系统基本设计方案

1. 项目简介

1.1 应用场景

- **场景概述**

  我们设计和实现的移动机器人是用于在KTV或者其他类似文娱休闲场景（如博物馆、展馆等）中为客人提供导航服务的。

- **目标用户**

  该机器人的目标用户是前来KTV、博物馆等场景消费、参观的客人。这些客人可能会因为不熟悉内部环境而迷路，因此有导航的需求。

- **应用场景**

  该机器人可以在KTV大厅、博物馆公共区域巡逻或待命，在客人有需要时，根据客人提供的目标地点提供导航服务。具体使用场景包括：

  - KTV：客人在到达KTV、确定好包厢之后，机器人会带领客人前往对应的包厢。在KTV娱乐的过程中，客人如果需要去洗手间或者前台等场所，可和机器人进行交互，告知机器人目标地点，机器人就会规划出路线，带领客人前往目的地。
  - 博物馆：机器人可以作为自助导览员，为游客提供导览服务。机器人可以利用内置的地图和导航系统，带领游客沿着特定的游览路线进行游览，同时可以配备显示屏播放展品的历史背景等信息。

1.2 完成工作概览

No.	Task	Details
#1	搭建基本开发环境	• 在 VMware 中安装 Ubuntu 18.04 虚拟机 • 在 Ubuntu 18.04 中安装 & 配置 ROS • 安装 & 配置 gazebo + 连接 ROS • 安装 & 配置 Rviz
#2	机器人选型	• 导入机器人模型 • 配置传感器（包括摄像头、激光雷达、IMU传感器） • 机器人相关代码放置在 `neor_mini` 功能包中（ `urdf` 文
		• `neor_mini` 功能包中的 `worlds` 文件夹

#3	在 gazebo 中导入 & 修改场景	
#4	实现建图功能	• 使用 gmapping 算法建图 • 建图完成后的 `.pgm` 文件放在 `mini_gmapping` 功能包中夹下
#5	实现导航 & 避障功能	• `mini_navigation` 功能包
#6	实现方向盘功能	• `steer_drive_ros` 功能包
#7	将机器人放置在场景中工作运行	\

