

ROS 机器人语音识别程序设计

摘 要

语音识别技术是人工智能领域的关键技术之一，它将人类的语音转换为可理解的文本，实现了人机交互。语音识别过程包括声音采集、预处理、特征提取、声学 and 语言模型匹配以及后处理等步骤。语音识别技术在机器人领域十分重要，它不仅能提升机器人的智能化水平，还能改善用户的使用体验。结合 ROS（Robot Operating System）机器人操作系统的模块化架构，开发者可以方便地将语音识别模块整合到机器人系统中，实现语音到文本的转换并传递信息，从而完成闭环交互。

本文基于 ROS 机器人操作系统，使用科大讯飞开放平台的离线命令词识别的软件开发工具包，设计一个离线语音识别程序。在此基础上，将语音识别与机器人控制结合，设计一个巡视小车 URDF 模型，并设计一个小车控制程序。由语音识别程序识别输入的语音，再由控制程序接收语音识别结果并发送控制指令，采用 `ros_control` 控制框架，最终在 Gazebo 仿真中实现语音控制巡视小车车体的运动和摄像机的转动。

语音识别测试与小车运动控制仿真结果表明，本文设计的语音识别程序和小车控制程序能准确地识别控制指令并控制巡视小车做出指定的动作，有着很好的控制效果。

本文设计的语音控制系统可以直接搭建在如树莓派、香橙派一类的单板计算机上，以实现语音控制实体机器人。与在线语音识别相比，本文采用的离线命令词识别具有识别范围限定和无需连接互联网等特点，故硬件要求上相对较低，识别速度更快，实现成本更少，与当下主流的采用联网进行语音交互的实现方案相比更有优势，有着很好的应用前景。

关键词：ROS；语音识别；命令词识别；机器人控制

目 录

1 绪论	1
1.1 研究背景及意义	1
1.2 国内外发展状况	2
1.2.1 国外语音识别发展状况	2
1.2.2 国内语音识别发展状况	2
2 语音识别技术概述	4
2.1 语音识别技术的概念	4
2.2 语音识别的基本原理	4
2.3 语音识别系统的分类	5
2.4 语音识别技术算法	6
2.5 语音识别功能包	6
2.6 命令词识别	6
2.6.1 巴科斯范式语法	7
3 ROS 机器人操作系统	8
3.1 ROS 架构	8
3.2 通信	10
3.2.1 通信机制	10
3.2.2 话题与服务的区别	12
3.3 机器人建模与仿真	12
3.4 ros_control 框架	15
3.5 Gazebo 物理仿真	17
4 语音控制小车运动程序设计	18
4.1 总设计	18
4.2 ROS 开发环境搭建	18
4.3 巡视小车模型设计	19
4.3.1 创建小车 URDF 模型	19
4.3.2 在 Rviz 中显示小车模型	21
4.4 Gazebo 仿真环境设计	22
4.4.1 构建仿真环境	22
4.5 语音识别程序设计	23

4.6 控制程序设计.....	25
4.7 小车模型添加 Gazebo 仿真属性.....	26
4.7.1 为 link 添加<gazebo>标签.....	27
4.7.2 为 joint 添加传动系统.....	27
4.7.3 添加关节约束.....	27
4.8 在 Gazebo 中显示小车模型.....	27
4.9 配置控制器.....	28
4.10 添加 Gazebo 控制器插件.....	29
4.11 添加 Gazebo 传感器仿真.....	30
4.11.1 2D 摄像机.....	30
4.11.2 RGB-D 摄像机.....	30
5 系统整体测试.....	31
5.1 命令行识别准确度测试.....	31
5.2 小车运动控制测试.....	32
5.3 摄像机图像显示.....	33
5.4 总体控制效果.....	33
5.5 总体通信数据流.....	34
6 总结.....	35
参考文献.....	36

1 绪论

1.1 研究背景及意义

语音识别技术，作为人工智能领域的一个重要分支，近年来随着深度学习等技术的发展，已经取得了显著的进步。这项技术使得机器能够将人类的语音转换为计算机可理解的文本，进而实现与人类的自然交互。在机器人领域，语音识别技术的应用尤为广泛，它不仅极大地提升了机器人的交互体验，也使得机器人能够更好地服务于人类社会。语音识别通常包括声音采集、声学信号预处理、特征提取、声学模型匹配、语言模型匹配和后处理几个关键步骤。

机器人产业的发展已经成为衡量一个国家科技实力和工业竞争力的重要尺度。它不仅反映了一个国家的科技创新能力，还体现了其在高端制造业领域的专业水平。目前，机器人产业正以蓬勃的势头发展，机器人在仓储物流、商品制造、安防巡查、医疗康复、矿产采集和国防军事等多个领域得到了广泛的应用，极大地改变了人类的生产和生活方式，为社会经济的发展注入了强大的动力^[1]。

在机器人领域，语音识别技术的应用可以显著地提升机器人的智能化水平。例如，在服务机器人中，通过语音识别技术，机器人能够理解顾客的指令和需求，提供更加个性化的服务。在家庭机器人中，语音识别技术使得家庭成员能够通过自然语言与机器人进行交流，如控制家电、查询信息等。此外，在工业机器人中，语音识别技术也可以用于简化操作流程，提高生产效率。

机器人操作系统（Robot Operating System，简称 ROS）是一个灵活的框架，专为开发机器人软件而设计。它提供了一系列类似于传统操作系统的功能，包括硬件抽象、设备底层控制、进程间通信以及功能包管理等。它也提供了一系列工具和库，可以帮助软件开发人员创建复杂且健壮的机器人应用程序^[2]。

结合语音识别与 ROS 机器人系统，可以为机器人提供一个强大的交互功能。利用 ROS 的模块化架构，开发者可以轻松地将语音识别功能设计为一个模块，集成到现有的机器人系统中。这个模块负责处理语音信号，将其转换为文本，并利用 ROS 的通信机制，将这些文本信息传递给机器人的其他部分或者其他机器人，如导航、任务规划、用户界面或群体机器人协同工作。例如，一个服务机器人可以使用语音识别来接收顾客的指令，然后通过 ROS 的话题发布机制将这些指令传递给相应的处理模块。同时，机器人的状态和反馈也可以通过语音合成（Text-to-Speech, TTS）技术以语音的形式提供给用户，形成一个闭环的交互系统。此外，ROS 的插件式设计允许开发者根据项目需求定制语音识别系统。开发者可以选择适合特定应用场景的声学 and 语言模型，或者根据特定环境（如嘈杂的工厂或多变的家庭环境）调整语音识别算法的参数。通过 ROS 与语音识别技术的结合，机器人不仅能够

更好地理解人类的指令，还能够提供更加自然和直观的交互体验。这种结合为服务机器人、家庭助理、探索机器人以及教育和娱乐机器人等多种应用领域开辟了新的可能性。语音识别技术作为连接人类与机器人的桥梁，其发展对于推动机器人智能化具有重要意义。

1.2 国内外发展状况

1.2.1 国外语音识别发展状况

语音识别技术的起源可以追溯到 20 世纪 50 年代，当时 AT&T 贝尔实验室开发的 Audy 系统成功实现了对 10 个英文数字的识别，这成为了语音识别历史上的一个重要起点。

60 年代到 70 年代，在计算机的推动作用下，语音识别开始快速发展。动态规划和线性预测分析技术很好地解决了语音信号产生模型和语音信号不等长的问题。矢量量化(VQ)和隐马尔可夫模型(HMM)理论的出现为语音识别领域带来了突破性进展，使得基于线性预测倒谱和动态时间规整技术的特定人孤立词语音识别系统成为可能^[3]。

80 年代，语音识别领域产生了巨大突破，一方面，诸如多级动态规划语音识别算法的连接词语音识别算法被开发；另一方面，像隐马尔可夫模型(HMM)这种基于统计学建立的语音识别系统提高了连续语音识别系统的实现可能性。隐马尔可夫模型和人工神经网络(ANN)成功得到应用^[4]。

在 90 年代之后，人工神经网络技术的发展为语音识别领域开辟了新的路径。这种技术具有自适应性、并行性、非线性处理、鲁棒性、容错性以及学习能力等显著特点，在语音识别的模型设计、参数提取、优化算法以及系统的自适应性方面取得了突破性进展，使语音识别技术变得更加成熟，语音识别系统也从研究阶段走向了实际应用。

国外许多科技公司研发出了比较出名的语音识别产品，比如微软的 Cortana、苹果的 Siri、OpenAI 的 Whisper 等等，这些产品不仅在语音识别的准确率上达到了高水平，更通过与各种应用操作的紧密结合，极大地丰富了用户的交互体验。

1.2.2 国内语音识别发展状况

中国的语音识别技术是在 20 世纪 50 年代后期由中国科学院发展起来的。最初的研究重点是学习研究外国的理论和技术。70 年代末，中国科学院声学研究所建立了基于模板匹配的孤立词语音识别系统。80 年代后期，中国科学院声学所、自动化研究所和北京大学共同推进“863”计划中的大词汇量语音识别技术研究^[5]。

随后，国内许多高校和重点研究机构也开始了对语音识别技术的研究，并取得了显著的科研成果。中国科学院自动化研究所开发的非特定人连续语音听写系统和汉语语音人机对话系统的单字识别准确率超过 90%。到了 90 年代中期，中国的语音识别技术取得了巨大的进步，研究水平已经与国际先进水平齐平^[6]。

近几年来，国内在声学模型上取得了可观的进步，例如阿里巴巴达摩院机器智能实验

室语音识别团队的 Deep-FSMN、百度的 SMLTA、中国科学技术大学和科大讯飞等合作的 USTC-iFlytek Systems^[7]。

另外，以语音识别技术为基础的智能语音产品在近几年发展迅猛。例如，科大讯飞的“讯飞听见”可以将会议发言实时转换成文字记录，极大提升了会议记录效率。在教育领域，其智能语音评测系统能够对学生的发音进行评估和指导，广泛应用于语言学习和考试评分中。智能个人助理产品，如小米的“小爱同学”和百度的“小度”，利用语音识别技术，可以方便地进行信息查询、设备控制和日常任务管理，这些智能设备不仅提高了生活的便捷性，也推动了家居自动化的发展。在汽车行业，许多国产汽车品牌如吉利、比亚迪、问界，都在其车载系统中集成了语音识别功能，驾驶员可以通过语音指令进行导航、播放音乐、拨打电话以及自动泊车，提高了驾驶安全性和便捷性。

总体来看，国内的语音识别技术已经取得了显著成就，在提升工作效率、改善用户体验和推动行业发展方面发挥着越来越重要的作用。

2 语音识别技术概述

2.1 语音识别技术的概念

语音识别技术是一种让计算机能够理解和处理人类语音的技术。它的核心功能是将语音信号转换为文本数据，使得机器能够理解人类的口头指令，并据此执行相应的操作。作为一门跨学科的技术，语音识别不仅关注语音信号的处理，也是模式识别领域的一部分，它综合了信号处理、语言学、心理学以及人工智能等多个学科的研究成果，并考虑了人说话时的体态语言，如面部表情和手势等非声学因素。语音识别技术的终极目标是实现人与机器之间流畅、自然的交互^[8]。

语音识别可以分为以下两大步骤，如图 2-1 所示。

第一步是“学习”。根据所要构建的语音识别系统的需求，选择一个合适的识别算法。从语音数据中提取出关键的特征参数，这些参数能够代表语音信号的独特属性。提取出的特征随后被用来构建一个标准模式，作为识别过程中的参考基准。

第二步是“识别”。在实际应用中，根据需要再次提取语音信号的特征参数，并形成待识别的测试模板。随后系统将这些测试模板与之前建立的参考模板进行逐一比较，通过特定的算法计算它们之间的相似度或距离。最后利用内置的决策机制，如专家系统知识库，从所有比较结果中选择与测试模板最匹配的参考模板，匹配结果即为系统的识别输出。

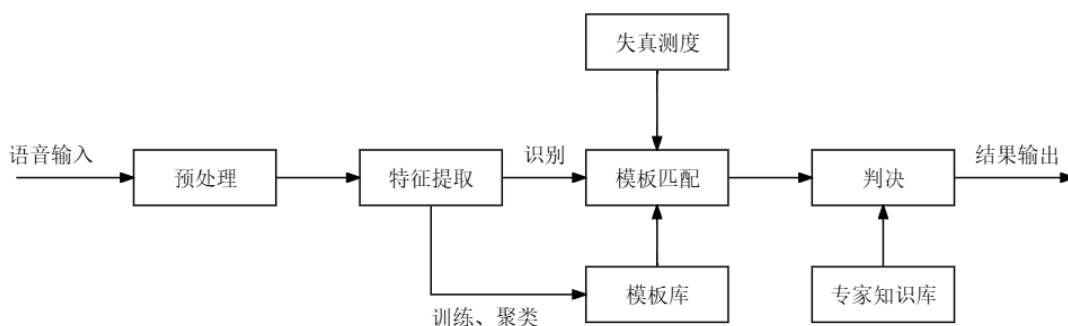


图 2-1 语音识别算法流程图

Fig. 2-1 Flowchart of speech recognition algorithm

2.2 语音识别的基本原理

语音识别系统的核心理论基础是统计模式识别，其主要目标是将输入的语音信号转换成相应的文本信息。具体而言，就是将一系列语音特征向量 $X = X_1, X_2, \dots, X_T$ 转化成词序列 $W = W_1, W_2, \dots, W_N$ 并输出。基于最大后验概率的语音识别模型如下式所示^[9]。

$$\begin{aligned}
\hat{W} &= \arg \max_w \{P(W | X)\} \\
&= \arg \max_w \left\{ \frac{P(X | W)P(W)}{P(X)} \right\} \\
&= \arg \max_w \{P(X | W)P(W)\} \\
&= \arg \max_w \{\log P(X | W) + \lambda \log P(W)\}
\end{aligned} \tag{2-1}$$

