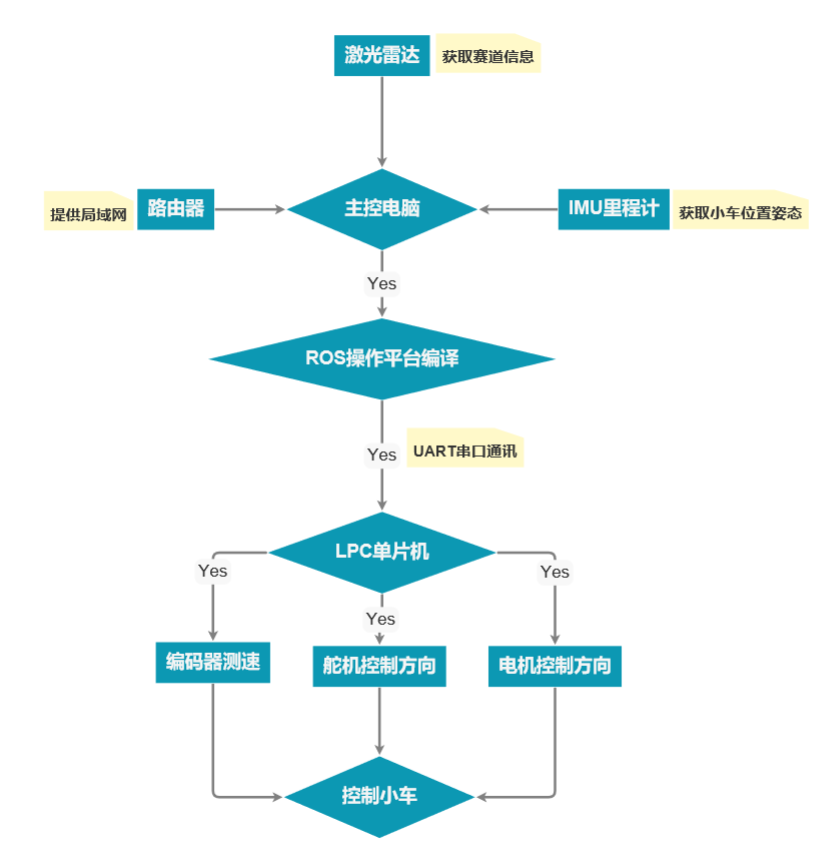
室外光电创意组技术报告草案

**ROS无人驾驶小车整体构架**

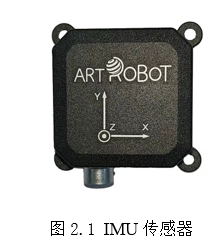


**1.主要思路**

室外光电创意组的赛题需要参赛队伍使用SLAM技术来完成比赛。在比赛之前首先需要利用SLAM技术对比赛的赛道进行构建地图。在比赛过程中利用之前构建的地图进行导航。从赛道的起点导航到赛道的终点。在此过程中需要避开所有障碍物。利用ROS系统许多开源的功能包，使得搭建小车的过程变得简单。我们使用gmapping算法构建地图（SLAM的一种），gmapping算法需要激光雷达的数据，还需要一个精确的里程计消息。对于小车来说，获取里程计的消息有好几种。我们采用的是编码器+IMU的方式获取小车的里程计消息。这种方式获取的里程计消息较为准确。在构建地图时小车可以跑的很快速，且建立出来的地图也很精确。节约了构建地图的时间。在小车运行时，需要从起点跑到终点。并且需要避障。利用编码器和IMU可以进行定位。但是编码器在运动的过程中会有许多噪声（地面不平、碰到小石子…）带来累计误差，使得长时间的定位不准。Amcl可以通过接收激光雷达数据来校正小车的位置。使得定位不受误差累加的影响。有了小车在地图上的位姿后，通过导航功能包，规划出路径，车模在运行过程中，将扫描到的信息引入到代价地图中，使得静态地图变成动态地图，规划出的路径避开障碍物。最后利用几何的方法使得小车跟踪路径，避开障碍物，到达目的地。

**2.里程计IMU**

惯性测量单元（IMU）通常指由3轴加速度计和3轴陀螺仪以及3轴地磁传感器外加一个气压计组成的组合单元，加速度计和陀螺仪安装在互相垂直的测量轴上。通过对加速度计积分可以计算车速。对角速度积分可以计算小车的欧拉角。角速度积分算出的俯仰角、翻滚角可以通过重力加速度在车模x，y，z轴的分量来辅助校正。偏航角可以根据地磁传感器来校正。IMU在小车上可以比较精确的感知小车当前的状态。



