法奥

发行版本 *1.0*

法奥意威（苏州）机器人系统有限公司

2023 年 11 月 27 日

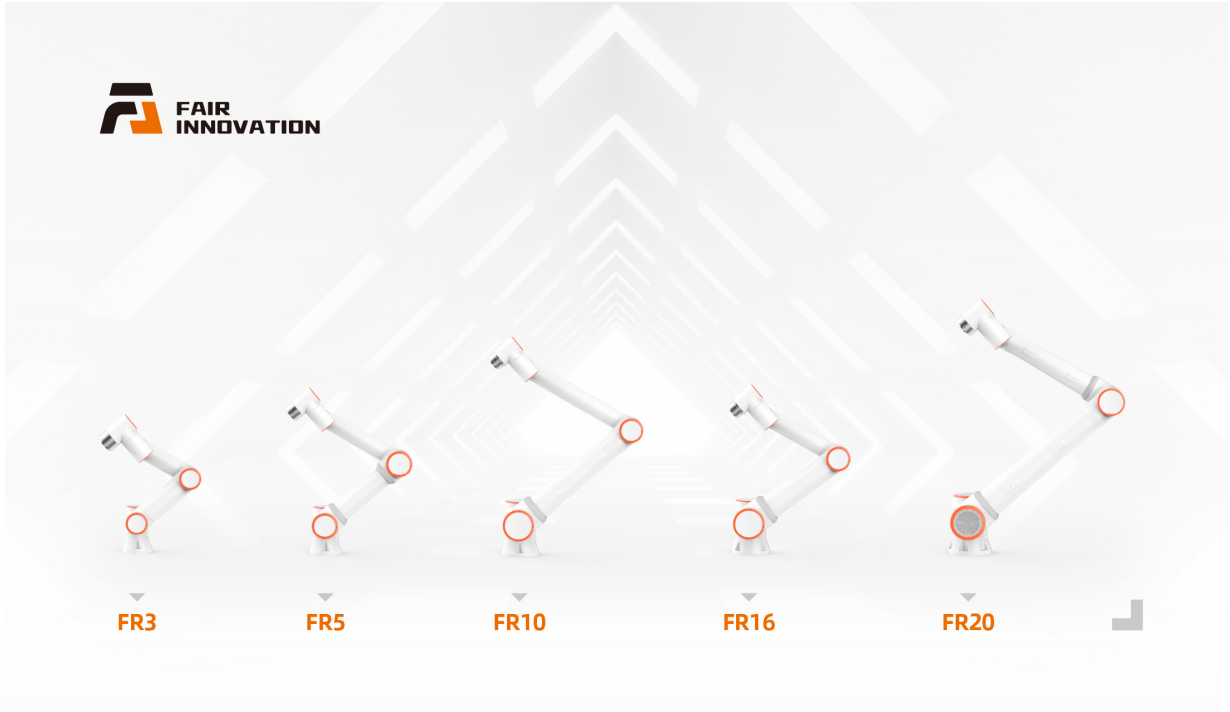
用户手册

FAIR ROBOTICS

CHAPTER 1

协作机器人

1.1 产品矩阵



1.2 快速开始

1.2.1 机器人安装上电

1.2.1.1 安装机器人手臂

协作机器人安装在安装座上时，使用合规数量螺栓（强度不低于 8.8 级）将机器人拧紧固定在安装座上；建议安装座上使用两个合规销孔配合销钉进行机器人定位，以提高机器人安装精度，防止因为碰撞等使机器人发生移动。当机器人有较高运行精度要求时，请务必增加销钉对机器人进行定位。

表格 1.1-1 机器人安装零件标准

协作机器人型号	螺栓	螺栓扭矩	销孔规格
FR3	4 颗 M6	≥10Nm	φ5mm
FR5	4 颗 M8	≥20Nm	φ8mm
FR10	4 颗 M8	≥25Nm	φ8mm
FR16	4 颗 M8	≥25Nm	φ8mm
FR20	6 颗 M10	≥45Nm	φ8mm

重要： 推荐机器人安装座满足以下几个要求，以保证机器人安装牢固、稳定：

（1）机器人安装座需要足够牢固且有足够的承载能力，应该至少能承载 5 倍的机器人重量，至少能承受 10 倍的 1 轴扭矩。

（2）机器人安装座应表面平整，以保证与机器人接触面紧密接触；

（3）机器人安装座应刚度足够强壮，固定牢固，不会和机器人发生共振；

（4）机器人和其他部件同时运动时，安装座与其他运动部件应隔离开，不要固定在一起避免运动过程中的振动干扰；

（5）如果机器人安装在移动平台或者外部轴上，移动平台或者外部轴的加速度应尽量低；

1.2.1.2 连接控制箱

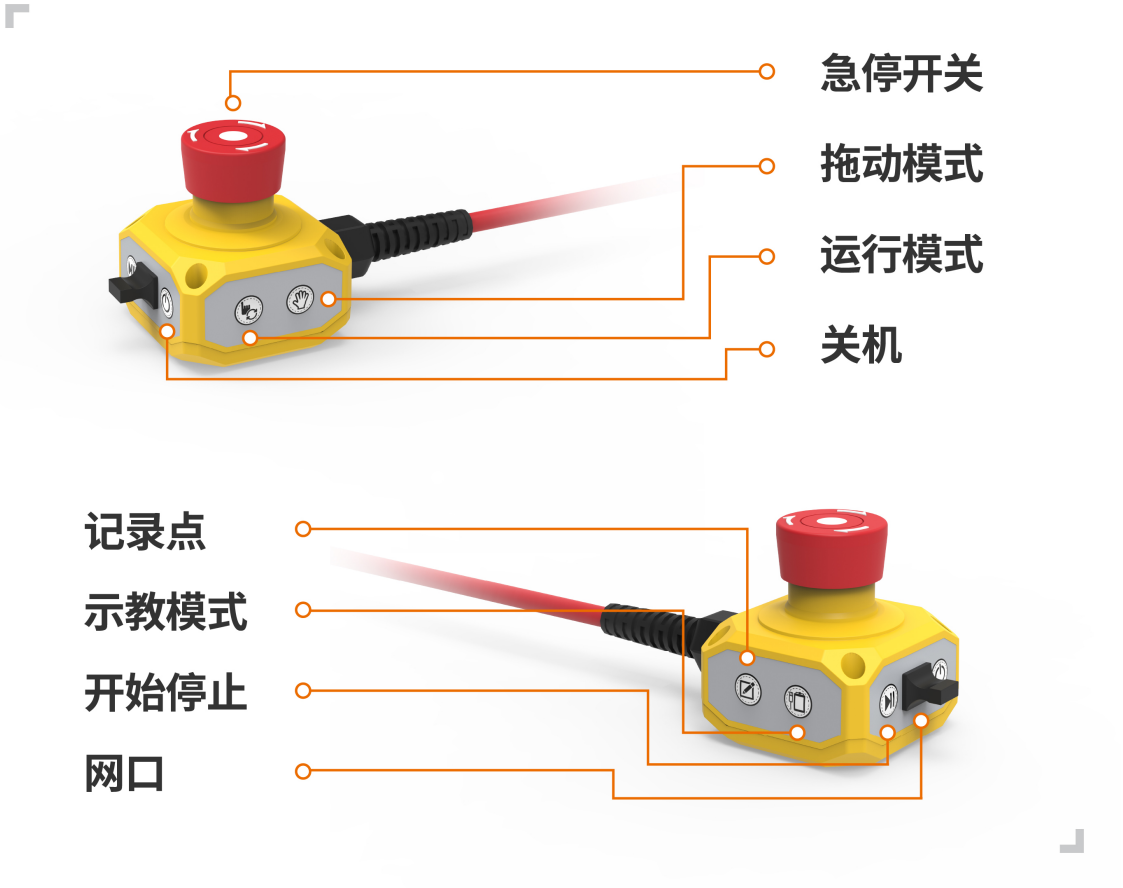
本系列机器人采用 TN-S 单相 220V 交流电源供电，设备自带 5 米电源线，三脚插头端插入现场提供的交流 220V 插座，机器人电气接地。机械手控制系统的外部连线均使用可插拔可快速安装的插头进行连接。协作机器人接线面板如下图表：

图表 1.2-1 控制箱接线面板

按钮盒接口默认为示教器控制端口，IP 地址为 192.168.58.2，使用网线连接按钮盒接口与电脑，电脑 IP 地址设为 192.168.58.10 或与之同一网段，打开谷歌浏览器输入 192.168.58.2 即可访问示教器页面。

1.2.1.3 认识按钮盒及末端 LED

1.2.1.3.1 按钮盒



图表 1.3-1 控制箱接线面板

表格 1.3-1 控制箱接线面板按键说明

按键名	功能
急停开关	当按下急停开关，机器人进入紧急停止状态
开始停止	开始/停止运行程序
网口	连接 web 示教器
关机	暂未启用
记录点	记录示教点
示教模式	进入/退出搭配示教器状态
运行模式	自动/手动模式切换
拖动模式	进入/退出拖动模式

1.2.1.3.2 末端 LED

表格 1.3-2 末端 LED 定义

功能	LED 颜色
通信未建立时	“灭”“红”“绿”“蓝”交替
自动模式	蓝色长亮
手动模式	绿色长亮
拖动模式	白青色长亮
按钮盒记录点（仅在使用按钮盒时）	紫色闪烁两下
进入未搭配按钮盒状态（仅在使用按钮盒时）	青蓝色闪烁两下
开始运行（仅在使用按钮盒时）	蓝色闪烁两下
停止运行（仅在使用按钮盒时）	红色闪烁两下
报错（仅在使用按钮盒时）	红色长亮
校零完成	白青色闪烁三下
去使能	黄色闪烁两下

1.2.1.4 上电使能

上电前，请确认按钮盒急停按钮处理松开状态，按下控制箱红色开关按钮上电，使能成功后末端 LED 灯处于绿色常亮状态。

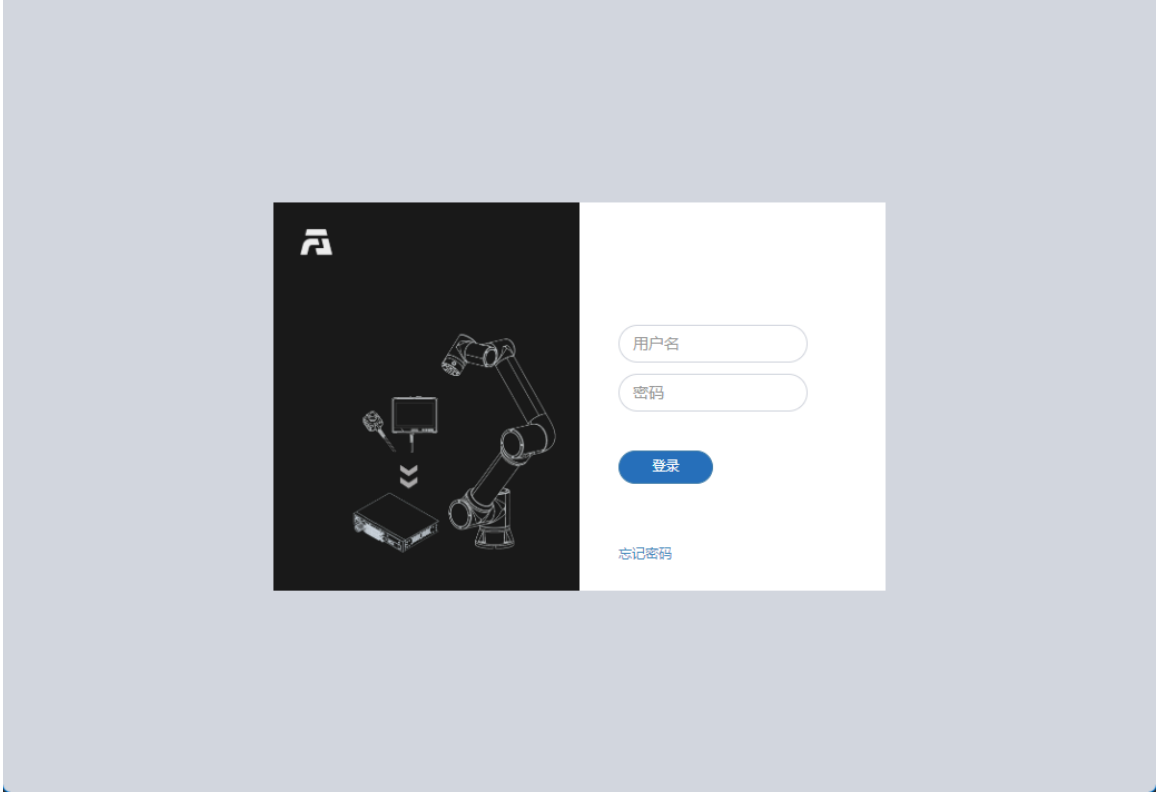
1.2.2 WebApp 访问登录

1.2.2.1 访问登录 WebApp 界面

1. 开启控制箱并将网线连接 PC；
2. PC 打开 chrome 浏览器访问目标网址 192.168.58.2；
3. 输入用户名和密码点击登录即可登录 WebApp。

初始用户名为 admin，密码为 123。

图表 2.1-1 登录界面



1.2.2.2 简单认识 WebApp 界面

登录成功后系统进入“初始界面”，初始界面展示了示教器主要包含法奥 LOGO 及返回初始页面按钮、菜单栏、菜单栏缩放按钮、机器人操作区、控制区、状态区、三维模拟机器人以及位姿及 IO 信息区，一共八个区域。如下图系统初始界面示意图所示：

图表 2.2-1 系统初始界面示意图

1.2.2.2.1 控制区

备注:

名称：**使能按钮**

作用：使能机器人

备注:

名称：**开始按钮**

作用：上传并开始运行示教程序

备注:

名称：**停止按钮**

作用：停止当前示教程序运行

备注:

名称：**暂停/恢复按钮**

作用：暂停和恢复当前示教程序

重要: 暂停指令在程序的末尾，无法进行判断

1.2.2.2.2 状态栏

备注:

名称：**机器人状态**

作用：Stopped-停止，Running-运行，Pause-暂停，Drag-拖动

备注:

名称：**工具坐标系编号**

作用：展示当前应用的工具坐标系编号

备注:

名称：**运行速度百分比**

作用：机器人当前模式运行时速度

备注:

名称：**机器人运行正常状态**

作用：当前机器人正常运行

备注:

名称：**机器人运行错误状态**

作用：当前机器人运行有错误

备注:

名称：**自动模式**

作用：机器人自动运行模式，开启手动切自动模式全局速度调整并指定速度时，全局速度会自动调整为指定速度

备注:

名称：**示教模式**

作用：机器人示教运行模式

备注:

名称：**机器人拖动状态**

作用：当前机器人可拖动

备注:

名称：**机器人拖动状态**

作用：当前机器人不可拖动

备注:

名称：**连接状态**

作用：机器人已连接

备注:

名称：**未连接状态**

作用：机器人未连接

备注:

名称：**账户信息**

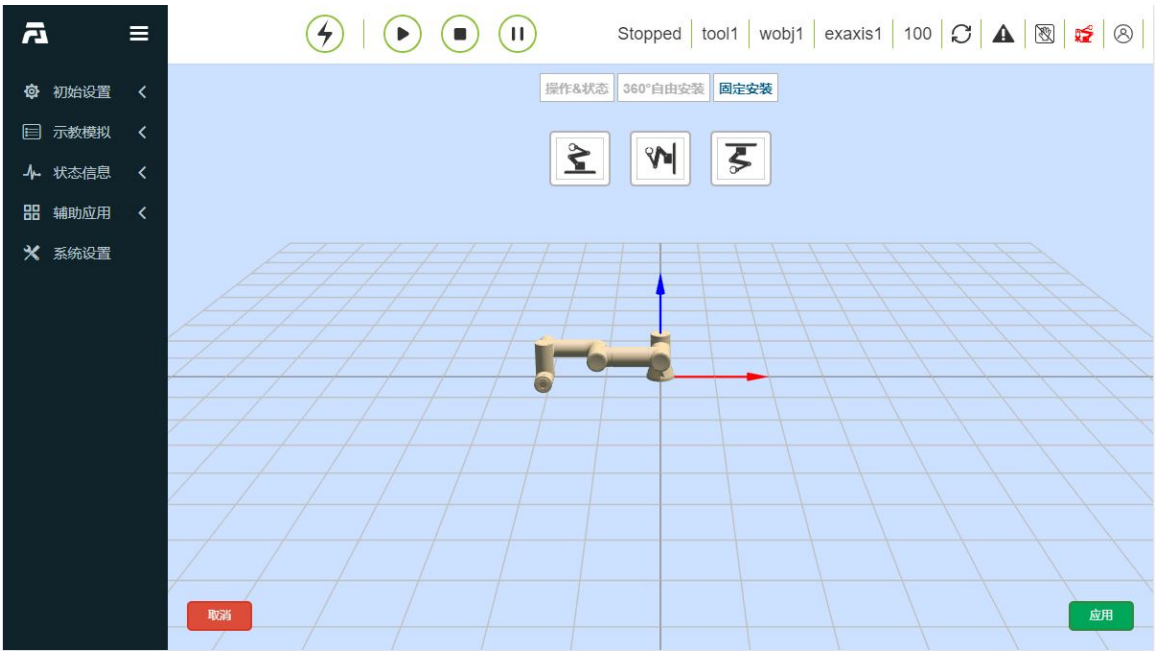
作用：显示用户名和权限及登出用户

1.2.3 机器人参数设置

1.2.3.1 设置安装方式

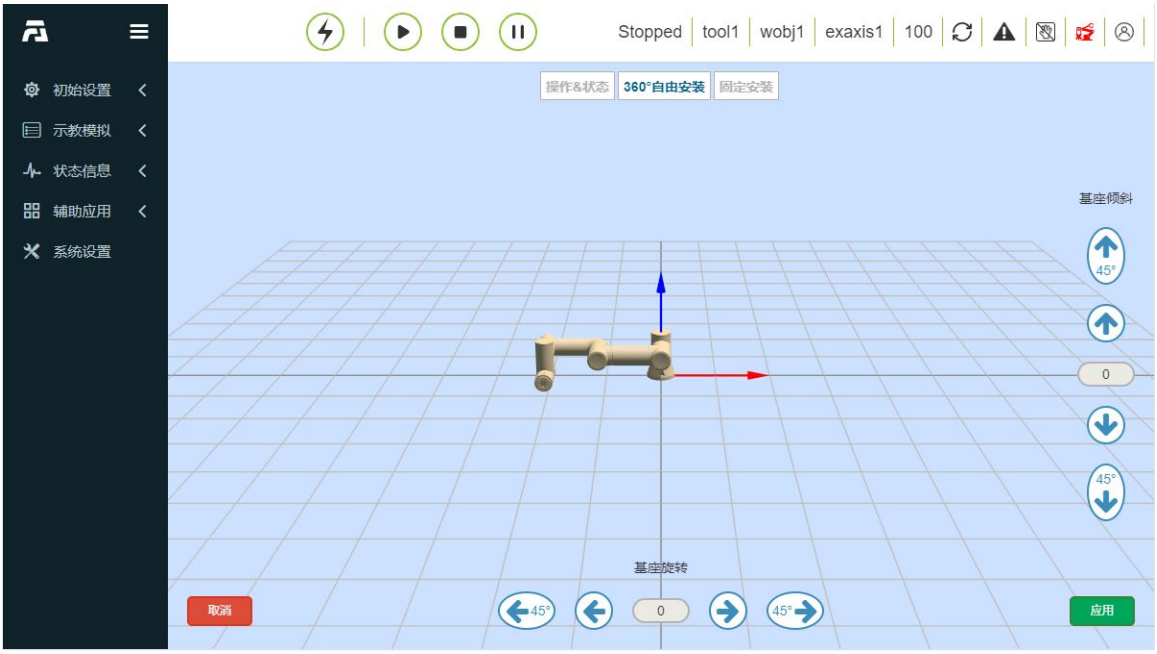
机器人默认安装方式为水平安装，当机器人安装方式更改时，需及时在此页面设置机器人的实际安装方式，以保证机器人正常工作。

用户点击机器人三维虚拟展示区域中的“固定安装”选项卡，进入机器人固定安装方式设置页面，选择“正装”、“倒装”或者“侧装”，点击“应用”按钮完成机器人安装方式设置。



图表 3.1-1 固定安装

考虑到更加灵活丰富的机器人部署场景，我们提供了自由安装功能，用户点击机器人三维虚拟展示区域中的“360 度自由安装”选项卡，进入机器人自由安装方式设置页面。手动调整“基座倾斜”和“基座旋转”角度，三维模型会对应展示安装效果。修改后点击“应用”按钮即可完成机器人安装方式设置。



图表 3.1-2 360 度自由安装

重要： 机器人安装完成后，必须正确设置机器人的安装方式，否则会影响机器人的拖动功能以及碰撞检测功能使用。

1.2.3.2 设置末端负载

在“初始设置”中的“机器人设置”的菜单栏下，点击“末端负载”进入末端负载页面。

图表 3.2-1 负载设定示意图

在配置末端负载时请将所使用的末端工具的质量以及对应的质心坐标分别输入“负载质量”和“负载质心坐标 X、Y 和 Z”输入框中并应用。

重要： 负载质量不可超过机器人的最大负载范围。对应型号负载如下：

- FR3：3kg

- FR5：5kg
- FR10：10kg
- FR16：16kg
- FR20：20kg

质心坐标设置范围为 0-1000，单位 mm。

重要： 机器人末端安装负载后，必须正确设置末端负载重量以及质心坐标，否则会影响机器人的拖动功能以及碰撞检测功能使用。

1.2.3.3 设置工具坐标

在“初始设置”中的“机器人设置”的菜单栏下，点击“工具坐标”进入工具坐标页面。工具坐标可实现工具坐标的修改、清空与应用。工具坐标系的下拉列表中共有 15 个编号，选择对应的坐标系（坐标系名称可自定义）后会在下方显示对应坐标值，工具类型以及安装位置（仅在传感器类型工具下显示），选择某一坐标系后点击“应用”按钮，当前使用的工具坐标系变为所选择的坐标，如下所示。

点击“修改”可根据提示对该编号的工具坐标系进行重新设置。工具标定方法分为四点法和六点法，四点法只标定工具 TCP，即工具中心点的位置，其姿态默认与末端姿态一致，六点法则在四点法的基础上增加了两点，用于标定工具的姿态。

