嵌入式物联网方向综合实践

智能车线上仿真竞赛

技 术 报 告

学　　校：　南京大学

队伍编号：　第 5 组

团队成员：　贺伟 侯为栋 胡文聪 黄昱霖

　　李晓康 蒲中正 刘璐 李洪妍

带队教师：　刘海涛

1 方案概述

通过对问题的分析，我们将解决方案分为小车运动控制、建图、定位、导航（包括全局路径规划和局部路径规划）四个模块，经过对各模块可能算法的理论分析和实践测试，我们的解决方案使用 PID 进行小车运动控制，使用 GMapping 进行建图， 使用 AMCL 进行定位，使用 Dijkstra 算法进行全局路径规划，使用 TEB 算法进行局部路径规划。

2 问题描述

2.1 问题概述

本方案需要解决的主要问题是：在给定的地图下，从规定起点出发，在最短的时间内使小车到达规定终点。具体而言，需要小车在出发前规划好基本初始路径，出发后在赛道进行自主导航及避障，在基本初始路径的基础上进行实际路径的局部调整，关键性问题在于小车的运动控制、建图、定位和导航。

2.2 子问题一：运动控制

车辆按照规划路径运行需要合适的控制算法，常见的控制算法包括：PID 算法、模型预测控制、纯跟踪算法、神经网络。考虑到工程具有一定程度计算量，控制输出与控制响应具有时间差，模型预测控制与神经网络对模型要求较高，纯跟踪算法无法满足竞速需求，而 PID 具有使用简单和方便调参的特点，因此本方案选择 PID 算法。经过实际测试，PID 算法可以满足控制需求。

2.3 子问题二：建图

本问题中，小车要完成从规定起点发车、到终点停车的任务，需有一个全局先验地图为小车提供大致的行驶方向。本方案中，传感器数据方面，可以使用的传感器包括激光雷达和里程计，环境为与地面垂直的墙壁围成的固定赛道，需要通过这两种传感器对环境进行感知并提供建图数据。建图算法方面，我们使用 SLAM 进行建图，常用的SLAM 算法有：Gmapping、Cartographer 等，相比于 Cartographer，Gmapping 鲁棒性更高、在构建小场景地图时不需要太多的粒子并且没有回环检测，因此计算量小于 Cartographer，但精度并没有差太多。考虑到需构建地图的场景较小、赛道地面较为平坦、环境较为单一，本方案选择 Gmapping 算法。

2.4 子问题三：定位

导航技术及无人驾驶技术中，要求车辆对自身所处环境进行识别和定位。本方案中，传感器数据方面，可以使用的传感器包括激光雷达和惯性测量单元，环境为与地面垂直的墙壁围成的固定赛道，需要通过这两种传感器对环境进行感知并提供定位数据。定位算法方面，我们使用应用范围较广的自适应性蒙特卡罗方法。

2.5 子问题四：导航

导航过程即路径规划过程，小车的路径规划是根据给出的驾驶任务和实时变化环境智能决策为小车提供可行驶区域和驾驶引导的过程，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地图数据的情况下确定最优路径，局部路径规划是在全局路径规划生成的可行驶区域路线指导下，根据小车行驶各阶段传感器感知到的局部环境信息制定出小车最优的可控行驶路径。本方案中，面临的道路条件为放置锥桶的赛道，需要通过传感器，在前 述的建图和定位工作的基础上完成全局路径规划，并且对行驶过程中遇到的锥桶进行自主避障，即局部路径规划。路径规划算法方面，全局规划算法使用较广的有 A* 和 Dijkstra 算法，局部规划算法使用较广的有 DWA 和 TEB 算法，理论分析方面对全局规划算法进行了比较，A* 算法通过比较当前路径栅格邻居的启发式函数值 F 来逐步确定下一个路径栅格，当存在多个最小值时 A* 算法不能保证搜索的路径最优，且 A* 算法规划出的路径相对 Dijkstra 平滑度较差，增大小车行驶困难程度，结合实践测试，发现使用 Dijkstra 算法作为全局规划算法、使用 TEB 算法作为局部规划算法这一方案效果最佳，并最终采用这一方案。

3 技术方案

3.1 智能车运动控制

本部分主要阐述 PID 控制方案，辅以介绍小车采用的模拟运动和优化方案。

3.1.1 PID 控制方案[6]

3.1.1.1 比例控制 P

比例控制是一种最简单的控制方式。系统偏差一旦产生，控制器立即产生控制作用使被控量朝着减小偏差的方向变化。
对于小车系统，假如小车当前的位置存在误差，那么小车控制器将通过改变速度或者改变行驶方向使得误差被消除，误差的大小决定了小车速度改变或方向改变的程度。

3.1.1.2 积分控制 I

小车系统的运动需要克服惯性力、阻尼力，需要满足一定的动力学方程，这就意味着又动力学因素，小车系统从一个稳态过渡到新的稳态，或小车受扰动作用又重新平衡后，小车系统会出现的偏差，这种偏差称为稳态误差。
为了消除稳态误差，引入积分控制。积分项是对误差取决于时间的积分，随着时间的增加，积分项会增大。这样即便误差很小，积分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到等于零。

3.1.1.3 微分控制 D

通过加入积分项，我们消除了稳态误差，但也引入了另一个问题：在误差消除之前，积分项所累积的被控量可能已经超过了实际需求。这样，当小车系统运动到超过目标位置时，只能通过“负误差”的累积来减掉多余的被控量。更糟糕的是，这种调整可能反复出现，体现在图像上，就是小车系统在目标位置附近反复振荡。实际上，比例控制P也引发了类似的效应，这种振荡削弱了小车的稳定性。
为了预测误差变化的趋势，引入微分的控制器，这样就能够提前使抑制误
差的控制作用等于零，甚至为负值，从而避免了被控量的严重超调。

3.1.2 PID 数值调节

由课程提供的车模中已经配置好小车的各运动关节，但是要操控小车更好地移动就需要配置其运动控制器。配置见相关技术实现部分。

3.1.2.1 速度 PID

根据开源仿真小车模型，小车的转速 pid 有四条命令语句，分别为前后左右四轮铰链的速度 pid。在开源小车中已经对其数值进行了预设，预设值均为：

```
pid: { p: 1000, i: 0.0, d: 0.0, i_clamp: 0.0 }
```

3.1.2.2 转向 PID

仿真小车的转向通过前驱两车轮的转动实现，因此设置有左右两个转向
PID 控制节点。在开源小车中同样已经对其数值进行了预设，预设值均为：

```
pid: { p: 10.0, i: 0.0, d: 0.5 }
```

3.1.2.3 参数调节方案概述

整体运用控制变量法思路，分别对速度 pid 和转向 pid 进行以下三步获取最优的参数值：
第一步：首先控制 i 与 d 为原本的预设值不变，通过改变 p 值（为简化实验次数，先将步长设置为较大的值，如 500；再在精化的区间中减少步长以获取更精确的最优值）、多次实验测试小车运动从同一起点到同一终点所用时间，时间最短对应的 p 值为最优的 p 值；
第二步：将 p 值设置为最优值后，控制 d 为原本的预设值不变，通过改变i值、多次实验（实验方法基本同第一步）得到最优的 i 值；
第三步：再将 i 值设置为最优值后，通过改变 d 值、多次实验（实验方法同第二步）得到最优的d值。

