机器人操作系统 ROS 大作业

利用 LLM 的机器人任务规划代理程序

Robotic task planning agent programme using LLM

大连理工大学

Dalian University of Technology

摘　　要

本项目旨在探索大语言模型（LLM）在机器人任务规划中的应用，特别是利用 LLM 控制机器人完成家庭任务。项目选用了 ROS2 作为机器人操作系统，TurtleBot4 作为实验平台，通过结合视觉系统、状态跟踪和任务规划等技术，展示了 LLM 在复杂任务规划中的潜力。项目的主要成果包括成功配置和运行 ROS2，并使用 TurtleBot4 机器人进行实验；采用 YOLOv8 目标检测模型，结合深度相机，实现对环境中物品的检测和定位；设计并实现状态服务，能够检测、跟踪和管理物品的位置，并提供多种查询接口支持任务规划；利用 ChatGPT4o 生成具体的任务计划，通过自然语言指令控制机器人完成物品获取任务；通过导航和制图模块实现环境的初始映射，并设计探索服务，帮助机器人在环境中进行自主探索；采用经典计算机视觉算法，实现房间的分割与标记，并通过 ChatGPT4o 对房间进行命名和分类。通过多项实验，项目验证了 LLM 在复杂任务规划中的有效性和潜力，展示了其在家庭任务中的应用前景。项目的成功为未来研究提供了新的方向，特别是在家庭、医疗和工业等领域的机器人应用。

关键词：ROS2、大语言模型（LLM）、机器人任务规划、自然语言处理（NLP）、路径规划

Robotic task planning agent programme using LLM

Abstract

This project aims to explore the application of large language models (LLM) in robotic task planning, especially the use of LLM to control robots to complete household tasks. The project selected ROS2 as the robot operating system and TurtleBot4 as the experimental platform to demonstrate the potential of LLM in complex task planning by combining technologies such as vision systems, state tracking, and task planning. The main achievements of the project include the successful configuration and operation of ROS2 and the use of TurtleBot4 robots for experiments; the use of YOLOv8 object detection model, combined with depth camera, to realize the detection and positioning of items in the environment; the design and implementation of status services, which can detect, track and manage the location of items, and provide multiple query interfaces to support task planning; the use of LLM to generate specific task plans, and control the robot to complete the task of object acquisition through natural language instructions; the initial mapping of the environment through the navigation and mapping module, and the design of exploration services to help the robot to explore autonomously in the environment; the use of classical computer vision algorithms to realize the segmentation and labeling of rooms, and the naming and classification of rooms through LLM. Through a number of experiments, the project validated the effectiveness and potential of LLM in complex task planning, demonstrating its application prospects in household tasks. The success of the project provides new directions for future research, especially in the field of robotics in the home, medical and industrial fields.

Key Words： ROS2, Large Language Modelling (LLM), Robotic Task Planning, Natural Language Processing (NLP), Path Planning

目　　录

摘　　要----------I
Abstract----------II
引　　言---------- 1
1. 项目背景和动机---------- 1
2. 相关工作概述---------- 1
1　项目背景----------2
1.1　大语言模型（LLM）简介----------2
1.2　任务规划和路径规划的基本概念----------3
2 实现过程---------- 5
2.1　系统架构---------- 5
2.1.1　硬件配置----------5
2.1.2　软件架构----------5
2.2　环境配置---------- 6
2.2　任务规划流程---------- 7
2.2.1　自然语言输入解析----------8
2.2.2　任务分解与规划----------8
2.2.3　任务分解与步骤生成----------8
2.2.4 任务上下文管理---------- 9
2.2.5 多轮对话与用户交互----------9
2.2.6 任务完成报告---------- 9
2.3　实现基本功能----------10
2.3.1　交互处理---------- 10
2.3.2　状态跟踪---------- 10
2.3.3　视觉模拟实现---------- 15
3 实验设计----------18
3.1　实验环境----------18
结　　论----------20
参 考 文 献---------- 21

引　　言

1. 项目背景和动机

在现代科技飞速发展的今天，人工智能（AI）和机器人技术正逐渐改变我们的生活方式。特别是大语言模型（LLM）的出现，如 OpenAI 的 GPT 系列，展示了强大的自然语言处理和理解能力。这些模型不仅能够生成高质量的文本，还能够理解和执行复杂的任务指令。将 LLM 应用于机器人任务规划中，能够显著提升机器人的智能化水平，使其能够更好地理解和执行人类的指令，从而在家庭、医疗、工业等多个领域发挥更大的作用。本项目的动机源于希望探索 LLM 在机器人任务规划中的潜力，特别是其在家庭任务中的应用。通过控制机器人完成家庭中的物品获取任务，展示 LLM 在复杂任务规划中的优势，并为未来的研究和应用提供新的方向。

2. 相关工作概述

在自然语言处理领域，LLM 如 GPT 系列展示了前所未有的能力，能够生成高质量的文本并理解复杂的任务指令。在机器人任务规划中，传统的方法主要依赖于预定义的规则和路径规划算法，而 LLM 的引入为这一领域带来了新的可能性。结合 LLM 与机器人技术的研究仍处于起步阶段，但已有一些前沿研究展示了其潜力。例如，已有研究尝试将 LLM 应用于机器人导航和任务执行，通过自然语言指令生成路径规划和任务计划。此外，一些研究还探索了结合视觉系统和状态跟踪技术，实现对环境中物品的检测和定位。然而，这些研究大多集中在单一任务或特定环境中，缺乏对复杂任务和多样化环境的全面探索。本项目在前人研究的基础上，进一步探索了 LLM 在复杂任务规划中的应用，特别是其在家庭任务中的潜力。通过结合 ROS2、视觉系统、状态跟踪和任务规划等技术，展示了 LLM 在机器人任务规划中的优势，并为未来的研究和应用提供了新的方向。

