Raybot机器人运动控制软件说明

目录

- 1. 文档说明
  - 1.1 适用范围
  - 1.2 读者对象
- 2. 术语表
- 3. 机器人硬件介绍
  - 3.1 机器人硬件介绍
  - 3.2 坐标系相关知识
- 4. 系统启动与环境配置
  - 4.1 ROS2 通信特别提醒
  - 4.2 启动命令详解
  - 4.3 启动参数配置
- 5. 总体架构
  - 5.1 核心组件与关系
  - 5.2 模块功能对照表
  - 5.3 系统架构图
  - 5.4 完整系统启动
  - 5.5 机械臂全局模式切换
  - 5.6 夹爪 PP 模式与指令控制
- 6. 规划组与预定义姿态
  - 6.1 规划组介绍
    - 6.1.1 规划组关节组成
    - 6.1.2 预定义姿态
- 7. 规划器配置
  - 7.1 支持的规划器类型
  - 7.2 pipeline_id / planner_id 配置说明
  - 7.3 规划失败原因分析
- 8. Arm Motion Controller(核心执行节点)
  - 8.1 模块功能

- 9. CLI Controller(命令行机械臂运动控制)
  - 9.1 模块功能
  - 9.2 命令格式与语法
    - 9.2.1 dual_arms 位姿命令规则
  - 9.3 对外接口说明
  - 9.4 完整测试命令
  - 9.5 实现细节
- 10. Action Recorder(动作编程器)
  - 10.1 模块功能
  - 10.2 配置参数
  - 10.3 对外接口说明
  - 10.4 完整测试命令
  - 10.5 实现细节
  - 10.6 交互流程
- 11. Object Grasp Node(物品抓取节点)
  - 11.1 模块功能
  - 11.2 配置参数
  - 11.3 命令与服务
  - 11.4 完整测试流程
    - 11.4.1 测试环境搭建
    - 11.4.2 测试命令
  - 11.5 实现细节
  - 11.6 交互流程
- 12. 故障排查
  - 12.1 常见错误及解决方案
- 13. 示例命令集合
  - 13.1 ROS2 环境配置命令
  - 13.2 系统启动命令
  - 13.3 CLI 控制器命令
  - 13.4 Action Recorder 命令
  - 13.5 物品抓取节点命令

- 13.6 运动停止指令

文档说明

本文档详细介绍了 Raybot 机器人运动控制软件的架构、模块功能、交互流程、参数配置和使用方法。该软件基于 ROS2 和运动规划框架，实现了对机器人的运动控制、动作编程和物品抓取功能。

适用范围

- Raybot 机器人运动控制
- 动作程序录制与执行
- 物品自动抓取与放置
- ROS2 机器人应用开发

读者对象

- 机器人系统集成工程师
- ROS2 应用开发者
- 机器人操作与维护人员
- 相关领域研究人员

术语表

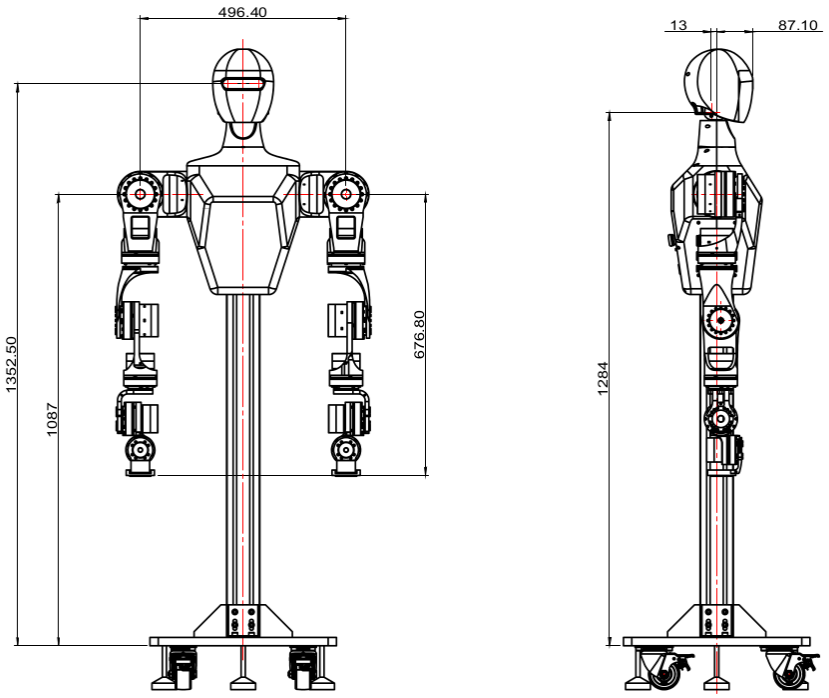
术语	英文全称	解释
ROS2	Robot Operating System 2	第二代机器人操作系统，提供机器人开发所需的库和工具
PTP 运动	Point-to-Point Motion	点对点运动，机械臂末端以最快路径到达目标位置
LIN 运动	Linear Motion	沿直线轨迹运动，适合需要精确路径控制的场景
MoveGroupInterface	MoveGroupInterface	ROS2 中用于控制机械臂运动的高级接口
TF	Transform	坐标变换，用于表示机器人各部件之间的位置关系
OMPL	Open Motion Planning Library	开源运动规划库，提供多种采样规划算法
PILZ	PILZ Industrial Motion	工业级运动规划器，提供PTP、LIN、CIRC等工业运动类型
规划器	Planner	用于计算机械臂运动路径的算法模块
规划管线	Planning Pipeline	运动规划的工作流程，包括规划器选择、碰撞检测等
关节空间	Joint Space	以关节角度为坐标的运动空间
笛卡尔空间	Cartesian Space	以末端执行器位置和姿态为坐标的运动空间
避障规划	Obstacle Avoidance Planning	自动避开障碍物的运动规划
话题	Topic	ROS2 中的异步通信机制，用于发布和订阅消息
服务	Service	ROS2 中的同步通信机制，采用请求-响应模式
动作	Action	ROS2 中的异步通信机制，支持长时间运行任务和取消功能

机器人硬件介绍

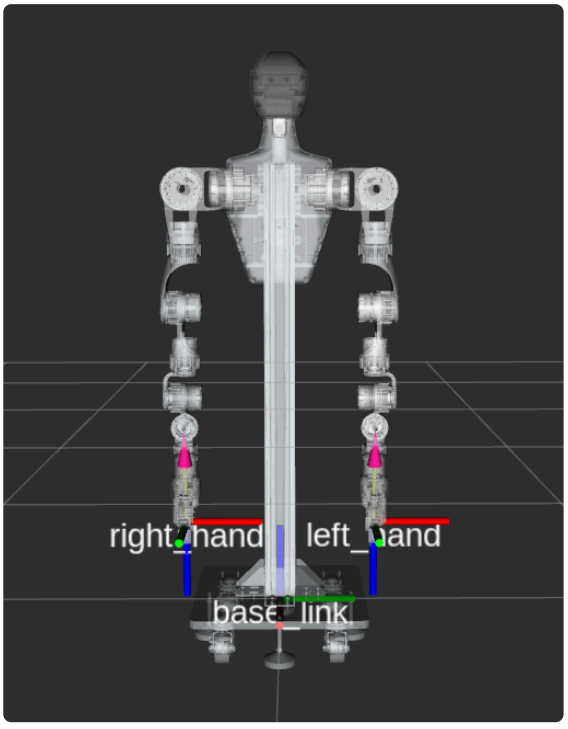
机器人硬件介绍

本节以 RB-A850-10 双臂机器人为例说明硬件结构。

谐波双臂机器人具有两条各7自由度机械臂、带有2自由度云台的RGB和深度相机组成，主要相对尺寸如下图所示：



机器人基坐标系和工具坐标系示意图如下所示：



图中，红、绿、蓝分别表示 x、y、z 轴。基坐标系中，x 轴正方向指向机械臂前方，y 轴正方向指向机械臂左侧，z 轴正方向指向上方。通过指定机器人末端工具坐标系相对于基坐标系的位姿，即可控制机械臂末端在空间中的位置和姿态。

坐标系相关知识

机器人工具坐标系意义

机器人工具坐标系 (Tool Coordinate System, TCP) 是定义在机器人末端执行器上的坐标系，用于精确描述末端执行器的位置和姿态。其主要意义包括：

- **简化编程**：通过工具坐标系，用户可以直接指定工具末端的目标位置，而无需考虑工具本身的尺寸和形状
- **提高精度**：工具坐标系的精确校准可以提高机器人的定位精度
- **便于更换工具**：更换不同工具时，只需重新校准工具坐标系，无需修改机器人程序
- **实现复杂动作**：工具坐标系是实现笛卡尔运动和复杂路径规划的基础

RPY旋转的背景知识

RPY(Roll-Pitch-Yaw)旋转是一种描述物体姿态的方式，通过绕三个坐标轴的旋转来表示物体的朝向：

- **Roll(滚转)**：绕X轴旋转
- **Pitch(俯仰)**：绕Y轴旋转
- **Yaw(偏航)**：绕Z轴旋转

RPY旋转采用固定轴旋转顺序，为Z-Y-X顺序，即先绕Z轴旋转Yaw角，再绕Y轴旋转Pitch角，最后绕X轴旋转Roll角。

机器人末端位姿变换实现

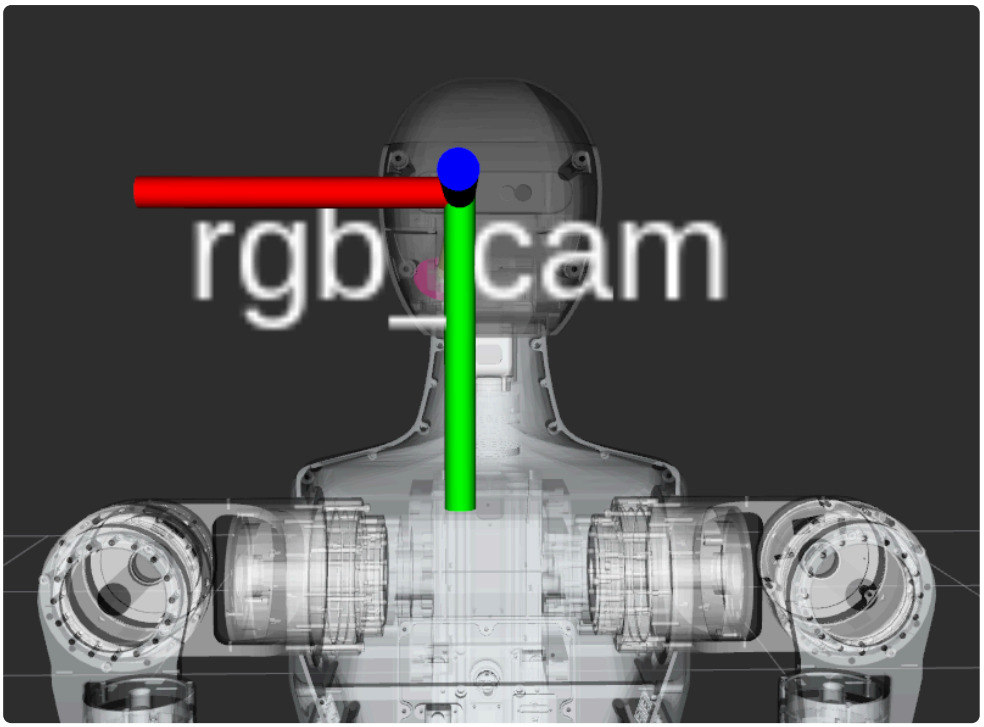
机器人末端位姿变换是指将工具坐标系从一个位置和姿态转换到另一个位置和姿态，主要通过以下步骤实现：