由式（2-1）可以看出，若要找出最符合语音信号的词序列 \hat{W} ，则应使 $P(X|W)$ 与 $P(W)$ 相乘得到的值达到最大。其中，声学模型决定特征矢量序列 X 在给定 W 条件下的条件概率 $P(X|W)$ ，语言模型决定 W 独立于语音特征矢量的先验概率 $P(W)$ 。 $\log P(X|W)$ 与 $\log P(W)$ 分别表示声学与语言的分值。 λ 为平衡声学模型与语言模型的权重。由于 W 的选取不受对数转换的影响，故第四个等式成立。

声学模型将声音信号转化为相应的音素序列，而语言模型则将这些音素序列转化为文字序列^[10]。

2.3 语音识别系统的分类

语音识别技术可以根据不同的研究目标将其细分为几个主要领域，根据领域的不同有如下三个分类方式。

（1）词汇量规模：根据可识别词汇的数量，语音识别系统可以分为大词汇量、中等词汇量和小词汇量三类。大词汇量系统涵盖了 500 个以上的词汇，中等词汇量系统则包含 100 到 500 个词汇，小词汇量系统则在 10 到 100 个之间。词汇量的增加意味着系统需要处理更多的语音变体和上下文信息，增加了模型的复杂度和计算负担，从而导致识别准确率有所降低。

（2）发音方式：根据处理的语音类型，语音识别系统可以分为孤立词识别、连接词识别、连续语音识别和关键词识别。孤立词识别处理的是单个独立的音节、字或词，要求输入的语音也是孤立的。连续语音识别则处理的是连续的自然语音，程序更加复杂，计算量大，对硬件性能要求高。连接词识别则介于前面二者之间，要求输入的语音的音节之间有明显的间隔。关键词识别则是从连续的语音中提取出与预设模板库匹配的关键词，忽略其他语音部分。

（3）说话人识别：根据识别系统对说话人的要求，语音识别系统可以分为特定人语音识别系统和非特定人语音识别系统。其中，特定人系统需要针对特定说话人进行大量的训练，只能识别该特定说话人的声音，适用于个人化的语音识别场景。非特定人系统则通过分析大量不同说话人的语音，提取共有的语音特征进行识别，能够适应不同说话人的声音，应用范围更广。但由于不同说话人之间存在语速、发音习惯、发音音量等众多语言差异，其开发难度也非常大。

这些分类方式有助于理解语音识别技术的多样性和复杂性，以及在不同应用场景下的

适用性和挑战^[11,12]。

2.4 语音识别技术算法

当今语音识别技术的主流算法有很多，包括动态时间规整算法、基于非参数模型的矢量量化方法、基于参数模型的隐马尔可夫模型和基于人工神经网络等^[13]。

（1）动态时间规整(DTW)

通过调整未知语音信号的时间尺度，使其与已知的语音模板匹配。DTW 算法在连续语音识别中是一种常用方法，特别是在小词汇量识别系统和孤立字词识别系统中。

（2）矢量量化(VQ)

一种信号压缩技术，它将语音信号的样点或参数映射到一个 k 维空间中的矢量，然后通过量化过程将这些矢量映射到有限数量的代表值（码字）。在小词汇量和孤立词的语音识别中，矢量量化方法有着很好的适用性。

（3）隐马尔可夫模型(HMM)

一种统计模型，它使用 Markov 链来模拟语音信号的统计特性变化，同时将每个状态与观测序列相关联。因其在描述语音信号的统计特性方面的独特优势，故而在语音识别领域中被广泛认为是非常理想的模型。

（4）人工神经网络(ANN)

通过模拟人脑的机制来处理信息，具有学习和理解的能力。在人工神经网络中，每个神经元执行特定的计算任务，而网络对各种输入模式的响应则由神经元之间连接的权重所决定。通过训练算法，人工神经网络能够调整这些权重，从而提高语音识别的准确性^[8,14]。

2.5 语音识别功能包

ROS 中集成了 CMU Sphinx 和 Festival 开源项目，推出了独立的语音识别功能包——pocketsphinx，可以实现语音识别的功能。该功能包不仅可以识别英文语音，还可以在使用中文语音引擎和模型的条件识别中文。

科大讯飞作为中国领先的智能语音服务供应商，在中文语音识别技术领域保持着显著的领先地位。该公司通过其开放平台，提供了丰富的、成熟的语音识别软件开发工具包（SDK）供开发者使用。

考虑到 pocketsphinx 功能包的中文识别效果并不理想，本文选择使用科大讯飞的中文语音识别 SDK，以实现中文的准确识别。

2.6 命令词识别

命令词识别，亦称为关键词检测，是语音识别的一个细分方向。此技术是基于既定的

语法规则，将符合这些规则的自然语言语音信号转换为文本格式。其主要目标是从输入的语音信号中准确辨识出用户预设的特定关键词。关键词检测技术的发展与整个语音识别领域的进步紧密相连，而命令词识别则可视作关键词检测技术在特定应用场景下的深化与细化。命令词往往简短精炼，直接表达用户的意图。由于识别结果的范围只在语法规则文件所列出的规则里，故有很好的识别准确度^[15,16]。

本文选择使用科大讯飞平台离线命令词的软件开发工具包进行语音识别程序的设计。

2.6.1 巴科斯范式语法

由上文的介绍可知，命令词的识别需要一个语法规则文件。语法作为语音识别系统的输入之一，是语音识别不可或缺的条件。巴科斯范式语法（BNF）则是一种描述语音识别的语法。语法规则文件被编译成语音识别网络后被送往语音识别器，语音识别器分析并提取语音信号的特征信息，在已构建的识别网络上进行路径匹配，进而识别出语音内容^[17,18]。

例如，设计一个简单的语音打电话程序，则可以定义一个语法如下：

```
#BNF+IAT 1.0;
!grammar call;
!slot <name>;
!start <commands>;
<commands>:(联系|打电话给)<name>;
<name>:张三|李四;
```

该语法支持以下语句：

- a. “联系张三”；
- b. “打电话给张三”；
- c. “联系李四”；
- d. “打电话给李四”。

上述四个语句中的任意一句话，都能被识别系统识别出来。如果语句不在上述范围以内，识别系统则拒绝识别。

3 ROS 机器人操作系统

随着硬件技术的飞速发展，机器人领域的进步和复杂化正在不断加速。新型传感器、执行器、处理器以及通信设备的不断涌现，为机器人系统提供了前所未有的功能和性能。这些硬件多样性和丰富性的提升对机器人系统的软件开发提出了更高的要求，特别是软件代码的复用性和模块化，然而现有的机器人系统已经无法满足当下的与需求。与硬件开发相比，软件的开发显得力不从心。为了解决当下软件开发面临的难题，全球各地的开发人员接连加入了机器人通用软件框架的研发工作中。机器人操作系统 ROS 便是近几年出现的一个优秀的软件框架。

ROS (Robot Operating System) 是一个用于开发机器人软件的灵活开源框架，它提供用于编译、获取、编写和跨计算机运行代码所需的工具、库函数和协议，具有类似操作系统所提供的功能，包括硬件抽象描述、底层驱动程序管理、常见函数的实现、共用功能的执行、程序间的消息传递以及功能包的管理，这对机器人平台下繁杂的任务构建与稳定的行为控制进行了极大地简化^[2]。

ROS 目前主要支持 Ubuntu 操作系统，同时也可以运行在 OSX、Android、Debian、Windows 等系统上。截至 2020 年，ROS 已经发布了 13 个版本。

3.1 ROS 架构

ROS 架构具有三个层次，分别为 OS 层、中间层和应用层，如图 3-1 所示。

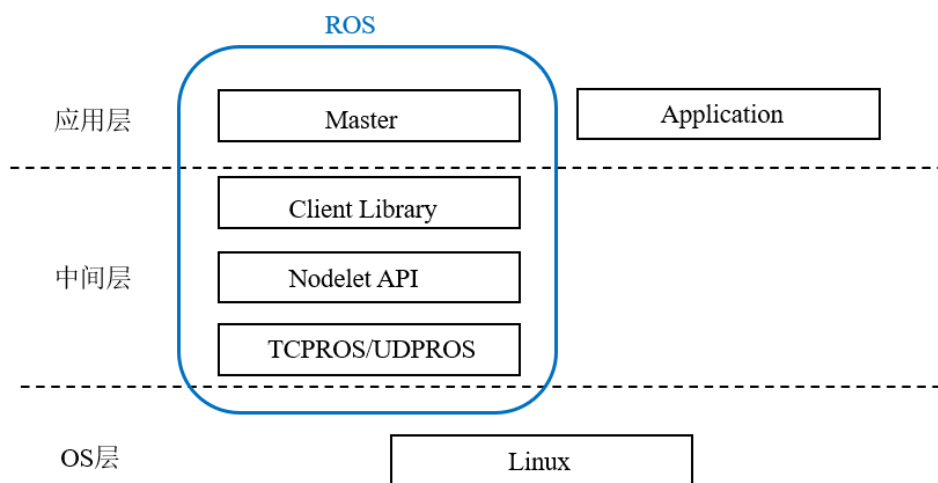


图 3-1 ROS 架构

Fig. 3-1 ROS Architecture

(1) OS 层

ROS 不是一个传统意义上的操作系统，需要在 Linux 系统上运行，无法直接在计算机硬件上运行。故可以使用 Ubuntu 操作系统运行 ROS，它是 ROS 官方支持性最高的系统。

(2) 中间层

在 Linux 系统中，尽管没有专门针对机器人开发的中间件，但 ROS 作为中间层框架，起到了至关重要的作用，特别是在实现复杂通信机制方面。ROS 通信系统基于 TCP/IP 和 UDP/IP 网络协议栈进行封装，构建了 TCPROS 和 UDPROS 协议，为节点间的通信提供了高效、可靠的数据传输能力。

此外，ROS 还为应用层提供了众多与机器人开发相关的库，包括数据类型定义、路径规划、坐标变换、物理仿真和运动控制等。

(3) 应用层

在 ROS 的应用层，存在一个核心协调者，名为 Master，它的职责是确保整个系统的顺畅运作。从系统实现机制的角度分析，ROS 的架构可以被划分为三个层次，如图 3-2 所示，包括计算图、文件系统以及一个活跃的开源社区。在这个架构中，定义了 ROS 的一系列基础而重要的组件，包括节点、消息、话题、服务、功能包、元功能包等。

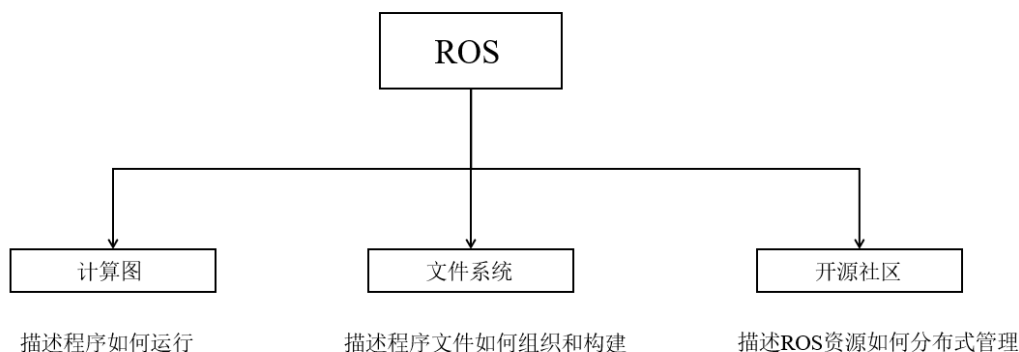


图 3-2 ROS 的三个层次（从系统实现角度）

Fig. 3-2 Three levels of ROS (System implementation perspective)

(1) 节点

节点（Node）是一些执行运算任务的进程，一个系统通常由多个节点构成。节点的存在让系统的运行与管理更加直观易懂。当多个节点同时运行时，端对端的通信关系可以绘制成如图 3-3 所示的节点关系图，图中的节点是正在运行的进程，而端对端的连接关系就是节点之间的连线。

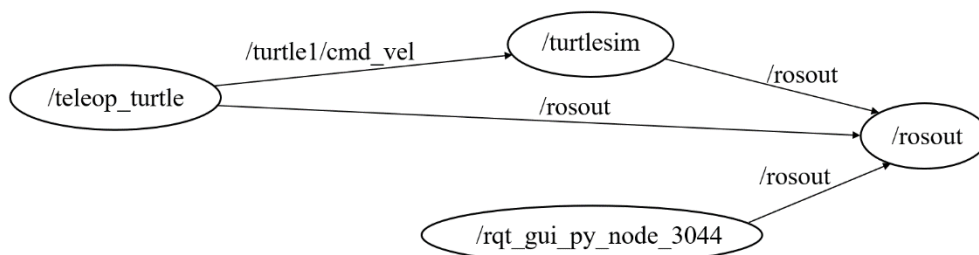


图 3-3 ROS 中的节点关系

Fig. 3-3 Node Relationships in ROS

（2）文件系统

文件系统类似于操作系统的机构，ROS 将所有文件按照一定的规则进行分类存储，不同功能的文件被分类放在不同的目录中，如图 3-4 所示。

功能包 (Package): ROS 软件中的基本单元，包含 ROS 节点、库、配置文件等。

功能包清单(PackageManifest): 每个功能包都含有一个名为 `package.xml` 的配置文件，该文件用于记录功能包的基本信息，如作者信息、许可信息、依赖的功能包、编译参数等。