1 项目背景

1.1 大语言模型（LLM）简介

大语言模型（LLM）是近年来人工智能领域的重大突破之一。以 OpenAI 的 GPT 系列为代表，这些模型通过训练海量的文本数据，展示了卓越的自然语言处理和生成能力。LLM 能够理解复杂的语言结构和语义关系，生成高质量的文本，并执行各种自然语言任务，如翻译、摘要、问答等。其强大的语言理解能力使其在多个领域展现出广泛的应用前景，包括文本生成、对话系统、内容推荐等。

LLM 的核心在于其深度神经网络架构，通常包含数十亿甚至上千亿个参数。这些模型通过大规模的无监督学习，从海量的文本数据中学习语言的统计特性和语义关系。训练过程中，模型会通过预测下一个词或填补句子中的空白来优化自身的参数，从而逐步掌握语言的复杂结构和语义。

在实际应用中，LLM 不仅能够生成连贯且富有创意的文本，还能够理解上下文，进行复杂的推理和决策。例如，GPT-3 能够根据上下文生成长篇文章、写代码、回答问题，甚至进行多轮对话。这些能力使得 LLM 在各种自然语言处理任务中表现出色，并引发了广泛的关注和研究。

本项目旨在探索 LLM 在机器人任务规划中的应用，特别是其在家庭任务中的潜力。通过控制机器人完成家庭中的物品获取任务，展示 LLM 在复杂任务规划中的优势，并为未来的研究和应用提供新的方向。具体来说，LLM 在本项目中扮演了核心角色，通过自然语言指令生成具体的任务计划，并结合视觉系统和状态跟踪，实现对环境中物品的检测和定位。

首先，LLM 能够理解用户的自然语言指令，并将其转化为具体的任务计划。例如，当用户要求机器人“从厨房拿一瓶水”时，LLM 能够解析指令中的关键元素，如“厨房”、“拿”和“水”，并生成相应的任务步骤。其次，LLM 能够结合视觉系统和状态跟踪，实时更新环境中的物品信息，确保任务计划的准确性和执行效果。最后，LLM 还能够通过多轮对话与用户进行交互，提供任务状态的反馈和调整建议，提升用户体验。

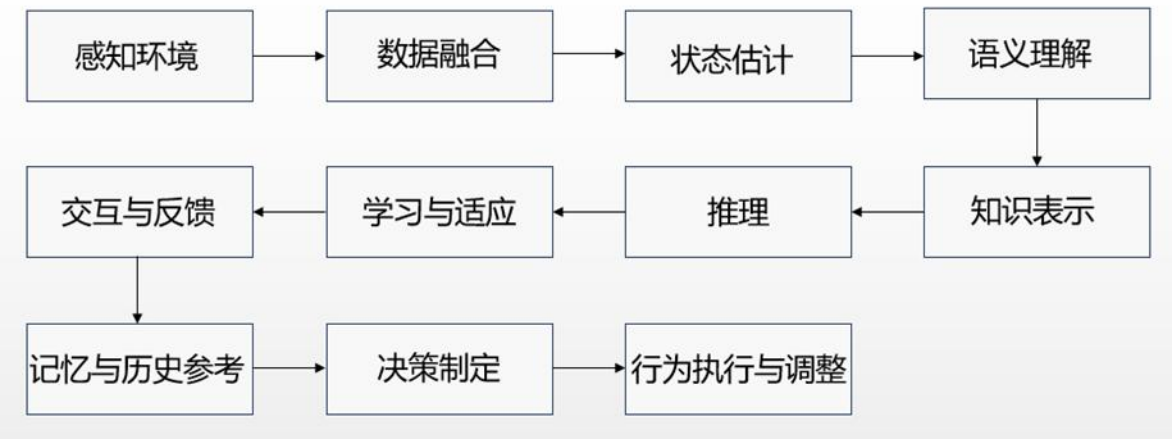

图 1　LLM 提供上下文理解原理

1.2　任务规划和路径规划的基本概念

任务规划和路径规划是机器人学中的两个核心概念。任务规划是指为机器人制定一系列步骤或操作，以完成特定任务的过程。这通常涉及任务的分解、步骤的排序以及资源的分配等。任务规划的目标是确保机器人能够高效、准确地完成指定任务。路径规划则是指在已知环境中为机器人找到从起点到终点的最优路径的过程。路径规划需要考虑机器人自身的运动约束、环境中的障碍物以及其他动态因素。传统的路径规划算法包括 A*算法、Dijkstra 算法等，而现代路径规划还结合了机器学习和优化技术，以提高规划效率和适应复杂环境的能力。

在本项目中，路径规划是机器人任务执行过程中的关键环节。路径规划的目标是为机器人找到从起点到目标点的最优路径，确保机器人能够高效、安全地完成任务。为了实现这一目标，本项目采用了一系列先进的路径规划技术和算法，并结合实际环境中的动态因素进行优化。

首先，项目使用了 ROS2 中的 navigation2 和 cartographer 模块。这些模块提供了强大的路径规划和地图构建功能，能够帮助机器人在未知环境中进行自主导航。navigation2 模块采用了 Dijkstra 和 A*等经典路径规划算法，能够在已知地图上找到最优路径。制图模块则通过激光雷达和深度相机等传感器数据，实时构建环境地图，帮助机器人识别障碍物和空闲区域。