IMU里程计在使用过程中会由于碰撞，程序BUG等问题产生误差，所以要及时对里程计进行矫正，方法如下：

1.获取上位机的IP地址

打开Ubuntu系统的终端，输入ifconfig，会出现以下界面

Inet 地址：192.168.43.151即为上位机的IP地址

打开一个新的终端，输入ssh rikirobot 192.168.43.151

再次打开一个终端，输入roslaunch rikirobot bringup.launch

2.IMU矫正开始

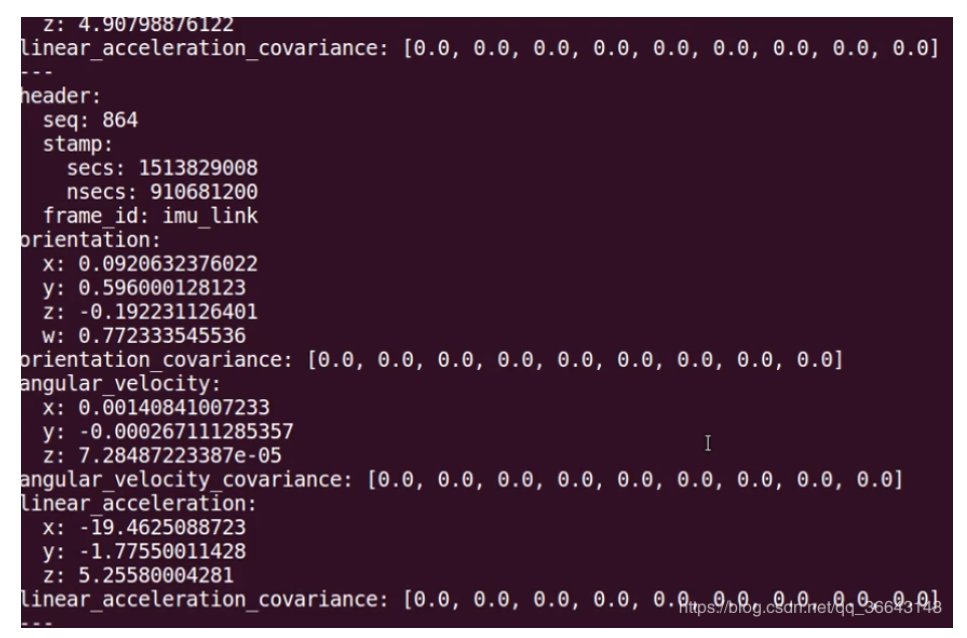
打开一个新的终端，依次输入roscd rikirobot

cd param

cd imu

rostopic echo /imu/data

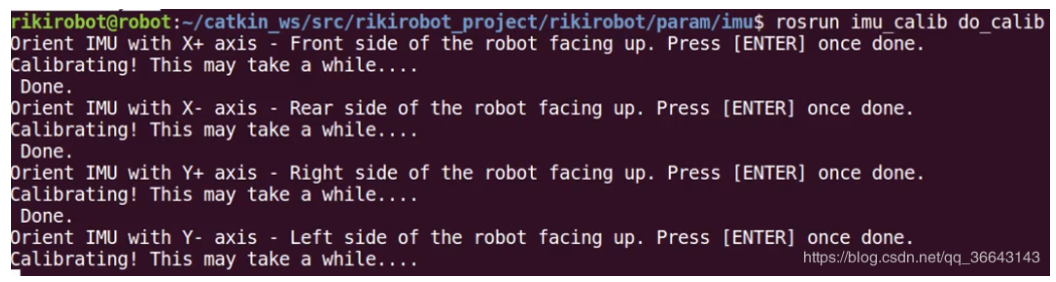
之后会出现以下界面



X轴为小车的线速度—linear

y轴为小车的角速度—angular

z轴速度不做考虑（本车模为四轮舵机式机器人）

再次打开终端，输入自动矫正指令rosrun imu\_calib do\_calib

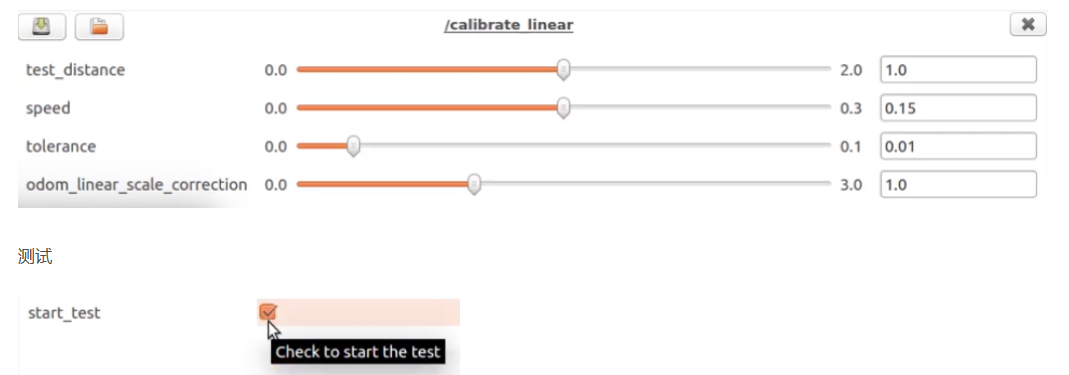
重复两到三次以上操作进行自动矫正。

3.仿真测试矫正结果

打开一个终端，输入rosrun rikirobot\_nav calibrate\_linear.py