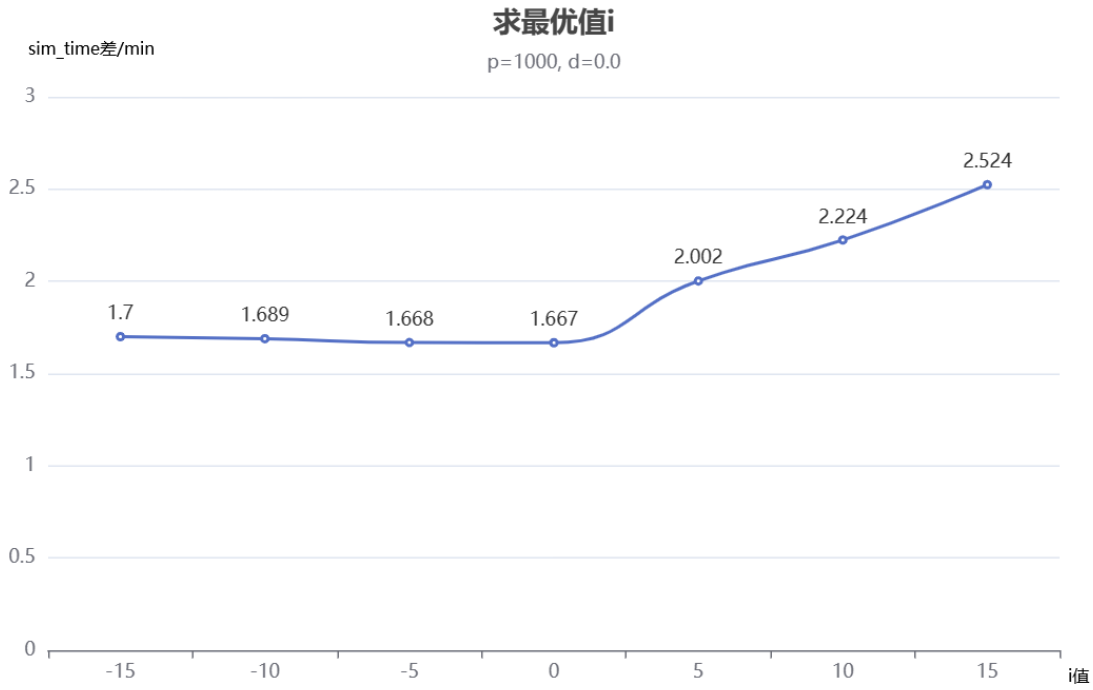
3.1.2.4 参数调节具体方案

注：不在考虑范围的 pid 值，小车容易发生撞墙从而无法到达终点（容易撞墙指：十次实验中超半数次撞墙）

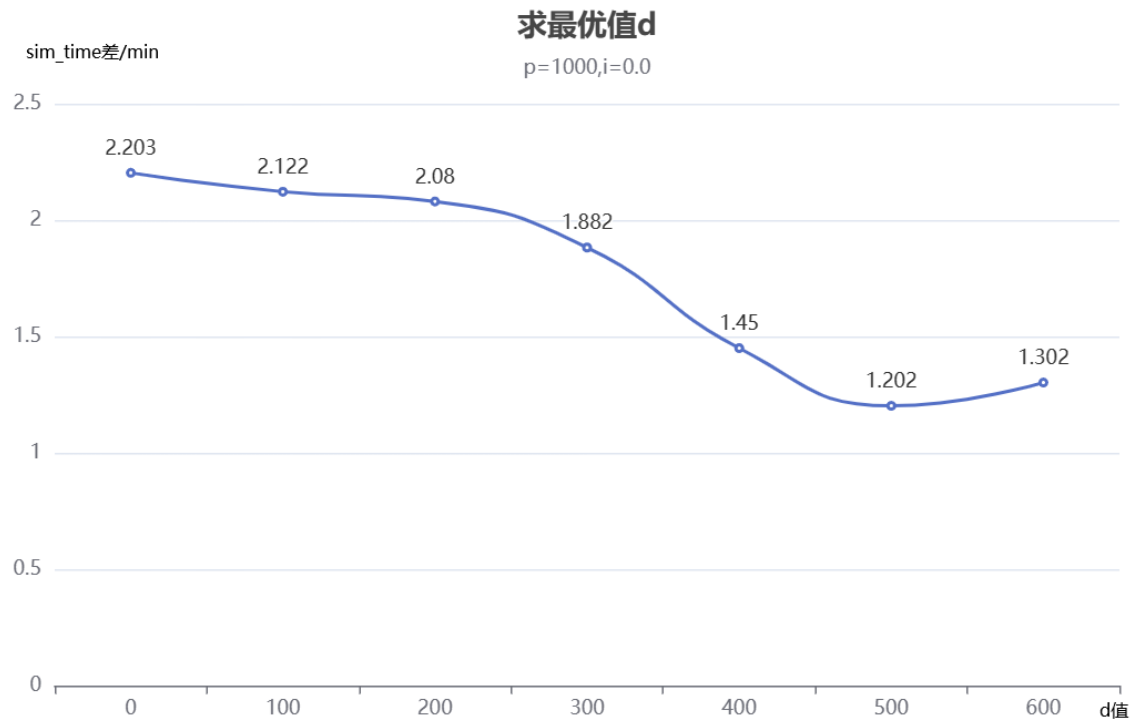
3.1.2.4.1 速度 PID 调节

为了保持小车稳定，改变参数时将四个轮子的转速 pid 参数调节为一致的值。
控制 i、d 为预设值，求最优 p 值：

由实验可知，当 i=d=0.0 时，p 取1000，小车从起点到终点所用平均时间最少。
在最优 p 值情况下，控制 d 值为预设值，求最优 i 值：



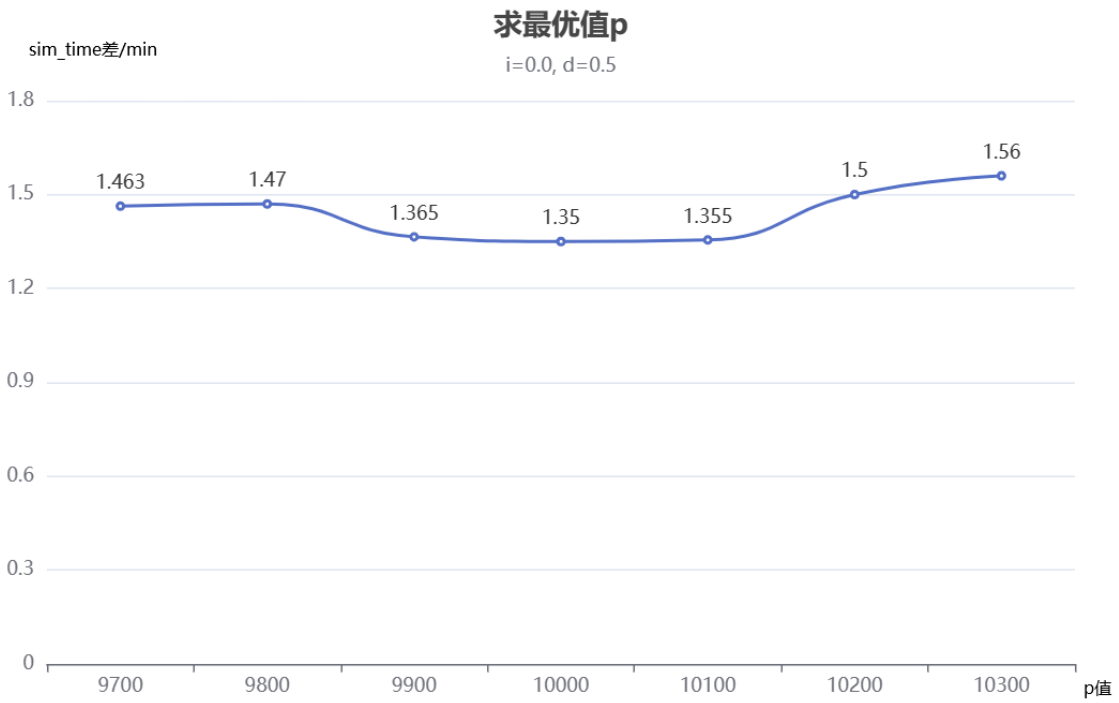
由实验可知，当 p=1000、d=0.0 时，i 取 0，小车从起点到终点所用平均时间最少。
在最优值 p、i 下，求最优值 d：



由实验可知，当 p=1000、i=0.0 时，i 取 500，小车从起点到终点所用平均时间最少

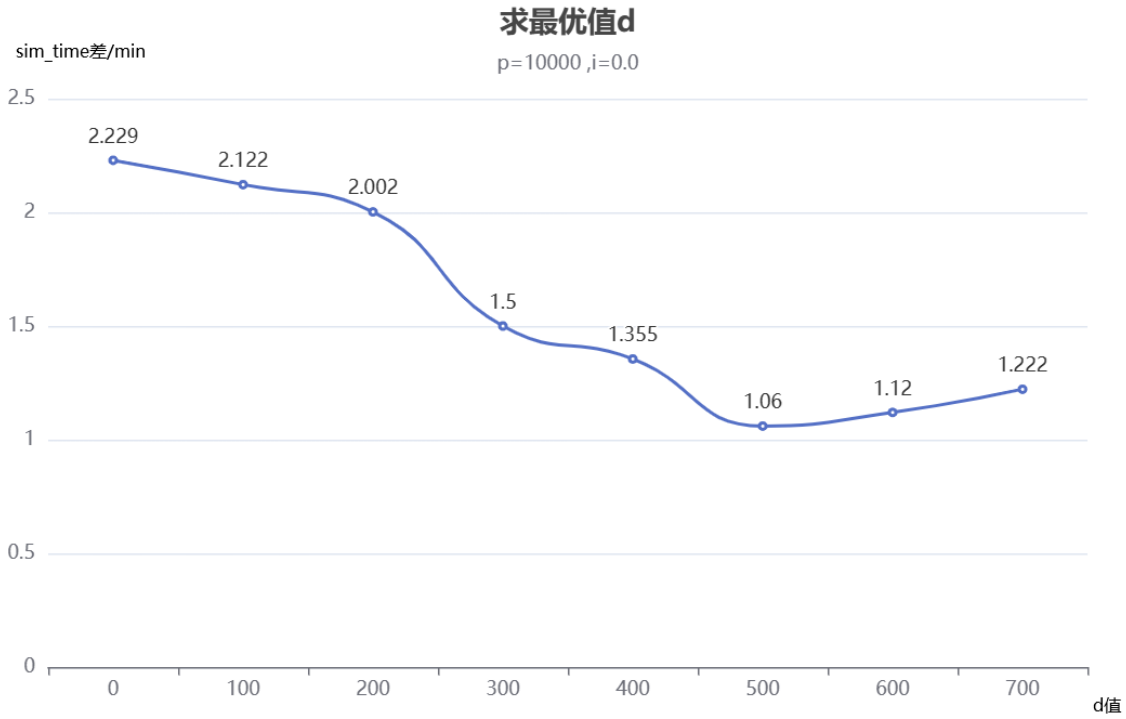
3.1.2.4.2 转向 PID 调节

为了保持小车稳定，改变参数时将两个前轮的转向 pid 参数调节为一致的值。
控制 i、d 值为预设值，求最优值 p：



由实验可知，当 i=0.0、d=0.5时，p 取10000，小车从起点到终点所用平均时间最少。
在最优值 p 下，控制 d 值为预设值，求最优值 i：

由实验可知，当 p=10000、d=0.5 时，i 取 0，小车从起点到终点所用平均时间最少。
在最优值 p、i 下，求最优值 d：



由实验可知，当 p=10000、i=0 时，i 取500，小车从起点到终点所用平均时间最少。

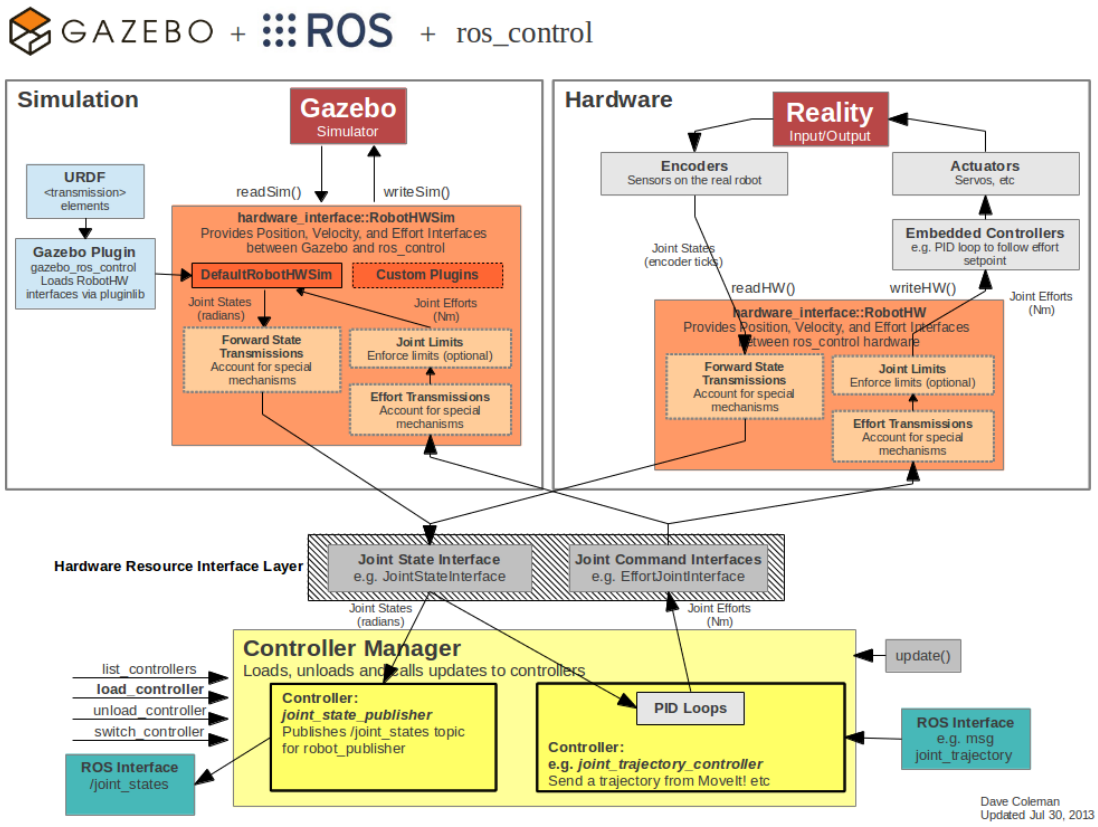
3.1.2.4.3 最优 PID 方案

由上述六个实验可知，最优 PID 方案如下：

```
# 速度
pid: { p: 1000, i: 0.0, d: 500, i_clamp: 0.0 }
# 转向
pid: { p: 10000, i: 0.0, d: 500 }
```