1. **运动学正解（FK）**
   根据当前关节角度计算末端执行器在指定参考坐标系下的位姿，用于获取起始状态或进行状态反馈。
2. **目标位姿构造与坐标变换（Pose Transform）**
   将目标位姿在不同坐标系之间进行转换（例如把相机坐标系下的目标点转换到基坐标系下），得到控制系统可用的目标末端位姿。
3. **运动学逆解（IK）**
   将目标末端位姿转换为一组（或多组）可行的关节角度解，并根据关节限位、连续性、避障等约束筛选合适解。
4. **轨迹规划（Trajectory Planning）**
   在关节空间或笛卡尔空间生成从当前状态到目标状态的平滑轨迹，并满足速度、加速度、加加速度（jerk）等约束。

5. **关节控制与伺服执行（Control / Servo）**

   控制器按照规划轨迹输出位置/速度/力矩等指令，驱动各关节运动并闭环跟踪，最终使末端到达目标位姿。

   机器人视觉相机坐标系示意图:



机器人视觉相机坐标系示意图中，同样约定红、绿、蓝三色分别表示相机坐标系的 x、y、z 轴。z 轴沿相机光轴正向指向前方(由相机指向被观测场景)，x 轴水平指向右侧，y 轴竖直指向下方。通过相机内外参标定，可以将空间中的三维点从机器人或世界坐标系转换到相机坐标系，再投影到图像平面，用于目标检测、位姿估计和路径规划等任务。

在该系统中，机器人视觉由 **RGB 相机** 和 **深度相机** 共同构成，它们的功能分工大致如下:

- **RGB 相机(彩色相机)**
  - 获取场景的二维彩色图像(R/G/B 三通道)，用于:
    - 目标检测、识别(如识别工件、障碍、人体、标志物等)
    - 纹理与颜色特征提取(用于分割不同区域或物体)
    - 辅助可视化与调试(人可以直观看到当前场景)
  - 输出结果是常规的 2D 图像，不直接包含深度信息。
- **深度相机(Depth Camera)**
  - 为图像中的每一个像素提供距离信息，即“相机到物体表面的距离”，形成深度图或点云数据。

- 主要用于：
  - 重建场景的三维几何结构(生成点云或栅格地图)
  - 计算目标物体的三维位置与相对姿态
  - 支持避障、抓取点规划等任务
- 深度信息以“灰度深度图”形式提供。

通过将 **RGB 图像** 与 **深度数据** 进行配准和融合(即同一像素位置既有颜色，又有深度)，系统可以在机器人坐标系 / 世界坐标系中恢复带颜色的三维环境信息，从而实现对目标物体的**精准定位**和对机械臂运动的**精确引导**。

系统启动与环境配置

ROS2 通信特别提醒

系统默认采用全局ROS2通信机制。在局域网中，所有主机上的ROS2进程会自动相互通信，（即一台主机上的ROS2默认可以发现另一台主机上ROS2进程的Topic），因此可能导致节点冲突或意外执行控制程序。为避免此类问题的发生，建议用户在启动ROS2程序前进行以下限制配置。

1. 设置ROS2通信范围。

```
# 设置ROS2通信范围至本地主机（隔离局域网通信）
echo "export ROS_LOCALHOST_ONLY=1" >> ~/.bashrc

# 设置ROS2通信范围至局域网（允许网络中其他设备通信）
echo "export ROS_LOCALHOST_ONLY=0" >> ~/.bashrc
```

2. 配置ROS_DOMAIN_ID以隔离通信域。

ROS_DOMAIN_ID 用于划分独立的通信域。当所有ROS2进程的 ROS_DOMAIN_ID 相同时，它们可以互相通信，反之，即使在同一台主机上，不同 ROS_DOMAIN_ID 的ROS2进程也无法通信。此配置类似于网络中的端口号设置，通过指定一个非0的数字来实现通信隔离。

```
# 查看当前ID
printenv ROS_DOMAIN_ID

# 设置ROS_DOMAIN_ID为72
export ROS_DOMAIN_ID=72
```

启动命令详解

设置ROS2环境变量：

```
source /opt/ros/humble/setup.bash
source ~/raybot_core_ws/install/setup.bash
```

完整启动命令：

```
ros2 launch raybot_bringup robot.launch.py model:=rb_a850_10
```

当前统一启动入口由 raybot_bringup 提供。 model:=rb_a850_10 启动 RB-A850-10，若现场机器人为 RB-A575-3，则使用：

```
ros2 launch raybot_bringup robot.launch.py model:=rb_a575_3
```

启动时间约 2 分钟，等待启动完成后即可正常操作。

启动参数配置

JTC 轨迹误差阈值

raybot_bringup robot.launch.py 支持在启动时覆盖 JointTrajectoryController(JTC) 的轨迹误差阈值。该配置用于控制轨迹执行过程中的误差检测：误差超过阈值时，JTC 可能中止轨迹并报错。参数不传时，系统使用 ros2_controllers.yaml 中的默认值。

参数	示例值	说明
jtc_trajectory_tolerance	0.8	轨迹执行过程中的关节路径误差阈值。旋转关节单位为 rad。
jtc_goal_tolerance	0.25	轨迹终点的关节位置误差阈值。旋转关节单位为 rad。
jtc_goal_time	3.0	轨迹结束后，允许继续满足终点误差阈值的额外时间，单位为秒。
jtc_stopped_velocity_tolerance	0.2	终点静止速度阈值，单位与关节速度接口一致，旋转关节通常为 rad/s。

示例：

```
ros2 launch raybot_bringup robot.launch.py model:=rb_a850_10 \
  jtc_trajectory_tolerance:=0.8 \
  jtc_goal_tolerance:=0.25 \
  jtc_goal_time:=3.0 \
  jtc_stopped_velocity_tolerance:=0.2
```

总体架构

```
robot_controller\
├── action
│   ├── ArmMotion.action
│   ├── ExecuteTrajectory.action
│   └── ProgramCommand.action
├── config
├── include
│   └── robot_controller
│   └── arm_motion_controller.hpp
├── msg
│   ├── CommandResult.msg
│   ├── JointStateSeed.msg
│   └── PlannedTrajectory.msg
├── src
│   ├── action_recorder.cpp
│   ├── arm_motion_controller.cpp
│   ├── arm_motion_server.cpp
│   ├── cli_controller.cpp
│   ├── object_grasp_node.cpp
│   └── robot_controller.cpp
└── srv
    ├── ClearProgram.srv
    ├── DeleteProgramSteps.srv
    ├── ExecuteCommand.srv
    ├── GetLastResult.srv
    ├── PerformGrasp.srv
    ├── PerformPlace.srv
    ├── ProgramEdit.srv
    └── StopMotion.srv
```

核心组件与关系

robot_controller 软件包是谐波双臂运动控制软件的核心，逻辑上由核心执行模块（arm_motion_controller）以及三个面向不同使用场景的上层驱动模块组成。各模块通过 ROS 2 的 topic、service 与 action 机制共享统一接口，协同构成完整的运动控制系统。控制器模式由 raybot_control_mode_manager 统一仲裁；启用 PP 的夹爪由 raybot_pp_controller/ProfilePositionController 提供专用的 Profile Position Action。

当前版本采用 arm_motion_supervisor 单进程托管模式：启动时由 supervisor 统一创建并管理 arm_motion_controller 、 cli_controller 、 action_recorder 、 object_grasp_node 等组件节点。核心模块 arm_motion_controller 负责运动规划与执行；上层模块依据各自的功能定位生成控制指令，并最终调用该执行引擎完成动作下发，从而确保整体运动逻辑的一致性与可复用性。

上层驱动模块介绍：

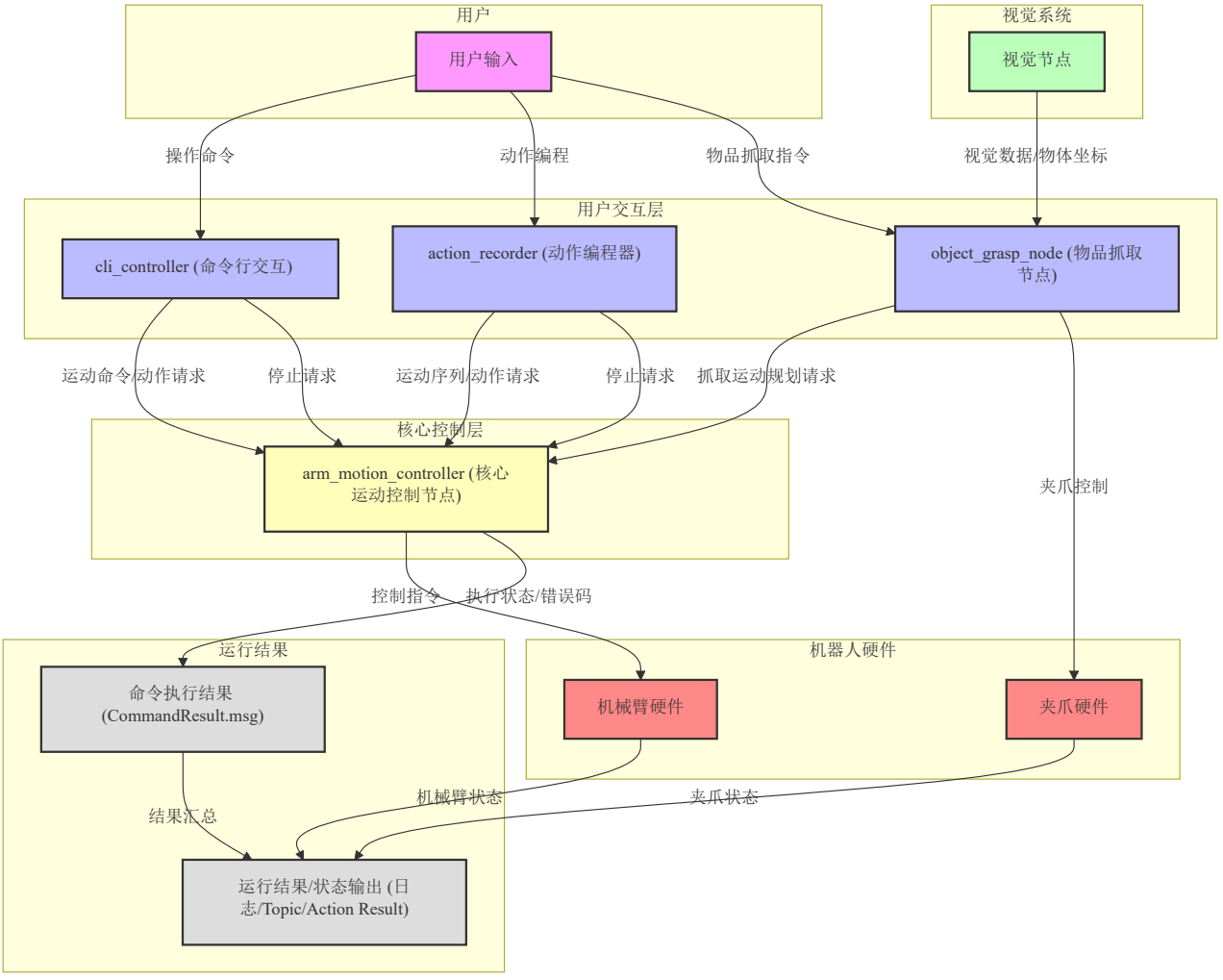
1. **cli_controller**：命令行交互模块，通过 topic/service 接收机械臂运动命令，解析后调用核心执行节点。适合调试阶段和手动控制场景。
2. **action_recorder**：动作编程与回放模块，用于创建、编辑和执行复杂的动作序列。支持动作程序的保存与加载，提供 service 和 action 接口，适合需要重复执行的复杂任务。
3. **object_grasp_node**：自动抓取与放置模块，基于视觉坐标信息规划抓取路径，控制夹爪完成整个抓取-放置流程。支持多阶段抓取策略，适合自动化抓取场景。