再次打开终端，输入rosrun rqt\_reconfigure rqt\_reconfigure

会出现以下界面

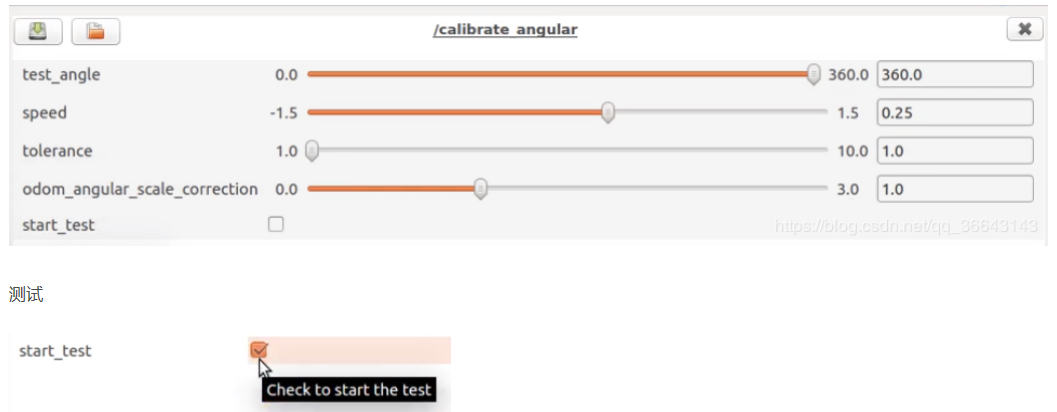


线速度测试

打开一个终端，输入rosrun rikirobot\_nav calibrate\_angular.py

再次打开终端，输入rosrun rqt\_reconfigure rqt\_reconfigure

会出现以下界面（要点击Refresh）



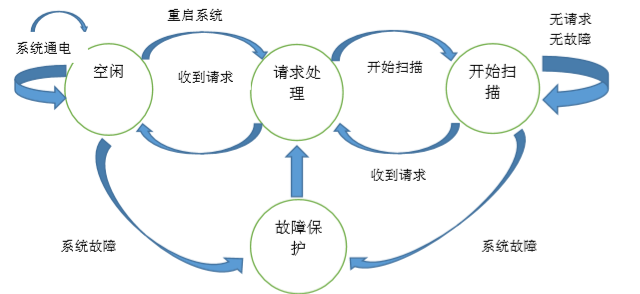
**3.激光雷达**

激光雷达是以发射激光束探测目标的位置、速度等特征量的雷达系统。向目标发射探测信号(激光束),然后将接收到的从目标反射回来的信号(目标回波)与发射信号进行比较,作适当处理后,就可获得目标的有关信息,如目标距离、方位、高度、速度、姿态、甚至形状等参数,从而对周围的物体进行识别。

小车上的激光雷达可以感知环境二维平面的深度信息。程序可以通过环境深度信息来判断障碍物距离小车的远近借此来达到避障的目的。Ros中的slam开源包gmapping可以通过获取激光雷达的数据对周围环境建立地图。通过对激光雷达数据的处理。可以实现对机器人的定位、计算机器人所在二维平面的姿态、计算机器人的速度