元功能包 (MetaPackage): 用来组织多个用于同一目的的功能包。

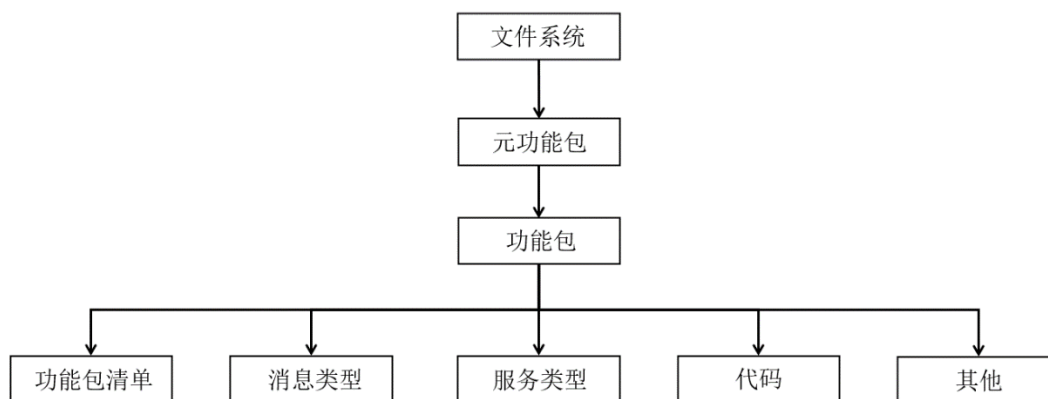


图 3-4 ROS 文件系统结构

Fig. 3-4 ROS File System Architecture

3.2 通信

3.2.1 通信机制

ROS 具有三种通信机制：基于发布/订阅的话题通信、基于客户端/服务器的服务通信以及基于 RPC 的参数服务器。

（1）消息

节点之间的通信以消息 (Message) 为载体。每一个消息都是一种严格的数据结构，支持标准数据类型，如整型、浮点型、布尔型等，也支持类似于 C 语言结构体 `struct` 的嵌套结构和数组，还可以根据需求由开发者自定义。

（2）话题通信机制

话题 (Topic) 是通信的常用方式，它有着复杂的模型，如图 3-5 所示。节点能作为发布者 (Talker) 发布消息到话题，或作为订阅者 (Listener) 订阅并接收特定话题的数据。发布者和订阅者独立运作，允许多个节点同时发布或订阅同一话题，二者没有直接连接，增加了系统的通信灵活性。

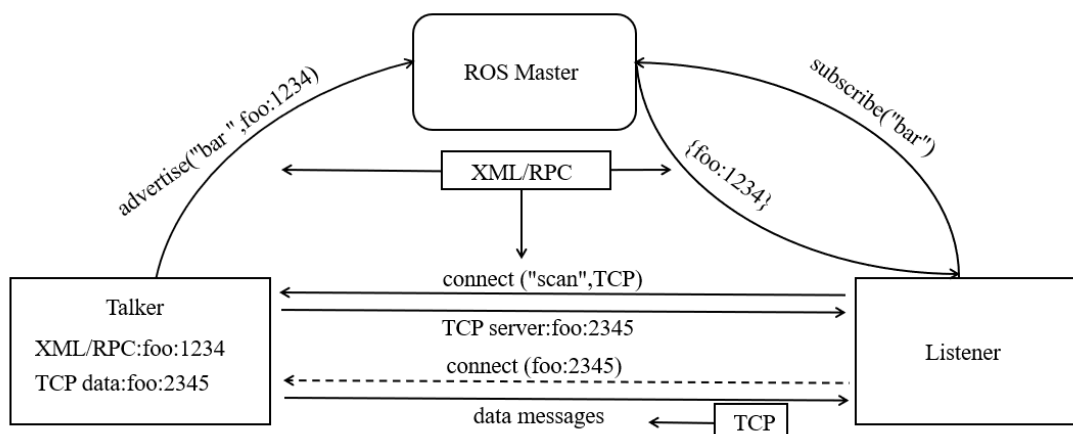


图 3-5 基于发布/订阅模型的话题通信机制

Fig. 3-5 Topic communication mechanism based on publish/subscribe model

(3) 服务通信机制

服务 (Service) 基于客户端/服务器 (Client/Server) 模型, 是一种带有应答的通信机制, 模型如图 3-6 所示。与话题的通信相比, 服务省去了发布者和订阅者之间的 RPC 通信, 是同步传输模式, 它包含两个通信数据类型: 一个用于请求, 另一个用于应答。此外, ROS 规定系统中只能有一个节点提供指定命名的服务。

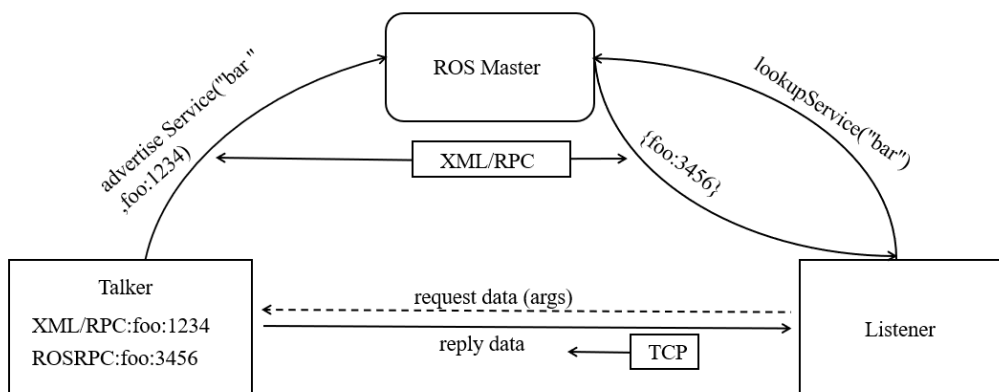


图 3-6 基于服务器/客户端的服务通信机制

Fig. 3-6 Server/Client based service communication mechanisms

(4) 参数管理机制

系统参数由 ROS Master 进行管理, 其通信机制不涉及 TCP/UDP 通信, 较为简单, 模型如图 3-7 所示。

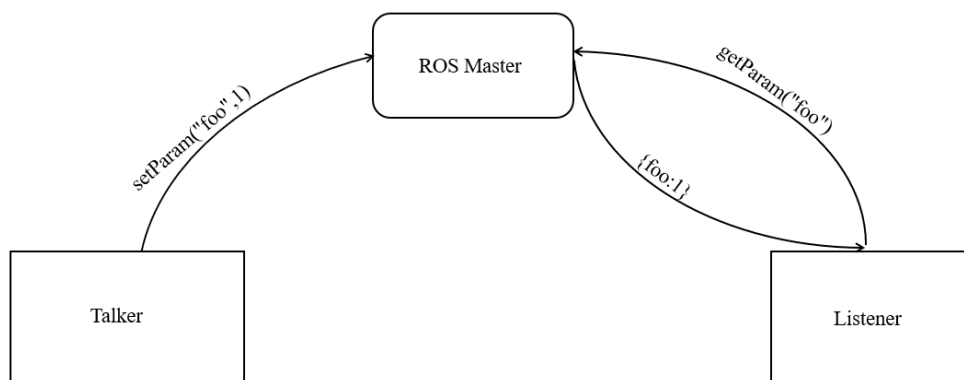


图 3-7 基于 RPC 的参数管理机制

Fig. 3-7 RPC based parameter management mechanism

3.2.2 话题与服务的区别

话题和服务是 ROS 中最基础的通信方法，从 3.2.1 节介绍的 ROS 通信机制中可以看到这两者有明显的差别，具体总结如表 3-1 所示。

表 3-1 话题和服务通信方式的比较

Tab. 3-1 Comparison of topic and service communication methods

特性	话题	服务
同步性	异步	同步
通信模型	发布/订阅	客户端/服务器
底层协议	ROSTCP/ROSUDP	ROSTCP/ROSUDP
反馈机制	无	有
缓冲区	有	无
实时性	弱	强
节点关系	多对多	一对多（一个 Server）
适用场景	数据传输	逻辑处理

话题通过解耦数据发布者和接收者进行通信，适用于频繁更新且逻辑简单的数据流，如传感器信息的传输。服务采用客户端/服务器架构，适用于数据量小但需要强逻辑处理的通信，如执行计算或系统命令。

3.3 机器人建模与仿真

机器人模型可以选择 ROS 中自带的 TurtleBot 模型，本文根据控制要求选择使用 URDF 自定义建立一个巡视小车模型。

URDF（Unified Robot Description Format），即统一机器人描述格式，它是一种用于详细描述机器人模型的语言，涵盖了机器人的多个方面，包括连杆的几何属性和视觉外观、关节的类型和属性以及机器人的物理特性等。该格式使用 XML 语言编写，下面列出机器人建模的主要标签。

<link>标签表示连杆，定义机器人某个刚体部分的视觉特征和物理属性，包括形状(shape)、尺寸(size)、颜色(color)、惯性矩阵(inertial matrix)、碰撞参数(collision properties)等。根据 URDF 模型，link 的形状可分为圆柱体、长方体和球体三种类型，其三维模型示意图如图 3-8 所示。

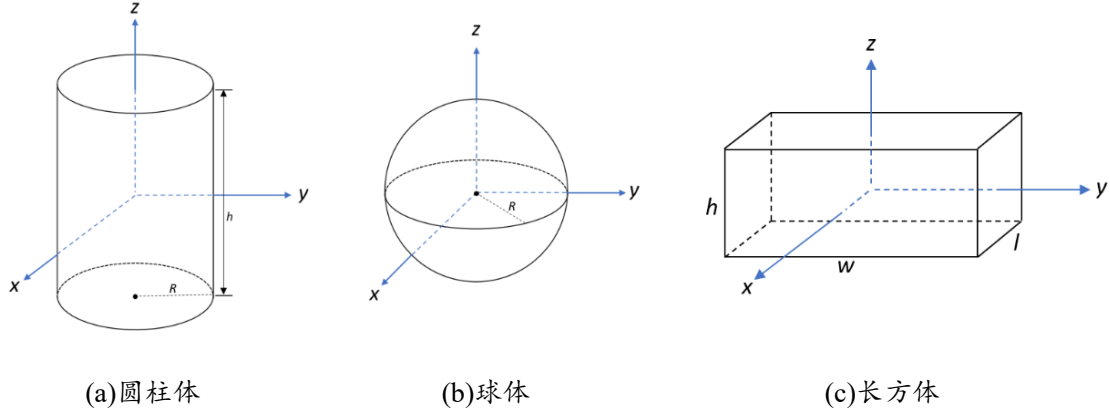


图 3-8 三种 link 类型的三维模型

Fig. 3-8 Three link types of 3D models

其相应的惯性矩阵计算方法如下：

一般惯性矩阵的计算公式如式 (3-1) 所示。

$$J = \sum m_i r_i^2 \quad (3-1)$$

根据式 (3-1)，圆柱体的惯性矩阵参数推导如下。

$$i_{xx} = \sum m_i (y_i^2 + z_i^2) = \frac{m}{V} \iiint (y^2 + z^2) dV = \frac{m}{12} (3R^2 + h^2) = i_{yy} \quad (3-2)$$

$$i_{zz} = \sum m_i (x_i^2 + y_i^2) = \frac{m}{V} \iiint (x^2 + y^2) dV = \frac{1}{2} m R^2 \quad (3-3)$$

同理可得式 (3-4)。

$$i_{xy} = i_{xz} = i_{yx} = i_{yz} = i_{zx} = i_{zy} = 0 \quad (3-4)$$

结果如式 (3-5) 所示。

$$I_{cylinder} = \begin{pmatrix} \frac{m}{12} (3R^2 + h^2) & 0 & 0 \\ 0 & \frac{m}{12} (3R^2 + h^2) & 0 \\ 0 & 0 & \frac{1}{2} m R^2 \end{pmatrix} \quad (3-5)$$

同理可推导出球体和长方体的惯性矩阵计算公式如式 (3-6)、式 (3-7) 所示。

$$I_{sphere} = \begin{pmatrix} \frac{2}{5}mR^2 & 0 & 0 \\ 0 & \frac{2}{5}mR^2 & 0 \\ 0 & 0 & \frac{2}{5}mR^2 \end{pmatrix} \quad (3-6)$$

$$I_{box} = \begin{pmatrix} \frac{m}{12}(l^2 + h^2) & 0 & 0 \\ 0 & \frac{m}{12}(w^2 + h^2) & 0 \\ 0 & 0 & \frac{m}{12}(w^2 + l^2) \end{pmatrix} \quad (3-7)$$

<joint>标签表示关节，用于描述机器人关节之间的位置关系和关节的运动属性，包括关节的空间相对位置、旋转轴、父子关节声明和位置速度限制。根据机器人的关节运动形式，分为固定关节（fixed）、旋转关节（continuous）、平面关节（planar）等类型。

<robot>标签是 URDF 模型层级最高的标签，所有的<link>和<joint>标签都必须被包含在<robot>标签里面。如图 3-9 所示，一系列<link>和<joint>构成一个完整的机器人模型。