图表 3.3-1 设置工具坐标

图表 3.3-2 设置工具坐标

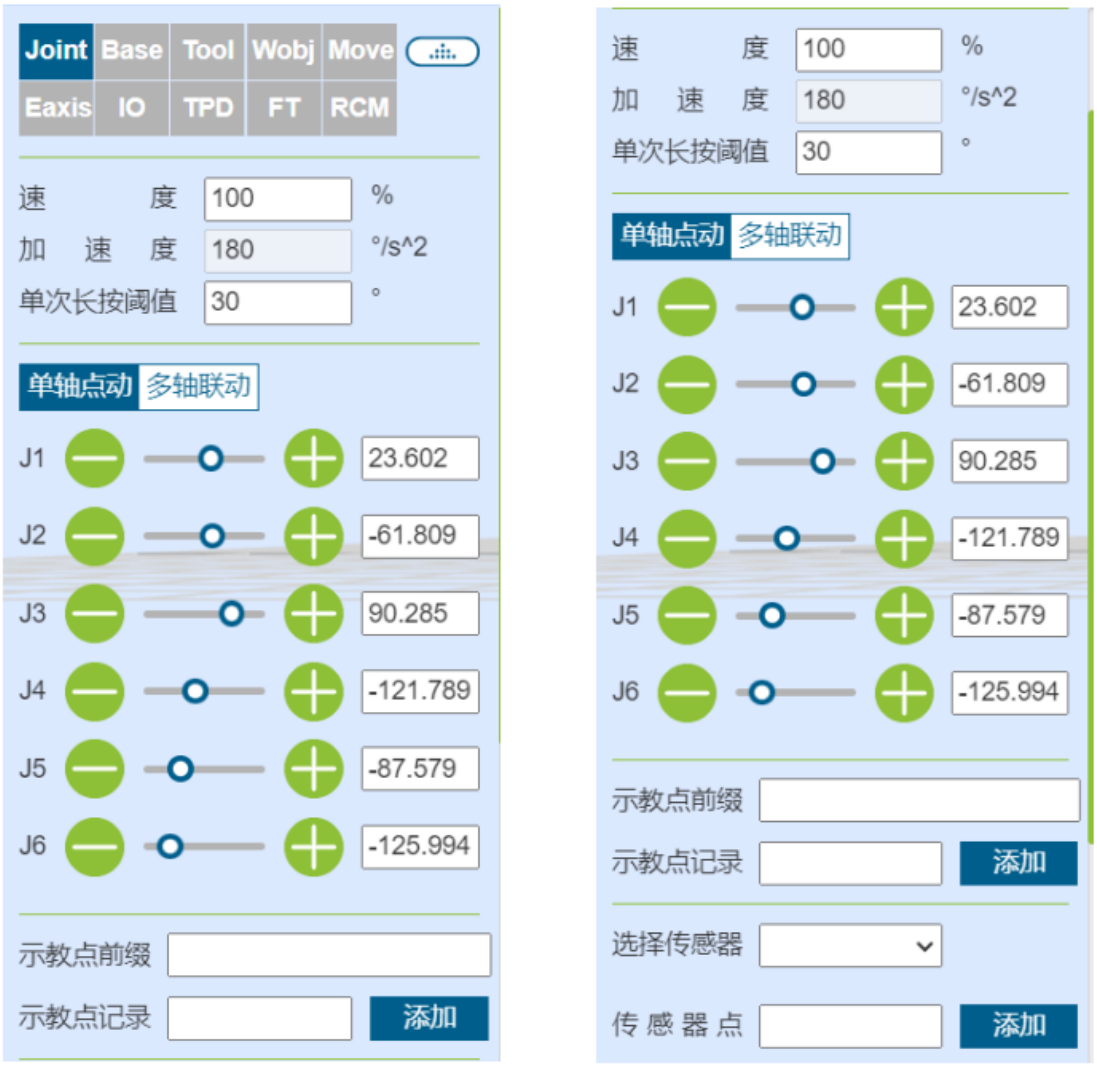
重要： 1. 末端安装工具后，必须要进行工具坐标系的标定及应用，否则会导致机器人执行运动指令时工具中心点的位置和姿态不符合预期值。

2. 工具坐标系一般使用 toolcoord1~toolcoord14, 应用 toolcoord0 代表工具 TCP 的位置中心在末端法兰中心，在进行工具坐标系标定时，首先需将工具坐标系应用至 toolcoord0，然后选择其他工具坐标系进行标定及应用。

1.2.4 机器人手动示教

1.2.4.1 手动示教并记录示教点

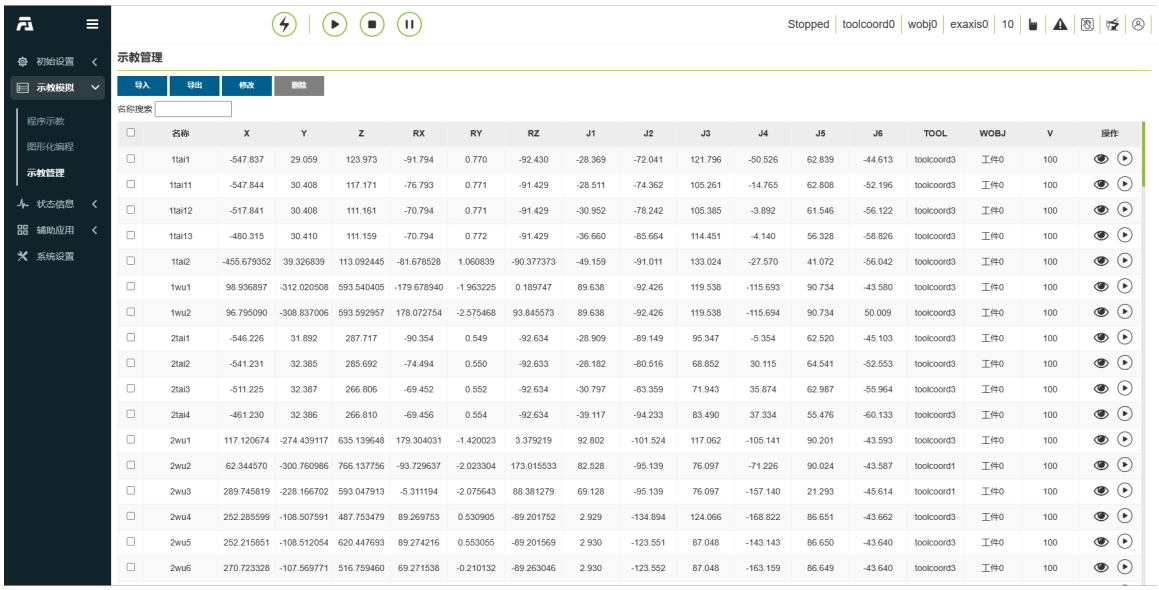
手动示教包含两种方式，一种是按住末端拖动按钮进行拖动示教，一种是在操作区进行点动。示教到目标位置后，即可保存示教点。保存示教点时，该示教点的坐标系为当前机器人应用的坐标系。在该操作区上方可以对示教点速度，加速度设置，设置数值为机器人标准速度百分比，若设置 100，即标准速度的百分之百。



图表 4.1-1 手动示教

1.2.4.2 查看示教点信息

点击“示教管理”可显示所有保存的示教点信息，在该界面中可对示教点文件导入和导出，选中一个示教点后点击“删除”按钮即可将该点信息删除，示教点 x,y,z,rx,ry,rz 和 v 数值可进行修改，输入修改值，勾选左侧勾选兰，点击上方修改即可修改示教点信息。点击“开始运行”按钮，进行局部示教点的单点运行，将机器人移动到该点的位置。此外，用户可以通过名称搜索示教点。



	名称	X	Y	Z	RX	RY	RZ	J1	J2	J3	J4	J5	J6	TOOL	WOBJ	V	操作
☐	1tai1	-547.837	29.059	123.973	-91.794	0.770	-92.430	-28.369	-72.041	121.796	-50.526	62.839	-44.613	toolcoord3	工件0	100	
☐	1tai11	-547.844	30.408	117.171	-76.793	0.771	-91.429	-28.511	-74.362	105.261	-14.765	62.808	-52.196	toolcoord3	工件0	100	
☐	1tai12	-517.841	30.408	111.161	-70.794	0.771	-91.429	-30.952	-78.242	105.385	-3.892	61.546	-56.122	toolcoord3	工件0	100	
☐	1tai13	-480.315	30.410	111.159	-70.794	0.772	-91.429	-36.660	-85.664	114.451	-4.140	56.328	-58.826	toolcoord3	工件0	100	
☐	1tai2	-455.679352	39.326839	113.092445	-81.678528	1.060839	-90.377373	-49.159	-91.011	133.024	-27.570	41.072	-56.042	toolcoord3	工件0	100	
☐	1wu1	98.936897	-312.020508	593.540405	-179.678940	-1.963225	0.189747	89.638	-92.426	119.538	-115.693	90.734	-43.580	toolcoord3	工件0	100	
☐	1wu2	96.795090	-308.837006	593.592957	178.072754	-2.575468	93.845573	89.638	-92.426	119.538	-115.694	90.734	50.009	toolcoord3	工件0	100	
☐	2tai1	-546.226	31.892	287.717	-90.354	0.549	-92.634	-28.909	-89.149	95.347	-5.354	62.520	-45.103	toolcoord3	工件0	100	
☐	2tai2	-541.231	32.385	285.692	-74.494	0.550	-92.633	-28.182	-80.516	68.852	30.115	64.541	-52.553	toolcoord3	工件0	100	
☐	2tai3	-511.225	32.387	266.806	-69.452	0.552	-92.634	-30.797	-83.359	71.943	35.874	62.987	-55.964	toolcoord3	工件0	100	
☐	2tai4	-461.230	32.386	266.810	-69.456	0.554	-92.634	-39.117	-94.233	83.490	37.334	55.476	-60.133	toolcoord3	工件0	100	
☐	2wu1	117.120674	-274.439117	635.139648	179.304031	-1.420023	3.379219	92.802	-101.524	117.062	-105.141	90.201	-43.593	toolcoord3	工件0	100	
☐	2wu2	62.344570	-300.760986	766.137756	-93.729637	-2.023304	173.015533	82.528	-95.139	76.097	-71.226	90.024	-43.587	toolcoord1	工件0	100	
☐	2wu3	289.745819	-228.166702	593.047913	-5.311194	-2.075643	88.381279	69.128	-95.139	76.097	-157.140	21.293	-45.614	toolcoord1	工件0	100	
☐	2wu4	252.285599	-108.507591	487.753479	89.269753	0.530905	-89.201752	2.929	-134.894	124.066	-168.822	86.651	-43.662	toolcoord3	工件0	100	
☐	2wu5	252.215851	-108.512054	620.447693	89.274216	0.553055	-89.201569	2.930	-123.551	87.048	-143.143	86.650	-43.640	toolcoord3	工件0	100	
☐	2wu6	270.723328	-107.569771	516.759460	69.271538	-0.210132	-89.263046	2.930	-123.552	87.048	-163.159	86.649	-43.640	toolcoord3	工件0	100	

图表 4.2-1 示教管理界面

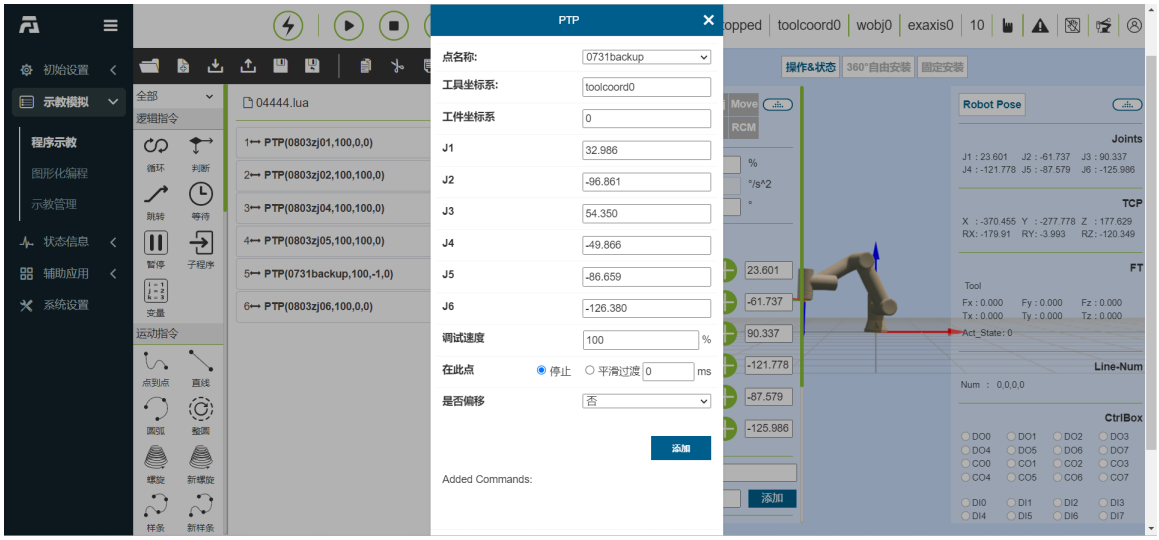
重要： 示教点 x,y,z,rx,ry,rz 的修改值不应超过机器人的工作范围。

1.2.5 机器人快速编程

1.2.5.1 简单运动指令介绍

PTP 命令：点击“点到点”图标进入 PTP 命令编辑界面

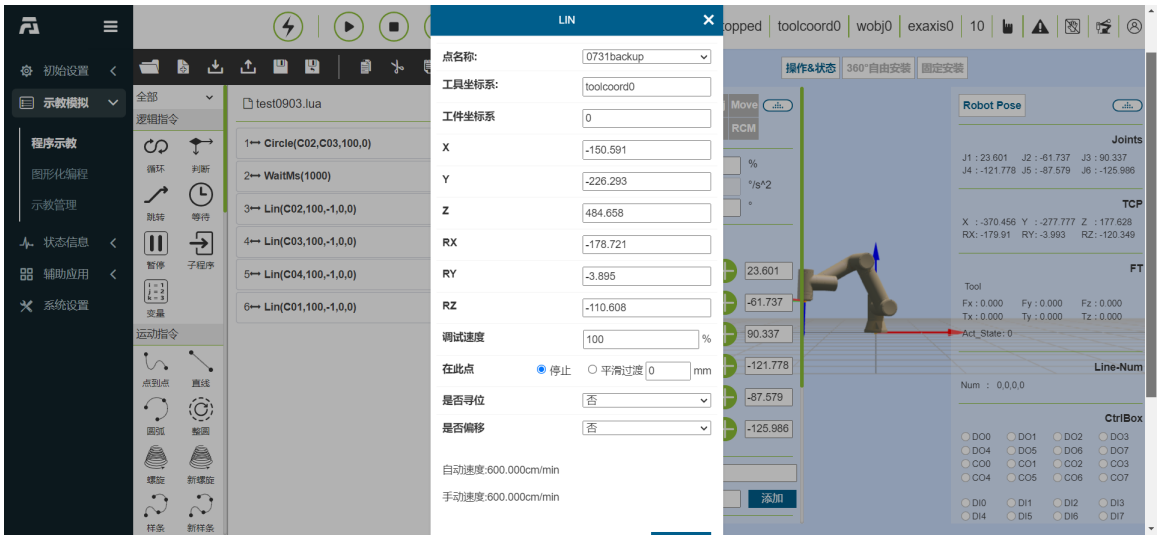
可以选择需要到达的点，平滑过渡时间设置可以实现该点到下一点的运动是连续的，是否偏移设置，可以选择基于基坐标系偏移和基于工具坐标偏移，并弹出 x,y,z,rx,ry,rz 偏移量设置，PTP 具体路径为运动控制器自动规划的最优路径，点击“添加”、“应用”后可保存该条指令。



图表 5.1-1 PTP 命令界面

Lin 命令：点击“直线”图标进入 Lin 命令编辑界面

该指令功能与“PTP”指令相似，但该指令所到达点的路径为直线



图表 5.1-2 Lin 命令界面

1.2.5.2 对程序文件进行操作

使用程序树底部的工具栏修改程序树。

备注:

名称：**打开**

作用：打开用户程序文件

备注:

名称：**新建**

作用：选择模板新建程序文件

备注:

名称：**导入**

作用：导入文件到用户程序文件夹中

备注:

名称：**导出**

作用：导出用户程序文件到本地点。

备注:

名称：**保存**

作用：保存文件编辑内容

备注:

名称：**另存为**

作用：给文件重命名存放到用户程序或模板程序文件夹中。

备注:

名称：**复制**

作用：复制一个节点，并允许将其用于其他操作（例如：将其粘贴到程序树的其他位置）。

备注:

名称：**粘贴**

作用：允许您粘贴之前剪切或复制的节点。

备注:

名称：**剪切**

作用：剪切一个节点，并允许将其用于其他操作（例如：将其粘贴到程序树的其他位置）。

备注:

名称：**删除**

作用：从程序树中删除一个节点。

备注:

名称：**上移**

作用：向上移动该节点。

备注:

名称：**下移**

作用：向下移动该节点。

备注:

名称：**切换编辑模式**

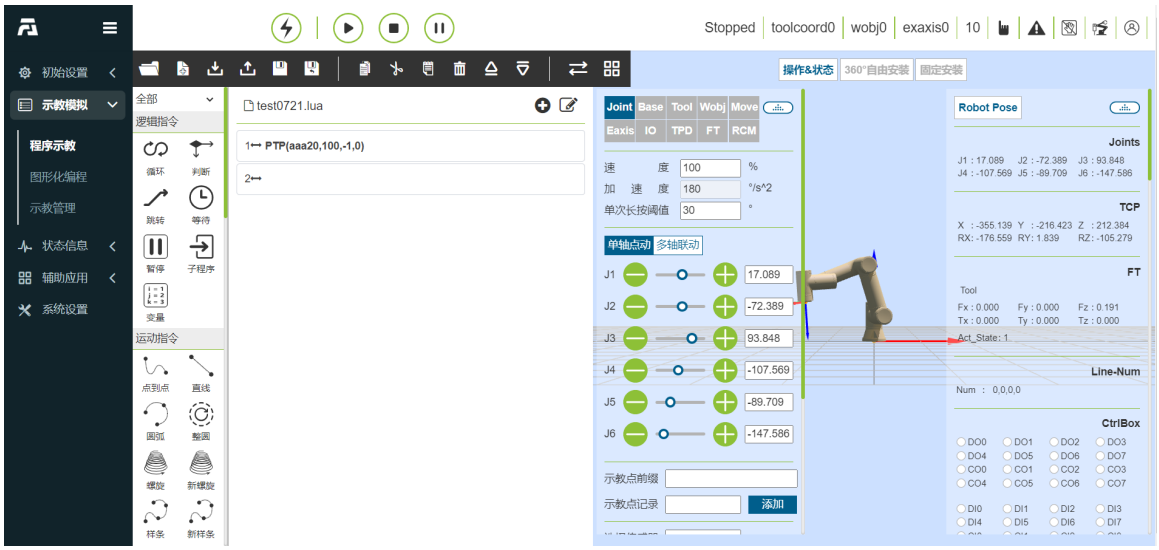
作用：程序树模式和 lua 编辑模式互相切换。

1.2.5.3 编写运行一个程序

左侧主要是程序命令的添加，点击各关键字上方图标进入详细界面，程序命令添加到文件中的操作主要分为两种，一种方式打开相关指令点击应用按键即可将该指令添加到程序中，另一种方式为先点击“添加”按键，此时命令并未保存到程序文件中，需要再点击“应用”方可将命令保存到文件中。第二种方式多出现在同类型指令多条下发的情况，我们对该类型命令增加添加按键和显示已添加指令内容功能，点击添加按键可添加一条指令，已添加指令显示所有已添加的指令，点击“应用”即可将添加的指令保存到右侧已打开的文件中。

点击开始按钮，运行程序；点击停止按钮，停止程序运行；点击暂停/恢复按钮，暂停/恢复程序；程序运行时，当前执行的程序节点灰色高亮显示。

在手动模式下，点击节点右侧第一个图标可以使机器人单独执行该指令，第二个图标为编辑该节点内容。



图表 5.3-1 程序树界面

1.3 使用手册

1.3.1 前言

首先，非常感谢您选购我们公司的 FR 系列协作机器人产品。我们的产品是经过多次精心设计和测试，以确保能够满足您各方面的需求。

请您仔细阅读本产品用户手册，以确保您能够正确地使用我们的产品并获得最佳的使用体验。如果您在使用过程中遇到任何问题，请参阅使用手册中的故障排除部分，或与我们的售后人员联系。我们非常感谢您的支持和信任，期待为您提供更好的服务和产品。

1.3.1.1 箱子里面装的什么

当您订购一个 FR 系列协作机器人后，您会收到一个箱子。里面包含：

- **协作机器人一台**
- **按钮盒一个**
- **控制箱一个（含控制箱线缆一根）**

1.3.1.2 重要安全说明

机器人是一种涉及人身安全的设备，因此每次安装机器人后都必须执行安全评估。您必须遵守第 1 章中的所有安全说明。

1.3.1.3 如何使用本手册

本手册包含机器人安装编程的指导信息。手册包括：

- **硬件安装部分**：机器人的机械安装和电气安装
- **示教器软件解析部分**：机器人示教及编程

本手册面向的机器人集成商，集成商应接受过基本的机械电气培训，并熟悉初级编程概念。

1.3.1.4 遵循的相关标准

标准	定义
2006/42/EC:2006	Machinery Directive: Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC(recast)
2004/108/EC:2004	EMC Directive: Directive 2004/108/EC of the European Parliament and of the Council of 15 December 2004 on the approximation of the laws of the Member States relating to electromagnetic compatibility and repealing Directive 89/336/EEC
EN ISO 13850:2008	Safety of machinery: Emergency stop - Principles for design
EN ISO 13849-1:2008	Safety of machinery: Safety-related parts of control systems - Part 1: General principles of design
EN ISO 13849-2:2012	Safety of machinery: Safety-related parts of control systems - Part 2: Validation
EN ISO 12100:2010	Safety of machinery: General principles of design, risk assessment and risk reduction
EN ISO 10218-1:2011	Industrial robots: Safety Note: Content equivalent to ANSI/RIA R.15.06-2012, Part 1
ISO/TS 15066: 2016	Safety requirements for collaborative industrial robot Robots and robotic devices —Collaborative robots

1.3.2 机器人简介

1.3.2.1 基本参数

表格 1.1-1 机器人基本参数

重要： FR 系列机器人在做姿态或坐标系变换时齐次变换矩阵计算的角度旋转顺序为浮动坐标系的 “ZYX”。

1.3.2.2 运动范围

机械臂安装空间：

机器人本体安装需要 3m×3m×2m（长 × 宽 × 高）的空间，以满足机器人最大臂展下的运动；若用户自行增加末端负载，请确保安装空间留有最少 500mm 间隙。