3.1.2 小车模拟控制

我们采用 gazebo + ros_control 的方式来进行小车的模拟运动控制。
gazebo 和 ros_control 的协作关系如下图所示：



若要将二者结合，则需要在模型中定义 transmission 部分，以及应用 gazebo_ros_control 插件。具体使用方法不赘述，可见相关官方文档。

3.1.3 消息传递优化

对于多个节点之间通信产生的大量资源消耗问题，我们选择使用 nodelet 来缓解这一情况。
ROS nodelet 是一种在同一进程中运行多个节点，并且节点之间的数据传输无需拷贝就可以实现。
同时，在系统中有不同的消息源可以发布小车的运动控制指令，这些指令通过 nodelet 加载的插件节点来统一接收并发布给下层运动控制模块。
具体使用方法见技术实现部分。

3.2 建图

目前较为流行的激光SLAM算法有 GMapping 和 Cartographer，我们打算从中选择一个作为我们的建图算法。

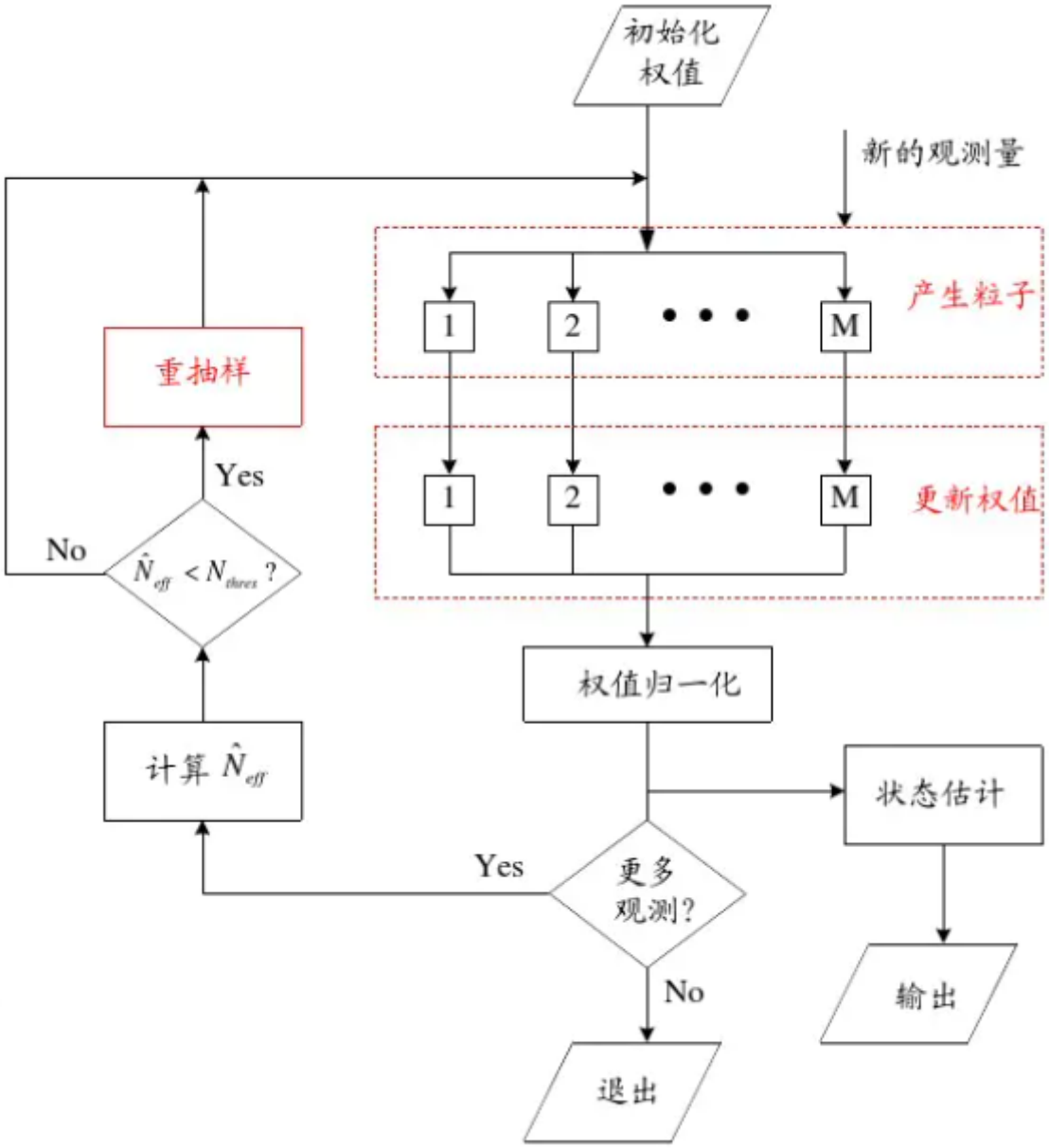
3.2.1 GMapping算法

Gmapping 是一个基于 2D 激光雷达使用改进后的 RBPF（Rao-Blackwellized Particle Filters）算法完成二维栅格地图构建的 SLAM 算法。

3.2.1.1 未改进的 RBPF 算法过程

1. 预测阶段：粒子滤波首先根据状态转移函数预测生成大量的采样，这些采样就称之为粒子，利用这些粒子的加权和来逼近后验概率密度。
2. 校正阶段：随着观测值的依次到达，为每个粒子计算相应的重要性权值。这个权值代表了预测的位姿取第个粒子时获得观测的概率。如此这般下来，对所有粒子都进行这样一个评价，越有可能获得观测的粒子，获得的权重越高。

3. 重采样阶段：根据权值的比例重新分布采样粒子。由于近似逼近连续分布的粒子数量有限，因此这个步骤非常重要。下一轮滤波中，再将重采样过后的粒子集输入到状态转移方程中，就能够获得新的预测粒子了。
4. 地图估计：对于每个采样的粒子，通过其采样的轨迹与观测计算出相应的地图估计，在新的观测值到达时从头评估粒子的权重，当轨迹的长度随着时间的推移而增加时，这个过程的计算复杂度将越来越高。



3.2.1.2 GMapping 中对 RBPF 算法的优化

1. 改进重要性概率密度函数

为了获得下一迭代步骤的粒子采样我们需要在预测阶段从重要性概率密度函数中抽取样本。显然，重要性概率密度函数越接近目标分布，滤波器的效果越好。典型的粒子滤波器应用里程计运动模型作为重要性概率密度函数。这种运动模型的计算非常简单，并且权值只根据观测模型即可算出。然而，这种模型并不是最理想的。当机器人装备激光雷达（如 SICK，Hokuyo 等）时，激光测得的数据比里程计精确的多，因此使用观测模型作为重要性概率密度函数将要准确的多。Doucet 等提出了最优重要性概率密度函数并加入到算法过程中：首先根据运动模型对机器人下一时刻位姿进行预测，得到预测的状态值并且对其进行采样；其次通过最优概率密度函数对各个粒子进行权值的计算；之后进行重采样，根据粒子的权重重新分布粒子，为下次预测提供输入；最后根据粒子的轨迹计算地图的后验概率密度函数。

通过将最近的里程计信息与观测信息同时并入重要性概率密度函数中，使用匹配扫描过程来确定观察似然函数的分布区域，这样就把采样的重点集中在可能性更高的区域。

2. 增加自适应重采样技术

在重采样期间，低权值的粒子通常由高权值的采样代替。由于用来逼近目标分布使用的粒子数量是有限的，所以重采样步骤非常重要。重采样步骤也可能把一些好的粒子滤去，随着的进行，粒子的数目会逐渐减少，最后导致粒子耗尽使该算法失效。Doucet 等为了减少进行重采样步骤的次数，提出了一种理论判定方法来判定是否需要进行重采样。只有当下降到阈值以下时，才进行一次重采样。由于重采样只在需要时进行，进行重采样的次数将大大减少。多次的实验证明这种方法大大降低了将好粒子滤去的风险。

3.2.1.3 优化效果

实验证明改进算法所需的粒子数量通常比原始 RBPF 算法所需的粒子数小一个数量级，并且使用了一种更准确的方式分布采样粒子。同时通过使用基于有效样本量的自适应重采样策略，减少了粒子滤波器中不必要的重采样过程的次数，从而大大降低了粒子耗尽的风险。

3.2.1.4 优点

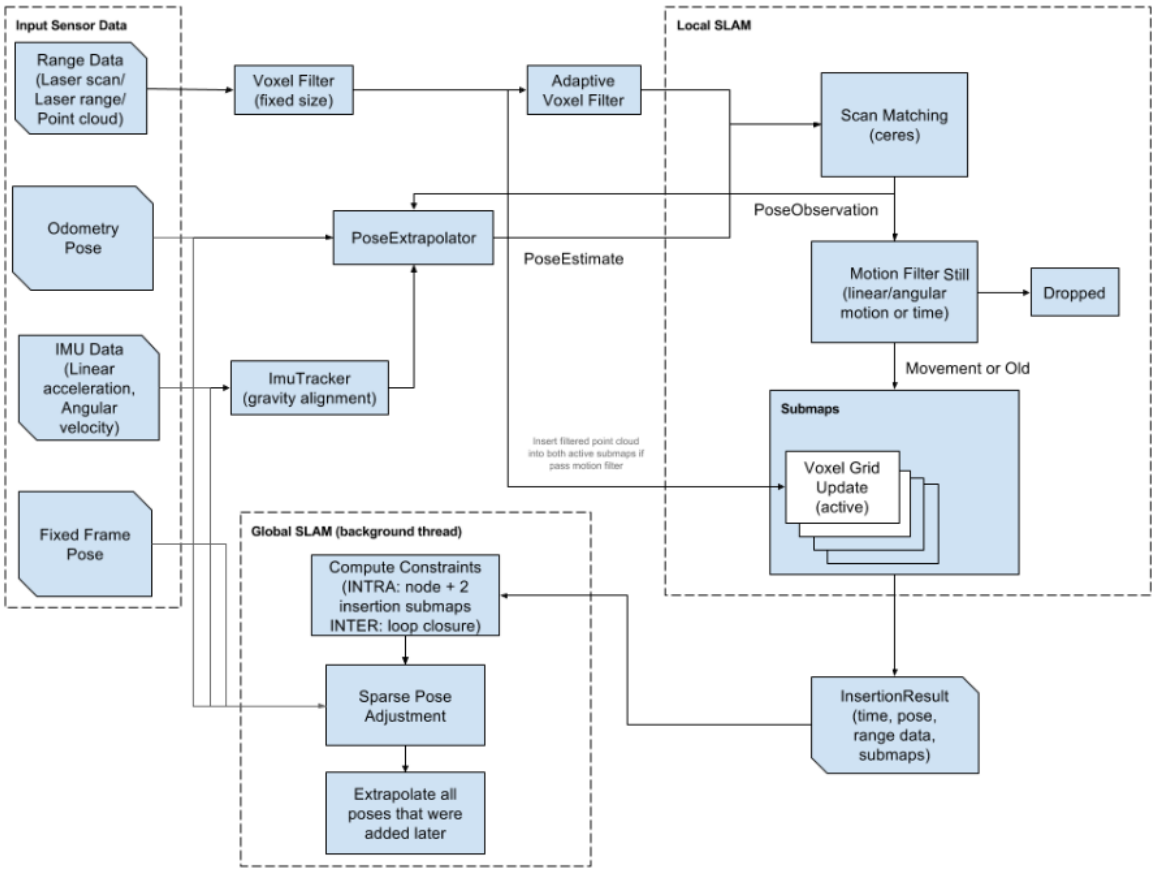
gmapping 可以实时构建室内环境地图，在小场景中计算量少，且地图精度较高，对激光雷达扫描频率要求较低。

