# Raybot桌面级机械臂控制说明

> *本文档面向桌面级机械臂客户独立使用。文档主体沿用 Raybot 机器人运动控制软件说明，并针对 rb_a630_2 桌面级六轴机械臂的模型、规划组、关节命名和启动方式进行了调整。*

## 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
- 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 圆弧控制（Pilz CIRC）
  - 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
- 12. 故障排查
  - 12.1 电机错误清除
  - 12.2 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 运动停止指令


## 文档说明

本文档面向 RB-A630-2 桌面级机械臂客户，介绍 ROS2 环境配置、系统启动、MoveIt 运动规划、CLI 控制、动作编程、停止流程和故障排查。

### 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

> ***功能边界***：*标准 rb_a630_2 配置仅包含 arm 规划组和 6 个机械臂关节，默认关闭夹爪与云台控制。*
> *| 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 |*

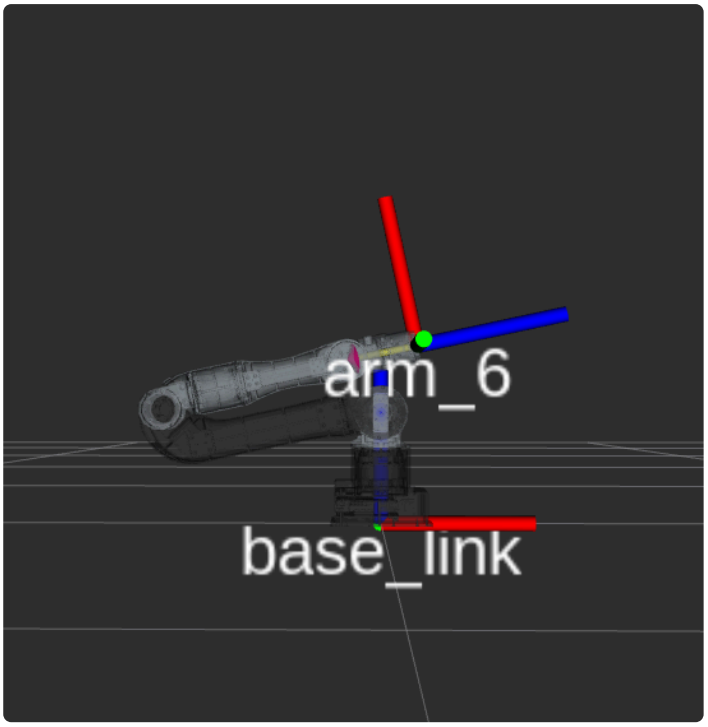
# 机器人硬件介绍

## 机器人硬件介绍

本节以 RB-A630-2 桌面级机械臂为例说明硬件结构。

标准桌面级机械臂由一条 6 自由度机械臂组成；夹爪、相机和其他末端执行器是否配置，以实际交付清单为准。

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。

## 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_a630_2 \
  use_rviz:=true
```

当前统一启动入口由 raybot_bringup 提供。使用以下命令启动 RB-A630-2:

