

# Rokae ROS2说明手册

## 介绍

### ROS2简介

ROS2 (Robot Operating System 2) 是机器人操作系统 (ROS) 的全面升级版本，专为满足工业级实时性、安全性和跨平台需求而设计。  
ROS2是ROS1的升级版，但是存在一些区别：

- ROS2采用去中心化架构，节点之间直接发现和通信
- 引入标准化的，受管理的节点生命周期，支持硬实时，可用于更高精度机器人
- 采用更高效的ament编译系统，并使用colcon作为新的构建工具

### 目的和范围

#### 目的

本文件提供了安装和配置 Rokae ROS2软件包的指南，并包含针对仿真和真实机器人操作的提供教程。  
本手册适用于熟悉基本ROS2概念、并希望将Rokae机器人集成到其应用中的开发者。

#### 范围

当前版本的Rokae ROS2软件包提供xMate系列CR7、CR12、CR18、CR20、CR35、ER3、ER7、Pro3、Pro7、SR3、SR4、SR5和AR机型机械臂，后续会适配更多的机型，用户也可根据自己的需要自定义适配新的机型

### 当前适配机型

AR 系列:xMateAR5L,xMateAR5R

CR 系列:xMateCR7,xMateCR12,xMateCR18,xMateCR20,xMateCR35

ER 系列:xMateER3,xMateER7

Pro 系列:xMatePro3,xMatePro7

SR 系列:xMateSR3,xMateSR4,xMateSR5

## 安装

### 环境配置

rokae ros2主要在ubuntu22.04+ROS2 humbe 上进行开发测试，在其他环境中可能存在不兼容情况。

- 硬件要求

组件	配置要求	备注
CPU	64 位 Intel i5 / i7 (或同等 AMD 处理器)	建议使用8核及以上处理器，保证1000hz实时模式

组件	配置要求	备注
RAM	8G或16G	MoveIt 2 运动规划和 RViz 可视化需要较大的内存
存储空间	20G以上	更快的存储可提高启动速度和数据处理能力
GPU	支持 CUDA 的 NVIDIA GPU	非必须

- ROS2 环境安装 (humble)

鱼香ROS2一键安装: <https://blog.csdn.net/pixelprodigy/article/details/147933853>

moveit和controller-manager及相关的包安装

```

sudo apt update
sudo apt install ros-humble-moveit
sudo apt install ros-humble-controller-manager
sudo apt install ros-humble-joint-state-broadcaster
sudo apt install \
    ros-humble-joint-state-publisher\
    ros-humble-forward-command-controller \
    ros-humble-effort-controllers \
    ros-humble-velocity-controllers \
    ros-humble-position-controllers \
    ros-humble-joint-trajectory-controller
##若有其余的包未找到，可自行sudo apt install安装
source /opt/ros/humble/setup.bash

```

- 创建本地工作空间

联系珞石开发人员获取最近的ros2软件包，将软件包复制到本地工作空间的src目录下

```

##创建ros2_ws工作空间，必须包含子目录src
mkdir -p ~/ros2_ws/src
##复制软件包到src
##在ros2_ws目录下编译
colcon build

```

！！建议刷新环境变量--在 .bashrc 文件末添加下面内容并保存

！！在 home 文件夹下输入快捷键 **ctrl+h** 显示隐藏文件 .bashrc

```

source /opt/ros/humble/setup.bash
source ~/ros2_ws/install/local_setup.sh
source ~/ros2_ws/install/setup.bash

```

# 工作空间概述

## rokae包概述

- |— doc-----手册目录
- |— rokae\_description-----存放 URDF、描述机器人模型的配置文件
- |— rokae\_hardware-----主要文件夹 具体结构如下
- |— rokae\_msgs-----包含在其他包中使用的自定义消息
- |— rokae\_xMateAR5L\_moveit\_config-----各机型的moveit\_config配置文件
- |— rokae\_xMateAR5R\_moveit\_config
- |— rokae\_xMateCR7\_moveit\_config
- |— rokae\_xMateCR12\_moveit\_config
- |— rokae\_xMateCR18\_moveit\_config
- |— rokae\_xMateCR20\_moveit\_config
- |— rokae\_xMateCR35\_moveit\_config
- |— rokae\_xMateER3\_moveit\_config
- |— rokae\_xMateER7\_moveit\_config
- |— rokae\_xMatePro3\_moveit\_config
- |— rokae\_xMatePro7\_moveit\_config
- |— rokae\_xMateSR3\_moveit\_config
- |— rokae\_xMateSR4\_moveit\_config
- |— rokae\_xMateSR5\_moveit\_config

## rokae\_hardware包结构

- |— CMakeLists.txt
- |— config-----控制器配置文件
- |— include-----硬件接口头文件
- |— launch-----启动各个节点的文件
- |— package.xml
- |— rokae\_hardware\_interface.xml
- ROS2 Control 框架中的插件描述文件，用于向 ROS2 控制系统注册硬件接口
- |— sdk-----sdk相关包
- |— src-----具体实现cpp，具体文件如下

## src文件结构

- |— connect\_test.cpp-----网络性能分析测试！！未写入节点，编译后在build目录下寻找二进制可执行文件运行
- |— movej\_client.cpp-----movej函数客户端示例，与rokae\_driver(服务端)一起使用
- |— movej\_moveit\_test.cpp-----基于moveit的movej实现（moveit规划）
- |— rokae\_driver.cpp-----6轴机器人 封装特定接口（ros2 service）
- |— rokae\_driver7.cpp-----7轴机器人 封装特定接口（ros2 service）
- |— rokae\_hardware\_interface.cpp-----硬件接口具体实现