在路径规划过程中，机器人首先需要进行初始环境映射。通过制图模块，机器人能够在探索过程中逐步构建环境的占据网格地图。这张地图将环境分割成若干小格子，每个格子标记为已知、未知或障碍物。机器人在探索过程中，不断更新这张地图，以获取更准确的环境信息。

为了提高路径规划的效率和可靠性，本项目还设计了一个探索服务（explorer）。该服务通过订阅机器人的位置和地图数据，定期进行环境探索，并选择最佳的探索目标点。探索服务使用 A*搜索算法，在当前地图上找到最优的探索路径。为了避免机器人陷入死胡同或重复探索，探索服务还引入了一些额外的规则和过滤机制。例如，检查目标点是否在已知区域的边界附近，避免选择过于狭窄或无法通过的路径。

在实际任务执行过程中，机器人需要根据任务规划生成的任务步骤，逐步移动到目标位置。路径规划模块会根据当前地图和目标位置，计算出一条最优路径，并生成一系列运动指令。机器人按照这些指令移动，并通过传感器实时监测环境，避免碰撞和障碍物。如果在执行过程中发现新的障碍物，路径规划模块会自动重新计算路径，确保任务的顺利完成。

此外，为了提高路径规划的鲁棒性，本项目还结合了机器学习和优化技术。例如，使用深度强化学习算法，帮助机器人在复杂环境中学习更优的路径规划策略；使用遗传算法和粒子群优化算法，优化路径规划的参数和权重，提高规划效率和精度。

2 实现过程

2.1 系统架构

2.1.1 硬件配置

本项目的硬件配置主要由以下几个关键组件组成：

1. TurtleBot4 机器人：TurtleBot4 是一款广泛应用于教育和研究的移动机器人平台。它配备了多种传感器，包括激光雷达（LiDAR）、深度相机、惯性测量单元（IMU）等，能够在复杂环境中进行自主导航和任务执行。TurtleBot4 的模块化设计使其易于扩展和定制，适合用于本项目的实验需求。
2. 传感器：TurtleBot4 配备了多种传感器，用于环境感知和状态跟踪。激光雷达（LiDAR）用于构建环境的二维占据网格地图，深度相机用于检测和定位物品，惯性测量单元（IMU）用于姿态估计和运动控制。这些传感器的数据通过 ROS2 进行处理和融合，为路径规划和任务执行提供支持。

2.1.2 软件架构

本项目的软件架构基于 ROS2，结合了多个开源库和自定义模块，实现了从任务规划到路径规划和任务执行的完整流程。软件架构包括以下几个主要部分：

1. ROS2 框架：ROS2 是一个开源的机器人操作系统，提供了丰富的工具和库，用于机器人应用的开发和部署。ROS2 的模块化设计和强大的通信机制，使得不同功能模块之间能够高效地进行数据交换和协同工作。本项目中，ROS2 用于管理传感器数据、执行路径规划、控制机器人运动等。
2. 任务规划模块：任务规划模块基于大语言模型（LLM），负责解析用户的自然语言指令，并生成具体的任务计划。LLM 通过深度学习模型，理解指令中的关键元素，并将其转化为一系列可执行的任务步骤。任务规划模块还负责与其他模块进行通信，确保任务的顺利执行。在这里我们使用 ChatGPT4o 的 token 作为本项目的 LLM。
3. 路径规划模块：路径规划模块使用 ROS2 中的导航 2（navigation2）和制图（cartographer）功能，负责在已知环境中找到从起点到目标点的最优路径。导航 2 模块采用 Dijkstra 和 A*等经典路径规划算法，结合激光雷达和深度相机的数据，实时更新环境地图，并生成运动指令。路径规划模块还包括一个探索服务（explorer），帮助机器人在未知环境中进行自主探索。

4. 视觉系统模块：视觉系统模块使用 YOLOv8 目标检测模型，结合深度相机，实现对环境中物品的检测和定位。视觉系统模块通过 ROS2 的通信机制，将检测到的物品信息发送给状态跟踪模块，并与任务规划模块进行交互，确保任务的准确执行。
5. 控制模块：控制模块负责接收路径规划模块生成的运动指令，并控制机器人的实际运动。控制模块通过 ROS2 的通信机制，与机器人底层硬件进行交互，确保机器人按照规划的路径移动，并避开障碍物。

2.2 环境配置

2.2.1 设置 Locale

```
sudo apt update
sudo apt install -y locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8export
LANG=en_US.UTF-8
```

2.2.2 添加额外的仓库

```
sudo apt install -y software-properties-common
sudo add-apt-repository universe
sudo apt update
sudo apt install -y curl
curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key |
sudo tee /usr/share/keyrings/ros-archive-keyring.gpg >
/dev/nullecho "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee
/etc/apt/sources.list.d/ros2.list > /dev/null
sudo apt update
sudo apt upgrade -y
```

2.2.3 安装 ROS2

```
sudo apt install -y build-essential
sudo apt install -y ros-$ROS_DISTRO-ros-core
sudo apt install -y python3-colcon-common-extensions
```

2.2.4 配置 ROS 环境

```
echo 'source /opt/ros/$ROS_DISTRO/setup.bash' >>
/home/vagrant/.bashrcecho 'source
/opt/ros/$ROS_DISTRO/setup.bash' | sudo tee -a /root/.bashrcecho
"set +e" >> /home/vagrant/.bashrc
```