```
ros2 launch raybot_bringup robot.launch.py \
  model:=rb_a630_2 \
  use_rviz:=true
```

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_a630_2 \
  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 软件包是桌面级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**：标准 A630-2 不启用。带视觉和夹爪的项目扩展可另行接入该模块。

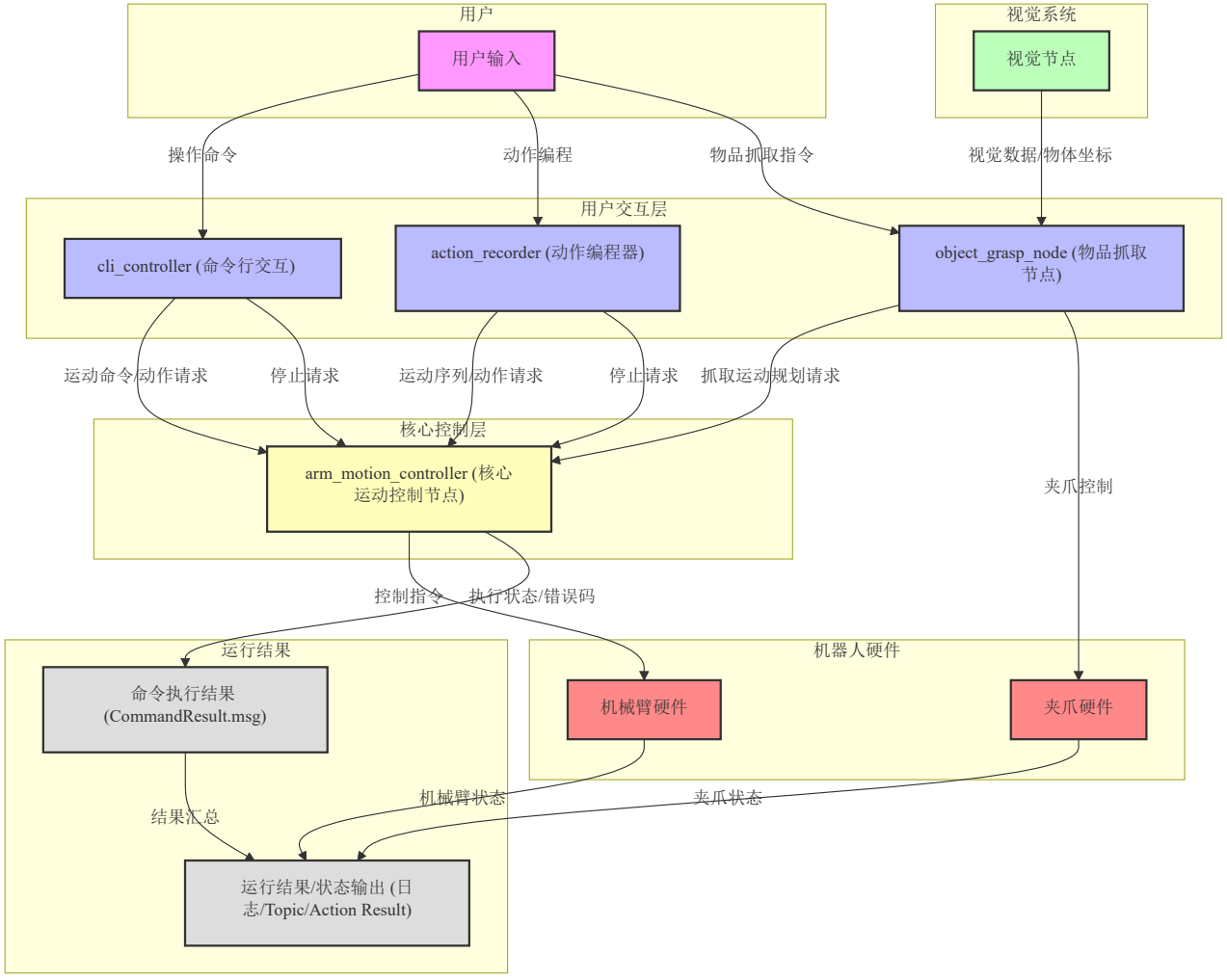
## 模块功能对照表

| 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/ClearPrograma.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 | 可选抓取扩展；标准 A630-2 不启用 | 仅在项目扩展配置中提供 |
| 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>_con |

| 模块名称 | 功能 | 对外接口 |
| --- | --- | --- |
| | | trol_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） |

标准 rb_a630_2 不加载上述夹爪 PP 控制器；该接口仅供带夹爪的扩展配置参考。

## 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 /home/raybot/raybot_core_ws/install/setup.bash
ros2 launch raybot_bringup robot.launch.py \
```

```
model:=rb_a630_2 \
use_rviz:=true
```