# 入门指南

## ROS2\_control架构

rokae ros2采用ros2\_control架构

参考学习网站: [https://control.ros.org/rolling/doc/getting\\_started/getting\\_started.html](https://control.ros.org/rolling/doc/getting_started/getting_started.html)

在这个架构下, 只需提供几个关键性文件

- 控制器相关: 这部分保存在一个yaml文件中, 一般是与实际控制算法相关的参数, 主要位于 **rokae\_hardware/config**目录下
- 硬件相关参数: 这部分保存在URDF中, 主要位于**rokae\_deacription/urdf**目录下
- 硬件接口实现: 位于**rokae\_hardware/src/rokae\_hardware\_interface.cpp**内部调用了rokae的API实现对机器人的控制, 硬件接口以**plugin**的形式被ROS2的controller\_manager加载并管理

## 在rviz中使用moveit规划真实机械臂

(1) 确保 MoveIt 2、rokae\_ros2、ros2\_control 包已正确安装

(2) 执行launch文件

```
ros2 launch rokae_hardware rokae_moveit_launch.py robot_type:=SR4
use_fake_hardware:=true robot_ip:=192.168.2.160 local_ip:=192.168.2.1
```

这里的rt是什么意思

!!! 注意!!!

将示例中的 **robot\_type** 换成你的机型 例如 AR5L、AR5R、SR4、ER7、Pro3、Pro7、CR7、CR12、CR18、CR20、CR35 等

**robot\_ip**对应机器人ip **local\_ip**对应本机ip

**use\_fake\_hardware**:使用虚拟硬件接口, 连接实体机器人或hmi时 设为**false**, 建议先使用 **use\_fake\_hardware = true** 虚拟硬件接口测试

连接实体机器人时, 注意hmi中的rci设置以及丢包率

更换机型(六轴/七轴): 在**rokae\_hardware\_interface.cpp**的227-228行及**rokae\_hardware\_interface.h**头文件95-96行取消注释对应行, 代码如下

```
/*rokae_hardware_interface.cpp*/
robot_ = std::make_shared<rokae::xMateRobot>(robot_ip_, local_ip_); //连六轴机型
// robot_ = std::make_shared<rokae::xMateErProRobot>(robot_ip_, local_ip_); //连七轴
机型
//根据机型轴数不同需要对共享指针robot_的定义和初始化进行修改。

/*rokae_hardware_interface.h*/
std::shared_ptr<rokae::xMateRobot> robot_; //连六轴机型
// std::shared_ptr<rokae::xMateErProRobot> robot_; //连七轴机型
```

(3)在rviz下进行路径规划:

- 黄色机械臂模型为goal position, 白色实体模型为真实机械臂模型, 灰色透明的机械臂是初始时的位置。
- 点击交互标记(表示为机器人末端执行器的球体), 将其移动到所需的目标位置, 或在 MotionPlanning下的

**Joints**修改goal position的关节角度。

- 点击 "Plan & Execute" 生成并可视化机器人的轨迹，可以看到白色机械臂运动到黄色机械臂姿态。
- 多次连续规划运动时，建议先点击rviz机械臂末端交互小球，更新MotionPlanning的Joints下的关节信息，便于进行下一次的运动规划。
- 运动速度修改：MotionpPlanning下的Planning右侧Options,**VelocityScaling**，比例为0-1。



