

ROS 机器人操作系统仿真

项目成员：张蕊，陈易菲，谢日排·艾拜都拉

指导老师：徐曼，刘大明

学校：上海电力大学

目录

摘要	1
第 1 章 概述	3
1.1 项目背景及意义	3
1.2 ROS 发展史简介	3
1.3 应用场景和需求	4
1.3.1 功能需求	4
1.3.2 非功能需求	5
1.4 本文创新点	6
1.5 组织结构	7
第 2 章 系统框架及模块设计	7
2.1 概述	7
2.2 整体架构设计	7
2.3 各模块功能描述及其相互关系	8
2.3.1 控制系统	8
2.3.2 传感器	9
2.3.3 执行器	9
2.3.4 环境 + 机器人本体	9
2.3.5 导航	10
2.3.6 通信	10
2.3.7 协作方式	10
2.4 本章小结	11
第 3 章 四轮小车整体设计与仿真的实现	11
3.1 概述	11
3.2 四轮小车的设计与实现	11
3.2.1 构建小车模型	11
3.2.2 RViz2 展示	13
3.3 差速模型	14
1. 速度控制机制	14
2. 速度与物理特性	17
3. 总结	18
3.4 控制器插件	18
3.4.1 控制器插件概述	18
3.4.2 设计思路	19
3.4.3 实现方法	20
3.5 Gazebo 仿真实现	21
1. 创建工作空间与初始化	21
2. 配置控制器插件	21
3. 启动 Gazebo 并显示模型	22
3.6 本章小结	22
第 4 章 实现多节点通讯	22
4.1 概述	22
4.2 Topic 方式通讯	23
4.2.1 发布者实现	23

4.2.2 订阅者实现	24
4.2.3 自定义消息类型	25
4.3 Service 方式通讯	26
4.3.1 服务端实现	26
4.3.2 客户端实现	27
4.3.3 自定义服务类型	29
4.4 本章小结	29
第 5 章 基于 Nva2 导航功能的设计与实现	29
5.1 概述	29
5.2 SLAM 算法选择与实现	30
5.3 单点导航与多点导航的实现	31
5.4 路径规划与动态避障策略	32
5.5 导航参数优化	33
5.6 本章小结	34
第 6 章 可视化界面设计与实现	34
6.1 概述	34
6.2 用户界面设计原则	35
6.3 控制命令输入界面	36
6.4 小车状态监控界面	37
6.5 本章小结	38
第 7 章 系统测试与分析	38
7.1 概述	38
7.2 测试环境搭建	38
7.3 功能测试	38
第 8 章 总结与展望	49
8.1 总结	49
8.2 展望	49

摘要

随着机器人技术的进步，移动机器人在工业自动化、物流配送、家庭服务等领域的重要性日益凸显。本项目旨在构建一个基于 ROS 框架的四轮自主导航小车平台，致力于解决复杂环境下的定位、地图构建与路径规划问题。

本次项目的主要工作如下：

1. 系统设计与实现

1) 物理模型构建

底盘与机械结构：选择了适合室内环境使用的四轮差速驱动结构，具备良好的稳定性和灵活性。

传感器配置：集成了激光雷达（LiDAR）、超声波传感器、IMU（惯性测量单元）等多种传感器，为环境感知提供了丰富的数据源。

XACRO 定义：利用 XACRO 语言简化了机器人的 URDF（统一机器人描述格式）文件，使得底盘、车轮以及各种传感器的位置、姿态等属性易于管理和调整。

碰撞检测与惯性参数设置：准确设定了各部件的碰撞体积和惯性矩阵，确保仿真环境中物理行为的真实性。

2) 高级控制逻辑

运动控制器：基于 ROS 的 Gazebo 模拟器中实现了差速驱动模型的运动控制器插件，支持精确的速度控制和转向管理。

关节状态发布：通过编写自定义节点定期更新并发布关节状态消息，保证 RViz2 可视化工具中的模型能实时反映实际运动情况。

2. 多节点通讯机制

Topic 定义：创建了一系列标准化的通信话题，用于传输传感器原始数据、处理后的感知信息、控制系统指令等。

定时通讯服务：开发了定制化的服务接口，允许用户按需请求特定类型的数据或执行特定动作。

rqt 工具应用：借助 rqt_gui 及其插件测试通信链路，优化了消息传递效率，并增强了系统的可调试性。

3. 导航功能

SLAM 算法集成：采用了 Cartographer 或 hector_slam 等开源 SLAM 库，结合 LiDAR 数据完成即时定位与地图构建。

路径规划与避障：引入了 ROS 2 Nav2 栈，它包含了从全局路径规划到局部避障的一整套解决方案，确保机器人能够在动态变化的环境中安全高效地移动。

单点/多点导航：实现了基于目标坐标的自动导航功能，支持用户指定多个中途停留点，适用于巡逻、导览等多种应用场景。

4. 可视化界面

主界面设计：设计了一个直观易用的操作面板，集成了启动、停止、重置等功能按钮，以及实时显示机器人状态的小部件。

路径显示页面：设置了用于展示路径规划结果的页面，包括当前路线、历史轨迹、障碍物分布等元素。

状态同步机制：采用 provider 模式集中管理所有 UI 组件的状态，确保界面上的信息始终保持最新且一致。

5. 测试与分析

测试环境搭建：构建了包含多种地形特征的虚拟测试场景，用以评估不同条件下机器人的性能表现。

测试案例设计：制定了涵盖静态环境、动态障碍物、复杂路径等多个维度的测试方案，全面检验系统的可靠性和适应性。

数据分析报告：收集并分析了大量实验数据，总结出影响系统效能的关键因素，并提出了针对性改进建议。

本项目通过对 ROS 框架下四轮小车平台的设计与实现，探索了移动机器人领域的前沿技术和实践方法。未来将着眼于进一步提升智能化水平，如加入深度学习算法提高环境理解能力；探索人机交互新方式，例如语音控制、手势识别等；并且尝试与其他智能设备互联互通，共同打造更加智慧的生活空间。

第 1 章 概述

1.1 项目背景及意义

随着科技的不断进步，机器人技术已经成为现代工业和日常生活中的重要组成部分。从自动化工厂到智能家居，机器人在提高效率、减少人力成本和改善生活质量方面展现了巨大的潜力。特别是在物流配送、环境监测、医疗辅助等领域，移动机器人的应用更是日益广泛。然而，要实现这些应用场景，机器人必须具备自主导航能力，能够感知周围环境、规划路径并避开障碍物，同时还要能与其他系统或设备进行有效的信息交换。

本项目旨在开发一款基于 ROS (Robot Operating System) 框架的四轮小车平台，它不仅能够执行基本的运动控制，还支持复杂的导航任务，如 SLAM (Simultaneous Localization and Mapping, 即时定位与地图构建)、单点或多点导航以及动态避障等。选择四轮差速驱动的小车作为研究对象，是因为其结构简单、稳定性和可控性好，适合初学者学习和研究，同时也满足了大多数室内移动操作的需求。

通过整合先进的传感器技术（例如激光雷达、摄像头、IMU 等），结合高效的算法处理（如 Graph-based SLAM 建图、A*或 Dijkstra 路径规划算法），我们将赋予小车智能感知环境的能力，使其能够在未知环境中创建精确的地图，并根据实时数据调整行进路线。此外，利用 ROS 强大的通信机制，可以轻松地将小车集成到更大的机器人网络中，或者与外部控制系统对接，从而扩展其应用范围和服务功能。

综上所述，本项目不仅对当前机器人技术进行了有益探索，增强了自主导航和智能感知能力，为未来机器人技术的发展奠定了坚实基础；同时，也为高校学生和业余爱好者提供了一个开放的学习平台，帮助他们通过实际搭建和编程实践掌握机器人技术的基本原理与应用方法，培养解决复杂问题的能力和创新能力。我们相信，随着项目的推进和完善，它将显著提升社会效益，加速服务型机器人的市场化进程，并为教育、科研及产业应用带来深远影响。

1.2 ROS 发展史简介

ROS (Robot Operating System) 并不是一个传统意义上的操作系统，而是一个为机器人软件开发提供的中间件集合。它提供了一系列的库和工具，旨在简化机器人应用程序的创建过程。ROS 项目起源于 2007 年斯坦福大学的人工智能实验室，并在 2010 年由非营利组织 Willow Garage 正式发布第一

个公开版本。最初，ROS 是作为研究项目的一部分开发的，旨在为学术界和工业界提供一个共同的标准平台来加速机器人技术的发展。

随着 ROS 1 的成功，开发者们意识到需要解决一些局限性，如单点故障问题、安全性不足以及对实时系统的支持不够等问题。因此，在 2015 年，ROS 2 被引入。ROS 2 采用了 DDS (Data Distribution Service) 作为底层通信协议，增强了系统的可靠性和安全性。此外，ROS 2 引入了分层架构，使得不同部分可以独立进化，同时保持向后兼容性。这些改进使 ROS 2 成为了一个更加强大和灵活的平台，适用于更大规模和更复杂的应用场景。

ROS 的关键特征包括分布式计算环境，允许跨多台计算机运行多个进程，并通过标准化的消息格式进行通信；丰富的库和工具，包括传感器驱动程序、算法实现、仿真器和其他辅助工具；强大的社区支持，拥有活跃的开源社区，不断贡献新的包和改进现有功能；模块化设计，使得开发者可以根据需求选择所需的组件，易于扩展和维护；以及跨平台支持，可以在 Linux、macOS 和 Windows 等多个操作系统上运行。

在机器人技术领域，ROS 扮演着至关重要的角色。它促进了研究人员之间的协作与资源共享，减少了重复劳动。对于初学者来说，ROS 降低了进入机器人编程领域的难度，提供了大量的教程和文档，帮助他们快速上手。由于其开放性和灵活性，ROS 成为了测试新想法的理想平台，许多前沿的研究成果都是基于 ROS 完成的。越来越多的企业也开始采用 ROS 或其衍生产品来开发商业级机器人解决方案，尤其是在服务型机器人和个人助理领域。

ROS 2 已经解决了许多 ROS 1 中存在的问题，并且正在逐步完善其生态系统。随着机器人技术的进步，预计 ROS 将集成更多先进的感知、规划和控制算法，同时也将进一步加强与其他 AI 技术的融合，例如深度学习和强化学习。此外，随着边缘计算和云计算的发展，ROS 可能会探索如何更好地利用这些新兴技术来优化机器人的性能和响应速度。总之，ROS 仍然是机器人技术研发中不可或缺的重要工具，并将继续引领这一领域的变革与发展。

1.3 应用场景和需求

机器人技术不仅大幅提高了生产效率、降低了人力成本，还改善了工作环境和生活质量。通过自动化和智能化的操作，机器人能够执行复杂且重复的任务，减少了人为错误并提升了工作的精确度。其在物流配送、医疗服务、环境监测、智能家居、农业自动化以及教育与研究等领域的广泛应用，显著提升了各行业的运营效率和服务质量，同时为人们的生活带来了更多的便利和安全保障。

1.3.1 功能需求

在开发基于 ROS (Robot Operating System) 框架的四轮小车平台过程中,明确的功能需求是确保项目成功实施的关键。为了实现一个高效、稳定且具备智能导航能力的小车系统,我们设定了多方面的需求目标,涵盖从基础的硬件设计到复杂的软件算法实现。

四轮小车设计:采用 Xacro 文件格式创建四轮小车的 URDF (Unified Robot Description Format) 模型,确保其结构合理且各部件尺寸比例准确。通过向模型中添加物理属性和传感器,使得小车更加贴近真实世界的行为和功能。此外,在 RViz2 中加载并展示小车模型,确保视觉效果符合预期,使开发者能够直观地查看和调整模型。最后,通过 Gazebo 仿真测试了小车的行为,验证了其在模拟环境中的性能,为后续的进一步仿真和实际应用打下了坚实的基础。

多节点通讯:通过定义和实现多个 Topic,确保各个节点可以实时共享数据,如传感器读数和控制指令等。为了保证客户端和服务端之间的稳定交互,开发了定时通讯服务,并定义和实现了自定义服务消息。利用 rqt 工具测试和验证通信链路的有效性,确保消息传递准确无误,并通过图形界面直观展示通信状态,便于监控和调试。此外,还提供了详细的命令行工具支持,使用户能够轻松检查节点和服务的状态信息,确保系统的透明性和可调试性,从而方便问题排查。

控制器插件与差速模型:通过集成 IMU 传感器和差速驱动模型,实现了精确的姿态测量和转向控制,增强了小车运动的稳定性和准确性。此外,还进行了多轮测试和参数调整,确保控制器性能满足设计要求,并能应对不同的操作环境和任务需求。

单点导航与自主导航功能:单点导航功能,使小车可以根据给定的目标位置自动规划路径并到达目的地,确保路径规划的效率和准确性。进一步利用 SLAM 技术扩展至自主导航,让小车在未知环境中探索并建立地图,同时避开障碍物找到最优路径,提升小车的智能水平和适应能力。

1.3.2 非功能需求

性能需求:

在实际应用中,特别是对于移动机器人而言,高效的性能是实现任务目标的基础。无论是物流配送中的快速响应,还是复杂环境下的精确导航,都需要小车具备出色的计算效率、响应速度和导航精度。只有这样,才能保证系统能够在各种条件下稳定可靠地执行任务,满足用户的期望。因此,本次项目需要达到以下性能要求:

1. 响应时间：确保小车能够快速响应控制指令和环境变化，特别是在动态环境中，如避障或路径调整。

2. 计算效率：优化算法以减少处理时间和资源消耗，保证系统在有限的硬件资源下高效运行。

3. 导航精度：通过精确的传感器数据处理和路径规划算法，确保小车能够在复杂环境中准确到达目标位置。

可靠性：

一个可靠的系统可以在长时间运行中保持稳定，避免因软件错误或硬件故障导致的意外停机。这对于需要持续运作的应用场景尤为重要，例如工业自动化或医疗服务等对连续性和稳定性有高要求的领域。因此，本次项目需要达到以下可靠性要求：

1. 稳定运行：设计和测试确保系统在长时间运行中保持稳定，避免因软件错误或硬件故障导致的意外停机。

2. 容错能力：实现冗余机制和错误恢复策略，使得小车可以在部分组件失效的情况下继续工作或安全停止。

1.4 本文创新点

1. 车辆外形的自定义设计：

本文的一个重要创新点在于四轮小车外形的自定义设计。我们结合实际应用需求，从零开始设计并搭建了车辆模型，包括机械结构、传感器安装位置和动力系统的布局优化。相比于传统的现成模型，我们的设计更加适配特定应用场景，并在结构紧凑性、传感器覆盖范围及运动稳定性上取得了显著优化。

2. 自定义消息和服务接口的开发：

本文开发了针对小车导航与运动控制的自定义消息类型（如 `VelocityAccelerate` 和 `TemperatureHumidity`），以满足特定的场景需求。这种灵活的消息设计不仅提高了系统的通信效率，还为机器人环境感知和运动状态反馈提供了更高的实时性和精确度。

3. 系统的集成创新：

本项目将车辆的自定义模型、控制器设计与 ROS 2 系统紧密结合，形成了一套完整的软硬件协同方案。从仿真到实车测试的全流程覆盖，使得该系统不仅具有理论价值，还具备实际应用潜力。尤其是在导航功能中，创新性地整合了路径规划与动态避障算法，提升了系统的自主性和智能性。

4. 开放性与扩展性设计

为提高系统在不同场景中的适用性，车辆外形设计、控制器逻辑和通信接口

都具有很强的可扩展性。本文提出的框架可以方便地扩展到其他类型的移动机器人或更多的多传感器融合场景，进一步拓展了系统的应用领域。

1.5 组织结构

本文的章节安排如下：

第一章概述，介绍项目的背景及意义和 ROS 的历史概述。同时介绍项目的主要创新点以及机器人的应用场景和需求。

第二章系统框架及模块设计，展示系统的整体架构设计，包含系统架构图或模块图。详细介绍每个模块的功能以及它们如何协作以完成整体任务。

第三章四轮小车整体设计与仿真的实现，详细描述四轮小车的设计（包括物理特性和传感器配置）、RViz2 展示、差速驱动模型、控制器插件设计以及 Gazebo 仿真的实现。