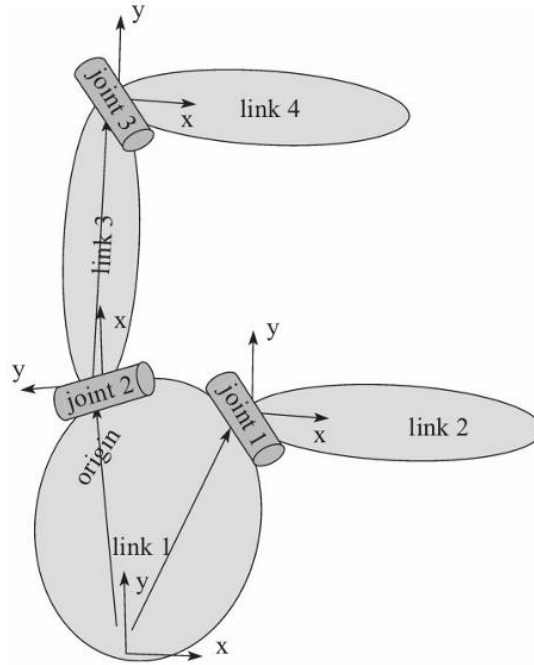


图 3-9 URDF 模型中的 robot

Fig. 3-9 Robot in URDF model

3.4 ros_control 框架

ROS 中的 `ros_control` 是为控制机器人而设计的一个中间件，它向开发者提供了一系列工具和接口，包括控制器接口、传动装置接口、硬件接口和控制器工具箱。这些组件可以加快机器人应用功能包的开发进程，提升开发效率。

`ros_control` 针对移动机器人、机械臂等不同种类的机器人，提供了多样化的控制器选项，且这些控制器具有不同的接口以适应不同的需求。为了增强代码的复用性，`ros_control` 提供了一个硬件抽象层，用来管理机器人的硬件资源。控制器通过这个抽象层来请求所需的资源，而无需直接与硬件交互，这样的设计既简化了开发过程，又提高了系统的安全性和可维护性。图 3-10 是 `ros_control` 的总体框架。`ros_control` 的数据流如图 3-11 所示，可以直观地看到每个层次具有的功能。

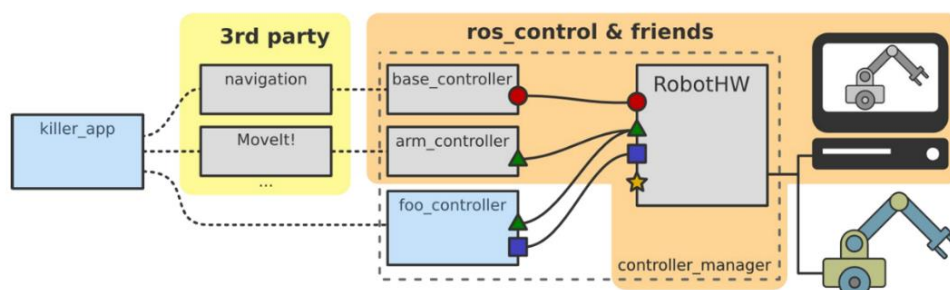


图 3-10 `ros_control` 总体框架

Fig. 3-10 General framework of `ros_control`

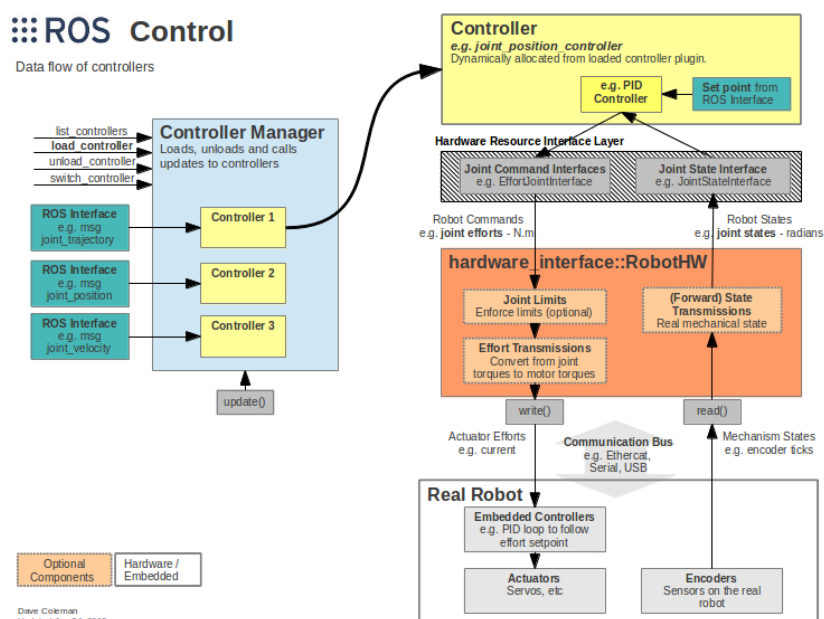


图 3-11 `ros_control` 框架数据流

Fig. 3-11 `ros_control` framework data flow

框架组成：

(1) 控制器管理器 (Controller Manager)

一个机器人通常有多个控制器，控制器管理器则提供一个通用的接口来管理不同的控制器。控制器管理器的输入就是 ROS 上层应用功能包的输出。它提供了一种多控制器控制的机制，可以实现控制器的加载、开始运行、停止运行、卸载等多种操作。图 3-12 所示的就是控制器管理器控制控制器进行状态跳转。

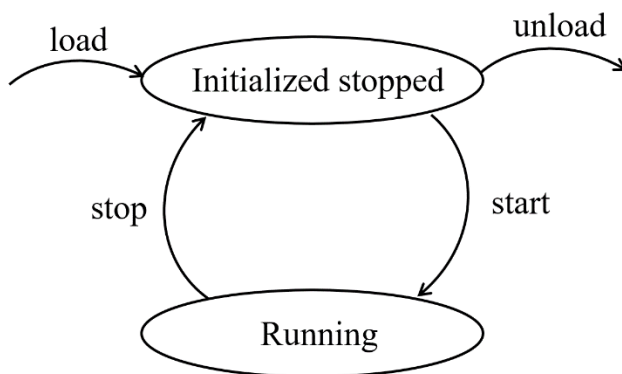


图 3-12 控制器的状态跳转流程图

Fig. 3-12 Controller's state jumping flowchart

(2) 控制器 (Controller)

控制器具备对每个关节 (joint) 进行控制的能力，它能够从硬件资源的接口处获取实时状态信息，并据此发出相应的操控指令，并且控制器集成了 PID 控制算法。ROS 中的控制器功能包 `ros_controllers` 提供了多种类型的控制器。

根据小车模型及控制需求，本文选择 `effort_controllers`（力矩控制器）中的 `joint_position_controller`（关节位置控制器）控制摄像机的转动位置，选择 `joint_velocity_controller`（关节速度控制器）控制驱动轮的速度，以及 `joint_state_controller`（关节状态控制器）中的 `joint_state_controller` 发布关节的状态信息。

(3) 硬件资源 (Hardware Resource)

控制器与 RobotHW 之间的交互接口，提供硬件资源的访问，与不同类型的控制器相匹配，如图 3-13 所示。

(4) 机器人硬件抽象 (RobotHW)

这一层直接与机器人的硬件资源交互，利用 `write` 和 `read` 方法来执行硬件操作。除了基本的读写操作，RobotHW 还负责处理关节的运动约束、力矩的转换以及状态的更新等高级功能，为控制器提供了一个稳定且抽象的接口。

(5) 真实机器人 (Real Robot)

真实机器人通常配备有嵌入式控制器，用于将接收到的高级命令转换为执行器的具体动作。

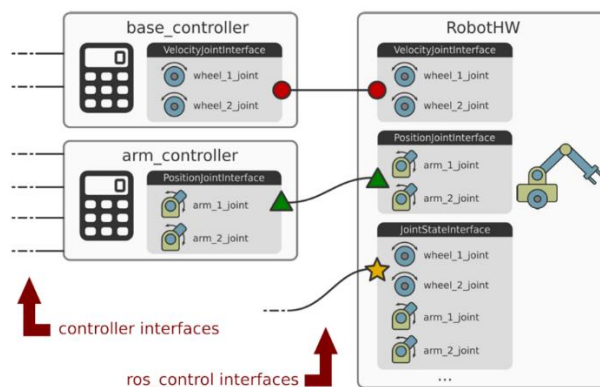


图 3-13 ros_controllers 与 ros_control 之间的多种控制接口

Fig. 3-13 Multiple control interfaces between ros_controllers and ros_control

3.5 Gazebo 物理仿真

仿真（Simulation）是一种模拟实际系统或过程的方法，它通过创建一个模型来捕捉和再现系统的关键特性和行为。这种技术特别适用于研究成本高昂、风险大或过程变化缓慢的系统。仿真技术已被广泛应用于化学工程、机械工程、电气工程和通信工程等领域。

ROS 中的 Gazebo 是一个功能强大的三维物理仿真平台，具有强大的物理引擎、精细的图形渲染能力和简便的编程与图形界面。尽管 Gazebo 使用的机器人模型与可视化工具 Rviz 中的模型一致，但为了实现更真实的仿真效果，模型中必须包含物理属性，如质量、摩擦系数和力矩等。此外，通过加入插件，可以在仿真环境中对机器人进行运动控制，同时将传感器数据集成到仿真中，并以可视化的方式展示出来^[19]。

在 Gazebo 中，控制机器人的控制器是通过使用 ros_control 和一个 Gazebo 插件适配器来完成。图 3-14 是仿真、硬件、控制器和传动之间的关系概览。

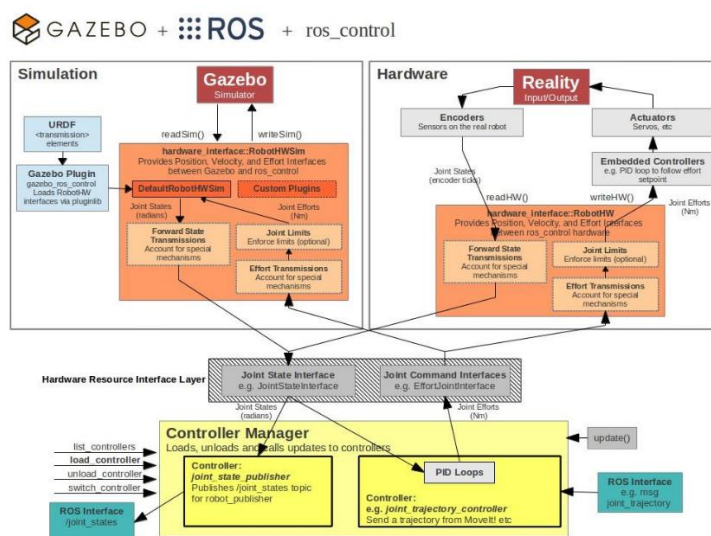


图 3-14 Gazebo 与 ros_control 结合数据流图

Fig. 3-14 Data flow diagram of Gazebo in combination with ros_control

4 语音控制小车运动程序设计

4.1 总设计

本控制系统在基于 Ubuntu 的 ROS 系统下开发语音控制程序，实现语音识别和小车运动控制。由电脑的麦克风阵列接受语音信息，通过语音识别程序将语音信息转化为文本控制指令，小车控制程序接收文本控制指令后发送详细的位置或速度控制信号给相应的控制器，再由控制器控制小车运动。控制系统的总设计如图 4-1 所示。

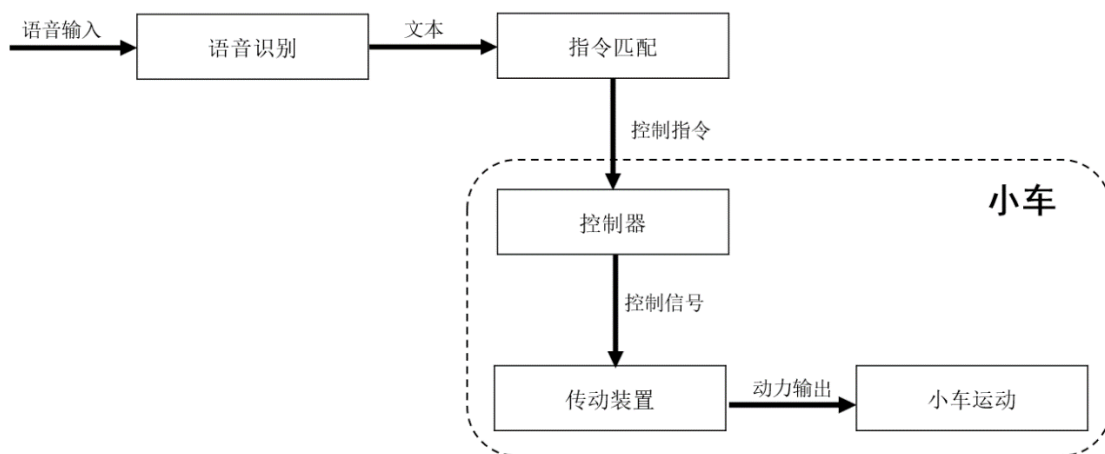


图 4-1 控制系统设计图

Fig. 4-1 Control system design diagram

4.2 ROS 开发环境搭建

由于 ROS 目前主要支持 Ubuntu 系统，故选择在虚拟机软件 VMWare Workstation Pro 上安装 Ubuntu 系统。不同的 ROS 版本对应不同的 Ubuntu 系统及系统的支持时间，具体如下表所示。

表 4-1 不同 ROS 版本对应的 Ubuntu 版本

Tab. 4-1 Ubuntu versions for different ROS versions

Ubuntu 版本	ROS 版本	官方支持时间
14.04 LTS	Indigo Lgloo	April, 2019
16.04 LTS	Kinetic Kame	April, 2021
18.04 LTS	Melodic Morenia	May, 2023
20.04 LTS	Noetic Ninjemys	May, 2025

为保证软件正常使用，本文选择 2020 年发布的长期支持版本 ROS Noetic Ninjemys，根据官方版本对应信息，选择在虚拟机上安装 20.04 LTS 版本的 Ubuntu 系统。首先在电脑