3.2.1.5 缺点

随着环境的增大，构建地图所需的内存和计算量就会变得巨大，所以 gmapping 不适合大场景构图。一个直观的感受是，对于 200x200 米的范围，如果栅格分辨率是 5cm，每个栅格占用一个字节内存，那么每个粒子携带的地图都要 16M 的内存，如果是 100 粒子就是 1.6G 内存。

3.2.2 Cartographer算法

Cartographer是谷歌的实时室内建图项目，生成分辨率为 5cm 的 2D 栅格地图。cartographer 方法没有采用粒子滤波器，由全局 SLAM 和局部 SLAM 两部分组成，通过不断进行位姿优化消除累计误差。算法结构图如下



3.2.2.1 局部 SLAM

- 利用里程计（Odometry）和 IMU 数据进行轨迹推算，给出小车位姿估计值（Scan）。
- 每一帧扫描结果都会匹配世界的一小块区域，世界的这些一小块一小块的区域称为 Submap。
- 使用非线性优化将位姿估计值与 Submap 匹配，这一过程会随时间存在累计误差，全局 SLAM 会消除这些误差。

3.2.2.2 全局 SLAM

- 回环检测，首先优化所有 Scan 和 Submap 的位姿，遵循稀疏姿态调整，Scan插入时的相对姿态（相对于 Submap 的）存储在内存中。其次考虑由 Scan 和 Submap 组成的对进行回环检测。
- 通过局部回环检测只消除了局部累积误差，全局误差并没有被消除。利用分支定位和预先计算的栅格，可以加速闭环的形成并在所有的子图完成后进行全局回环检测，这样便消除了全局累积误差。

3.2.2.3 优点

累计误差低，能天然的输出协方差矩阵，后端优化的输入项。成本较低的雷达也能跑出不错的效果。没有imu和odom，只有雷达也可以建图。

3.2.2.4 缺点

内存占用较大，算法体量较大。

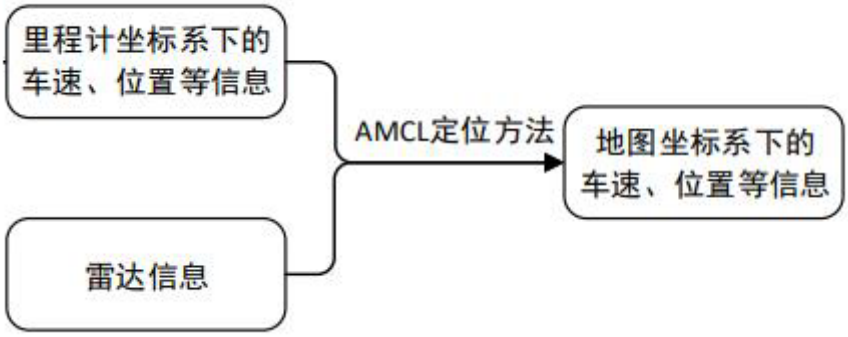
3.3 定位

制图与定位是智能车系统的基础，精确的制图与定位能使智能车系统与其所处环境进行良好的交互。作为智能车技术中的重要组成部分，定位技术也是进行道路导航的前提。

3.3.1 AMCL算法

近年来，世界各国学者一直致力于室内定位方法和精度的研究。剑桥 AT&T 实验室根据 TOA(Time of Arrival)定位法设计出的 Active Bat 系统可以对跟踪标签进行 3D 定位，精度达到 3cm 内[1-2]。韩国 Chakchai 等人利用无线传感网络定位技术和软计算方法 结合，增强了整体系统的实用性[3]。微软研究院的 Bahl 等人设计出的 RADAR 系统可 对室内人员进行定位跟踪[4]。在我国，上海交通大学的钱久超结合惯导技术和地图匹配技术实现室内定位，精度达到 0.8cm[5]。
定位方法按照测量方式分类，可分为基于测距(Rang-based)的和无需测距(Rang-free)的两种。常用的主要有超声波定位、RFID(Radio Frequency Identification)定位、UWB(Ultra Wide-band)定位、蓝牙定位、雷达定位和航位推算定位等。
在机器人定位的算法层面上，基于 ROS 机器人操作系统，AMCL 是该系统中最官方的定位模块，是 move_base 导航模块中唯一指定的定位算法。在 ROS/ROS2 系统中，乃至整个移动机器人领域都是举足轻重的重要地位[6]。AMCL 是 Adaptive Monte Carlo Localization（即自适应蒙特卡洛定位）的简称，是基于多种蒙特卡洛融合算法在 ROS/ROS2 系统中的一种实现。它是移动机器人在二维环境下的概率定位系统，针对已有地图使用粒子滤波器跟踪机器人的姿态，执行贝叶斯滤波器的预测和更新，不受场景的限制，算法简捷快速，同时也可以兼顾算法的精度问题。粒子滤波使用粒子密度（也就是单位区间内的粒子数）表征事件的发生概率。根据选定的评估方程来推算事件的置信程度，并根据该结果重新调整粒子的分布情况。经过若干次迭代之后，粒子就高度分布在可能性高的区域。
传感器模型按照物理意义可分为**波束模型**（Beam model）和**似然域模型**（Likelihood Field）。
波束模型是以传感器为中心向空间发射、通过光或者其它物质的直线传播的时间差来获得距离信息的纯物理模型。给定K个观测，，其在某给定位置的概率表达为：
这里提到的观测 z 实际指的就是传感器获得的数据，是一个包含 K 个值的集合。m 是给定地图信息，表征的是传感器活动的物理世界在数学上的表示，x 是传感器在地图 m 上的位置。因为各个观测 是相互独立的，因此可以拆开为各个分量概率的乘积。基于波束的传感器模型，与几何学和物理学紧密联系的同时，也具有两个主要的缺点：
(1) **波束模型缺乏光滑性**。特别是在又许多小的障碍物的混乱环境中，分布 在 xt 处是不光滑的。不光滑的意思是指，如果机器人在xt处做微小的位移，期望值会有巨大变化，其得到的传感器数据很可能就发生很大的差异。

(2) **波速模型涉及的计算量偏大。**该模型要求对每一个波束都要做射线投射，这需要很大的计算量。如果将机器人工作空间的地图表示离散栅格化，可以将一部分的计算预处理，可以起到一定的运行时间的优化，然而这种优化是以消耗大量的内存作为代价的，因而也是一种不可能的方案。
似然域模型在假定传感器模型的噪声跟误差来源后，对于噪声用以地图上距离该测量终点最近的障碍物为中心，两者之间的欧氏距离为方差的高斯分布来拟合。也就是说，这种模型对概率分布的求解，只关注测量到已知障碍物的末端，而不是关心所有波束的数学跟物理特性。和波束模型相比，考虑了地图的因素，不再是对激光的扫描线物理建模。因此说对于给定地图 m 和机器人方位（也就是传感器方位）的情况下，对观测结果 z 的分布概率为 ,这里的d是测量末端到最近的障碍物的欧氏距离，σ 表示了障碍物的膨胀级别，也反映了算法的收敛效率，可以根据传感器的精度跟地图的精度来设置。这种模型计算不基于传感器物理特性而生成的条件概率，而只是将工作空间的障碍物情况做整体的模糊处理，可以使用各种空间情况，得到的后验概率更光滑，计算更高效，计算量更低。 基于上述推论分析，AMCL 将复杂的数学运算，转换成了计算机更易理解的迭代求解，在机器人定位问题上得到了很好的泛化。



如上图所示，为解决车身定位问题，利用里程计来作为定位的传感器来获取车身在里程计坐标下的位置，使用编码里程计获取里程计信息；其中里程计、雷达作为传感器获取外界信息时，存在一小部分噪声，不影响实验。
为获得车身在地图坐标系下的位置，解决车辆运动过程中在地图上的坐标漂移问题， 通过 AMCL 来校正定位的误差。该定位算法是 move_base 框架中不可替换的模块之一，起到定位和发布 odom 和 map 的 tf 树的作用。在 AMCL 算法中，选择似然域模型，其模型计算高效，使用 2D 的似然域预查表， 跟 ROS/ROS2 的假设完全吻合；光滑连续可求导，即便在障碍物混乱的场景中；可使用诸如爬坡算法、梯度下降算法等做优化，有一定的改进空间；不需要依赖波束的物理特性。
使用 AMCL 包时，最主要的是配置参数文件。图示为部分重要参数。

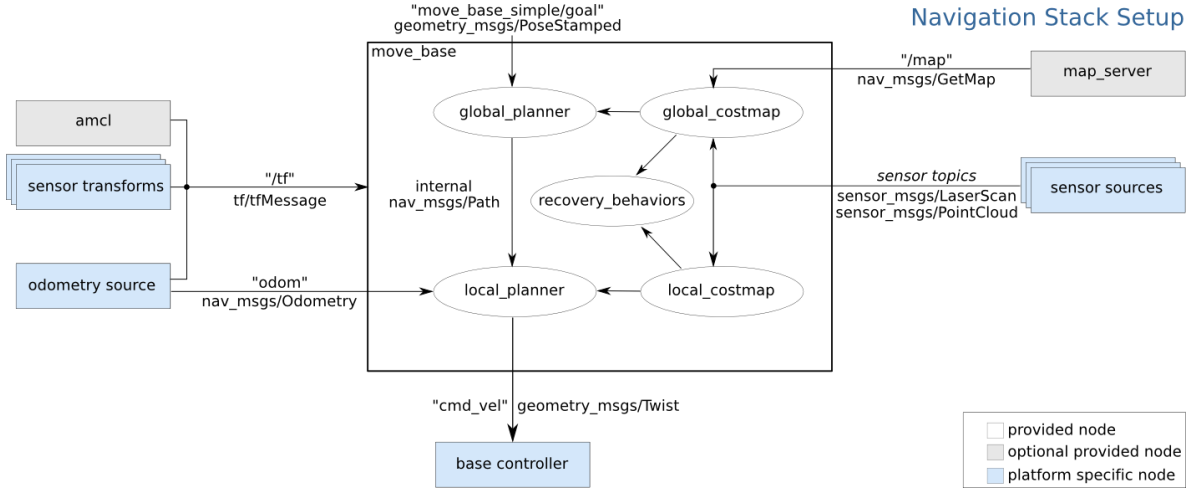
参数	**说明**
laser_max_beams	更新滤波器时，每次扫描中多少个等间距的光束被使用
odom_alpha-4	旋转、平移的误差，默认值为 0.2。odom_alpha 值小意味着更依靠 odometry 定位，大则意味着依靠匹配修正。在走道室 内良好环境和低精度 odometry 下，应该使用更大的 odom_alpha 以得到良好的定位结果。
min_particles	最小粒子数
max_particles	最大粒子数
odom_model_type	用的里程计模型，可以是"diff", "omni", "diff□corrected", "omnicorrected" 。diff 使用的是 sample_motion_model_odometry 算法，使用了 odom_alpha_1 到 4 的参数

