径。路径规划指的是机器人的最优路径规划问题，即依据某个或某些优化准则（如工作代价最小、行走路径最短、行走时间最短等），在工作空间中找到一个从起始状态到目标状态能避开障碍物的最优路径。

在此采用相对折中的方法——以中线作为路径，这样小车能够更稳定地行驶。为了得到中线，先将左边界线和右边界线进行了四等分，再计算左右对应等分点的像素位置的平均值，这样就得到了四个中间点，将中间点依次连，得到线条，可将其看作中间线。因为边界线是由连续的像素点组成，并且像素点是按圆锥的顺序依次连接的，所以把边界像素点放入了二维向量数组中，数组大小除以 4 就是等分点的间隔值。

将计算结果四舍五入，从而获得整数间隔值。找到中线后需要在中线上设定一个合适距离的局部目标点，将该点的横坐标与车体的横坐标（永远是横轴的一半）进行做差操作，这个偏差值乘以一定的比例系数就可以作为要输出的转向值，也可以尝试用三角函数求出偏角（如下图）乘以一定的比例系数作为转向角。转向角亦称转角，即中线由一个方向转向另一个方向时，转变后的方向与原方向延长线的夹角。在中线交点标定以后，即可测定其转向角。由于中线在交点处转向的不同，转向角有左转角和右转角之分。在线路测量中，一般不直接测转角，而是先直接测转折点上的水平夹角，然后计算出转角。

与此同时，输出固定的速度值，小车即可实现循迹的功能。

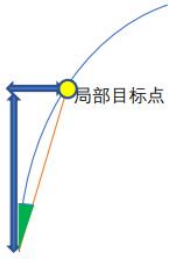

图 4.2 转向角

以上是小车将两侧锥桶均识别成功的情况，取前方左右两侧锥桶中心作为小车循迹目标点。若丢失左侧红色锥桶目标，则会假设未丢失侧锥桶，即蓝色锥桶的左侧一个赛道宽度距离外有一个红色锥桶，以实现拐弯。

由于光照环境、摄像机曝光、小车的速度过快等多方面因素共同影响，在第一圈锥桶识别中，往往会出现同时丢失两侧锥桶的情况，采用补帧算法（即在两侧锥桶均丢失时采用上一帧的局部地图）。在丢失目标后的 Counting 帧内，小车将会保持原速度向前继续运动。由于竞速过程中时间很快，经过实际多次测试，经验取 Couting 值为 1。

实验步骤：

<1> 摆好锥桶，运行 launch 程序

将 middlelane 实现过程写在 launch 文件里，故运行 launch 文件即可在识别锥桶后直接规划出中线路径，如下图两种情况所示。

图 4.3 左侧为路径规划，右侧为锥桶实际实验位置

图 4.4 左侧为路径规划，右侧为锥桶实际实验位置

<2>对代码进行分析，middle_lane 程序取前方左右两侧锥桶中心作为小车循迹目标点。若丢失左侧红色锥桶目标，则会假设未丢失侧锥桶，即蓝色锥桶的左侧一个赛道宽度距离处有一个红色锥桶，以实现拐弯。

```
ConeInfo middle_lane(ConeInfo leftPoint, ConeInfo rightPoint)
{
  static int Point[4]; //ConeInfo Point;
  ConeInfo pointLane, tempLane;
  Point[0] = int(n / 2);
  Point[1] = int(m - 1);
  for (int i = 0; i < GAP; i++)
  {
      vector<int>::iterator l_u = leftPoint.u.begin() + i;
      vector<int>::iterator l_v = leftPoint.v.begin() + i;
      vector<int>::iterator r_u = rightPoint.u.begin() + i;
      vector<int>::iterator r_v = rightPoint.v.begin() + i;
      Point[2] = int((*l_u + *r_u) / 2);
      Point[3] = int((*l_v + *r_v) / 2);
      tempLane = bresenham(Point[0], Point[1], Point[2], Point[3]);
      pointLane.u.insert(pointLane.u.end(), tempLane.u.begin(),
tempLane.u.end());
      pointLane.v.insert(pointLane.v.end(), tempLane.v.begin(),
tempLane.v.end());
      Point[0] = Point[2];
      Point[1] = Point[3];
  }
  return pointLane;
}
```