上安装虚拟机软件 VMWare WorkStation Pro，然后在 Ubuntu 系统官网下载系统镜像资源，在虚拟机上完成系统安装，系统界面如图 4-2 所示。

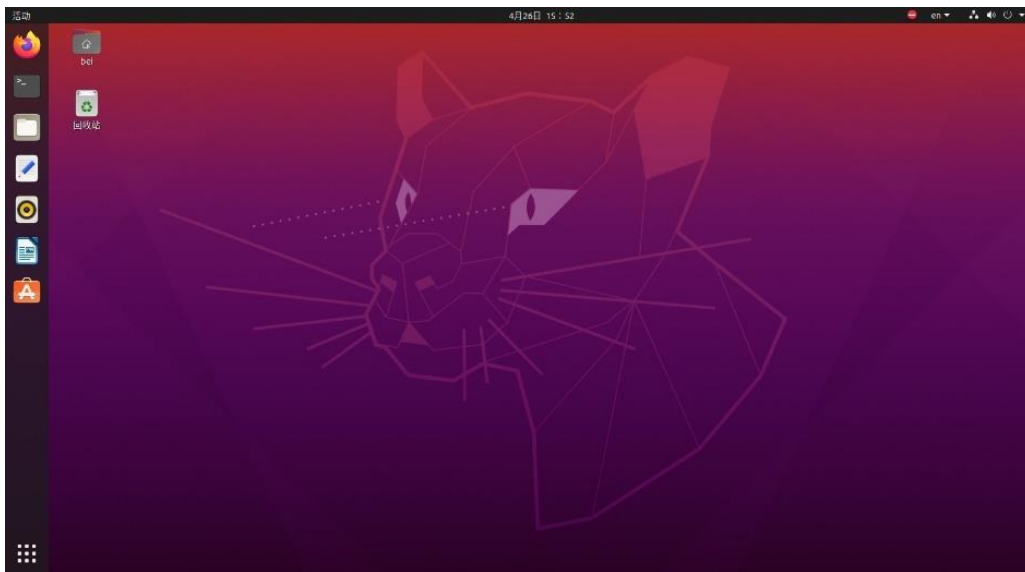


图 4-2 Ubuntu 20.04 界面

Fig. 4-2 Ubuntu20.04 Interface

Ubuntu 系统上安装 ROS 的过程如下：

1. 添加 ROS 软件源。由于 Ubuntu 系统的默认软件源连接不稳定，故本文选择使用国内复旦大学的软件源。
2. 添加密钥。从 Ubuntu 的 Keyserver 添加密钥。
3. 更新软件源缓存数据。
4. 安装 ROS。为保证后续的程序开发和仿真能顺利进行，本文选择安装桌面完整版，即 `ros-noetic-desktop-full`。
5. 安装 `rosdep` 工具。
6. 设置环境变量。

4.3 巡视小车模型设计

4.3.1 创建小车 URDF 模型

根据上一章的机器人建模方法，创建一个巡视小车模型。模型由底盘、驱动轮、万向轮、转轴、2D 摄像机和 RGB-D 摄像机组成。其中，驱动轮由电机驱动，2D 摄像机可在 Z 轴无限位旋转和 Y 轴限位旋转。巡视小车模型如图 4-3 所示。

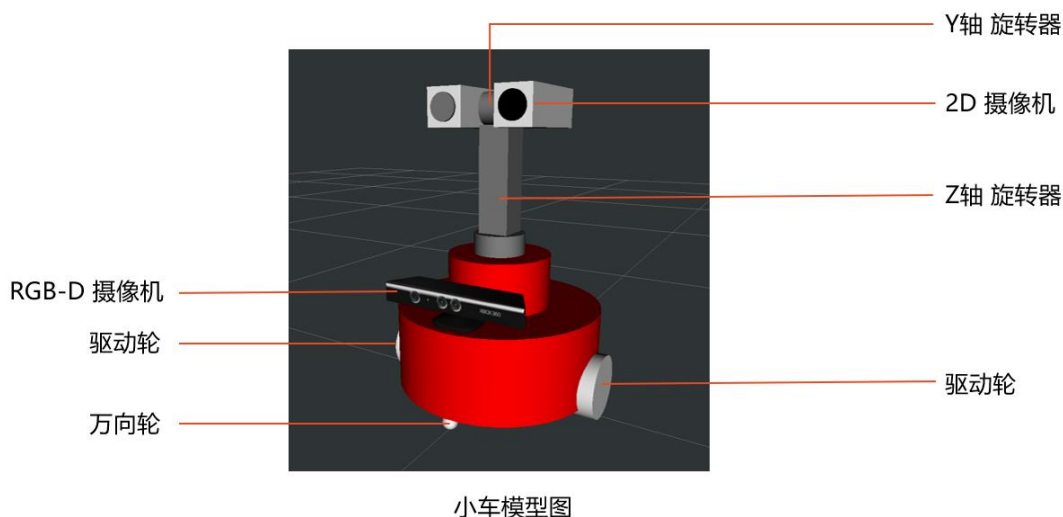


图 4-3 巡视小车 URDF 模型结构

Fig. 4-3 URDF model structure for car

2D 摄像机可以获取环境中的 RGB 彩色图像信息。RGB-D 摄像机则是一种新型的先进传感器，它具有获取环境中的彩色图像（RGB）和深度信息（D）的能力，不仅具有普通摄像机的功能，还通过红外结构光、Time-of-Flight 等原理获取每个像素的深度信息等及其丰富的数据^{错误!未找到引用源。}。RGB-D 摄像机不仅可用于即时定位与地图构建，还可用于图像处理、物体识别等多个领域。但是 RGB-D 摄像机也存在着一些局限性，例如测量视野窄、盲区大、噪声大等。Kinect 传感器（见图 4-4）便是 RGB-D 摄像机的一种。



图 4-4 kinect 相机

Fig. 4-4 Kinect Camera

使用命令行工具 `urdf_to_graphviz` 命令查看小车的 URDF 模型的整体结构，生成一个 pdf 文件，看到小车模型的整体结构如图 4-5 所示。

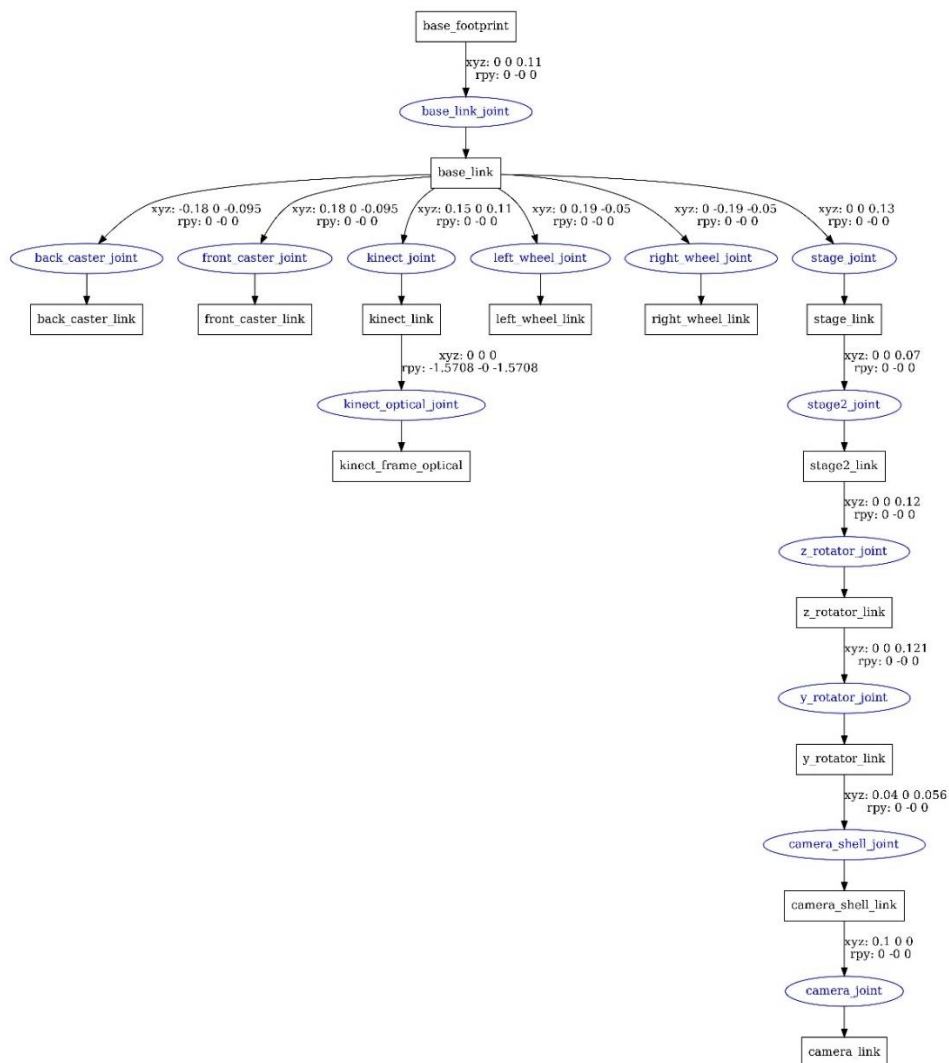


图 4-5 小车的 URDF 模型结构树形图

Fig. 4-5 Tree diagram of the URDF model structure of the car

由于 URDF 文件并不支持代码复用，当为一个包含多个连杆和关节的复杂机器人创建模型时，URDF 代码会变得非常繁杂，而这恰恰违背了 ROS 的设计目标——提高代码的复用率。而另外一种精简、可复用、模块化的描述形式 XACRO 具有代码精简，含可编程接口的特点，于是用此格式将原 URDF 模型文件进行简化。

4.3.2 在 Rviz 中显示小车模型

完成小车 URDF 模型的设计后，使用 ROS 中的三维可视化工具 Rviz 将该模型可视化显示出来，以便检查是否符合设计目标，小车模型如图 4-6 所示。

再启动节点 `joint_state_publisher`，该节点发布每个 joint（除 fixed 类型）的状态，而且可通过 UI 对 joint 进行控制，拖动控制条，对应的关节会转动。

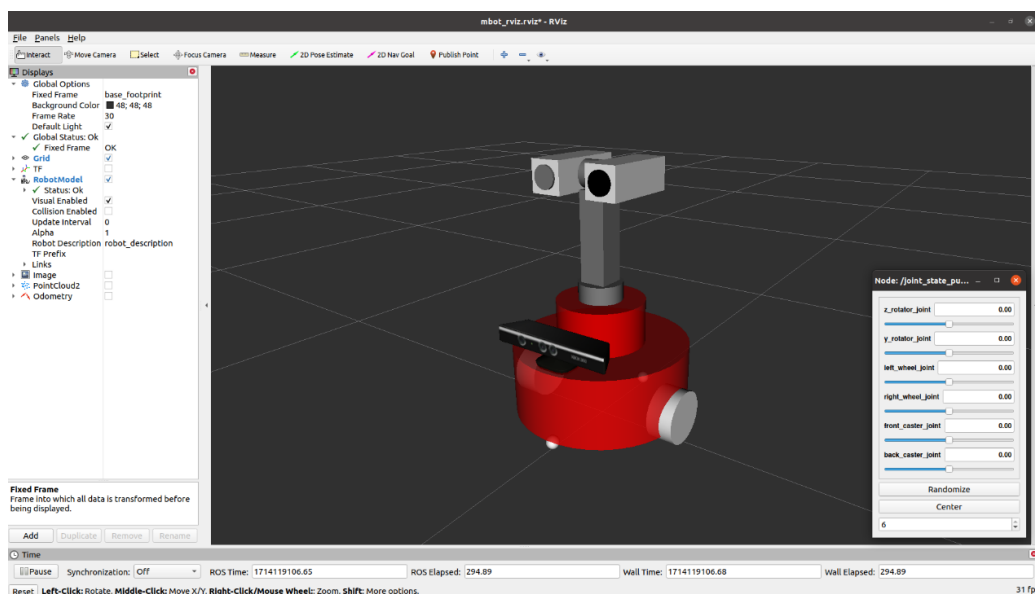


图 4-6 Rviz 导入小车模型界面

Fig. 4-6 Rviz interface for importing car model

4.4 Gazebo 仿真环境设计

4.4.1 构建仿真环境

在控制仿真之前需要创建一个仿真环境。Gazebo 中有两种创建仿真环境的方法：直接插入模型和使用 Gazebo 提供的 BuildingEditor 工具手动绘制地图环境。本文选择直接插入模型的方法，使用 Gazebo 的模型库中的物件模型，构建一个仿真环境，其效果如图 4-7 所示。