备注： 高度空间受安装底座高度的影响，此处 2m 是指高出安装基准面的距离

控制柜安装空间：

1. 控制箱应放在易于操作，防止水淹触电，距离地面 0.6m-1.5m。

2. 柜体必须远离热源。

3. 控制箱重载线一侧应满足 150mm 以内无遮挡，其余侧满足 100mm 以内无遮挡，便于散热和取放。

图表 1.1-1 FR3 型号协作机器人运动范围

图表 1.1-2 FR5 型号协作机器人运动范围

图表 1.1-3 FR10 型号协作机器人运动范围

图表 1.1-4 FR16 型号协作机器人运动范围

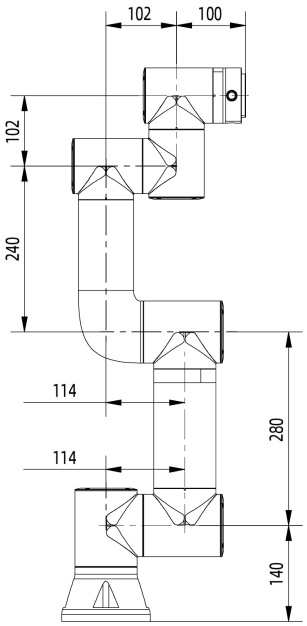
图表 1.1-5 FR20 型号协作机器人运动范围

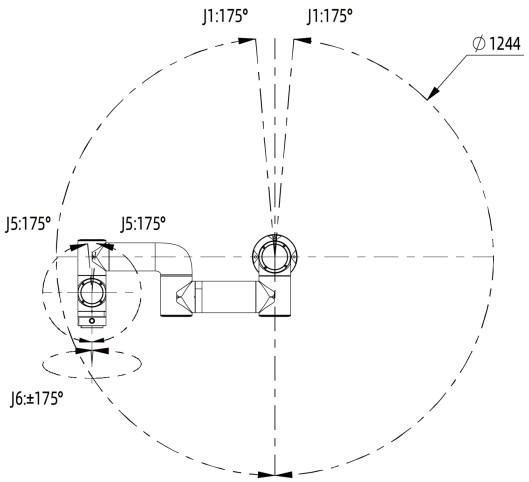
1.3.2.3 机器人坐标系

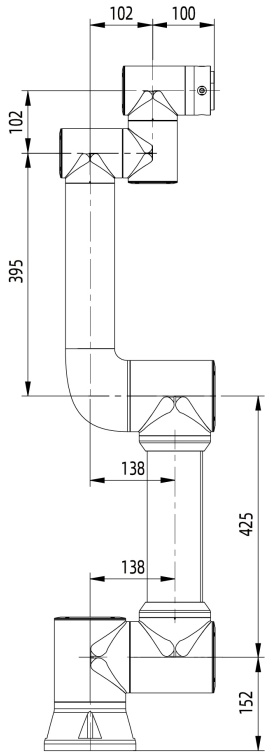
图表 1.1-6 机器人 DH 参数坐标系

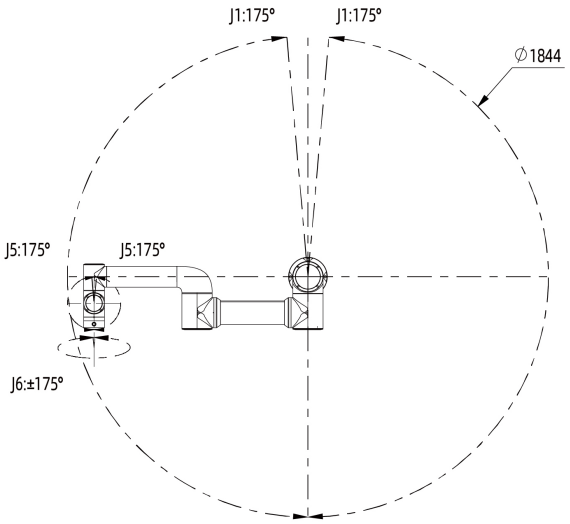
图表 1.1-7 机器人末端法兰坐标系

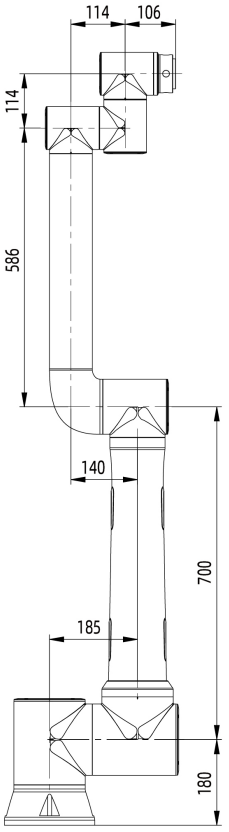
参数 \ 型号	FR3	FR5	FR10	FR16	FR20
负载	3kg	5kg	10kg	16kg	20kg
最大工作范围	622mm	922mm	1400mm	1076mm	1868mm
自由度	6个旋转自由度				
重复定位精度	±0.02mm	±0.02mm	±0.05mm	±0.03mm	±0.1mm
关节活动范围（软限位）	1轴：+175°, -175°; 2轴：+85° , -265°; 3轴：+150°, -150°; 4轴：+85° , -265°; 5轴：+175°, -175°; 6轴：+175°, -175°;	1轴：+175°, -175°; 2轴：+85° , -265°; 3轴：+160°, -160°; 4轴：+85° , -265°; 5轴：+175°, -175°; 6轴：+175°, -175°;			
关节活动范围（硬限位）	1轴：+179°, -179°; 2轴：+89° , -269°; 3轴：+152°, -152°; 4轴：+89° , -269°; 5轴：+179°, -179°; 6轴：+179°, -179°;	1轴：+179°, -179°; 2轴：+89° , -269°; 3轴：+162°, -162°; 4轴：+89° , -269°; 5轴：+179°, -179°; 6轴：+179°, -179°;			
关节最快速度	180°/s	180°/s	1,2关节120°/s 其他关节180°/s		1,2,3关节120°/s 其他关节180°/s
典型TCP速度	1m/s	1m/s	1.5m/s	1m/s	2m/s
防护等级	IP54(可选IP66)				
噪音	<65dB				
安装方向	任何方向				
输入输出	电源(24V/1.5A)、数字IO、模拟IO、485通信、以太网				
通讯	I/O、TCP/IP、Modbus_TCP/RTU、Profinet				
工作温度	0-45℃				
工作湿度	90%RH(non-condensing)				
整机质量	约15kg	约20.6kg	约40kg	约40kg	约65kg
设备存放	-25~60℃(无结霜)				
平均故障修复时间	2h				

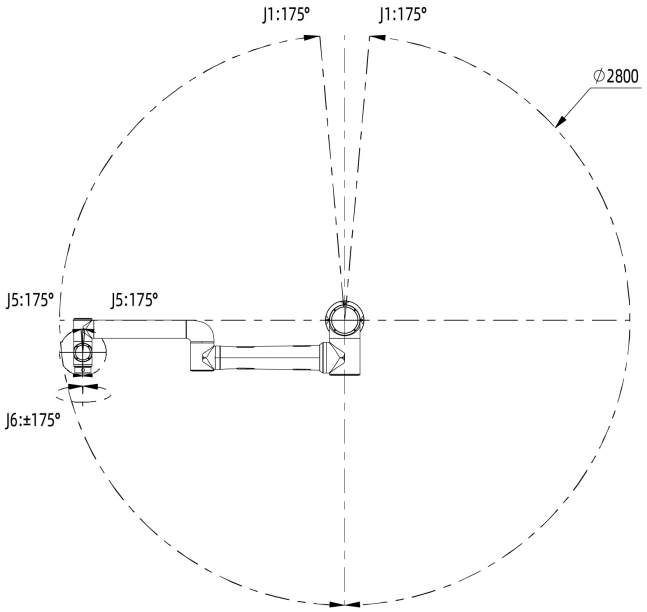

102
100
102
240
114
114
280
140

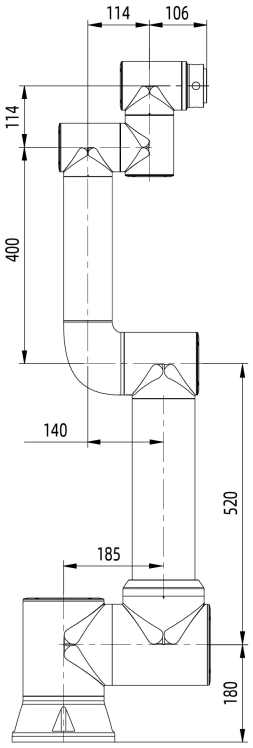

J1:175°
J1:175°
∅1244
J5:175°
J5:175°
J6:±175°

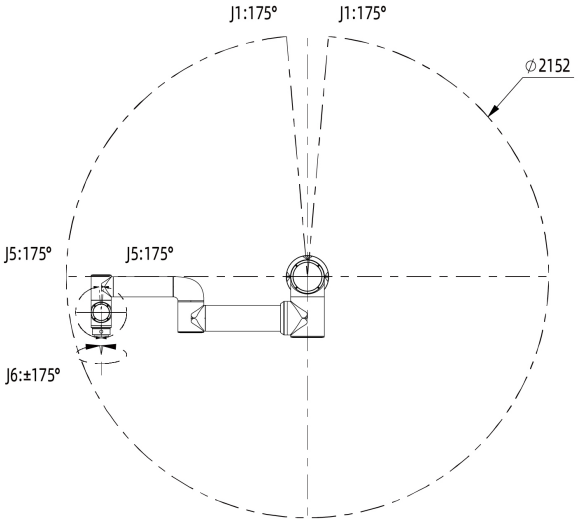

102
100
102
395
138
425
138
152

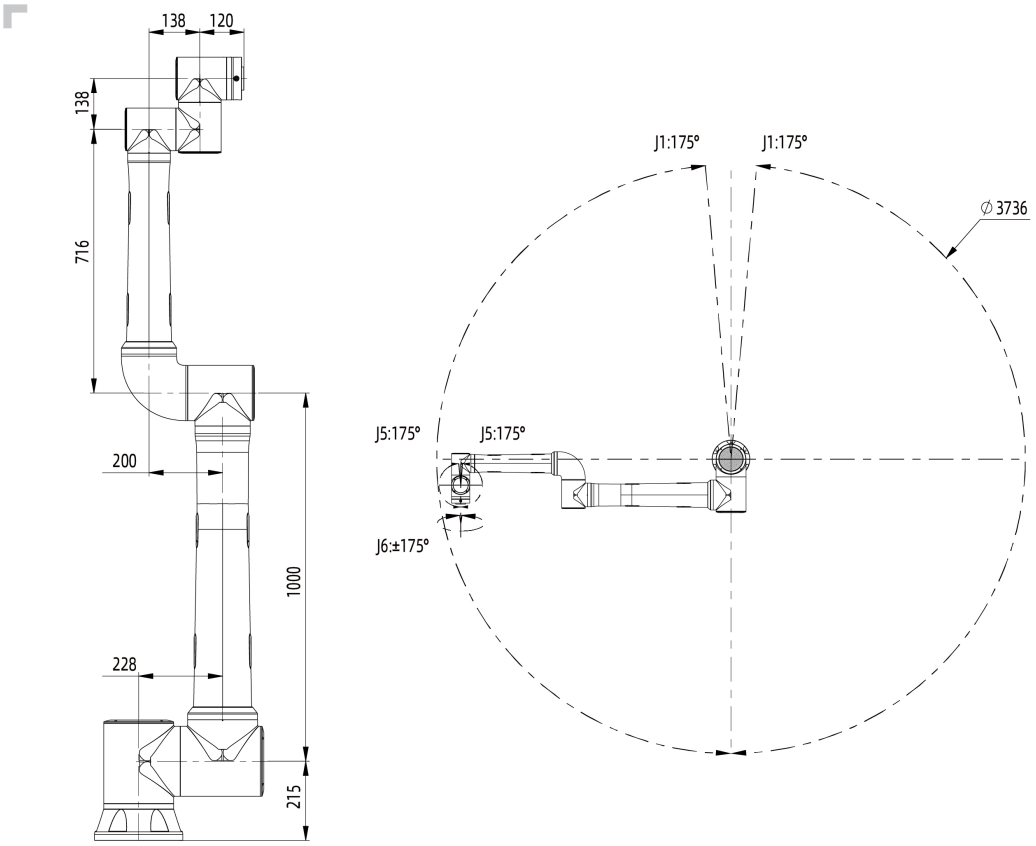

J1:175°
J1:175°
∅1844
J5:175°
J5:175°
J6:±175°

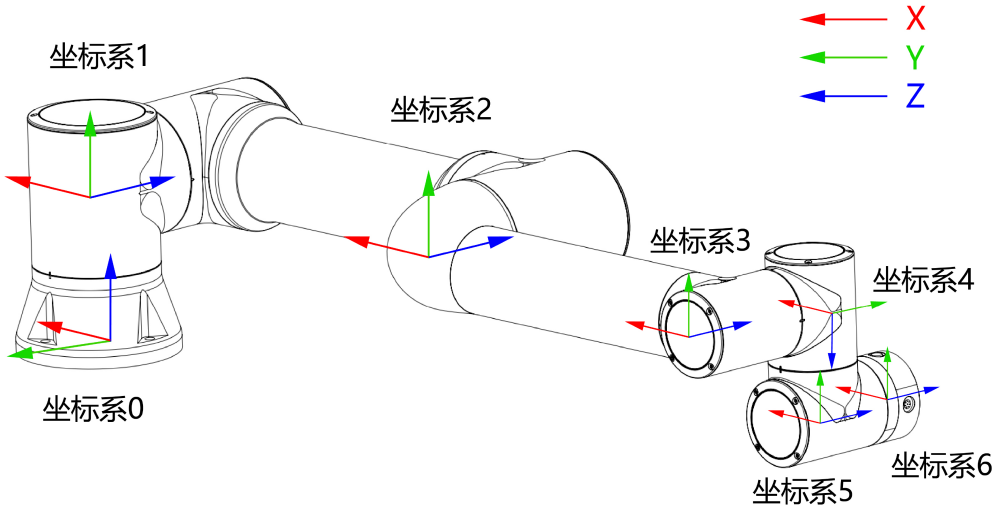

114
106
114
586
140
700
185
180

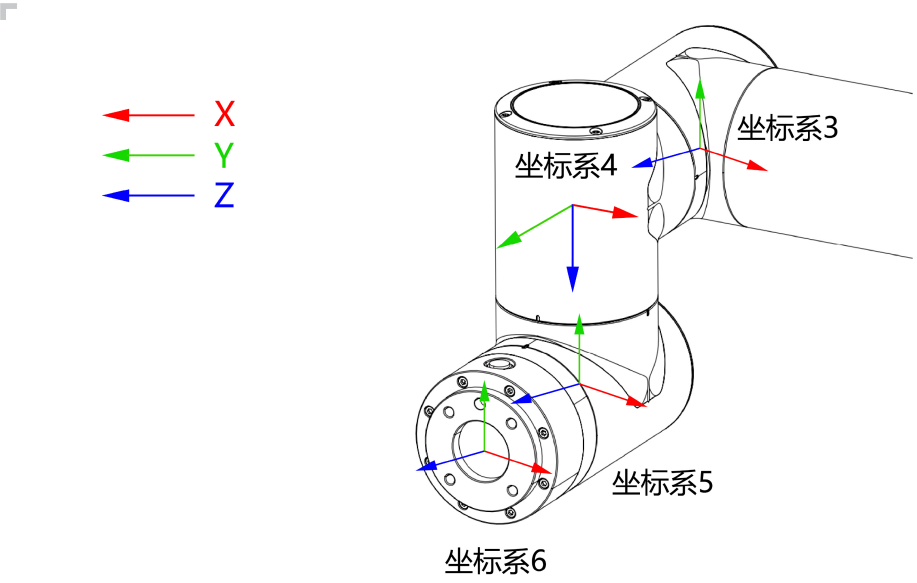

J1:175°
J1:175°
∅2800
J5:175°
J5:175°
J6:±175°


114
106
114
400
140
520
185
180


J1:175°
J1:175°
∅2152
J5:175°
J5:175°
J6:±175°


138
120
138
716
200
1000
228
215
J1:175°
J1:175°
∅3736
J5:175°
J5:175°
J6:±175°


X
Y
Z
坐标系1
坐标系2
坐标系3
坐标系4
坐标系0
坐标系5
坐标系6



1.3.2.4 机器人 DH 参数

DH 参数用于计算 FR 系列协作机器人的运动学和动力学。

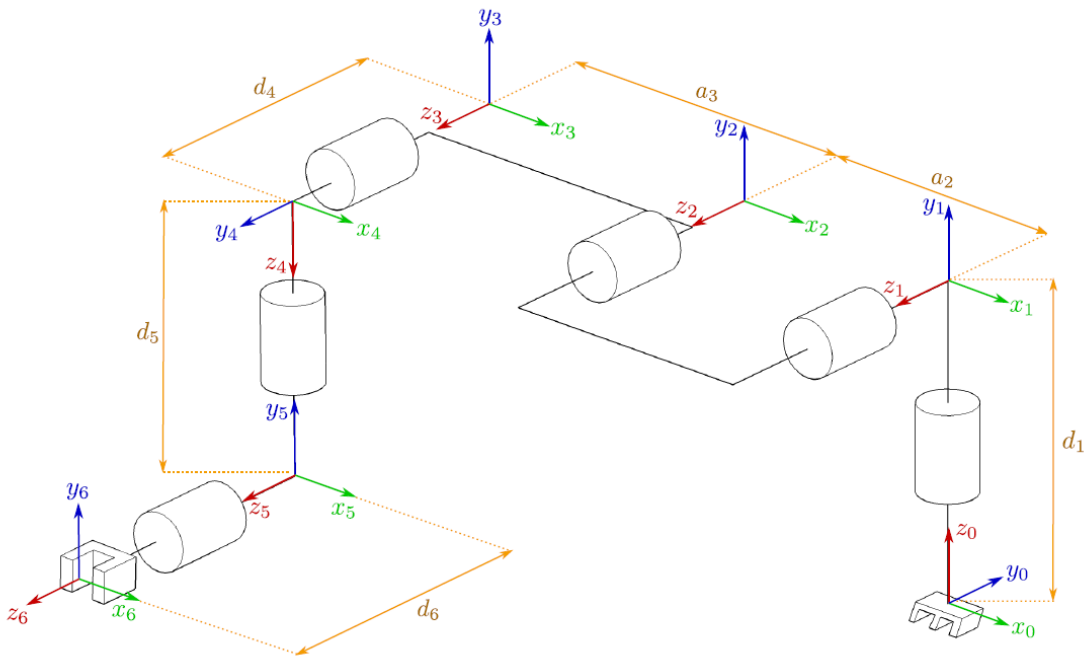
图表 1.1-8 FR 系列协作机器人 DH 参数

FR 系列协作机器人 DH 参数展示如下：

Table 1.1-2 FR3 协作机器人 DH 参数表

运动学	theta[rad	a[m]	d[m]	alpha[rad]	动力学	Mass[kg]	Center of Mass[m]
Joint1	0	0	140	π/2	Link1	1.98	[-0.05, -15.92, 2.26]
Joint2	0	-280	0	0	Link2	3.4445	[139.49, 0, 99.54]
Joint3	0	-240	0	0	Link3	1.437	[58.99, 0.08, 12.99]
Joint4	0	0	102	π/2	Link4	0.871	[0.05, -2.33, 14.67]
Joint5	0	0	102	-π/2	Link5	0.805	[-0.05, 2.33, 14.67]
Joint6	0	0	100	0	Link6	0.261	[-0.05, -1.11, -20.05]

Table 1.1-3 FR5 协作机器人 DH 参数表



运动学	theta[rad	a[m]	d[m]	al-pha[rad]	动力学	Mass[kg]	Center of Mass[m]
Joint1	0	0	152	π/2	Link1	4.64	[-0.19, -18.28, 2.26]
Joint2	0	-425	0	0	Link2	10.08	[212.47, 0, 121.2]
Joint3	0	-395	0	0	Link3	2.71	[122.62, 0.17, 12.59]
Joint4	0	0	102	π/2	Link4	1.56	[0.05, -2.33, 14.68]
Joint5	0	0	102	-π/2	Link5	1.56	[-0.05, 2.33, 14.68]
Joint6	0	0	100	0	Link6	0.36	[0.93, 0.81, -20.05]

Table 1.1-4 FR10 协作机器人 DH 参数表

运动学	theta[rad	a[m]	d[m]	al-pha[rad]	动力学	Mass[kg]	Center of Mass[m]
Joint1	0	0	180	π/2	Link1	11.97	[-0.10, -26.12, 4.04]
Joint2	0	-700	0	0	Link2	19.59	[480.27, 0.01, 164.68]
Joint3	0	-586	0	0	Link3	3.7	[211.22, 0.11, 54.21]
Joint4	0	0	159	π/2	Link4	1.69	[0.12, -3, 12.18]
Joint5	0	0	114	-π/2	Link5	1.69	[-0.12, 3, 12.18]
Joint6	0	0	106	0	Link6	0.35	[1.24, 0.85, -20.34]

Table 1.1-6 FR16 协作机器人 DH 参数表

运动学	theta[rad	a[m]	d[m]	alpha[rad]	动力学	Mass[kg]	Center of Mass[m]
Joint1	0	0	180	π/2	Link1	11.97	[-0.10, -26.12, 4.04]
Joint2	0	-520	0	0	Link2	18.18	[364.4, 0.01, 163.09]
Joint3	0	-400	0	0	Link3	3.22	[135.03, 0.12, 55.58]
Joint4	0	0	159	π/2	Link4	1.69	[0.12, -3, 12.18]
Joint5	0	0	114	-π/2	Link5	1.69	[-0.12, 3, 12.18]
Joint6	0	0	106	0	Link6	0.35	[1.24, 0.85, -20.34]

Table 1.1-7 FR20 协作机器人 DH 参数表

运动学	theta[rad	a[m]	d[m]	alpha[rad]	动力学	Mass[kg]	Center of Mass[m]
Joint1	0	0	215	π/2	Link1	20.79	[-0.19, -36.57, 5.68]
Joint2	0	-1000	0	0	Link2	42.84	[605.25, 0.06, 202.94]
Joint3	0	-716	0	0	Link3	9.88	[262.84, 0.22, 43.08]
Joint4	0	0	166	π/2	Link4	4.64	[0.23, -2.28, 18.42]
Joint5	0	0	138	-π/2	Link5	4.64	[-0.23, 2.28, 18.42]
Joint6	0	0	120	0	Link6	0.6	[-2.11, -1.96, -20.38]

1.3.2.5 DH 参数表

法奥协作机器人 - DH 参数表

1.3.3 硬件安装

1.3.3.1 安全须知

1.3.3.1.1 简介

本说明书会使用到以下警示，这些警示的作用是确保人身及设备的安全，当您在阅读本说明书时，必须遵守并执行本手册其他章节中的所有组装说明和指南，这一点非常重要。应特别注意与警告标志相关的文本。

重要:

- 如果机器人（机器人本体、控制箱、示教器或按钮盒）因人为原因被损坏、更改或修改，法奥意威拒绝承担所有责任；
- 法奥意威对由于客户编写的程序出错而对机器人或任何其他设备造成的任何损坏概不负责。

1.3.3.1.2 人员安全

在运行机器人系统时，首先必须要确保作业人员的安全，下面列出一般性的注意事项，请妥善采取确保作业人员安全的相应措施。

1. 使用机器人系统的各作业人员，应通过法奥意威（苏州）机器人系统有限公司主办的培训课程接受培训。用户需确保其充分掌握安全、规范的操作流程，具备机器人操作资格。培训详情请向我公司查询，邮箱为 jiling@frtech.fr。
2. 使用机器人系统的各作业人员请不要穿宽松的衣服，不要佩戴珠宝。操作机器人时请确保长头发束在脑后。
3. 在设备运转之中，即使机器人看上去已经停止，也有可能是因为机器人在等待启动信号而处在即将动作的状态。即使在这样的状态下，也应该将机器人视为正在动作中。
4. 应在地板上画上线条来标清机器人的动作范围，使操作者了解机器人包含握持工具（机械手、工具等）的动作范围。
5. 确保在机器人操作区域附近建立安全措施（例如，护栏、绳索、或防护屏幕），保护操作者及周边人群。应根据需要设置锁具，使得负责操作的作业人员以外者不能接触机器人电源。
6. 在使用操作面板和示教器时，由于戴上手套可能会出现操作上的失误，务必在摘下手套后进行作业。
7. 在人被机器人夹住或围在里面等紧急和异常情况下，通过用力（至少 700 N）推动或拉动机器人手臂，迫使关节移动。无电力驱动情况下手动移动机器人手臂仅限于紧急情况，并且可能会损坏关节。

1.3.3.1.3 安全设置

在“辅助应用”中的“安全性设置”的菜单栏下，点击“安全停止设置”进入安全停止设置功能界面。

启用安全停止模式，当机器人报错或警告停止后，机器人会自动去使能，起到安全防护的作用。

图表 1.1-1 安全停止设置

- **降速模式**：该模式被激活后，机械臂在关节空间中的运动速度将受到限制，相应文本框中的数值即为各关节运动速度的极限值，其中 1，2，3 关节的设定范围为 15~150°/s，4，5，6 关节的设定范围为 15~180°/s；机械臂在笛卡尔空间的运动速度极限即为 TCP 速度限制值，设定范围为 0~80mm/s。

1.3.3.1.4 危险识别

风险评估应考虑正常使用期间操作人员与机器人之间所有潜在的接触以及可预见的误操作。操作人员的颈部、脸部和头部不应暴露，以免发生碰触。在不使用外围安全防护装置的情况下使用机器人需要首先进行风险评估，以判断相关危险是否会构成不可接受的风险，例如