图1 hmi状态监控

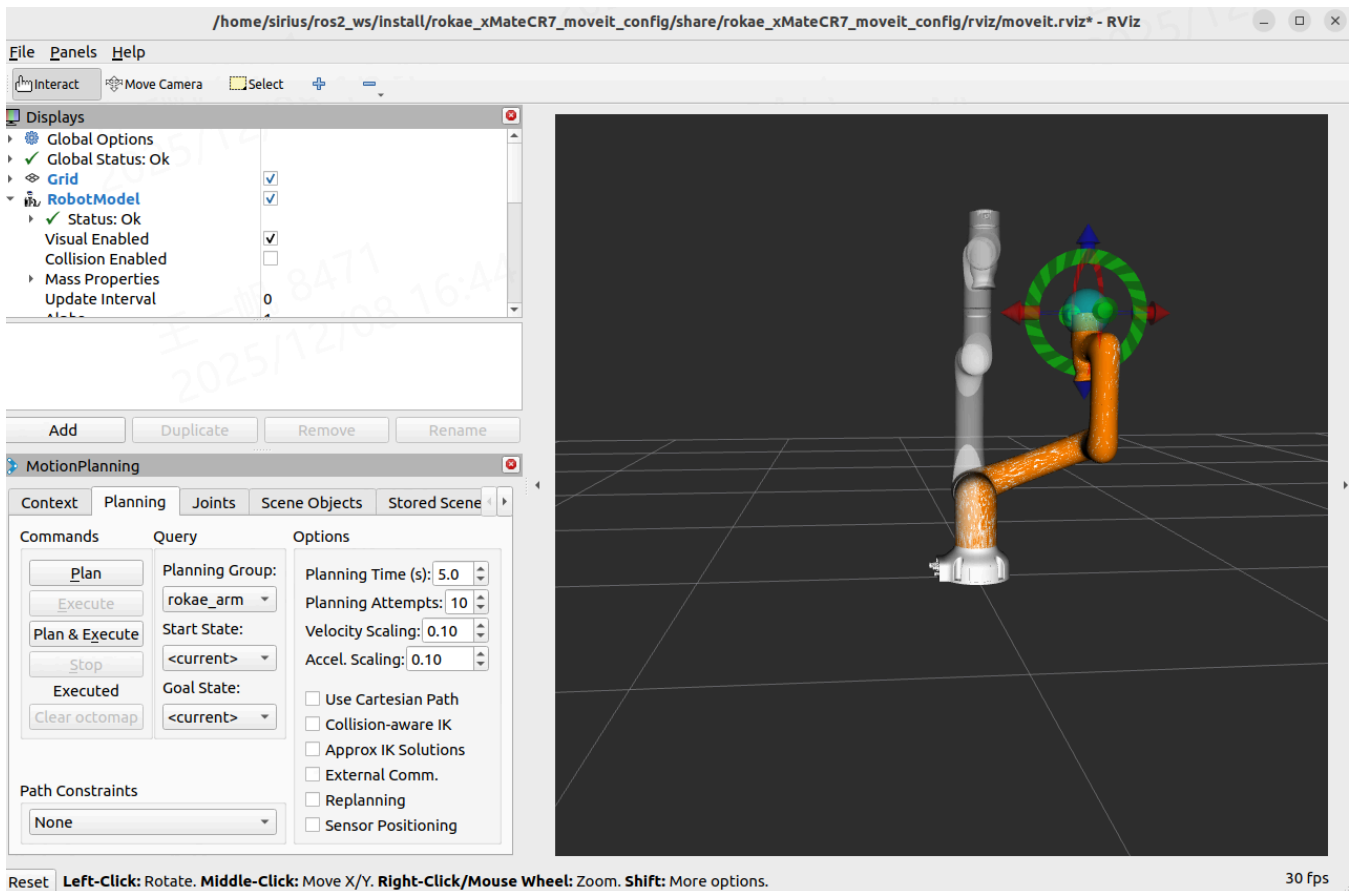


图2 rviz可视化

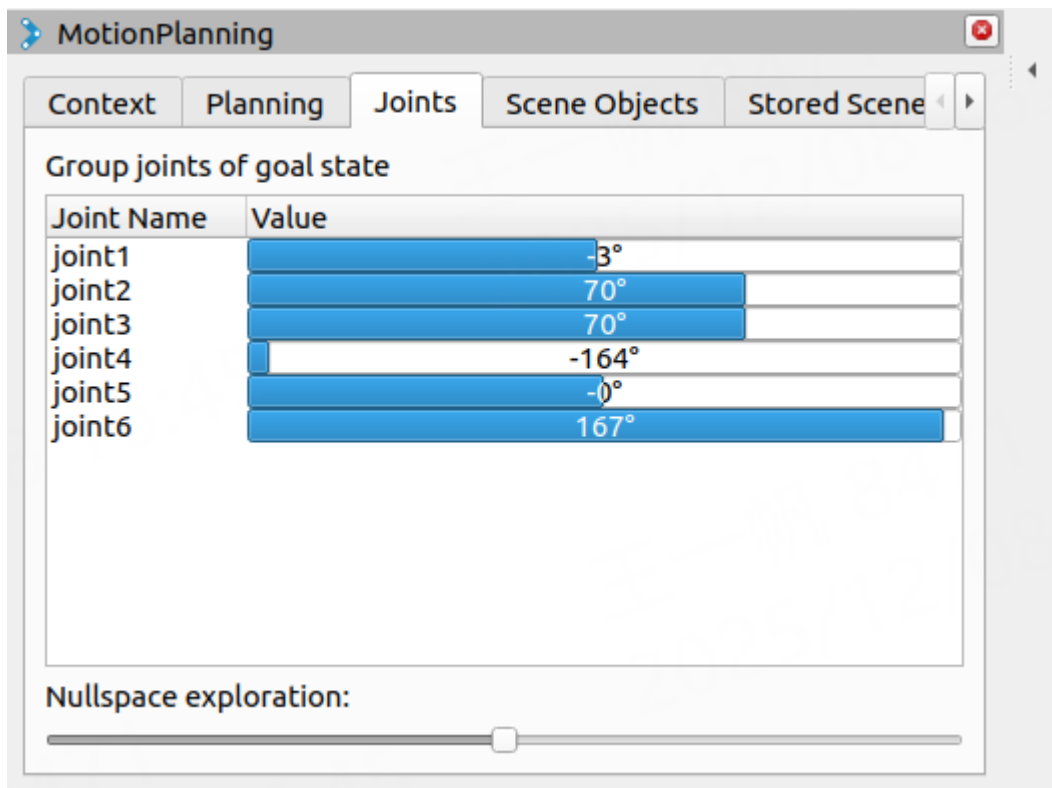
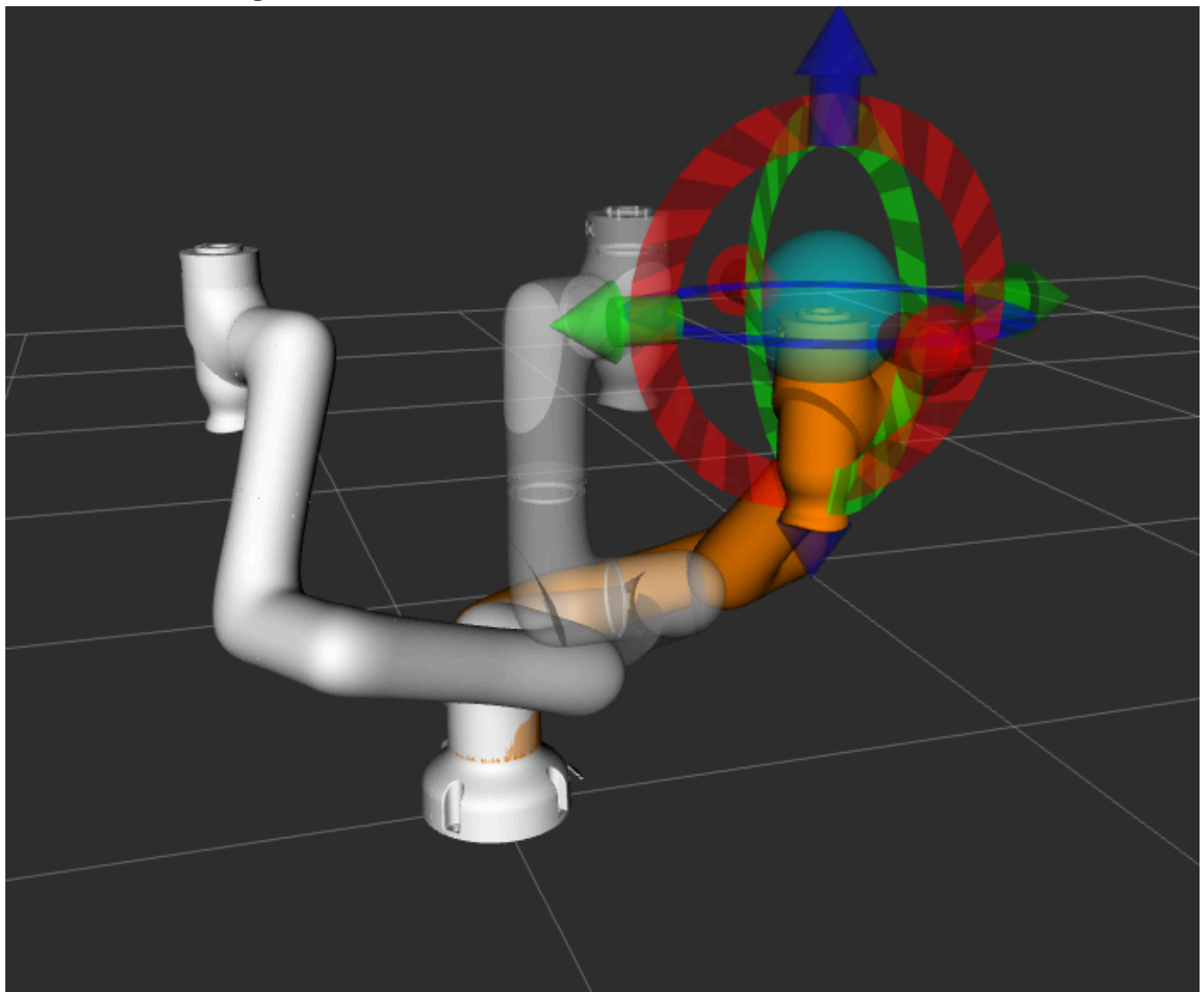


图3 根据MotionPlanning的joints 修改目标位置



rviz中目标姿态和实际姿态

(4)使用movej\_moveit\_test.cpp控制机械臂  
这是基于moveit规划的movej实现  
在保证上述launch正常运行的情况下，启动controll\_movej.launch.py

```
ros2 launch rokae_hardware controll_movej.launch.py robot_type:=SR4
```

!!! movej.cpp中注意以下修改!!!

```
/*149行    更换对应机型的基座(在相应机型srdf下)*/
arm.setPoseReferenceFrame("AR5-5_07R-W4C4A2_base");    //xxx_base    建议更改（注释可用）

/*158行    目标关节角度(注意轴数)*/
std::vector<double> joint_target = {0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5};

/*284行    更换对应的planning group(在相应机型srdf下)*/
auto move_group = std::make_shared<moveit::planning_interface::MoveGroupInterface>(node,
"AR5R_arm");
//xmate系列默认为rokae_arm
```

主要节点

- 在终端使用 ros2 node list 查询
- 具体每个节点信息使用 ros2 node info /<node\_name> 显示节点下的topic/service/action通信

节点名称	描述
/controller_manager	控制器节点，ROS2 自带，硬件接口以 plugin 的形式实现，并被 controller_manager 动态加载
/interactive_marker_display_100663865840352	Rviz 中交互式标记显示
/joint_state_broadcaster	关节状态广播器，读取机器人各关节的实际位置、速度和力矩，发布到 /joint_states 话题
/joint_state_publisher	关节状态发布者（当机器人没有硬件接口时使用）
/move_group	Movelt2 的核心规划节点
/move_group_private_105329056153616	Movelt2 的内部私有节点
/moveit_simple_controller_manager	连接 Movelt 规划器和实际的机器人控制器
/position_joint_trajectory_controller	位置关节轨迹控制器，接收 Movelt 规划出的关节轨迹，并发送给机器人硬件接口
/robot_state_publisher	机器人状态发布者，订阅 /joint_states 话题，根据机器人 URDF 计算 TF
/rviz2	Rviz2 节点
/rviz2_private_126572619198304	Rviz2 内部私有节点