模块功能对照表

模块名称	功能	对外接口
arm_motion_controller	核心运动控制节点，负责机械臂运动规划与执行，提供统一的运动控制接口和动作服务接口	/arm_motion_controller/result（robot_controller/msg/CommandResult.msg）、/arm_motion_controller/get_last_result（robot_controller/srv/GetLastResult.srv）、/arm_motion_controller/stop（robot_controller/srv/StopMotion.srv）、/arm_motion_controller/arm_motion（robot_controller/action/ArmMotion.action）、/arm_motion_controller/execute_trajectory（robot_controller/action/ExecuteTrajectory.action）
cli_controller	命令行交互节点，适合调试和手动操作场景	Topic Subscriber /cli_controller/cmd（std_msgs::msg::String）、Service /cli_controller/execute_command（robot_controller/srv/ExecuteCommand）
action_recorder	动作编程与回放节点，用于创建、编辑和执行复杂的动作序列，支持动作程序的保存与加载	Topic Subscriber /action_recorder/cmd（std_msgs::msg::String）、Service /action_recorder/edit（robot_controller/srv/ProgramEdit.srv）、Service /action_recorder/clear_program（robot_controller/srv/ClearProgram.srv）、Service /action_recorder/delete_steps（robot_controller/srv/DeleteProgramSteps.srv）、Service /action_recorder/plan_program（robot_controller/srv/PlanProgram.srv）、Action /action_recorder/command（robot_controller/action/ProgramCommand.action）
object_grasp_node	自动抓取与放置节点，基于视觉坐标规划抓取路径，控制夹爪完成抓取-放置流程，支持多阶段抓取策略	services /object_grasp_node/pick（robot_controller/srv/PerformGrasp.srv）、/object_grasp_node/place

模块名称	功能	对外接口
		(robot_controller/srv/PerformPlace.srv)
raybot_control_mode_manager	管理全局及规划组控制器切换，保证同一关节同一时刻只有一个控制器占用	/raybot_robot/switch_to_* 、 /raybot_robot/switch_<group>_to_* 、 /raybot_robot/get_<group>_control_mode (std_srvs/srv/Trigger)
raybot_pp_controller	为启用 PP 的单轴夹爪提供目标位置、轮廓参数、反馈和到位结果	/l_hand_pp_controller/profile_position 、 /r_hand_pp_controller/profile_position (raybot_pp_controller/action/ProfilePosition)

系统架构图



完整系统启动

启动方式：使用 raybot_bringup robot.launch.py 启动完整系统。该 launch 文件会启动控制管理流程，并拉起 arm_motion_supervisor （由 manager 管理）；随后由 supervisor 在进程内托管 arm_motion_controller 、 cli_controller 、 action_recorder 、 object_grasp_node 等组件节点。

```
source ~/raybot_core_ws/install/setup.bash
ros2 launch raybot_bringup robot.launch.py model:=rb_a850_10
```

说明:

- 该 launch 文件位于 raybot_bringup 软件包中
- 通过 model:=rb_a850_10 或 model:=rb_a575_3 选择机器人型号
- 会自动加载所有必要的配置参数
- 通过 arm_motion_supervisor 统一托管控制组件，减少跨进程通信复杂度

机械臂全局模式切换

系统启动完成后，可通过全局模式切换服务统一切换左右臂、双手与头部控制模式。当前支持切换到 moveit 、 position 、 velocity 、 effort 四种全局模式，并可通过查询服务读取当前整机模式状态。全局服务会作用于所有已启用规划组；对于已启用 PP 的夹爪，全局切换到 moveit 会暂时激活夹爪 JTC，需再调用夹爪专用 PP 服务恢复 PP。

目标模式	Service 名称	调用命令	说明
moveit	/raybot_robot/switch_to_moveit	ros2 service call /raybot_robot/switch_to_moveit std_srvs/srv/Trigger "{}"	将整机切换到 MoveIt 轨迹控制模式。
position	/raybot_robot/switch_to_position	ros2 service call /raybot_robot/switch_to_position std_srvs/srv/Trigger "{}"	将整机切换到底层位置控制模式。
velocity	/raybot_robot/switch_to_velocity	ros2 service call /raybot_robot/switch_to_velocity std_srvs/srv/Trigger "{}"	将整机切换到底层速度控制模式。
effort	/raybot_robot/switch_to_effort	ros2 service call /raybot_robot/switch_to_effort std_srvs/srv/Trigger "{}"	将整机切换到底层力矩控制模式。
当前模式查询	/raybot_robot/get_control_mode	ros2 service call /raybot_robot/get_control_mode std_srvs/srv/Trigger "{}"	查询当前整机控制模式状态。

使用说明:

- get_control_mode 返回当前整机模式状态。

- 若返回 mixed ，表示当前各规划组控制模式不一致。
- 切换到 moveit 时需要 move_group 已运行，否则切换请求会失败。

服务返回值说明:

上述 5 个服务均使用 std_srvs/srv/Trigger ，响应字段统一如下:

字段名	类型	说明
success	bool	服务调用是否成功。对切换类服务表示是否完成模式切换；对查询类服务表示是否成功读取当前模式。
message	string	返回说明文本。切换成功时通常包含目标模式；查询成功时该字段直接返回当前整机模式。

get_control_mode 的 message 常见取值如下:

message 示例	含义
moveit	当前整机处于 MoveIt 轨迹控制模式
position	当前整机处于底层位置控制模式
velocity	当前整机处于底层速度控制模式
effort	当前整机处于底层力矩控制模式
mixed	当前各规划组控制模式不一致

查询示例返回:

```
success: true
message: "moveit"
```

夹爪 PP 模式与指令控制

PP（Profile Position）是夹爪专用的单轴位置控制模式，PP模式设定目标点和速度、加速度后电机底层会规划出运动到目标点的轨迹。同一夹爪不能同时激活 JTC 和 PP controller。 rb_a850_10 、 rb_a575_3 默认使用PP模式控制夹爪:

规划组	关节	PP controller	Profile Position Action
l_hand	left_hand_virtual_joint	l_hand_pp_controller	/l_hand_pp_controller/profile_position
r_hand	right_hand_virtual_joint	r_hand_pp_controller	/r_hand_pp_controller/profile_position

整机与夹爪模式准备

启动运动控制系统后，可通过服务指令将机械臂保持在 MoveIt/JTC，并将左右夹爪切换到 PP。

以下指令应在已加载工作空间环境的终端中执行：

```
source /home/raybot/raybot_core_ws/install/setup.bash

# 将整机规划组切换到 MoveIt/JTC
ros2 service call /raybot_robot/switch_to_moveit std_srvs/srv/Trigger "{}"

# 将左右夹爪切换到 PP
ros2 service call /raybot_robot/switch_l_hand_to_pp std_srvs/srv/Trigger "{}"
ros2 service call /raybot_robot/switch_r_hand_to_pp std_srvs/srv/Trigger "{}"

# 查询左右夹爪当前模式
ros2 service call /raybot_robot/get_l_hand_control_mode std_srvs/srv/Trigger "{}"
ros2 service call /raybot_robot/get_r_hand_control_mode std_srvs/srv/Trigger "{}"
```

启用 PP 的夹爪在标准 bringup 流程完成后已默认激活 PP controller。上述切换指令主要用于显式恢复控制模式或排查模式错误。

`switch_to_moveit` 依赖 `move_group`；如果服务返回失败，应先确认 MoveIt 已启动，再逐项查询左右夹爪模式。

夹爪模式服务

以下服务均使用 `std_srvs/srv/Trigger`：

操作	左夹爪	右夹爪
查询当前模式	/raybot_robot/get_l_hand_control_mode	/raybot_robot/get_r_hand_control_mode
切换到 PP	/raybot_robot/switch_l_hand_to_pp	/raybot_robot/switch_r_hand_to_pp
切换到 MoveIt/JTC	/raybot_robot/switch_l_hand_to_moveit	/raybot_robot/switch_r_hand_to_moveit

启用 PP 的夹爪启动后默认激活 PP controller。仅为调试或兼容旧 JTC 应用时才切换到 MoveIt/JTC；完成
调试后应显式切回 PP。

Profile Position Action

Action 类型为 `raybot_pp_controller/action/ProfilePosition`。Goal 使用 SI 单位，运行参数随每个
goal 传入，不写死在电机 YAML 中：

字段	单位	说明
position	rad	绝对目标位置，必须提供
profile_velocity	rad/s	轮廓速度
profile_acceleration	rad/s²	轮廓加速度
profile_deceleration	rad/s²	轮廓减速度
quick_stop_deceleration	rad/s²	快速停止减速度
timeout	s	单个 goal 的总超时

controller 的默认 PP 参数由 robots.yaml 按型号提供。当前配置如下，左右夹爪使用同一组

型号默认值：

型号	速度 (rad/s)	加速度 (rad/s²)	减速度 (rad/s²)	快速停止减速度 (rad/s²)	超时	力矩上限
rb_a850_10	15.0	60.0	60.0	60.0	10 s	30%
rb_a575_3	5.0	10.0	10.0	10.0	10 s	30%
rb_a575_3_tmp	5.0	10.0	10.0	10.0	10 s	30%