- 使用尖锐的末端执行器或工具连接器可能存在危险；
- 处理毒性或其他有害物质可能存在危险；
- 操作人员手指有被机器人底座或关节夹住的危险；
- 被机器人碰撞发生的危险；
- 机器人或连接到末端的工具固定不到位存在的危险；
- 机器人有效负载与坚固表面之间的冲击造成的危险。

集成商必须通过风险评估来衡量此类危险及其相关的风险等级，并且确定和实施相应的措施，以将风险降低至可接受的水平。请注意，特定机器人设备可能还存在其他重大危险。

通过将 FR 机器人所应用的固有安全设计措施与集成商和最终用户所实施的安全规范或风险评估相结合，将与 FR 协作性操作相关的风险尽可能降低至合理可行的水平。通过此文档可将机器人在安装前存在的任何剩余风险传达给集成商和最终用户。如果集成商的风险评估测定其特定应用中存在可能对用户构成不可接受风险的危险，集成商必须采取适当的风险降低措施，以消除或最大限度降低这些危险，直至将风险降低至可接受的水平为止。在采取适当的风险降低措施（如有需要）之前使用是不安全的。

如果对机器人进行非协同性安装（例如，当使用危险工具时），风险评估可能推断集成商需要在其编程时连接额外的安全设备（例如，安全启动设备）确保人员及设备安全。

1.3.3.1.5 铭牌信息

图表 1.1-2 FR3 型号协作机器人

图表 1.1-3 FR5 型号协作机器人

图表 1.1-4 FR10 型号协作机器人

图表 1.1-5 FR16 型号协作机器人

图表 1.1-6 FR20 型号协作机器人

FAIR INNOVATION

产品名称:	协作机器人
产品型号:	FR3
产品颜色:	白色
本体重量:	14KG
包装重量:	25KG
包装尺寸:	长 宽 高 100cm*50cm*50cm
包装内容:	协作机器人、控制箱、按钮盒

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
山东省淄博市高新区尊贤路5888号
相关信息请查询销售网站：www.frtech.fr
服务电话：0512-68562005

本产品已经过安全认证

FAIR INNOVATION

产品名称:	**协作机器人**
产品型号:	FR3
负载能力:	3kg
工作半径:	622mm
本体质量:	14kg
额定输入电压:	220VAC/单相/50Hz
额定输出电压:	48VDC
输出短路额定值	48V22A
序列号:	XXXXXXXXXXXXXXXXXX
生产日期:	2021.05.06

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
山东省淄博市高新区尊贤路5888号
服务电话：0512-68562005
网　　址：www.frtech.fr

FAIR INNOVATION

产品名称:	控制箱
产品型号:	FR5
额定电源:	220VAC/6A/单相/50Hz
防护等级:	IP54（IO引出版为IP20）
使用温度:	0-45℃
输入:	数字量：16 模拟量：2
输出:	数字量：16 模拟量：2
序 列 号:	
生产日期:	

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
山东省淄博市高新区尊贤路5888号
服务电话：0512-68562005
网　　址：www.frtech.fr

FAIR INNOVATION

产品名称:	**协作机器人**
产品型号:	FR5
负载能力:	5kg
工作半径:	922mm
本体质量:	21kg
额定输入电源:	220VAC/6A/单相/50Hz
额定输出电压:	48VDC
输出短路额定值	48V22A
序列号:	XXXXXXXXXXXXXXXXXX
生产日期:	2021.05.06

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
山东省淄博市高新区尊贤路5888号
服务电话：0512-68562005
网　　址：www.frtech.fr

FAIR INNOVATION

产品名称: 协作机器人
产品型号: FR10
产品颜色: 白色

本体重量: 41KG
包装重量: 52KG
包装尺寸: 长 宽 高 100cm*50cm*50cm
包装内容: 协作机器人、控制箱、按钮盒

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
　　山东省淄博市高新区尊贤路5888号
相关信息请查询销售网站：www.frtech.fr
服务电话：0512-68562005

本产品已经过安全认证

FAIR INNOVATION

产品名称: **协作机器人**
产品型号: FR10
负载能力: 10kg
工作半径: 1400mm
本体质量: 41kg
额定输入电压: 220VAC/单相/50Hz
额定输出电压: 48VDC
输出短路额定值 48V44A

序列号:
生产日期:

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
　　山东省淄博市高新区尊贤路5888号
服务电话：0512-68562005
网　　址：www.frtech.fr

FAIR INNOVATION

产品名称：控制器
产品型号：FRC100-AC
产品颜色：黑色

包装尺寸：长 宽 高 350cm*350cm*150cm
包装内容：控制器

制 造 商：法奥意威（苏州）机器人系统有限公司
地　　址：江苏省苏州市高新区竹园路209号
　　山东省淄博市高新区尊贤路5888号
电　　话：0512-68562005

www.frtech.fr

FAIR INNOVATION

产品名称: **协作机器人**
产品型号: FR16
负载能力: 16kg
工作半径: 1076mm
本体质量: 38kg
额定输入电压: 220VAC/单相/50Hz
额定输出电压: 48VDC
输出短路额定值 48V41A

序列号:
生产日期:

制造商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
　　山东省淄博市高新区尊贤路5888号
服务电话：0512-68562005
网　　址：www.frtech.fr

1.3.3.1.6 有效性和责任

本手册中的信息不包含设计、安装和操作一个完整的机器人应用，也不包含所有可能对这一完整的系统的安全造成影响的周边设备。该完整系统的设计和安装需符合该机器人安装所在国的标准和规范中确立的安全要求。

法奥意威的集成商有责任确保遵循相关国家的法律法规，确保完整的机器人应用中不存在任何重大危险。这包括但不限于以下内容：

- 对完整的机器人系统做一个风险评估
- 将风险评估定义的其他机械和附加安全设备连接在一起
- 在软件中建立适当的安全设置
- 确保用户不会对任何安全措施加以修改
- 确认整个机器人系统的设计和安装准确无误
- 明确使用说明
- 在机器人上标明集成商的相关标志和联系信息
- 收集技术文件中的所有文档，包括本手册

1.3.3.1.7 责任有限

本手册所包含的任何安全信息都不得视为通用的机器人安全保证，即使遵守所有安全说明，依然有可能引起人员伤害或设备损坏。

1.3.3.1.8 该手册中的警告标志

下面的标志定义了本手册中所包含的危险等级规定说明。产品上也使用了同样的警告标志。

重要:

危险：这指的是即将引发危险的用电情况，如果不避免，可导致人员死亡或严重伤害。

重要:

触电危险：这指的是即将引发危险的触电情况，如果不避免，可导致人员触电死亡或严重伤害。

重要:

烫伤危险：这指的是可能引发危险的热表面，如果不避免接触了，可造成人员伤害。

1.3.3.1.9 使用前评估

首次使用机器人或进行任何修改之后，机器人默认速度低于 250mm/s，请勿登录管理员修改速度进入高速模式，之后必须进行以下测试。确认所有安全输入和输出是正确的，并且连接正确。测试所有连接的安全输入和输出（包括多台机器或机器人共有的设备）是否功能正常。因此您必须：

- 测试紧急停止按钮和输入是否可以停止机器人并启动刹车。
- 测试防护输入是否可以停止机器人的运动。如果配置了防护重置，请在恢复运动之前检查是否需要激活。
- 测试操作模式是否可以切换操作模式，参见用户界面右上角的图标。
- 测试 3 档位使动装置是否必须按下才能在手动模式下启动动作，并且机器人处于减速控制下（机器人软件版本 V3.0 前不支持该功能）。
- 测试系统紧急停止输出是否能够将整个系统带到安全状态。

1.3.3.1.10 紧急停止

紧急停止按钮为 0 类停机，按下紧急停止按钮，立即停止机器人的一切运动。

下表显示触发 0 类停机的停止距离和停止时间。这些测量结果对应于机器人的以下配置：

- 延伸：100%（机器人手臂完全水平展开）
- 速度：100%（机器人一般速度设为 100%，以 180°/s 的关节速度移动）
- 有效负载：最大有效负载

关节 1，关节 6 测试机器人水平移动，旋转轴垂直于地面。关节 2，关节 3，关节 4，关节 5 测试机器人遵循垂直轨迹，旋转轴平行于地面，并在机器人向下移动时停止。

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	0.47	0.60	0.56	0.29	0.10	0.06
FR5	0.51	0.63	0.60	0.33	0.16	0.10
FR10	0.64	0.70	0.69	0.42	0.25	0.13
FR16	0.60	0.67	0.65	0.39	0.22	0.12
FR20	0.69	0.75	0.80	0.48	0.31	0.22

表格 1.1-1 0 类停止距离 (rad)

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	400	470	450	280	120	90
FR5	420	500	480	310	150	120
FR10	460	540	510	330	170	140
FR16	440	530	490	320	160	130
FR20	540	600	700	400	260	170

表格 1.1-2 0 类停止时间 (ms)

紧急停止后，关闭电源，旋转紧急停止按钮，打开电源即可重新启动机器人。

同时机器人安全停止和软限位停止的停止时间和停止距离，见下表。这些测量结果对应于机器人的以下配置:

- 延伸：100%（机器人手臂完全水平展开）
- 速度：100%（机器人一般速度设为 100%，以 180°/s 的关节速度移动）
- 有效负载：最大有效负载

关节 1，关节 6 测试机器人水平移动，旋转轴垂直于地面。关节 2，关节 3，关节 4，关节 5 测试机器人遵循垂直轨迹，旋转轴平行于地面，并在机器人向下移动时停止。

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	0.49	0.63	0.58	0.32	0.12	0.09
FR5	0.54	0.65	0.63	0.35	0.19	0.12
FR10	0.66	0.73	0.71	0.45	0.27	0.16
FR16	0.63	0.69	0.68	0.41	0.25	0.14
FR20	0.71	0.78	0.82	0.51	0.33	0.25

表格 1.1-3 安全停止距离 (rad)

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	410	490	410	300	130	110
FR5	450	520	510	330	180	140
FR10	480	570	530	360	190	170
FR16	470	550	520	340	190	150
FR20	560	630	720	430	280	200

表格 1.1-4 安全停止时间 (ms)

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	0.52	0.65	0.61	0.34	0.15	0.11
FR5	0.56	0.68	0.65	0.38	0.21	0.15
FR10	0.69	0.75	0.74	0.47	0.30	0.18
FR16	0.65	0.72	0.70	0.44	0.27	0.17
FR20	0.74	0.80	0.85	0.53	0.36	0.27

表格 1.1-5 软限位停止距离 (rad)

	关节 1	关节 2	关节 3	关节 4	关节 5	关节 6
FR3	430	500	430	310	150	120
FR5	460	540	520	350	190s	160
FR10	500	580	550	370	210	180
FR16	480	570	530	360	200	170
FR20	580	640	740	440	300	210

表格 1.1-6 软限位停止时间 (ms)

重要： 根据 IEC 60204-1 和 ISO 13850，紧急停止设备不是安全防护装置。它们是补充性防护措施，并不用于防止伤害。

1.3.3.1.11 无电力驱动的移动

如果发生必须移动机器人关节但无法为机器人供电或者其他紧急情况，请联系机器人经销商，必要时可以使用暴力手段强制移动机器人以解救被困人员。

1.3.3.2 设备运输

1.3.3.2.1 运输

机器人和控制箱已作为成套设备校准。请勿将它们分开，那样将需要重新校准。

只能将机器人放在原包装中运输。如果今后要搬运机器人的话，请将包装材料保存在干燥处。

将机器人从包装移动到安装空间时，同时托住机器人的两个臂体。扶住机器人直至机器人机座的所有安装螺栓全部紧固好。

1.3.3.2.2 搬运

协作机器人根据型号不同，总质量（含包装）范围在 15kg-80 kg，当人力对协作机器人进行搬运或转移时，需要多人协助抬起，不推荐单人搬运，在运输过程中务必平稳，避免设备倾翻或者滑落。

警告:

- 若采用专业设备进行搬运，请务必由具有相应操作资格的专业人员使用吊车或者叉车对协作机器人进行运输或者搬运，否则有可能会引起人员伤害或者其他事故。
- 若采用人工搬运，请注意搬运途中人身安全；
- 协作机器人包含精密零部件，在运输或者搬运过程中应该避免剧烈的振动或者晃动，否则有可能降低设备的性能。

1.3.3.2.3 存放

协作机器人应在-25~60℃，无凝霜环境下存放。

1.3.3.3 维护、报废处理

1.3.3.3.1 维护处置

请用户间隔 1 个月对急停和保护性停止进行检测。判断安全功能是否有效。

急停和保护性停止接线请参考接线章节。

1.3.3.3.2 废弃处置

FR 机器人需要根据适用的国家法律法规及国家标准处置，详情可联系厂商。

1.3.3.4 安装规范

1.3.3.4.1 机器人手臂安装

重要： 推荐机器人安装座满足以下几个要求，以保证机器人安装牢固、稳定：

（1）机器人安装座需要足够牢固且有足够的承载能力，应该至少能承载 5 倍的机器人重量，至少能承受 10 倍的 1 轴扭矩。

（2）机器人安装座应表面平整，以保证与机器人接触面紧密接触；

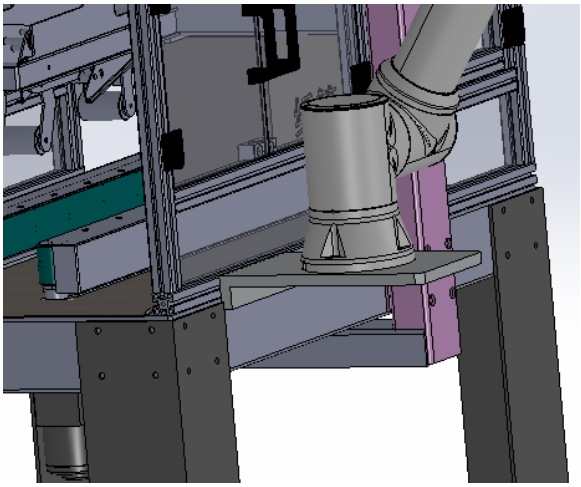
（3）机器人安装座应刚度足够强壮，固定牢固，不会和机器人发生共振；

（4）机器人和其他部件同时运动时，安装座与其他运动部件应隔离开，不要固定在一起避免运动过程中的振动干扰；

（5）如果机器人安装在移动平台或者外部轴上，移动平台或者外部轴的加速度应尽量低；

警告： 应该避免以下安装方式：

（I）避免将机器人固定在其它运动设备上



图表 1.5-1 避免安装在其它运动设备上

确保机器人手臂正确并安全地安装到位。安装不稳定会导致事故。

备注： 可以采购精确的基座作为附件来使用。图表 1.5-2、1.5-5、1.5-8、1.5-11 显示了销孔位置和螺丝安装位置。

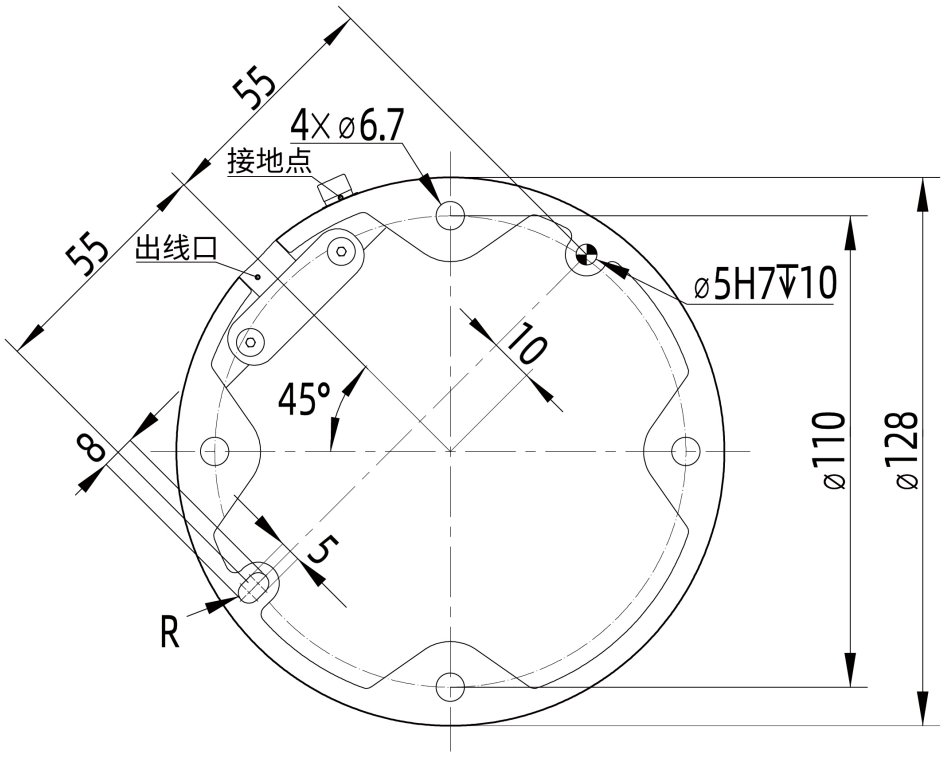
1.3.3.4.1.1 FR3 机器人手臂安装要求

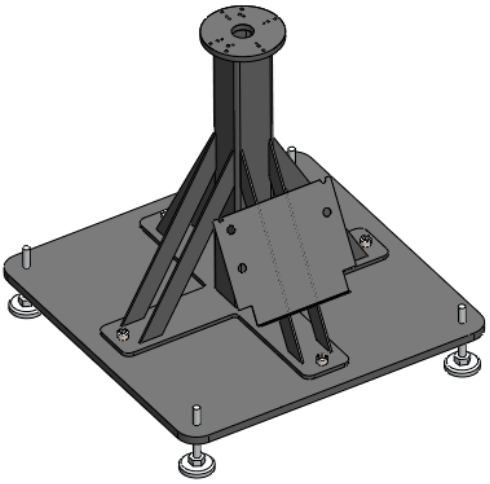
机器人安装在安装座上时，使用 4 颗强度不低于 8.8 级的 M6 螺栓将机器人固定在安装座上，螺栓须使用不少于 10Nm 扭矩拧紧；建议安装座上使用两个 φ5mm 销孔配合销钉进行机器人定位，以提高机器人安装精度，防止因为碰撞等使机器人发生移动。当机器人有较高运行精度要求时，请务必增加销钉对机器人进行定位。

图表 1.5-2 FR3 型号协作机器人安装尺寸

重要： 根据不同的应用场景推荐几款机器人安装底座如下：

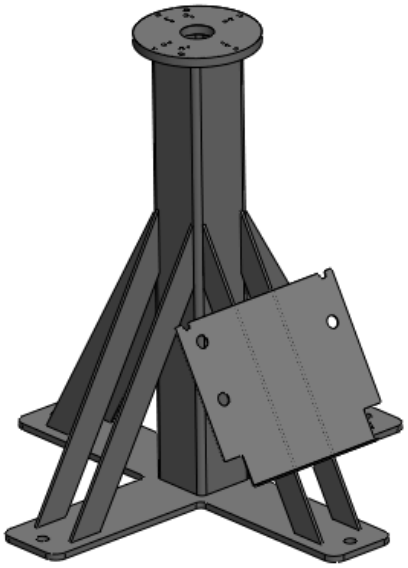
（I）针对运动速度不太快，运行速度不太大，精度要求一般，且不方便固定在地面上的场合，推荐机器人安装底座如下：


55
4X ø6.7
接地点
55
出线口
ø5H7↧10
10
45°
8
5
R
ø110
ø128



图表 1.5-3 FR3 型号协作机器人低要求安装底座

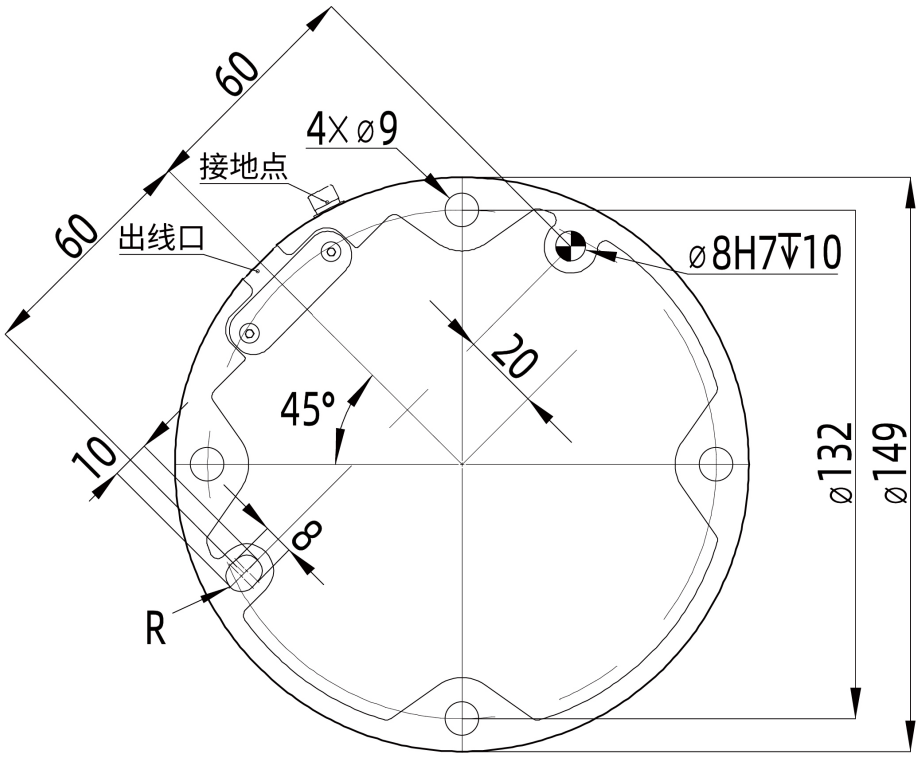
（II）针对运动速度较快，运行速度较大，精度要求较高的场合，推荐机器人安装底座如下，并将机器人固定在牢固的地面上：



图表 1.5-4 FR3 型号协作机器人高要求安装底座

1.3.3.4.1.2 FR5 机器人手臂安装要求

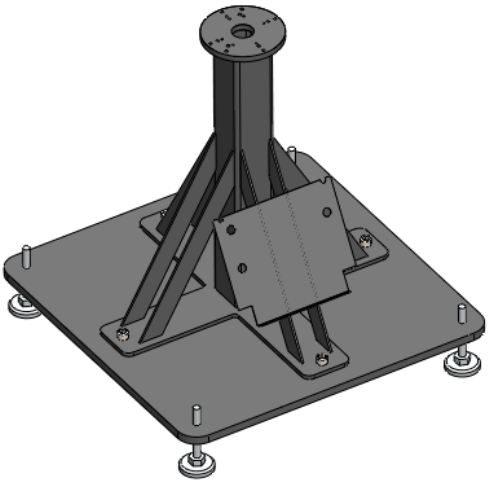
机器人安装在安装座上时，使用 4 颗强度不低于 8.8 级的 M8 螺栓将机器人固定在安装座上，螺栓须使用不少于 20Nm 扭矩拧紧；建议安装座上使用两个 φ8mm 销孔配合销钉进行机器人定位，以提高机器人安装精度，防止因为碰撞等使机器人发生移动。当机器人有较高运行精度要求时，请务必增加销钉对机器人进行定位。