2.2.5 安装 Navigation2

```
sudo apt install -y ros-$ROS_DISTRO-navigation2
sudo apt install -y ros-$ROS_DISTRO-nav2-bringup
sudo apt install -y ros-$ROS_DISTRO-slam-toolbox
sudo apt install -y ros-$ROS_DISTRO-tf-transformations
sudo apt install -y ros-$ROS_DISTRO-rmw-cyclonedds-cppecho 'export
RMW_IMPLEMENTATION=rmw_cyclonedds_cpp' >> /home/vagrant/.bashrc
```

2.2.6 *可直接运行 install.bash

```
# 确保 install.bash 文件在当前目录下
sudo bash install.bash
```

2.2 任务规划流程

任务规划流程是本项目的核心部分，通过将用户的自然语言指令转化为具体的任务计划，并结合路径规划和状态跟踪，实现机器人的自主任务执行。

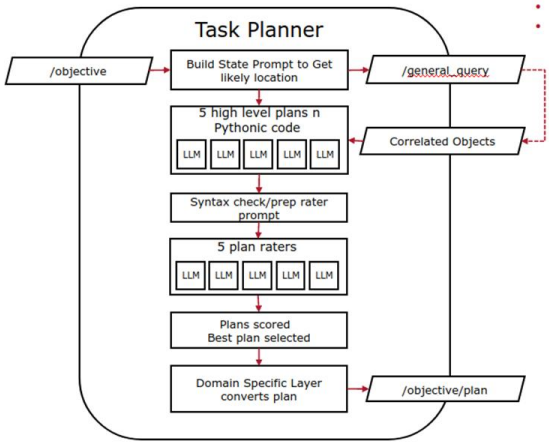

图 2　任务规划六步骤

2.2.1　自然语言输入解析

任务规划流程的第一步是解析用户的自然语言指令。用户通过语音或文本输入向机器人发出指令，例如“从厨房拿一瓶水”。任务规划模块基于大语言模型（LLM），能够理解指令中的关键元素，如“厨房”、“拿”和“水”。

1. 语音识别：如果用户通过语音输入指令，首先需要进行语音识别。语音识别模块将语音信号转化为文本，传递给 LLM 进行进一步处理。
2. 自然语言理解：LLM 解析文本指令，提取出任务的关键元素和意图。LLM 能够理解指令的语义，并生成一个初步的任务描述。

2.2.2　任务分解与规划

预处理后的文本指令需要通过自然语言理解（NLU）技术进行解析，以提取出任务的关键元素和意图。NLU 在本项目中的具体使用如下：

1. 意图识别：使用预训练的大语言模型（GPT-4o）来识别用户指令的意图。例如，指令“从厨房拿一瓶水”的意图是“获取物品”。
2. 实体识别：识别指令中的关键实体，如地点（“厨房”）、物品（“水”）等。实体识别可以通过命名实体识别（NER）技术实现。
3. 依存解析：通过依存句法解析技术，分析指令中的句法结构，确定各个词语之间的关系。例如，确定“拿”是动词，“厨房”和“水”分别是地点和物品。

2.2.3　任务分解与步骤生成

在理解用户指令后，需要将任务分解为一系列具体的步骤。大语言模型（LLM）在此过程中发挥关键作用：

1. 任务分解：LLM 将复杂的任务分解为多个子任务。例如，“从厨房拿一瓶水”可以分解为“前往厨房”、“找到水瓶”、“抓取水瓶”、“返回用户位置”等子任务。
2. 步骤生成：对于每个子任务，LLM 生成具体的执行步骤。这些步骤包括机器人需要执行的动作和路径规划的目标位置。例如，“前往厨房”步骤包括导航到厨房的路径规划，“找到水瓶”步骤包括使用视觉系统检测水瓶的位置。

2.2.4 任务上下文管理

为了确保任务的顺利执行，机器人需要管理任务的上下文信息。这包括跟踪当前任务的状态、记录已完成的步骤和待完成的步骤等：

1. 任务状态跟踪：使用状态跟踪模块记录每个任务的执行状态，包括已完成的步骤和待完成的步骤。
2. 上下文更新：在任务执行过程中，实时更新任务的上下文信息。例如，当机器人到达厨房时，更新任务状态为“已到达厨房”，并开始执行下一步“找到水瓶”。

2.2.5 多轮对话与用户交互

在任务执行过程中，机器人需要与用户进行多轮对话，提供任务状态的反馈和调整建议：

1. 状态反馈：机器人通过自然语言生成（NLG）技术，向用户报告任务的当前状态。例如，“我已经到达厨房，现在正在寻找水瓶。”
2. 用户交互：如果在任务执行过程中遇到问题，机器人可以向用户请求帮助或调整任务计划。例如，“我找不到水瓶，请问它通常放在哪里？”

2.2.6 任务完成报告

当所有任务步骤执行完毕后，机器人需要向用户报告任务的完成情况，并提供任务总结：

1. 任务总结：生成任务总结报告，包含任务的执行步骤、遇到的问题和解决方案等信息。
2. 用户报告：通过自然语言生成（NLG）技术，向用户报告任务的完成情况。例如，“我已经从厨房拿回了一瓶水。”

2.3 实现基本功能

我们基于 ROS2 的 colcon 构建系统简单实现了 llm、explorer、robot、planner、room_segmentation 模块，其中主要贡献为使用 opencv 优化 room_segmentation 并联合整个系统进行调优。