第四章实现多节点通讯，讨论多节点通讯的实现，涵盖 Topic 和 Service 机制、消息格式定义、rqt 工具验证及命令行工具支持。

第五章基于 Nav2 导航功能的设计与实现，介绍基于 Nav2 的导航功能，包括 SLAM 算法选择、单点与多点导航、路径规划和动态避障策略的具体实施与设计。

第六章可视化界面设计与实现，探讨用户界面的设计原则，描述控制命令输入和小车状态监控界面的具体实现，确保用户能够直观操作和实时监控小车的状态，提供流畅的交互体验。

第七章系统测试与分析，描述系统测试环境的搭建、功能测试用例设计、测试结果分析及性能评估，总结测试过程中学到的经验教训。

第八章总结与展望，汇总项目的主要发现，分析项目中遇到的挑战和未解决的问题，提出未来的改进方向和技术研究领域。

第 2 章 系统框架及模块设计

2.1 概述

本章详细介绍了四轮小车平台的系统框架及其各个模块的设计与实现。首先，我们展示了系统的整体架构设计，通过系统架构图或模块图直观呈现了系统的高层次布局，并解释了架构选择的理由及其优势。接着，我们深入探讨了每个模块的具体功能及其相互之间的协作机制，确保读者能够全面理解系统的工作原理和各部分的协同作用。

2.2 整体架构设计

为了清晰展示四轮小车平台的整体结构，我们首先引入了系统架构图（如图 2-1 所示）。该架构图涵盖了所有主要组件及其相互关系，提供了系统的高层次视图。通过这一图表，可以直观地理解各个模块的功能和它们之间的交互方式。

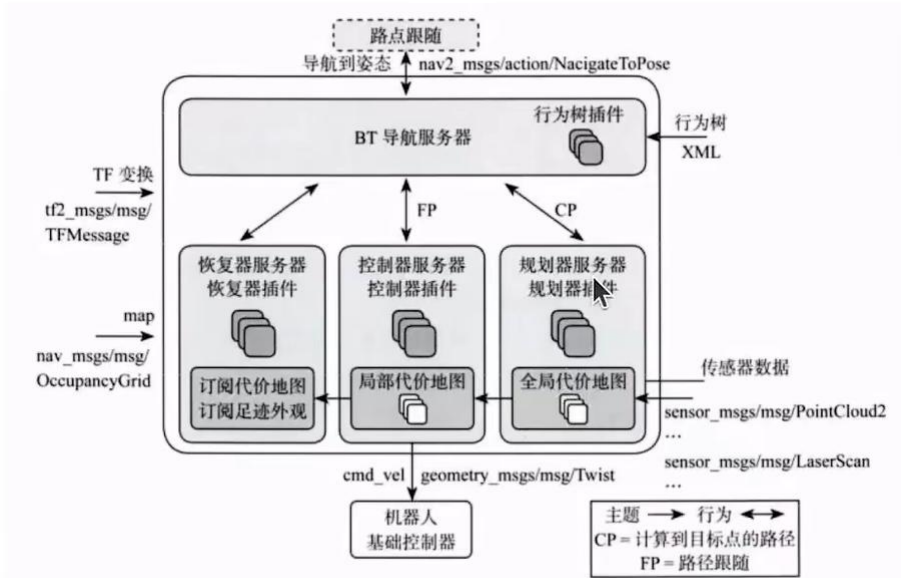


图 2-1 系统架构图

根据此次的设计，整个系统可以被划分为以下几个关键模块：

1. 控制系统：负责处理传感器数据并生成控制指令。
2. 传感器：收集环境信息和机器人状态数据。
3. 执行器：根据控制系统的指令执行具体动作。
4. 环境+机器人本体：包括外部环境和机器人本身的物理状态。
5. 导航模块：实现自主导航，包括路径规划、避障和目标点导航。
6. 通信模块：实现小车与外部系统的通信。

2.3 各模块功能描述及其相互关系

2.3.1 控制系统

控制系统是整个系统的“大脑”，负责处理来自传感器的数据，并根据预定义的算法或学习模型生成相应的控制指令。它决定了小车的行为模式，如前进、后退、转弯、停止等动作。此外，控制系统还负责监控所有子系统的状态，确保它们在安全范围内运行，并能在遇到异常情况时采取适当的措施。

控制系统接收来自传感器的数据，进行实时分析并决定下一步的动作，随后向执行器发送控制指令以驱动小车运动。同时，它通过与环境 and 机器人本体的持续交互，利用反馈机制动态调整策略，确保小车能够适应变化的环境条件并稳定运行。

2.3.2 传感器

传感器模块由多个不同类型的传感器组成，包括但不限于激光雷达（LiDAR）、摄像头、惯性测量单元（IMU）等。这些传感器用来收集周围环境的信息，例如障碍物位置、地形特征、速度和方向等。准确的传感信息对于实现精确的定位、导航和避障至关重要。

传感器模块为控制系统提供关键的输入数据，使其能够实时了解当前环境状况并做出相应决策。同时，部分传感器如 IMU 可以直接影响执行器的行为，例如通过调节电机速度来维持小车的稳定性和精确控制。这种紧密的传感器与执行器之间的交互确保了系统的响应速度和动作精度。

2.3.3 执行器

执行器是指那些根据控制系统发出的命令执行具体物理动作的组件，比如电动机、舵机等。它们的作用是将电能转换成机械能，从而驱动小车移动或者操作某些机构。执行器的表现直接影响到了小车的响应速度、精度及可靠性。

执行器从控制系统接收指令后，根据指令改变自身状态（如转速、角度），并将执行结果反馈给控制系统，以便实时调整策略和优化性能。这种闭环控制机制确保了系统的响应精度和稳定性。

2.3.4 环境 + 机器人本体

这部分包含了小车所在的外部环境以及小车本身的物理特性，如尺寸、重量、动力系统等。理解这一点非常重要，因为它界定了小车的操作范围和能力边界。同时，考虑到环境因素（如地面类型、光照条件），可以帮助优化传感器的选择和布置。

环境和机器人本体的特性不仅影响传感器的有效性和适用性（例如，不同的光照条件会影响摄像头的效果），还限制了执行器的最大性能（如地面摩擦力会制约电机所能提供的最大加速度），同时作为控制系统决策的重要参考依据，特别是在路径规划和潜在风险预测时，这些因素至关重要。通过

综合考虑这些方面，控制系统能够做出更加精准和适应性的决策，确保小车在各种环境下都能高效、安全地运行。

2.3.5 导航

导航模块是机器人系统中实现自主移动的关键部分，它通过融合来自传感器的数据，使小车能够构建环境模型、确定自身位置，并规划从当前位置到目标点的最优路径。导航不仅依赖于高精度的地图构建和自我定位（例如使用 SLAM 技术），还需要具备全局路径规划能力，以计算出避开所有已知障碍物的安全路径。此外，局部路径规划功能允许小车根据实时传感器数据动态调整其路线，确保在遇到未预见的障碍时能迅速做出反应，避免碰撞。导航模块还必须集成动态避障算法，以便小车可以在变化的环境中安全有效地行驶。这种复杂的决策过程要求导航模块持续与控制系统交互，接收最新的环境感知信息并反馈执行结果，确保小车能够在复杂多变的环境中高效运行。

2.3.6 通信

通信模块负责保障机器人与外部世界之间的信息交换，确保了小车可以接收来自远程操作员或其它系统的指令，并且能够反馈自身的状态和周围环境的信息。该模块支持多种通信协议，如 Wi-Fi、蓝牙或 4G/5G 网络连接，使得小车可以在不同环境下保持稳定的数据传输。通信模块不仅促进了小车与控制中心之间的命令和状态更新，还支持多机器人之间的协作通信，使得它们可以共享任务相关信息，协同完成复杂作业。此外，为了保证通信的安全性和可靠性，通信模块通常会包含加密机制和错误检测/纠正功能，确保数据完整无误地传递。通过提供一个双向、实时的信息交流渠道，通信模块加强了小车的适应性和灵活性，使其能够更好地融入更广泛的自动化系统中。

2.3.7 协作方式

各个模块之间并非孤立存在，而是紧密相连并通过一系列接口和协议实现了高效协作：

数据流：传感器持续不断地采集环境数据并传输给控制系统；控制系统基于接收到的数据计算出最优行动方案，并通过通信总线或网络将指令传达给执行器。

反馈循环：执行器的动作结果会被再次感知，形成闭环控制，使得系统可以自我校正，保证动作的准确性。

同步与协调：各模块之间的时间同步至关重要，尤其是在涉及多传感器融合或多执行器协同工作的情况下。通过采用统一的时间基准或事件触发机制，可以确保所有模块之间的良好配合。

资源分配：当多个任务并发执行时，控制系统需要合理地调配计算资源和电力供应，优先保障关键任务的顺利完成。

2.4 本章小结

综上所述，本章通过对四轮小车平台系统框架及模块设计的深入探讨，揭示了其内部运作原理和各组成部分之间的复杂交互。从整体架构到具体模块的功能，再到它们之间的协作机制，我们提供了一个完整的视角来理解和构建一个高效、稳定的四轮小车系统。

第3章 四轮小车整体设计与仿真的实现

3.1 概述

本章节详细介绍了四轮小车的设计与实现，涵盖了从物理模型构建到高级控制逻辑的各个方面。主要分为四个个主要部分：构建小车模型、差速驱动模型的应用、控制器插件的设计与实现，以及 Gazebo 仿真实现中的视觉反馈增强方法。

3.2 四轮小车的设计与实现

3.2.1 构建小车模型

为了创建四轮小车的 URDF 模型，我们首先使用 XACRO (XML 宏) 来简化和模块化定义，我们创建了多个 XACRO 文件，分别定义底盘、车轮、传感器（如 IMU、相机和激光雷达）、惯性参数及物理特性等组件。每个组件的位置、姿态和碰撞体积也被明确指定，确保仿真准确性。通过 XACRO 宏，我们将这些独立的部分连接起来，最后，利用 `xacro` 命令或 ROS 2 中的 `robot_state_publisher` 节点，将所有 XACRO 文件合并并转换为完整的 URDF 文件，用于在仿真环境中加载和运行四轮小车模型，具体框架如图 3-1。以下是此次建模的关键点：

1. 基础部件：

底盘 (Base Link)：作为整个机器人模型的基础链接，它决定了其他所有组件的位置和方向。底盘的几何形状、材质属性以及视觉和碰撞属性需要被详细定义。

车轮 (Wheels)：每个车轮都是一个独立的链接，并通过旋转关节 (Revolute Joint) 连接到底盘上。为了确保仿真时的行为准确，需要为每个车轮定义适当的惯性矩。使用 XACRO 宏可以简化重复代码，使得四个车轮的定义更加简洁和一致。

2. 传感器：

IMU (惯性测量单元)：用于提供加速度和角速度数据。

相机 (Camera)：用于视觉感知。定义相机的视场角 (FOV)、图像分辨率以及其他相关参数，确保其位置能够有效捕捉周围环境。

激光雷达 (Laser Scanner)：用于距离测量。定义扫描角度范围、频率等参数，以适应不同的应用场景需求。同样地，明确其相对于底盘的位置，以便正确获取周围环境信息。

3. 位姿：

在定义每个组件时，必须明确指出其相对于底盘或其他参考点的位置 (x, y, z 坐标) 和姿态 (roll, pitch, yaw 角度)。通过 XACRO 宏中的参数传递来实现，确保所有部件都能按照预期进行组装。

4. 碰撞体积：

为每个链接定义适当的碰撞体 (collision geometry)，以确保 Gazebo 能够正确模拟碰撞检测。通常使用简单的几何形状如盒体 (box)、圆柱 (cylinder) 或球体 (sphere) 近似实际物体。这样不仅可以提高仿真的准确性，还能优化计算性能。

5. 惯性单元：

使用 XACRO 宏计算并设置每个链接的惯性矩。对于复杂的几何形状，可以创建一个专门的 XACRO 文件来处理不同类型的惯性矩计算，如长方体、圆柱体和球体。这些惯性参数对机器人的运动行为至关重要，因此需要根据实际情况精确调整。

6. 物理特性：

定义材料属性，如质量、摩擦系数等，以影响机器人的运动行为。这同样可以在 XACRO 文件中完成。

7. 贴合地面 (虚拟部件)：

为了让车轮保持贴地状态，可以降低车轮的高度，使其默认情况下接触地面。

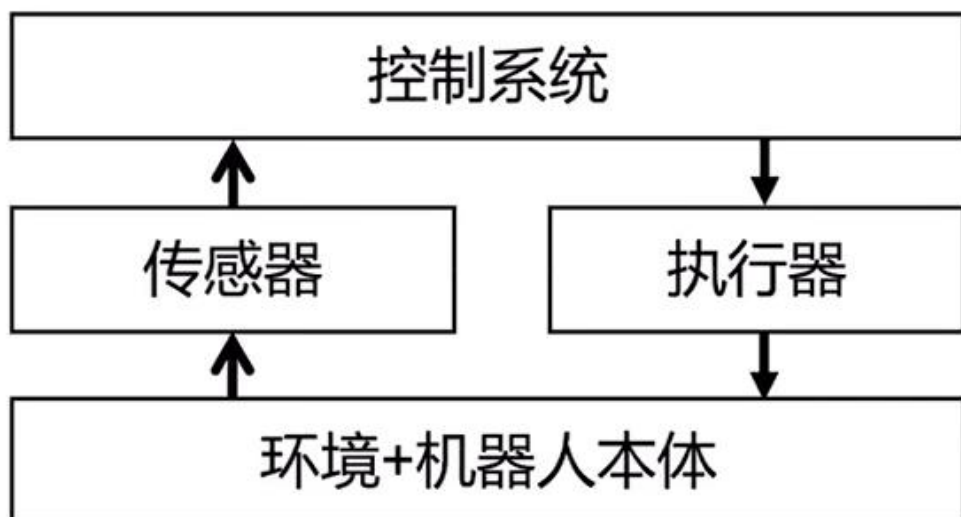


图 3-1 小车架构图

3.2.2 RViz2 展示

为了在 RViz2 中正确展示四轮小车模型，需要完成以下几个关键步骤：

1. 启动 RViz2 节点

在发射文件中加入 RViz2 节点启动命令，并指定配置文件路径。RViz2 配置文件可以自定义显示设置，如相机视角、显示的传感器数据等。

2. 加载机器人描述

通过 `robot_state_publisher` 发布机器人的静态 TF 变换和链接信息。这一步骤至关重要，因为它提供了机器人各个部件之间的相对位置关系，使得 RViz2 能够正确地渲染模型。

3. 发布关节状态

使用 `joint_state_publisher_gui` 或 `joint_state_publisher` 来手动输入或自动更新关节的角度，使模型能够在 RViz2 中动态展示。这对于模拟车轮旋转和其他运动非常有用。

如图 3-2 四轮小车在 rviz2 中展示效果示意图：

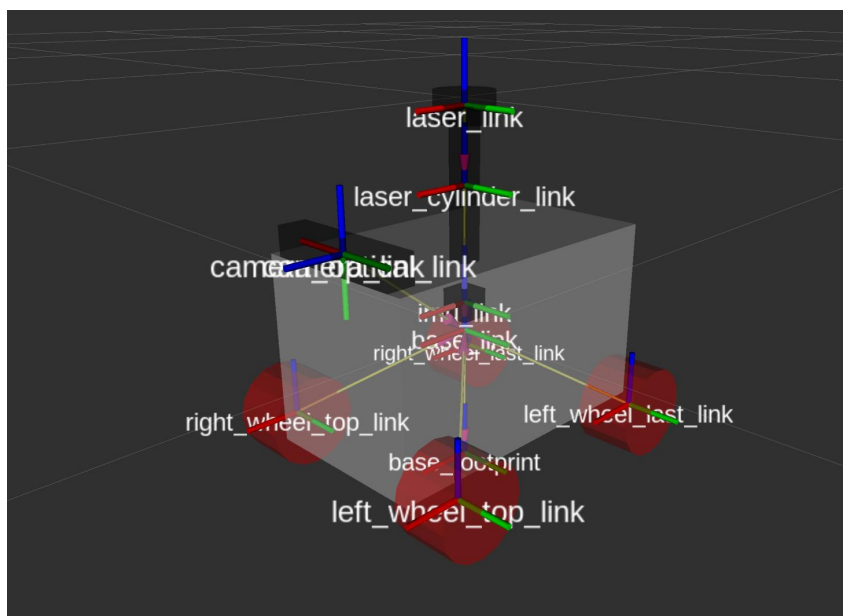


图 3-2 四轮小车模型展示

3.3 差速模型

1. 速度控制机制

在四轮差速运动模型中，四个轮子都是驱动轮，且同一侧的前后两个轮子的速度相同，通过控制左右两侧轮子的同速或差速来实现机器人的直行或转弯。在不考虑物理特性的情况下，小车底盘的运动可以简化为在二维平面上的运动。因此先建立二维坐标系，标记出底盘运动模型的参数。