终端发送 Action goal 时可逐项显式指定这些参数；字段为零或未显式赋值时，controller 采用当前型号的默认值。

max_torque 不属于 Profile Position Action，需通过 EtherCAT 独立命令写入对象 0x6072 。

robots.yaml 中的默认值为 30% ，按 0～1000 原始值换算后为 300 。以左夹爪别名 1116 为例：

```
ros2 service call /raybot_robot/ethercat/command \
  ethercat_control/srv/EthercatCommand \
  "{command: '1116 max_torque 300'}"
```

夹爪位置和测试命令

当前三种启用 PP 的机器人 URDF 将夹爪关节限制为 0～0.45 rad ： 0 rad 对应闭合，0.45 rad

接近张开极限。首次测试应使用小速度、小位移，并准备硬件急停。

```
# 确认左夹爪处于 PP 模式
ros2 service call /raybot_robot/switch_l_hand_to_pp std_srvs/srv/Trigger "{}"

# 左夹爪闭合到 0 rad
```

```
ros2 action send_goal /l_hand_pp_controller/profile_position \
  raybot_pp_controller/action/ProfilePosition \
  "{position: 0.0, profile_velocity: 15.0, profile_acceleration: 60.0, profile_deceleration: 60.0,
quick_stop_deceleration: 60.0, timeout: {sec: 10, nanosec: 0}}" \
  --feedback

# 左夹爪张开到 0.45 rad
ros2 action send_goal /l_hand_pp_controller/profile_position \
  raybot_pp_controller/action/ProfilePosition \
  "{position: 0.45, profile_velocity: 15.0, profile_acceleration: 60.0, profile_deceleration: 60.0,
quick_stop_deceleration: 60.0, timeout: {sec: 10, nanosec: 0}}" \
  --feedback
```

Action 到达目标后以 REACHED 完成；命令行使用 --feedback 可持续查看运行反馈，Action Result 返回成功后才能判定夹爪到位。

运行中收到新的 PP goal 时直接抢占旧 goal；切换到其它运动 controller 时执行新的 controller 指令。

取消、超时、following error 或驱动器故障会结束当前 goal 并执行 CSV 零速停止。

查询当前激活模型

bringup 启动后，可查询当前加载的变体名称、URDF/SRDF 路径及启用的规划组：

```
ros2 service call /raybot_robot/get_active_model ethercat_control/srv/GetActiveRobotModel "{}"
```

该服务使用 ethercat_control/srv/GetActiveRobotModel （不是 std_srvs/srv/Trigger ）。完整返回值字段说明见《模块化配置与自动生成使用说明》。

规划组与预定义姿态

规划组介绍

规划组(Planning Group)是指一组关节的集合，用于指定机器人的运动部分。当前系统支持以下规划组：

规划组名称	对应部位	说明
l_arm	左臂	控制左臂的7个关节，负责左臂的运动控制
r_arm	右臂	控制右臂的7个关节，负责右臂的运动控制
dual_arms	双臂	同时控制左右两条机械臂，适合双臂同步定位或协同动作
l_hand	左夹爪	控制左夹爪的1个关节，用于左夹爪开合控制
r_hand	右夹爪	控制右夹爪的1个关节，用于右夹爪开合控制
head	头部	控制头部的2个关节，负责相机云台的俯仰和偏航

规划组关节组成

如果只需要记住日常控制入口，优先关注 l_arm 、 r_arm 、 l_hand 、 r_hand 、 head 这 5 个规划组名称；详细关节组成如下：

规划组	关节数量	关节名称	电机别名
l_arm	7	left_arm_1_joint left_arm_2_joint left_arm_3_joint left_arm_4_joint left_arm_5_joint left_arm_6_joint left_arm_7_joint	1102 1103 1104 1105 1106 1107 1108
r_arm	7	right_arm_1_joint right_arm_2_joint right_arm_3_joint right_arm_4_joint right_arm_5_joint right_arm_6_joint right_arm_7_joint	1109 1110 1111 1112 1113 1114 1115
l_hand	1	left_hand_virtual_joint	1116
r_hand	1	right_hand_virtual_joint	1117
head	2	neck_joint head_joint	1100 1101

预定义姿态

预定义姿态(Predefined Pose)，也称为named坐标，是指预定义的机器人姿态，通过简单的名称即可调用，无需指定复杂的关节角度或笛卡尔坐标。

特点与优势：

- 简化编程：使用名称替代复杂的坐标值，提高编程效率
- 一致性：确保每次调用都能到达相同的精确位置
- 可靠性：经过验证的安全姿态，降低运动风险
- 易用性：适合快速测试和手动操作场景

当前系统定义的named坐标：

- 左右臂均支持 home 姿态，即机器人的零位位置，是机器人的初始安全姿态

使用预定义姿态可以简化机器人编程，提高运动的一致性和可靠性，特别适合调试和常规操作场景。

规划器配置

运动规划器负责根据起始状态与目标状态生成一条可执行的运动轨迹（关节空间轨迹或笛卡尔轨迹），RAYBOT谐波双臂运动控制软件支持多种运动规划器，可根据不同的运动需求选择合适的规划器。

支持的规划器类型

规划器类型	规划器ID	特点	适用场景
PILZ工业运动规划器	pilz_ptp	工业级点对点运动规划，速度快，路径平滑	大部分工业应用场景，特别是需要快速定位的任务
PILZ工业运动规划器	pilz_lin	工业级直线运动规划，精度高，路径严格沿直线，运动平稳可靠	需要精确直线运动的场景，如焊接、喷涂、装配等
OMPL采样规划器	RRTConnectkConfigDefault	基于采样的随机树算法，擅长复杂环境避障，能够处理高维空间规划问题	复杂环境下的避障规划，特别是存在多个障碍物的场景，如狭小空间操作、动态障碍物避让等

PIPELINE_ID / PLANNER_ID 配置说明

CLI 和动作编程器均支持两种规划器配置方式：

写法	示例	说明
显式配置	pipeline_id=pilz_industrial_motion_planner planner_id=PTP	最清晰，适合排查规划器配置问题
planner 简写	planner=ptp / planner=lin	会自动转换为 Pilz 的 PTP / LIN 配置

常用对应关系如下：

目标运动类型	推荐写法	等价显式写法
Pilz PTP	planner=ptp	pipeline_id=pilz_industrial_motion_planner planner_id=PTP
Pilz LIN	planner=lin	pipeline_id=pilz_industrial_motion_planner planner_id=LIN
OMPL	planner=ompl	pipeline_id=ompl

注意：

- 建议使用 pipeline_id=pilz_industrial_motion_planner 完整名称，不建议写非标准缩写如 pipeline=pilz 。
- 只写 planner_id=PTP 或 planner_id=LIN 时，若当前默认 pipeline 不是 Pilz，可能会形成错误组合；建议使用 planner=ptp/lin 或同时写完整 pipeline_id 和 planner_id 。
- exec=0 表示只规划不执行，适合首次验证； exec=1 表示规划成功后执行真实运动。

规划失败原因分析

在使用不同类型的运动规划时，可能会出现规划失败。提前理解常见失败原因，有助于针对性调整规划参数、优化环境建模，并提升整体规划成功率。

LIN（直线）指令失败原因

- **运动学与工作空间限制**：直线轨迹上的某些点可能超出工作空间，或导致关节超限/不可达。
- **避障能力受限**：LIN 强制末端走直线，若直线穿过障碍物，通常无法通过“绕行”来规避碰撞。
- **动态约束不满足**：速度、加速度设置过高，或规划时间/采样精度不足，可能导致轨迹无法生成或无法通过时间参数化。
- **奇异点影响**：轨迹经过或接近奇异点时，可能出现关节速度需求异常增大，导致规划或执行阶段失败。

PTP（点对点）指令失败原因

- **目标位姿/关节目标不可达**：目标点超出工作空间，或在约束条件下（限位、姿态要求等）无可行解。
- **碰撞检测失败**：规划得到的路径与环境或自身结构发生碰撞。
- **规划器与参数配置不匹配**：规划器类型选择不合适，或规划时间、采样密度、约束权重等参数设置不当，导致难以找到可行路径。

提高规划成功率的建议

针对上述规划失败原因，以下是提高规划成功率的通用建议：

1. **优化路径设计**

- 减少路径点数量，只保留关键路标点
- 确保相邻路径点之间的距离适中
- 避免设置机械臂运动范围边界的路径点
- 调整直线轨迹，避免经过奇异点或障碍物

2. **调整规划参数**

- 降低速度和加速度设置，避免关节运动无法实现
- 增加规划时间，特别是对于复杂场景

3. **合理选择规划器**

- 开阔环境、常规点到点移动优先尝试 pilz_ptp
- 需要末端严格走直线的工艺动作优先尝试 pilz_lin
- 存在障碍物、空间狭窄或需要绕障时优先尝试 ompl
- 若一种规划器持续失败，不要反复重试同一参数组合，应先更换规划器或降低动作复杂度

规划器特点比较

比较维度	PILZ规划器	OMPL规划器
规划速度	快	中等
路径质量	平滑、可预测	平滑、每次规划随机生成
避障能力	基础避障	强大避障能力，支持复杂环境
适用场合	工业应用	科研和复杂环境
实时性	高	中等

Arm Motion Controller(核心执行节点)

模块功能

Arm Motion Controller 是谐波双臂运动控制软件的核心执行节点，负责机械臂的运动规划与执行。它封装了运动规划接口，提供统一的运动控制服务，确保多个上层模块可以安全地共享机械臂资源。

支持的运动类型：

- **PTP运动**：点对点运动，机械臂末端以最快路径到达目标位置
- **LIN直线运动**：沿直线轨迹运动，适合需要精确路径控制的场景

- **关节运动**：直接控制关节角度，适合调试和精确位置调整
- **回0操作**：返回机械臂零位位置
- **动作停止**：紧急停止当前运动

交互流程

Arm Motion Controller作为系统的核心执行节点，其交互流程如下：

1. **输入**：
   - 接收来自上层模块的运动指令
   - 支持多种运动类型：PTP运动、LIN直线运动、关节运动、回0操作、动作停止
2. **处理**：
   - 解析命令参数，配置MoveGroupInterface
   - 选择合适的规划器和规划管线
   - 执行运动规划
   - 执行规划结果（或仅规划不执行）
   - 实时监控执行过程
3. **输出**：
   - 通过动作反馈实时返回执行状态
   - 通过 /arm_motion_controller/result Topic发布执行结果
   - 支持通过 /arm_motion_controller/get_last_result Service查询最近结果
4. **在系统总流程中的位置**：
   - 位于核心控制层，接收来自用户交互层的命令
   - 控制机器人硬件层的运动