2.3.1 路径规划

我们使用一种名为 litterbug 的服务，它能够感知模拟环境中的物品和生成地图，跟踪它们，并在机器人 "抓取 "它们时生成路径和将它们从环境中移除。

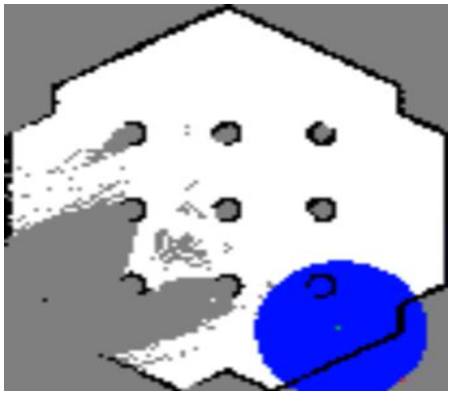

图 3 litterbug 根据地图状态确定机器人探索最佳地点

2.3.2 基于 opencv 分水岭算法的改进

在本项目中，我们需要将生成的占据地图（occupancy map）进行房间分割和标记，以便为大语言模型（LLM）代理提供状态上下文信息。具体来说，我们希望确定给定位置属于哪个房间，并将物品与其所在的房间关联起来。为了解决这个问题，我们采用了经典的计算机视觉流水线，而不是深度学习模型，因为时间限制使得生成足够大的数据集来训练深度学习模型变得不现实。

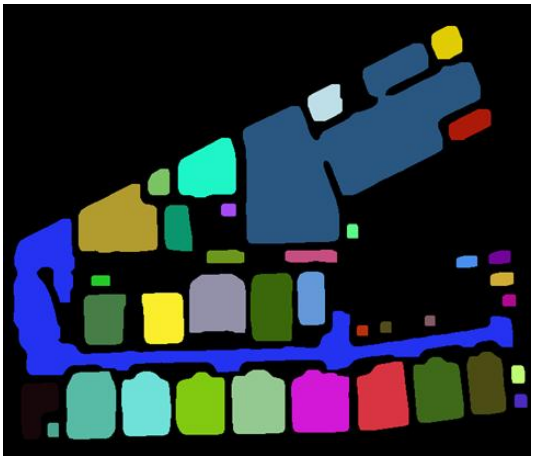

图 4 通过 opencv 分水岭算法将房间分割

最初的尝试是使用轮廓检测方法对占据地图进行房间分割。然而，结果并不理想。轮廓检测未能形成完整的形状，导致大多数房间无法正确检测，且某些房间仅检测到房间的轮廓，分割效果非常不稳定。我们尝试了形态学操作（如腐蚀和膨胀）和距离阈值化的方法，结果有所改进，但仍然不理想，特别是分割房间的边界较厚。

经过进一步研究，我们发现了一些利用分水岭算法进行房间分割的论文，并且 OpenCV 中内置了对分水岭算法的支持。虽然仍需进行一些实验来微调设置，但结果非常有希望。

分水岭算法是一种基于形态学的图像分割算法，适用于处理具有复杂边界的图像。它将图像视为拓扑表面，将每个像素视为一个点，高度由像素值决定。算法通过模拟水从低处向高处漫延的过程，将图像分割成不同的区域。

以下是我们使用分水岭算法进行房间分割和标记的详细流程：

a. 预处理：

读取占据地图并进行灰度化处理。使用高斯模糊平滑图像，减少噪声对分割结果的影响。

b. 距离变换和阈值化：

应用距离变换，将二值图像中的前景对象转化为距离图。对距离图进行阈值化，生成标记图像。

c. 分水岭算法：

使用 OpenCV 的 cv2.watershed 函数进行分水岭变换，生成分割后的图像。将分割结果中的边界标记为不同的颜色，以便于可视化和进一步处理。

d. 后处理：

检测分割房间的面积，如果某个房间的面积低于可配置的阈值（例如房间中间的沙发区域），则将其合并到周围的房间中。计算每个房间的质心，并将其添加到状态服务中。

e. 物品与房间关联：

在探索过程中，记录所有检测到的物品。将每个物品的位置转换为地图上的坐标，并与分割后的房间标签进行比较，确定物品所属的房间。

f. 房间标签生成：

通过查询状态服务，获取每个房间中的物品列表。使用 LLM 生成房间标签。例如，如果一个房间中有床和衣柜，则将其标记为卧室；如果有沙发和电视，则标记为客厅。

g. 服务接口：

状态服务提供两个新的 ROS 服务：一个用于查询房间的质心位置，另一个用于查询给定坐标位置所属的房间。状态服务还使用分割后的地图自动为新检测到的物品标记房间标签。

具体代码思想如下：

```
import cv2
import numpy as np
from random import randint
from typing import Optional, Tuple, List, Union
from skimage.transform import resize


def segment_rooms(
    image: np.ndarray,
    blur_kernel_size: int = 7,
    noise_threshold: int = 25,
    apply_mask: bool = True,
    dist_threshold_min: float = 0.3,
    dist_threshold_max: float = 1.0,
    resize_dim: int = 500,
    min_area: int = -1,
) -> np.ndarray:
    """
    使用分水岭算法进行房间分割。通过距离变换和形态学操作，
    尝试识别图像中的房间区域。
    """
    img = image.copy()
    if img.ndim == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    original_height, original_width = img.shape
    img = cv2.resize(img, (resize_dim, int(
        resize_dim * img.shape[0] / img.shape[1])))
    blurred_img = cv2.GaussianBlur(
        img, (blur_kernel_size, blur_kernel_size), 0)
    _, binary_img = cv2.threshold(
        blurred_img, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
    if apply_mask:
        mask = np.zeros_like(img)
        contours, _ = cv2.findContours(
            binary_img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        for contour in contours:
            if cv2.contourArea(contour) > noise_threshold:
                cv2.fillPoly(mask, [contour], 255)
        img[mask == 0] = 0
    laplacian_kernel = np.array(
        [[1, 1, 1], [1, -8, 1], [1, 1, 1]], dtype=np.float32)
    laplacian = cv2.filter2D(img, cv2.CV_32F, laplacian_kernel)
    img = np.float32(img) - laplacian
    img = np.clip(img, 0, 255).astype("uint8")
    _, binary_threshold = cv2.threshold(
        img, 40, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