四轮差速运动模型如图 3-3 所示：

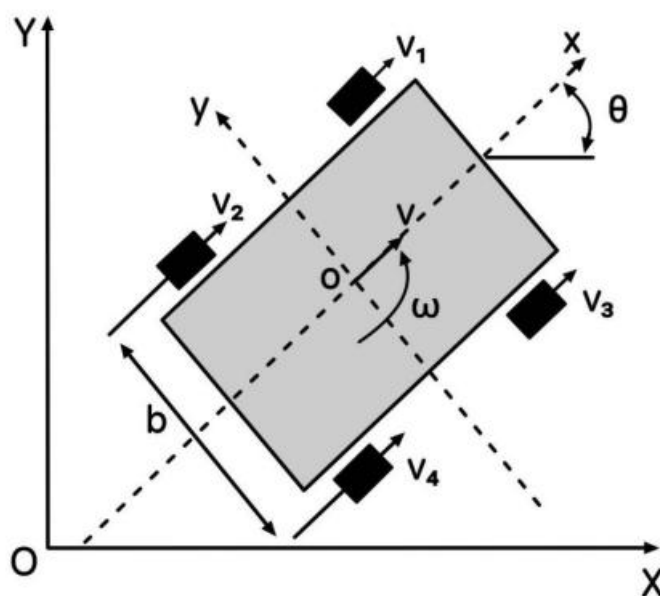


图 3-3 四轮差速运动模型图

图 1 中的符号说明见表 1。

表 1 符号说明表

符号	说明
XOY	全局坐标系
xoy	局部坐标系
v	底盘线速度
ω	底盘角速度
v_i	第 i 个车轮的线速度
b	左右车轮间距
θ	位姿角

1) 直线运动

当小车底盘进行直线运动时，四个轮子的速度都相同，角速度为 0。即

$$\begin{cases} v_L = v_R = v_1 = v_2 = v_3 = v_4 \\ \omega = 0 \end{cases} \quad (1-1)$$

式中, v_L 为左侧轮子的速度, v_R 为右侧轮子的速度。

2) 曲线运动

当小车底盘进行曲线运动时，可以看作刚体绕瞬时旋转中心 ICR 做圆周运动。因此底盘的线速度和角速度满足下式：

$$\omega = \frac{v}{r} \quad (2-1)$$

式中, r 为点 o 到点 ICR 的距离，即圆周运动的半径。

由于同一侧的前后轮速度相等，所以各个轮子的速度满足下式：

$$\begin{cases} v_L = v_{1x} = v_{2x} \\ v_R = v_{3x} = v_{4x} \end{cases} \quad (2-2)$$

式中, v_{ix} 为第 i 个车轮的线速度 v_i 在 x 轴的分量。

在局部坐标系 xoy 中，假设 r 与 y 轴的夹角为 α ，由式 (2-1) 可得：

$$\omega = \frac{v}{r} = \frac{v \cos \alpha}{r \cos \alpha} = \frac{v_x}{r_y} = \frac{v \sin \alpha}{r \sin \alpha} = \frac{v_y}{r_x} \quad (2-3)$$

式中, v_x 为 v 在 x 轴的分量, v_y 为 v 在 y 轴的分量, r_x 为 r 在 x 轴的分量, r_y 为 r 在 y 轴的分量。

刚体在旋转过程中，各个位置的角速度都相同，因此四个轮子的角速度也是 ω 。由式 (2-3) 类推可得

$$\omega = \frac{v_i}{r_i} = \frac{v_i \cos \alpha_i}{r_i \cos \alpha_i} = \frac{v_{ix}}{r_{iy}} = \frac{v_i \sin \alpha_i}{r_i \sin \alpha_i} = \frac{v_{iy}}{r_{ix}} \quad (2-4)$$

式中， r_i 为第 i 个轮子到 ICR 的距离， r_{ix} 为 r_i 在 x 轴的分量， r_{iy} 为 r_i 在 y 轴的分量， v_{iy} 为第 i 个车轮的线速度 v_i 在 y 轴的分量。

将式 (2-3) 和式 (2-4) 整理可得

$$\omega = \frac{v}{r} = \frac{v_x}{r_y} = \frac{v_y}{r_x} = \frac{v_{ix}}{r_{iy}} = \frac{v_{iy}}{r_{ix}} \quad (2-5)$$

同时， r_i 和 r 在 y 轴上的分量满足下式

$$\begin{cases} r_{1y} = r_{2y} = r_y - \frac{b}{2} \\ r_{3y} = r_{4y} = r_y + \frac{b}{2} \end{cases} \quad (2-6)$$

由式 (2-5) 和式 (2-6) 可得

$$\begin{cases} v_L = \omega \cdot (r_y - \frac{b}{2}) = \omega \cdot r_y - \omega \cdot \frac{b}{2} = v_x - \omega \cdot \frac{b}{2} \\ v_R = \omega \cdot (r_y + \frac{b}{2}) = \omega \cdot r_y + \omega \cdot \frac{b}{2} = v_x + \omega \cdot \frac{b}{2} \end{cases} \quad (2-7)$$

将式 (2-7) 整理可得

$$\begin{bmatrix} v_x \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{b} & \frac{1}{b} \end{bmatrix} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \quad (2-8)$$

小车底盘的位姿状态用状态向量 $q = [x, y, \theta]^T$ 表示，速度向量可以用 $\eta = [v, \omega]^T$ ，全局坐标系 XOY 与底盘坐标系 xoy 之间的关系如下：

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \quad (2-9)$$

由于小车不能直接沿某些方向进行运动，所以当转向时，底盘不可避免地会发生侧移现象。因此引入 Pfaffian 形式的非完整约束^[1]以确保模型更贴近实际情况。非完整约束是指那些不能直接由位置坐标积分得到的约束条件，符合当前小车的情况。具体公式如下：

$$A(q)\dot{q} = 0 \quad (2-10)$$

其中， $A(q) = [\cos \theta \quad \sin \theta \quad 0]$ ，即

$$[\cos \theta \quad \sin \theta \quad 0] \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = 0 \quad (2-11)$$

由图 1 可知，式 (2-11) 的表达式为

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2-12)$$

将式 (2-8) 代入式 (2-12) 得

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\cos \theta}{2} & \frac{\cos \theta}{2} \\ \frac{\sin \theta}{2} & \frac{\sin \theta}{2} \\ -\frac{1}{b} & \frac{1}{b} \end{bmatrix} \begin{bmatrix} v_L \\ v_R \end{bmatrix} \quad (2-13)$$

2. 速度与物理特性

在模型进行仿真时，会加入物理特性。因此计算小车底盘的实际速度和角速度需要考虑摩擦力、惯性等因素。

1) 摩擦力

摩擦力是机器人与地面之间相互作用的重要因素，它分为静摩擦力和动摩擦力：

静摩擦力：当机器人试图开始移动或停止时起作用。

动摩擦力：当机器人正在移动时起作用，通常小于静摩擦力。

假设我们有最大摩擦力 F_{\max} ，它可以限制轮子的最大牵引力。如果小车达到设定的速度所需加速度超过了摩擦力所能提供的最大加速度，则实际速度会受到限制。

由滑动摩擦公式可知

$$F_{\max} = \mu mg \quad (3-1)$$

式中， μ 为摩擦系数， m 为小车质量， g 为重力加速度。

实际线加速度 a 受到摩擦力限制：

$$a = \min\left(\frac{F}{m}, \frac{\mu mg}{g}\right) = \min\left(\frac{F}{m}, \mu g\right) \quad (3-2)$$

式中， F 为小车牵引力。

因此，实际速度 v 的变化率会受到摩擦力的影响：

$$\dot{v} = \min\left(\frac{F}{m}, \mu g\right) \quad (3-3)$$

2) 惯性

惯性描述了物体保持其运动状态的能力，对于机器人来说，主要体现在旋转惯性上。旋转惯性通过转动惯量 I 和角加速度 α 来表达。当施加扭矩 τ 时，小车会产生角加速度 α ，进而改变角速度 ω 。

$$\dot{\omega} = \min\left(\frac{\tau}{I}, \frac{\mu m g b}{2I}\right) \quad (3-4)$$

3. 总结

在小车底盘运动过程时，控制系统通常会直接给小车的轮子设定速度。通过上述公式的计算，可以得到底盘的线速度 v 和角速度 ω 。然后，基于 v 和 ω 再利用前一时刻的底盘位姿可以通过数值积分的方法推算当前时刻的底盘位姿。由于这个计算依赖于几何模型来进行连续的位姿预测，因此即使是微小的初始误差也会随着车辆行驶距离的增加而逐渐累积。这种误差累积效应最终会导致位姿估计的准确性大幅下降，特别是在长时间或大范围移动后，位姿信息可能会变得不可靠甚至完全失效。

3.4 控制器插件

3.4.1 控制器插件概述

控制器插件在四轮小车的仿真和实际运行中扮演着至关重要的角色。如图 3-4 所示，它们作为连接物理模型和控制算法的桥梁，允许开发者通过编写自定义逻辑来精确控制机器人的运动和其他行为。对于四轮小车而言，控制器插件主要用于模拟电机驱动、转向机制、速度控制等功能，并且可以集成高级控制策略如 PID 控制、路径规划或避障算法。

```

35 # Diff drive controller (仍然使用速度控制, 四轮差速模型)
36 firstbot_diff_drive_controller:
37
38   ros_parameters:
39     left_wheel_names: ["left_wheel_top_joint", "left_wheel_last_joint"] # 确保关节名称正确
40     right_wheel_names: ["right_wheel_top_joint", "right_wheel_last_joint"] # 确保关节名称正确
41
42     wheel_separation: 0.17
43     wheel_radius: 0.032
44
45     wheel_separation_multiplier: 1.0
46     left_wheel_radius_multiplier: 1.0
47     right_wheel_radius_multiplier: 1.0
48
49     publish_rate: 50.0
50     odom_frame_id: odom
51     base_frame_id: base_footprint
52     pose_covariance_diagonal: [0.001, 0.001, 0.0, 0.0, 0.0, 0.01]
53     twist_covariance_diagonal: [0.001, 0.0, 0.0, 0.0, 0.0, 0.01]
54
55     open_loop: true
56     enable_odom_tf: true
57
58     cmd_vel_timeout: 0.5
59     use_stamped_vel: false
60     use_velocity_cmd: true # 确保使用速度命令

```

图 3-4 控制器插件配置信息

主要特点：

灵活性：支持根据特定需求定制控制器逻辑。

可扩展性：易于添加新的控制逻辑或修改现有功能。

实时性：确保控制命令能够及时响应并执行。

反馈机制：通过传感器数据（如 IMU、编码器）实现更稳定的控制效果。

3.4.2 设计思路

1. 选择合适的 API

根据所使用的仿真平台，选择适合的 API 接口。Gazebo 提供了丰富的 C++ 和 Python API，使得开发人员可以直接操作物理引擎和传感器数据。在本案例中，我们使用了图 3-5 ros2_control 框架结合 gazebo_ros2_control 插件，以实现四轮小车的高效控制。

2. 确定控制目标

明确需要实现的具体控制目标，包括但不限于：

- 1) 速度控制：维持恒定的速度或根据输入信号调整速度。
- 2) 转向控制：精确控制车轮的角度以实现转弯。
- 3) 位置控制：基于当前位置和目标位置进行导航。

3. 考虑反馈机制

引入反馈机制（如 PID 控制器）以提高控制精度。反馈机制可以通过读取传感器数据来调整输出，从而实现更稳定的控制效果。例如，在速度控制中，可以通过编码器获取当前速度并与目标速度对比，动态调整电机输出。

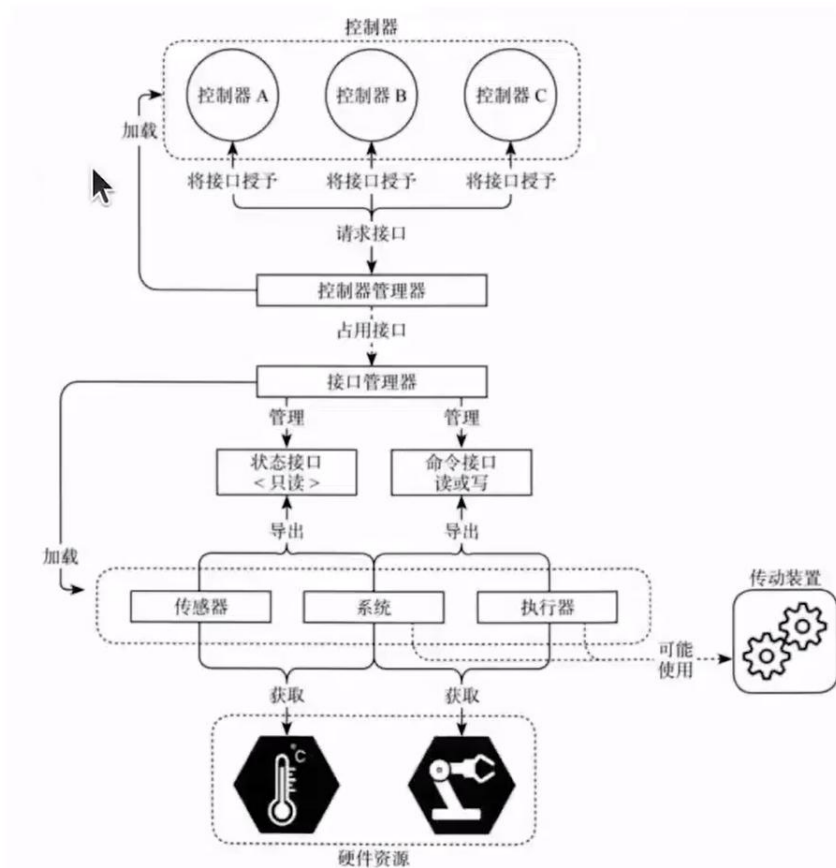


图 3-5 ros2_control 框架示意图

3.4.3 实现方法

1. 继承基类：大多数仿真平台提供了一个基类供用户继承。例如，在 Gazebo 中，你可以继承 `gazebo::SystemPlugin` 或 `gazebo::ModelPlugin`。但在本案例中，我们主要依赖 `ros2_control` 框架提供的接口。

2. 配置 ROS 2 控制系统：

1) 定义系统类型控制器：为四轮小车定义一个名为 `FirstBotGazeboSystem` 的系统类型控制器，该控制器集成了传感器和执行器的功能。

2) 指定硬件驱动插件：使用 `gazebo_ros2_control/GazeboSystem` 作为硬件驱动插件，它负责与 Gazebo 物理引擎交互，传递控制命令并接收状态反馈。

3) 设置关节接口：为每个关节（如 `left_wheel_top_joint`、`right_wheel_top_joint` 等）定义命令接口（速度和力/扭矩）以及状态接口（位置、速度、力/扭矩）。这些接口允许通过 ROS 2 发布命令并订阅状态信息，从而实现对小车运动的精确控制。

3. 加载 Gazebo 插件：

1) 指定 Gazebo 插件库：使用 libgazebo_ros2_control.so 作为 Gazebo 插件库，它实现了将 ROS 2 控制系统与 Gazebo 仿真环境桥接的功能。