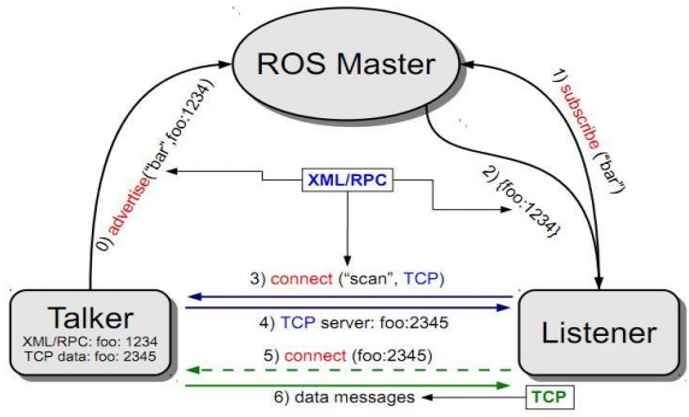


激光雷达采用串口通讯，波特率：230400bps，帧头：0xA5，帧尾：0x81，数据位： 8 bits； 停止位： 1 bits； 数据格式：十六进制

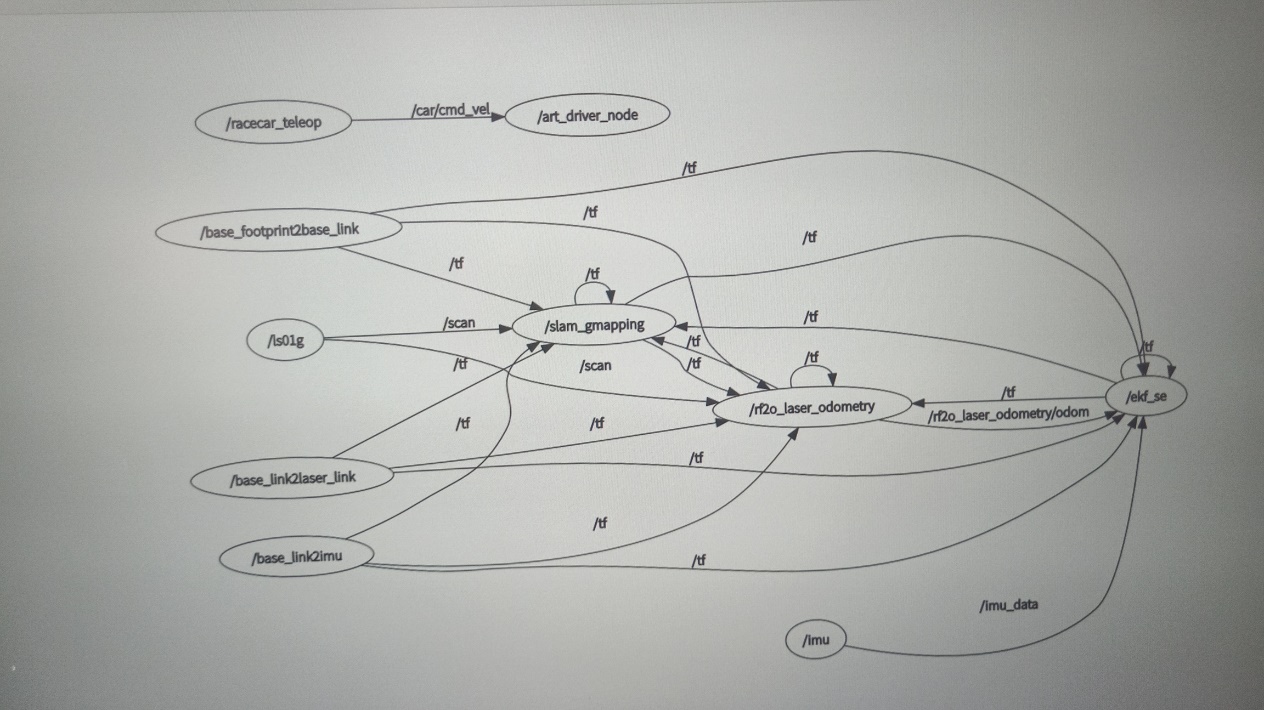
工作状态：

**4.上位机ROS的通讯机制**

ROS系统中的通信方式有三种，分别为话题（msg）、服务（srv）和动作（action），其中话题属于基本的发布/订阅通信方式，适用于单个节点发布消息，一个或多个节点接受消息的情况；服务属于实时获取结果的情况，应用于请求/响应式的应用环境，类似于一个函数调用关系，适用于基本功能的调用或者状态的查看。在使用过程中，我们使用的最多的还是话题这种通信方式。



ROS是一种分布式软件框架，节点之间通过松耦合的方式进行组合，节点可以运行在不同的计算平台上，通过Topic、Service进行通信。ROS的分布式十分强大，在不同计算机上运行不同的节点就如同在一台计算机上运行。

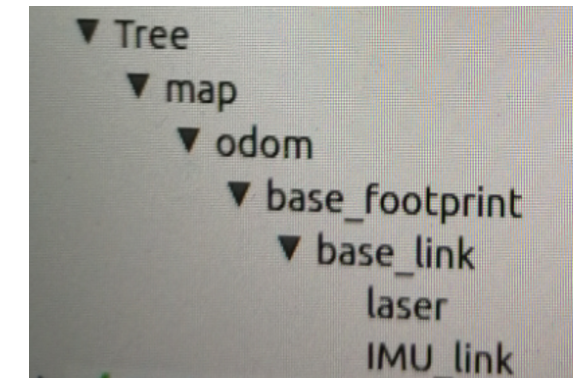


小车中的通讯节点

TF坐标变换：

任何机器人进行移动，都离不开空间的变换，我们所知道：在不同坐标系下，同义一物体的坐标是不同的。而对于本车而言，在雷达坐标系探测下与车体坐标系下的物体的空间位置是不一样的。而我们进行车体的移动，是需要在整个地图坐标系下进行移动的。不同坐标系下的数据最终都需要汇聚到map坐标系下的车体移动。tf功能包提供了存储、计算不同数据在不同参考系之间变换的功能，我们只需要告诉tf树这些参考系之间的变换公式即可，这颗tf树就可以以树的数据结构，管理我们所需要的参考系变换。

机器人运行过程中，激光雷达可以采集到距离前方障碍物的数据，这些数据当然是以激光雷达为原点的测量值，换句话说，也就是base\_laser参考系下的测量值。现在，如果我们想使用这些数据帮助机器人完成避障功能，当然，由于激光雷达在机器人之上，直接使用这些数据不会产生太大的问题，但是激光雷达并不在机器人的中心之上，会始终存在一个雷达与机器人中心的偏差值。这个时候，如果我们采用一种坐标变换，将及激光数据从base\_laser参考系变换到base\_link参考下，问题就解决了。