```
    distance_transform = cv2.distanceTransform(
        binary_threshold, cv2.DIST_L2, 3)
    cv2.normalize(distance_transform, distance_transform,
                  0, 1.0, cv2.NORM_MINMAX)
    _, dist_thresholded = cv2.threshold(
        distance_transform, dist_threshold_min, dist_threshold_max, cv2.THRESH_BINARY)
    kernel = np.ones((3, 3), dtype=np.uint8)
    dilated = cv2.dilate(dist_thresholded, kernel)
    contours, _ = cv2.findContours(dilated.astype(
        "uint8"), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    markers = np.zeros_like(dilated, dtype=np.int32)

    for idx, contour in enumerate(contours):
        cv2.drawContours(markers, contours, idx, color=(idx + 1), thickness=-1)
    cv2.circle(markers, (5, 5), 3, (255, 255, 255), -1)
    cv2.watershed(cv2.cvtColor(img, cv2.COLOR_GRAY2BGR), markers)
    markers = resize(markers, (original_height, original_width),
                     order=0, preserve_range=True, anti_aliasing=False)

    if min_area > 0:
        markers = filter_small_zones(markers, min_area=min_area)

    return markers


def visualize_segments(
    markers: np.ndarray, colors: Optional[List[Tuple[int]]] = None, show_labels: bool
= False
) -> np.ndarray:
    """
    根据分割标记生成彩色图像。可以提供颜色列表，如果设置了 show_labels，则在每个区域中心标记区域索
引。
    """
    colored_img = np.zeros_like(markers, dtype=np.uint8)
    colored_img = cv2.cvtColor(colored_img, cv2.COLOR_GRAY2BGR)

    unique_markers = np.unique(markers)
    unique_markers = unique_markers[(
        unique_markers > 0) & (unique_markers < 255)]

    if colors is None:
        colors = {index: (randint(0, 255), randint(0, 255),
                          randint(0, 255)) for index in unique_markers}

    for index in unique_markers:
        colored_img[markers == index] = colors[index]

    if show_labels:
        for index in unique_markers:
            y_coords, x_coords = np.where(markers == index)
```

```
            y_center = int(np.mean(y_coords))
            x_center = int(np.mean(x_coords))
            cv2.putText(
                colored_img,
                str(index + 1),
                (x_center, y_center),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5,
                (0, 0, 0),
                1,
                cv2.LINE_AA,
            )

    return colored_img

def filter_small_zones(markers: np.ndarray, min_area: float = 1500.0) -> np.ndarray:
    """
    对分割标记进行阈值处理，消除面积小于指定阈值的区域，并将其设置为周围最常见的区域标签。
    """
    markers = markers.copy()
    unique_markers = np.unique(markers)
    unique_markers = unique_markers[(
        unique_markers > 0) & (unique_markers < 255)]

    for index in unique_markers:
        mask = (markers == index)
        y_coords, x_coords = np.where(mask)
        y_min, y_max = np.min(y_coords), np.max(y_coords)
        x_min, x_max = np.min(x_coords), np.max(x_coords)

        area = (y_max - y_min) * (x_max - x_min)
        if area < min_area:
            y_min = max(int(y_min - (y_max - y_min) * 0.1), 0)
            y_max = min(int(y_max + (y_max - y_min) * 0.1), markers.shape[0])
            x_min = max(int(x_min - (x_max - x_min) * 0.1), 0)
            x_max = min(int(x_max + (x_max - x_min) * 0.1), markers.shape[1])

            surrounding = markers[y_min:y_max, x_min:x_max]
            surrounding = surrounding[(surrounding != index) & (
                surrounding > 0) & (surrounding < 255)]

            if len(surrounding) == 0:
                continue

            most_common = np.bincount(surrounding).argmax()
            markers[mask] = most_common

    return markers
```

```python
def identify_zones(
    img: np.ndarray,
    unknown_label: Union[int, Tuple[int, int, int]],
    empty_label: Union[int, Tuple[int, int, int]],
):
    """
    接受图像并尝试将其分割为已知、未知和其他（基本上是墙壁）区域。
    """
    base = np.ones_like(img)
    base = np.where(img == unknown_label, -1, base)
    base = np.where(img == empty_label, 0, base)
    return base


if __name__ == "__main__":
    img = cv2.imread("./house.jpeg", cv2.IMREAD_GRAYSCALE)
    segmented_markers = segment_rooms(img)
    filtered_markers = filter_small_zones(segmented_markers)
    final_image = visualize_segments(segmented_markers, show_labels=True)
    filtered_image = visualize_segments(filtered_markers, show_labels=True)

    cv2.imshow("Final Segmentation", final_image)
    cv2.imshow("Filtered Segmentation", filtered_image)
    cv2.waitKey()
    print(np.unique(filtered_markers, return_counts=True))