2) 配置参数文件：通过配置文件路径（如图 3-6 中的 \$(find car_description)/config/firstbot_ros2_controller.yaml）加载控制器的具体参数设置。

```
<!-- 包含基础部分 -->
<xacro:include filename="$(find car_description)/urdf/firstbot/base.urdf.xacro"/>
<xacro:include filename="$(find car_description)/urdf/firstbot/firstbot_ros2_control.xacro"/>

<!-- 包含传感器部分 -->
<xacro:include filename="$(find car_description)/urdf/firstbot/sensor/imu.urdf.xacro"/>
<xacro:include filename="$(find car_description)/urdf/firstbot/sensor/camera.urdf.xacro"/>
<xacro:include filename="$(find car_description)/urdf/firstbot/sensor/laser.urdf.xacro"/>

<!-- 插件的引用 -->
<!-- <xacro:include filename="$(find car_description)/urdf/firstbot/plugins/gazebo_control_plugin.xacro"/> -->
<xacro:include filename="$(find car_description)/urdf/firstbot/plugins/gazebo_sensor_plugin.xacro"/>
```

图 3-6 配置参数文件

3) 话题重映射：通过<remapping>标签重映射话题名称，例如图 3-7 中将 /firstbot_diff_drive_controller/cmd_vel_unstamped 映射为 /cmd_vel，这有助于简化节点间通信或适配不同的命名约定。

```
<gazebo>
<!-- 这个插件里面也肯定是 ros2 的节点，然后才能加载这些参数 -->
<plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
  <parameters>$(find car_description)/config/firstbot_ros2_controller.yaml</parameters>
  <ros>
    <!-- 查找相关的话题 重映射 -->
    <remapping>/firstbot_diff_drive_controller/cmd_vel_unstamped:=/cmd_vel</remapping>
    <remapping>/firstbot_diff_drive_controller/odom:=/odom</remapping>
  </ros>
</plugin>
</gazebo>
</xacro:macro>
```

图 3-7 话题重映射

3.5 Gazebo 仿真实现

1. 创建工作空间与初始化

首先，需要创建一个新的 ROS 工作空间，并初始化它以准备后续的操作。这一步骤确保了所有相关的包和依赖项都能够被正确管理。接着，编译工作空间以确保所有包都已准备好。

2. 配置控制器插件

为了使小车能够在 Gazebo 中移动，需要配置适当的控制器插件。这通常涉及到编辑世界文件 (.world)，如图 3-8 所示，并在其中指定要使用的插件类型及其参数。对于四轮差速驱动的小车来说，常用的插件是 gazebo_ros_diff_drive，它可以处理速度命令并向 Gazebo 提供必要的接口。

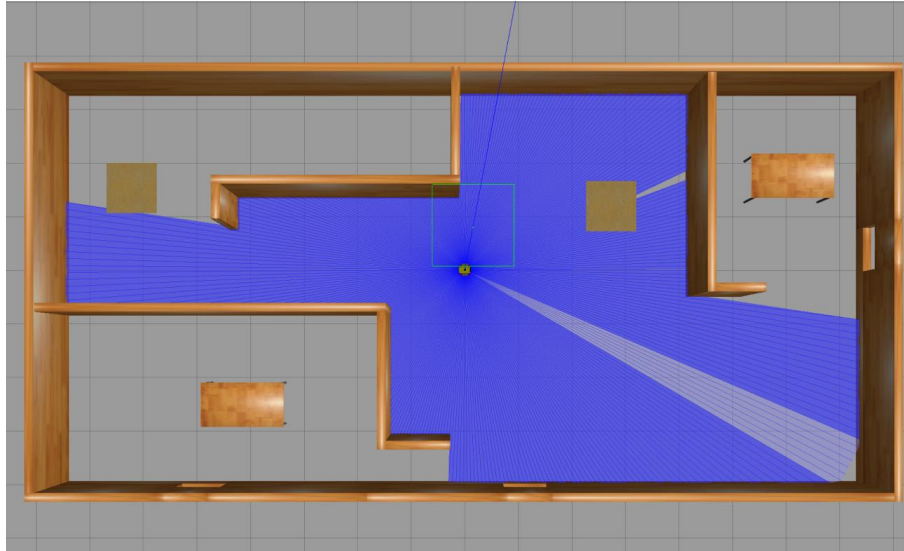


图 3-8 小车模型加入 gazebo 仿真

3. 启动 Gazebo 并显示模型

完成上述准备工作后，调用 launch 文件来启动 Gazebo，并将机器人模型加载到仿真环境中。

3.6 本章小结

本章节全面介绍了四轮小车设计和仿真的全过程，首先通过使用 XACRO 简化和模块化定义底盘、车轮、传感器等组件，并明确各部件的位置、姿态、碰撞体积、惯性参数及物理特性；接着探讨了如何在 RViz2 中正确展示四轮小车模型，包括启动 RViz2 节点、加载机器人描述以及发布关节状态。随后，针对差速驱动模型的应用，解释了其对于小车运动控制的重要性，并详细说明了控制器插件的设计思路与实现方法，强调了选择合适 API 接口、确定具体控制目标以及引入反馈机制来提升控制精度的关键步骤。最后，在 Gazebo 中进行仿真。

第 4 章 实现多节点通讯

4.1 概述

在分布式计算环境中，多节点间的数据交换与协同工作是构建高效系统的基础。本章旨在深入探讨适用于此类环境下的两种主要通信模式：基于消息（Topic）的发布-订阅机制和基于服务（Service）的请求-响应机制。我们将详细讲解这两种机制的设计原理、实现方法及其应用场景，并介绍如何通过自定义消息类型来满足特定需求，以及利用 rqt 工具和命令行接口进行通信验证与监控。

4.2 Topic 方式通讯

4.2.1 发布者实现

在基于 Topic 的通讯模型中，发布者负责向指定的主题发送数据。为了确保信息的有效传递，系统设计了 SysStatusPub 类作为发布者，它定时生成并发布温度湿度信息和速度加速度信息。发布者首先创建两个主题，一个用于发布温度湿度数据，另一个用于发布速度和加速度数据。每 5 秒，发布者会模拟一组温度湿度值，并将其打包成 TemperatureHumidity 消息格式发布出去。同时，发布者还订阅了来自控制系统的速度指令（Twist）和里程计数据（Odometry），根据这些数据实时计算车辆的速度和加速度，并将结果以 VelocityAccelerate 消息格式发布到相应的主题上。具体如下图的工作流程。

发布者的设计考虑到了系统的灵活性和可扩展性。通过采用发布-订阅模式，SysStatusPub 可以轻松地与其他多个订阅者节点建立连接，而无需考虑每个订阅者的具体位置或状态。这种方式极大地简化了系统的架构设计，同时也提高了系统的容错能力——即便某个订阅者暂时离线，也不会影响其他订阅者继续接收数据。此外，发布者使用了 ROS2 中的 `create_publisher` 函数来创建发布者对象，并设置了队列长度参数以优化消息传输效率。对于温度湿度信息的模拟，我们引入了 Python 标准库中的 `random` 模块，用以生成符合实际情况的随机数值，保证测试数据的真实性与多样性。速度和加速度信息则来源于订阅者接收到的实际传感器数据，这保证了数据的准确性和时效性。

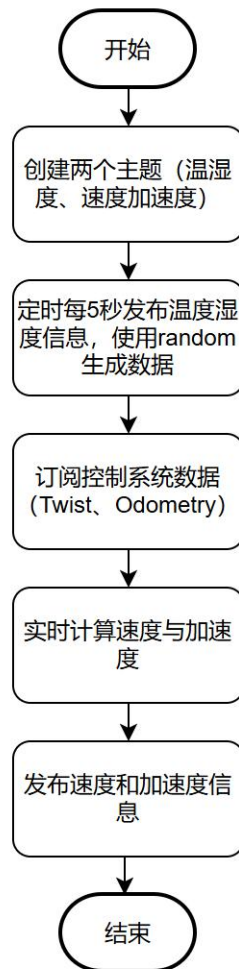


图 4-1 发布者流程图

4.2.2 订阅者实现

订阅者监听特定主题以接收并处理发布者发出的数据。在本系统中，SysStatusPub 不仅作为一个发布者，同时也充当订阅者的角色。具体如下图的工作流程。它通过监听两个关键话题来获取必要的输入数据：

一、速度指令订阅者 (/cmd_vel)：每当有新的速度指令到达时，velocity_callback 方法会被调用。此回调函数解析速度数据，并构建和发布 VelocityAccelerate 消息。此时加速度被设置为 0.0，因为加速度需要根据时间差计算，而这一信息将在里程计数据中获取。订阅者的作用在于及时捕捉到来自控制系统的新命令，并据此调整自身的操作行为，确保整个系统的协调一致。

二、里程计数据订阅者 (/odom)：当有新的里程计消息到达时，odom_callback 方法会被调用。此方法根据当前的速度和上一次记录的速度来计算加速度，并更新 VelocityAccelerate 消息中的加速度字段，然后再次发布。此外，该方法还会更新上次的速度和时间戳，以便于后续的加速度

计算。里程计数据的订阅对于精确跟踪机器人运动轨迹至关重要，它提供的连续位置和速度信息可以帮助系统更好地理解自身所处环境，以便进行相关的运动控制。

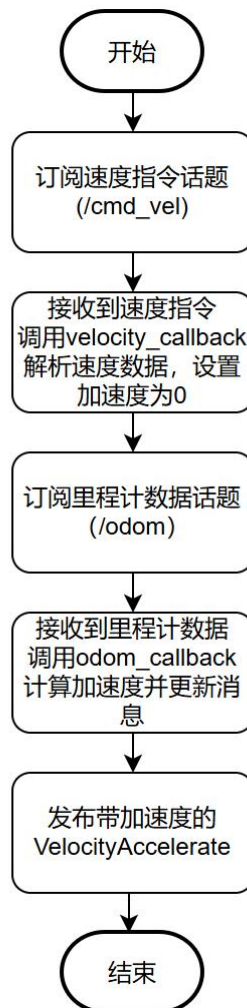


图 4-2 订阅者流程图

4.2.3 自定义消息类型

为了适应不同类型的传感器数据传输需求，同时也是为了更好地迎合现实场景。系统引入自定义消息类型，不仅大大增加了系统的灵活性，而且旨在满足特定应用场景的要求。如下图表 2 中 TemperatureHumidity 消息类型包含温度（float32）和湿度（int32）字段，用于表示环境条件。它允许系统准确描述周围环境的状况，这对于室内温控系统、农业自动化等领域尤为重要。温度和湿度是衡量环境舒适度的关键指标，也是许多工业过程控制的重要参数。通过使用 TemperatureHumidity 消息类型，我们可以确保不同设备之间能够无缝交流这些重要信息，从而提高系统的整体性能。

表 2 TemperatureHumidity 消息类型

消息名称	数据类型	说明
temperature	Float32	环境温度，单位为摄氏度
humidity	Int32	环境湿度，单位为百分比

另一方面，如下图表 3 中 VelocityAccelerate 消息类型不仅包括速度（float32）和加速度（float32），还加入了时间戳（builtin_interfaces/Time），以便于追踪数据的时间属性。这种消息结构有助于更精确地分析和理解机器人的动态行为。速度和加速度是描述物体运动状态的基本物理量，它们的变化反映了物体受力情况及其运动趋势。在移动机器人领域，实时获取和处理速度加速度信息对于路径规划、避障等任务至关重要。通过引入时间戳字段，VelocityAccelerate 消息类型还可以帮助解决不同节点间时钟同步的问题，确保所有参与者在同一时间框架内工作，避免因时间差异导致的数据偏差。

表 3 VelocityAccelerate 消息类型

消息名称	数据类型	说明
velocity	float32	小车当前的速度，单位为米每秒（m/s）
acceleration	float32	小车当前的加速度，单位为米每秒平方（m/s ² ）
timestamp	builtin_interfaces/Time	时间戳，记录数据的采集时间，以确保时钟同步

4.3 Service 方式通讯

4.3.1 服务端实现

在分布式系统中，基于服务（Service）的请求-响应机制提供了一种一对一的直接通信模式，适用于需要即时反馈和处理结果的应用场景。本段为一个名为 PoseFeedbackServer 的服务端节点，它能够接收来自客户端的位

置反馈请求，并返回相应的确认信息。服务端的设计不仅考虑到了高效的数据处理能力，还注重了系统的稳定性和可靠性。

服务端首先通过创建一个名为 PoseFeedbackServer 的类实例来启动。该类继承自 `rclcpp::Node`，并以 `pose_feedback_service` 作为节点名称。在构造函数中，服务端创建了一个服务对象 `srv_`，绑定到名为 `pose_feedback` 的服务上。这个服务使用了自定义的消息类型 `service_interfaces/srv/PoseFeedback`，它包含了位置坐标 (x 、 y)、角度 θ 以及发送方的名字等字段。服务端通过 `lambda` 表达式定义了一个回调函数 `handle_pose_feedback`，用于处理接收到的每一个请求。这一过程确保了服务端可以立即响应任何到达的请求，同时保持高效的资源利用率。服务端接收到新的请求时，会调用 `handle_pose_feedback` 方法进行处理。此方法负责解析请求中的数据，包括机器人的当前位置坐标 (x 、 y) 及其朝向角度 θ ，还有发送请求的实体名称。服务端记录这些信息的日志，以便于后续调试和分析。接下来，服务端构建响应消息 `feedback_message`，这条消息通常包含对收到数据的确认以及可能的一些附加信息或指令。

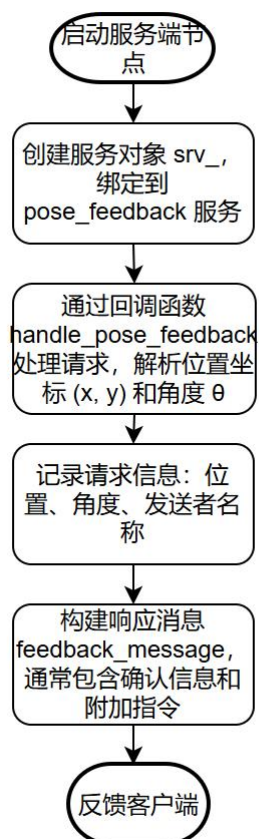


图 4-3 服务端流程图

4.3.2 客户端实现

在基于服务的请求-响应机制中，客户端负责发起请求并与服务端建立通信连接。本段将详细描述名为 PoseFeedbackClient 的客户端节点的设计与实现过程，展示它是如何通过 ROS 2 的服务接口与服务端进行交互的。

客户端同样基于 `rclcpp::Node` 类创建，命名为 `pose_feedback_client`。在构造函数中，客户端首先创建了一个服务客户端对象 `client_`，用于连接至服务端提供的 `pose_feedback` 服务。为了保证服务端已准备好接受请求，客户端会在初始化阶段循环等待，直到检测到服务端可用为止。这种方式有效地避免了因服务端未启动而导致的请求失败问题，增强了系统的鲁棒性。此外，客户端还设置了超时机制，以防止无限期等待服务端上线，从而提高了系统的响应速度。

为了提高系统的并发处理能力，客户端采用了多线程执行的方式。具体来说，客户端在一个独立的线程中运行 ROS 2 的执行器 `executor`，负责管理节点的生命周期和事件驱动逻辑。主线程则专注于周期性地检查已完成的服务调用结果，并根据需要更新系统的内部状态。这种分离式的架构使得客户端可以在不影响核心业务逻辑的前提下，高效地管理与服务端之间的通信。

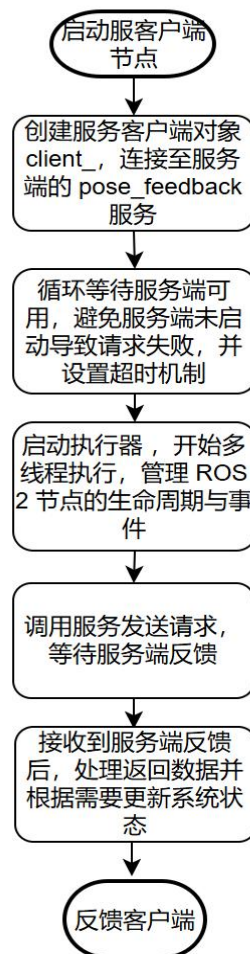


图 4-4 客户端流程图

4.3.3 自定义服务类型

为了适应不同类型的传感器数据传输需求，同时也是为了更好地迎合现实场景，系统引入了自定义消息类型 PoseFeedback，它不仅增加了系统的灵活性，而且旨在满足特定应用场景的要求。