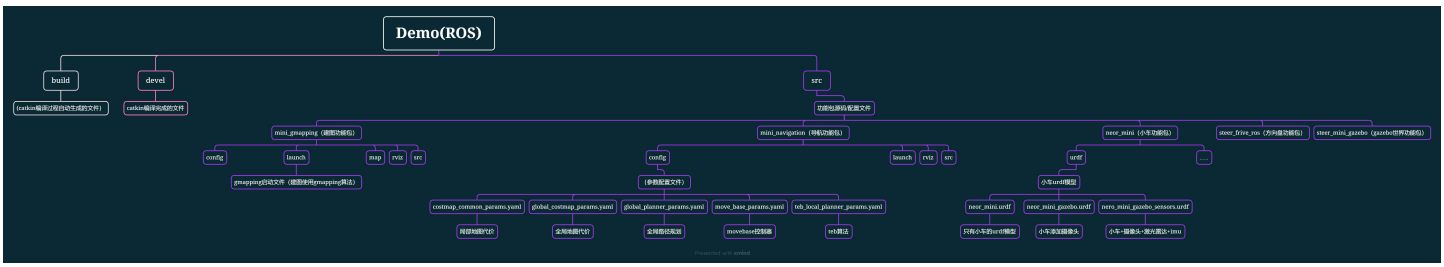
1.3 最终技术方案

总体来说，我们的实现方案使用了：Ubuntu 18.04 + ROS + gazebo 仿真 + Rviz 可视化传感器数据。

- **Ubuntu 18.04**：我们小组使用了 VMware 配置 Ubuntu 18.04 虚拟机作为基本的开发环境；
- **ROS**：ROS（Robot Operating System）是一个开源的机器人操作系统框架，旨在帮助开发者构建灵活、可扩展的机器人应用软件。ROS 提供了一系列工具、库和协议，用于处理硬件抽象、设备驱动、消息传递、包管理、可视化等方面的任务。我们在 Ubuntu 18.04 中：配置 ROS 环境 ----> 创建工作环境 -----> 编写不同功能包，实现机器人的各个功能；
- **gazebo**：gazebo是一个用于仿真机器人系统的开源三维动力学仿真环境。它提供了一个功能强大的虚拟环境，用于模拟机器人、传感器、物理环境和交互行为。我们小组使用gazobo搭建我们的场景模型；
- **Rviz**：Rviz（ROS visualization）是 ROS（Robot Operating System）中的一个强大的可视化工具，用于可视化机器人的感知、状态和行为。它提供了丰富的可视化功能，使开发者能够直观地观察和调试机器人系统的各个方面。我们使用RVIZ可视化传感器数据。
- **硬件选型**：导入NEOR mini无人车的urdf模型文件，传感器选型如下：激光雷达为 RPLIDAR A1 M8，IMU为 LPMS_CURS2 9-axis。