3.4 路径规划

完成小车在所绘制地图中的定位后，就可以在地图上设定目标点与障碍物，实现小车在赛道地图中的完全自主导航运行和自主避让障碍物。这需要通过小车的全局路径规划和局部路径规划来完成，全局路径规划是根据地图寻找小车运行的全局最优路径，即最终目标，而局部路径规划是根据设定障碍物实现在尽量保证速度与安全的前提下避障，即短期目标，实现最终目标，需要依次实现一个个短期目标。

3.4.1 move_base 功能包

选择使用 ROS 中的功能包 move_base 来实现导航。move_base 是 ROS 下关于机器人路径规划的中心枢纽，它通过订阅激光雷达、map 地图、amcl 的定位等数据，规划出全局和局部路径，再将路径转化为小车的速度信息，最终实现小车导航，其总体框架如下：



3.4.2 全局路径规划

在 rviz 中设定目标点后，可以根据已有的地图和设定的目标位置来实现全局路径规划，选择分别使用 Dijkstra 与 A* 算法来计算出全局最佳路径。

3.4.2.1 Dijkstra 算法

Dijkstra 算法是典型的求最短路径的算法，是全局路径规划中常用的算法，解决的是有权无向图中从一个顶点到其余各顶点的最短路径问题。
算法原理是从起始点开始，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。

3.4.2.2 A* 算法

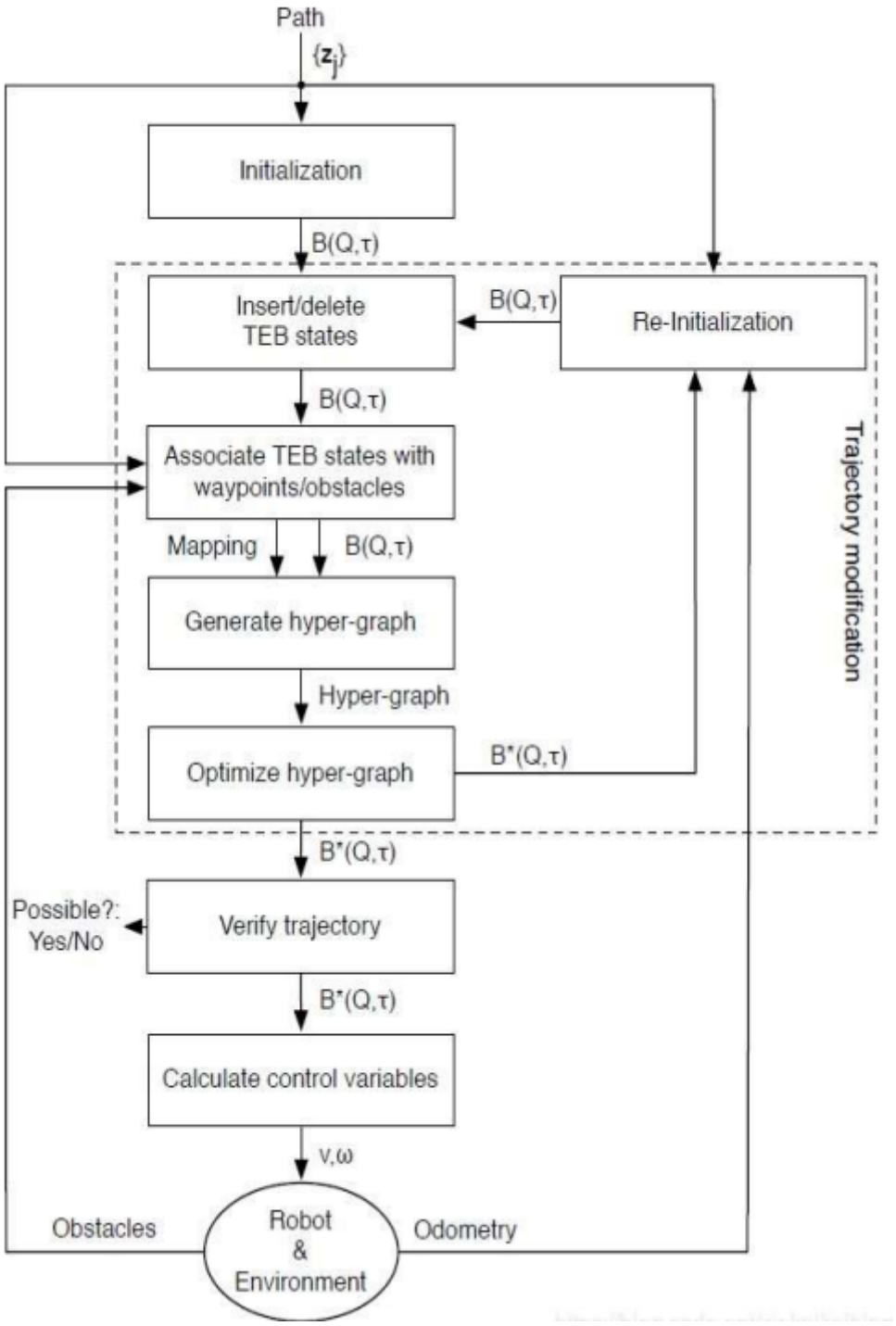
A* 算法是一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。该算法综合了最良优先搜索和Dijkstra算法的优点：在进行启发式搜索提高算法效率的同时，可以保证找到一条最优路径(基于评估函数)。
A* 算法主要使用了如下的估算函数：f(n) = g(n) + h(n)。其中，f(n) 是从初始状态经由状态 n 到目标状态的代价估计；g(n) 表示在状态空间中从初始状态到状态 n 的实际代价；h(n) 表示从状态n到目标状态的最佳路径的估计代价。

3.4.3 局部路径规划

局部路径规划用于动态环境下基于地图信息和障碍物的导航和避障，选择分别使用 Timed-Elastic-Band(TEB) 和 DWA 算法来进行小车的局部路径规划。
局部路径规划的整体思路为：
（1）采样小车当前的状态（dx, dy, dtheta）；
（2）针对每个采样的速度，计算小车以该速度行驶一段时间后的状态，得出一条行驶的路线。
（3）利用一些评价标准为多条路线打分。
（4）根据打分，选择最优路径。
（5）重复上面过程。

3.4.3.1 Timed-Elastic-Band(TEB) 算法

eletic band（橡皮筋）的意思是：将给定的路径视为受内外力影响的弹性橡皮筋，使其变形，而内外力相互平衡，使路径收缩，同时与障碍物保持一定的距离，其中内外力就是对机器人运动的所有约束。而time eletic band 则是在给定路径中间插入N个控制橡皮筋形状的控制点，即机器人姿态。
通过上述定义可以看出，Time-Elastic-Band 算法把路径规划问题描述为一个多目标优化问题，即对最小化轨迹执行时间、与障碍物保持一定距离并遵守运动动力学约束等目标进行优化。因为优化的大多数目标都是局部的，只与机器人的某几个连续的状态有关，所以该优化问题为对稀疏模型的优化。通过求解稀疏模型多目标优化问题，可以有效获得机器人的最佳运动轨迹。算法框架图如下

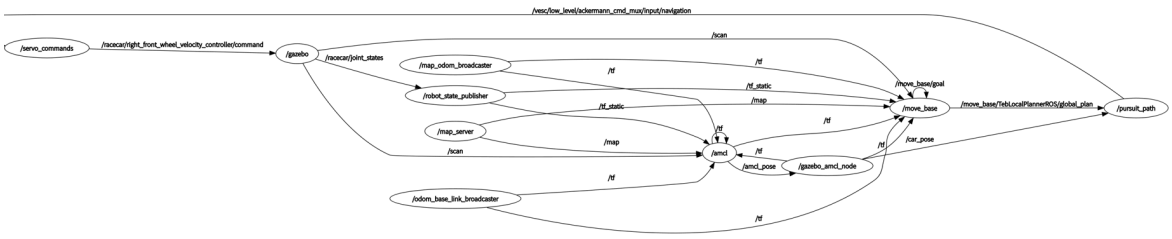


3.4.3.2 DWA 算法

DWA 算法是 ROS 中主要的局部规划算法，有计算复杂度低、可以实时避障等优点。
DWA 算法的原理是在速度 (v,w) 空间中采样多组速度，并模拟机器人在这些速度下一定时间内的轨迹。在得到多组轨迹后，对这些轨迹进行评价。评价函数包括三个方面：与目标的接近程度、机器人前进的速度、与下一个障碍物的距离，通过权衡以上三个部分得到一条最优路径，希望与目标越来越近，且速度较快，与障碍物尽可能远。

4 方案实现

项目节点通信图如下所示：



其中：

- servo_commannds 节点，gazebo 节点为运动控制相关；
- path_pursuit 节点为主要执行节点，负责接收导航消息并发布运动控制消息；
- move_base 为导航消息发布节点；
- amcl 为定位消息发布节点。

4.1 智能车运动控制

4.1.1 概述

基本步骤：
• 小车在接收到导航消息后，通过发布阿克曼消息传递运动控制指令；
• 阿克曼消息会经过 nodelet 转发，最终到达运动控制模块；
• 运动模块通过发布 gazebo 接口定义的话题来最终控制小车运动。
详细描述：
系统接收到 move_base 节点的消息后，将其解析并发布话题为 /vesc/low_level/ackermann_cmd_mux/input/navigation 的运动控制消息。
同时，键盘控制信息也会发布话题为 /vesc/low_level/ackermann_cmd_mux/input/navigation 的运动控制信息。
上述话题被 /vesc/low_level/ackermann_cmd_mux 节点订阅，并统一成 /vesc/low_level/ackermann_cmd_mux/output 话题输出。该话题被 servo_commands 节点订阅，并转化成 /racecar/*_controller/command 的话题输出。*
*节点 gazebo 会订阅 /racecar/*_controller/command 话题，并用于控制小车运动。

4.1.2 配置

Nodelet 的实现目录是 ./src/system/racecar/ackermann_cmd_mux，其中./launch/standalone.launch 描述了 nodelet manager 和 nodelet node 的配置： 启动一个 manager 节点加载入一个 ackermann_cmd_mux 插件节点；./plugins/nodelets.xml 描述了具体的实现类：ackermann_cmd_mux：：AckermannCmdMuxNodelet；./param/example.yaml 描述了被插件类用到的一些配置，其中包括键盘和导航等发布的运动控制 topic。
Nodelet 的配置位于 ./src/system/racecar/launch/mux.launch 中，它两次使用了上述配置，初始化出 high-level-muxer 和 low-level-muxer。本项目主要使用 low-level-muxer。
servo_commands 节点的实现位于 ./src/racecar_control/scripts/servo_commands.py。它接收宏观运动控制消息，并发布各个轮子的运动控制信息。
各个轮子的运动控制配置位于 ./src/racecar_control/config/racecar_control.yaml。这里定义了每个轮子的消息类型，以及对应的 PID 控制信息。小车和 gazebo 模拟器的接口位于 ./src/racecar_control/launch/racecar_control.launch 中：它启动 controller_manager 节点并通过参数定义了各个轮子的控制器。gazebo 插件的配置和 transmission 配置不加赘述。