表 4 PoseFeedback 服务类型

消息名称	数据类型	说明
x	float32	机器人当前位置的 x 坐标
y	float32	机器人当前位置的 y 坐标
Theta	float32	机器人当前的朝向角度（弧度）
Name	String	发送请求的实体名称
响应消息		
feedback_message	String	服务端对请求的反馈消息，通常为确认信息

4.4 本章小结

本章对通信部分进行了介绍。主要介绍了消息的发布-订阅机制和基于服务的请求响应机制以及相关的自定义消息和自定义服务。之后分别介绍了利用 rqt 和命令接口相关的功能测试。

第 5 章 基于 Nva2 导航功能的设计与实现

5.1 概述

在本章中，我们将重点介绍基于 ROS 2 和 Nav2 框架实现的机器人导航系统的设计与实现。导航系统是自动驾驶机器人中非常关键的组成部分，

其主要任务是使机器人能够在已知或未知的环境中自主地规划路径、避开障碍物并到达指定目标。为了实现这一目标,本系统结合了 SLAM(Simultaneous Localization and Mapping)、路径规划、动态避障等技术,通过与 ROS 2 中的 Nav2 包结合,提供了稳定和高效的导航解决方案。

5.2 SLAM 算法选择与实现

SLAM (Simultaneous Localization and Mapping) 技术是机器人导航中的核心技术之一,它能够使机器人在探索未知环境时,同时完成自我定位与地图构建的任务。在本系统中,我们选择了基于图优化的 SLAM 算法来进行建图,特别是采用了 Graph-based SLAM,它通过构建一个图来表示机器人在环境中的位姿以及与其他位姿之间的约束关系,如图 5-1 从而在后期通过优化算法修正误差,实现高精度的建图。

Graph-based SLAM 算法的基本思想是通过不断地记录机器人的位置信息(通常为位姿,包含位置与朝向),并将每一时刻的位姿作为图中的节点,通过传感器(激光雷达或相机)提供的观测信息来为这些节点之间构建边。每条边表示两次观测之间的约束关系,算法通过图优化方法来最小化所有边的误差,从而得到全局最优的地图。

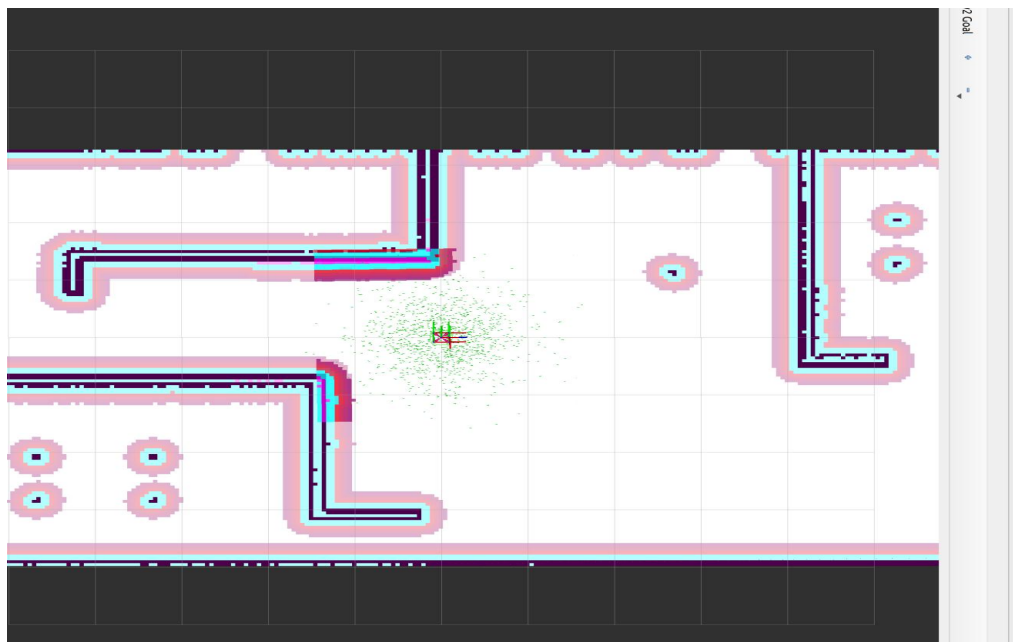


图 5-1 导航界面

在本系统中,SLAM 的实现依赖于 ROS 2 中内置的 AMCL (Adaptive Monte Carlo Localization) 模块以及激光雷达数据进行实时定位和建图。AMCL 算法通过对激光雷达数据的滤波和采样,生成一系列的粒子,并通过粒子的加权与重采样实现机器人的定位。此外,通过图优化,我们能够精确修正地图中的

偏差，生成清晰的 2D 或 3D 点云地图，为后续的路径规划和动态避障提供高质量的环境数据。

5.3 单点导航与多点导航的实现

单点导航和多点导航是机器人在不同任务需求下的常见导航模式。单点导航是指机器人从当前位姿出发，规划路径到达指定的目标点。而多点导航则是指机器人依次到达多个目标点，通常应用于巡逻或多点任务的场景。

在单点导航中，我们通过 Nav2 Basic Navigator 来控制机器人从当前位置导航到目标点。首先，导航系统会根据地图信息和当前的机器人位姿，规划出一条从当前位姿到目标点的路径。导航系统利用全局路径规划器生成一条最优路径，并根据机器人的起始位置与目标位置，确定各个路径点的顺序。接着，系统会通过控制器来跟踪路径，实时调整机器人的速度和方向，确保能够顺利到达目标。

为了实现这一过程，我们使用了 PoseStamped 消息类型来表示目标点，通过设置目标点的坐标与朝向，调用 Nav2 的 goToPose 接口来触发导航任务，该接口启动路径规划和跟踪过程。导航系统会不断向目标点进发过程中，并在此过程中监控当前状态，实时判断任务的进度。如果超过预设的最大时间或任务被用户取消，系统会自动终止导航任务，确保操作的灵活性与安全性。

多点导航的实现依赖于将多个目标点存储为一个路径点集合，机器人需要依次完成对每个目标点的导航。在该模式下，使用 followWaypoints 接口来控制机器人依次前往每一个目标点。在机器人到达某个目标点后，系统会自动将下一个目标点设为新的目标，并继续执行导航。此模式常用于需要多次路径调整或依次完成多个任务的场景，如巡逻、仓库作业等。

在实现过程中，我们通过 PoseStamped 消息类型依次定义了多个目标点，创建了一个目标点列表。机器人在导航过程中会通过反馈信息（如当前目标点编号）来判断是否到达了目标，并通过计算导航时间来决定是否需要取消任务。

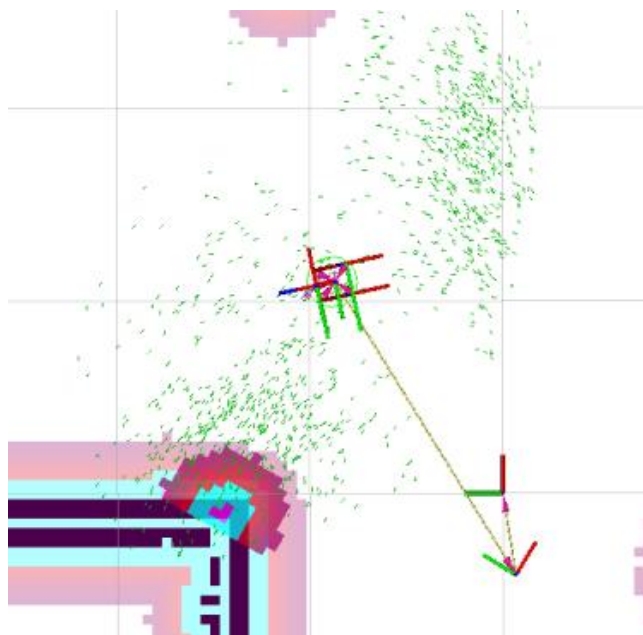


图 5-2 自主导航过程

5.4 路径规划与动态避障策略

路径规划和动态避障是小车导航系统中不可或缺的部分。在实际应用中，小车需要在复杂和动态变化的环境中规划合理的路径，并在遇到障碍物时做出合理的判断，以便能够及时避障。路径规划主要依赖于 Nav2 的 Grid-based Planner 插件。在该插件中，机器人会根据当前的地图信息以及设置的规划参数（如最大偏差、路径宽度等），通过网格地图（Grid Map）进行路径搜索。在进行路径搜索时，规划器会考虑障碍物的位置，并选择一条代价最低的路径。

Grid-based Planner 使用 A 算法来进行路径搜索，A 算法是一种广泛应用的图搜索算法，用于在已知环境中找到从起点到目标点的最短路径。其基本思想是通过启发式函数来估算从当前位置到目标位置的距离，并结合实际代价（通常是路径长度）来选择最优路径。在 A 算法中，每个节点的代价由两个部分组成：从起点到当前节点的实际代价（g 值）和从当前节点到目标点的估算代价（h 值）。总的代价（f 值）为 g 值与 h 值之和，算法的目标是通过最小化 f 值来找到最短路径。A 算法通过优先队列（通常是最小堆）来选择代价最小的节点进行扩展，直到找到目标节点或搜索区域被完全探索。为了提高路径规划的效率，我们结合了 NavfnPlanner 插件，并根据机器人的需求进行优化，确保规划出的路径能够适应复杂的环境。

动态避障引入动态避障机制以确保机器人在复杂环境中的安全性和高效性。在本系统中，我们采用了动态窗口法（Dynamic Window Approach, DWA）

与局部路径调整相结合的策略，通过实时感知和评估环境变化，动态地调整机器人的运动轨迹。DWA 算法通过在机器人当前位置附近的速度空间内采样多组候选速度（包括线速度和角速度），并根据代价函数评估每组速度的优劣，考虑因素包括目标距离、障碍物回避、机器人动力学约束等。在此基础上，系统会选取代价最小的速度组合作为运动指令，指引机器人在避障的同时尽可能快速地向目标前进。此外，系统还会定期更新局部代价地图，实时反映环境中障碍物的变化，并结合新的地图信息进一步优化机器人运动轨迹，确保机器人能够在复杂动态环境中安全、高效地执行任务。通过这种方式，系统能够在面对突发障碍物时及时调整路径，有效避免碰撞并保证导航任务的顺利完成。

5.5 导航参数优化

1. 优化导航速度

优化导航速度不仅关系到小车的速度，还涉及到速度的平稳性和稳定性。在 FollowPath 插件中的速度限制，`max_vel_x`，`max_vel_y`，`max_vel_theta` 可以进行微调，以便提升机器人的速度。

优化后的参数：

```
# DWB parameters
FollowPath:
  plugin: "dwb_core::DWBLocalPlanner"
  debug_trajectory_details: True
  min_vel_x: 0.0
  min_vel_y: 0.0
  max_vel_x: 0.26
  max_vel_y: 0.0
  max_vel_theta: 1.0
  min_speed_xy: 0.0
  max_speed_xy: 0.26
  min_speed_theta: 0.0
# Add high threshold velocity for turtlebot 3 i
```

图 5-3 FollowPath 插件参数控制

2. 优化膨胀半径

膨胀半径影响机器人的路径规划中障碍物的避免距离，优化膨胀半径可以避免小车在避障时与障碍物发生碰撞。膨胀半径会影响小车的路径规划功能，尤其是在狭小空间中。

在 InflationLayer 插件中，调整 `inflation_radius` 的参数为 0.3 以便更好地控制障碍物的膨胀范围，从而优化代价地图。

优化后的参数：

```
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLa
  cost_scaling_factor: 3.0
  inflation_radius: 0.3
voxel_layer:
  plugin: "nav2_costmap_2d::VoxelLayer"
```


图 5-4 InflationLayer 插件参数控制

3. 优化小车到点的精度

到达目标点的精度主要与目标容忍度相关,优化目标点精度可以通过调整 `xy_goal_tolerance` 和 `yaw_goal_tolerance` 来提高。

优化后的参数:

```
# xy_goal_tolerance: 0.15
# stateful: True
general_goal_checker:
  stateful: True
  plugin: "nav2_controller::SimpleGoalChecker"
  xy_goal_tolerance: 0.15
  yaw_goal_tolerance: 0.15
# DWB parameters
FollowPath:
  plugin: "dwb::DWBLocalPlanner"
```

图 5-5 优化点精度

4. 全局和路径成本地图更新频率: 可以根据实际情况调整成本地图的更新频率,以优化小车在变换环境中的响应速度。

5.6 本章小结

本章介绍了基于 ROS 2 和 Nav2 框架的机器人导航系统的设计与实现。首先,结合图优化的 SLAM 算法和 AMCL 模块,实现了高精度的地图构建与实时定位,为后续的路径规划与避障提供了环境数据。

我们分别实现了单点导航与多点导航,单点导航通过 Nav2 的 Basic Navigator 控制机器人到达目标点,而多点导航则通过 `followWaypoints` 接口完成多个目标点的导航任务。

路径规划采用了 Grid-based Planner 插件,并结合动态窗口法(DWA)进行避障,确保机器人能够安全、快速地绕过障碍物。系统通过优化路径规划策略和动态避障,保证机器人在复杂环境中的高效运行。

最后,我们对导航参数进行了优化,调整了导航速度、膨胀半径、目标精度和地图更新频率等,提升了系统性能。总体而言,本章提供了一个完整的机器人导航解决方案,涵盖了从定位到避障的各个环节。

第 6 章 可视化界面设计与实现

6.1 概述

可视化界面在本项目中起着桥梁的作用,连接了用户与四轮智能小车之间的互动。它不仅简化了复杂的机器人控制和状态监控过程,还为用户提供

了一个直观且易于操作的环境。通过图形化的界面，可以实时查看小车的状态信息、发送控制指令，并观察到即时反馈。此外，该界面还支持远程访问，允许用户从不同位置对小车进行管理和监控，极大地提升了系统的灵活性和可操作性。

6.2 用户界面设计原则

在设计用户界面时，我们遵循了几项核心原则以确保其高效性和用户体验：

1. 简洁性

尽量减少不必要的元素，保持界面清晰易懂。简洁的设计不仅能够降低用户的认知负担，还能提高他们完成任务的速度。通过去除冗余信息和简化交互流程，用户可以更快地找到所需功能，并且不会因为过多的选择而感到困惑或不知所措。

2. 一致性

统一颜色、字体以及交互方式，让用户感到熟悉并能快速适应。一致性的设计可以帮助用户建立对产品的预期，从而更容易学习和记住如何使用产品。例如，在整个应用中保持相同的按钮样式、图标风格和操作逻辑，可以使用户无需每次重新学习新的规则就能顺利完成任务。此外，一致性还有助于塑造品牌形象，增强品牌的辨识度。

3. 响应式布局

考虑到多种设备（如桌面端和移动端）的使用场景，采用灵活的布局方案以适应不同的屏幕尺寸。响应式设计允许同一个界面根据设备的不同自动调整其外观和行为，确保无论是在大屏幕电脑还是小屏幕手机上都能有良好的显示效果和用户体验。这不仅提高了产品的兼容性，也为用户提供了一致的操作体验。

4. 直观性

通过图标和文字说明结合的方式，使功能一目了然；同时提供详细的帮助提示，帮助初次使用的用户理解各项操作。直观的设计意味着用户不需要经过复杂的培训就能上手使用。设计师应该尽量让界面中的每个元素都能够自解释，即用户只需看一眼就能知道它的用途。对于那些可能不那么显而易见的功能，则可以通过工具提示、引导教程等形式给予额外的支持。

5. 安全性

对于关键性的控制命令，加入二次确认机制，防止误操作导致意外情况发生。安全性是任何用户界面设计中不可或缺的一部分，尤其是在涉及到敏

感数据处理或者可能导致不可逆后果的操作时更为重要。具体来说，我们可以采取以下措施来保障安全：

1) 明确的反馈：当用户执行某个动作后，系统应当立即给出相应的反馈，告知用户他们的操作已被接收并且正在被执行。比如，点击删除文件时，屏幕上会出现一个短暂的消息框，告诉用户“正在删除...”。