对外接口说明

Arm Motion Controller 对外暴露的核心结果接口如下：

接口类型	名称	消息/服务类型	功能描述
Topic Publisher	/arm_motion_controller/result	robot_controller/msg/CommandResult	发布最近一次规划/执行结果
Service	/arm_motion_controller/get_last_result	robot_controller/srv/GetLastResult	查询最近一次结果，可按 group 或 command_id 过滤
Service	/arm_motion_controller/stop	robot_controller/srv/StopMotion	请求停止指定组或指定命令对应的运动/规划

/arm_motion_controller/get_last_result 返回值

响应字段如下：

字段名	类型	说明
available	bool	是否查询到结果。 false 表示当前没有匹配结果缓存。
result	robot_controller/msg/CommandResult	最近一次匹配的运动结果；仅在 available=true 时有效。

CommandResult 字段展开如下：

字段名	类型	说明
command_id	string	命令唯一 ID
group	string	规划组名称，如 l_arm 、 r_arm
goal_type	string	目标类型，如 named 、 joints 、 pose 、 stop
success	bool	规划/执行是否成功
moveit_code	int32	MoveIt 返回码，便于定位规划失败原因
cartesian_fraction	float64	笛卡尔路径完成比例，非笛卡尔场景通常为 0.0
executed	bool	是否已实际执行轨迹；仅规划不执行时为 false
message	string	结果描述或失败原因
start_time	builtin_interfaces/Time	开始时间
end_time	builtin_interfaces/Time	结束时间

示例返回：

```
available: true
result:
```

```
command_id: "cmd_20260521_001"
group: "r_arm"
goal_type: "pose"
success: true
moveit_code: 1
cartesian_fraction: 0.0
executed: true
message: "Execution completed"
```

/arm_motion_controller/stop 返回值

响应字段如下：

字段名	类型	说明
accepted	bool	是否已受理停止请求；若目标组或命令不存在，可能为 false
success	bool	是否成功触发停止/取消逻辑
message	string	返回说明文本，例如当前是中断执行、取消规划，或无可停止目标

CLI Controller(命令行机械臂运动控制)

模块功能

CLI控制器是一个命令行交互模块，允许用户通过topic或service命令直接控制机械臂运动。它适合调试阶段和手动控制场景，支持多种运动类型和参数配置。

核心功能：

- 接收并解析命令行格式的运动指令
- 支持多种运动类型：
  - 预定义姿态运动(type=named)
  - 关节空间运动(type=joints)
  - 笛卡尔点运动(type=pose)
  - 回0操作
- 动作停止功能

命令格式与语法

1. **命令格式**

命令采用 key=value 键值对格式，多个键值对之间用空格分隔，通过 type 字段指定运动类型：

```
group=l_arm type=named target=home exec=1
```

2. 支持的命令类型

命令类型	关键字	描述	示例
预定义姿态运动	type=named	运动到预定义姿态	group=l_arm type=named target=home exec=1
关节空间运动	type=joints	直接控制关节角度	group=l_arm type=joints joints=left_arm_1_joint:0.5,left_arm_2_joint:0.5 exec=1
笛卡尔点运动	type=pose	运动到指定笛卡尔坐标	group=l_arm type=pose pose=0.4,0.2,0.3,0,1.57,0 frame=base_link exec=1
双臂多末端位姿	type=multi_pose	同时指定左右手末端目标位姿，适用于 group=dual_arms	group=dual_arms type=multi_pose links=left_hand,right_hand poses=0.012,0.248,0.255,-0.589,1.571,0.983;0.012,-0.248,0.255,-0.352,1.570,-1.924 frame=base_link planner=ptp exec=1
停止运动	type=stop	停止当前运动	group=l_arm type=stop

3. 常用命令参数

参数名	类型	说明
group	string	机械臂组名称 (l_arm/r_arm/head)
type	string	运动类型 (named/joints/pose/cart/stop)
target	string	预定义姿态名称(仅 type=named时使用)
joints	string	关节角度值，支持顺序或 name:value格式
pose	string	笛卡尔坐标 (x,y,z,roll,pitch,yaw)
links	string	type=multi_pose 使用，多个末端 link 用逗号分隔，如 left_hand,right_hand
poses	string	type=multi_pose 使用，多个 6 自由度位姿用分号分隔，顺序与 links 一一对应
frame	string	参考坐标系(默认base_link)
vel	double	速度缩放因子(0-1)
acc	double	加速度缩放因子(0-1)
exec	bool	是否执行规划结果(true/false)
planner	string	规划器简写，常用： ptp 、 lin 、 ompl
planner_id	string	规划器 ID，如 Pilz 的 PTP / LIN 或 OMPL 的 RRTConnectkConfigDefault
pipeline / pipeline_id	string	规划管线，支持： pilz_industrial_motion_planner (工业规划)、 ompl (避障规划)
lin_position_only	bool	仅 Pilz LIN 场景使用。设为 1 时只使用目标位置，姿态保持当前姿态

说明：CLI 命令不支持 replan 键，传入会被判定为非法参数。

DUAL_ARMS 位姿命令规则

group=dual_arms 表示同时规划左右两条机械臂，因此不支持单臂常用的 type=pose pose=... 写法。双臂末端目标必须使用 type=multi_pose ：

> group=dual_arms type=multi_pose links=left_hand,right_hand poses=<左手6DOF>;<右手6DOF> frame=base_link planner=ptp exec=1

其中每个 pose 的格式为：

```
x,y,z,roll,pitch,yaw
```

links 和 poses 的数量必须一致，且建议使用 left_hand,right_hand 作为双臂末端 link。若需要先验证格式和可达性，请将 exec=1 改为 exec=0 。

双臂 PTP 示例：

```
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command:
'group=dual_arms type=multi_pose links=left_hand,right_hand
poses=0.012,0.248,0.255,-0.589,1.571,0.983;0.012,-0.248,0.255,-0.352,1.570,-1.924 frame=base_link
planner=ptp vel=0.1 acc=0.1 exec=1', wait_for_result: false, timeout: 1.0}"
```

双臂 LIN 示例：

```
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command:
'group=dual_arms type=multi_pose links=left_hand,right_hand
poses=0.012,0.248,0.255,0,1.5708,0;0.012,-0.248,0.255,0,1.5708,0 frame=base_link planner=lin
lin_position_only=1 vel=0.1 acc=0.1 exec=0', wait_for_result: true, timeout: 30.0}"
```

LIN 说明： lin_position_only=1 表示只改变末端位置并保持当前姿态。LIN 对起点、姿态、关节限位和奇异点更敏感，建议先用 exec=0 验证规划成功，再改为 exec=1 执行。

对外接口说明

CLI 控制器提供以下对外接口，供外部系统或用户直接调用：

接口类型	名称	消息/服务类型	功能描述
Topic Subscriber	/cli_controller/cmd	std_msgs::msg::String	接收外部命令字符串 (格式： key=value ...)
Service	/cli_controller/execute_command	robot_controller/srv/ExecuteCommand	执行单个运动命令并返回结果（包含 type=stop 停止指令）

说明：CLI 不提供独立的 /cli_controller/result 、 /cli_controller/get_last_result 、 /cli_controller/stop_broadcast 、 /cli_controller/stop 接口；结果由 execute_command 返回，并可通过 /arm_motion_controller/result 观察执行状态。

/cli_controller/execute_command 返回值

响应字段如下：

字段名	类型	说明
accepted	bool	命令是否被受理。命令解析失败、资源忙或停止目标无效时通常为 false 。
command_id	string	命令唯一 ID。若命令在解析前即失败，可能为空。
success	bool	当前调用最终是否成功。若 wait_for_result=false ，该字段可能仅表示命令已成功下发。
result_received	bool	是否已收到来自执行层的最终结果。
timed_out	bool	是否等待结果超时。仅在 wait_for_result=true 时有意义。
result	robot_controller/msg/CommandResult	最终结果详情；未收到结果时仅包含基础字段或默认值。
message	string	调用结果说明、错误原因或超时信息。

其中 result 结构与 /arm_motion_controller/get_last_result 中的 CommandResult 一致，包含 command_id 、 group 、 goal_type 、 success 、 moveit_code 、 cartesian_fraction 、 executed 、 message 、 start_time 、 end_time 等字段。

同步等待结果示例返回：

```
accepted: true
command_id: "cmd_20260521_002"
success: true
result_received: true
timed_out: false
result:
  command_id: "cmd_20260521_002"
  group: "r_arm"
  goal_type: "pose"
  success: true
  moveit_code: 1
  cartesian_fraction: 0.0
  executed: true
  message: "Execution completed"
message: "Execution completed"
```

完整测试命令

测试命令示例详见 示例命令集合

实现细节

- **实现文件**： src/cli_controller.cpp
- **核心功能**：通过 /cli_controller/cmd Topic和 /cli_controller/execute_command Service接收命令，解析后执行运动控制
- **特点**：提供简洁的命令行风格接口，适合调试和手动操作场景
- **命令格式**：采用 key=value 键值对格式，支持多种运动类型和参数配置

Action Recorder(动作编程器)

模块功能

Action Recorder 是一个用于机械臂动作编程、记录、编辑和回放的模块。它允许用户创建复杂的动作序列，并支持通过命令或服务接口进行管理和执行。

核心功能：

- 动作序列的录制与编辑
- 支持多种运动类型的组合
- 提供动作程序的保存与加载功能
- 支持预规划和实时规划模式
- 支持动作步骤间的等待和延迟
- 支持 l_hand / r_hand 夹爪 PP 步骤；夹爪步骤直接调用 Profile Position Action，不经过 MoveIt 规划

配置参数

参数名	类型	默认值	说明
group	string	"l_arm"	默认机械臂组
planner_mode	string	"ompl"	默认规划器模式
vel_scaling	double	0.2	默认速度缩放因子
acc_scaling	double	0.2	默认加速度缩放因子
planning_time	double	5.0	默认规划时间(秒)
execute	bool	true	默认是否执行规划结果

对外接口说明

Action Recorder 提供多种对外接口，支持通过命令行、Topic、Service和Action进行交互：

Topic接口

接口类型	名称	消息类型	功能描述
Topic Subscriber	/action_recorder/cmd	std_msgs::msg::String	接收外部命令字符串(格式：key=value ...)