    np.save("house_segmentation.npy", filtered_markers, allow_pickle=False)
```

2.3.3 视觉模拟实现

我们使用 YOLOv8 作为 ROS 视觉的解决方案，YOLO 可以检测物体边角，转换为 x/y 坐标，最后将位置转换为真实世界帧，即真实位置。

图 5　yolov8 不断识别物品并添加标签

2.3.3 基于 LLM 的任务规划

在本项目中，为了实现机器人任务的灵活规划和执行，我们设计了一个动作抽象类（Action）来处理异步取消和结果分配的功能。这个设计的目标是将动作的执行与任务规划解耦，使得任务规划器能够专注于高层次的任务逻辑，而不必关心具体的动作执行细节。

动作抽象类（Action）封装了与异步取消和结果分配相关的功能。例如，如果一个动作原语（如移动机器人）由于某种原因被取消，这个取消操作可以在不影响任务规划器的情况下进行处理。同样，动作的结果（如在最近的移动过程中是否发现了某个物品）可能会在未知的时间通过另一个通道传入，需要以一种安全的方式报告给任务规划器。

每个具体的动作都可以扩展这个抽象类，并通过一个通用的处理器（ActionPlanner）执行代码。ActionPlanner 将生成的代码中的动作映射到函数名称，从而实现动作的统一管理和执行。

在设计动作时，我们创建了一组可以组合成更复杂任务的小动作。为了让大语言模型（LLM）能够利用这些动作，我们需要定义动作的语法，并在提示中明确告知 LLM 其功能。以下是 LLM 可以使用的函数及其描述：

- move_to_object(object) → (bool, str)：移动到指定物品。
- move_to_room(room) → (bool, str)：移动到指定房间。
- move_to_human() → (bool, str)：移动到人类最后已知的位置。
- pickup_object(object) → bool：拾取指定物品。
- give_object(object) → bool：将物品交给人类。
- do_i_see(object) → bool：检查是否看到指定物品。
- look_around_for(object) 或 look_around_for([object, object, ...]) → [(label: str, distance: float)]：原地旋转寻找物品。
- complete()：动作成功完成时返回。
- fail()：动作失败时返回。

通过上述动作抽象类和具体动作的设计，我们能够实现复杂任务的灵活规划和执行。任务规划器可以调用这些动作函数，通过异步方式执行具体的动作，并根据动作的结果调整任务计划。例如：

1．导航到物品：任务规划器可以调用 move_to_object 函数，导航机器人到指定物品的位置。如果导航失败，任务规划器可以根据失败原因调整计划，例如重新计算路径或尝试其他方法找到物品。

2．拾取物品：到达物品位置后，任务规划器调用 pickup_object 函数尝试拾取物品。如果拾取失败，任务规划器可以检查物品是否在视野范围内，并采取相应的措施。

3．交付物品：成功拾取物品后，任务规划器调用 give_object 函数，将物品交给人类。如果交付失败，任务规划器可以重新定位人类位置并再次尝试交付。

3 实验设计

3.1 实验环境

实验在 ROS2 仿真环境中进行，使用 Gazebo 模拟真实的机器人操作场景。Gazebo 是一个强大的机器人仿真工具，它可以模拟机器人在各种环境和条件下的行为和传感器输出，Rviz 是一个可视化工具，用于在 ROS 中显示和交互机器人的数据，二者通常配合使用，在 Gazebo 中设置仿真环境和机器人模型，进行任务规划和测试，然后通过 Rviz 实时监控仿真过程中的机器人状态和传感器输出，进行调试和优化。实验环境包括一个带有多个障碍物和目标物品的房间，机器人在该环境中执行任务。

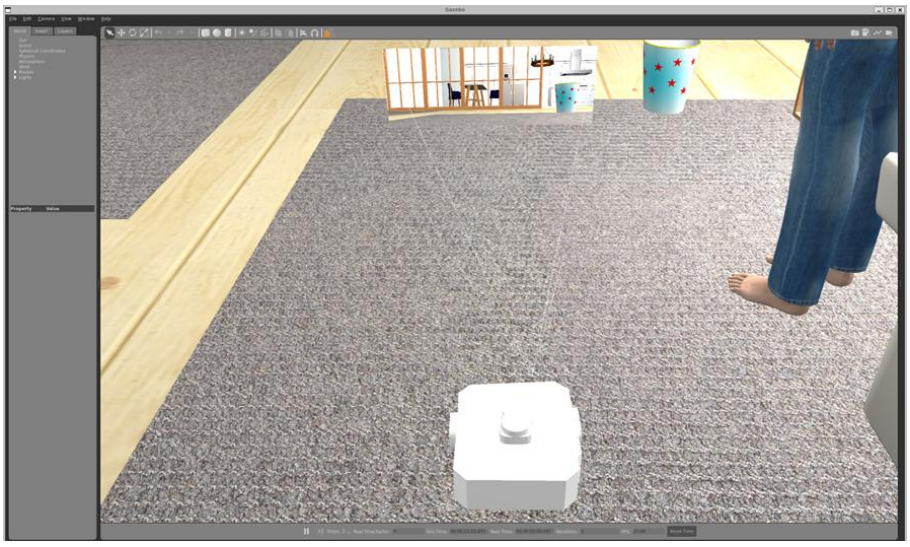

图 6　Gazebo 演示截图

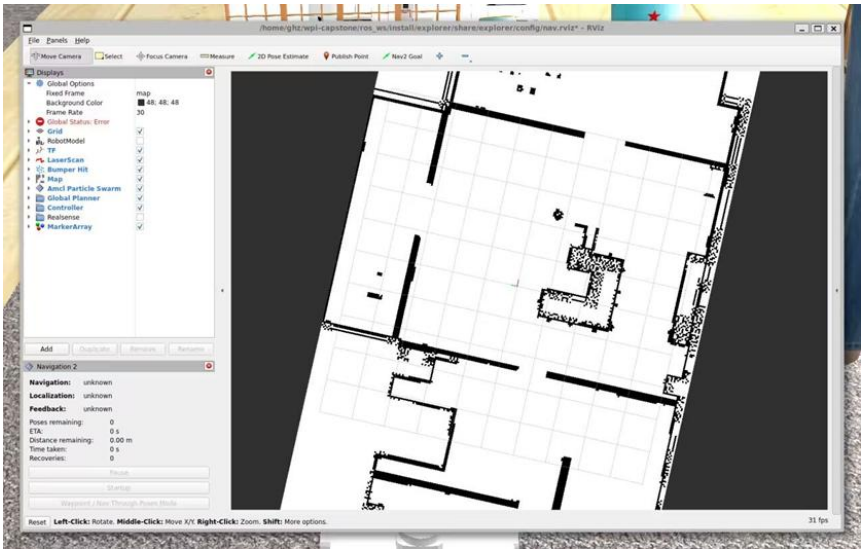