4.2 建图

根据技术方案部分我们选择使用GMapping算法建图，ROS已经包含了开源的Gmapping功能包， 通过直接调用gmapping功能包即可实现建图。使用gmapping构建地图需要事先对各个传感器的相对位置坐标进行tf变换， 将激光发布frame跟baselink和odom连接起来，通过在launch文件中修改部分参数实现。仿真过程中，地图启动后，使用

```
roslaunch racecar_gazebo slam_gmapping.launch
```

命令启动slam和rviz，控制车辆跑完全图后地图即被构建好，使用

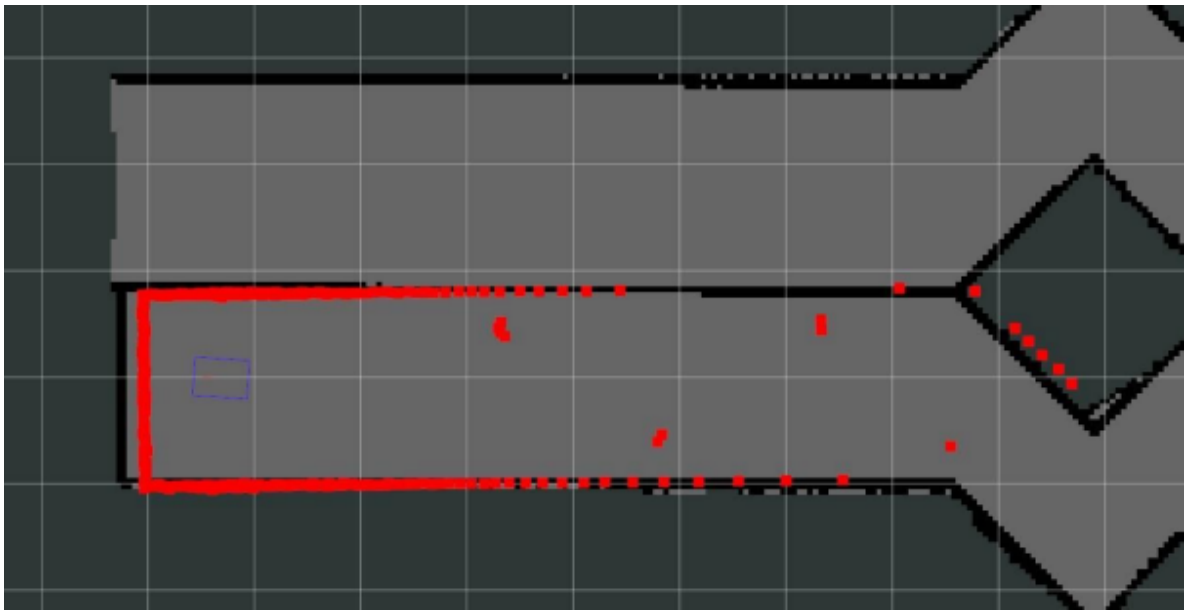
```
rosrun map_server map_saver -f ~/racecar_ws/src/racecar_gazebo/map/map_runway
```

命令保存地图，关闭建图窗口。

4.3 定位

4.3.1 雷达数据

当车辆启动后，激光雷达开始工作，其在仿真地图中获取到的雷达信息直接传输到上位机中，将 scan 格式的数据通过添加操作呈现在 Rviz 可视化软件中。在车辆运动过程中，通过观察雷达信息与小车所处局部范围内的赛道边缘信息或桶锥等障碍物信息的重合程度来判断雷达信息获取是否正常，准确程度是否足够。



4.3.2 tf 坐标变换

在 ROS 平台中，不同坐标系之间的平移与旋转变换关系即 tf 转换信息的重要性是毋庸置疑的。在调试车辆定位和控制等算法时，也必不可少地需要检查 base_link、odme 和 map 等坐标系之间的 tf 关系，包括父子坐标系 id、时间戳、表示旋转关系的四元数 和坐标系平移量。通过 view_frames 监听当前时刻所有通过 ROS 广播的 tf 坐标系，并绘制出树状图表示坐标系之间的连接关系保存到离线文件中，也可用 rqt_tf_tree 实时刷 新显示坐标系关系，或者用 tf_echo 工具直接查看两个参开坐标系之间的转换关系。

添加 amcl.launch 文件内容如下所示：

```
<launch>
<!-- amcl -->
  <arg name="use_map_topic"  default="true"/>
  <arg name="scan_topic"     default="scan"/>
  <arg name="initial_pose_x" default="-5.388334"/>
  <arg name="initial_pose_y" default="-4.094883"/>
  <arg name="initial_pose_a" default="0.0"/>

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic"             value="$(arg use_map_topic)"/>
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha5"               value="0.1"/>
    <!-- Publish scans from best pose at a max of 4 Hz -->
    <param name="gui_publish_rate"          value="4.0"/>
    <param name="laser_max_beams"           value="60"/>
    <param name="laser_max_range"           value="12.0"/>
    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="2000"/>
    <param name="kld_err"                   value="0.05"/>
    <param name="kld_z"                     value="0.99"/>
    <param name="odom_alpha1"               value="0.2"/>
    <param name="odom_alpha2"               value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3"               value="0.2"/>
    <param name="odom_alpha4"               value="0.2"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_model_type"          value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"              value="0.25"/>
    <param name="update_min_a"              value="0.2"/>
    <param name="odom_frame_id"             value="odom"/>
    <param name="base_frame_id"             value="base_link"/>
    <param name="global_frame_id"           value="map"/>
    <param name="resample_interval"         value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance"       value="1.0"/>
    <param name="recovery_alpha_slow"       value="0.0"/>
    <param name="recovery_alpha_fast"       value="0.0"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <remap from="scan"                      to="$(arg scan_topic)"/>
```

设置 odom 与 map 之间的静态坐标变换、base_link 与 odom 之间的静态坐标变换：

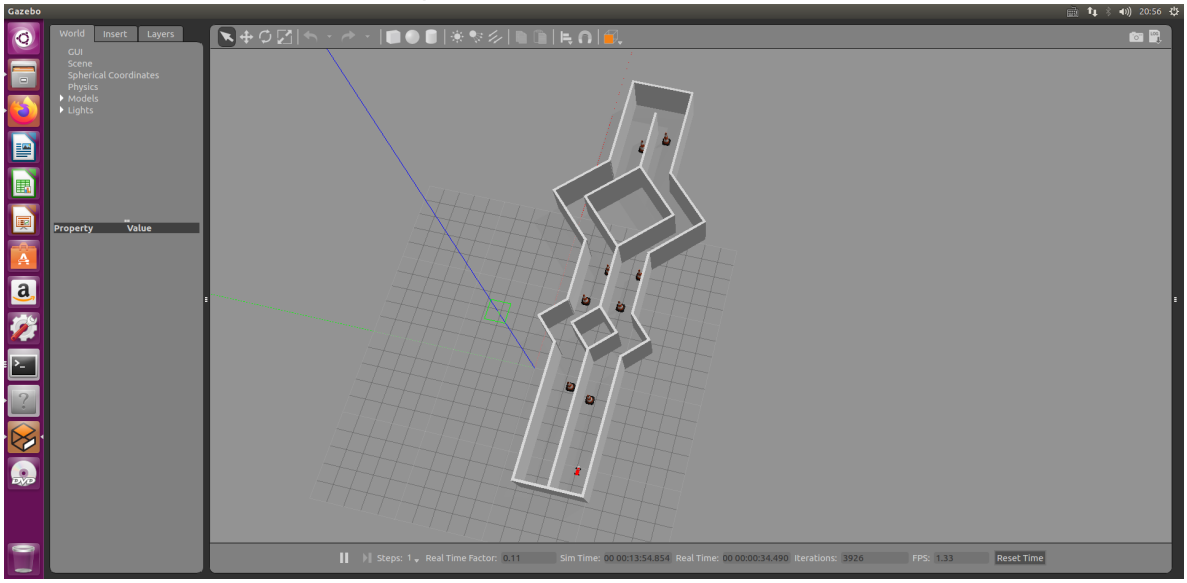
```
  <!-- 设置一个/odom与/map之间的静态坐标变换 -->
  <node pkg="tf" type="static_transform_publisher" name="map_odom_broadcaster" args="-5.388334 -4.094883 0 0 0 0 /map /odom 100" />

  <!-- 设置一个/base_link与/odom之间的静态坐标变换 -->
  <node pkg="tf" type="static_transform_publisher" name="odom_base_link_broadcaster" args="0 0 0 0 0 0 /odom /base_link 100" />

</launch>
```

4.4 路径规划

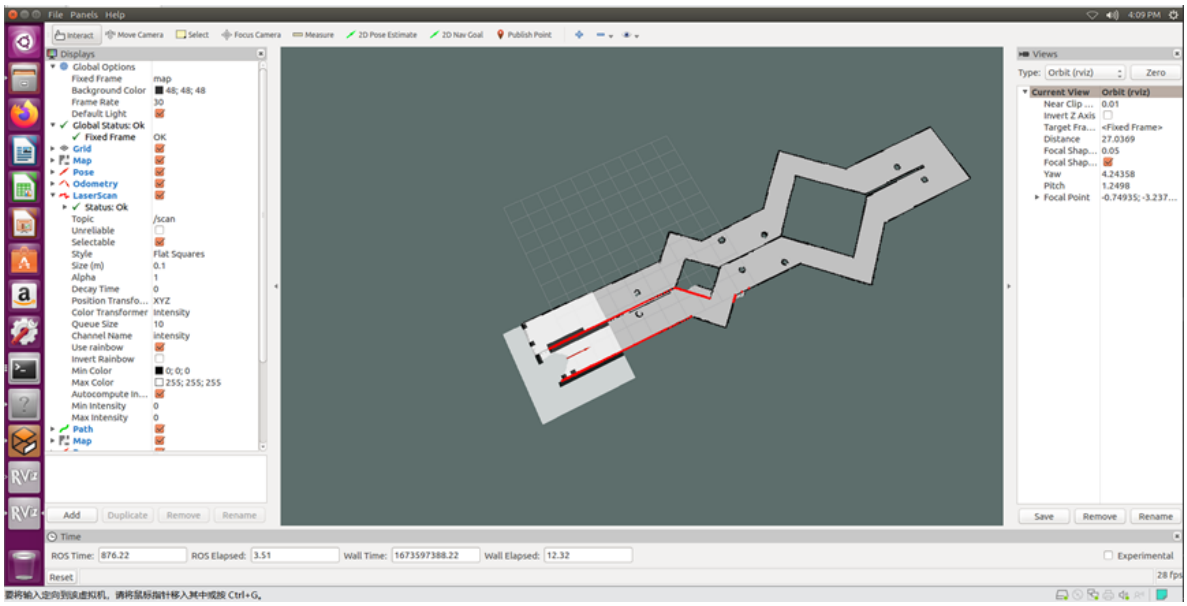
路径规划分为全局和局部路径规划，全局路径规划是为了寻找小车运行的全局最优路径，而局部路径规划是为了小车在运行过程中更符合约束以及控制小车速度等参数，使得导航更具备实时性，二者的结合能够使得导航具备更好的可用性。