图表 1.5-5 FR5 型号协作机器人安装尺寸

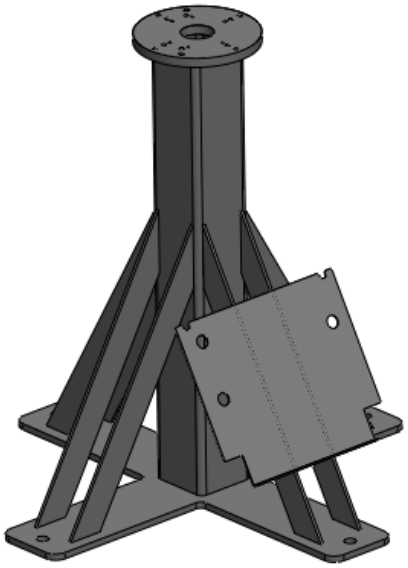
重要： 根据不同的应用场景推荐几款机器人安装底座如下：

（I）针对运动速度不太快，运行速度不太大，精度要求一般，且不方便固定在地面上的场合，推荐机器人安装底座如下：



图表 1.5-6 FR5 型号协作机器人低要求安装底座

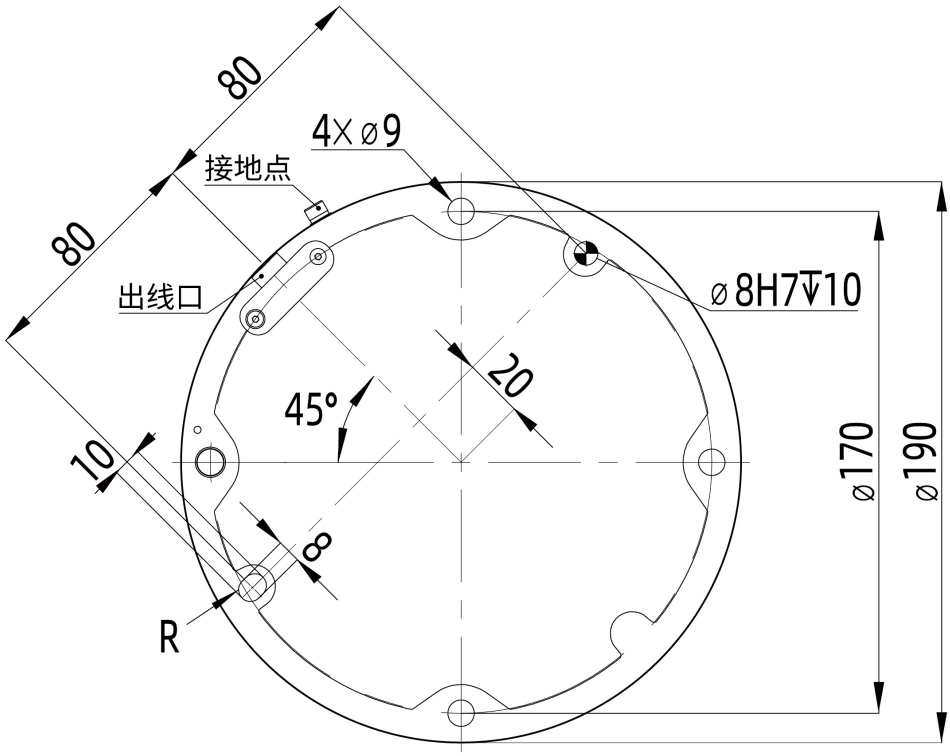
（II）针对运动速度较快，运行速度较大，精度要求较高的场合，推荐机器人安装底座如下，并将机器人固定在牢固的地面上：



图表 1.5-7 FR5 型号协作机器人高要求安装底座

1.3.3.4.1.3 FR10、FR16 机器人手臂安装要求

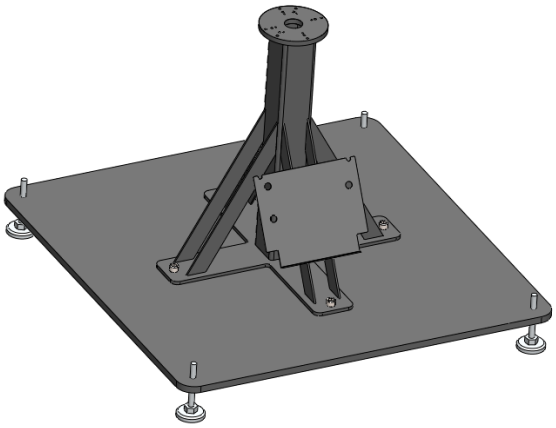
机器人安装在安装座上时，使用 4 颗强度不低于 8.8 级的 M8 螺栓将机器人固定在安装座上，螺栓须使用不少于 25Nm 扭矩拧紧；建议安装座上使用两个 φ8mm 销孔配合销钉进行机器人定位，以提高机器人安装精度，防止因为碰撞等使机器人发生移动。当机器人有较高运行精度要求时，请务必增加销钉对机器人进行定位。



图表 1.5-8 FR10、FR16 型号协作机器人安装尺寸

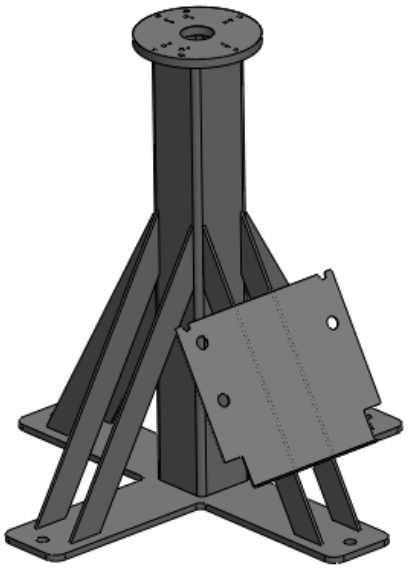
重要： 根据不同的应用场景推荐几款机器人安装底座如下：

（I）针对运动速度不太快，运行速度不太大，精度要求一般，且不方便固定在地面上的场合，推荐机器人安装底座如下：



图表 1.5-9 FR10、FR16 型号协作机器人低要求安装底座

（II）针对运动速度较快，运行速度较大，精度要求较高的场合，推荐机器人安装底座如下，并将机器人固定在牢固的地面上：

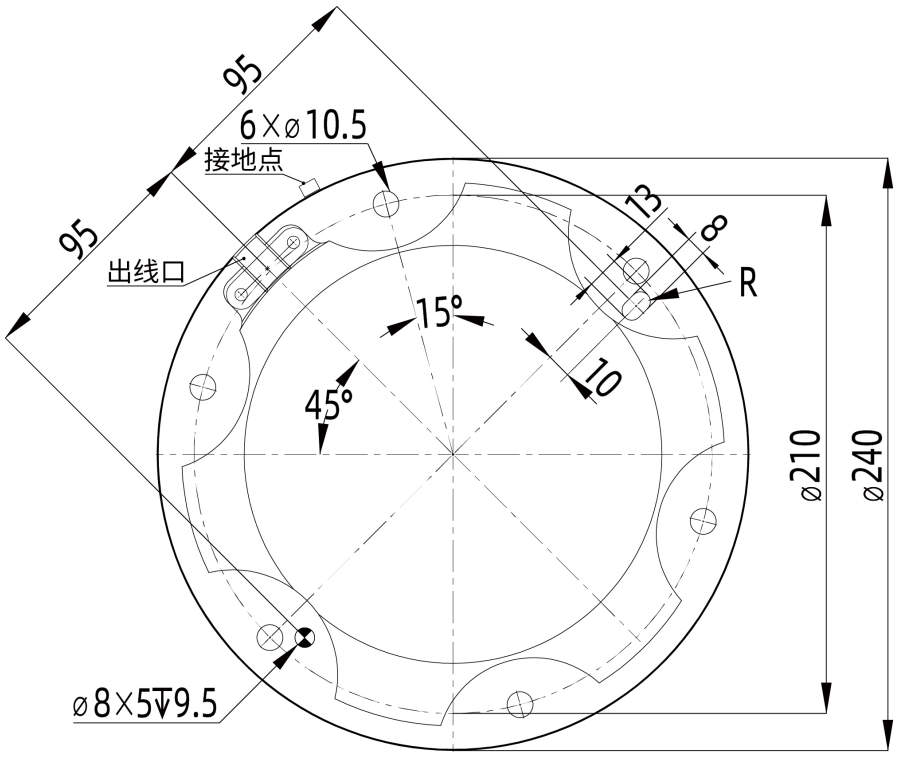


图表 1.5-10 FR10、FR16 型号协作机器人高要求安装底座

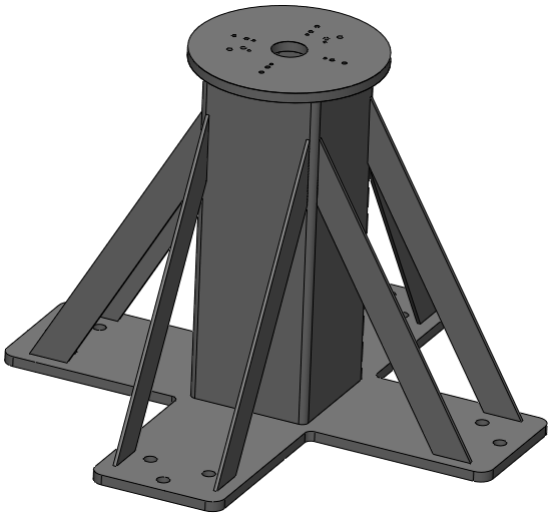
1.3.3.4.1.4 FR20 机器人手臂安装要求

机器人安装在安装座上时，使用 6 颗强度不低于 8.8 级的 M10 螺栓将机器人固定在安装座上，螺栓须使用不少于 45Nm 扭矩拧紧；建议安装座上使用两个 φ8mm 销孔配合销钉进行机器人定位，以提高机器人安装精度，防止因为碰撞等使机器人发生移动。当机器人有较高运行精度要求时，请务必增加销钉对机器人进行定位。

图表 1.5-11 FR20 型号协作机器人安装尺寸


95
95
6×ø10.5
接地点
出线口
13
8
R
15°
45°
10
ø210
ø240
ø8×5↧9.5

重要： 因为 FR20 机器人自重较大及运行惯量较大，建议直接固定在地面上使用。推荐底座如下：



图表 1.5-12 FR20 型号协作机器人安装底座

1.3.3.4.2 工具末端安装

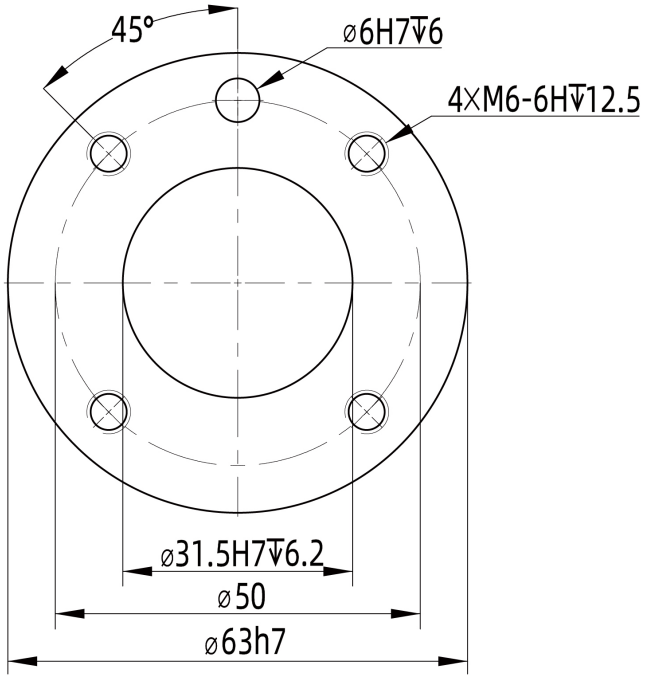
机器人工具法兰有四个 M6 螺纹孔，可用于将工具连接到机器人。M6 螺栓必须使用 8Nm 的扭矩拧紧，其强度等级不低于 8.8 级。为了准确地重新定位工具，请在预留的 Ø6 销孔中使用销钉。

图表 1.5-13 FR3/FR5/FR10/FR16 型号机器人末端法兰图纸

图表 1.5-14 FR20 型号机器人末端法兰图纸

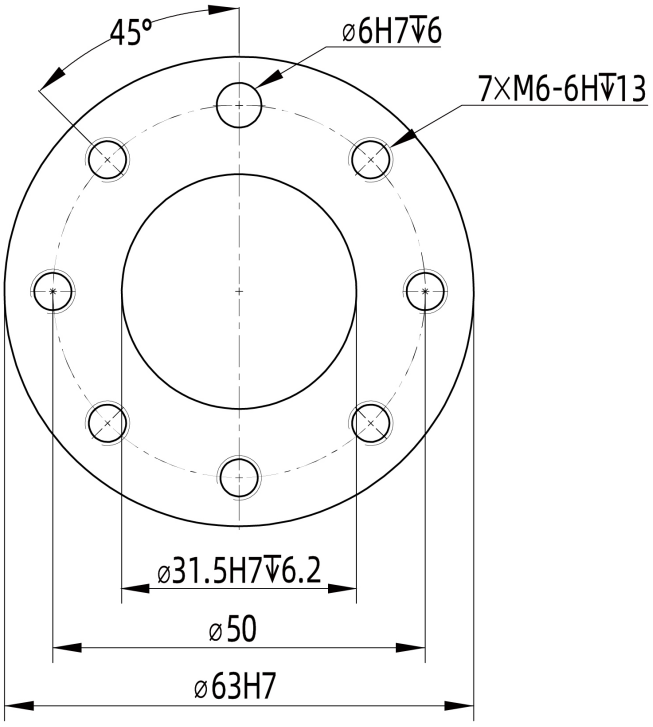
重要：

- 确保工具正确并安全地安装到位。
- 确保工具安全架构，不会有零件意外坠落造成危险。
- 在机器人上法兰上安装长度超过 8 毫米的 M6 螺栓可能会破坏工具法兰并造成无法修复的损坏，从而导致必须更换工具法兰。



机器人末端
均采用国际标准

机器人末端兼容工业机器人末端连接方式



1.3.3.4.3 安装环境

在安装及使用协作机器人时，确保满足以下要求：

- 环境温度 0-45℃
- 湿度 20-80RH 不结露
- 无机械冲击和震动
- 海拔要求 2000m 以下
- 无腐蚀性气体，无液体，无爆炸性气体，无油污，无盐雾，无尘埃或金属粉末，无放射性材料，无电磁噪声，无易燃物品
- 避免设备在电流的不稳定条件下工作
- 用户需要在机器人电源前增加不小于 10A/250V 关断能力的空气开关。

备注： 如果要将协作机器人吊装或者装到竖直面时，请联系我们。

1.3.3.4.4 地板承载能力

将机器人安装在一个坚固的表面，该表面应足以承受至少 5 倍的机器人手臂的重量，而且该表面不能有震动。

1.3.3.4.5 最大有效载荷

机器人手臂的最大允许有效载荷取决于重心偏移。当负载重心距离变远，机器人承受的负载会变小。

图表 1.5-15 FR3 型号协作机器人负载曲线

图表 1.5-16 FR5 型号协作机器人负载曲线

图表 1.5-17 FR10 型号协作机器人负载曲线

图表 1.5-18 FR16 型号协作机器人负载曲线

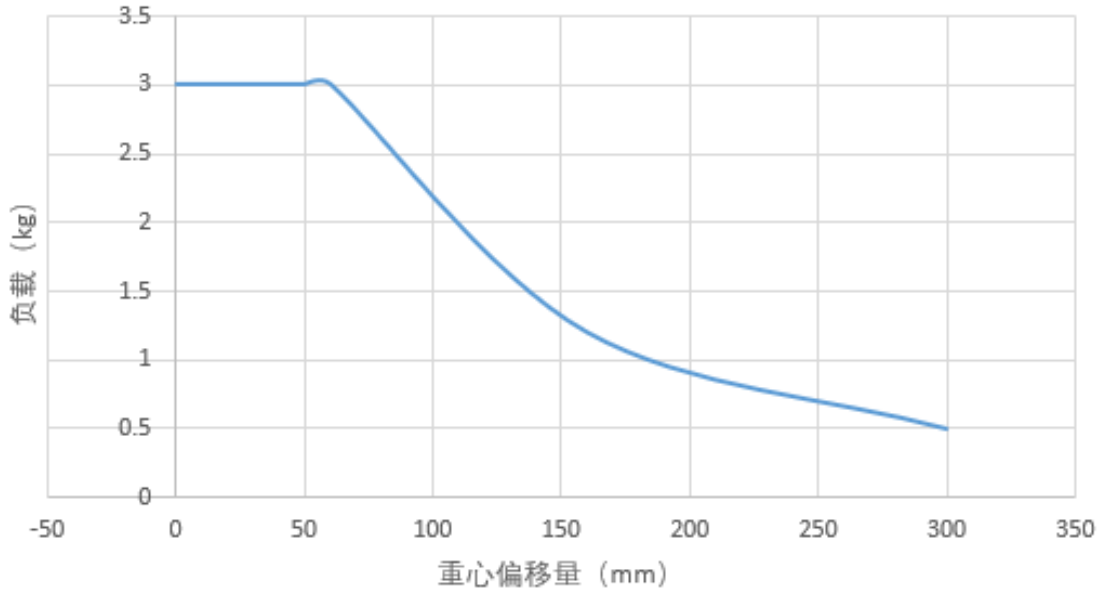
图表 1.5-19 FR20 型号协作机器人负载曲线

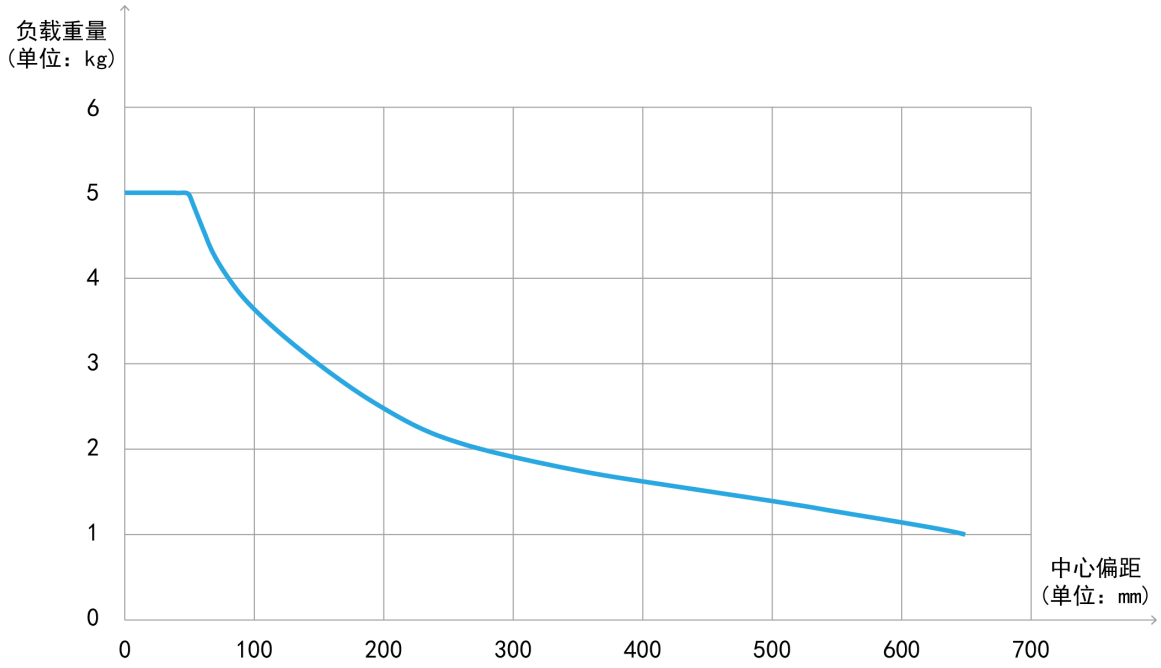
1.3.3.5 控制连接

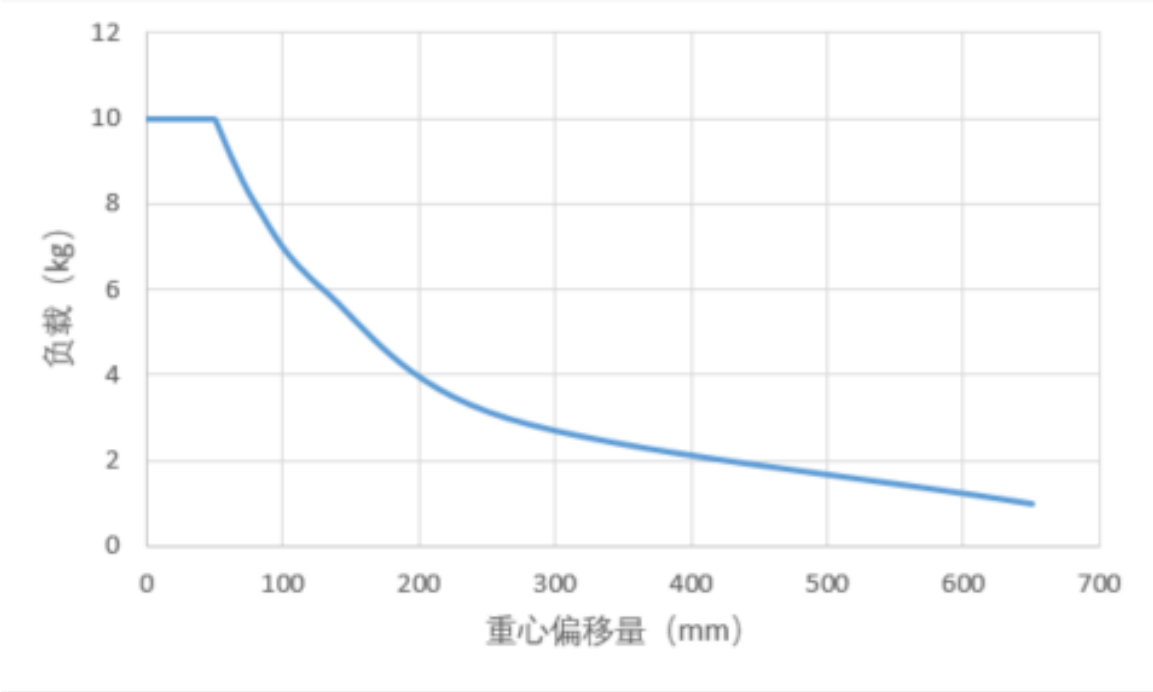
1.3.3.5.1 控制器接口

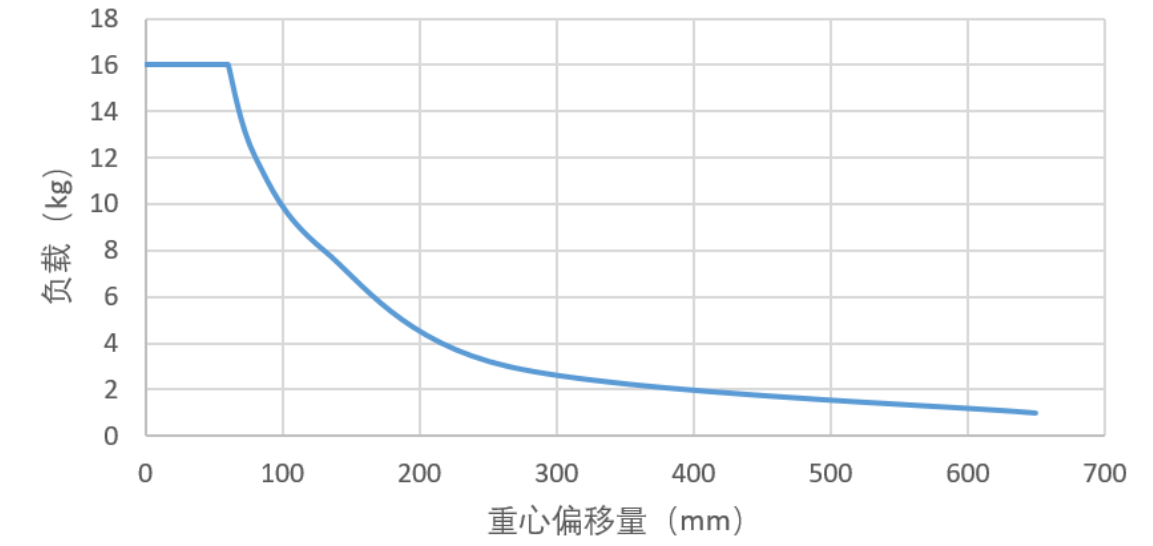
本系列机器人采用 TN-S 单相 220V 交流电源供电，设备自带 5 米电源线，三脚插头端插入现场提供的交流 220V 插座，机器人电气接地。