实验五　路径规划与导航实现

实验目的：

1、了解并运用 move_base 工具包

2、使用相关算法实现路径最优规划

3、尝试掌握全局路径规划和本地实时规划

实验内容：

1、在得到前述定位模板所提供的锥桶的二维定位的基础上，可确定边界线，从而运用 move_base 功能包等相关算法，从而实现小车前进的路径最优规划

2、发布小车相应路径消息，节点 art_car_controller 跟随该路径，通过运动控制模块实现智能车的导航运动

实验仪器：

ROS 智能车、编码器、惯导传感器等

实验原理：

路径规划指的是机器人的最优路径规划问题，即依据某个或某些优化准则（如工作代价最小、行走路径最短、行走时间最短等），在工作空间中找到一个从其实状态到目标状态能避开障碍物的最优路径。

路径规划算法特点总结可得出如下特点：

完备性：起始点与目标点之间有路径解存在，那么一定可以找到解，若找不到解则说明一定没有解存在；

概率完备性：是指若起始点与目标点之间有路径解存在，只要规划及搜索时间足够长，就一定能够确保找到一条路径解；

最优性：规划得到的路径在某个评价指标上是最优的；

渐进最优性：是指经过有限次规划迭代后得到的路径是接近最优的的次优路径，且每次迭代都是与最优路径更加接近，是一个逐渐收敛的过程。

根据对环境信息的把握程度可把路径规划分为基于先验信息的全局路径规划和基于传感器信息的局部路径规划。其中，从获取障碍物信息是静态或是动态的角度看，全局路径规划属于静态规划（又称离线规划）。全局路径规划需要掌握所有的环境信息，根据环境地图的所有信息进行路径规划；局部路径规划只需要有传感器实时采集环境信息，了解环境地图信息，然后确定出所在地图的位置及其障碍物分布情况，从而可以选出从当前节点到某一子目标的最优路径。

1、全局路径规划的环境建模一般采用自由空间法、栅格法，其主要通过 Dijkstra 算法和 A*算法得到最优解：

①Dijkstra 算法：在一个有向赋权图中，从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

具体而言，Dijkstra 算法从物体所在的初始点开始，访问图中的节点。它迭代检查节点集中的结点，并将和该节点最靠近的尚未检查的结点加入待检查点集。该结点集从初始结点向外扩展，直到到达目标节点。Dijkstra 算法保证能够找到一条从初始点到目标点的最短路径。

Dijkstra 算法是一种经典的广度优先的状态空间搜索算法，算法会搜索整个空间直到到达目标点，这就导致了 Dijkstra 算法计算时间和数据量很大，而且搜索得到的大量数据对于移动机器人的运动是无用的。

②A*算法

A*算法：是一种静态路网中求解最短路径的最有效的直接搜索方法。A*算法在 Dijkstra 算法的基础上增加了启发式特性，搜索的效率大大提升。A*算法按照 Dijkstra 算法类似的流程运行，不同的是它能够评估任意结点到达目标节点的代价。与 Dijkstra 算法选择离初始结点最近的结点不同，它根据启发式函数选择离目标最近的节点。A*算法无法保证找到最优路径，但是速度比 Dijkstra 速度快很多。

2、本地实时规划的主要思路如下：

常见的局部路径规划的方法有：人工势场法、动态窗口法。动态窗口法和 A*算法进行融合，构造一种估计全局最优路径评价函数，可实时避障，路径更加平滑，曲率变化的连续性以及可输出的运动控制参数更符合移动机器人动力学控制。

动态窗口法充分考虑了移动机器人的物理限制、环境约束以及当前速度等因素，得到的

路径安全可靠，适用于局部路径规划。由采样机器人当前的状态（dx，dy，dθ），针对每个采样的速度，计算机器人以该速度行驶一段时间后的状态，得出一条行驶的路线，利用一些评价标准为多条路线打分，采取评分最高的路线作为最优解，则小车将按此路线前进。