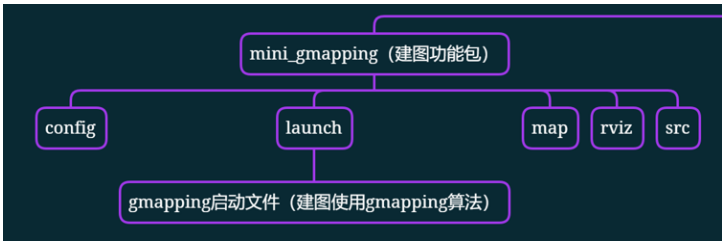
2. 项目架构

整个项目目录结构如下图所示：

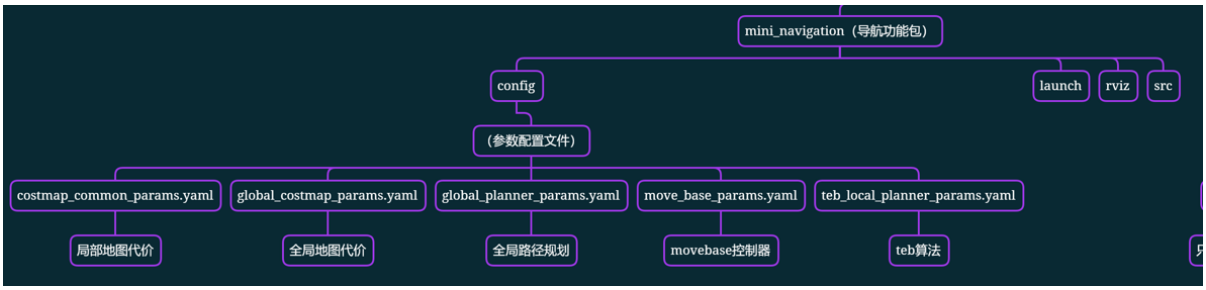


Robot Architecture

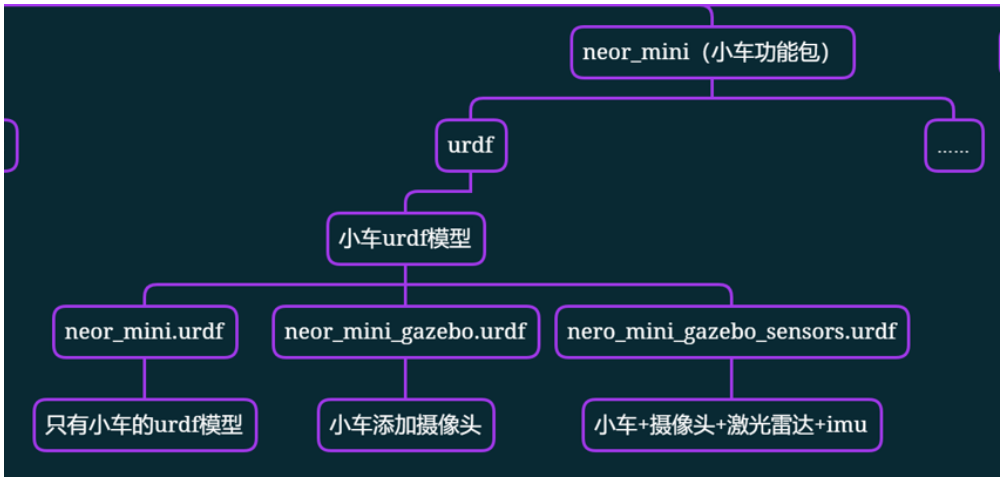
- 建图功能包



- 导航功能包



- 小车功能包



- 方向盘功能包
- gazebo世界功能包

3. 核心功能实现

3.1 建图功能实现

在本项目中，ROS 中构建地图的选用算法是gmapping，该算法会订阅一些话题数据，然后发布栅格地图话题。

```
subscribed Topics:
  tf(tf/tfMessage):
    static tf: From laser to base_link
    dynamic tf: From odometry to base_link
  scan(sensor_msgs/LaserScan): From laser  device
Published Topics:
```

```
  map_metadata (nav_msgs/MapMetaData)  Get the map data from this topic, which i
  map (nav_msgs/OccupancyGrid)  Get the map data from this topic, which is latch
  ~entropy (std_msgs/Float64)  Estimate of the entropy of the distribution over
```

运行 gampping 算法。该算法有很多的参数可以设置，同样以 launch 文件的形式加载。这里创建了一个 ROS 功能包，来存储启动的 launch 文件和记录编译时所需要的依赖包。文件树如下：

```
.
├── CMakeLists.txt
├── config
│   └── dual_ekf_navsat_mini.yaml       # ekf params file
├── include
│   └── mini_gmapping
├── launch
│   ├──  dual_ekf_navsat_mini.launch     #  ekf node params
│   ├── gmapping.launch            # gmapping node params
│   └── gmapping_steer_mini_sensors.launch   # total launch file
├── map
│   ├── cooneo_office_map.pgm         # saved map file
│   └── cooneo_office_map.yaml
├── package.xml
├── rviz
│   └── gmapping_rviz.rviz              # saved rviz file
└── src
     └── mini_gmapping_node.cpp          # node code file
```

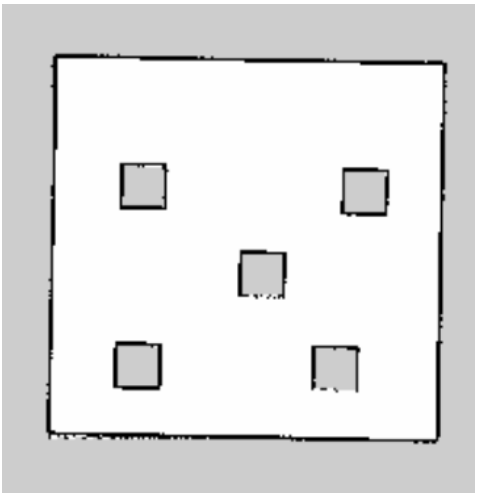
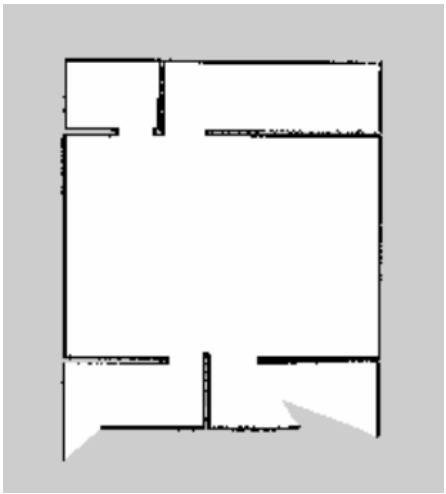
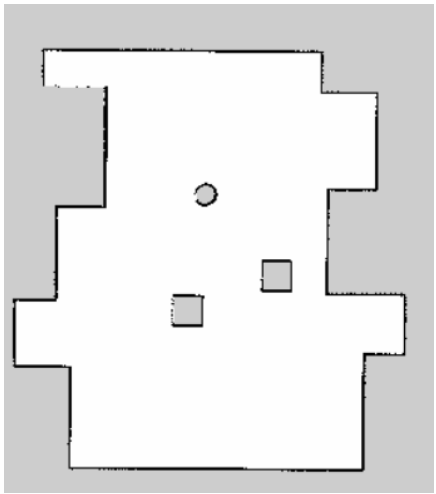
这里重点介绍 gmapping_steer_mini_sensors.launch 文件

```
<launch>

    <!--  launch steer_mini_gazebo model node -->
    <include file="$(find
steer_mini_gazebo)/mini_gazebo/launch/steer_mini_sim_sensors.launch" />

    <!-- load robot_localication node Integrate imu and odometer information,
output the fused odom information and tf conversion from base_link to odom for
Gmapping mapping -->
    <include file="$(find mini_gmapping)/launch/dual_ekf_navsat_mini.launch" />

    <!--  launch gmapping node -->
    <include file="$(find mini_gmapping)/launch/gmapping.launch" />

    <!-- Load model into Rviz and Load Rviz configuration file into rviz-->
```

```
13      <param name="robot_description" textfile="$(find
    neor_mini)/urdf/neor_mini_gazebo_sensors.urdf" />
14      <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
    mini_gmapping)/rviz/gmapping_rviz.rviz" />
15
16  </launch>
```