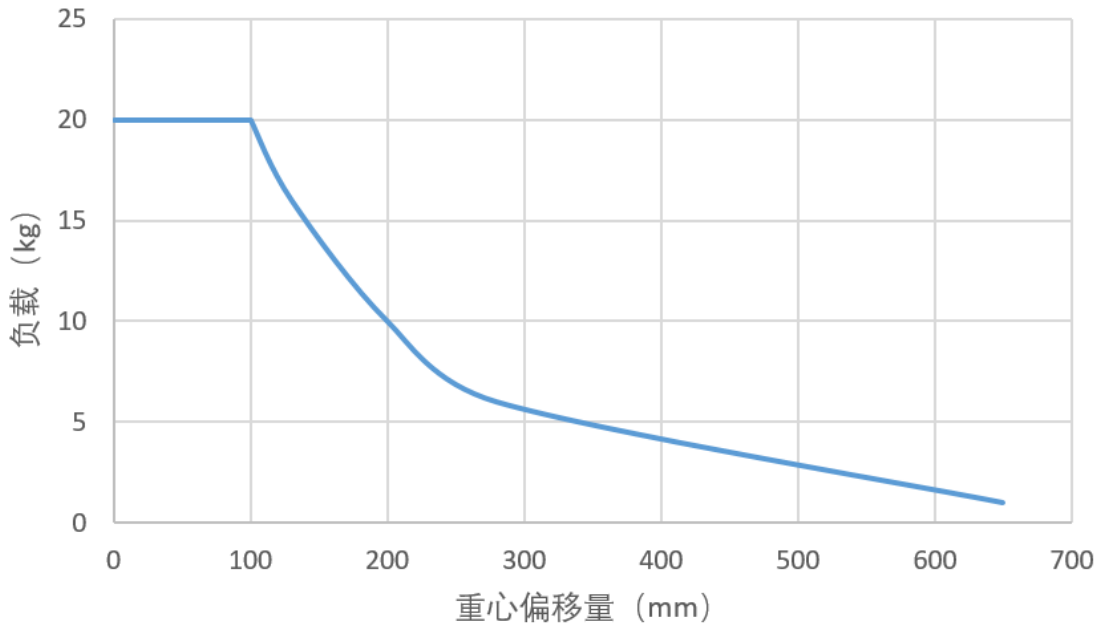
- 额定输入电压：6A/220VAC


3.5
3
2.5
2
1.5
1
0.5
0
负载（kg）
-50
0
50
100
150
200
250
300
350
重心偏移量（mm）


负载重量
（单位：kg）
6
5
4
3
2
1
0
0
100
200
300
400
500
600
700
中心偏距
（单位：mm）


12
10
8
6
4
2
0
负载（kg）
0
100
200
300
400
500
600
700
重心偏移量（mm）


18
16
14
12
10
8
6
4
2
0
负载（kg）
0
100
200
300
400
500
600
700
重心偏移量（mm）



- 额定输出电压：48V/21A
- 相数：单相
- 频率：50Hz
- 输出短路额定值：48V/22A

警告： 在接线前，请务必确保电源处于关闭状态，并在旁边挂放安全警示牌。

本系列机械手控制系统的外部连线均使用可插拔可快速安装的插头进行连接。协作机器人接线面板如图表 1.6-1 所示。

- 确保控制箱电源按钮关闭情况下（按钮打到 0）将 220V 电源线接到电源插口 (满载输入电压为 6A/220VAC~7A/210VAC)
- 将机器人本体重载线缆连接到控制箱重载接口
- 将按钮盒航空插头插到控制箱示教器接口
- 控制箱两侧散热口，间隔距离不少于 15CM
- 控制箱正面（用户钣金，开关电源键、重载与示教器线束）处，间隔距离不少于 25CM
- 控制箱距离地面 0.6-1.5m
- 不允许用户自行更换电源线缆

图表 1.6-1 机器人接线示意图

1.3.3.5.2 控制器 I/O 面板

您可以使用控制箱内的 I/O 来控制各种设备，包括气动继电器、PLC 和紧限位装置止按钮。图表 1.6-2 显示了控制箱的电气接口组，图表 1.6-3 显示了易制造控制箱的电气接口组。

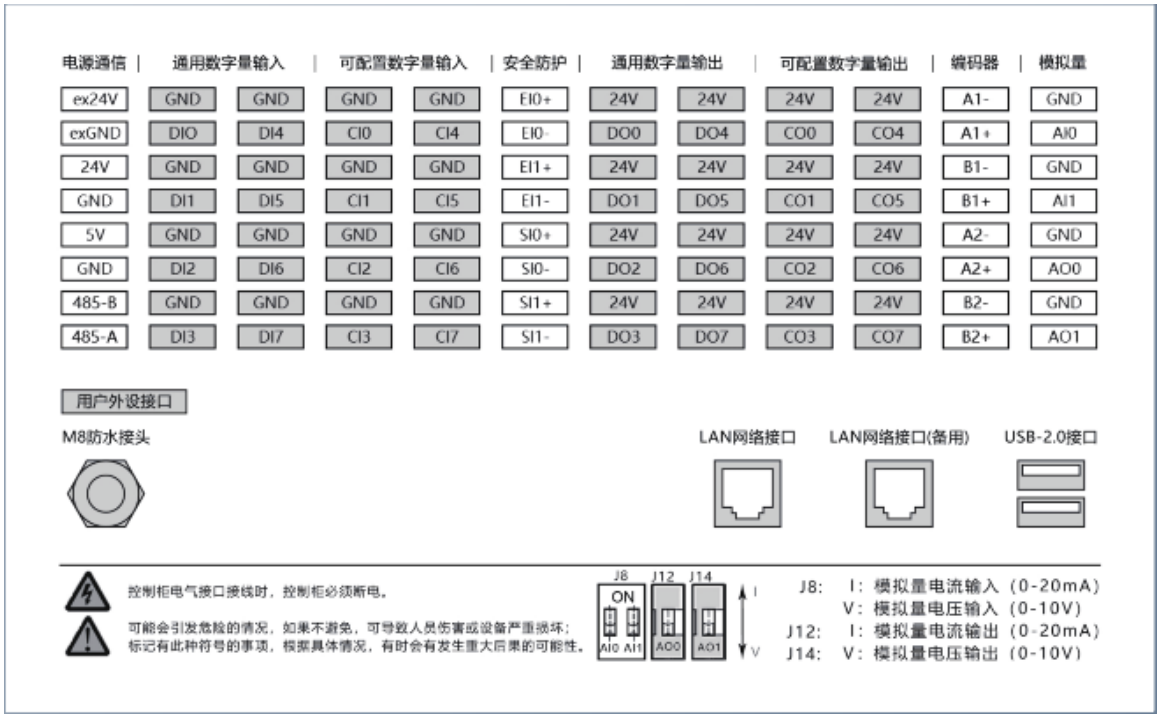
图表 1.6-2 控制箱电气接口示意图

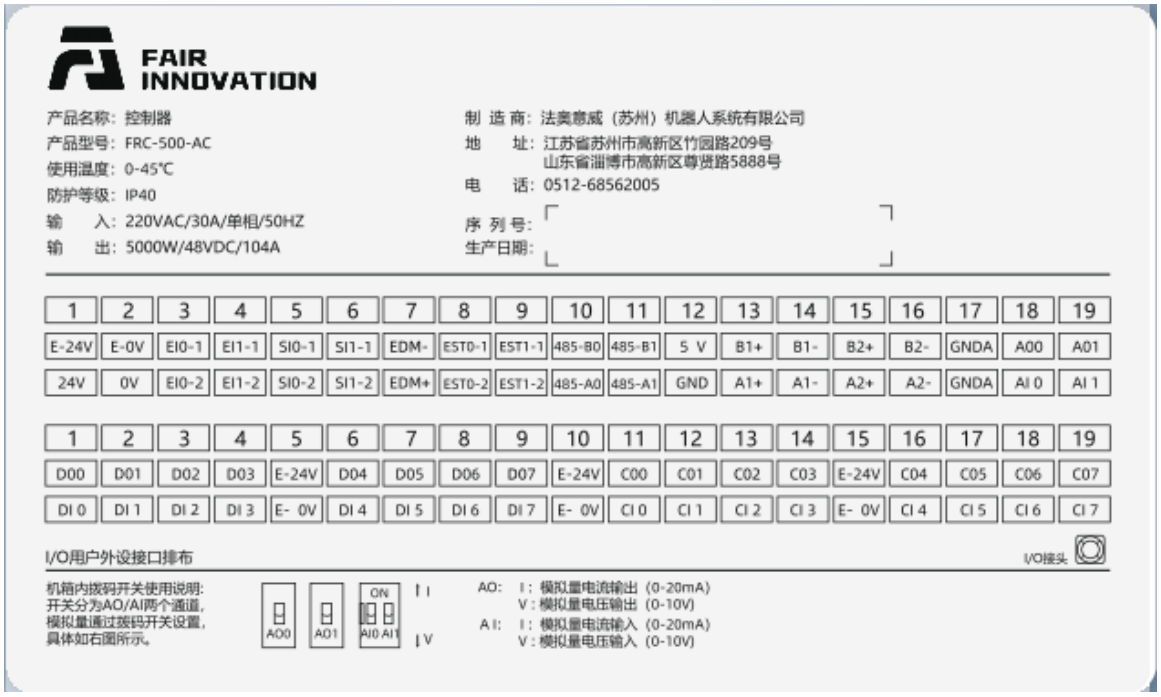
图表 1.6-3 易制造控制箱电气接口示意图

1.3.3.5.3 RJ45 网络接口组

控制箱内的网络接口组地址如图表 1.6-3 所示，注意该图与控制箱内部网口地址顺序对应，机器人默认端口禁止插拔。用户网口可以用来与相机等设备通信，IP 地址为 192.168.57.2。按钮盒接口默认为示教器控制端口，IP 地址为 192.168.58.2，使用网线连接按钮盒接口与电脑，电脑 IP 地址设为 192.168.58.10 或与之同一网段，打开谷歌浏览器输入 192.168.58.2 即可访问示教器页面。易制造控制箱通过连接按钮盒的网口，访问示教器页面。

图表 1.6-4 网络接口组示意图


电源通信
通用数字量输入
可配置数字量输入
安全防护
通用数字量输出
可配置数字量输出
编码器
模拟量
ex24V GND GND GND GND EI0+ 24V 24V 24V 24V A1- GND
exGND DI0 DI4 CI0 CI4 EI0- DO0 DO4 CO0 CO4 A1+ AI0
24V GND GND GND GND EI1+ 24V 24V 24V 24V B1- GND
GND DI1 DI5 CI1 CI5 EI1- DO1 DO5 CO1 CO5 B1+ AI1
5V GND GND GND GND SI0+ 24V 24V 24V 24V A2- GND
GND DI2 DI6 CI2 CI6 SI0- DO2 DO6 CO2 CO6 A2+ AO0
485-B GND GND GND GND SI1+ 24V 24V 24V 24V B2- GND
485-A DI3 DI7 CI3 CI7 SI1- DO3 DO7 CO3 CO7 B2+ AO1
用户外设接口
M8防水接头
LAN网络接口
LAN网络接口(备用)
USB-2.0接口
控制柜电气接口接线时，控制柜必须断电。
可能会引发危险的情况，如果不避免，可导致人员伤害或设备严重损坏；
标记有此种符号的事项，根据具体情况，有时会有发生重大后果的可能性。
J8 J12 J14
ON
AI0 AI1 AO0 AO1
I
V
J8: I：模拟量电流输入（0-20mA）
V：模拟量电压输入（0-10V）
J12: I：模拟量电流输出（0-20mA）
J14: V：模拟量电压输出（0-10V）


FAIR INNOVATION
产品名称：控制器
产品型号：FRC-500-AC
使用温度：0-45℃
防护等级：IP40
输　入：220VAC/30A/单相/50HZ
输　出：5000W/48VDC/104A
制 造 商：法奥意威（苏州）机器人系统有限公司
地　址：江苏省苏州市高新区竹园路209号
山东省淄博市高新区尊贤路5888号
电　话：0512-68562005
序 列 号：
生产日期：
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
E-24V E-0V EI0-1 EI1-1 SI0-1 SI1-1 EDM- EST0-1 EST1-1 485-B0 485-B1 5 V B1+ B1- B2+ B2- GNDA AO0 AO1
24V 0V EI0-2 EI1-2 SI0-2 SI1-2 EDM+ EST0-2 EST1-2 485-A0 485-A1 GND A1+ A1- A2+ A2- GNDA AI 0 AI 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
D00 D01 D02 D03 E-24V D04 D05 D06 D07 E-24V C00 C01 C02 C03 E-24V C04 C05 C06 C07
DI 0 DI 1 DI 2 DI 3 E- 0V DI 4 DI 5 DI 6 DI 7 E- 0V CI 0 CI 1 CI 2 CI 3 E- 0V CI 4 CI 5 CI 6 CI 7
I/O用户外设接口排布
I/O接头
机箱内拨码开关使用说明:
开关分为AO/AI两个通道,
模拟量通过拨码开关设置,
具体如右图所示。
AO0 AO1 ON AI0 AI1
I
V
AO: I：模拟量电流输出（0-20mA）
V：模拟量电压输出（0-10V）
AI: I：模拟量电流输入（0-20mA）
V：模拟量电压输入（0-10V）

1.3.3.5.4 末端板

您可以使用末端板的 I/O 和 485 通讯接口来控制各种设备，包括气动继电器、PLC 和紧急停止按钮。Pin 脚分布及其 pin 脚说明如图表 1.6-4 所示。I/O 连接器型号为 M12 连接器 8 芯母头。

图表 1.6-5 末端版电气接口示意图

1.3.3.5.5 接地说明

1. 控制箱接地位于电源开关左上方 M4 组合螺钉处，如图表 1.6-5 所示。

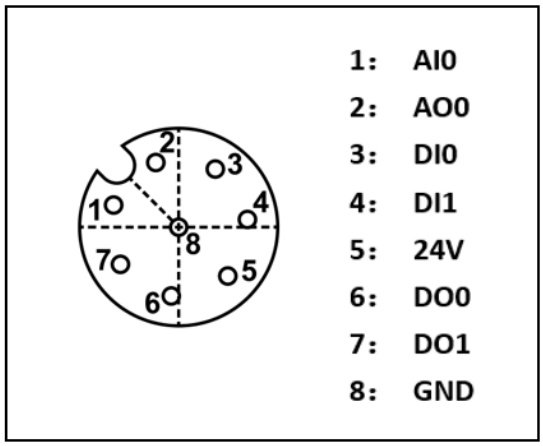
   图表 1.6-6 控制箱接地示意图

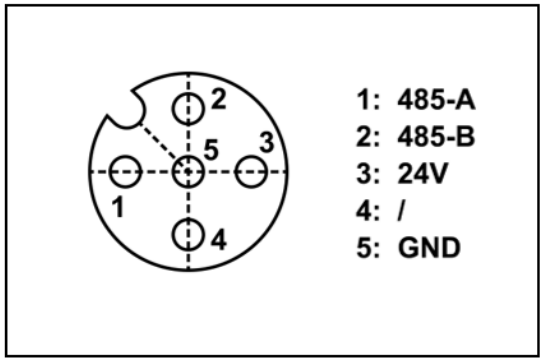
2. 本体接地位于基座出线处的右侧位置，如如图表 1.6-6 所示。

   图表 1.6-7 本体接地示意图

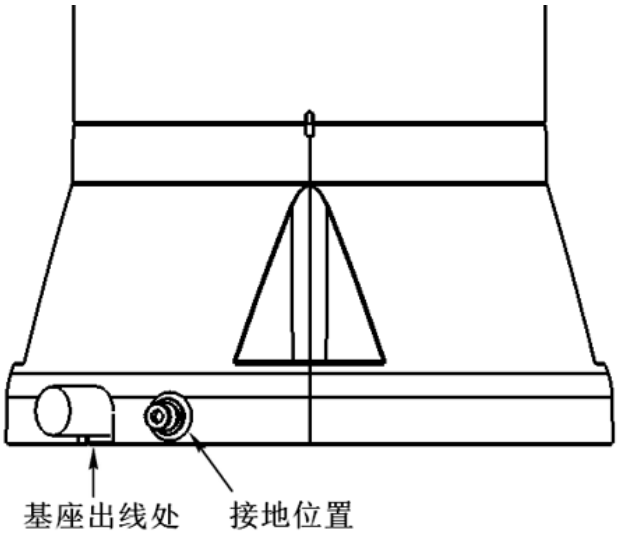
单独使用的保护导线，截面积不应小于：

- 2.5mm^2 铜或 16mm^2 铝，如果提供机械损伤防护（导线管、管道等）
- 4mm^2 铜或 16mm^2 铝，如果没有提供机械损伤防护


1： AI0
2： AO0
3： DI0
4： DI1
5： 24V
6： DO0
7： DO1
8： GND


1: 485-A
2: 485-B
3: 24V
4: /
5: GND


控制箱外部
接地点，使用
M4 组合螺钉
IO 1
AC176-264V
ROBOT
PWR
PROFT
HMI
LAN1
LAN2
USB
USB
HDMI


基座出线处
接地位置

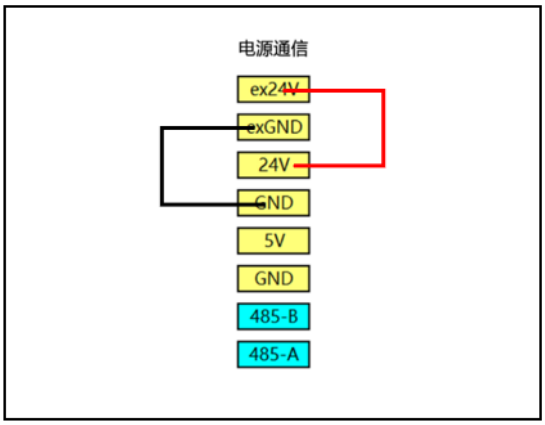
1.3.3.5.6 所有数字 I/O 的通用规范

本节规定了下列控制箱 24 伏数字输入/输出的电气规范：

- 安全 I/O
- 通用数字量 I/O

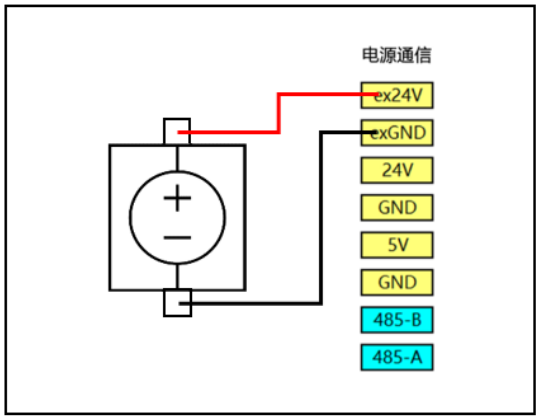
机器人必须按照电气规范进行安装。

通过配置“电源通讯”接口，可以使用内部或外部 24V 电源为数字 I/O 供电。该接口中上面两个端子（ex24V 和 exon）为外部电源的 24V 和地，下面两个端子（24V 和 GND）为内部电源的 24V 和地。默认配置是使用内部电源，如图表 1.6-7 所示。



图表 1.6-8 电源通信示意图 01

如果负载功率较大，可以按如图 1.6-8 连接外部电源。



图表 1.6-9 电源通信示意图 02

内部和外部电源的电气规格如表格 1.6-1 内外部电气规格所示：

表格 1.6-1 内外部电源电气规格

端子	参数	最小值	典型值	最大值	单位
内部24V电源					
[ex24V – exGND]	**电压**	23	24	25	V
[ex24V – exGND]	**电流**	0	-	2	A
内部24V电源					
[24V – GND]	**电压**	23	24	25	V
[24V – GND]	**电流**	0	-	1.5	A

数字量 I/O 的电气规格如表格 1.6-2 数字量 I/O 电气规格所示：

端子	参数	最小值	典型值	最大值	单位
数字量输出					
[COx / DOx]	**电流**	0	-	1	A
[COx / DOx]	**压降**	0	-	0.5	V
[COx / DOx]	**漏电流**	0	-	0.1	mA
[COx / DOx]	**功能**	-	NPN	-	Type
数字量输入					
[EIx/SIx/CIx/DIx]	OFF	-3	-	5	V
[EIx/SIx/CIx/DIx]	ON	11	-	30	V
[EIx/SIx/CIx/DIx]	**电流**(11-30V)	2	-	15	mA
[EIx/SIx/CIx/DIx]	**功能**	-	NPN	-	Type

表格 1.6-2 数字量 I/O 电气规格

1.3.3.5.7 安全 I/O

本节描述了安全 I/O 的电气规范，必须遵守第 1.6.6 节中的通用电气规范。

安全装置和设备必须按照安全说明和风险评估进行安装，见第 1.1。所有安全 I/O 都是成对的（冗余），必须作为两个独立的分支保存。单一故障不应导致安全功能丧失。

安全 I/O 包括紧急停止和安全停止。紧急停止输入仅用于紧急停止设备，安全停止输入用于各种安全相关保护设备。功能差异如表格 1.6-3 所示：

表格 1.6-3 功能差异

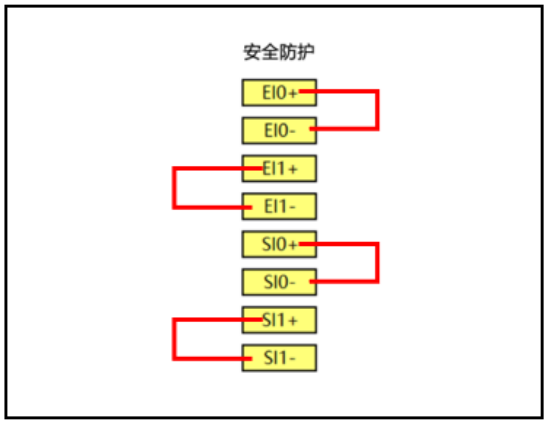
	紧急停止	安全停止
机器人停止移动	是	是
停止类别	类别 0	类别 1
程序执行	停止	暂停
机器人电源	关闭	打开
重启	手动	自动或手动
使用频率	不频繁	经常
需要重新初始化	需要	不需要

警告:

- 切勿将安全信号连接到不具有正确安全级别的 PLC。如果不遵守此警告，则可能导致严重伤害或死亡，因为其中一个安全停止功能可能被覆盖。必须将安全接口信号与正常 I/O 接口信号分开。
- 所有与安全相关的 I/O 都是冗余构建的（两个独立通道）。必须保持两个通道分开，以便单个故障不会导致安全功能丧失。
- 在将机器人投入运行之前，必须验证急停安全功能（机器人通电使能，按下急停按钮，机器人断电停止，关闭电源，旋转急停按钮，打开电源，机器人重新上电使能）。必须定期测试安全功能。
- 机器人安装应符合这些规范。否则可能导致严重伤害或死亡，因为安全停止功能可能被越过。