2) 撤销选项：为用户提供撤回上一步操作的能力，特别是在提交表单或更改设置之后。这样即使发生了错误，也可以轻松恢复到之前的状态。

3) 视觉区分：用醒目的颜色（如红色）标示危险操作，并附带警告图标，提醒用户注意潜在的风险。例如，“删除账户”按钮可能会被设计成红色背景加白色文字的形式，旁边还配有感叹号图标。

4) 时间延迟：某些情况下，可以在最终执行前设置一段短暂的时间间隔，给用户足够的时间思考是否真的要这么做。这段时间内，用户可以选择取消操作。

6.3 控制命令输入界面

为了方便用户向小车发送控制指令，我们使用了一个专门的控制面板。该面板不仅包含了基本的操作按钮，如前进、后退、左转、右转等，还集成了更多高级功能以满足不同层次用户的需求。以下是关于控制命令输入界面更为详细的介绍：

1. 基本操作按钮

控制面板的核心部分是一系列直观易用的操作按钮，它们被布置在界面上最显眼的位置，以使用户能够迅速定位和使用。每个按钮都配有清晰的文字标签和图标，确保即使是初次接触系统的用户也能轻松理解其含义。例如，“前进”按钮采用了一个向上的箭头图标，旁边标注有“前进”的文字说明；类似地，“左转”则配有一个指向左侧的弯曲箭头。

2. 速度调节滑块

考虑到不同的应用场景可能需要不同的行驶速度，我们在控制面板上加入了速度调节滑块。用户可以通过拖动滑块来精确调整小车的速度设定值，从最低速到最高速度范围内的任意一点都可以自由选择。此外，我们还在滑块旁边设置了几个预设的速度档位，一键即可快速切换至常用速度，极大地方便了用户的日常操作。

3. 安全措施

由于控制命令直接关系到小车的行为，因此我们在设计过程中特别重视安全性。一方面，对于一些关键性的操作（如急停），我们增加了二次确认

机制，确保不会因为误按而导致危险情况的发生；另一方面，我们也考虑到了网络传输过程中的安全问题，采用了加密通信协议保证数据的安全性。

6.4 小车状态监控界面

小车状态监控界面是整个系统中不可或缺的一部分，它为用户提供了一个全面了解小车运行状况的窗口，如图 6-1。这一界面不仅集成了丰富的图表和仪表盘来展示关键指标，还提供了地图视图以显示小车的位置及其周围的环境信息。当遇到异常情况时，系统会自动触发警报，弹出警告框提醒用户采取相应措施。这种即时反馈机制有效地提高了系统的可靠性和安全性。以下是关于小车状态监控界面更为详细的介绍：

关键指标展示：

在界面上，我们使用了多个专用区域用于呈现不同类型的键数据。例如，行驶速度、方向角度等重要参数被放置在最显眼的位置，确保用户可以一目了然地获取这些信息。每个参数旁边都有相应的单位标识，并且通过颜色编码（如绿色表示正常范围，红色表示警告或危险）帮助用户快速判断当前状态是否需要关注。

异常情况处理：

考虑到实际应用中的复杂性，我们特别加强了对异常情况的监测与响应能力。一旦发现诸如低电量、传感器故障等问题，系统将立即启动一系列预定义的操作流程。

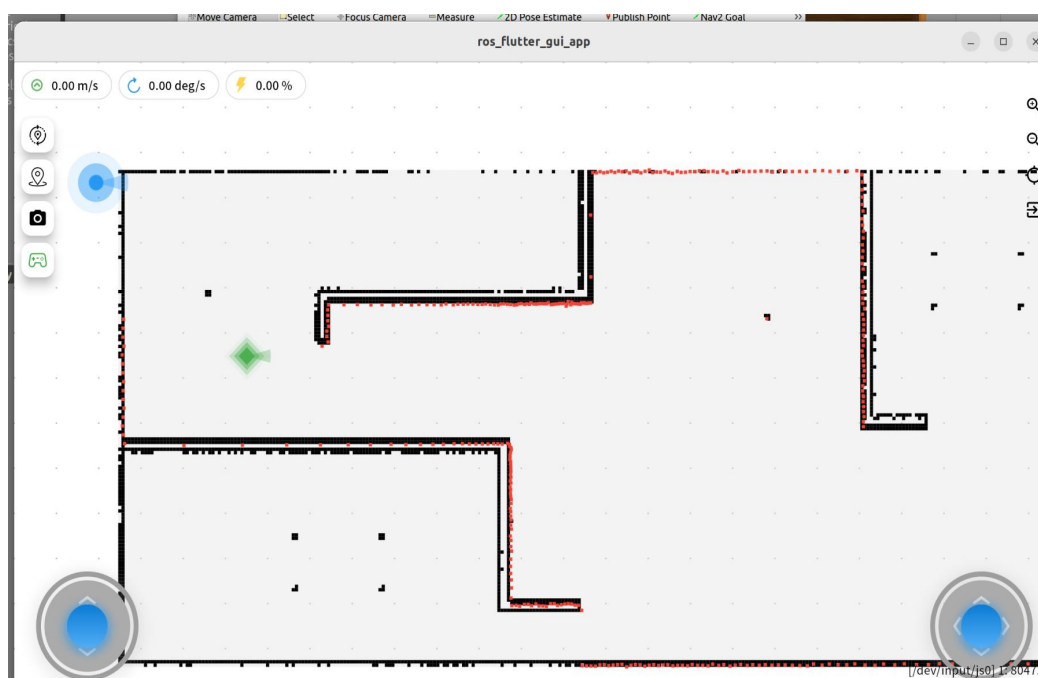


图 6-1 小车界面展示

6.5 本章小结

综上所述，通过可视化界面，我们成功地将复杂的机器人控制系统转化为一个简单易用的工具，大大降低了用户的入门门槛。与此同时，我们也注重提升用户体验，在保证功能性的同时追求美观大方的设计风格。未来的工作将继续围绕如何进一步增强界面的智能化水平展开，例如引入语音识别技术来实现更加自然的人机对话，或是探索虚拟现实 (VR) / 增强现实 (AR) 技术的应用可能性，为用户提供更为沉浸式的操作体验。

第 7 章 系统测试与分析

7.1 概述

在本章中，我们将对所开发的机器人导航系统全面的测试与分析。系统测试是软件开发过程中的重要环节，旨在验证系统各个模块的功能是否按照设计要求正常运行，确保系统在真实环境中的稳定性和可靠性。本章将介绍测试环境的搭建、功能测试的设计与执行、以及性能评估的结果，最后总结测试过程中的发现和改进措施。

7.2 测试环境搭建

测试环境搭建是系统测试的基础，目的是模拟与实际应用环境相似的条件，保证测试结果的准确性和有效性。在本系统中，测试环境包括以下几个方面：

- 硬件环境：搭建了一个具备多个传感器（如激光雷达、IMU、轮速编码器等）的移动机器人平台，并与 ROS 2 系统连接。机器人安装了执行器和传感器，用于进行路径规划、动态避障和环境感知。
- 软件环境：使用了 ROS 2 Humble 版本，并安装了相关的导航包（如 Nav2、slam_toolbox 等）以及自定义消息类型。所有测试均在本地环境中进行，确保测试与真实场景尽可能一致。
- 测试工具：使用了 ros2 测试框架和其他辅助工具（如 rviz、rqt、gazebo 模拟器等）来进行功能验证与性能分析。

7.3 功能测试

1. 四轮小车仿真设计与仿真测试：

1) gazebo 中模型加载测试: 在终端中输入 `ros2 launch car_description gazebo_sim.launch.py` 进行模型加载, 如图 7-1 所示, 各个关节加载出来。如图 7-2 也可以看到相关控制器插件也能展示出来。

```
[INFO] [launch]: All log files can be found below /home/zr/.ros/log/2024-12-25-03-29-42-482709-zr-virtual-machine-5785
[INFO] [launch]: Default logging verbosity is set to INFO
XACRO path: /home/zr/chapt7/chapt7_ws/install/car_description/share/car_description/urdf/firstbot/firstbot.urdf.xacro
[INFO] [robot_state_publisher-1]: process started with pid [5788]
[INFO] [gzserver-2]: process started with pid [5790]
[INFO] [gzclient-3]: process started with pid [5792]
[INFO] [spawn_entity.py-4]: process started with pid [5794]
[robot_state_publisher-1] [INFO] [1735068585.024222163] [robot_state_publisher]: got segment base_footprint
[robot_state_publisher-1] [INFO] [1735068585.024530993] [robot_state_publisher]: got segment base_link
[robot_state_publisher-1] [INFO] [1735068585.024568025] [robot_state_publisher]: got segment camera_link
[robot_state_publisher-1] [INFO] [1735068585.024583588] [robot_state_publisher]: got segment camera_optical_link
[robot_state_publisher-1] [INFO] [1735068585.024597083] [robot_state_publisher]: got segment imu_link
[robot_state_publisher-1] [INFO] [1735068585.024609391] [robot_state_publisher]: got segment laser_cylinder_link
[robot_state_publisher-1] [INFO] [1735068585.024622305] [robot_state_publisher]: got segment laser_link
[robot_state_publisher-1] [INFO] [1735068585.024634483] [robot_state_publisher]: got segment left_wheel_last_link
[robot_state_publisher-1] [INFO] [1735068585.024646983] [robot_state_publisher]: got segment left_wheel_top_link
[robot_state_publisher-1] [INFO] [1735068585.024658866] [robot_state_publisher]: got segment right_wheel_last_link
[robot_state_publisher-1] [INFO] [1735068585.024671187] [robot_state_publisher]: got segment right_wheel_top_link
[gzclient-3] Gazebo multi-robot simulator, version 11.10.2
[gzclient-3] Copyright (C) 2012 Open Source Robotics Foundation.
[gzclient-3] Released under the Apache 2 License.
[gzclient-3] http://gazebo.org
[gzclient-3]
[gzserver-2] Gazebo multi-robot simulator, version 11.10.2
```

图 7-1 模型加载

```
[gzserver-2] [INFO] [1735068593.646686865] [gazebo_ros2_control]: error
[gzserver-2] [INFO] [1735068593.646698798] [gazebo_ros2_control]: Command:
[gzserver-2] [INFO] [1735068593.646805366] [gazebo_ros2_control]: velocity
[gzserver-2] [INFO] [1735068593.647412622] [gazebo_ros2_control]: effort
[gzserver-2] [INFO] [1735068593.647874198] [resource_manager]: Initialize hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.648536695] [resource_manager]: Successful initialization of hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.648976968] [resource_manager]: 'configure' hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.649022593] [resource_manager]: Successful 'configure' of hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.649045269] [resource_manager]: 'activate' hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.649060426] [resource_manager]: Successful 'activate' of hardware 'FirstBotGazeboSystem'
[gzserver-2] [INFO] [1735068593.649307021] [gazebo_ros2_control]: Loading controller_manager
[gzserver-2] [INFO] [1735068593.802081385] [gazebo_ros2_control]: Loaded gazebo_ros2_control.
[gzserver-2] [INFO] [1735068593.312633389] [controller_manager]: Loading controller 'firstbot_joint_state_broadcaster'
[gzserver-2] [INFO] [1735068593.382682783] [controller_manager]: Configuring controller 'firstbot_joint_state_broadcaster'
[gzserver-2] [INFO] [1735068593.384105499] [firstbot_joint_state_broadcaster]: 'joints' or 'interfaces' parameter is empty. All available state interfa
[ros2-5] Successfully loaded controller firstbot_joint_state_broadcaster into state active
```

图 7-2 控制器插件加载

2) 使用控制命令 `ros2 run teleop_twist_keyboard teleop_twist_keyboard` 操作四轮小车, 测试小车是否能够转向, 在 gazebo 的具体展示:

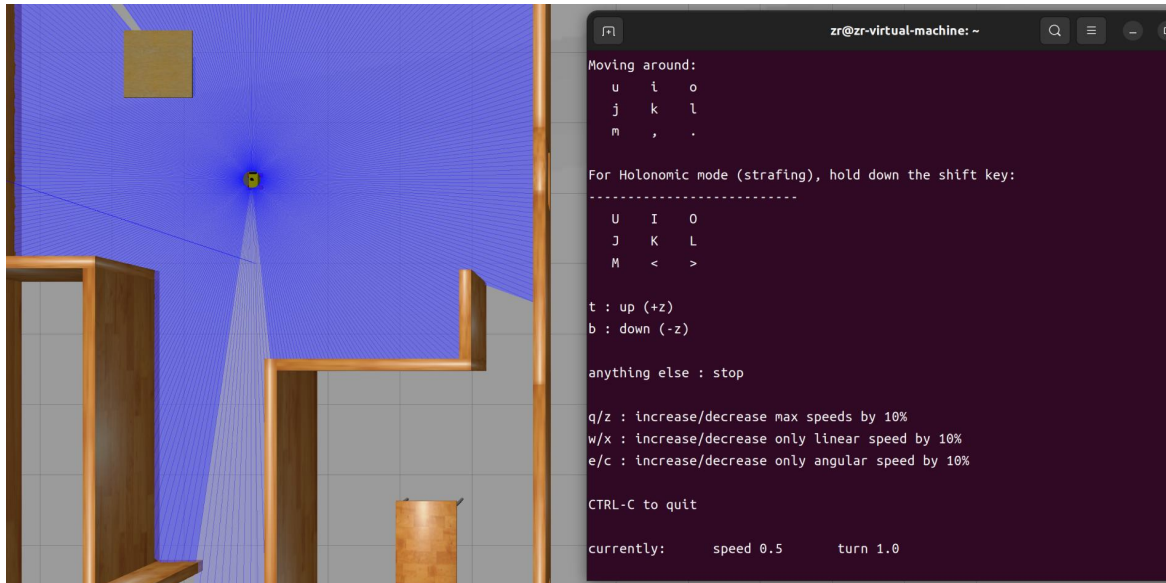


图 7-3 gazebo 小车通过命令行运行

首先查明代码中的控制小车转向相关 twist 的设置：

```

30
31
32
33
34
35 # Diff drive controller (仍然使用速度控制，四轮差速模型)
36 firstbot_diff_drive_controller:
37
38   ros_parameters:
39     left_wheel_names: ["left_wheel_top_joint", "left_wheel_last_joint"] # 确保关节名称正确
40     right_wheel_names: ["right_wheel_top_joint", "right_wheel_last_joint"] # 确保关节名称正确
41
42     wheel_separation: 0.17
43     wheel_radius: 0.032
44
45     wheel_separation_multiplier: 1.0
46     left_wheel_radius_multiplier: 1.0
47     right_wheel_radius_multiplier: 1.0
48
49     publish_rate: 50.0
50     odom_frame_id: odom
51     base_frame_id: base_footprint
52     pose_covariance_diagonal: [0.001, 0.001, 0.0, 0.0, 0.0, 0.01]
53     twist_covariance_diagonal: [0.001, 0.0, 0.0, 0.0, 0.0, 0.01]
54
55     open_loop: true
56     enable_odom_tf: true
57
58     cmd_vel_timeout: 0.5
59     use_stamped_vel: false
60     use_velocity_cmd: true # 确保使用速度命令
61