4.4.1 程序运行

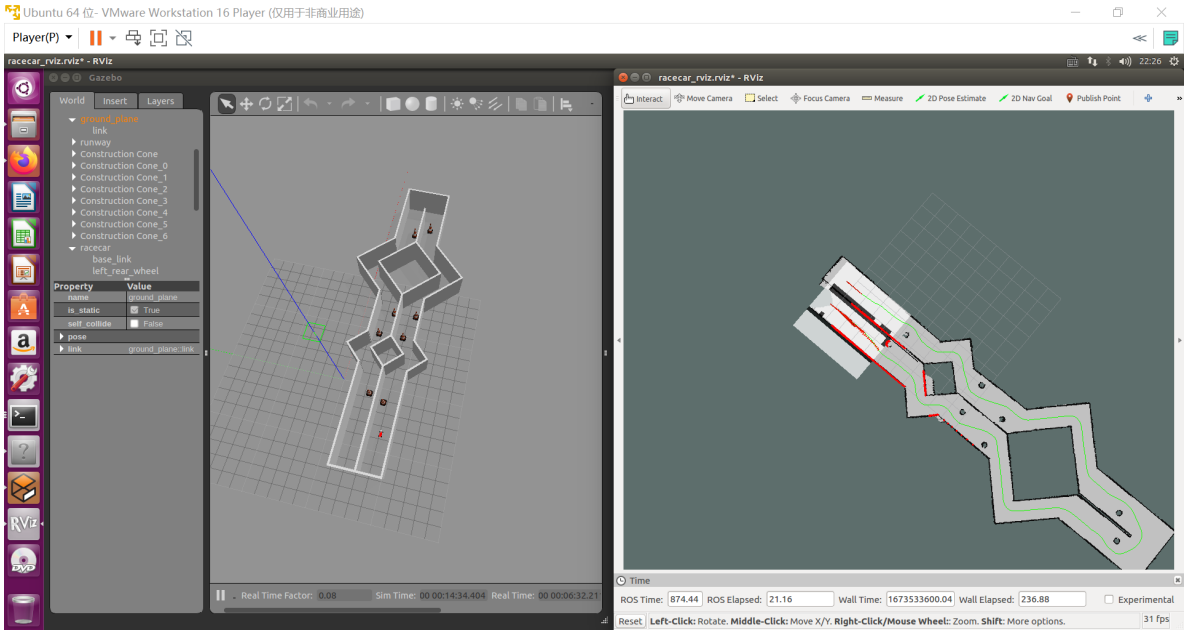
首先，根据提供的 launch 文件启动 gazebo，加载赛道和小车模型。



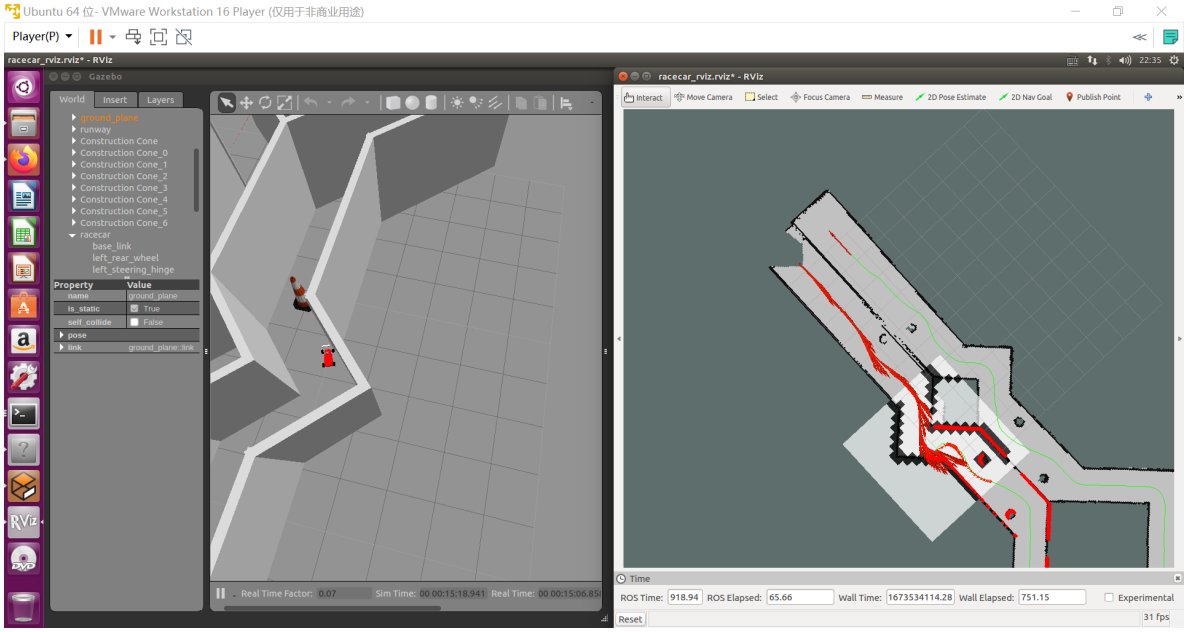
然后，启动 rviz.launch 文件将已经建立好的全局代价地图导入到 rviz 平台中，启动后便可在 rviz 平台中看到小车模型和全局地图。



接着，通过 rviz 功能栏中的 2D Nav Goal 按钮在地图中设置小车目的地，再在命令行执行小车自主导航脚本 path_pursuit.py，开启小车自主导航。



通过多次尝试可以观察到，小车存在行驶速度缓慢与一定概率的撞墙、翻车等问题。



4.4.2 导航算法选择及参数优化

由于使用原提供的导航算法时，小车自主导航完全通过赛道过程耗时较长，且存在一定概率的翻车情况，所以我们尝试调整使用的全局和局部路径规划算法，并且对算法中和配置文件中部分与小车速度相关的参数进行调整，使得最终小车完全通过赛道的运行速度得到提高。

首先，完全阅读并理解 path_pursuit.py 脚本后，发现其中的小车最大速度 MAX_VELOCITY 和小车最大转角 max_angle 对小车运行过程中的速度和碰到障碍物和转弯处时的反应有着重要影响，并且小车在运行过程中调用 speed_control 方法进行动态速度调整，经过多次调整与模拟验证后，发现将小车最大速度 MAX_VELOCITY 和小车最大转角 max_angle 更改为 0.7 和 1.0 时，小车在赛道中的运行情况最佳。

小车进行路径规划时需要使用代价地图中存储的有关障碍物的信息，其中代价地图的膨胀半径 inflation_radius 为在代价地图中与障碍物保持安全的最大距离，影响着小车过狭窄地域的效果，inflation_radius 越大则小车越难通过狭窄地区，越小则越容易通过，经过多次调整与模拟验证后发现 inflation_layer 中的 inflation_radius 为 2.5 时小车避免障碍物效果最佳。

在当前场景中，全局和局部路径规划常用的算法分别是 Dijkstra 和 A* 算法、DWA 和 Teb 算法，我们对以上算法分别进行了尝试，经过调整与验证后对其配置参数进行了部分更改。在 racecar_runway_navigation.launch 文件中配置了小车导航使用的路径规划算法及其他参数，其中局部路径算法配置参数中 controller_frequency 值为 3.0 时较默认的 5.0 能取得更好效果。

4.4.3 各算法对比

可以通过对 racecar_gazebo/launch/racecar_runway_navigation.launch 文件中的配置参数进行修改更改使用的全局和局部路径规划算法。

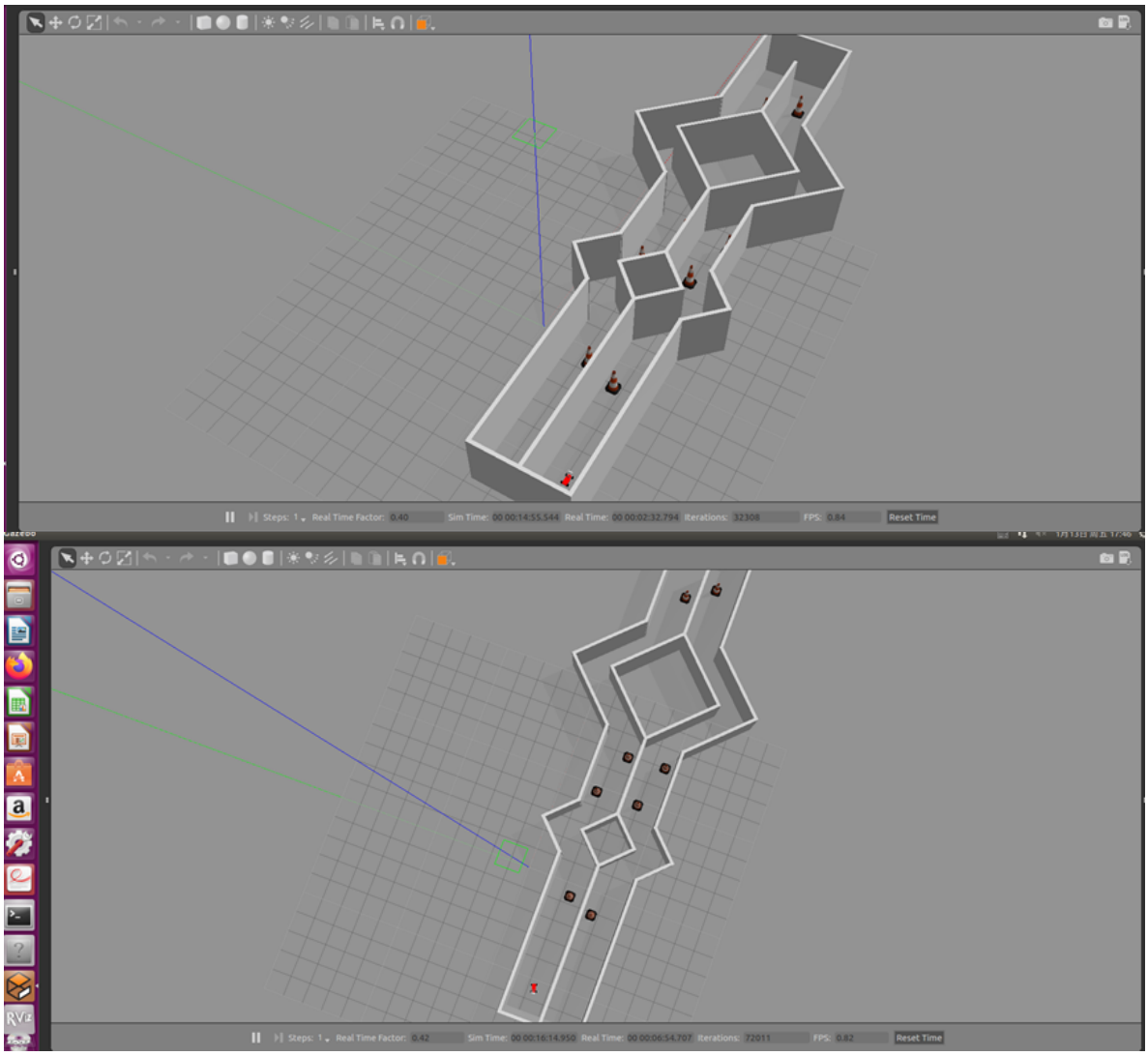
全局路径规划算法 A* & Dijkstra：通过默认参数 base_global_planner 引入 move_base 中的算法包，在配置文件中添加，修改参数的 value 值即可设置不同算法。

```
<param name="base_global_planner" value="global_planner/GlobalPlanner" />
<param name="planner_frequency" value="0.01" />
<param name="planner_patience" value="5.0" />
<param name="use_dijkstra" value="false" />
```