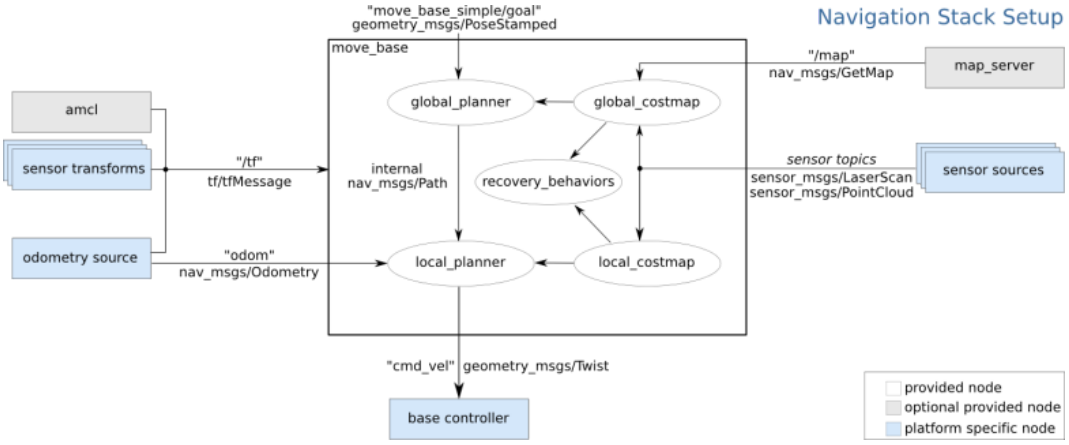
- 第一个加载的文件是配置好的带传感器的 mini 仿真模拟文件。该文件启动后将发布三个话题，分别是里程计话题 /ackermann_steering_controller/odom、机器人姿态话题 /imu 和激光雷达话题 /scan。
- 第二个加载的 .launch 文件的功能是将里程计和姿态话题数据融合，使用的是 robot_localization 功能包，其输出新的 base_link 到 odom 的一个 tf 转换，该 tf 转换被 gmapping 订阅。
- 第三个文件启动的是 gmapping，并加载其参数。
- 第四部分是启动 Rviz，目的是为了可视化各个传感器的信息，以及构建的地图。

建图完毕后，使用 map_server 功能包保存地图，得到文件 map_name.pgm 和 map_name.yaml，都保存在 /mini_gmapping/map 目录下。以下是我们项目中用到的三个不同场景建图后的.pgm文件。

3.2 导航 & 避障功能实现

在本项目中，我们选择使用 ROS 的导航功能包集 navigation 中所提供的 move_base 功能包来实现此功能，即使用 move_base 功能包，通过订阅激光雷达、map 地图、amcl 定位等数据规划出全局和局部路径，再将路径转化为机器人的速度信息，最终实现机器人导航。



move_base功能包框架

在实现过程中，我们通过鼠标点击目标点来模拟在 KTV、博物馆等场景中机器人被告知要去往的目的地，然后机器人通过路径规划来确定最优路径，以一定的速度导航去往目的地，以此来实现对实际应用场景的模拟。