节点名称	描述
/transform_listener_impl_5b8da1b44ef0	TF 变换监听器实现
/transform_listener_impl_5fcdb6902460	TF 变换监听器实现
/transform_listener_impl_731dfddfc690	TF 变换监听器实现

## 主要topic

- 在终端使用 `ros2 topic list` 查询
- 具体每个话题信息使用 `ros2 topic info /<topic_name>` 显示消息类型/发布者和订阅者数量
- 使用 `ros2 topic echo /<topic_name>` 输出消息

topic名称	描述
/joint_states	机器人的所有关节实时状态
/display_planned_path	规划的轨迹插值点信息
/position_joint_trajectory_controller/controller_state	控制器状态
/position_joint_trajectory_controller/joint_trajectory	向控制器发送轨迹命令（未使用，使用action通信）

## 主要Action

- 在终端使用 `ros2 action list` 查询
- 具体每个动作信息使用 `ros2 action info /<action_name>` 查询

action名称	描述
/execute_trajectory	执行路径，返回成功失败
/move_action	规划并执行路径，返回成功失败
/position_joint_trajectory_controller/follow_joint_trajectory	与MoveIt通信，发送控制器执行轨迹

## rokae\_driver 驱动机器人

rokae\_driver包负责低级别的机器人通信和控制, 使用ros2 service、topic通信，封装一些xCore API

### 已封装的功能

- service  
使用方法： `ros2 service call`/编写客户端  
服务消息位于/rokae\_msgs/srv

服务/函数名称	描述	消息类型
get_robot_info	获取机器人基本信息，如型号、SDK版本	<code>GetRobotInfo</code>

服务/函数名称	描述	消息类型
jog_control	Jog模式控制	JogCon
drag_control	拖动模式开启/关闭	DragCon
calculate_fk	计算正运动学（正解）	CalculateFK
calculate_ik	计算逆运动学（逆解）	CalculateIK
get_di	读取DI（数字输入）信号	GetDI
set_di	设置DI（数字输入）信号	SetDI
get_do	读取DO（数字输出）信号	GetDO
set_do	设置DO（数字输出）信号	SetDO
movej	MoveJ实时关节运动	MoveJ
movel	MoveL直线实时运动	MoveL
movec	MoveC圆弧实时运动	MoveC
read_register	读寄存器	ReadRegister
write_register	写寄存器	WriteRegister

- topic  
使用方法：ros2 topic echo/subscriber监听

话题名称	描述	消息类型
/rokae_driver/joint_states	发布机器人轴关节角度状态	sensor_msgs/msg/JointState
/rokae_driver/cartesian_pose	发布机器人笛卡尔空间位姿	geometry_msgs/msg/PoseStamped

启动方法

```
ros2 run rokae_hardware rokae_driver --ros-args -p robot_ip:=192.168.2.160 -p
local_ip:=192.168.2.100
ros2 launch rokae_hardware rokae_driver.launch.py robot_ip:=192.168.2.160
local_ip:=192.168.2.100
#launch或run 二选一即可，注意ip地址

##示例：get_robot_info
ros2 service call /rokae_driver/get_robot_info rokae_msgs/srv/GetRobotInfo
```

## 适配新的机型

- `robot_description`中导入相应rviz,mesh,xacro文件，文件格式可模仿现有机型
- `robot_description` 的urdf文件中xMate.urdf.xacro,xMate\_macro.xacro添加相应机型配置，并编写新机型 ros2\_controll.xacro文件，具体格式可参照现有文件机型
- 使用moveit\_setup\_assistant配置相应机型moveit\_config文件夹，存在区别的地方以现有机型格式为准

## Gazebo 仿真与轨迹开发

### 说明

- 下文示例中工作空间路径以 `~/ros2_ws`、源码目录以 `~/ros2_ws/src/rokae_ros2` 为例，请按本机实际路径替换；每个新终端均建议先执行：  
`source ~/ros2_ws/install/setup.bash`
- 关节命名：指令、轨迹、控制器配置在文档与示例中统一为 `joint1 ~ joint6`（六轴）或 `joint1 ~ joint7`（七轴）；各轴软限位、奇异与安全空间仍以 `rokae_description/urdf/` 中对应机型（含 `*_Gazebo*.urdf.xacro`）的 `<limit lower upper>` 为准；发轨迹前务必 `ros2 topic echo /joint_states` 核对轴数与关节名字。
- 当前存在的机型,只有CR35目前没有适配Gazebo相关功能的开发。

## 一、环境与场景支持

### 1.1 障碍场景一键加载

参数 `gazebo_world_file:=obstacles.world` 时，由 `test_model.launch.py` 载入 `rokae_gazebo/worlds/obstacles.world`（障碍物空间）。机型与场景解耦，只改 `robot_type` 即可。

终端 1 — 加载启动指令

```
source ~/ros2_ws/install/setup.bash

ros2 launch rokae_hardware test_model.launch.py robot_type:=Pro3
gazebo_world_file:=obstacles.world gui:=true
```

已支持机型（如 `SR3`、`SR4`、`SR5`、`CR7`、`CR12`、`CR18`、`CR20`、`ER3`、`ER7`、`Pro3`、`Pro7`、`AR5L`、`AR5R` 等）同理替换 `robot_type`。默认空世界为 `empty.world`；不加 `gazebo_world_file` 即空世界。