图 7　Rivz 演示截图

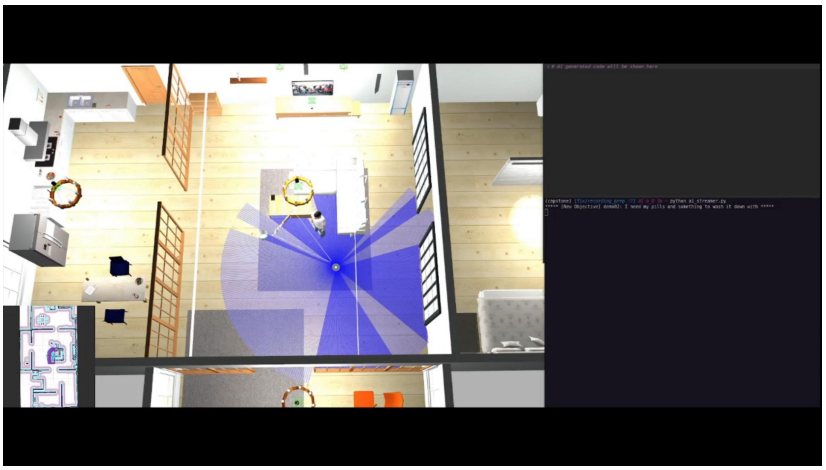

图 8　结果展示

左侧为模拟，左下角为导航地图，其中包括了激光雷达。右上角是任务计划的生成代码，右下角是 AI 操作的反馈

3.2　结果分析

分析结果表明，LLM 在任务规划中的应用显著提高了机器人的智能化程度和任务执行的成功率。与传统的任务规划方法相比，LLM 能够更好地理解复杂的自然语言指令，生成合理的任务执行计划。此外，结合强化学习和控制模型的应用，进一步增强了机器人在复杂环境中的适应能力。通过自然语言处理，机器人能够快速解析任务指令，减少了人工干预的需求；通过强化学习，机器人能够在动态环境中不断优化路径选择和任务执行策略，提高了任务完成的效率和成功率；通过多种控制模型的集成，机器人能够实现精确的动作执行，适应复杂的操作环境。

项目的成功之处在于验证了 LLM 在机器人任务规划中的应用可行性，展示了其在理解和执行复杂任务方面的优势。然而，项目也存在一些不足之处，如 LLM 对环境感知的依赖较大，当传感器数据不准确或缺失时，任务规划和执行可能受到影响。此外，LLM 的计算资源需求较高，对硬件配置提出了较高的要求。

结　论

本项目成功地展示了大语言模型（LLM）在机器人任务规划中的应用潜力。通过结合 ROS2、视觉系统、状态跟踪和路径规划等先进技术，项目实现了机器人在家庭环境中自主完成复杂任务的目标。以下是项目的主要结论和贡献：

本项目证明了大语言模型（LLM）可以有效地解析用户的自然语言指令，并将其转化为具体的任务计划。LLM 的强大语言理解和生成能力，使其能够准确提取指令中的关键元素和意图，并生成一系列可执行的任务步骤。这为机器人在复杂任务规划中提供了强有力的支持。

项目设计并实现了从任务规划到任务执行的完整流程。通过自然语言处理（NLP）技术，机器人能够解析用户指令，并生成具体的任务计划。结合状态跟踪模块和视觉系统，机器人能够实时感知环境变化，并更新任务状态。在路径规划模块的支持下，机器人能够找到从当前位置到目标位置的最优路径，并安全、高效地执行任务。

项目采用了 ROS2 中的 navigation2 和 cartographer 模块，实现了高效、可靠的路径规划功能。结合激光雷达和深度相机的数据，机器人能够实时构建环境地图，并生成最优路径。此外，项目还设计了一个探索服务，帮助机器人在未知环境中进行自主探索，提高了任务执行的灵活性和鲁棒性。

通过多项实验，项目验证了 LLM 在复杂任务规划中的有效性和潜力。实验结果显示，机器人能够自主完成多种家庭任务，包括物品获取、环境探索和状态报告等。项目展示了 LLM 在机器人任务规划中的应用前景，为未来的研究和应用提供了新的方向。

尽管本项目取得了显著的成果，但仍有一些挑战和改进空间。未来工作可以进一步优化任务规划算法，提高任务执行的效率和准确性。此外，可以探索 LLM 与其他人工智能技术的结合，如深度强化学习和多模态感知，以提升机器人的智能化水平。项目的成功为未来在家庭、医疗和工业等领域的机器人应用提供了坚实的基础。

参 考 文 献

[1] hlfshell. (2023). Utilizing LLMs as a Task Planning Agent for Robotics.

[2] Dhiraj, V. (2020). Introduction to ROS2. Retrieved from https://roboticsbackend.com/introduction-to-ros2/

[3] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.

[5] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. The International Journal of Robotics Research, 5(1), 90-98.

[6] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[7] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.