当前实现不提供独立的程序状态发布 Topic；执行过程状态通过 /action_recorder/command 的 Action Feedback 返回，编辑类操作通过 Service 响应返回结果。

命令格式

Action Recorder 支持以下命令类型：

命令关键字	功能描述	示例
add	添加动作步骤	add type=pose pose=0.4,0.2,0.3,0,1.57,0 wait=1.5
del	删除动作步骤	del index=1
wait	添加等待步骤	wait duration=2.0
play	执行动作序列（支持重放）	play exec=1 replan=0 repeat_count=3
save	保存动作程序	save file=/tmp/program.txt
load	加载动作程序	load file=/tmp/program.txt
clear	清空动作程序	clear

动作步骤命令格式：

> add type=pose pose=0.4,0.2,0.3,0,1.57,0 planner=pilz_ptp wait=1.5 group=l_arm

支持的运动类型包括：

- type=named ： 预定义姿态运动
- type=joints ： 关节空间运动
- type=pose ： 笛卡尔点运动

当 group=l_hand 或 group=r_hand 且 type=joints 时，步骤使用夹爪 PP 控制。可在步骤中配置

pp_vel 、 pp_acc 、 pp_dec 、 pp_qs_dec 和 pp_timeout ； 未提供时采用 PP controller 默认值。

PP 步骤在收到 REACHED 后才算执行成功。夹爪力矩上限不属于动作步骤参数，应通过

EtherCAT 独立命令

配置 0x6072 。

SERVICE接口

服务名	服务类型	功能描述
/action_recorder/edit	robot_controller::srv::ProgramEdit	编辑动作程序（添加/修改步骤）；支持通过命令 save/load 完成保存/加载
/action_recorder/clear_program	robot_controller::srv::ClearProgram	清空当前动作程序
/action_recorder/delete_steps	robot_controller::srv::DeleteProgramSteps	删除指定动作程序步骤
/action_recorder/plan_program	robot_controller::srv::PlanProgram	仅对指定步骤执行规划验证（不执行）

SERVICE 返回值说明

/action_recorder/edit 响应字段：

字段名	类型	说明
success	bool	编辑命令是否成功执行
code	int32	内部结果码，可用于区分参数错误、忙状态、执行失败等类别
message	string	返回说明文本，如命令解析结果、保存/加载结果、错误原因
steps	string[]	当前动作程序各步骤的可读描述；执行 list 、 add 、 del 、 load 后尤其有用

/action_recorder/clear_program 响应字段：

字段名	类型	说明
success	bool	是否成功清空动作程序
message	string	返回说明文本，如 Program cleared 、 Program already empty
remaining_steps	uint32	清空后剩余步骤数；成功时通常为 0

/action_recorder/delete_steps 响应字段：

字段名	类型	说明
success	bool	删除操作是否成功
message	string	返回说明文本；当存在越界索引时会说明跳过情况或失败原因
removed_count	int32	实际删除的步骤数量
failed_indices	int32[]	删除失败的步骤编号列表，常见于索引越界
remaining_steps	string[]	删除后的剩余步骤描述；仅在请求 report_remaining=true 时返回完整列表

/action_recorder/plan_program 响应字段：

字段名	类型	说明
success	bool	指定步骤是否全部规划成功
failed_step	int32	首个失败步骤编号（1-based）；成功时为 0
message	string	规划结果说明或失败原因

/action_recorder/edit 示例返回：

```
success: true
code: 0
message: "Command completed"
steps:
  - "1. pose group=r_arm frame=base_link ee=right_hand ..."
```

ACTION接口

Action名	Action类型	功能描述
/action_recorder/command	robot_controller::action::ProgramCommand	执行动作程序，提供实时执行反馈

完整测试命令

测试命令示例详见 示例命令集合

实现细节

- **实现文件**： src/action_recorder.cpp
- **核心功能**：维护动作程序与状态，支持编辑、执行动作序列
- **关键函数**：
  - handleEditService() ：处理动作程序编辑服务请求

- handlePlayCommand() ： 处理动作程序执行命令
- executeProgram() ： 执行动作程序序列

- **特点**：提供services和action server，支持保存/加载动作程序

交互流程

动作编程器作为动作序列管理模块，其交互流程如下：

1. **输入**：
   - 接收来自 /action_recorder/cmd Topic的命令字符串
   - 支持的命令类型：add、del、wait、play、save、load、clear
   - 通过服务接口接收编辑请求
   - 通过Action接口接收执行请求
2. **处理**：
   - 编辑动作程序：添加、删除、清空动作步骤
   - 解析和验证动作指令参数
   - 执行动作序列：按顺序执行动作步骤
   - 保存/加载动作程序到文件
3. **输出**：
   - 通过 /action_recorder/command Action接口提供实时执行进度反馈
   - 通过服务响应返回操作结果
   - 记录详细的执行日志

Object Grasp Node(物品抓取节点)

模块功能

物品抓取节点(Object Grasp Node)是一个用于机械臂自动抓取和放置物品的高级模块。它能够根据视觉节点提供的坐标信息，自动规划抓取路径和放置位置，并控制夹爪完成整个抓取-放置流程。

核心功能：

- 基于视觉坐标的自动抓取
- 多阶段抓取策略(预抓、接近、抓取、提升、退避)
- 灵活的放置位置配置

- 支持多种抓取方向和重试机制
- 夹爪控制与点云协同
- 支持自定义抓取/放置配置

抓取流程：

1. **预抓阶段**：移动到预抓取位置
2. **接近阶段**：接近目标物体
3. **抓取阶段**：关闭夹爪抓取物体
4. **提升阶段**：将物体提升到安全高度

放置流程：

1. **接近阶段**：接近放置位置
2. **下降阶段**：将物体下降到放置面
3. **释放阶段**：打开夹爪释放物体
4. **提升阶段**：提升到安全高度
5. **退避阶段**：退回到安全位置

使用注意事项：

1. **可达性要求**：需要将物品和放置目的地置于机械臂的可达性范围内，否则规划会失败
2. **LIN移动限制**：如果采用大量LIN直线移动，可能会导致规划失败率增加，建议适当微调物品位置或调整规划器参数
3. **硬件依赖**：该模块依赖于夹爪硬件，如果没有安装夹爪或夹爪通信异常，程序无法正常运行

配置参数

参数名	类型	默认值	说明	使用建议
group_name	string	"l_arm"	机械臂组名称	根据需要选择左臂(l_arm)或右臂(r_arm)
ee_link	string	"l_arm_ee"	末端执行器链接名称	根据实际末端执行器型号调整
target_frame	string	"object_tf"	目标物体TF名称	由视觉系统发布的物体坐标框架
reference_frame	string	"base_link"	参考坐标系	通常使用 base_link 作为固定参考系
planner_mode	string	"cartesian"	规划器模式	根据运动需求选择合适的规划器模式
approach_speed	double	0.1	接近速度比例	调试阶段建议设置为 0.1-0.3，实际运行可根据需求调整
pre_grasp_offset	double	0.12	预抓点到抓取点的距离(米)	根据物体大小调整，确保安全接近
approach_directions	string	"1,0,0"	接近方向(分号分隔多个方向)	多个方向用分号分隔，如 "1,0,0;0,1,0"
direction_variation_deg	double	30.0	方向变化角度(度)	增大角度可增加抓取成功概率，但会增加规划时间
direction_variation_samples	int	8	方向变化采样数	采样数越多，抓取成功概率越高，但规划时间越长
try_alternate_direction	bool	true	是否尝试替代方向	建议开启，提高抓取成功率
verify_before_execute	bool	true	执行前是否验证规划结果	建议始终开启，确保运动安全性
auto_clear_octomap	bool	true	是否重新生成障碍物地图	建议开启，确保环境信息最新
pause_point_cloud	bool	true	是否暂停点云采集	建议开启，避免运动过程中环境变化影响规划
gripper_open_command	string	空	夹爪打开CAN命令	根据实际夹爪硬件配置

参数名	类型	默认值	说明	使用建议
gripper_close_command	string	空	夹爪关闭CAN命令	根据实际夹爪硬件配置
auto_place_after_pick	int	0	是否自动放置(-1/0/1：禁用/默认/启用)	自动化场景建议设置为1，手动控制场景设置为-1
auto_place_retreat_home	bool	false	放置后是否退回到Home位置	连续作业场景建议设置为false，节省时间；空闲状态可设置为true
retreat_after_pick	bool	true	抓取后是否退避	建议始终开启，避免抓取后与周围物体碰撞
retreat_after_place	bool	true	放置后是否退避	建议始终开启，避免放置后与工件或夹具碰撞

命令与服务

服务接口

服务名	类型	功能描述
/object_grasp_node/pick	robot_controller::srv::PerformGrasp	抓取物体服务
/object_grasp_node/place	robot_controller::srv::PerformPlace	放置物体服务

PERFORMGRASP 服务

功能：抓取指定TF名称的物体

请求格式：

```
group_name: "r_arm"       # 机械臂规划组，指定使用哪只手臂进行抓取
ee_link: "right_hand"     # 末端执行器链接名称
target_frame: "object_1"    # 物体的TF名称，由视觉系统发布
retreat_after_pick: 0      # 抓取后是否返回预定义姿态，0表示不返回
retreat_target: "home"     # 抓取后返回的预定义姿态
auto_clear_octomap: 1      # 是否清空并重建障碍物点云，-1/0/1 表示沿用默认/关闭/开启
pre_grasp_offset: 0.1      # 预抓取位置到抓取位置的距离(米)
lift_distance: 0.1         # 抓取物体后向上提升的距离(米)
```

返回字段：

字段名	类型	说明
success	bool	抓取流程是否成功完成
message	string	返回说明文本；成功时通常为 Pick succeeded ，失败时返回具体失败原因

典型成功返回：

```
success: true
message: "Pick succeeded"
```

PERFORMPLACE 服务

功能：将当前抓取的物体放置到指定目标位姿或目标 TF

请求格式：

```
group_name: "r_arm"            # 机械臂规划组
ee_link: "right_hand"          # 末端执行器链接名称
reference_frame: "base_link"    # 参考坐标系
target_frame: "place_target"    # 放置目标 TF 名称
retreat_after_place: 1         # 放置后是否退避，-1/0/1 表示沿用默认/关闭/开启
retreat_target: "home"         # 放置后退避使用的预定义姿态
place_offset_x_mm: 0.0         # 放置偏移 x，单位 mm
place_offset_y_mm: 0.0         # 放置偏移 y，单位 mm
place_offset_z_mm: 0.0         # 放置偏移 z，单位 mm
lift_distance: 0.05            # 放置完成后的抬升距离(米)
```

返回字段：

字段名	类型	说明
success	bool	放置流程是否成功完成
message	string	返回说明文本；成功时通常为 Place succeeded ，失败时返回具体失败原因

典型成功返回：

```
success: true
message: "Place succeeded"
```

完整测试流程

测试环境搭建

建议使用纸箱搭建简易测试环境，用于验证机械臂的抓取与放置功能。将测试物体（如可口可乐易拉罐、杯垫）放置在约 80 cm 高的纸箱上，并将纸箱摆放在机械臂右臂前方约 30–40 cm 处。启动抓取流程后，机械臂将自动完成抓取动作，并将易拉罐放置到杯垫上。