3.2.1 move_base功能包

move_base 主要由全局路径规划和局部路径规划组成。

1. 全局路径规划（global planner）

   根据给定的目标点和全局地图实现总体的路径规划，计算出机器人从当前位置到目标位置的最优路线作为全局路线。move_base 功能包的 global_planner 提供了使用 Dijkstra 算法和 A* 算法对于全局路径规划的实现。

2. 局部路径规划（local planner）

   在实际的导航过程中，机器人可能无法完全按照给定的全局最优路线运行，因为在机器人运动过程中随时都可能会出现一定的障碍物。而局部规划的作用就是通过一定的算法来实现对障碍物的规避，并选取当前的最优路径以尽可能符合全局最优路径。move_base 功能包中提供了 Trajectory Rollout 算法、DWA 算法、TEB 算法等局部路径规划算法的实现。

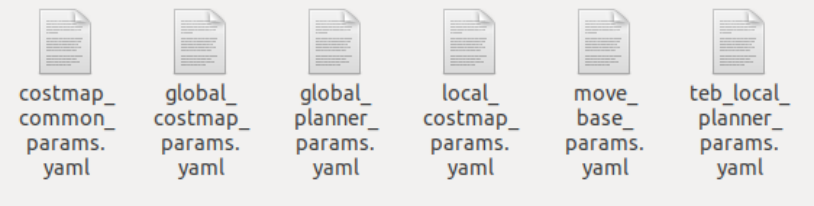
- **代价地图**

  通过 SLAM 构建的地图在导航中是不可以直接使用的，这是因为 SLAM 构建的地图是静态地图，但是在导航过程中，障碍物的信息是可变的，例如可能有障碍物被移走了，也可能有新的障碍物出现。因此导航中需要实时获取障碍物信息。此外，如果机器人处在障碍物边缘，也有可能因为惯性或者形状不规则而导致在转弯时和障碍物发生碰撞。所以，静态地图无法直接应用于导航，我们应在其基础上添加一些额外的辅助信息，例如基于传感器实时获取的障碍物信息、基于静态地图添加的膨胀区域等，而经过优化的地图就被称为代价地图。

  这样一来，我们的机器人就能在导航的同时实现实时避障的功能。

- **配置文件**

  本项目中 move_base 功能包所需的配置文件如下：



1. costmap_common_params.yaml：全局代价地图和局部代价地图的共同配置文件，包含在全局路径规划与局部路径规划时需调用的通用参数，如机器人的尺寸、距离障碍物的安全距离以及传感器的信息等。
2. global_costmap_params.yaml：全局代价地图的配置文件
3. local_costmap_params.yaml：局部代价地图的配置文件
4. move_base_params.yaml：move_base 本身的配置文件
5. global_planner_params.yaml：全局规划器的配置文件
6. teb_local_planner_params.yaml：使用 TEB 算法的局部规划器的配置文件，其中就包含了设定机器人最大速度、最小速度、最大加速度的参数。

3.2.2 部分参数设置

在配置文件 costmap_common_params.yaml 中，我们有

```
obstacle_range: 5.0     # 检测到距离小于 5 米的障碍物时，将其更新到代价地图中
raytrace_range: 6.0     # 清除掉代价地图中距离大于 6 米的障碍物
```

在配置文件 teb_local_planner_params.yaml 中，我们有

```
max_vel_x: 0.4                  # 最大线速度为 0.4 m/s
acc_lim_x: 1.0                  # 线速度的加速度限制为 1 m/s²

max_vel_theta: 1.0              # 最大角速度为 1.0 rad/s
acc_lim_theta: 0.5              # 角速度的加速度限制为 0.5 rad/s²
```

3.2.3 全局路径规划算法

我们全局路径规划使用了 move_base 功能包中所提供的 global_planner，它提供了 Dijkstra 算法和 A* 算法的实现。我们只需在配置文件 global_planner_params.yaml 中修改 use_dijkstra 这一配置项，设置为 true 时规划器使用 Dijkstra 算法进行路径规划，设置为 false 则使用 A* 算法进行路径规划。