**说明**:

- 该 launch 文件位于 raybot_bringup 软件包中
- 通过 model:=rb_a630_2 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机械臂与头部控制模式。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、云台与其他末端执行器

标准 rb_a630_2 配置不包含夹爪规划组、云台规划组或 PP 控制器，默认仅启用 6 轴机械臂 arm 。如果项目另行配置末端执行器，应以项目专用配置和验收说明为准；本客户版不提供未交付硬件的控制命令。

## 规划组与预定义姿态

### 规划组介绍

rb_a630_2 默认规划组为 arm ， 末端链接为 arm_6 ， 预定义姿态包括 home 和 test 。执行命令时应使用 group=arm ， 本型号只使用 group=arm 。

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。标准 rb_a630_2 当前仅支持以下规划组：

| 规划组名称 | 对应部位 | 说明 |
| --- | --- | --- |
| arm | 机械臂 | 控制 6 个关节，负责机械臂运动 |

**规划组关节组成**

对于 rb_a630_2 ，日常控制入口是 arm 规划组；本型号默认没有左机械臂、夹爪或云台规划组。详细关节组成如下：

| 规划组 | 关节数量 | 关节名称 | 电机别名 |
| --- | --- | --- | --- |
| arm | 6 | arm_1_joint<br>arm_2_joint<br>arm_3_joint<br>arm_4_joint<br>arm_5_joint<br>arm_6_joint | 1100<br>1101<br>1102<br>1103<br>1104<br>1105 |

**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：**

- 机械臂支持 home 和 test 姿态，其中 home 是机器人的零位姿态，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桌面级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 |
| **PILZ**工业运动规划器 | pilz_circ | 指定 TCP 沿圆弧运动，支持圆心与途经点定义 | 单臂短弧、半圆与大弧 |
| **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 / planner=circ | 会自动转换为 Pilz 的 PTP / LIN / CIRC 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 |
| Pilz CIRC | type=arc planner=circ | pipeline_id=pilz_industrial_motion_planner planner_id=CIRC |
| 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 是桌面级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，本型号使用 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: "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)
  - 圆弧运动(type=arc)
  - 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=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=arm type=named target=home exec=1 |
| 关节空间运动 | type=joints | 直接控制关节角度 | group=arm type=joints joints=arm_1_joint:0.5,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=arm type=pose pose=0.4,0.2,0.3,0,1.57,0 frame=base_link exec=1 |
| 圆弧运动 | type=arc | 单臂圆心或途经点圆弧 | 已通过测试的完整命令 |
| 停止运动 | type=stop | 停止当前运动 | group=arm type=stop |

3. **常用命令参数**

| 参数名 | 类型 | 说明 |
| --- | --- | --- |
| group | string | 标准 A630-2 使用 arm |
| type | string | 运动类型 (named/joints/pose/arc/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) |
| arc_mode | string | type=arc 必填： center 或 interim |
| arc_point | string | type=arc 必填：辅助点 x,y,z ，单位 m |
| ee | string | 圆弧 TCP link，示例中显式填写 |
| 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 、 circ 、 ompl |
| planner_id | string | 规划器 ID，如 Pilz 的 PTP / LIN / CIRC 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。

### 双臂命令适用性说明

标准 A630-2 仅支持单臂命令。请使用 group=arm ，并先用 exec=0 验证规划，再用 exec=1 执行；本客户版不提供双臂 multi_pose 命令示例。

### 圆弧控制（PILZ CIRC）

type=arc 让指定 TCP 沿圆弧运动。起点取本次真实关节反馈的 FK， pose 是终点完整位姿， arc_point 是圆心或途经点。圆弧默认使用 pilz_industrial_motion_planner/CIRC ，示例显