自动抓取功能依赖视觉检测，使用时需要控制相机云台，将待抓取物品移入相机视野范围内。视觉节点介绍详见视觉模块软件说明

测试命令

测试抓取与放置服务调用：

```
ros2 service call /object_grasp_node/pick robot_controller/srv/PerformGrasp "{
  group_name: 'r_arm',                    # 机械臂规划组，指定使用哪只手臂进行抓取
  ee_link: 'right_hand',                  # 末端执行器链接名称
  target_frame: 'cocacola_target',        # 待抓取物体的TF名称，由视觉系统发布
  retreat_after_pick: 0,                  # 抓取流程后不返回预定义姿态
  retreat_target: 'home',                 # 返回目标姿态（retreat_after_pick=0 时不生效）
  gripper_group_name: 'r_hand',           # 夹爪规划组（示例）
  gripper_joint_name: 'right_hand_virtual_joint', # 夹爪关节名称（示例）
  gripper_open_angle: '0.45',             # 夹爪张开角度（rad，字符串）
  gripper_close_angle: '0.0',             # 夹爪闭合角度（rad，字符串）
  approach_directions: '1,1,0',           # 接近方向向量列表，指定机械臂接近物体的方向
  approach_speed: 0.1,                    # 所有运动阶段的速度缩放因子
  approach_acc: 0.1,                      # 所有运动阶段的加速度缩放因子
  pre_grasp_offset: 0.12,                 # 预抓取位置到抓取位置的距离
  lift_distance: 0.05,                    # 抓取后向上提升的距离，也用于放置时的下降距离
  action_pause: 1.0,                      # 各运动阶段之间的暂停时间（秒）
  auto_clear_octomap: 1,                  # 规划前自动清除 Octomap 障碍物地图
  pause_point_cloud: 1,                   # 运动过程中暂停接收新的障碍物点云
  octomap_wait_time: 4.0,                 # 清空八叉树后的等待时间（秒）
  pre_grasp_pipeline_id: 'ompl',          # 预抓取阶段使用的规划管线
  pre_grasp_planner_id: 'RRTConnectkConfigDefault', # 预抓取阶段使用的规划器ID
  approach_pipeline_id: 'pilz_industrial_motion_planner', # 接近阶段使用的规划管线
  approach_planner_id: 'LIN',             # 接近阶段使用的规划器ID，LIN表示直线运动
  lift_pipeline_id: 'pilz_industrial_motion_planner',    # 提升阶段使用的规划管线
  lift_planner_id: 'LIN',                 # 提升阶段使用的规划器ID
  auto_place: 1,                          # 是否启用自动放置流程，1表示启用
  place_target_frame: 'coaster_target',     # 放置物体的目标TF名称
  place_translate_pipeline_id: 'pilz_industrial_motion_planner', # 放置平移阶段使用的规划管线
  place_translate_planner_id: 'LIN',      # 放置平移阶段使用的规划器ID
  place_drop_pipeline_id: 'pilz_industrial_motion_planner',      # 放置下降阶段使用的规划管线
  place_drop_planner_id: 'LIN',           # 放置下降阶段使用的规划器ID
  verify_before_execute: true,            # 执行前是否验证所有阶段的规划结果，true表示验证
  auto_place_retreat: 1                   # 放置后是否自动退避，1表示退避
}"
```

实现细节

- **实现文件**： src/object_grasp_node.cpp
- **核心功能**：基于视觉坐标的物品自动抓取与放置
- **关键函数**：
  - handleGraspService() ： 处理抓取服务请求
  - handlePlaceService() ： 处理放置服务请求
  - planAndExecuteGrasp() ： 规划并执行抓取流程
  - planAndExecutePlace() ： 规划并执行放置流程
  - generateGraspPoses() ： 生成多个可能的抓取姿态
- **特点**：支持多阶段抓取策略，自动处理夹爪控制与环境感知

交互流程

物品抓取节点作为自动抓取与放置模块，其交互流程如下：

1. **输入**：
   - 通过服务接口接收抓取/放置请求
   - 接收视觉节点提供的物体坐标和点云数据
2. **处理**：
   - 解析服务请求参数
   - 规划多阶段抓取/放置路径
   - 调用Arm Motion Controller执行运动指令
   - 控制夹爪完成抓取/释放动作
   - 处理环境点云数据，确保安全操作
3. **输出**：
   - 通过服务响应返回执行结果
   - 发布抓取/放置状态信息
4. **在系统总流程中的位置**：
   - 位于用户交互层，接收用户的抓取指令
   - 与视觉系统交互获取物体坐标
   - 调用核心控制层的Arm Motion Controller执行运动

故障排查

常见错误及解决方案

错误现象	可能原因	解决方案
规划失败	目标点不可达	调整目标位置，确保在工作空间内
规划失败	碰撞检测失败	检查环境中是否有障碍物，调整规划参数
规划成功，但执行失败	速度/加速度过高	降低速度和加速度参数，在动作间增加延迟
服务调用失败	节点未启动	检查相关节点是否正常运行
运动不流畅	PID 不合适	调整 PID 参数
电机掉线 / 从站掉线	CPU 负载过高导致实时线程超时	先确保安全停机→ 重启控制软件→ 仍无法恢复时，断电重启电机驱动电源（掉电≥5~10s 再上电）并重新启动软件，同时降低系统负载
电机掉线 / 通讯中断	网线/接头接触不良、屏蔽/接地不佳、线缆过长或靠近强干扰源	更换网线与接头，使用工业屏蔽线，优化布线远离电机动力线，检查接地与屏蔽端连接方式
电机全部或部分无法进入 OP	从站顺序/别名(alias)不匹配、与alias配置文件不一致	核对从站拓扑与 alias 配置，确认与实际设备一致，必要时重新导出/生成配置并同步到设备
规划正常但轨迹跟踪误差大	关节零位/方向配置错误、单位/减速比不一致、控制器参数不合适	核对关节方向、减速比、编码器单位，复核 URDF/ros2_control 配置，重新整定控制参数
MoveIt 规划成功但控制器拒收轨迹	控制器未激活、接口不匹配（位置/速度/力矩）、trajectory 分段太密	确认 controller 状态为 active，检查接口类型一致，适当降低轨迹采样密度/延长时间参数
延迟变大、指令响应慢	临时 CLI 进程发布（topic pub/service call）有启动开销，DDS 发现耗时	用“常驻节点”做指令入口，减少频繁启动/退出的命令行发布方式，必要时优化 DDS 配置
存储空间不足	系统内存或磁盘空间不足	清理不必要的缓存数据，确保有足够的空间存储日志、配置文件等

示例命令集合

ROS2环境配置命令

设置ROS2通信范围

```
# 设置ROS2通信范围至本地主机
echo "export ROS_LOCALHOST_ONLY=1" >> ~/.bashrc

# 设置ROS2通信范围至局域网
echo "export ROS_LOCALHOST_ONLY=0" >> ~/.bashrc
```

配置ROS_DOMAIN_ID

```
# 查看当前ROS_DOMAIN_ID（用于确认通信域隔离状态）
printenv ROS_DOMAIN_ID

# 临时设置ROS_DOMAIN_ID为5（仅当前终端有效，用于通信域隔离）
export ROS_DOMAIN_ID=5

# 永久设置ROS_DOMAIN_ID（写入bash配置文件，重启终端后生效）
echo "export ROS_DOMAIN_ID=5" >> ~/.bashrc
```

系统启动命令

启动运动控制系统

```
# 启动 RB-A850-10 运动控制系统（默认配置）
# 该命令会启动所有必要的节点，包括机器人控制器、运动规划器和可视化工具
ros2 launch raybot_bringup robot.launch.py model:=rb_a850_10
```

```
# 启动 RB-A575-3
ros2 launch raybot_bringup robot.launch.py model:=rb_a575_3
```

```
# 启动时覆盖 JTC 轨迹误差阈值
ros2 launch raybot_bringup robot.launch.py model:=rb_a850_10 \
  jtc_trajectory_tolerance:=0.8 \
  jtc_goal_tolerance:=0.25 \
  jtc_goal_time:=3.0 \
  jtc_stopped_velocity_tolerance:=0.2
```

CLI控制器命令

发送运动指令

首次联调建议先执行 1 条最小命令，确认 group 、 pose 、 planner_id 与底层控制链路都正常，再尝试更多组合。

最小可用示例：

```
# 右臂执行 1 条 PTP 末端位姿运动
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command:
'group=r_arm type=pose pose=0.45,-0.20,0.4,0,1.57,0 frame=base_link
```

```
pipeline=pilz_industrial_motion_planner planner_id=PTP vel=0.2 acc=0.2 exec=1', wait_for_result: true, timeout: 0}"
```

更多示例：

```
# 发送PTP运动指令（使用OMPL规划器，点对点运动）
# 控制右臂（r_arm）以PTP模式移动到指定位姿，使用OMPL规划器
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command: 'group=r_arm type=pose pose=0.6,-0.18,0.80,0,-1.57,3.14 frame=base_link vel=0.2 acc=0.2 pipeline=ompl planner_id=RRTConnectkConfigDefault exec=1', wait_for_result: true, timeout: 0}"

# 发送LIN运动指令（使用Pilz工业运动规划器，直线运动）
# 控制右臂（r_arm）以LIN模式沿直线移动到指定位姿
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command: 'group=r_arm type=pose pose=0.45,-0.30,0.4,0,1.57,0 frame=base_link pipeline=pilz_industrial_motion_planner planner_id=LIN vel=0.2 acc=0.2 exec=1', wait_for_result: true, timeout: 0}"

# 发送JOINT运动指令（使用OMPL规划器，关节空间运动）
# 控制右臂（r_arm）的第5、6、7关节移动到指定角度
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command: 'group=r_arm type=joints joints=right_arm_7_joint:0.1,right_arm_6_joint:0.1,right_arm_5_joint:0.1 pipeline=ompl planner_id=RRTConnectkConfigDefault exec=1', wait_for_result: true, timeout: 0}"
```

Action Recorder命令

编辑动作程序

初次使用建议按“添加 1 步 -> list 检查 -> play 执行”的顺序验证，先不要一开始就堆叠很多动作。

最小可用示例：

```
# 添加 1 条右臂 PTP 末端动作
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=pose group=r_arm frame=base_link ee=right_hand pose=0.45,-0.20,0.4,0,1.57,0 planner=pilz_ptp vel=0.2 acc=0.2 exec=1 wait=1.0 note=右臂抓取前姿态'}"

# 查看当前动作程序
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'list'}"

# 执行动作程序
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command: 'play exec=1 replan=1'}"
```