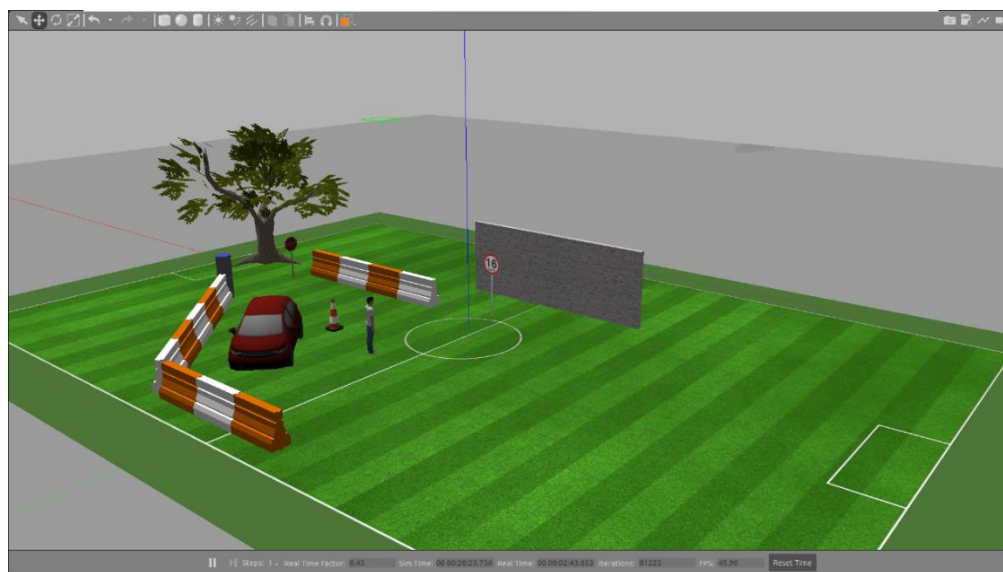


图 4-7 Gazebo 仿真环境

Fig. 4-7 Gazebo simulation scenario

4.5 语音识别程序设计

根据前文对语音识别技术的介绍，ROS 中的语音识别功能包 `pocketsphinx` 对中文的识别效果较差，考虑到语音识别准确度和开发难度，本文选择科大讯飞开放平台的语音识别软件开发工具包来实现语音识别的功能。又考虑到识别程序仅需要识别简单的控制命令词，本文选择使用科大讯飞离线命令词 Linux SDK 进行语音识别程序的设计。

小车的控制指令分为车体运动指令和 2D 摄像机转动指令，车体运动指令为：“前进”、“后退”、“左转”、“右转”和“停止”，2D 摄像机转动指令为：“向左看”、“向右看”、“向前看”、“向后看”、“看中间”、“向上看”和“向下看”。

根据科大讯飞 SDK 开发指南，在第一次使用语法进行识别时，需要先编写一个 BNF 语法文件，然后在程序中调用 `QISRBuildGrammar` 接口编译本地语法文件，获得语法 ID，并在会话时，传入语法 ID，以使用该语法，在之后的会话中，继续使用此语法进行识别，无需再次构建。

根据控制指令编写 BNF 语法文件如下：

```
#BNF+IAT 1.0 UTF-8;
!grammar call;
!slot <want>;
!slot <direction>;
!slot <action>;
!start <callstart>;

<callstart>:[<want>]<control>;

<want>:向|看;

<control>:<direction><action>|<direction>|<action>;

<direction>:前|后|左|左边|右|右边|上|下|中间;
<action>:走|转|看|停止|前进|后退|左转|右转;
```

此语法涵盖需要的控制指令，在词语的基础上增添了单个字，以提高识别容错率，从而能更好地控制小车车体的运动和 2D 摄像机的摆动。

语音识别程序的流程图如图 4-8 所示。

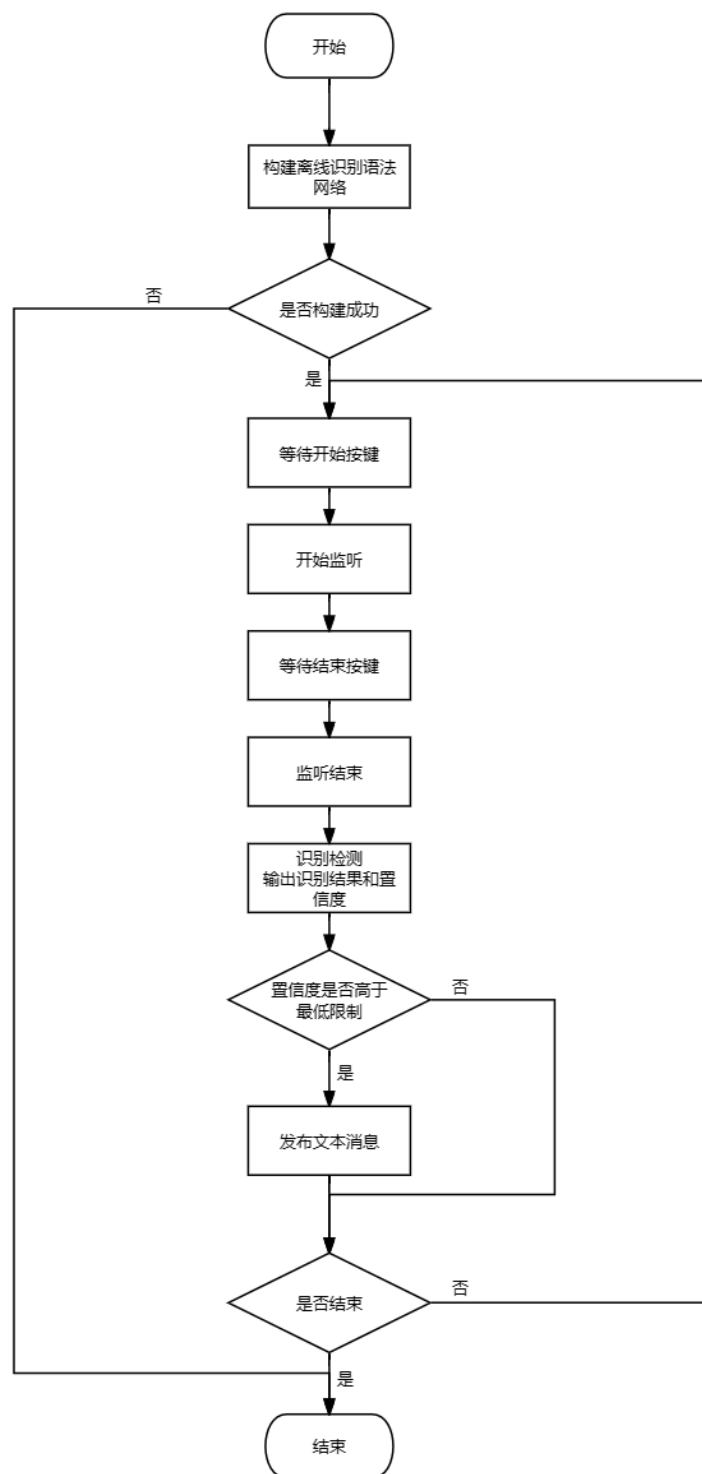


图 4-8 语音识别程序流程图

Fig. 4-8 Flowchart of speech recognition program

使用命令词识别 SDK 设计的语音识别程序识别命令词有以下几个步骤：

- (1) 构建离线识别语法网络。
- (2) 等待开始按键输入。
- (3) 输入语音。语音输入为笔记本电脑麦克风阵列接收的语音信号。

- (4) 等待结束按键。
- (5) 语音信号识别。
- (6) 输出识别结果以及置信度。语法识别结果包含结果的置信度，置信度为识别是否准确的一个度量。
- (7) 根据置信度大小决定是否发布指令消息。只有置信度大于程序设置的最低限值时，识别结果才会被发布。
- (8) 一次语音识别结束。

语音识别程序中需要设置相关参数。在此程序中，设置识别引擎类型为离线引擎 `local`，音频采样率为 16000，输出结果字符串编码格式为 `utf-8`，输出结果文本格式为 `plain`。

4.6 控制程序设计

语音识别会将识别的文本以字符串的形式发布，语音控制程序在接收文本指令后先对指令进行关键词比对，只有匹配到对应的关键词后，控制程序才会进行下一步。文本与关键词的对照关系如表 4-2 所示。

表 4-2 文本与指令关键词对应

Tab. 4-2 Text to command keyword correspondence

文本指令	对应控制指令关键词
前进，向前走	前进
后退，向后走	后退
左转，向左转，向左走	左转
右转，向右转，向右走	右转
停止	停止
向左看，看左边	看、左
向右看，看右边	看、右
向前看，看前面	看、前
向后看，看后面	看、后
向上看，看上面	看、上
向下看，看下面	看、下
看中间	看、中间

在接收到符合关键词的文本指令后，语音控制程序便会向 `Gazebo` 控制节点发布控制消息。控制程序的流程图如图 4-9 所示。

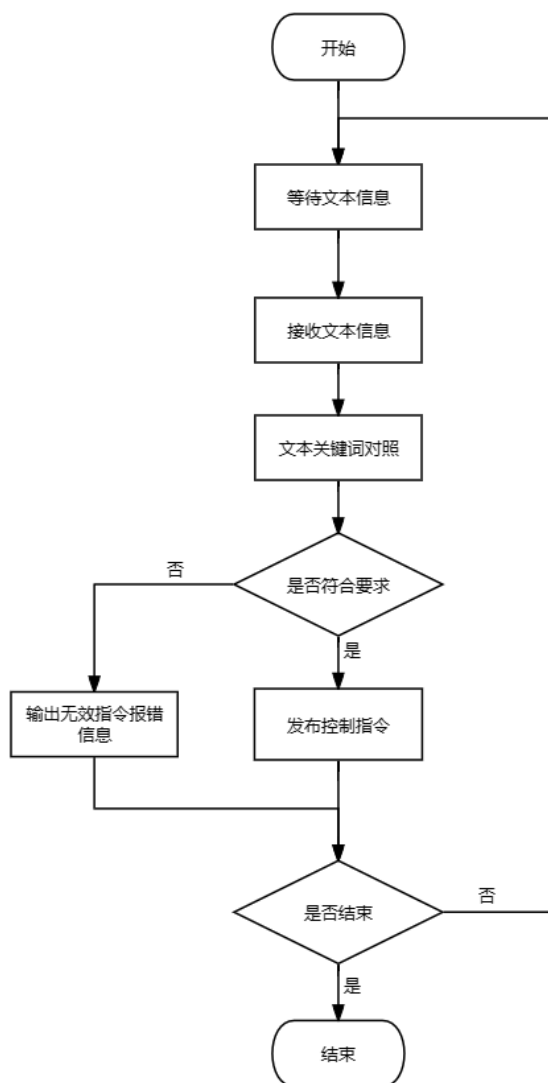


图 4-9 控制程序流程图

Fig. 4-9 Control program flow chart

控制程序的运行有以下几个步骤：

- (1) 等待并接受语音识别程序发布的消息文本；
- (2) 将文本内容与指令关键词进行对比匹配；
- (3) 若文本内容符合要求，则发布相应的控制指令，否则发出指令无效警告。

4.7 小车模型添加 Gazebo 仿真属性

<gazebo>标签用于描述机器人模型在 Gazebo 中仿真所需要的相关参数，例如机器人的外观属性、Gazebo 控制器插件和传感器插件及其配置参数等。此标签在 Gazebo 仿真时必须加入到 URDF 模型中。

4.7.1 为 link 添加<gazebo>标签

针对小车模型，需要对每一个 link 添加 gazebo 标签，并在该标签中添加颜色配置属性 material。颜色属性与 link 标签中配置的 color 一致。

4.7.2 为 joint 添加传动系统

传动系统（Transmission）将机器人的关节动作指令转换成执行器的驱动信号。机器人中每个需要运动的关节都需要配置对应的传动系统。在小车模型中加入 transmission 元素，将传动装置与对应的 joint 绑定。小车驱动轮的传动系统配置如下：

```
<transmission name="left_wheel_joint_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="left_wheel_joint">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="left_wheel_joint_motor">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

以上代码中，<joint name="">标签定义了将要绑定驱动器的 joint，此处绑定了左驱动轮关节，<type>标签声明了所使用的传动装置类型，此处为 SimpleTransmission，<hardwareInterface>定义了硬件接口的类型，此处使用的是力矩控制接口。

4.7.3 添加关节约束

关节约束（JointLimits）是硬件抽象层中的一部分，是维护一个关节约束的数据结构，这些约束数据不仅包含关节速度、位置、加速度、加加速度、力矩等方面的约束，还包含起安全作用的位置软限位、速度边界和位置边界等。在 URDF 中设置摄像机 y_rotator 的 JointLimits 参数：

```
<limit lower="-1.047" upper="1.047" effort="3.3" velocity="1.0" />
```

其中，lower 和 upper 分别为位置下限和位置上限，此处设置 y_rotator 的旋转角度范围为-1.047rad~1.047rad，即-60°~60°，effort 为最大力矩，此处设置为 3.3N·m，velocity 为最大速度，此处设置为 1.0m/s。

4.8 在 Gazebo 中显示小车模型

创建一个启动文件 car_asr_control.launch，运行 Gazebo，加载小车模型，并且启动一些必要的节点。小车在 Gazebo 中仿真如图 4-10 所示。

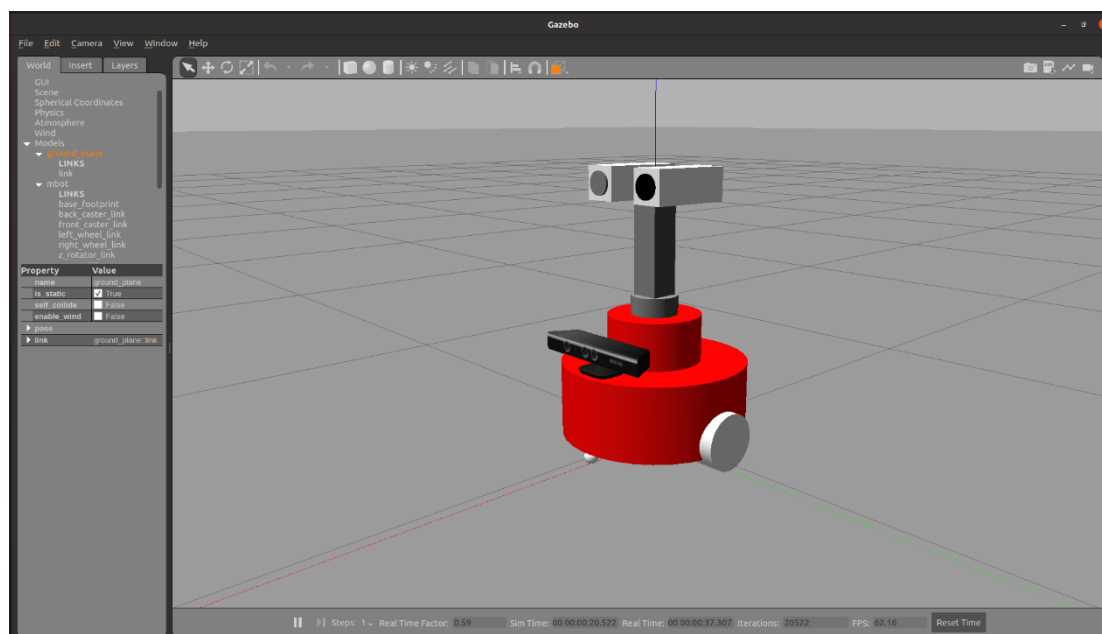


图 4-10 Gazebo 导入小车模型界面

Fig. 4-10 Gazebo interface for importing car model

4.9 配置控制器

根据前文对 `ros_control` 框架的介绍，在小车 URDF 模型的基础上，为小车的两个驱动轮关节配置速度控制器 `joint_velocity_controller`，为 2D 摄像机的两个旋转轴关节配置位置控制器 `joint_position_controller`。这些控制器需要使用 YAML 配置文件加载入 ROS 的参数服务器，YAML 配置文件如下所示。