式填写 planner=circ 。

| 字段 | 含义与单位 |
| --- | --- |
| arc_mode=center | arc_point=x,y,z 是圆心，选择小于 180° 的短弧 |
| arc_mode=interim | arc_point=x,y,z 是弧上的途经点，可定义短弧、半圆或大弧；途经点不是停顿点 |
| pose=x,y,z,roll,pitch,yaw | 终点位置为 m，RPY 为 rad；姿态从起点插值到终点 |
| frame / ee | 本节使用 base_link 及明确的 TCP link；CLI 中辅助点和终点共用 frame |
| vel / acc | 速度、加速度缩放，范围 (0,1] ；相同缩放在不同型号上不代表相同 TCP 速度 |
| exec=0 / exec=1 | 仅规划 / 规划成功后执行；一次 CLI 执行会重新从当前反馈规划 |
| planner=circ | 等价于 pipeline_id=pilz_industrial_motion_planner planner_id=CIRC ；不得换成 OMPL 或 PTP |

- 只支持单臂链式组。双臂机器人选 l_arm 或 r_arm ； dual_arms 不支持同步圆弧。运行时需要加载 Pilz pipeline 和笛卡尔限速配置。
- 圆心模式不能定义半圆或大弧，半圆和大弧使用途经点模式；整圆暂不支持。圆心模式起终点半径差须不超过 0.00001 m 。
- 不要用 type=pose planner=circ ，也不要在圆弧中设置 lin_position_only=1 。不完整或不兼容的请求会失败，不会自动改成 PTP。
- 很小、很浅的三点圆弧可能被 Pilz 判为平面不明确。本次选用 50 mm 半径；不要任意缩小示例或省略坐标精度。

**网页圆弧控制**

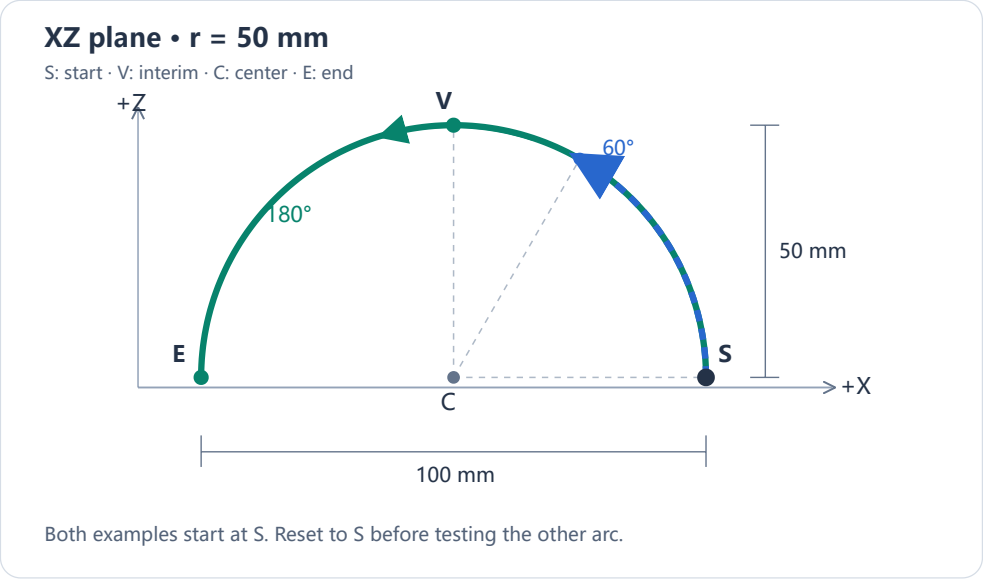
在“机器人控制”页选择单臂规划组，然后进入“圆弧 CIRC”。组名和 TCP 由当前型号或变体配置决定；组合规划组及夹爪不支持圆弧。