```

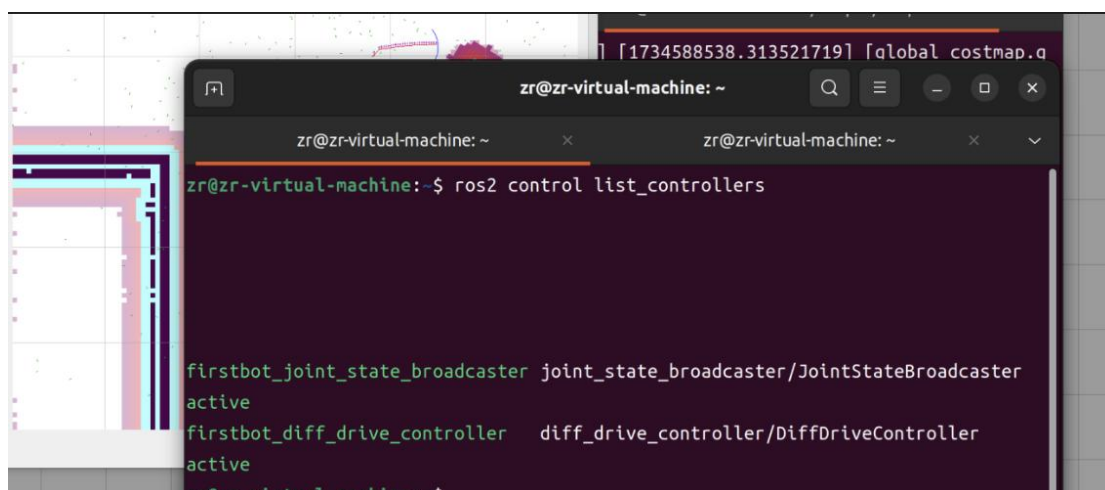
图 7-4 代码中控制小车转向部分

监听/cmd 话题中的消息内容，可以看到是有参数运行的

```
---
linear:
  x: 0.24631578947368427
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.84
---
linear:
  x: 0.21894736842105267
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: -0.6799999999999999
---
linear:
  x: 0.21894736842105267
  y: 0.0
  z: 0.0
```

图 7-5 /cmd 话题的监听

检查控制器是否正常运行，如图 7-6.



```
zr@zr-virtual-machine: ~  
zr@zr-virtual-machine:~$ ros2 control list_controllers  
  
firstbot_joint_state_broadcaster joint_state_broadcaster/JointStateBroadcaster  
active  
firstbot_diff_drive_controller diff_drive_controller/DiffDriveController  
active
```

图 7-6 控制器的运行

具体在 rqt 中的 TF 中各节点都能正确显示：

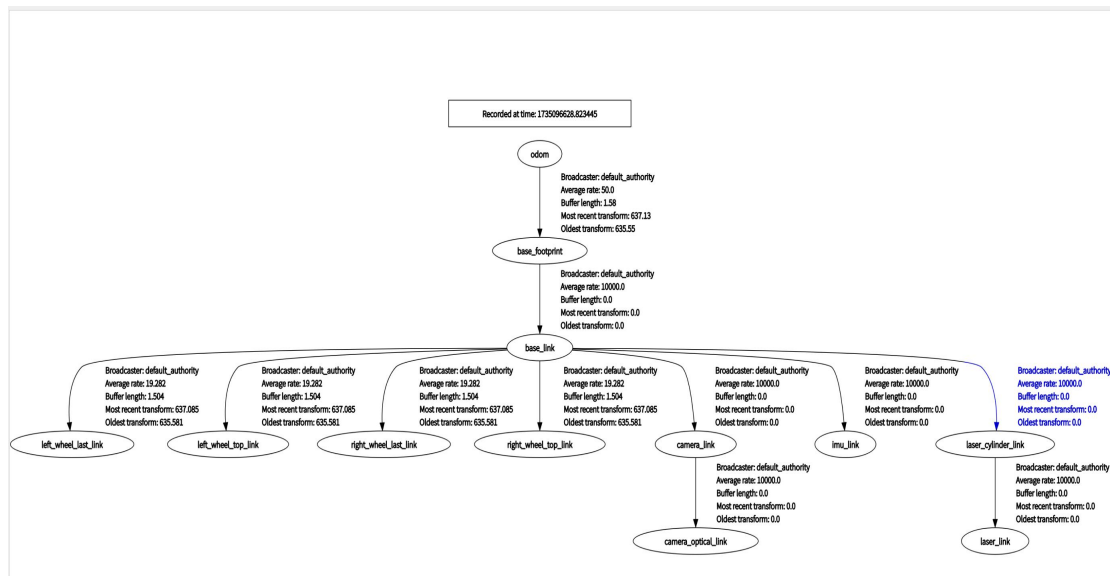


图 7-7 各节点的控制状况

2. 多节点通讯测试：

1) 使用指令 `ros2 run status_publisher sys_status_pub` 来运行自定义消息 `status_publisher.py` 文件。观察到是有数据的，因为二者的发布频率不同，所以导致二者的数据量不同，如图 7-8，此时小车是停止状态。

```
[INFO] [1735098317.144068475] [sys_status_publisher]: Calculated acceleration: 0.000000
[INFO] [1735098317.192019991] [sys_status_publisher]: Calculated acceleration: 0.000000
[INFO] [1735098317.226422362] [sys_status_publisher]: Calculated acceleration: 0.000000
[INFO] [1735098317.288278428] [sys_status_publisher]: Publishing temperature=58.40314311378495, humidity=15
[INFO] [1735098317.328212577] [sys_status_publisher]: Calculated acceleration: 0.000000
[INFO] [1735098317.357511666] [sys_status_publisher]: Calculated acceleration: 0.000000
[INFO] [1735098317.432951681] [sys_status_publisher]: Calculated acceleration: 0.000000
```

图 7-8 停止状态的小车

2) 使用 `ros2 run need_cpp_service service_server` 来运行自定义服务内容 `need_cpp_service.py` 文件，如图 7-9 显示正在运行中。

```
on the given context, at ./src/rcl/rcl.c:241
[ros2run]: Process exited with failure 1
zr@zr-virtual-machine:~/parameter_cpp_pkg/parameter_ws$ ros2 run need_cpp_service service_server
```

图 7-9 服务端正在运行

使用 `ros2 run need_cpp_service_client service_client` 命令启动客户端，如图 7-10 所显示，定时发布消息 `x`, `y`, `theta`。

```
Summary: 4 packages finished [11.9s]
zr@zr-virtual-machine:~/parameter_cpp_pkg/parameter_ws$ source install/setup.bash
zr@zr-virtual-machine:~/parameter_cpp_pkg/parameter_ws$ ros2 run need_cpp_service service_client

[INFO] [1734606765.192911215] [pose_feedback_client]: Sending request: x=8.87, y=2.55, theta=-0.17, name=Robot1
[INFO] [1734606765.234482654] [pose_feedback_client]: Received response: Received pose from Robot1: x=8.866695, y=2.554875, theta=-0.168765
[INFO] [1734606770.192703695] [pose_feedback_client]: Sending request: x=8.54, y=1.50, theta=-2.63, name=Robot1
[INFO] [1734606770.273658056] [pose_feedback_client]: Received response: Received pose from Robot1: x=8.542959, y=1.497986, theta=-2.629970
```

图 7-10 客户端定时发布消息

3) 使用 `rat` 查看相关消息定义及内容以及相关的消息测试界面:

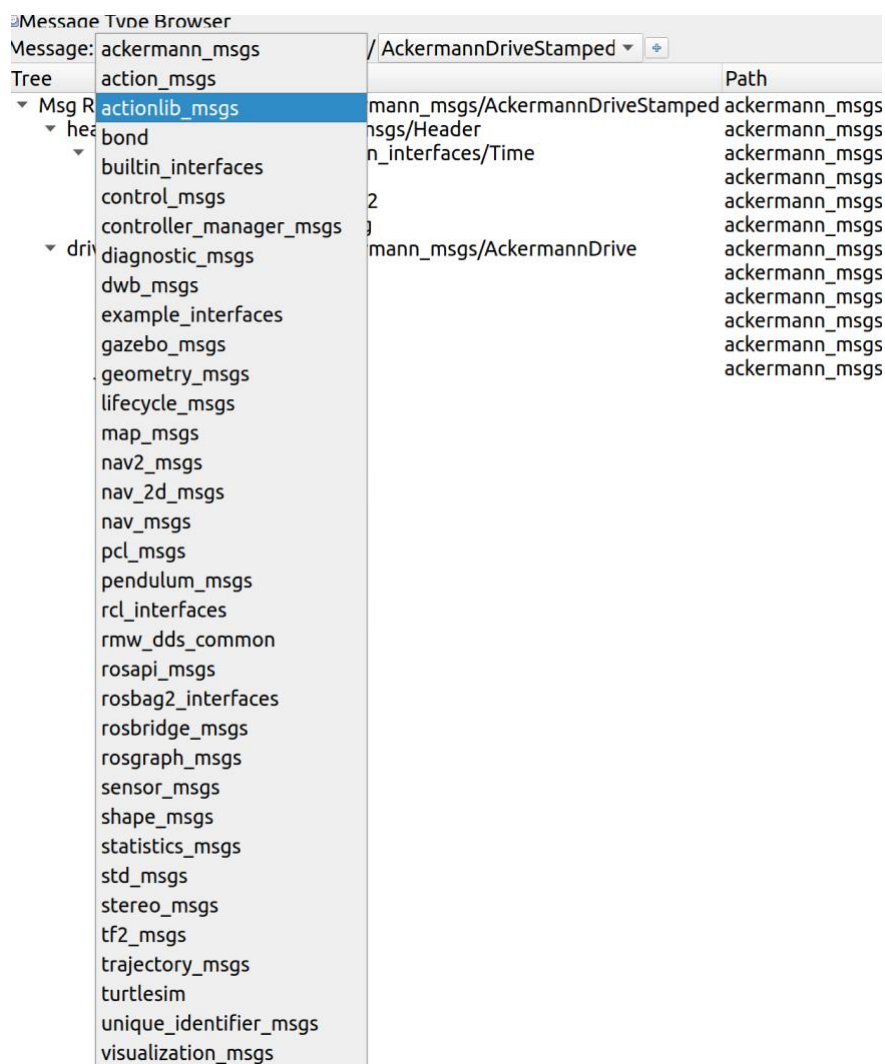


图 7-11 消息的所有内容

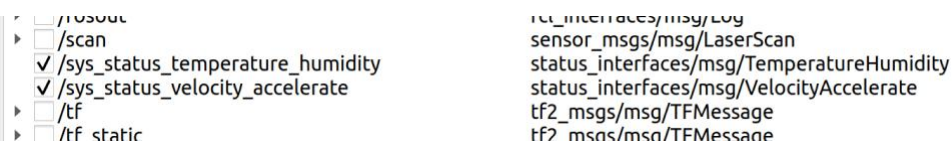


图 7-11 消息测试界面

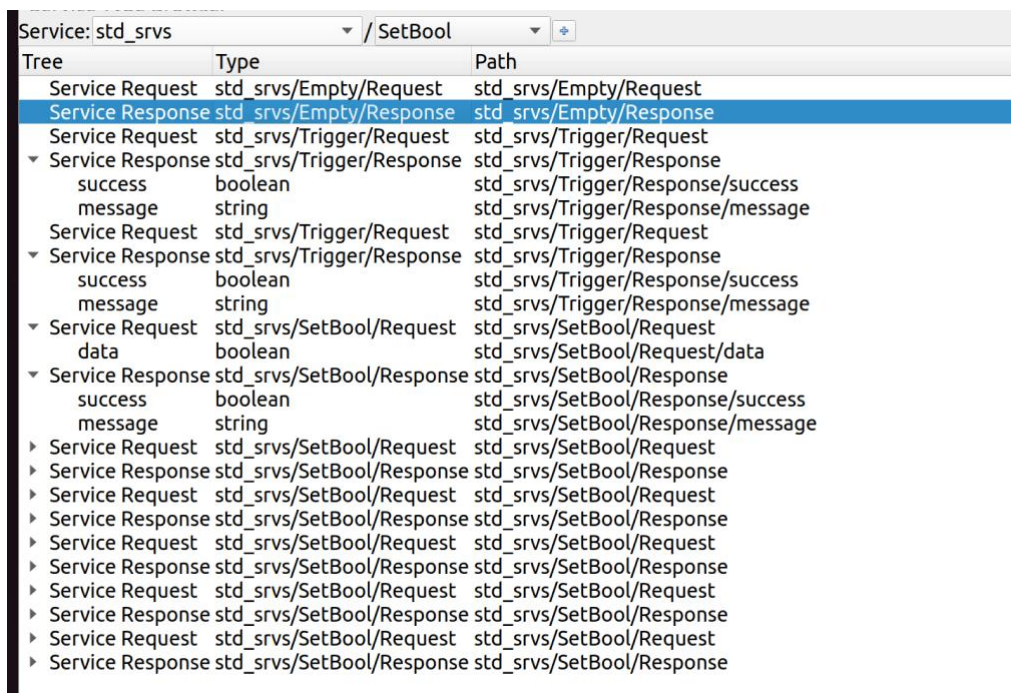


图 7-12 服务测试界面

3. 导航测试：

1) 使用导航 `ros2 run firstbot_navigation2 navigation2.launch.py` 命令进行导航节点的启动：

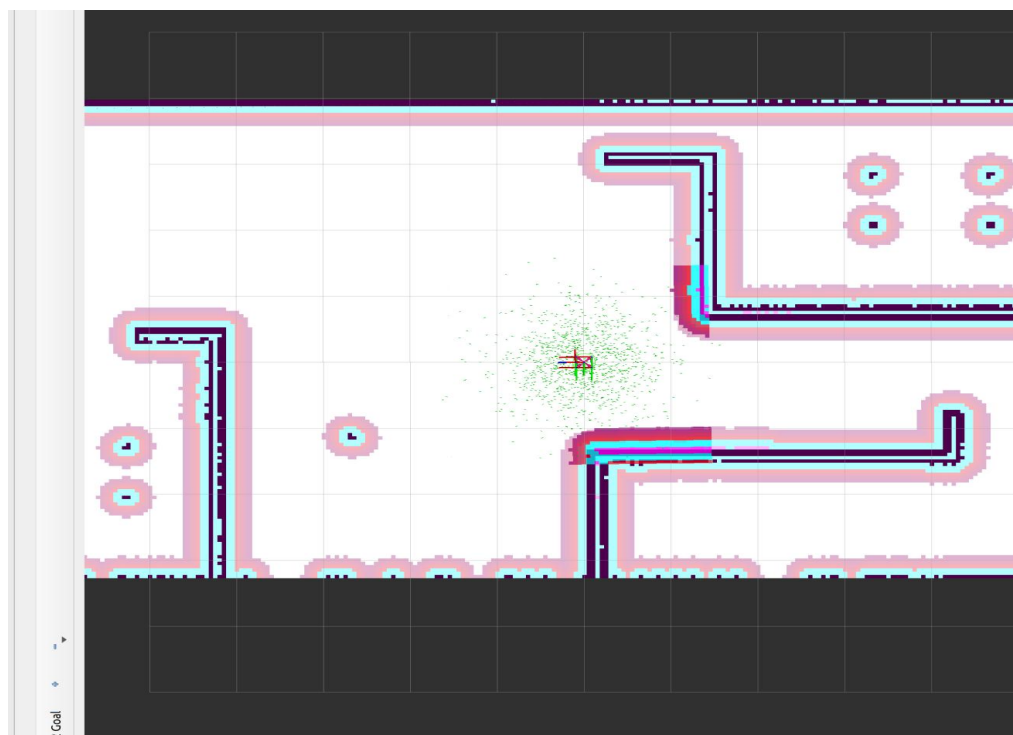


图 7-13 导航界面

2) 导航消息的发布如图 7-14:

```
zr@zr-virtual-machine:~/chapt7/chapt7_ws$ ros2 topic list
/ancel/transition_event
/ancel_pose
/behavior_server/transition_event
/behavior_tree_log
/bond
/bt_navigator/transition_event
/camera_sensor/camera_info
/camera_sensor/depth/camera_info
/camera_sensor/depth/image_raw
/camera_sensor/image_raw
/camera_sensor/points
/clicked_point
/clock
/cnd_vel
/cnd_vel_nav
/cnd_vel_teleop
/controller_server/transition_event
/cost_cloud
/diagnostics
/downsampled_costmap
/downsampled_costmap_updates
/dynamic_joint_states
/evaluation
/firstbot_diff_drive_controller/transition_event
/firstbot_joint_state_broadcaster/transition_event
/global_costmap/costmap
/global_costmap/costmap_raw
/global_costmap/costmap_updates
/global_costmap/footprint
/global_costmap/global_costmap/transition_event
/global_costmap/published_footprint
/global_costmap/voxel_marked_cloud
/goal_pose
/luu
/initialpose
/joint_states
/local_costmap/clearing_endpoints
/local_costmap/costmap
/local_costmap/costmap_raw
/local_costmap/costmap_updates
/local_costmap/footprint
/local_costmap/local_costmap/transition_event
/local_costmap/published_footprint
/local_costmap/voxel_grid
/local_costmap/voxel_marked_cloud
/local_plan
/map
/map_server/transition_event
/map_updates
/marker
/mobile_base/sensors/bumper_pointcloud
/odom
/parameter_events
/particle_cloud
/performance_metrics
/plan
/plan_smoothed
/planner_server/transition_event
/preempt_teleop
/received_global_plan
/robot_description
/rosout
/scan
/smoother_server/transition_event
/speed_limit
/tf
/tf_static
/transformed_global_plan
/velocity_smoother/transition_event
/waypoint_follower/transition_event
/waypoints
zr@zr-virtual-machine:~/chapt7/chapt7_ws$
```