```

mbot:

  # Publish all joint states -----
  mbot_joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers -----
  z_rotator_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: z_rotator_joint
    pid: {p: 10.0, i: 0.001, d: 0.0, i_clamp_min: 0.0, i_clamp_max: 1.0}
  y_rotator_joint_position_controller:
    type: effort_controllers/JointPositionController
    joint: y_rotator_joint

```

```
pid: {p: 0.05, i: 0, d: 0.01, i_clamp_min: 0.0, i_clamp_max: 10.0}
```

```
# Velocity Controllers -----
```

```
left_wheel_joint_velocity_controller:
```

```
  type: effort_controllers/JointVelocityController
```

```
  joint: left_wheel_joint
```

```
  pid: {p: 0.3, i: 0.0, d: 0.0}
```

```
right_wheel_joint_velocity_controller:
```

```
  type: effort_controllers/JointVelocityController
```

```
  joint: right_wheel_joint
```

```
pid: {p: 0.3, i: 0.0, d: 0.0}
```

Yaml 文件定义了控制器的类型、关联的关节和 PID 的各项参数及其最值。

4.10 添加 Gazebo 控制器插件

到现在为止，小车还是一个静态显示的模型，如果要控制它运动，还需要使用 Gazebo 插件。

Gazebo 插件扩展了 URDF 模型的功能，使模型可以与 ROS 通信，从而进行电机的控制和传感器的仿真输出，让机器人模型更加贴近现实。根据作用范围的不同，Gazebo 插件可以应用到 URDF 模型的<robot>、<link>和<joint>上，插件均需要使用<gazebo>标签进行封装。

Gazebo 提供了一系列的控制插件，本文选择常规控制插件 libgazebo_ros_control.so，并在 xacro 文件中添加如下插件声明：

```
<plugin name="controllers" filename="libgazebo_ros_control.so">
  <robotNamespace>/mbot</robotNamespace>
  <controlPeriod>0.01</controlPeriod>
  <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
</plugin>
```

在加载控制器插件的过程中，需要配置一系列参数，其中比较关键的参数如下。

<robotNamespace>：机器人的命名空间，控制器插件发布或订阅的数据都保存在此命名空间下。此处设置小车的命名空间为/mbot。

<controlPeriod>：控制周期，单位为秒。此处设置为 0.01s。

<robotSimType>：仿真类型。此处设置为默认类型。

4.11 添加 Gazebo 传感器仿真

根据先前建立的小车模型，要想两个摄像机能在 Gazebo 中正常工作，需要添加相应的配置参数和传感器插件。

4.11.1 2D 摄像机

配置传感器插件 `libgazebo_ros_camera.so`，同时设置插件的参数，包括命名空间、发布图像的话题、参考坐标系等，并配置相应的参数，如类型（`type`）为 `camera`；传感器的命名；摄像机的分辨率、编码格式、图像范围、噪音参数等。

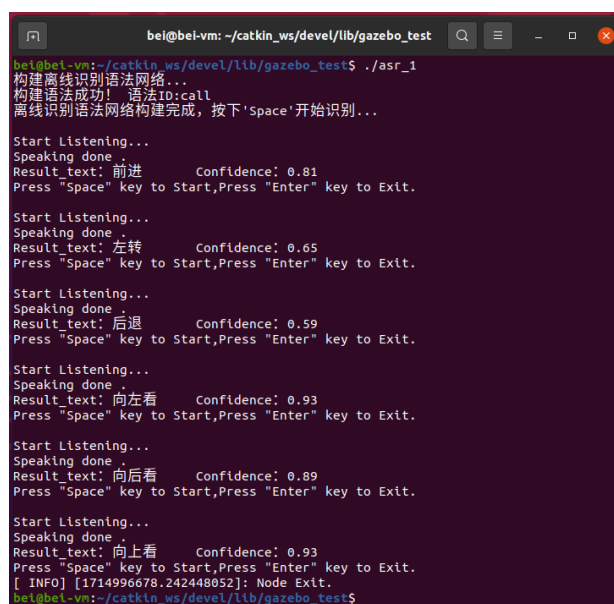
4.11.2 RGB-D 摄像机

与 2D 摄像机相同，配置传感器插件 `libgazebo_ros_openni_kinect.so`，同时设置相关参数，如图像大小、参考坐标系、发布数据的话题等。

5 系统整体测试

5.1 命令词识别准确度测试

完成语音识别程序的编写后，启动语音识别节点，通过麦克风阵列接收语音信号，识别结果如图 5-1 所示。



```

bel@bel-vm: ~/catkin_ws/devel/lib/gazebo_test
bel@bel-vm:~/catkin_ws/devel/lib/gazebo_test$ ./asr_1
构建离线识别语法网络...
构建语法成功! 语法ID:call
离线识别语法网络构建完成, 按下'Space'开始识别...

Start Listening...
Speaking done
Result_text: 前进      Confidence: 0.81
Press "Space" key to Start,Press "Enter" key to Exit.

Start Listening...
Speaking done
Result_text: 左转      Confidence: 0.65
Press "Space" key to Start,Press "Enter" key to Exit.

Start Listening...
Speaking done
Result_text: 后退      Confidence: 0.59
Press "Space" key to Start,Press "Enter" key to Exit.

Start Listening...
Speaking done
Result_text: 向左看    Confidence: 0.93
Press "Space" key to Start,Press "Enter" key to Exit.

Start Listening...
Speaking done
Result_text: 向后看    Confidence: 0.89
Press "Space" key to Start,Press "Enter" key to Exit.

Start Listening...
Speaking done
Result_text: 向上看    Confidence: 0.93
Press "Space" key to Start,Press "Enter" key to Exit.
[ INFO] [1714996678.242448052]: Node Exit.
bel@bel-vm:~/catkin_ws/devel/lib/gazebo_test$
  
```

图 5-1 语音识别程序运行界面

Fig. 5-1 Speech recognition program running interface

在室外环境下，距离麦克风阵列 20cm 左右，对不同控制指令各进行 100 次左右的识别测试，得到的命令词识别结果如表 5-1 所示。

表 5-1 命令词识别统计

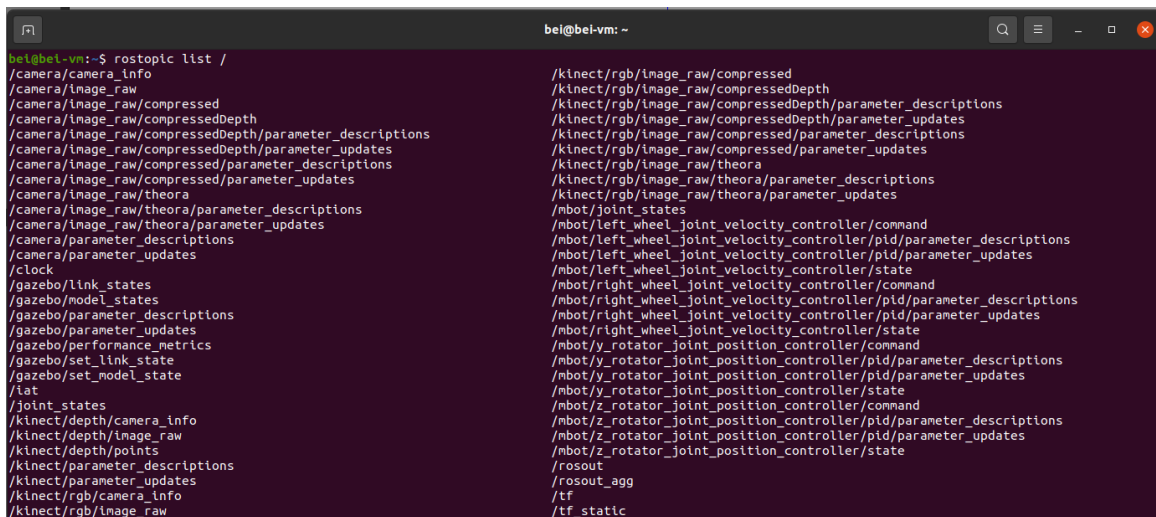
Tab. 5-1 Command word recognition statistics

语音输入	正确识别次数	错误识别次数	识别率
前进	107	3	97.27%
后退	93	9	91.18%
左转	90	5	94.74%
右转	89	6	93.68%
停止	105	0	100.00%
向前看	95	4	95.96%
向后看	96	5	95.05%
向左看	92	6	93.88%
向右看	94	9	91.26%
向上看	91	7	92.86%
向下看	95	8	92.23%
看中间	97	1	98.98%

由测试结果可知,在室外有一定噪声的条件下,命令词识别的准确率均在 90%以上,基本上可以满足使用的要求。

5.2 小车运动控制测试

小车模型中已经加入了 libgazebo_ros_control.so 插件,可以使用控制器控制小车运动。查看系统当前的话题列表如图 5-2 所示。