1. 确认参考坐标系，查看“起点 S（实际 TCP 反馈）”。起点来自真实反馈，不取目标虚影；可点击“刷新起点”重新读取。
2. 选择圆心或途经点模式，填写辅助点 XYZ 和终点完整 XYZ/RPY，单位分别为 m、rad。“读取当前末端为终点”仅复制一次，之后可独立编辑；终点仍需与起点不同。
3. 查看带 S、C/V、E 标记的虚线圆弧及半径、扫角。它仅为几何示意，不代表 IK、碰撞和动力学约束通过。
4. 在模拟模式点击“仅规划”；真实模式的按钮为“规划并执行”。结果区区分已规划、已执行、请求失败和结果未确认；受理不代表完成。

5. 点击“添加圆弧动作”加入动作列表。列表刷新或加载后仍显示 CIRC；摘要未提供的 TCP、参考系和缩放显示为“由程序保存”。单步回放使用后端原始步骤参数；模拟模式下，选中的步骤包含圆弧时只规划。
6. 切换模型、规划组或 TCP 后重新设置圆弧。应用新参考系会清空终点和辅助点；TF 不可用时不能提交。圆弧内部保留坐标精度，不以起点显示的小数位重建目标。

下方示例可作为首次联调的坐标和起始状态参考。先完成相应准备动作，再在网页填写圆弧；加入程序时，本步起点由前序步骤的结束状态决定。

**观察方式与复现顺序**




1. 选择对应型号和示例 TCP，将 3D 视角对准 XZ 平面（沿 Y 轴观察）。绿色半圆从 S 经 V 到 E；蓝色短弧从同一个 S 出发。
2. 先规划准备动作，确认路径后执行到位。仅规划准备动作不会改变当前关节状态，移动蓝色目标虚影也不会改变圆弧起点。
3. 在准备姿态下发送示例 A 的 exec=0 命令，在 RViz 的 MotionPlanning 中查看规划轨迹。返回 success=true 、 result_received=true 且 result.outcome=1 （ PLANNED ）表示仅规划成功。
4. Web 3D 模型通过关节反馈显示实际状态； exec=0 不会让它运动。需要观察实际运动时，将圆弧命令改为 exec=1 。 result.outcome=3 （ EXECUTED ）且 result.executed=true 才表示执行完成并通过反馈验收； accepted=true 仅表示受理。
5. 每次演示前都回到同一个准备姿态；A、B 两例分别从 S 出发，不能直接连续执行而省略复位。

下列数值来自 2026-10-28 三型号测试中已通过真实 MoveIt/Pilz CIRC 及 TCP FK 检查的轨迹，并按文档显示精度复测了 CLI 和动作程序。测试使用模型反馈、未连接真机；

“通过”指规划及接口验收。所有示例默认 exec=0 ，圆弧 TCP 速度上限约为 20 mm/s。

### RB-A630-2：桌面臂示例

模型为 rb_a630_2 ，圆弧组为 robot_1_arm ，TCP 为 arm_6 。这里使用测试模型 SRDF 的实际组名，与本文其他示例中的 arm 名称不同；复现本节时使用 robot_1_arm ，其他配置先核对其实际规划组和 TCP 。准备到位后，TCP 起点约为 S=(0.225774561,0.056496106,0.470095219) m 。原始通过样本为起点 0、XZ 平面、半径 50 mm 的样本 12（圆心 60°）和样本 14（途经点半圆）。

**准备到起点：** 以下为起点关节目标，单位 rad。先只规划；确认路径后将这一条准备命令的 exec=0 改为 exec=1 ，等待关节反馈到位。

```
ros2 service call /cli_controller/execute_command \
  robot_controller/srv/ExecuteCommand \
  "{command: 'group=robot_1_arm type=joints
joints=0.30000000,1.20000000,1.00000000,0.30000000,0.50000000,0.20000000 planner=ptp vel=0.02
acc=0.017777777777777778 exec=0', wait_for_result: true, timeout: 60.0}"
```