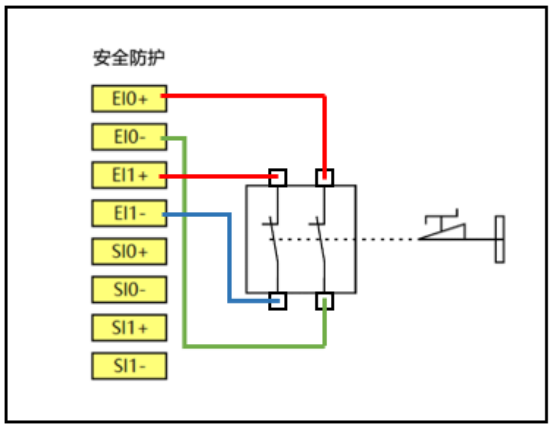
以下小节给出了一些如何使用安全 I/O 的示例。

默认安全配置机器人出厂时带有默认配置，无需任何附加安全设备即可进行操作，请参见图表 1.6-9。



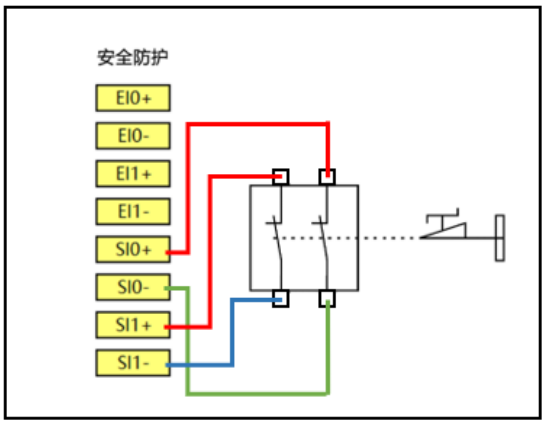
图表 1.6-10 安全防护示意图 01

连接紧急停止按钮在大多数应用中，需要使用一个或多个额外的紧急停止按钮。见图表 1.6-10。



图表 1.6-11 安全防护示意图 02

连接安全停止按钮安全停止装置的一个例子是当门打开时机器人停止的门开关，见图表 1.6-11。



图表 1.6-12 安全防护示意图 03

1.3.3.5.8 通用数字量 I/O

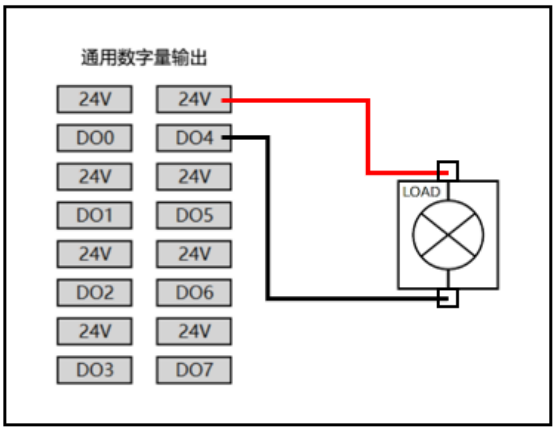
本节描述了通用数字量 I/O 的电气规范，必须遵守第 1.6.6 节中的通用电气规范。

通用数字量 I/O 可用于驱动继电器、电磁阀等设备或与其他 PLC 进行交互。

数字量输出控制负载

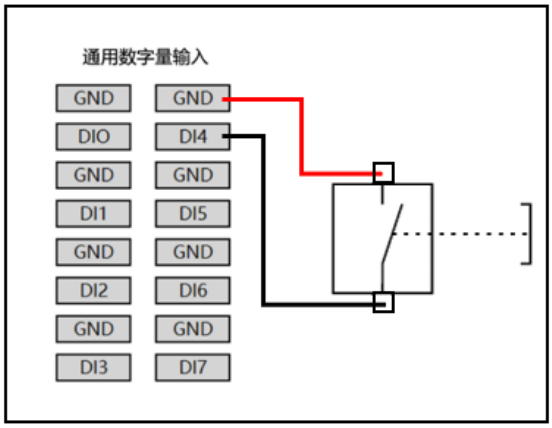
此示例演示如何连接数字量输出从而控制负载，请参见图表 1.6-12。

图表 1.6-13 通用数字量输出示意图 01



1.3.3.5.9 从按钮进行的数字输入

下面的示例演示如何将简单按钮连接到数字量输入。

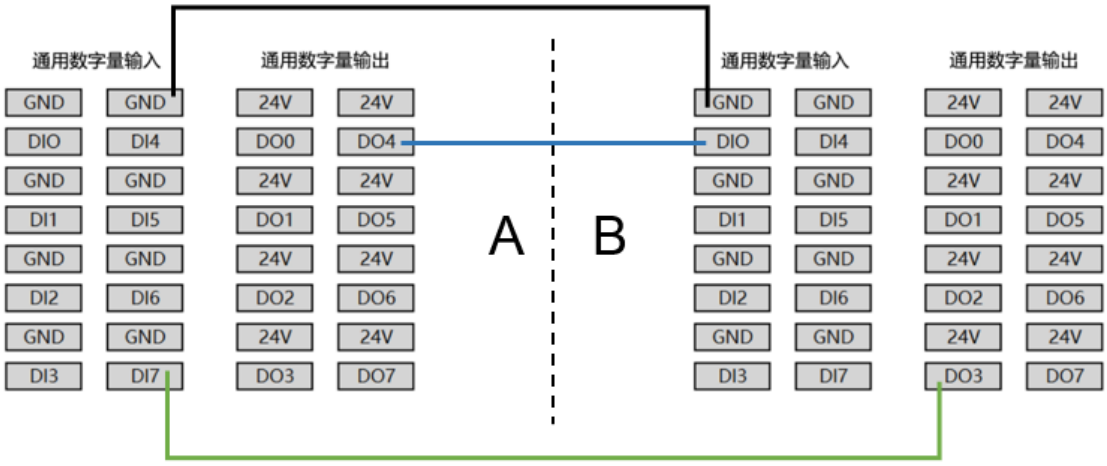


图表 1.6-14 通用数字量输出示意图 02

1.3.3.5.10 与其他设备或 PLC 交互

下面的示例演示如何与其他设备或 PLC 进行数字量输入输出交互。

图表 1.6-15 与其他设备或 PLC 交互示意图



1.3.3.5.11 模拟量 I/O

表格 1.6-4 模拟量电流电压

模拟量 I/O 用来设置或测量其它设备的电压（0-10V）或电流（0-20mA）。

为了达到高精度，建议采用以下方法。

- 设备和控制箱使用相同的地（GND）。
- 使用屏蔽电缆或双绞线。

下面的示例演示如何使用模拟量 I/O。

使用模拟量输出

下面的示例是演示使用模拟量输出控制传送带。

图表 1.6-16 模拟量输出示意图

使用模拟量输入

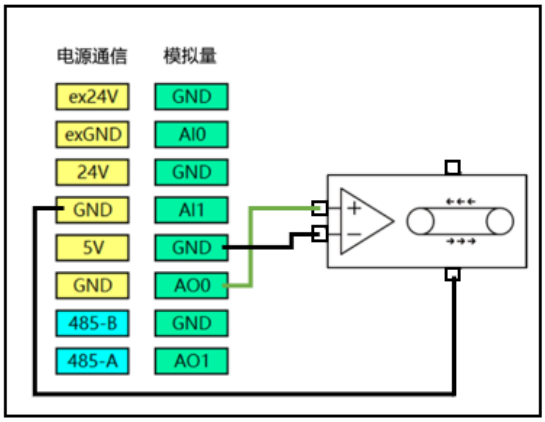
下面的示例是演示使用模拟量输入连接模拟传感器。

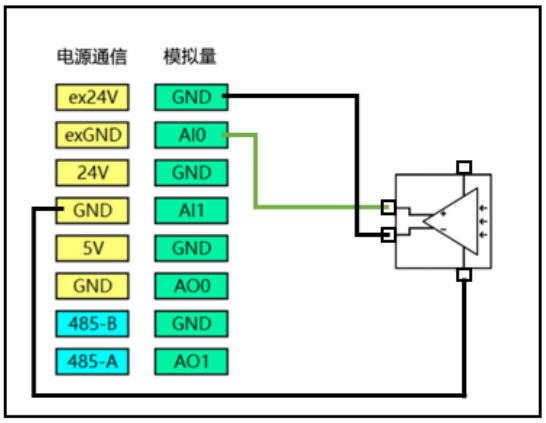
图表 1.6-17 模拟量输入示意图

1.3.3.6 示教器及末端 LED

机器人示教器可以使用一台电脑或者平板访问并控制机器人，连接方式可参考 1.6.3 节说明，此外用户也可以使用我们的 FR-HMI 示教器，该款示教器是选配件。

端子	参数	最小值	典型值	最大值	单位
模拟量电流输入					
[AIx - END]	**电流**	0	-	20	mA
[AIx - END]	**阻抗**	-	500	-	ohm
[AIx - END]	**分辨率**	-	12	-	bit
模拟量电压输入					
[AIx - END]	**电压**	0	-	10	V
[AIx - END]	**阻抗**	-	510	-	Kohm
[AIx - END]	**分辨率**	-	12	-	bit
模拟量电流输出					
[AOx - END]	**电流**	0	-	20	mA
[AOx - END]	**电压**	0	-	10	V
[AOx - END]	**分辨率**	-	12	-	bit
模拟量电压输出					
[AOx - END]	**电压**	0	-	10	V
[AOx - END]	**电流**	0	-	20	mA
[AOx - END]	**阻抗**	-	100	-	ohm
[AOx - END]	**分辨率**	-	12	-	bit





1.3.3.6.1 按钮盒简介

第一版按钮盒

图表 1.7-1 第一版按钮盒

急停开关：当按下急停开关，机器人进入紧急停止状态。

Type-c 接口：连接 web 示教器的端口。

按键 1：短按自动/手动模式切换，长按进入/退出拖动模式。

按键 2：短按记录示教点，长按进入/退出不搭配示教器状态。

按键 3：短按开始/停止运行程序。

第二版按钮盒

图表 1.7-2 第二版按钮盒

急停开关：当按下急停开关，机器人进入紧急停止状态。

开始停止：开始/停止运行程序。

网口：连接 web 示教器。

关机：暂未启用。

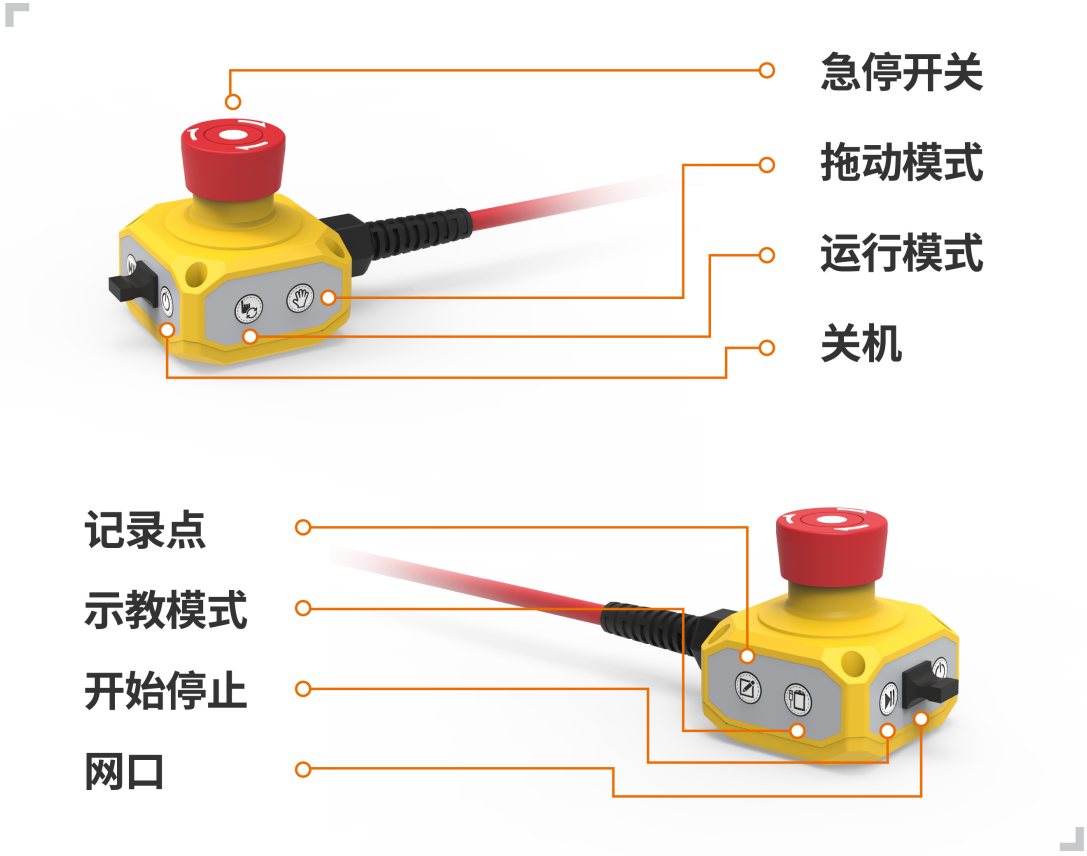
记录点：记录示教点。

示教模式：进入/退出搭配示教器状态。

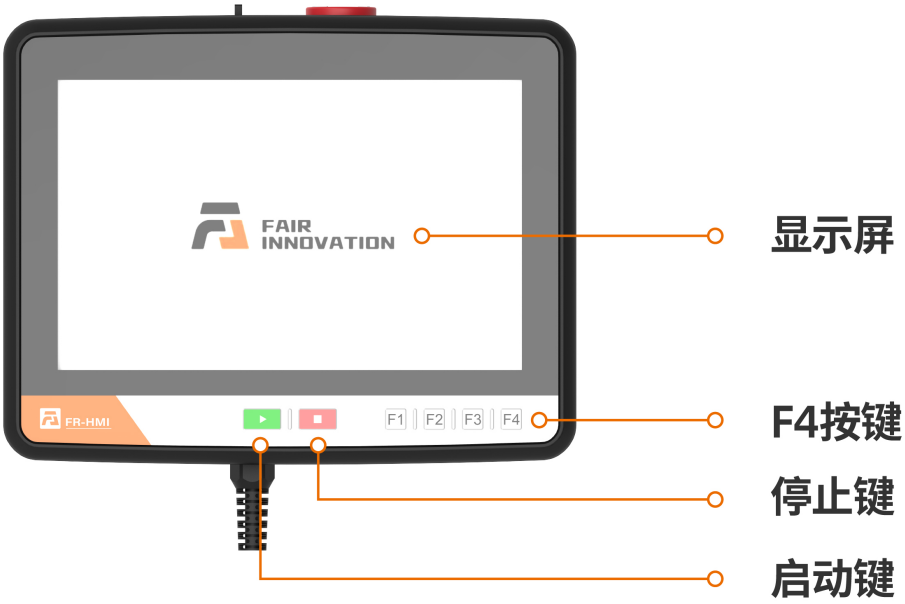
运行模式：自动/手动模式切换。

拖动模式：进入/退出拖动模式。


急停开关
按键 1
按键 2
Type-c 接口
按键 3


急停开关
拖动模式
运行模式
关机
记录点
示教模式
开始停止
网口

1.3.3.6.2 FR-HMI 示教器简介



图表 1.7-3 FR-HMI 示教器正面

图表 1.7-4 FR-HMI 示教器背面

显示屏：示教器的触摸操作与显示界面。

启动键：启动程序。

停止键：停止当前运行的程序。

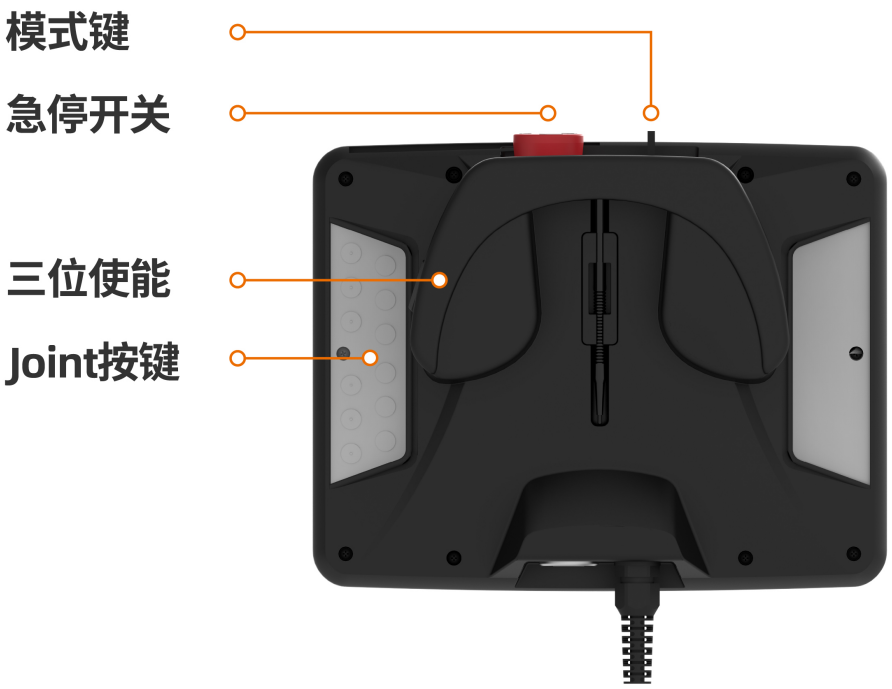
F4 按键：输入键盘调用按键。

Joint 按键：机器人关节点动。

三位使能：手动模式使能机器人

急停开关：当按下急停开关，机器人进入紧急停止状态。

模式键：旋转按钮切换手自动模式。


模式键
急停开关
三位使能
Joint按键

1.3.3.6.3 末端 LED 定义

功能	LED 颜色
通信未建立时	“灭” “红” “绿” “蓝” 交替
自动模式	蓝色长亮
手动模式	绿色长亮
拖动模式	白青色长亮
按钮盒记录点（仅在使用按钮盒时）	紫色闪烁两下
进入未搭配按钮盒状态(仅在使用按钮盒时)	青蓝色闪烁两下
开始运行（仅在使用按钮盒时）	蓝色闪烁两下
停止运行（仅在使用按钮盒时）	红色闪烁两下
报错（仅在使用按钮盒时）	红色长亮
校零完成	白青色闪烁三下
去使能	黄色闪烁两下

表格 1.7-1 末端 LED 定义表

1.3.4 快速启动机器人

1.3.4.1 安装机器人手臂和控制箱

根据 2. 硬件安装中的 2.5 和 2.6 安装连接机器人手臂和控制箱。

- 开箱取出机器人手臂，使用 4 颗强度不低于 8.8 级强度的 M8 螺栓安装机器人手臂。将机器人手臂安装在一个坚固且防震的表面，若用铝板固定，铝板厚度不小于 16mm，若用铁板固定，铁板厚度不小于 8mm；
- 将控制箱放置在其支脚上；

- 将机器人手臂本体重载线缆连接到控制箱重载接口；
- 将按钮盒航空插头插到控制箱示教器接口，若配备有触摸屏版示教器，还需要使用一根两端都为 Type-c 接口的数据线，分别插入按钮盒与示教器的 Type-c 接口；
- 确保控制箱电源按钮关闭情况下（按钮打到 0）将 220V 电源线接到电源插口；
- 插上电源控制箱插头。

警告： 如果机器人没有安全地放置在坚固的表面上，机器人有可能会倾倒并造成伤害。

1.3.4.2 示教器启动控制机器人

控制箱连接机器人手臂、示教盒和任何周边设备的物理电气输入/输出端。必须打开控制箱才能给机器人手臂通电。

- 按下控制箱的电源按钮开启控制箱；
- 启动机器人后，此时机器人为手动模式且未使能，若需要在手动模式下操作机器人，需要按压示教器上三位使能开关 OFF（放开）⇒ ON ⇒ OFF（按压），当开关处于 ON 状态时，拖动或控制机器人运动。
- 若无需在手动模式操作机器人，可用示教器上钥匙开关旋转按钮切换机器人工作模式：自动、手动、自定义；
- 当切换机器人手动状态时，应检查安全空间内外是否存在异常并谨慎操作机器运行；
- 当切换机器人自动状态时，应检查安全措施并恢复到正常状态下并谨慎操作机器运行；
- 当无法正常打开示教器时，请查看设备连接是否正常。

1.3.4.3 按钮盒控制机器人运动

参照 2. 硬件安装的 2.7.3. 末端 LED 定义来控制机器人

1.3.4.3.1 未搭配示教器

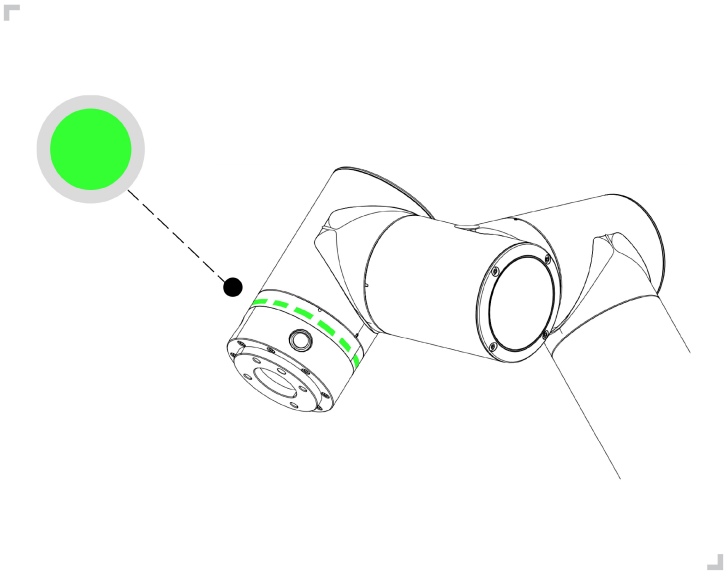
- **Step1**：打开机器人控制箱电源开关，启动机器人，等待末端 LED 长显绿色后，方可操作机器人如图表 2.3-1。

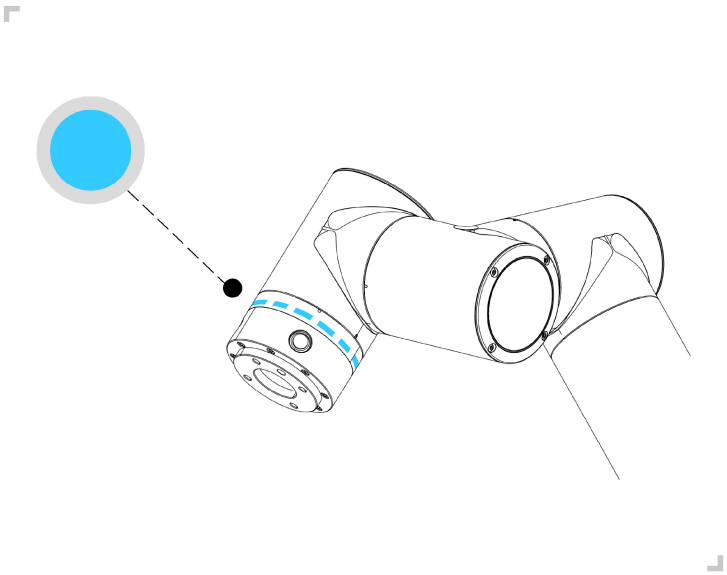
图表 2.3-1 末端 LED 绿色示意图

- **Step2**：长按按钮盒“按键 2”，进入未搭配示教器模式，末端 LED 青蓝色闪烁三下，如图表 2.3-2。

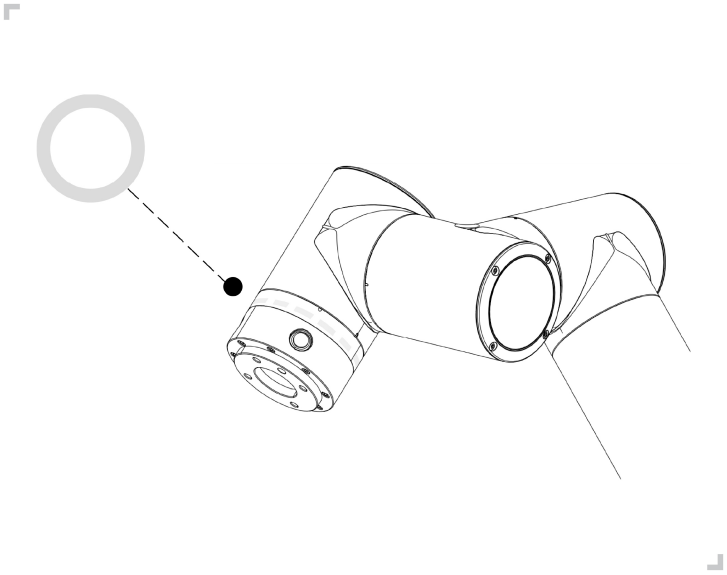
图表 2.3-2 末端 LED 青蓝色示意图

- **Step3**：长按按钮盒“按键 1”切换机器人到拖动模式，此时末端 LED 为白青色，如图表 2.3-3。移动机器人至任意位置，长按“按键 1”退出拖动模式，短按按钮盒“按键 2”记录 P1 点，末端 LED 紫色闪烁三下，如图表 2.3-4。