```
bel@bel-vm:~$ rostopic list /
/camera/camera_info
/camera/image_raw
/camera/image_raw/compressed
/camera/image_raw/compressedDepth
/camera/image_raw/compressedDepth/parameter_descriptions
/camera/image_raw/compressedDepth/parameter_updates
/camera/image_raw/compressed/parameter_descriptions
/camera/image_raw/compressed/parameter_updates
/camera/image_raw/theora
/camera/image_raw/theora/parameter_descriptions
/camera/image_raw/theora/parameter_updates
/camera/parameter_descriptions
/camera/parameter_updates
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/iat
/joint_states
/kinect/depth/camera_info
/kinect/depth/image_raw
/kinect/depth/points
/kinect/parameter_descriptions
/kinect/parameter_updates
/kinect/rgb/camera_info
/kinect/rgb/image_raw
/kinect/rgb/image_raw/compressed
/kinect/rgb/image_raw/compressedDepth
/kinect/rgb/image_raw/compressedDepth/parameter_descriptions
/kinect/rgb/image_raw/compressedDepth/parameter_updates
/kinect/rgb/image_raw/compressed/parameter_descriptions
/kinect/rgb/image_raw/compressed/parameter_updates
/kinect/rgb/image_raw/theora
/kinect/rgb/image_raw/theora/parameter_descriptions
/kinect/rgb/image_raw/theora/parameter_updates
/mbot/joint_states
/mbot/left_wheel_joint_velocity_controller/command
/mbot/left_wheel_joint_velocity_controller/pid/parameter_descriptions
/mbot/left_wheel_joint_velocity_controller/pid/parameter_updates
/mbot/left_wheel_joint_velocity_controller/state
/mbot/right_wheel_joint_velocity_controller/command
/mbot/right_wheel_joint_velocity_controller/pid/parameter_descriptions
/mbot/right_wheel_joint_velocity_controller/pid/parameter_updates
/mbot/right_wheel_joint_velocity_controller/state
/mbot/y_rotator_joint_position_controller/command
/mbot/y_rotator_joint_position_controller/pid/parameter_descriptions
/mbot/y_rotator_joint_position_controller/pid/parameter_updates
/mbot/y_rotator_joint_position_controller/state
/mbot/z_rotator_joint_position_controller/command
/mbot/z_rotator_joint_position_controller/pid/parameter_descriptions
/mbot/z_rotator_joint_position_controller/pid/parameter_updates
/mbot/z_rotator_joint_position_controller/state
/rosout
/rosout_agg
/tf
/tf_static
```

图 5-2 系统话题列表

Fig. 5-2 List of system topics

从图 5-2 中可以看到,系统中已经存在 4 个 controller 的各项话题。此时就可以向控制器下的 command 话题发布控制消息,以控制小车在 Gazebo 中运动。

由于 4 个控制器的控制规律均为 PID 控制,故需要进一步对控制器的 pid 参数进行调试整定。ROS 提供了一个 Qt 架构的后台图形工具套件—rqt_common_plugins,这里使用该套件中的 Plot 工具、Dynamic Reconfigure 工具和 Message Publisher 工具。

Plot 是一个二维数值曲线绘制工具,在 Topic 输入框中输入需要显示的话题消息,消息包含的数据则会以曲线的形式在 xy 坐标系中描绘出来。

Dynamic Reconfigure 工具可以在系统运行时,实时地动态配置 ROS 系统中的各项参数。

Message Publisher 工具可以直接向指定的话题发布消息,且发布频率和内容自定义。

利用以上三个工具,可以非常便捷地调试四个控制器的 PID 参数。调试界面如图 5-3 所示。

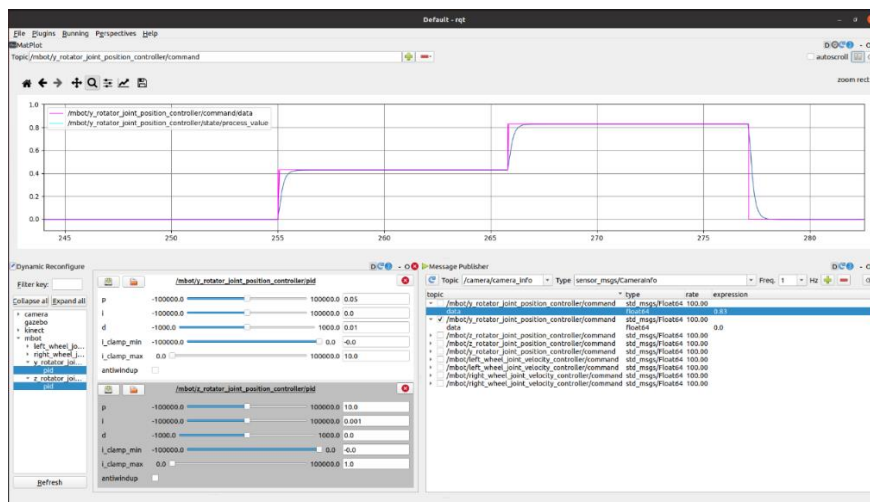


图 5-3 Rqt 工具参数调试界面

Fig. 5-3 Rqt tools parameter debugging interface

经过不断地调试后，将左右驱动轮控制器配置为纯比例控制器，Z 轴旋转器配置为比例积分控制器，Y 轴旋转器配置为比例微分控制器。以 Y 轴旋转器为例，由图 5-3 可以看出，其在输入阶跃干扰下的动态响应为过阻尼响应，无超调且调节时间短。

5.3 摄像机图像显示

小车上的 2D 摄像机和 RGB-D 摄像机拍摄的画面以消息的形式发布到话题中。在图 5-1 可以看到/camera 和/kinect 分别是两个摄像机发布的话题。

Qt 工具箱中的 Image View 工具可以显示摄像机的画面，选择对应的摄像机发布的图像话题/camera/image_raw 和/kinect/depth/image_raw，就可以看到如图 5-4 所示的图像信息。

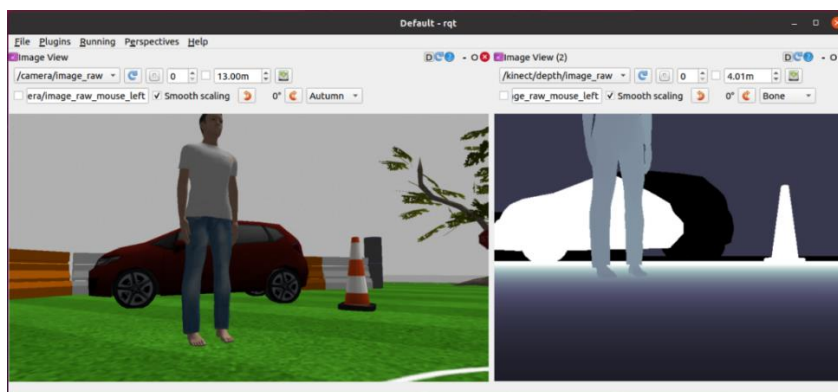


图 5-4 Rqt 工具摄像机画面显示界面

Fig. 5-4 Rqt tools camera screen display interface

5.4 总体控制效果

启动 Gazebo 仿真软件，加载小车模型和仿真环境，加载控制器，同时启动语音识别程序。系统运行效果如图 5-5 所示。图中左侧两个终端窗口分别为语音识别程序运行窗口和

控制程序运行窗口，中间为 Gazebo 仿真界面，右侧窗口的两个画面分别为 2D 摄像机和 RGB-D 摄像机的实时拍摄画面。

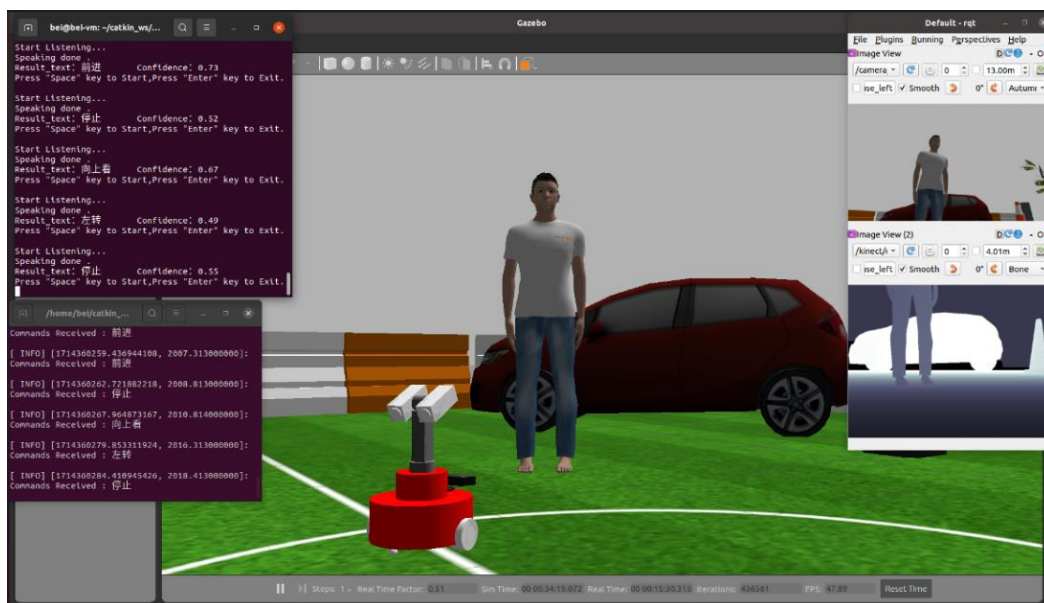


图 5-5 语音识别控制系统运行界面

Fig. 5-5 Speech Recognition Control System Operation Interface

在图 5-5 中，语音识别程序依次识别出“前进”、“停止”、“向上看”、“左转”、“停止”命令，并向控制程序发布相应控制指令，Gazebo 中的小车接收到指令后执行相应的动作。从运行图可以看出，该系统能准确地识别出文本信息并控制小车按照指令进行运动。

5.5 总体通信数据流

rqt_common_plugins 工具套件中的 rqt_graph 工具可以将 ROS 系统中的计算图以图形化的方式显示出来，如图 5-6 所示。

可以直观的看出，语音识别节点/asr_1 发布消息到/iat 话题，控制节点/control 订阅该话题接收识别文本消息，同时发布控制指令消息到四个控制器话题，再由 Gazebo 中的节点/gazebo 订阅控制器话题，进而控制小车运动。同时节点/gazebo 发布消息到/camera 和 /kinect 两个摄像机话题。以上节点间的通信构成了整个控制系统的通信流。

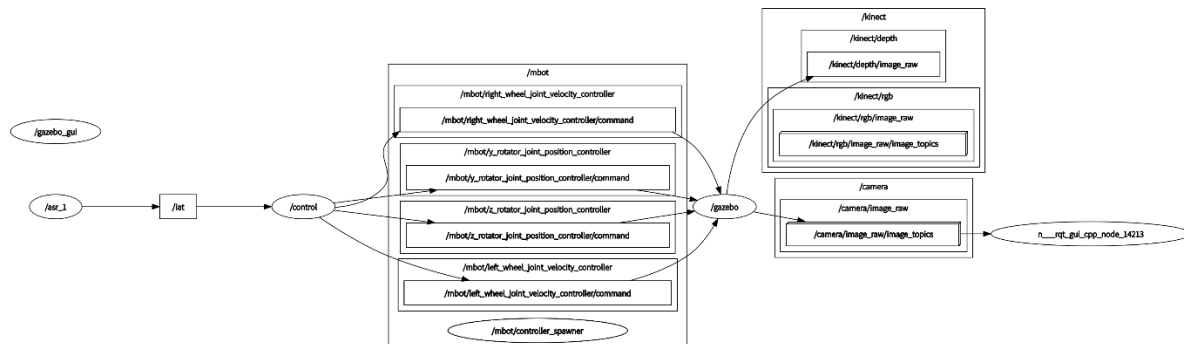


图 5-6 系统通信数据流图

Fig. 5-6 System communication data flow diagram

6 总结

本文基于 ROS 机器人操作系统设计了一个离线语音识别程序，并在此基础上，将语音识别与机器人控制结合，设计了一个巡视小车控制程序，在 Gazebo 仿真中实现语音控制巡视小车车体的运动和摄像机的转动。

本论文的主要工作归纳如下：

(1) 查阅文献，了解国内外语音识别技术的发展状况与主流的技术算法，通过对比各种语音识别实现方案，最终选择科大讯飞开放平台的软件开发包进行语音识别程序的开发。

(2) 深入了解 ROS 机器人操作系统，主要包括 ROS 架构、通信机制、ros_control 机器人控制框架和机器人的建模仿真。

(3) 环境搭建与程序设计，在虚拟机上安装 Ubuntu 系统并搭建 ROS 的开发环境。设计离线命令词识别程序，并将语音识别与 ROS 相合，设计巡视小车模型及其运动控制程序。

(4) 系统测试，对各个程序的功能和效果进行测试。在室外环境下测试命令词识别的准确度，在 Gazebo 仿真中测试巡视小车的控制性能。测试结果表明，此系统能准确地识别出控制指令并控制巡视小车执行相应的动作。

本文虽然实现了基本的语音识别与小车的运动控制，但是由于时间和个人能力的限制，本设计仍有以下可以改进的地方：

(1) 提高语音识别程序的稳定性。本文设计的语音识别程序虽有较高的识别准确度，但偶尔会出现有语音输入而无结果输出的不稳定情况。故语音识别程序还有优化的空间。

(2) 增强小车运动控制的响应效果。本文小车驱动轮控制器的控制规律为纯比例控制，在实际控制时有余差，仿真时在直行指令下会出现路径为曲线的现象。在测试中加入积分控制规律时小车模型在 Gazebo 仿真会出现严重的抖动甚至使整个模型崩溃。因此控制方案有待进一步的改进。

此外，本文设计的语音控制系统可以直接搭建在如树莓派、香橙派一类的单板计算机上，以实现语音控制实体机器人。与在线语音识别相比，本文采用的离线命令词识别具有识别范围限定和无需连接互联网等特点，这使得它在硬件要求上相对较低，识别速度更快，实现成本更少，与当下主流的采用联网进行语音交互的实现方案相比更有优势，有着很好的应用前景。

参考文献

- [1] 工信部. 十五部门联合印发《“十四五”机器人产业发展规划》[J]. 机器人技术与应用, 2022(1):1.
- [2] 胡春旭. ROS 机器人开发实践[M]. 北京:机械工业出版社, 2018.
- [3] 马晗, 唐柔冰, 张义, 等. 语音识别研究综述[J]. 计算机系统应用, 2022,31(01): 1-10. DOI: 10.15888/j.cnki.csa.008323.
- [4] Rabiner L R. A tutorial on hidden Markov models and selected applications in speech recognition[J]. Proceedings of the IEEE, 1989, 77(2): 257-286.
- [5] 杨行峻, 迟惠生等. 语音信号数字处理[M]. 北京: 电子工业出版社, 1995.
- [6] 靳双燕. 基于隐马尔可夫模型的语音识别技术研究[D]. 郑州大学, 2013.
- [7] 耿立波, 酆格斐, 詹卫东, 等. 中国计算语言学研究现状与展望[J]. 语言科学, 2021, 20(05): 491-499.
- [8] 柳春. 语音识别技术研究进展[J]. 甘肃科技, 2008(09): 41-43.
- [9] 禹琳琳. 语音识别技术及应用综述[J]. 现代电子技术, 2013, 36(13): 43-45. DOI: 10.16652/j.issn.1004-373x.2013.13.022.
- [10] 李森. 基于视觉和语音识别的护理机器人交互系统设计[D]. 哈尔滨工业大学, 2022. DOI: 10.27061/d.cnki.ghgdu.2022.001631.
- [11] 肖爱民. 基于语音识别技术的智能家居控制系统的设计[D]. 南昌大学, 2018.
- [12] Dahl G E, Yu D, Deng L, et al. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition[J]. IEEE Transactions on Audio, Speech, and Language Processing, 2012, 20(1): 30-42.
- [13] 赵力. 语音信号处理[M]. 北京: 机械工业出版社, 2003.
- [14] Zaremba W, Sutskever I, Vinyals O. Recurrent Neural Network Regularization[J]. arXiv: Neural and Evolutionary Computing, 2014.
- [15] Colin Champion, S.M. Houghton. Application of continuous state Hidden Markov Models to a classical problem in speech recognition[J], Computer Speech & Language 2016 36.
- [16] 魏扬. 面向嵌入式设备的命令词识别方法研究[D]. 北京交通大学, 2023. DOI: 10.26944/d.cnki.gbfju.2022.002058.22
- [17] 语法开发指南[EB/OL]. [2014-03-31]. <https://bbs-1257412061.cos.ap-nanjing.myqcloud.com/public/attachments/201404/01/205142xflkfch6idfhdw62u.attach>.
- [18] 张鹤鸣, 邓军. 一种低成本语音识别解决方案[J]. 通信技术, 2019, 52(12): 2893-2896.
- [19] Koenig N, Howard A. Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator[C]// Proceedings of IEEE International Conference on Intelligent Robots and Systems, 2004: 2149-2154.