小车的树状图

一个子坐标系只有一个父坐标系，而一个父坐标系可以有多个子坐标系。

Tf树直观的反应了车体的tf变换关系。

Map是地图坐标系

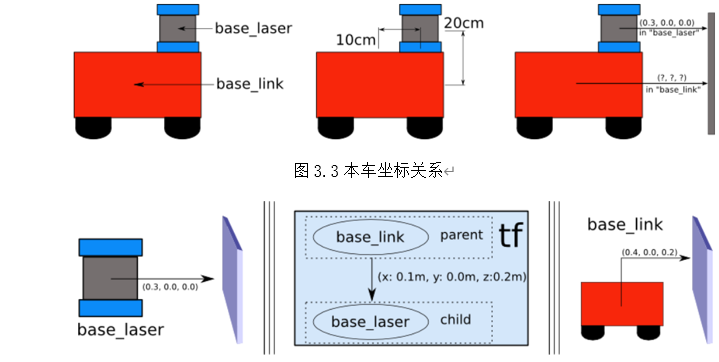
Odom是里程计坐标系

Base\_link 是机器人的基坐标系，一般来说与机器人的中心重合

Laser 是激光雷达的坐标系

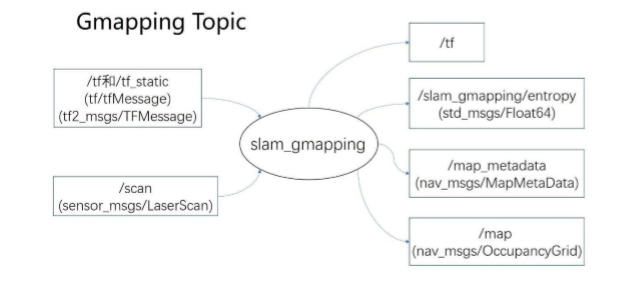
IMU\_link 是IMU的坐标系

具体应用：

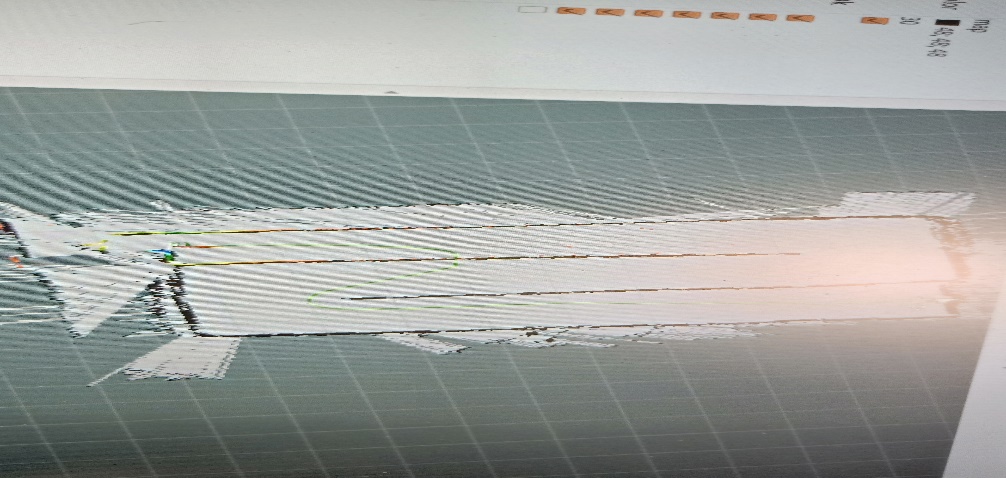


**5.地图建立**

在建立地图时，我们选用已经开发成熟的Gmapping功能包。Gmapping 算法是目前基于激光雷达和里程计方案里面比较可靠和成熟的一个算法,它基于粒子滤波,采用 RBPF 的方法效果稳定，Gmapping可以实时构建室内地图，在构建小场景地图所需的计算量较小且精度较高。相比 Hector SLAM 对激光雷达频率要求低、鲁棒性高（Hector 在机器人快速转向 时很容易发生错误匹配，建出的地图发生错位，原因主要是优化算法容易陷 入局部最小值）。Gmapping 有效利用了车轮里程计信息，这也是 Gmapping 对激光雷达频率要求低的原因：里程计可以提供机器人的位姿先验。随着场景增大所需的粒子增加，因为每个粒子都携带一幅地图，因此在构建大地图 时所需内存和计算量都会增加，并且没有回环检测，因此在回环闭合时可能会造成地图错位，虽然增加粒子数目可以使地图闭合但是以增加计算量和内存为代价。Gmapping 牺牲空间复杂度保证时间复杂度，这就造成 Gmapping不适合构建大场景地图，优化图需要复杂的矩阵运算。这个软件包位于 rosperception 的slam\_gmapping 中。其中的 slam\_gmapping 是一个metapackage,它依赖了 gmapping ,而算法具体实现都在 gmapping 软件包中, 该软件包中的 slam\_gmapping 程序就是我们在ROS中运行的 SLAM节点。



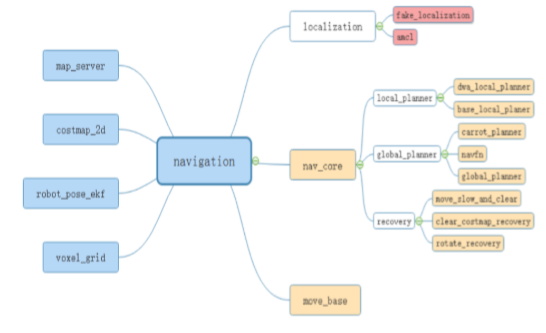
因为Gmapping功能包已经开发的非常成熟，所以在建立地图时我们不需要对其程序进行过多的修改，只需熟练调用就行。如果在建立地图时发现地图的边界线不明显或者出现断层的现象，可以适当提高膨胀系数。