实验步骤：

<1> 导航需要 ROS 具有 move_base 工具包和 map_sever、rviz 工具包，进行安装后可直接调用

```
  while(n.ok()){
      sensor_msgs::PointCloud cloud;
      cloud.header.stamp = ros::Time::now();
      cloud.header.frame_id = "sensor_frame";

      cloud.points.resize(num_points);

      //we'll also add an intensity channel to the cloud
      cloud.channels.resize(1);
      cloud.channels[0].name = "intensities";
      cloud.channels[0].values.resize(num_points);

      //generate some fake data for our point cloud
      for(unsigned int i = 0; i < num_points; ++i){
        cloud.points[i].x = 1 + count;
        cloud.points[i].y = 2 + count;
        cloud.points[i].z = 3 + count;
        cloud.channels[0].values[i] = 100 + count;
      }

      cloud_pub.publish(cloud);
      ++count;
      r.sleep();
    }
```

<2> 移植功能包：订阅 move_base 发布的路径（由多个连续带方向的点构成）。通过类似于到达目标点判断所使用的定位方法，将智能车当前位姿与路径中的当前点进行对比，使智能车跟随路径运动

<3> 加载建好的地图，其中包含丰富的信息，较为灰色的是全局的代价地图，供全局路径规划使用；深黑色的是局部代价地图，供局部路径规划使用，即提供避障功能。标定起点后，终端显示计算的起点与朝向点，由此进行导航。

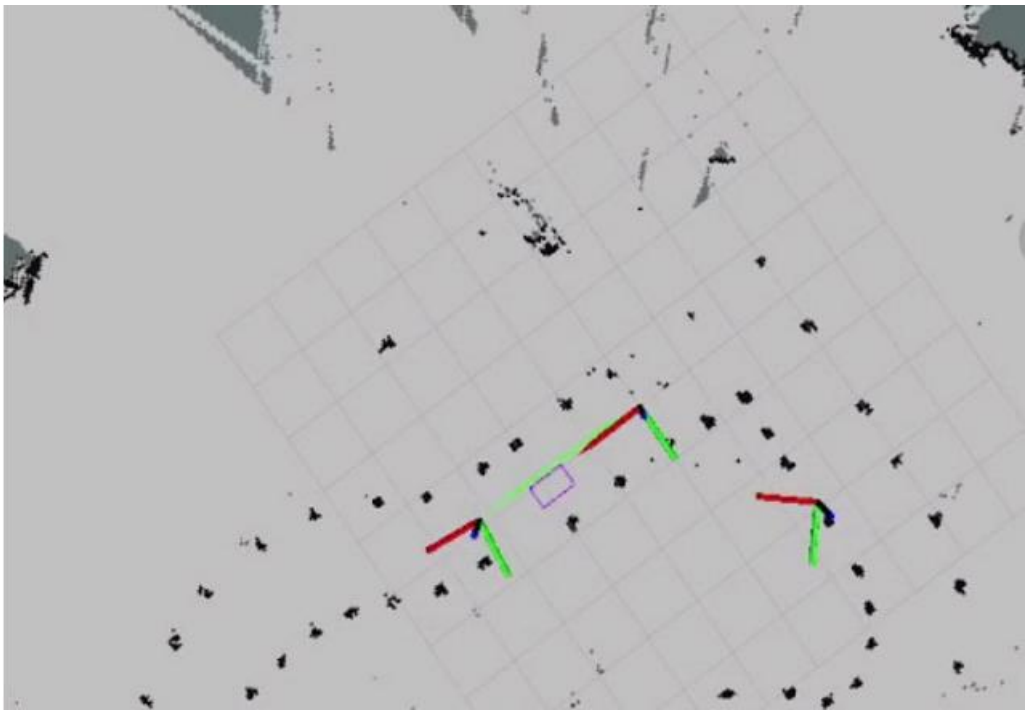

图 5.1 建好的地图