**示例 A：途经点半圆，优先观察这一条。** 从 S 出发，经上方 50 mm 的途经点 V，到达 X 方向减少 100 mm、Z 回到起始高度的终点 E；Y 和末端姿态保持不变。

```
ros2 service call /cli_controller/execute_command \
  robot_controller/srv/ExecuteCommand \
  "{command: 'group=robot_1_arm type=arc arc_mode=interim
arc_point=0.175774561,0.056496106,0.520095219
pose=0.125774561,0.056496106,0.470095219,-0.773447403,-0.860998183,-2.092098530 frame=base_link
ee=arm_6 planner=circ vel=0.02 acc=0.017777777777777778 planning_time=5 exec=0', wait_for_result:
true, timeout: 60.0}"
```

**示例 B：圆心短弧。** 先重新执行准备动作回到 S，再使用圆心定义 60° 短弧；终点相对 S 的 X 减少 25 mm，Z 增加约 43.30 mm。

```
ros2 service call /cli_controller/execute_command \
  robot_controller/srv/ExecuteCommand \
  "{command: 'group=robot_1_arm type=arc arc_mode=center
arc_point=0.175774561,0.056496106,0.470095219
pose=0.200774561,0.056496106,0.513396489,-0.773447403,-0.860998183,-2.092098530 frame=base_link
ee=arm_6 planner=circ vel=0.02 acc=0.017777777777777778 planning_time=5 exec=0', wait_for_result:
true, timeout: 60.0}"
```

**加入动作程序：** 起点已到位时，可将示例 A 添加为圆弧步骤。若程序还需要移动到起点，先在它前面添加上述关节准备步骤。

```
ros2 service call /action_recorder/edit \
  robot_controller/srv/ProgramEdit \
  "{command: 'add group=robot_1_arm type=arc arc_mode=interim
arc_point=0.175774561,0.056496106,0.520095219
pose=0.125774561,0.056496106,0.470095219,-0.773447403,-0.860998183,-2.092098530 frame=base_link
ee=arm_6 planner=circ vel=0.02 acc=0.017777777777777778 planning_time=5'}"
```

添加后可规划当前程序的全部步骤；以下调用只规划，不执行。若程序已包含其他步骤，它们也会参与规划。

```
ros2 service call /action_recorder/plan_program \
  robot_controller/srv/PlanProgram \
  "{use_current_state_start: true, steps: []}"
```

圆弧辅助点、终点、参考系或前序起点改变后应重新规划。

**结果判断与失败排查**

终点可达不保证整条圆弧可达。 NO_IK_SOLUTION 或错误码 -31 需检查终点和中间点；速度/加速度超限可先降速，但关节限位附近的 IK 解跳变可能在降速后仍失败。 Plane for motion is not properly defined 表示圆弧构造失败，可检查几何或改用圆心短弧。圆弧不使用笛卡尔完成比例验收， cartesian_fraction=-1 不等于失败。

下载本节示例的起点、命令与测试结果。坐标绑定上述型号、TCP 和完整起点，不能直接当作其他变体或工具的目标。

## 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: "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
- 支持动作步骤的规划、执行和回放；标准 A630-2 不包含夹爪 PP 步骤

### 配置参数

| 参数名 | 类型 | 默认值 | 说明 |
| --- | --- | --- | --- |
| group | string | "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=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
- type=arc ： 圆弧步骤，字段与 CLI 圆弧指令 相同