小车构建的地图

**6.导航**

在此小车中导航的程序存放在navigation功能包中。



我们可以简单把 navigation stack 分成 5 个部分：

◼ map\_server:地图服务器，主要功能为保存和导入建好的 slam 地图。

◼ costmap\_2d:可以产生代价地图，以及提供各种相关函数。

◼ nav\_core：包含了导航功能包集的关键接口。他使所有规划器 （planner）和修复行为机制可以插件方式在 move\_base node 中使 用，且必须继承这些接口。

◼ localization：关于定位，这里是两个定位用的工程包。其中fake\_localization 一般是仿真用的，amcl 为实际定位用的工程包。

**◼ move\_base：在 move\_base 中规范了整个导航的流程。对全局路径规 划、局部路径规划和 recovery\_action 的调用都在这个工程包内完 成，可以说是整个导航的核心。**

在 navigation 工程包中，move base 起到宏观调控的作用。move\_base 主要包括以下两部分

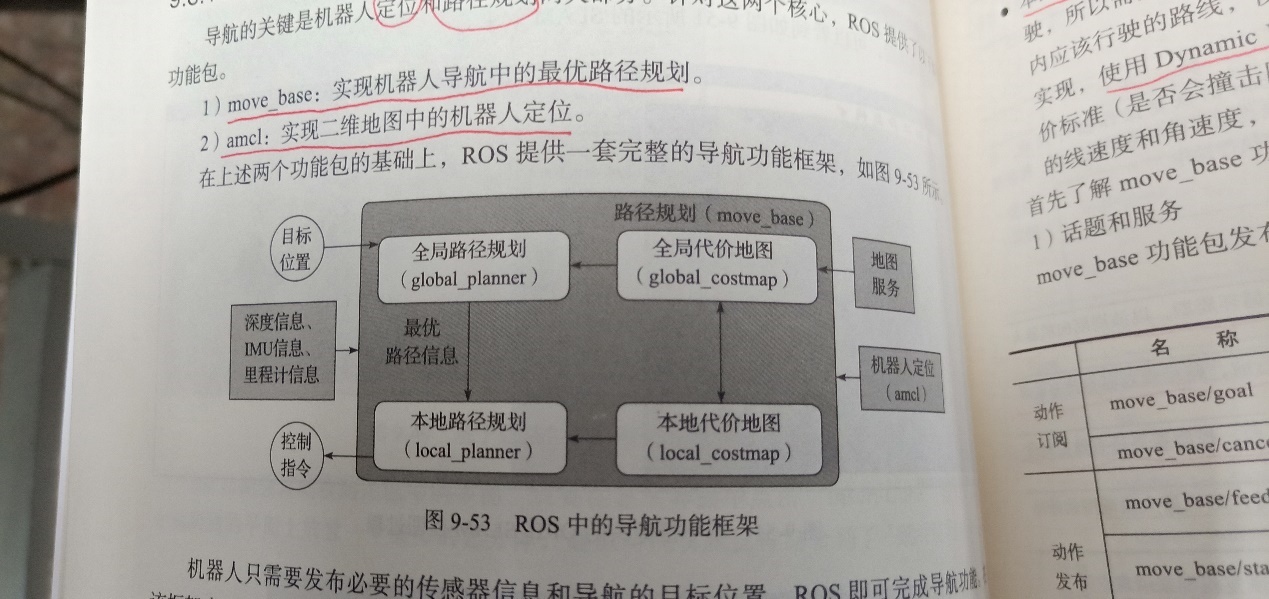
◼ Planner： nav\_core::BaseGlobalPlanner

nav\_core::BaseLocalPlanner

nav\_core::RecoveryBehavior

◼ costmap：global planner costmap

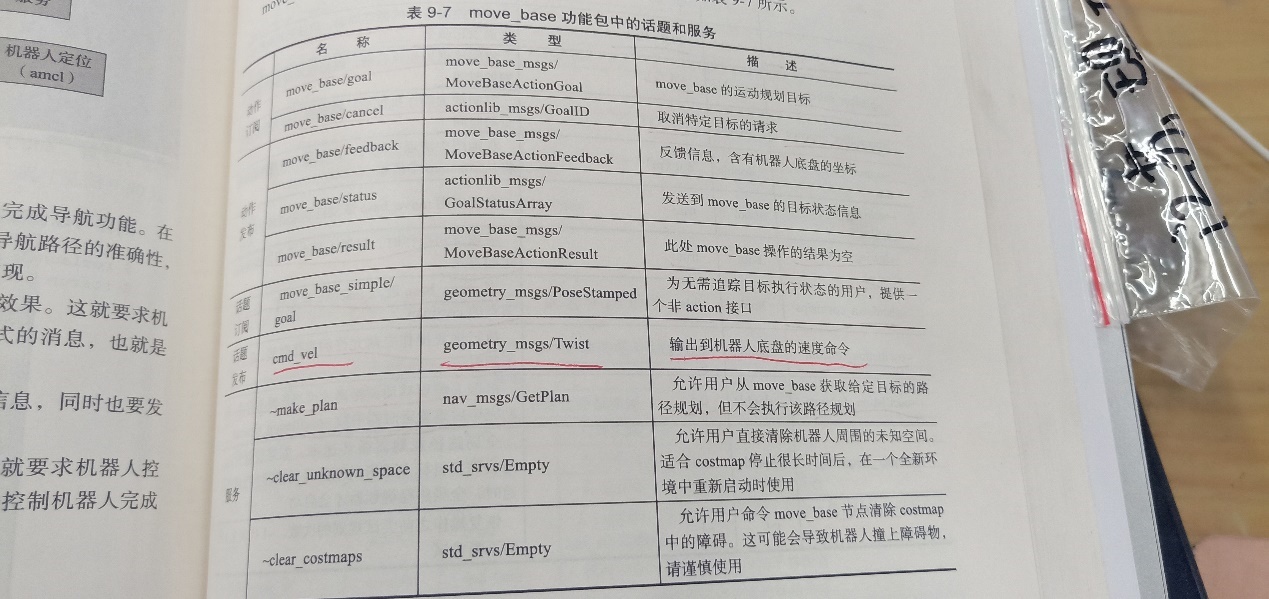
local planner costmap



全局路径规划（global planner）。全局路径规划是根据给定的目标位置和全局地图进行总体路径规划。在导航中，使用Djikstra或A\*算法进行全局路径规划，计算出机器人到目标的最优路线（考虑到竞赛目的这里设置为最短路线）

本地实时路径规划（local planner）在实际情况中，机器人往往无法严格按照全局路线行驶，所以需要针对地图信息和机器人附近随时可能出现的障碍规划机器人每个周期内应行驶的路线，使之尽量符合全局最优路径。本地实时路径由local planner模块实现，使用Dynamic Window Approaches算法搜索避障和进行多条路径规划，综合各评价标准选择最优避障路线。

Move base功能包发布/订阅的动作、话题以及提供的服务：



Move base功能包的参数和代价地图的配置具体查看《ROS机器人开发实践》

ROS开发无人驾驶小车使用的都是目前开发成熟的功能包，我们只需要熟练掌握怎样使用，并且理解各各程序和参数的含义，再根据自己的任务和目标进行调试，基本可以达到正常行驶，但是若想提高，则需要加入其他功能包进行互补，这里我采用TEB功能包（目前我还在学习阶段，暂时没有添加成功）