### 1.2 场景加载与保存服务

节点 `scene_service`（包 `rokae_hardware`）提供：

服务名	作用
<code>/load_scene</code>	按场景名查找 <code>.world</code> ，校验路径并登记为参数，供其它模块引用。
<code>/save_scene</code>	将模板 world 复制到用户目录，保存自定义场景快照。

服务类型：`rokae_hardware/srv/SceneService`，字段 `scene_name`（字符串）。

终端 1 — 启动带障碍的仿真（示例 SR3）

```
source ~/ros2_ws/install/setup.bash

ros2 launch rokae_hardware test_model.launch.py robot_type:=SR3
gazebo_world_file:=obstacles.world gui:=true
```

保持运行。

终端 2 — 启动场景服务

```
source ~/ros2_ws/install/setup.bash

ros2 run rokae_hardware scene_service
```

典型日志一行类似：

```
[INFO] [scene_manager]: 场景服务: /load_scene /save_scene (world 包: rokae_gazebo)
```

终端 3 — 调用 `/load_scene`（须在终端 2 已运行后执行）

```
source ~/ros2_ws/install/setup.bash

ros2 service call /load_scene rokae_hardware/srv/SceneService "{scene_name:
'obstacles.world'}"
```

成功时 `success: true`，`message` 中会给出解析后的 world 绝对路径；并提示：若需替换已在运行的 Gazebo 世界，应先关闭 Gazebo，再用 `launch` 的 `world` 参数启动该文件。

终端 3 — 调用 `/save_scene` 保存快照示例

```
ros2 service call /save_scene rokae_hardware/srv/SceneService "{scene_name:
'my_cr7_snapshot'}"
```

成功时 `message` 中会给出保存路径，一般在用户目录下 `~/ .rokae_gazebo/saved_worlds/<名称>.world`。

## 1.3 碰撞检测验证

障碍世界与 `arm_controller` 已运行后，可用 `rokae_gazebo` 包中的 `collision_test.launch.py` 订阅 `ContactsState`（默认 `/obstacle_bumper_contact`），在终端日志中查看碰撞体对。

终端 1 — 仿真（示例 SR5，与终端 2 的 `robot_type` 必须一致）

```
source ~/ros2_ws/install/setup.bash

ros2 launch rokae_hardware test_model.launch.py robot_type:=SR5
gazebo_world_file:=obstacles.world gui:=true
```

终端 2 — 碰撞监听

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 launch rokae_gazebo collision_test.launch.py robot_type:=SR5 publish_motion:=false  
contact_topic:=/obstacle/bumper_contact
```

- `publish_motion:=false`: 不在本节点内自动发轨迹, 靠终端 1 的滑条或外部轨迹顶障碍。
- 默认按 `robot_collision_substring:=xMate` 过滤与机械臂无关的接触 (可在 launch 参数中调整)。

启动后终端 2 可能出现类似:

```
[INFO] [collision_test_node]: 碰撞监听: /obstacle/bumper_contact; 轨迹话题:  
/arm_controller/joint_trajectory (请已启动 arm_controller)
```

发生碰撞时, 日志中可能出现:

```
[WARN] [collision_test_node]: 检测到新碰撞接触 ... 碰撞体: [table::link::collision] <->  
[xMateSR5::xMateSR5_link2::xMateSR5_link2_collision]
```

## 二、ros2\_control 集成与轨迹控制

### 2.1 仿真控制闭环

- Gazebo 插件 `libgazebo_ros2_control.so` 启动 `controller_manager`, 与 ROS 2 `ros2_control` 栈对接。
- `test_model.launch.py` 延时后通过 `spawner` 依次加载:
  - `joint_state_broadcaster`: 读仿真关节状态, 发布 `/joint_states`;
  - `arm_controller` (`JointTrajectoryController`): 订阅 `/arm_controller/joint_trajectory`, 将 `trajectory_msgs/JointTrajectory` 转为关节位置指令驱动仿真。
- 链路: ROS 2 轨迹话题 → 轨迹控制器 → Gazebo 关节。

### 2.2 controll\_movej.launch.py

在 Gazebo 中加载机型 URDF、`gazebo_ros2_control`、`joint_state_broadcaster` 与 `arm_controller`; 相对 `test_model.launch.py` 另可带入 MoveIt 配置, 并可选 `movej` 演示节点。

重要: 勿同时 `enable_gui:=true` 与 `enable_movej:=true`, 避免多源争抢同一轨迹控制器。

模式 A — GUI 滑条 (终端 1)

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 launch rokae_hardware controll_movej.launch.py robot_type:=SR3 enable_gui:=true  
enable_movej:=false
```

约 5~6 秒后再拖动 `joint_state_publisher_gui` 滑条; 滑条经 `gui_to_joint_trajectory` 打成短时轨迹发往 `/arm_controller/joint_trajectory`。关闭 `joint_state_publisher_gui` 界面后, 也可直接切换到模式 B。

模式 B — movej 演示 + 终端发轨迹 (终端 1)

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 launch rokae_hardware control1_movej.launch.py robot_type:=SR3 enable_gui:=false  
enable_movej:=true
```

另开终端发 `ros2 topic pub ...` (见 2.4) 。

## 2.3 验证 /joint\_states

验证时确保终端 1 已启动 Gazebo 场景，且机型已成功加载显示。

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 topic echo /joint_states --once
```