- **Dijkstra 算法**

Dijkstra 算法是一种经典的图搜索算法，用于在加权图中找到从起点到目标节点的最短路径。它以起点为中心逐步扩展搜索，通过不断更新节点到目标点的距离值来找到最短路径。Dijkstra 算法适用于没有启发式信息的情况下，计算任意两点之间的最短路径。对于移动机器人的路径规划而言，Dijkstra 算法常用于全局路径规划，以计算机器人从起点到目标点的最短路径。

- **A* 算法**

  A* 算法是一种启发式搜索算法，结合了 Dijkstra 算法的最短路径搜索和启发式估计函数的启发式搜索。A* 算法通过评估节点的代价函数（通常是节点到目标节点的距离估计和已经走过的路径的代价之和）来进行搜索，以找到从起点到目标节点的最佳路径。A* 算法对于移动机器人的全局路径规划来说非常常用，因为它考虑了环境中的障碍物，可以在搜索过程中通过启发式估计函数来引导搜索方向，减少搜索空间，从而提高了搜索的效率。

3.2.4 局部路径规划算法

我们局部路径规划使用了 move_base 功能包中所提供的 teb_local_planner，它使用 TEB 算法来进行局部实时路径规划。

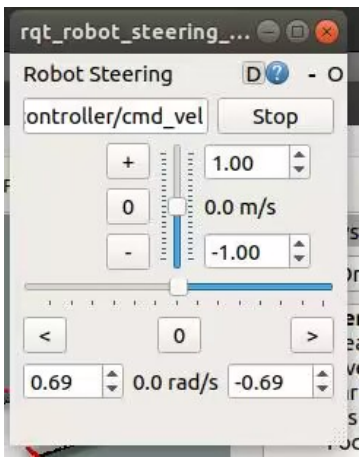
- **TEB 算法**

  TEB（Timed Elastic Band）算法是一种用于移动机器人的局部路径规划算法。TEB 算法通过建模机器人的运动约束和障碍物信息，生成平滑连续的轨迹，以满足机器人的运动约束并且避开障碍物。TEB 算法可以有效地应对复杂的运动约束和动态环境，同时提供可行的、舒适的轨迹规划。

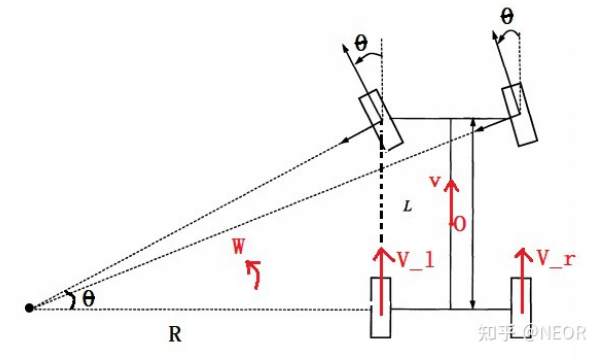
总的来说，我们通过将全局路径规划和局部路径规划相结合，来实现本项目整体的路径规划以及导航的功能。全局路径规划算法（Dijkstra 算法和 A* 算法）负责在整个地图中搜索最短路径，以指导机器人从起点到目标点的整体导航；而局部路径规划算法（TEB 算法）负责在机器人当前位置附近生成平滑且连续的局部最优路径，以适应实时环境的变化和优化机器人的运动轨迹，同时也可以实现实时避障的功能。

3.3 方向盘功能实现

在本项目中，steer_drive_ros 功能包实现了方向盘的功能，模拟应用场景中在有需要时通过人为的方式来远程控制机器人的移动。如下图所示，可以通过点击“+”，“-”，“<"，">"来分别控制小车“前”，“后”，“左”，“右”的移动。



在导入机器人的 urdf 模型后，需要程序去驱动对应的关节和实体完成指定的转动。驱动程序需要将接收到的速度指令转换为机器人每个轮子的转速和转向关节的旋转角度，即需要将机器人整体的线速度和角速度分别转换成后轮各自的转速和前轮转向关节的旋转角度。示意图如下：