标准 A630-2 不支持夹爪 PP 动作步骤。

### 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=arm frame=base_link ee=arm_6 ..."
```

### 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

## 可选扩展功能说明

标准 rb_a630_2 交付配置默认关闭夹爪、云台、视觉抓取和物品抓取节点。本客户版不提供这些功能的可执行命令；如项目包含相应硬件和软件扩展，应以项目专用配置和验收说明为准。

## 故障排查

### 电机错误清除

机器人控制已启动且硬件插件仍在运行时，可手动调用 clear_motor_errors 服务。网页机器人控制栏的 **错误清除** 按钮位于 **关闭** 与 **紧急停止** 之间，清除当前运行实例全部已映射电机的错误；点击后等待逐轴确认，可展开按钮下方的结果面板查看每轴结果、清错前后错误码及最终状态字。完整服务请求与回复也记录在执行日志中。

服务类型为 ethercat_control/srv/ClearMotorErrors ，默认服务路径为 /raybot_robot/clear_motor_errors 。使用自定义 interface_ns 或多个实例时，请替换为目标实例的路径；网页会自动使用当前运行实例的命名空间。命令行须与控制程序使用相同的 ROS_DOMAIN_ID ，并加载包含该服务类型的工作空间：

```
source /opt/ros/humble/setup.bash
source /home/raybot/raybot_core_ws/install/setup.bash

# 单轴清错：1107 仅为示例，请替换为当前电机配置的 EtherCAT alias。
ros2 service call /raybot_robot/clear_motor_errors ethercat_control/srv/ClearMotorErrors \
  "{clear_all: false, axis_aliases: [1107]}"

# 多轴清错：列表只接受 alias，不接受关节序号或内部轴索引。
ros2 service call /raybot_robot/clear_motor_errors ethercat_control/srv/ClearMotorErrors \
  "{clear_all: false, axis_aliases: [1107, 1108]}"

# 清除当前硬件插件映射的全部电机错误。
ros2 service call /raybot_robot/clear_motor_errors ethercat_control/srv/ClearMotorErrors \
  "{clear_all: true, axis_aliases: []}"
```

clear_all: true 时忽略 axis_aliases ；指定列表会去重， clear_all: false 且列表为空时拒绝请求。无故障轴直接返回“原本无故障”，不改变其运动或使能状态。

服务等待操作完成再回复，全批次共用 30 秒截止时间；自行设置客户端超时时应大于 31 秒，网页等待 35 秒。 success: true 仅表示全部目标均为“已清除”或“原本无故障”；部分成功时整体为 false ，须查看 results ，不要重复清除仍持续存在的故障。每项包含 alias 、 pdo_profile 、 status 、 error_before 、 error_after 、 status_word 和 message ：

| status | 状态 | 说明 |
| --- | --- | --- |
| 1 / CLEARED | 已清除 | 三组新反馈确认错误码为零且退出 Fault / Fault reaction，并完成恢复保持确认 |
| 2 / ALREADY_CLEAR | 原本无故障 | 新反馈确认无故障，未改变该轴输出 |
| 3 / INVALID_TARGET | 目标无效 | alias 不存在、越界或未映射到当前硬件插件 |
| 4 / UNSUPPORTED | 不支持 | PDO profile 未声明支持 MT_Device 或 CrabNova 清错协议 |
| 5 / BUSY | 忙碌 | 已有清错批次，或目标轴正在置零、保存参数等维护操作 |
| 6 / COMMUNICATION_FAILED | 通信失败 | 目标离线、无法获取新反馈或周期线程未运行 |
| 7 / FAULT_REMAINS | 故障仍存在 | 一次复位后仍存在故障，按 message 排查具体原因 |
| 8 / TIMEOUT | 超时 | 截止时间内未完成清错确认或 CSV 接管 |
| 9 / CANCELED | 取消 | 硬件停用、停止或关机锁定抢占了清错 |

通信失败时，返回的错误码零值不代表已确认无故障，必须同时检查 status 。网页断连或等待超时也只表示结果未确认，不会自动重试；检查实际电机状态后再决定后续操作。

设备清错由 PDO profile 声明的协议分派：

- **MT_Device**： standard 、 pvt 、 pdo260224 、 standard260330 、 standard2026090301 ，按位解释 0x603F ；兼容旧固件复位后短暂重启掉线及新固件通信连续两种行为。
- **CrabNova**：仅 crabnova ，按厂家 Fxxx 单一报警码解释；报警稳定至少 20 ms 后才发复位。
- 两类设备每次调用每轴最多发送一次 0x0000 → 0x0080 → 0x0000 控制字序列，等待三组不同的新鲜反馈确认，不循环复位；其他 profile 返回“不支持”。

另一耦合器支路持续掉电、Domain WKC 不完整时，服务可使用在线目标轴的新异步 SDO 回复确认；目标本身须恢复到可交换 PDO 的 OP 状态。目标仍离线、无法进入 OP 或故障条件持续存在时，服务会失败或超时，需排查电源、连线及具体报警原因。

对需要复位的故障轴，清错会丢弃旧运动目标、PP 请求和前馈会话，清错期间的新运动命令也不会在结束后补执行。原下使能轴保持下使能；原上使能轴在清错成功并确认

CSV 模式后进入零速保持。清错失败或超时也保留保持状态，不恢复旧轨迹。确认结果后须重新发送有效运动指令；前馈模式须重新建立会话。非目标轴不受此操作影响。清错等待期间仍可使用“关闭”和“紧急停止”。

## 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-A630-2 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_a630_2
```

```
# 启动 RB-A630-2
ros2 launch raybot_bringup robot.launch.py model:=rb_a630_2
```

```
# 启动时覆盖 JTC 轨迹误差阈值
ros2 launch raybot_bringup robot.launch.py model:=rb_a630_2 \
  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：**