局部路径规划算法 TEB & DWA：在 racecar_gazebo/config 中创建并编写 teb_local_planner_params.yaml 与 dwa_local_planner_params.yaml，修改 navigation 中的参数，并且配置 base_local_planner，在 path_pursuit.py 中更改加载参数。

```
<param name="/use_sim_time" value="true" />
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
  <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
  <rosparam file="$(find racecar_gazebo)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
  <rosparam file="$(find racecar_gazebo)/config/local_costmap_params.yaml" command="load" />
  <rosparam file="$(find racecar_gazebo)/config/global_costmap_params.yaml" command="load" />
  <rosparam file="$(find racecar_gazebo)/config/teb_local_planner_params.yaml" command="load" />
  <!--rosparam file="$(find racecar_gazebo)/config/dwa_local_planner_params.yaml" command="load" /-->

  <param name="base_global_planner" value="global_planner/GlobalPlanner" />
  <param name="planner_frequency" value="0.01" />
  <param name="planner_patience" value="5.0" />
  <!--param name="use_dijkstra" value="false" /-->

  <!--param name="base_local_planner" value="dwa_local_planner/DWALocalPlannerROS" /-->
  <param name="base_local_planner" value="teb_local_planner/TebLocalPlannerROS" />
  <param name="controller_frequency" value="5.0" />
  <param name="controller_patience" value="15.0" />

  <param name="clearing_rotation_allowed" value="false" />
</node>
```

经过多次模拟验证，得出使用各算法组合时小车的平均运行时间。下图为 Dijkstra + TEB 算法组合的运行情况，可以看出，小车在 1s 内前进了一定量的距离：



1. A* + TEB：小车从起始点开始运行到达目的地，平均 sim time 为 1min15s，平均每次行驶碰撞墙壁2次，偶尔会有翻车情况发生；
2. A* + DWA：小车从起始点开始运行到达目的地，平均 sim time 为 1min08s，平均每次行驶碰撞墙壁1次，存在碰撞墙壁或锥桶后小车停止前进的情况；
3. Dijkstra + TEB：小车从起始点开始运行到达目的地，平均 sim time 为 1min05s，几乎不存在碰撞情况；
4. Dijkstra + DWA：小车从起始点开始运行到达目的地，平均 sim time 为 1min12s，平均每次行驶碰撞墙壁 1 次，有时会发生翻车；

所以经过测试比较后，发现 Dijkstra 和 TEB 算法的组合相对最适合小车在该赛道上的运行，所以最终选择了使用 Dijkstra 算法和 TEB 算法作为小车的全局和局部路径规划算法。

5 测试分析

5.1 数据来源

赛道模型数据和小车模型数据均统一提供，赛道模型已填加锥桶障碍物，且相关参数已配置完全，均不允许修改。

5.2 环境配置

此次实验的虚拟机环境为 ubuntu16.04，在此基础上安装了 ROS Kinetic， 采用 Desktop-Full Install 方法一同安装了 gazebo7.16、rviz、rqt 等 packsges。在基本的环境配置完成后， 又安装了程序运行所需要依赖的包：
(1) ros-kinetic-driver-base
(2) ros-kinetic-ackermann-msgs
(3) ros-kinetic-controller-manager
(4) ros-kinetic-gazebo-ros-control
(5) ros-kinetic-effort-controllers
(6) ros-kinetic-joint-state-controller
(7) ros-kinetic-rtabmap-ros
(8) ros-kinetic-move-base
(9) ros-kinetic-map-server
(10) ros-kinetic-teb-local-planner
(11) ros-kinetic-global-planner

5.3 测试过程

我们在提供的已有程序框架基础上，对建图算法、全局规划算法、局部路径规划算法进行了测试并作出选择。基于整体最优和源程序框架基础上，通过分别对各算法组合进行调参及模拟验证，根据各算法组合下小车运行效果，我们选择了 GMapping 算法建图、Dijkstra 算法做全局规划、TEB 做局部路径规划、原有的 path_pursuit 算法控制小车导航和 PID 来纵向控制。在确定好程序所使用的算法后，我们对各种算法的参数进行了进一步深化的调整，重点是 PID 参数、TEB 参数和小车速度参数的调整。

5.4 分析与结论

在调整参数的时候，我们先调整了 PID，将控制做到所能做到的最好程度，然后再调整 TEB 参数， 由于 TEB 是软约束，所以测试效果具有随机性，为了保证调整后的程序具有较强的鲁棒性，我们每套参数都测试 3 次及以上。由于路径规划较为靠近锥桶，导致小车经过障碍物的时候容易擦边，为此我们特意调大了最小障碍物距离以及膨胀距离等参数，为了保证较好的避障性能，我们将速度设定的不是特别高，因为碰撞一次对速度的影响很大，调大速度得不偿失。最后通过多次模拟，在提供的赛道模型中，小车完整跑完整个过程的模拟时间 sim time 为 1min19s，没有碰撞锥桶及墙壁。

6 作品总结

6.1 总述

本作品的问题域就像是自动驾驶问题域的精简版，要求智能车在指定的赛道中进行建图、定位和导航，在导航的过程中对障碍物进行自主规避。我们以技术方案为理论支撑，各模块及系统框架按技术方案搭建完成后，实际运行效果并不如理论中理想，经过对各模块对比测试及大量关键参数多轮次调整后达到最佳的状态。

6.2 技术路线

运动控制方面，本方案通过在传统 PID 算法基础上进行参数调整，最终使得模型车辆在弯道处的响应速度更加快速，并在曲率较小的地方具有更强的稳定性，达到了较好的轨迹跟随控制效果。
建图方面，本方案采取的基于RBPF粒子滤波算法、改进提议分布和选择性重采样的GMapping算法计算量小且精度较高，能够满足在所处环境较为狭长的情况下快速准确构建栅格地图的任务要求，并利于后续实现精准定位和路径规划。
定位方面，本方案使用激光雷达和里程计获取周围环境，从而得到车身姿态信息，使用自适应蒙特卡罗算法分析这些数据得到定位信息，能够周期性的矫正里程计定位随时间增 长的累积误差，进而快速准确地实现模型车辆的全局定位。
全局路径规划方面，本方案采用的 Dijkstra 路径规划算法，每次遍历到始点距离最近且未访问过的顶点的邻接节点，保证能够找到最短路径，并通过修改代价地图参数后，可使规划出的路径符合阿克曼角模型车辆的运动要求，充分满足了自主导航对于上层规划决策器的功能需要。
局部路径规划方面，本方案采用的TEB算法，包含了最小转向半径、障碍物缓冲区、最优时间权重等参数，能够更好地被阿克曼模型利用，更好地满足了灵活避障的功能需求。

6.3 工作量

我们的工作量主要集中在理论分析和实践测试部分。理论分析部分，我们在建图、定位和导航各模块都选取了多种不同算法进行比较，初步选取后我们也根据总体效果进行了二次删选，最终达到最佳整体效果。实践测试部分，我们通过对各模块采用算法实现的源码分析，选取确定多个关键参数，通过控制变量的方式进行调参，对每次调参后项目进行多轮运行，获得充分测试数据，通过分析逐步确定各参数的较优范围，最终经过综合考量确定整体效果最好、鲁棒性最优的参数值并应用。

6.4 数据和测试效果

考虑规划过程中的随机性，为保证鲁棒性，通过对各模块及算法大量关键参数的对比测试和调整，我们的小车在自主导航和避障上达到了较好效果，但由于实践测试部分均在给定赛道完成，所以当前算法组合及参数对于其他赛道未必也是最优的，但在当前赛道都可以保证正常驾驶全程并轻碰障碍及墙壁一次以内。

7 参考文献

[1] Kim S Y, Choi J Y, Lee M H. Indoor Positioning System Using Incident Angle Detection of Infrared Sensor[J].Journal of Institute of Control, 2010, 16(10): 991-996.
[2] Ward A, Jones A, Hopper A. A New Location Technique for the Active Office[J]. Personal Communication IEEE, 1997, 4(05): 42-47.
[3] So-In C, Permpol S, Rujirakul K. Soft Computing-based Localization in Wireless Sensor Networks[J]. Pervasive & Mobile Computing, 2015,29:17-37.
[4] Bahl P, Padmanabhan V N. RADAR: An In-Building RF-based User Location and Tracking System[C].Tel Aviv: IEEE INFOCOM, 2000,775-784. [11] Qian J, Pei L, Ma J, et al. Vector Graph Assisted Pedestrian Dead Reckoning Using an Unconstrained Smart-phone[J]. Sensors, 2015, 15(03):5032-5057.
[5] 王振. 用于自动驾驶系统的障碍物检测技术研究[D].中国科学院大学(中国科学院 西安光学精密机械研究所),2019.
[6] 小明工坊. ROS 实验 | PID 控制做了什么？.古月居.
[7]Grisetti G , Stachniss C , Burgard W . Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters[J]. IEEE Transactions on Robotics, 2007, 23(1):34-46.
[8]Kohlbrecher S , Stryk O V , Meyer J , et al. A flexible and scalable SLAM system with full 3D motion estimation[C]. IEEE International Symposium on Safety. IEEE, 2011.
[9]Hess W , Kohler D , Rapp H , et al. Real-time loop closure in 2D LIDAR SLAM[C]. IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016.
[10]Dieter Fox. Adapting the Sample Size in Particle Filters Through KLD-Sampling. 2003, 22(12):985-1003.

[11]Constantino Tsallis. Retirement of Prof. H. Eugene Stanley as Main Editor of Physica A/Elsevier[J]. Physica A: Statistical Mechanics and its Applications,2020,556.
[12]Ying Feng,Min Wu,Xin Chen,Luefeng Chen,Sheng Du. A fuzzy PID controller with non□linear compensation term for mold level of continuous casting process[J]. Information Sciences,2020
[13]ROS导航包参数设置 - 古月居