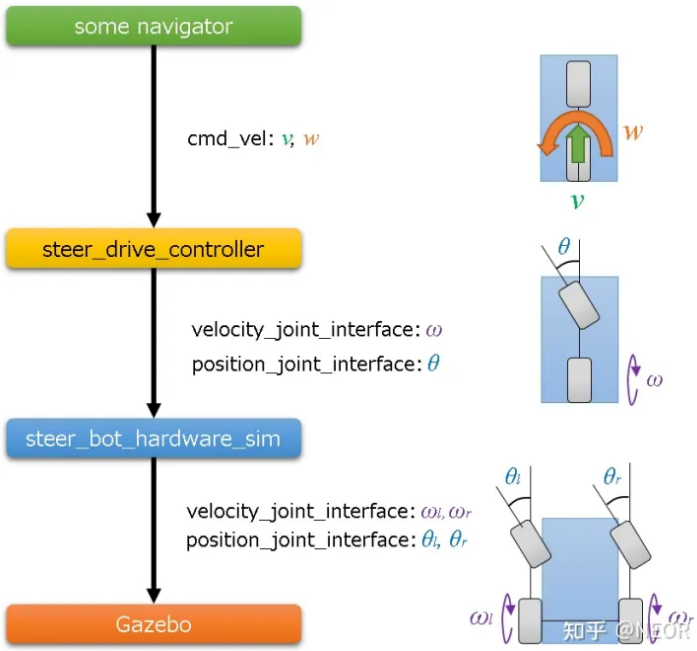


假设 t 时刻，o 点接收到速度指令（v, w），mini 左前、右前轮转向角度 θ 一致（现实中内侧转向角度略大），前后轴距为 L；可求解出后轮的转速和前轮的转向角度。公式如下：

$$\begin{cases} v_l = v \bullet \left(1 + \frac{\tan\theta}{2}\right) \\ v_r = v \bullet \left(1 - \frac{\tan\theta}{2}\right) \\ \theta = \tan^{-1}\left(\frac{\omega \bullet L}{v}\right) \end{cases}$$

同理，根据实际的执行情况，可以根据左右后轮的转速和前轮的转向角度逆推出后轮中点出的线角速度（v, w），从而推算出机器人的轨迹信息。这样一来，就可以通过计算程序驱动 urdf 模型了。

steer_drive_ros 功能包集中内置了 urdf 在 gazebo 下的驱动程序 steer_bot_hardware_gazebo 以及完成计算的程序 steer_drive_controller；整个功能包集的运行过程如下图示：



4. 总结与展望

4.1 目前的局限性

1. 我们目前仅搭建了几个结构简单的场景供机器人工作，切换到较为复杂的环境中后机器人还能否顺利工作还处于未知状态。
2. 硬件选型，如传感器选型，以及机器人运行速度等参数的设定目前还并不是最优的。
3. 人机交互方面，目前只能通过简单的点击场景中的某个点来实现移动，缺乏更灵活和智能的交互方式。

4.2 未来改进方向

1. 导入与现实环境复杂程度类似的场景供机器人工作运行。
2. 改进导航算法和技术，使机器人能够适应复杂的实际环境。可以考虑使用深度学习技术，来实现实时感知和环境理解能力，从而更好地避开障碍物并选择最优路径。
3. 引入更先进的人机交互方式，例如语音识别和自然语言理解，使用户能够通过语音指令与机器人进行交互，提供更智能和自然的导航体验。
4. 优化机器人的移动平台，改进底层控制算法和硬件设计，以提高机器人的移动性能和稳定性，适应更多种类的地面环境。
5. 进一步扩展应用场景，除了导航服务外，可以考虑为用户提供更多个性化的服务，如点播歌曲、订餐等，提升用户体验。

项目GitHub地址：https://github.com/WSYits/Robotics

项目成员分工占比：

学号	姓名	主要工作	贡献比
2051840	梁厚	• 前期技术方案调研 • 环境配置 • 建图功能实现及可视化	33.3%
2053171	白钰	• 前期技术方案调研 • 机器人选型 & 场景导入 • 方向盘功能实现	33.3%
2051849	王崧宇	• 前期技术方案调研 • 路径规划算法调研 • 导航 & 避障功能实现	33.3%