```
# 机械臂执行 1 条 PTP 末端位姿运动
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command:
'group=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）
# 控制机械臂（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=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）
# 控制机械臂（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=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）
# 控制机械臂（arm）的第4、5、6关节移动到指定角度
ros2 service call /cli_controller/execute_command robot_controller/srv/ExecuteCommand "{command:
'group=arm type=joints joints=arm_6_joint:0.1,arm_5_joint:0.1,arm_4_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 条机械臂 PTP 末端动作
ros2 service call /action_recorder/edit robot_controller/srv/ProgramEdit "{command: 'add type=pose
group=arm frame=base_link ee=arm_6 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=机械臂抓取前姿态'}"

# 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：**

```
# 为机械臂添加Pilz PTP末端动作（执行后等待1秒）
# 使用Pilz规划器的PTP模式，控制机械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=arm frame=base_link ee=arm_6 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=机械臂抓取前姿态2'}"

# 为机械臂添加Pilz LIN直线运动指令
# 使用Pilz规划器的LIN模式，控制机械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=arm frame=base_link ee=arm_6 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=机械臂直线轨迹'}"

# 为机械臂添加OMPL回Home位置指令
# 使用OMPL规划器，控制机械臂返回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=arm frame=base_link target=home planner=ompl vel=0.3 acc=0.3 exec=1 note=机械臂回home'}"

# 为机械臂添加OMPL PTP末端动作指令
# 使用OMPL规划器，控制机械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=arm frame=base_link ee=arm_6 pose=0.50,-0.10,0.35,0,1.57,0 planner=ompl vel=0.25 acc=0.25 exec=1 note=机械臂OMPL姿态'}"

# 为机械臂添加关节空间运动指令
# 直接控制机械臂各关节的角度，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=arm joints=0,0.4,-0.2,0,0.6,0 frame=base_link planner=ompl vel=0.25 acc=0.25 exec=1 note=机械臂关节动作'}"

# 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'}"
```

## 可选扩展命令

标准配置不提供物品抓取命令。请勿直接复用其他型号的夹爪或视觉抓取示例。

## 运动停止指令

### 停止指定动作组运动

```
# 停止机械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: 'arm'}"

# 停止机械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: 'arm'}"
```

### 关闭控制系统

```
# 停止桌面级控制软件（正常关闭流程）
source /home/raybot/raybot_core_ws/install/setup.bash
ros2 service call /raybot_robot/bringup_shutdown std_srvs/srv/Trigger {}
```