常见编辑与执行命令：

```
# 为右臂添加Pilz PTP末端动作（执行后等待1秒）
# 使用Pilz规划器的PTP模式，控制右臂移动到指定的末端位姿
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=pose group=r_arm frame=base_link ee=right_hand pose=0.45,-0.20,0.4,0,1.57,0 planner=pilz_ptp vel=0.2 acc=0.2 exec=1 wait=1.0 note=右臂抓取前姿态2'}"

# 为右臂添加Pilz LIN直线运动指令
# 使用Pilz规划器的LIN模式，控制右臂沿直线移动到指定位姿
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=pose group=r_arm frame=base_link ee=right_hand pose=0.55,-0.20,0.4,0,1.57,0 planner=pilz_lin vel=0.2 acc=0.2 exec=1 wait=1.0 note=右臂直线轨迹'}"

# 为右臂添加OMPL回Home位置指令
# 使用OMPL规划器，控制右臂返回home位置（预定义的安全姿态）
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=named group=r_arm frame=base_link target=home planner=ompl vel=0.3 acc=0.3 exec=1 note=右臂回home'}"

# 为右臂添加OMPL PTP末端动作指令
# 使用OMPL规划器，控制右臂移动到指定的末端位姿
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=pose group=r_arm frame=base_link ee=right_hand pose=0.50,-0.10,0.35,0,1.57,0 planner=ompl vel=0.25 acc=0.25 exec=1 note=右臂OMPL姿态'}"

# 为右臂添加关节空间运动指令
# 直接控制右臂各关节的角度，实现精确的关节空间运动
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=joints group=r_arm joints=0,0.4,-0.2,0,0.6,0,-0.1 frame=base_link planner=ompl vel=0.25 acc=0.25 exec=1 note=右臂关节动作'}"

# 插入等待步骤（等待2.5秒）
# 在动作序列中添加一个等待时间，用于等待外部信号或其他系统准备
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'wait wait_duration=2.5 note=等待外部信号'}"

# 列出所有已添加的动作步骤
# 查看当前动作程序中包含的所有动作步骤和顺序
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'list'}"

# 保存动作序列到文件
# 将当前编辑的动作程序保存到指定文件，以便后续加载和执行
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'save file=demo_program4'}"

# 加载程序
# 从指定文件加载已保存的动作程序
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'load file=demo_program'}"
```

```
# 删除某个动作
# 删除动作程序中索引为3的动作步骤（索引从0开始）
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'del index=3'}"

# 删除多个指定索引的动作步骤
# 批量删除动作程序中索引为2、4、5的动作步骤，保留其余步骤
ros2 service call /action_recorder/delete_steps robot_controller/srv/DeleteProgramSteps "{indices: [2,4,5],
ignore_missing: false, report_remaining: true}"

# 执行动作序列（逐步重新规划路径）
# 执行当前动作程序的所有步骤，每个步骤都会重新规划路径以适应环境变化
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command:
'play exec=1 replan=1 steps=[]'}"

# 执行动作序列（不重新规划路径）
# 执行当前动作程序的所有步骤，使用预先规划好的路径，执行速度更快但不适应环境变化
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command:
'play exec=1 replan=0'}"

# 执行动作序列（重放多轮）
# repeat_count 为正整数，表示将选中步骤完整重复执行的轮数
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command:
'play exec=1 replan=0 repeat_count=3'}"

# 清除内存中存储的所有动作步骤
# 删除当前动作程序中的所有动作步骤，清空动作序列
ros2 service call /action_recorder/clear_program robot_controller/srv/ClearProgram {}

# 仅规划指定步骤（不执行）
# 对动作程序中索引为1、3、5的步骤进行路径规划，但不执行这些步骤
# 用于验证规划的可行性和安全性
ros2 service call /action_recorder/plan_program robot_controller/srv/PlanProgram "
{use_current_state_start: true, steps: [1,3,5]}"

# 仅执行指定步骤
# 仅执行动作程序中索引为1、3、5的步骤，跳过其他步骤
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command:
'play exec=1 replan=1 steps=[1,3,5]'}"

# 仅执行指定步骤并重放
ros2 action send_goal /action_recorder/command robot_controller/action/ProgramCommand "{command:
'play exec=1 replan=1 steps=[1,3,5] repeat_count=2'}"
```

物品抓取节点命令

执行抓取

建议先用最小参数确认视觉 TF、右臂规划组和夹爪链路是通的；只有在最小示例跑通后，再切换到完整参数模板。

最小可用示例：

```
ros2 service call /object_grasp_node/pick robot_controller/srv/PerformGrasp "{
  group_name: 'r_arm',
  ee_link: 'right_hand',
  target_frame: 'cocacola_target',
  retreat_after_pick: 0,
  retreat_target: 'home',
  auto_clear_octomap: 1
}"
```

完整参数示例：

```
# 物品抓取演示1（使用直线运动规划，运动平稳）
# 控制右臂抓取位于cocacola_target坐标系的物体，并放置到coaster_target坐标系
# 使用LIN（直线）运动规划器，运动轨迹平稳，适合对路径要求较高的场景
ros2 service call /object_grasp_node/pick robot_controller/srv/PerformGrasp "{
  group_name: 'r_arm',                    # 运动规划组名称（右臂）
  ee_link: 'right_hand',                  # 末端执行器链接名称
  target_frame: 'cocacola_target',        # 要抓取物体的坐标系
  retreat_after_pick: 0,                  # 抓取后不返回指定姿态
  retreat_target: 'home',                 # 抓取后返回的目标姿态（当前未使用）
  gripper_group_name: 'r_hand',           # 夹爪规划组（示例）
  gripper_joint_name: 'right_hand_virtual_joint', # 夹爪关节名称（示例）
  gripper_open_angle: '0.45',             # 夹爪张开角度（rad，字符串）
  gripper_close_angle: '0.0',             # 夹爪闭合角度（rad，字符串）
  approach_directions: '1,1,0',           # 接近方向向量列表
  approach_speed: 0.1,                    # 所有阶段的速度缩放因子
  approach_acc: 0.1,                      # 所有阶段的加速度缩放因子
  pre_grasp_offset: 0.12,                 # 预抓取位置与抓取位置之间的距离
  lift_distance: 0.05,                    # 抓取后向上提升的距离（也用于放置时的下降）
  action_pause: 1.0,                      # 各阶段之间的暂停时间（秒）
  auto_clear_octomap: 1,                  # 规划前自动清除Octomap障碍物地图
  pause_point_cloud: 1,                   # 运动过程中暂停接收新的障碍物点云
  octomap_wait_time: 4.0,                 # 清除Octomap后等待时间
  pre_grasp_pipeline_id: 'ompl',          # 预抓取阶段使用的规划管线
  pre_grasp_planner_id: 'RRTConnectkConfigDefault', # 预抓取阶段使用的规划器
  approach_pipeline_id: 'pilz_industrial_motion_planner', # 接近阶段使用的规划管线
  approach_planner_id: 'LIN',             # 接近阶段使用LIN（直线）运动规划
  lift_pipeline_id: 'pilz_industrial_motion_planner', # 提升阶段使用的规划管线
  lift_planner_id: 'LIN',                 # 提升阶段使用LIN（直线）运动规划
  auto_place: 1,                          # 启用自动放置功能
  place_target_frame: 'coaster_target',   # 放置物体的目标坐标系
  place_translate_pipeline_id: 'pilz_industrial_motion_planner', # 放置平移阶段使用的规划管线
  place_translate_planner_id: 'LIN',      # 放置平移阶段使用LIN（直线）运动规划
  place_drop_pipeline_id: 'pilz_industrial_motion_planner', # 放置下降阶段使用的规划管线
  place_drop_planner_id: 'LIN',           # 放置下降阶段使用LIN（直线）运动规划
```

```
  verify_before_execute: true,            # 先规划所有阶段，然后执行
  auto_place_retreat: 1                   # 放置后自动撤退
}"

# 物品抓取演示2（使用点对点运动规划，响应更快）
# 控制右臂抓取位于cocacola_target坐标系的物体，并放置到coaster_target坐标系
# 使用PTP（点对点）运动规划器，响应速度更快，适合对时间要求较高的场景
ros2 service call /object_grasp_node/pick robot_controller/srv/PerformGrasp "{
  group_name: 'r_arm',                    # 运动规划组名称（右臂）
  ee_link: 'right_hand',                  # 末端执行器链接名称
  target_frame: 'cocacola_target',        # 要抓取物体的坐标系
  retreat_after_pick: 0,                  # 抓取后不返回指定姿态
  retreat_target: 'home',                 # 抓取后返回的目标姿态（当前未使用）
  gripper_group_name: 'r_hand',           # 夹爪规划组（示例）
  gripper_joint_name: 'right_hand_virtual_joint', # 夹爪关节名称（示例）
  gripper_open_angle: '0.45',             # 夹爪张开角度（rad，字符串）
  gripper_close_angle: '0.0',             # 夹爪闭合角度（rad，字符串）
  approach_directions: '1,1,0',           # 接近方向向量列表
  approach_speed: 0.1,                    # 所有阶段的速度缩放因子
  approach_acc: 0.1,                      # 所有阶段的加速度缩放因子
  pre_grasp_offset: 0.12,                 # 预抓取位置与抓取位置之间的距离
  lift_distance: 0.05,                    # 抓取后向上提升的距离（也用于放置时的下降）
  action_pause: 1.0,                      # 各阶段之间的暂停时间（秒）
  auto_clear_octomap: 1,                  # 规划前自动清除Octomap障碍物地图
  pause_point_cloud: 1,                   # 运动过程中暂停接收新的障碍物点云
  octomap_wait_time: 4.0,                 # 清除Octomap后等待时间
  pre_grasp_pipeline_id: 'ompl',          # 预抓取阶段使用的规划管线
  pre_grasp_planner_id: 'RRTConnectkConfigDefault', # 预抓取阶段使用的规划器
  approach_pipeline_id: 'pilz_industrial_motion_planner', # 接近阶段使用的规划管线
  approach_planner_id: 'PTP',             # 接近阶段使用PTP（点对点）运动规划
  lift_pipeline_id: 'pilz_industrial_motion_planner', # 提升阶段使用的规划管线
  lift_planner_id: 'PTP',                 # 提升阶段使用PTP（点对点）运动规划
  auto_place: 1,                          # 启用自动放置功能
  place_target_frame: 'coaster_target',   # 放置物体的目标坐标系
  place_translate_pipeline_id: 'pilz_industrial_motion_planner', # 放置平移阶段使用的规划管线
  place_translate_planner_id: 'PTP',      # 放置平移阶段使用PTP（点对点）运动规划
  place_drop_pipeline_id: 'pilz_industrial_motion_planner', # 放置下降阶段使用的规划管线
  place_drop_planner_id: 'PTP',           # 放置下降阶段使用PTP（点对点）运动规划
  verify_before_execute: true,            # 先规划所有阶段，然后执行
  auto_place_retreat: 1                   # 放置后自动撤退
}"
```

运动停止指令

停止指定动作组运动

```
# 停止右臂动作组的所有运动（紧急停止指令）
ros2 topic pub --once /arm_motion_controller/stop_request std_msgs/msg/String "{data: 'r_arm'}"
```

```
# 停止左臂动作组的所有运动（紧急停止指令）
ros2 topic pub --once /arm_motion_controller/stop_request std_msgs/msg/String "{data: 'l_arm'}"
```

关闭控制系统

```
# 停止谐波双臂控制软件（正常关闭流程）
source install/setup.bash
ros2 service call /raybot_robot/bringup_shutdown std_srvs/srv/Trigger {}
```