在统一命名下，`name` 字段应为 `joint1...joint6` 或含 `joint7` (七轴)，与 `rokae_hardware/config/xMate{机型}_controllers.yaml` 中控制器 `joints` 列表一致。

## 2.4 轨迹话题与 ros2 topic pub 示例

(1) 确认当前由谁在订阅轨迹

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 control list_controllers
```

```
ros2 topic list | grep joint_trajectory
```

- Gazebo `test_model / control1_movej` 场景下，常见为 `/arm_controller/joint_trajectory`。
- 若仅 `position_joint_trajectory_controller` 为 active (例如单独使用 `rokae_moveit_launch.py` 时默认拉起该控制器)，则应向 `/position_joint_trajectory_controller/joint_trajectory` 发布。

(2) 六轴 — 终端 1

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 launch rokae_hardware control1_movej.launch.py robot_type:=SR3 enable_gui:=false  
enable_movej:=false
```

终端 2 — 两段路点 (单位 rad)

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 topic pub --once /arm_controller/joint_trajectory trajectory_msgs/msg/JointTrajectory "  
joint_names: [joint1, joint2, joint3, joint4, joint5, joint6]  
points:  
- positions: [0.20, -0.20, 0.15, 0.00, 0.20, 0.00]  
  time_from_start: {sec: 3, nanosec: 0}  
- positions: [0.20, -0.55, 0.25, 0.10, 0.30, 0.10]  
  time_from_start: {sec: 6, nanosec: 0}  
"
```

### (3) 七轴 — `joint_names` 与每条 `positions` 须为 7 维

```
ros2 topic pub --once /arm_controller/joint_trajectory trajectory_msgs/msg/JointTrajectory "  
joint_names: [joint1, joint2, joint3, joint4, joint5, joint6, joint7]  
points:  
- positions: [0.10, -0.20, 0.15, 0.00, 0.20, 0.00, 0.00]  
  time_from_start: {sec: 3, nanosec: 0}  
- positions: [0.20, -0.35, 0.25, 0.10, 0.30, 0.10, 0.00]  
  time_from_start: {sec: 6, nanosec: 0}  
"
```

### (4) 各机型关节角软限位参考（弧度）

下表数值来自仓库 Gazebo xacro 中 `<limit>`，仅作参考指令前粗查；改 URDF 后以文件为准。轴名统一为 `joint1 ~ joint6`（及 `joint7`）顺序对应机械第 1~7 轴。

机型	joint1	joint2	joint3	joint4	joint5	joint6	joint7
SR3	[-3.0543, 3.0543]	[-2.3562, 2.2689]	[-3.0543, 2.3562]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	—
SR4	[-3.0543, 3.0543]	[-2.3562, 2.3562]	[-2.3562, 2.3562]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	—
SR5	[-6.2832, 6.2832]	[-2.7925, 2.618]	[-2.9671, 2.4435]	[-6.2832, 6.2832]	[-6.2832, 6.2832]	[-6.2832, 6.2832]	—
CR7	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	—
CR12	[-6.2832, 6.2832]	[-2.9671, 2.9671]	[-6.2832, 6.2832]	[-6.2832, 6.2832]	[-6.2832, 6.2832]	[-6.2832, 6.2832]	—
CR18	[-3.0543, 3.0543]	[-2.9671, 2.9671]	[-2.8798, 2.8798]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	—
CR20	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-2.9671, 2.9671]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	[-3.0543, 3.0543]	—
ER3	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-2.0944, 2.0944]	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-6.2832, 6.2832]	—
ER7	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-2.0944, 2.0944]	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-6.2832, 6.2832]	—

机型	joint1	joint2	joint3	joint4	joint5	joint6	joint7
Pro3 / Pro7	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-2.9671, 2.9671]	[-2.0944, 2.0944]	[-6.2832, 6.2832]
AR5L / AR5R	[-3.1067, 3.1067]	[-2.0944, 2.0944]	[-3.1067, 3.1067]	[-1.0472, 2.5307]	[-3.1067, 3.1067]	[-1.0472, 1.0472]	[-1.0472, 1.0472]

### 三、一键 launch（Gazebo 仿真 / 真机指令驱动）

#### 3.1 入口与内部链路

```
ros2 launch rokae_hardware gazebo_moveit.launch.py ...
```

条件	子 launch	作用
<code>mode:=sim</code>	<code>control_movej.launch.py</code>	Gazebo + <code>gazebo_ros2_control</code> ; <code>world</code> → 子 launch 的 <code>gazebo_world_file</code> ; 可选 <code>enable_gui</code> / <code>enable_movej</code> 。
<code>mode:=real</code>	<code>real_moveit.launch.py</code>	本机 <code>ros2_control_node</code> + 真机 IP, 加载 <code>xMate{机型}_real_controllers.yaml</code> ; 不启动 Gazebo。

#### 3.2 参数约束

- 1. `mode:=real` 时 `use_sim_time:=false`（真机用系统时钟）。
- 2. `mode:=sim` 时不可同时 `enable_gui:=true` 与 `enable_movej:=true`。
- 3. `mode:=real` 必须同时提供 `robot_ip` 与 `local_ip`。

#### 3.3 仿真模式示例

空世界 + 滑条（SR3）