**7.小车导航的优化**

在添加/更换TEB功能包之前我们需要了解全局定点导航和局部多点导航。

移动这一简单动作，对于人类来说相当容易，但对机器人而言就变得极为复杂，说到机器人移动就不得不提到路径规划，路径规划是移动机器人导航最基本的环节，指的是机器人在有障碍物的工作环境中，如何找到一条从起点到终点适当的运动路径，使机器人在运动过程中能安全、无碰撞地绕过所有障碍物。这不同于用动态规划等方法求得的最短路径，而是指移动机器人能对静态及动态环境作出综合性判断，进行智能决策。总的来说，路径规划主要涉及这3大问题：①明确起点位置及终点;②规避障碍物；③尽可能的做到路径上的优化。

**机器人路径规划有全局与局部规划之分**

根据对环境信息的掌握程度不同，机器人路径规划可分为全局路径规划和局部路径规划。全局路径规划是在已知的环境中，给机器人规划一条路径，路径规划的精度取决于环境获取的准确度，全局路径规划可以找到最优解，但是需要预先知道环境的准确信息，当环境发生变化，如出现未知障碍物时，该方法就无能为力了。它是一种事前规划，因此对机器人系统的实时计算能力要求不高，虽然规划结果是全局的、较优的，但是对环境模型的错误及噪声鲁棒性差。而局部路径规划则环境信息完全未知或有部分可知，侧重于考虑机器人当前的局部环境信息，让机器人具有良好的避障能力，通过传感器对机器人的工作环境进行探测，以获取障碍物的位置和几何性质等信息，这种规划需要搜集环境数据，并且对该环境模型的动态更新能够随时进行校正，局部规划方法将对环境的建模与搜索融为一体，要求机器人系统具有高速的信息处理能力和计算能力，对环境误差和噪声有较高的鲁棒性，能对规划结果进行实时反馈和校正，但是由于缺乏全局环境信息，所以规划结果有可能不是最优的，甚至可能找不到正确路径或完整路径。全局路径规划和局部路径规划并没有本质上的区别，很多适用于全局路径规划的方法经过改进也可以用于局部路径规划，而适用于局部路径规划的方法同样经过改进后也可适用于全局路径规划。两者协同工作，机器人可更好的规划从起始点到终点的行走路径。

TEB 局部规划算法的思想是：整个运动路径比作为一条橡皮筋，连接起始点，

目标点，并让这个路径可以变形，变形的条件就是将所有约束当作橡皮筋的外

力。起始点和目标点由全局规划器指定，在中间插入 N 个控制橡皮筋形状的控制点（控制对象的姿态），并在点与点之间定义运动时间，形成一个路径点序列与时间序列对应合并的一个新的序列，再通过加权多目标优化获取最优的路径点。（这里只做简单介绍，具体教程参考：<https://www.ncnynl.com/archives/201809/2598.html>）