图 7-14 相关消息的发布

3) 导航相关的服务发布, 如图 7-15:


```

/amcl/change_state
/amcl/describe_parameters
/amcl/get_available_states
/amcl/get_available_transitions
/amcl/get_parameter_types
/amcl/get_parameters
/amcl/get_state
/amcl/get_transition_graph
/amcl/list_parameters
/amcl/set_parameters
/amcl/set_parameters_atomically
/apply_joint_effort
/apply_link_wrench
/behavior_server/change_state
/behavior_server/describe_parameters
/behavior_server/get_available_states
/behavior_server/get_available_transitions
/behavior_server/get_parameter_types
/behavior_server/get_parameters
/behavior_server/get_state
/behavior_server/get_transition_graph
/behavior_server/list_parameters
/behavior_server/set_parameters
/behavior_server/set_parameters_atomically
/bt_navigator/change_state
/bt_navigator/describe_parameters
/bt_navigator/get_available_states
/bt_navigator/get_available_transitions
/bt_navigator/get_parameter_types
/bt_navigator/get_parameters
/bt_navigator/get_state
/bt_navigator/get_transition_graph
/bt_navigator/list_parameters
/bt_navigator/set_parameters
/bt_navigator/set_parameters_atomically
/bt_navigator_navigate_through_poses_rclcpp_node/describe_parameters
/bt_navigator_navigate_through_poses_rclcpp_node/get_parameter_types
/bt_navigator_navigate_through_poses_rclcpp_node/get_parameters
/bt_navigator_navigate_through_poses_rclcpp_node/list_parameters
/bt_navigator_navigate_through_poses_rclcpp_node/set_parameters
/bt_navigator_navigate_through_poses_rclcpp_node/set_parameters_atomically
/bt_navigator_navigate_to_pose_rclcpp_node/describe_parameters
/bt_navigator_navigate_to_pose_rclcpp_node/get_parameter_types
/bt_navigator_navigate_to_pose_rclcpp_node/get_parameters
/bt_navigator_navigate_to_pose_rclcpp_node/list_parameters
/bt_navigator_navigate_to_pose_rclcpp_node/set_parameters
/bt_navigator_navigate_to_pose_rclcpp_node/set_parameters_atomically
/clear_joint_efforts
/clear_link_wrenches
/controller_manager/configure_controller

```

图 7-15 相关服务的发布

4) 可以看到自主导航的过程，如图 7-16：

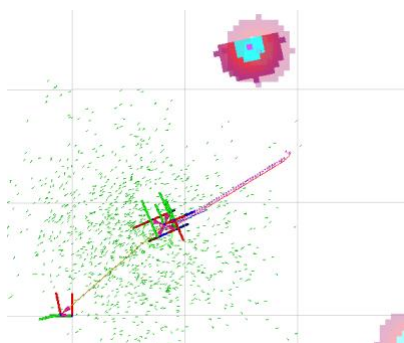


图 7-16 自主导航的过程

5) 此时通信的情况正常，如 7-17 所示：

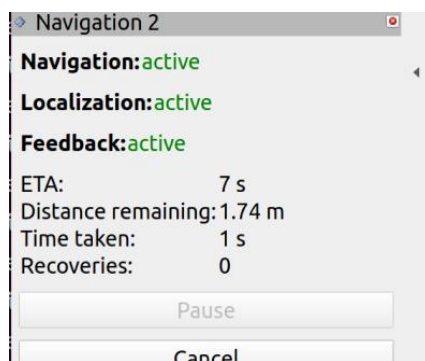


图 7-17 导航通信界面展示

6) 定点导航：在启动小车节点和导航节点的同时，还要启动定点导航的 python 文件，`ros2 run firstbot_application nav_to_pose`，如图 7-18 所示。

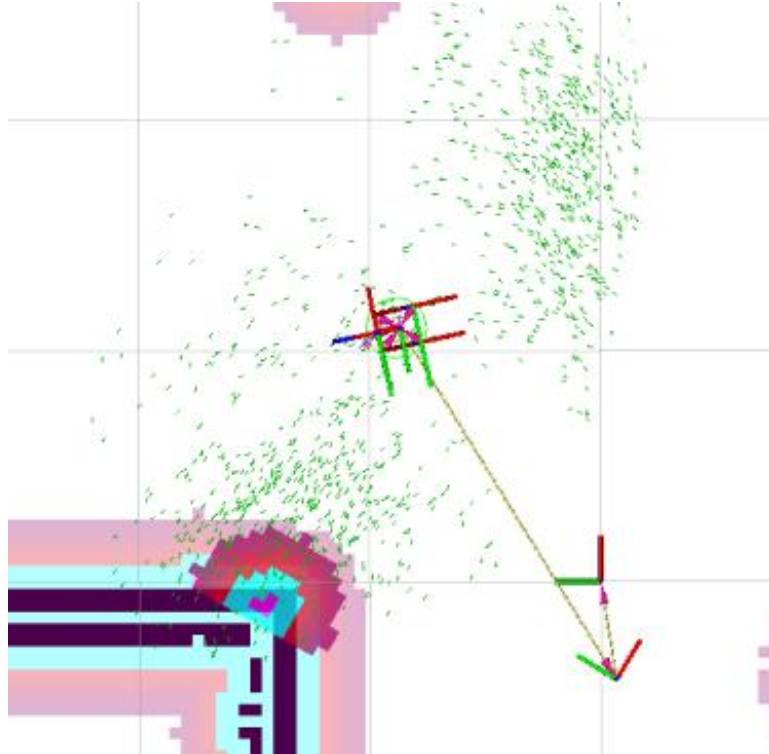


图 7-18 定点导航测试

4. 界面测试：

1) 首先搭建简单服务平台，如图 7-19 所示：

```
“python” 命令来自 Debian 软件包 python-is-python3
zr@zr-virtual-machine:~$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

图 7-19 简单服务平台的搭建

2) 然后开启 socket 通信，如 7-20 所示：

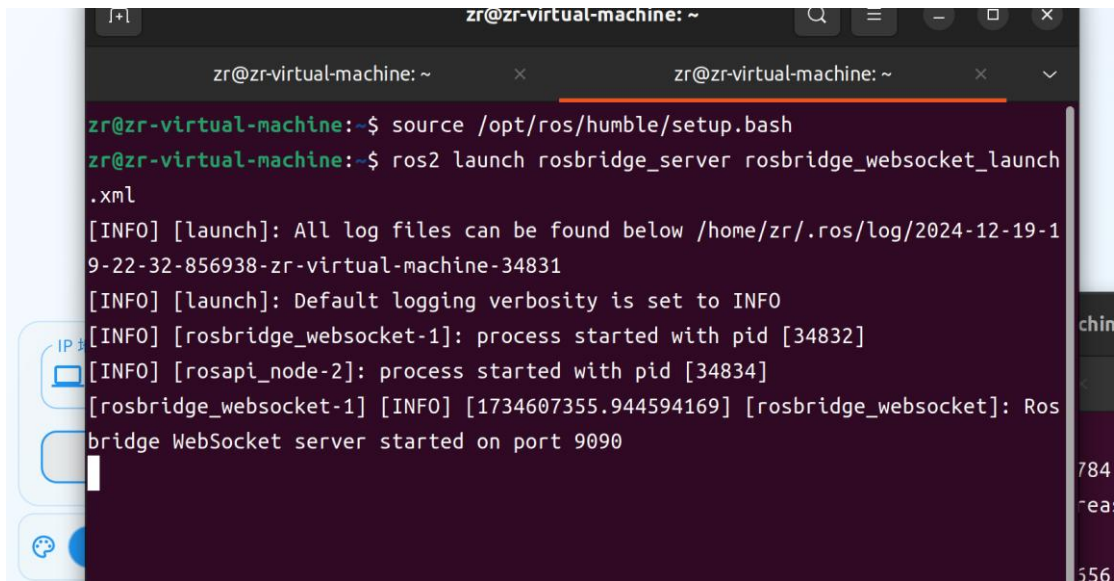


图 7-20 开启 Socket 通信

3) 界面的最终展示, 如图 7-21 所示:

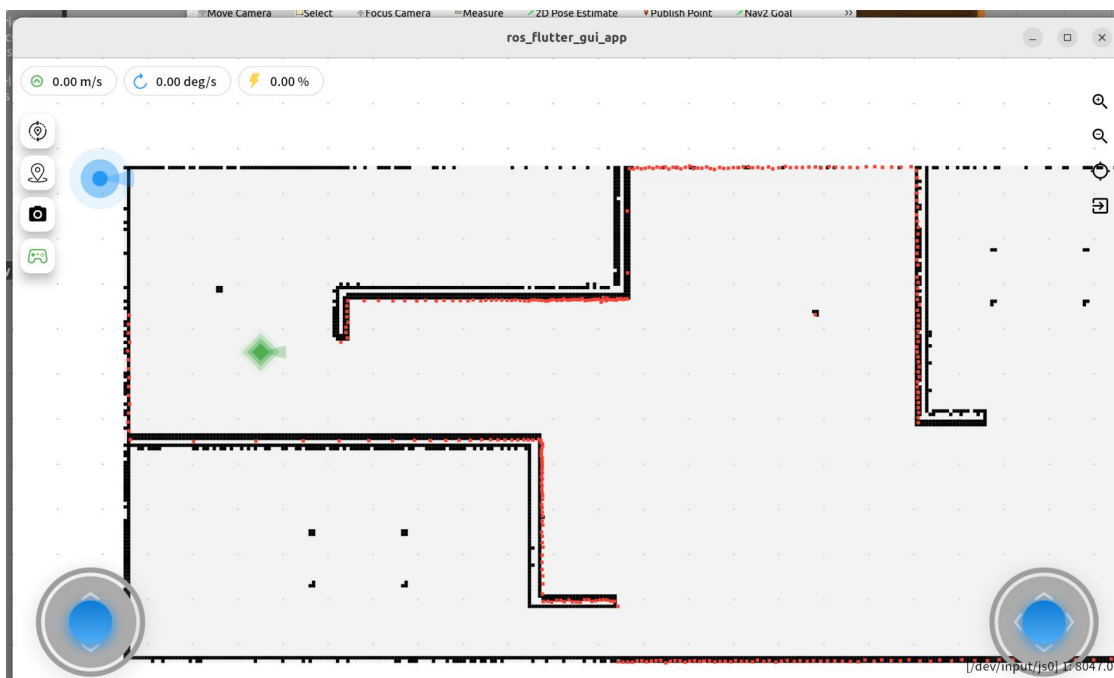


图 7-21 界面展示

7.4 本章小结

本章对系统的测试过程进行了全面的分析和总结。通过搭建真实与仿真相结合的测试环境, 验证了机器人导航系统的功能与性能。首先, 在硬件和软件环境的支持下, 成功完成了四轮小车的模型加载与运动测试, 验证了小车的运动控制、话题通信及控制器的正确性。其次, 通过多节点的通讯测试, 验证了自定义消息和服务的功能及其在复杂场景中的稳定性。功能测试显示, 系统能

够正确发布和接收消息，服务端与客户端的交互逻辑清晰，确保了数据通信的可靠性。此外，导航功能测试表明，小车可以实现自主导航及定点导航，系统在路径规划、动态避障等关键任务中表现出较高的稳定性和准确性。在界面测试中，系统通过 Socket 通信实现了服务平台与用户界面的交互，进一步提升了系统的实用性和用户体验。本次测试充分验证了系统的设计目标和功能要求，为系统的进一步优化提供了可靠依据，同时也展示了该系统在真实环境中的应用潜力。

第 8 章 总结与展望

8.1 总结

在此次基于 ROS 2 的四轮小车系统开发项目中，团队成功实现了多节点通讯、精确控制和自主导航功能，为机器人技术的发展提供了宝贵的实践经验和技術积累。

1. 四轮小车建模与仿真。团队通过构建四轮小车的 URDF 模型、应用差速驱动模型、设计控制器插件及增强 Gazebo 仿真的视觉反馈，展示了从物理模型到高级控制逻辑的全面实现。

2. 多节点通讯模块。实现了基于 Topic 的消息发布与订阅机制及基于 Service 的定时通讯功能，确保数据流的及时准确传递。使用 rqt 工具验证了通讯机制的有效性，并通过命令行工具检查节点和服务信息。

3. 导航功能的设计与实现：团队结合 SLAM (Simultaneous Localization and Mapping)、路径规划、动态避障等技术，通过与 ROS 2 中的 Nav2 包结合，提供了稳定和高效的导航解决方案。实现了单点导航和多点导航功能，使机器人能够在已知或未知环境中自主规划路径、避开障碍物并到达指定目标。通过 Graph-based SLAM 算法进行高精度建图，并利用全局路径规划器生成最优路径，优化相关参数，确保机器人能够顺利到达目标位置。

4. 界面的设计与实现：主导了主界面 (HomePage) 和路径显示页面 (PathDisplayPage) 的设计与实现，确保这些界面不仅美观而且功能完善。通过引入 provider 包进行状态管理，确保所有 UI 组件都能监听并响应最新的机器人状态变化。

8.2 展望

随着项目的完成，团队不仅积累了丰富的实践经验，也为未来的研究和發展奠定了坚实的基础。接下来，可以从以下几个方面继续探索和改进，以推动机器人技术的进一步发展：

1. 智能化提升：集成先进感知、规划与控制算法

1) 深度学习与强化学习的应用：进一步集成先进的感知、规划和控制算法，如深度学习和强化学习，以提高机器人的智能水平和适应能力。特别是利用预训练大模型对机器人进行预训练，让其具备更强的学习能力。例如，在视觉识别任务中应用卷积神经网络（CNN），在决策过程中采用强化学习算法来优化路径选择和动作执行。

2) 计算机视觉增强：通过 OpenCV 等库增强 Gazebo 仿真的视觉反馈与调试^[2]，确保仿真环境中的图像处理更加贴近真实世界。这将有助于开发更复杂的视觉任务，如目标检测、跟踪、分类以及场景理解。结合 ROS 2 平台，可以实现从摄像头输入到行为决策的完整闭环系统，使机器人能够在动态环境中做出更快更准确的反应。

2. 用户交互体验：引入新技术改善人机交互

1) 语音识别与自然语言处理：考虑引入语音识别技术和自然语言处理（NLP）框架，如 TensorFlow Speech Recognition 或 Microsoft Azure Cognitive Services，为用户提供更为自然的语言交流方式。机器人可以通过对话理解用户的意图，并据此调整自身的行为模式，实现真正的双向互动。

2) 虚拟现实（VR）/增强现实（AR）：借助 VR/AR 技术创造沉浸式操作体验，让用户仿佛置身于机器人所处的实际环境中，直观地观察其运作情况，并通过手势或语音指令直接控制机器人的行动。这种新型的人机交互方式不仅提升了用户体验，也为远程协作带来了新的可能性。

参考文献

[1]Tchoń K, Ratajczak J. General Lagrange-type Jacobian Inverse for Nonholonomic Robotic Systems[J]. IEEE Transactions on Robotics,2017, 34(1): 256-263

[2]张伟杰. 揭秘机械臂视觉抓取中的 OpenCV: 图像处理与物体识别的 10 大技巧[EB/OL]. (2024-08-07) [2024-12-25]. <https://wenku.csdn.net/column/81ucqcf1c8>.