```
source ~/ros2_ws/install/setup.bash

ros2 launch rokae_hardware gazebo_moveit.launch.py mode:=sim robot_type:=SR3
world:=empty.world enable_gui:=true enable_movej:=false
```

障碍世界

```
ros2 launch rokae_hardware gazebo_moveit.launch.py mode:=sim robot_type:=CR7
world:=obstacles.world enable_gui:=true enable_movej:=false
```

开 `movej` 演示、关 GUI

```
ros2 launch rokae_hardware gazebo_moveit.launch.py mode:=sim robot_type:=CR7
world:=empty.world enable_gui:=false enable_movej:=true
```



### 3.4 真机模式示例

终端 1 — 启动真机控制栈（无 Gazebo）

```
source ~/ros2_ws/install/setup.bash

ros2 launch rokae_hardware gazebo_moveit.launch.py mode:=real robot_type:=CR7
robot_ip:=192.168.2.160 local_ip:=192.168.2.162 enable_moveit:=false use_sim_time:=false
```

终端 2 — 发关节轨迹（六轴）

```
source ~/ros2_ws/install/setup.bash

ros2 topic pub --once /arm_controller/joint_trajectory trajectory_msgs/msg/JointTrajectory "
joint_names: [joint1, joint2, joint3, joint4, joint5, joint6]
points:
- positions: [0.10, -0.20, 0.15, 0.00, 0.20, 0.00]
  time_from_start: {sec: 3, nanosec: 0}
- positions: [0.20, -0.35, 0.25, 0.10, 0.30, 0.10]
  time_from_start: {sec: 6, nanosec: 0}
"
```

## 四、仿真数据录制与回放

### 4.1 脚本说明

- `rokae_hardware/scripts/record_sim.sh`
  - 用法: `bash .../record_sim.sh [bag输出目录] [robot_type可选]`
  - 默认在 `~/rosbags/<robot小写>_sim_日期时间>` 创建；脚本会等待 `/arm_controller/joint_trajectory` 或 `/position_joint_trajectory_controller/joint_trajectory` 出现后再开始 `ros2 bag record`，录制 `/joint_states`、两类控制器的 `.../joint_trajectory` 与 `follow_joint_trajectory` action 各话题、`/tf`、`/tf_static`、`/clock` 等（完整列表见脚本内）。
- `rokae_hardware/scripts/replay_sim.sh`
  - 用法: `bash .../replay_sim.sh <bag目录> [倍速rate] [with_clock true|false]`
  - 默认 `rate=1.0`；第三参数为 `true` 时增加 `ros2 bag play --clock`；仅回放与轨迹相关的话题子集。

### 4.2 推荐流程（三终端）

以下以 `robot_type:=CR7`、bag 目录 `~/rosbags/cr7_sim0002` 为例；请替换为本机路径与机型。

终端 1 — 启动仿真（无滑条，与录制脚本习惯一致）

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 launch rokae_hardware gazebo_moveit.launch.py mode:=sim robot_type:=CR7  
world:=empty.world enable_gui:=false enable_movej:=false
```

保持运行。

终端 2 — 开始录制

```
source ~/ros2_ws/install/setup.bash
```

```
bash ~/ros2_ws/src/rokae_ros2/rokae_hardware/scripts/record_sim.sh ~/rosbags/cr7_sim0002 CR7
```

结束录制：在本终端按 Ctrl+C。

终端 3 — 录制过程中下发轨迹（六轴；关节名统一）

```
source ~/ros2_ws/install/setup.bash
```

```
ros2 topic pub --once /arm_controller/joint_trajectory trajectory_msgs/msg/JointTrajectory "  
joint_names: [joint1, joint2, joint3, joint4, joint5, joint6]  
points:  
- positions: [0.10, -0.20, 0.15, 0.00, 0.20, 0.00]  
  time_from_start: {sec: 3, nanosec: 0}  
- positions: [0.30, -0.35, 0.25, 0.10, 0.30, 0.10]  
  time_from_start: {sec: 6, nanosec: 0}  
"
```

录制期间可多次执行 `ros2 topic pub`。七轴须 `joint7` 且 `positions` 为 7 个数。

## 4.3 检查 bag

```
source ~/ros2_ws/install/setup.bash
```

```
ls ~/rosbags/
```

```
ros2 bag info ~/rosbags/cr7_sim0002
```

## 4.4 回放

1. 终端 1 再次启动与录制时相同的仿真（同一 `robot_type`）。
2. 终端 2 — 原速回放

```
source ~/ros2_ws/install/setup.bash
```

```
bash ~/ros2_ws/src/rokae_ros2/rokae_hardware/scripts/replay_sim.sh ~/rosbags/cr7_sim0002
```

3. 二倍速

```
bash ~/ros2_ws/src/rokae_ros2/rokae_hardware/scripts/replay_sim.sh ~/rosbags/cr7_sim0002 2.0
```

4. 带 `--clock` (第三参数 `true`)

```
bash ~/ros2_ws/src/rokae_ros2/rokae_hardware/scripts/replay_sim.sh ~/rosbags/cr7_sim0002 1.0 true
```

## 4.5 使用注意

1. 回放前 `robot_type`、URDF、关节轴数须与录制时一致。
2. 用 `ros2 control list_controllers` 确认轨迹控制器为 active。
3. 回放异常时检查 `use_sim_time`、bag 是否含 `/clock`、以及 `replay_sim.sh` 第三参数。
4. 建议保留 `ros2 bag info` 输出备查。