**8.上位机与下位机的通讯**

在激光雷达和IMU获取信息经过电脑运算后，需要发送到单片机来控制小车进行运动，这里就涉及到了上位机与下位机的通讯算法。

电脑接口通讯规则设定及上位机发送

udev 是Linux的设备管理器,动态地创建和删除节点的硬件设备。默认情况下,硬件设备连接到 Linux(Ubuntu)电脑将属于根用户。这意味着运行的任何程序(例如 ROS 节点)作为 unpriveleged(即不是根)的用户将不能访问它们。在我们的小车运行过程中，udev就担当着微型电脑与单片机通信的作用，即给电脑端的接口设定规则。匹配的udev允许一部分设备管理器匹配规则，主控电脑将尝试匹配所有新设备时的插入,所以重要的是规则具体只足以捕捉你正在寻找的设备。有许多潜在的匹配标签,选择有用的最好的方法是让所有的设备直接更改 udev属性。我们现在用到的配置串口的方式，是利用权限运行了art\_init.sh 文件，内部包含car.rules laser.rules imu.rules运行原理如下：

内核通常会产生自定义的设备名如sda1 sdb1等，但是udev可以通过设备的其他信息如总线，生产商等不同来区分不同的硬件设备，在/etc 的目录下。通过 lsusb 命令可以查看到所有外接 usb 的接口设备的描述列表，可以得到id product 和 id VENDOR 的数值，来进行后续的绑定。ls/dev 可以查看 所有外接串口设备的名字列表。首先，所有制定的udev规则最终会在/etc/ udev /rule.d 这个文件夹下面。通过芯片的特定码，来产生软链接别名，绑定串口设备，会在 /etc/udev/rule.d 下产生相应的 规则文件，ttyUSB 产生新的别名。但是有时候，很多厂商由于特殊原因，在串口芯片上，写入的设备号的名字都是一样 的，所以要用到另外一个办法，就是通过绑定设备的硬件端口号 ，如下命 令：

Echo“KERNELS==”3

1.1", MODE:="0666", GROUP:="dialout", SYMLINK+="usb\_0"' >/etc/ude v/rules.d/usb.rules 我们可以用设备的硬件端口号，在 udev 添加端口设备规

则进行串口的绑定，只不过绑定的是 具体的 usb 硬件对应的 KERNELS 值。

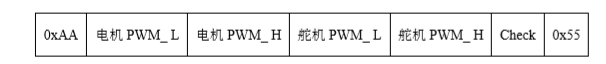
例如 car.rules 中，我们用到的方式即为：

KERNEL=="ttyUSB\*",ATTRS{idVendor}=="1a86",ATTRS{idProduct}==" 7523", MODE:="666", SYMLINK+="car"

在配置好电脑udev之后便可以通过串口向下位机发送数据，程序如下：

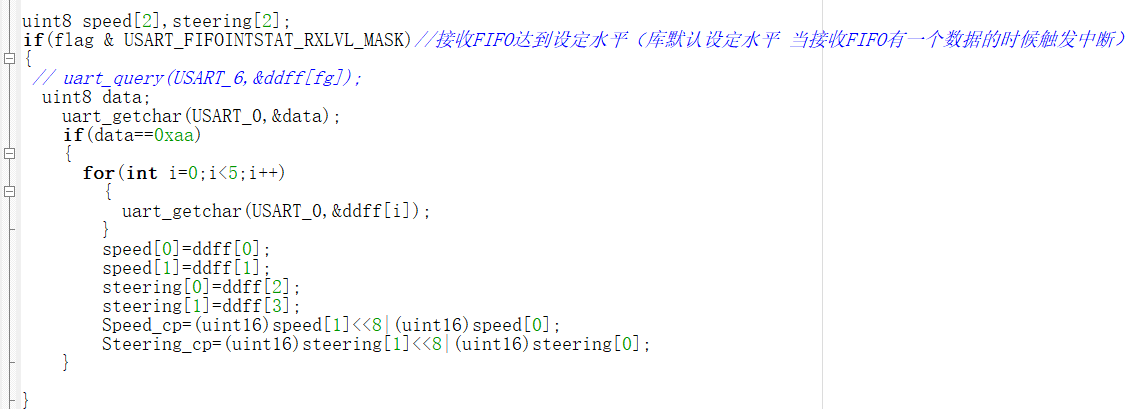


通讯协议：

波特率为38400，8位数据位1位停止位

下位机接收

在单片机的接收中我采用中断接收，即8位产生中断，接收一次数据。但是电脑发送的是16位的数据，所以要进行按位运算进行转换，程序如下：



Speed\_cp即为速度信息，Steer\_cp即为方向信息，在经过pid运算后输出给舵机和电机，即可实现小车的自动行驶。

**9.我在学习中收藏的资料**

https://www.jianshu.com/p/3599d2782683

http://blog.itpub.net/69914126/viewspace-2642753/

https://blog.csdn.net/fantasysolo/article/details/80910222

https://blog.csdn.net/cungudafa/article/details/89065204

https://blog.csdn.net/sinat\_34130812/article/details/81666728

https://github.com/huchunxu/ros\_21\_tutorials

http://wiki.ros.org/ROS/Tutorials