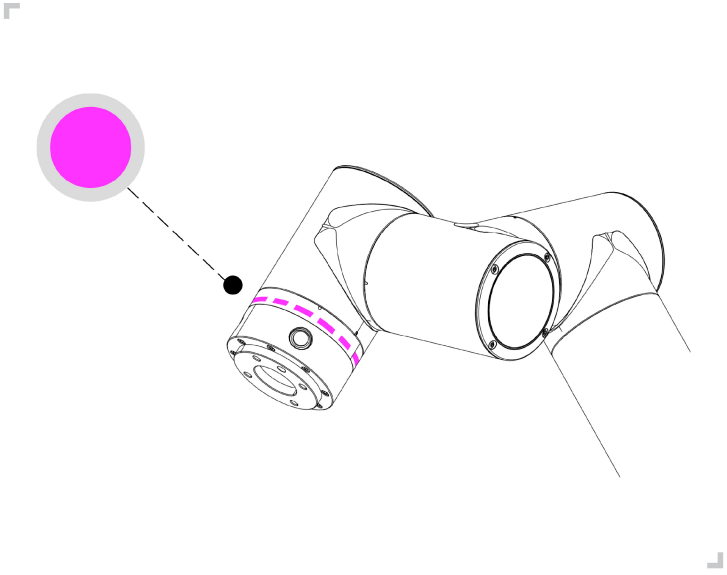




- **Step4**：移动机器人，短按按钮盒“按键 2”记录 P2 点，末端 LED 紫色闪烁三下，如图表 2.3-4。



图表 2.3-3 末端 LED 白青色示意图

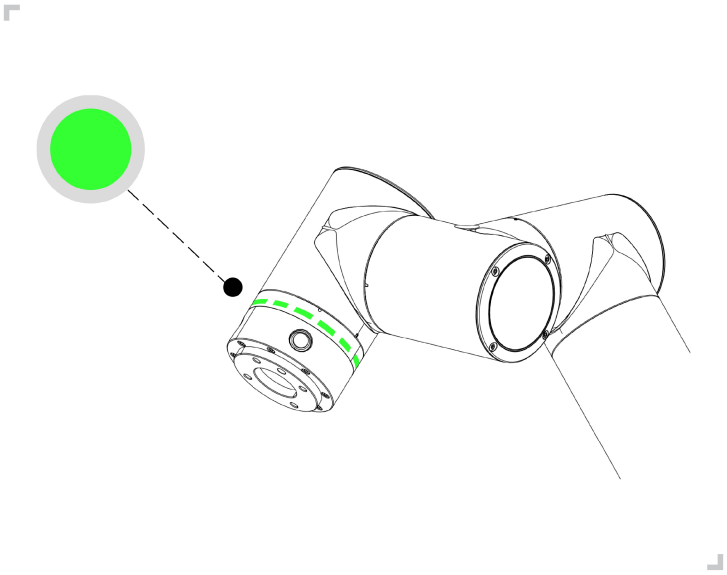


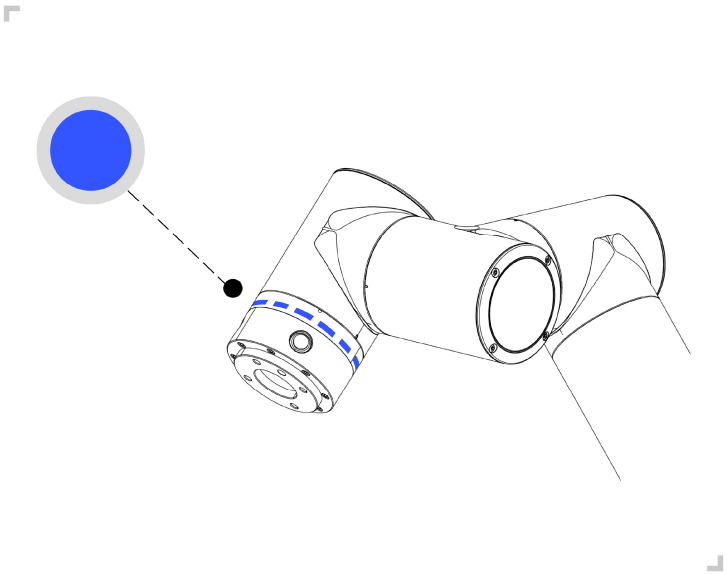
图表 2.3-4 末端 LED 紫色示意图

- **Step5**：长按按钮盒“按键 1”退出拖动模式，此时为手动模式，末端 LED 为绿色，如图表 2.3-5。短按“按键 1”切换机器人到自动模式，此时末端 LED 为蓝色，如图表 2.3-6。
- **Step6**：短按按钮盒“按键 3”运行该程序，末端 LED 蓝色闪烁两下，如图表 2.3-6。

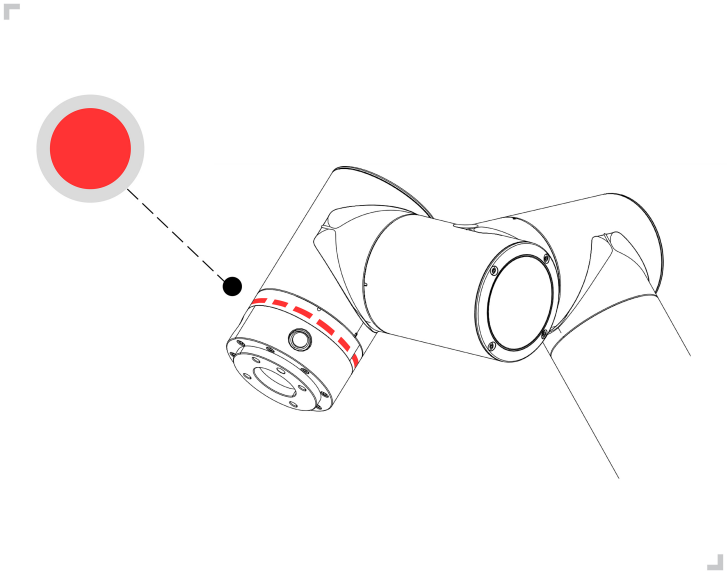
图表 2.3-5 末端 LED 绿色示意图

图表 2.3-6 末端 LED 蓝色示意图





- **Step7**：短按按钮盒“按键 3”停止运行该程序，末端 LED 红色闪烁三下，如图表 2.3-7。



图表 2.3-7 末端 LED 红色示意图

1.3.4.3.2 搭配示教器

- **Step1**：启动机器人，等待末端 LED 绿色停止闪烁，方可操作机器人。
- **Step2**：打开示教器进入到程序编辑界面。
- **Step3**：选择空白模板新建一个程序文件。
- **Step4**：短按按钮盒按键 1 切换机器人到手动模式，此时末端 LED 为绿色。
- **Step5**：长按按钮盒按键 1 切换机器人到拖动模式，此时末端 LED 为白青色，移动机器人至任意位置，短按按钮盒按键 2 记录 P1 点，末端 LED 紫色闪烁三下，手动添加“PTP:P1”指令到程序文件中。

图表 2.3-8 记录并添加点 P1

- **Step6**：移动机器人，短按按钮盒按键 2 记录 P2 点，末端 LED 紫色闪烁三下，手动添加“PTP：P2”指令到程序中。

图表 2.3-9 记录并添加点 P2

- **Step7**：保存程序文件内容。
- **Step8**：长按按钮盒按键 1 退出拖动模式，此时为手动模式，末端 LED 为绿色，短按按钮盒按键 1 切换机器人到自动模式，此时末端 LED 为蓝色。
- **Step9**：短按按钮盒按键 3 运行该程序，末端 LED 蓝色闪烁两下。

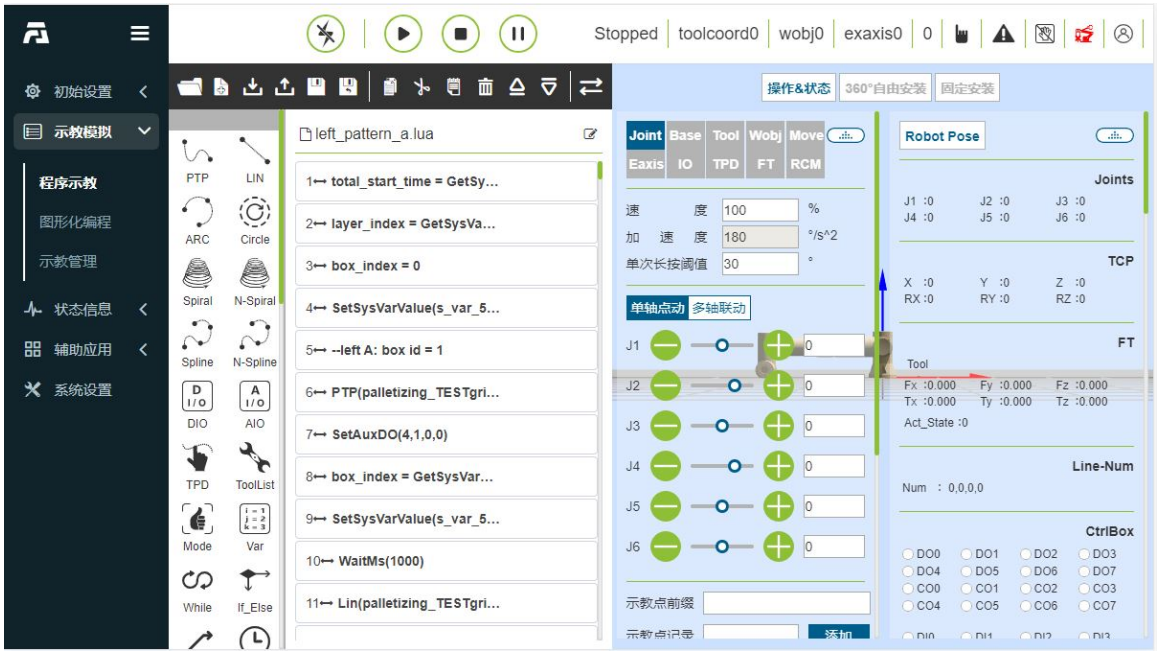

test.lua
1 ⟷ PTP:P1,100


test.lua
1 ⟷ PTP:P1,100
2 ⟷ PTP:P2,100

1.3.4.4 示教器控制机器人运动

点击示教器左侧一级菜单中的“示教模拟”按钮，点击其子菜单“程序示教”进入程序示教界面，该界面中主要实现机器人示教程序的编写以及修改。

点击“新建”图标按钮后，用户命名该文件，并选择一个模板作为该新建文件的内容，点击新建即可创建成功并打开该程序文件。



图表 2.4-1 示教程序运行示意图

警告： 您的头和躯干不能位于机器人可接触到的范围（工作区）。请不要将您的手指放在机器人可抓住的地方。

重要:

- 不要让机器人移到自身或其他物体中，因为这会对机器人造成损害。
- 这只是一个快速启动指南，教您如何轻松地使用 FR 协作机器人。该指南的前提是环境安全无害，用户谨慎小心。请不要将速度或者加速度上调至默认值之上。在使机器人进入操作之前，始终进行风险评估。

1.3.5 示教器软件解析

1.3.5.1 基础信息

1.3.5.1.1 系统简介

示教器软件是针对机器人开发的配套软件，运行于示教器操作系统中，其主要功能和技术特点如下：

- 能够对机器人进行示教程序的编写；
- 能够实时显示机器人位置坐标，三维模拟实体机器人，并能控制机器人运动；
- 能够实现对机器人的单轴点动以及各轴联动操作；
- 能够查看控制 IO 状态；
- 用户可以修改密码、查看系统信息等。

1.3.5.1.2 启动软件

1. 控制箱上电；
2. 示教器打开浏览器访问目标网址 192.168.58.2；
3. 输入用户名和密码点击登录即可登录系统。

1.3.5.1.3 用户登录及权限更新

表格 3.1-1 初始用户

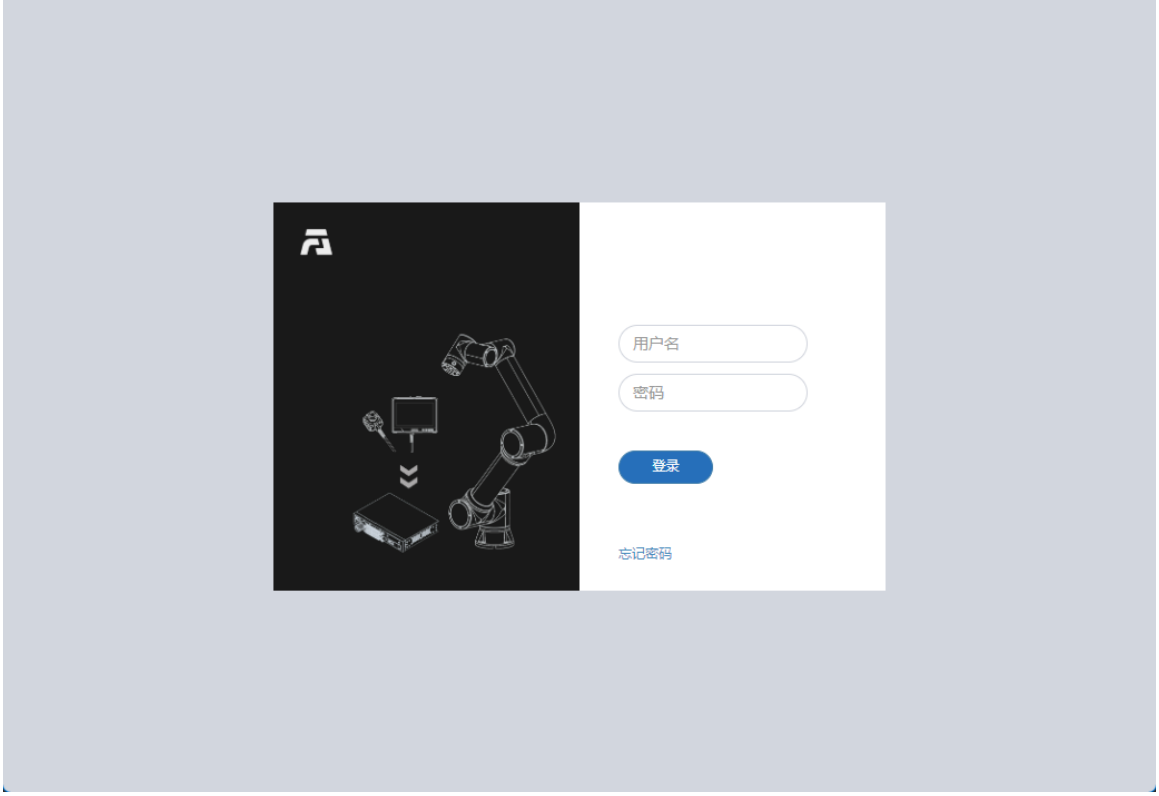
工号	初始用户名	密码	职能代号
111	admin	123	1
222	MEenginer	222	2
333	PEenginer	333	3
444	programmer	444	4
555	operator	555	5
666	monitor	666	6

用户（用户管理参考*5.10.2.1 用户管理*）默认分为六个等级，管理员无功能限制，操作员和监视员少部分功能可以使用，ME 工程师、PE&PQE 工程师和技术员 & 班组长部分功能限制，管理员无功能限制，具体默认职能代号权限参考*5.10.2.2 权限管理*。

登录界面如图表 3.1-1 登录界面所示。

图表 3.1-1 登录界面

登录成功后，系统会加载模型等数据，加载完毕后进入初始页面。



1.3.5.2 系统初始界面

登录成功后系统进入“初始界面”，初始界面展示了示教器主要包含法奥 LOGO 及返回初始页面按钮、菜单栏、菜单栏缩放按钮、机器人操作区、控制区、状态区、三维模拟机器人以及位姿及 IO 信息区，一共八个区域。如图表 3.2-1 系统初始界面示意图所示。

图表 3.2-1 系统初始界面示意图

1.3.5.2.1 控制区

备注:

名称：**使能按钮**

作用：使能机器人

备注:

名称：**开始按钮**

作用：上传并开始运行示教程序

备注:

名称：**停止按钮**

作用：停止当前示教程序运行

备注:

名称：**暂停/恢复按钮**

作用：暂停和恢复当前示教程序

重要: 暂停指令在程序的末尾，无法进行判断

1.3.5.2.2 状态栏

备注:

名称：**机器人状态**

作用：Stopped-停止，Running-运行，Pause-暂停，Drag-拖动

备注:

名称：**工具坐标系编号**

作用：展示当前应用的工具坐标系编号

备注:

名称：**运行速度百分比**

作用：机器人当前模式运行时速度

备注:

名称：**机器人运行正常状态**

作用：当前机器人正常运行

备注:

名称：**机器人运行错误状态**

作用：当前机器人运行有错误

备注:

名称：**自动模式**

作用：机器人自动运行模式，开启手动切自动模式全局速度调整并指定速度时，全局速度会自动调整为指定速度

备注:

名称：**示教模式**

作用：机器人示教运行模式

备注:

名称：**机器人拖动状态**

作用：当前机器人可拖动

备注:

名称：**机器人拖动状态**

作用：当前机器人不可拖动

备注:

名称：**连接状态**

作用：机器人已连接

备注:

名称：**未连接状态**

作用：机器人未连接

备注:

名称：**账户信息**

作用：显示用户名和权限及登出用户

1.3.5.2.3 菜单栏

菜单栏如表格 3.2-1 示教器菜单分栏

一级	二级
初始设置	机器人设置
	用户外设配置
示教模拟	程序示教
	图形化编程
	示教管理
状态信息	系统日志
	状态查询
辅助应用	机器人本体
	焊接专家库
	安全性设置
系统设置	/

表格 3.2-1 示教器菜单分栏

1.3.5.2.4 操作区

IO 设置可参考 4.5 控制箱 I/O 中的*4.5.1 I/O 设置*。

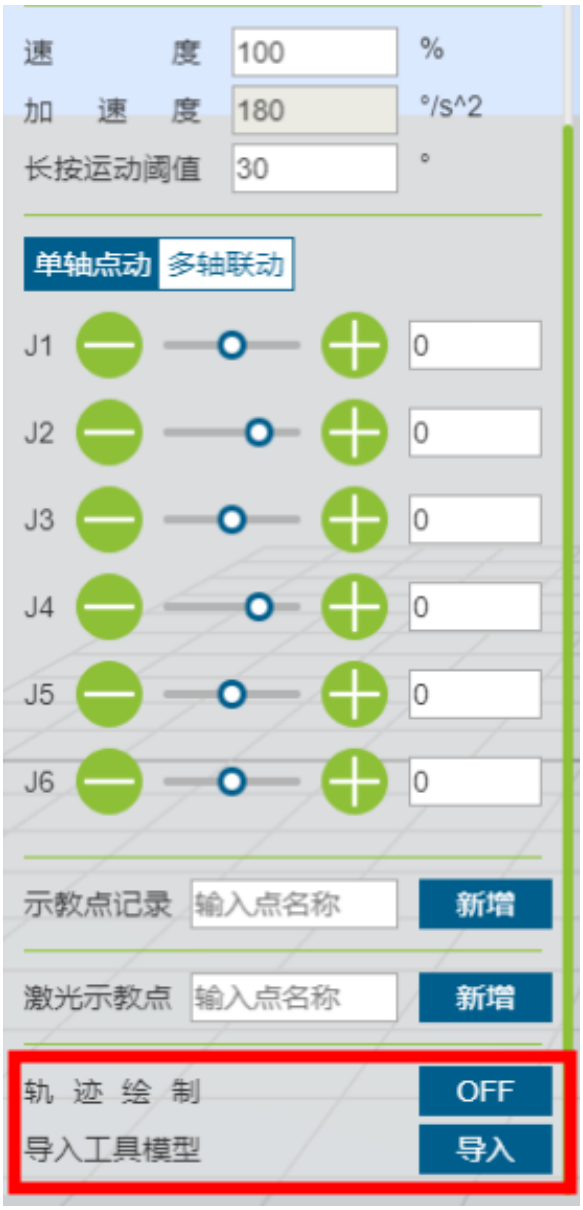
Joint、Base 等功能可参考*4.6 机器人操作*。

1.3.5.3 三维模拟机器人

1.3.5.3.1 三维虚拟轨迹和导入工具模型

轨迹绘制：运行示教程序时，打开轨迹绘制功能，机器人三维模型会描绘机器人运动的轨迹路线。

导入工具模型：点击“导入”按钮，导入工具模型后即可在机器人末端进行工具模型展示，目前支持的工具模型文件格式有 STL 和 DAE。

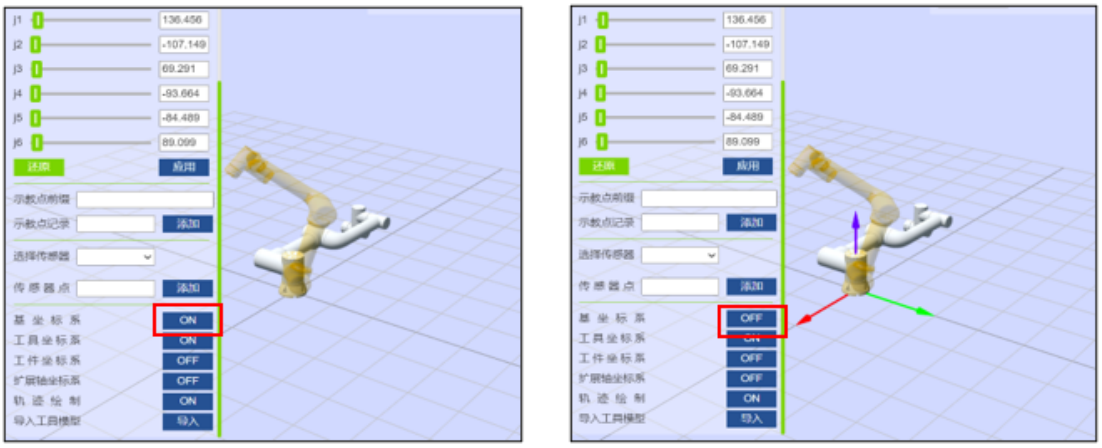


图表 3.3-1 虚拟轨迹绘制及工具模型导入

1.3.5.3.2 机器人坐标系系统三维可视化展示

在 WebAPP 机器人三维虚拟区域中创建各类三维虚拟坐标系，以基坐标系展示为例，如下图所示。其中 X 轴红色，Y 轴绿色，Z 轴蓝色。

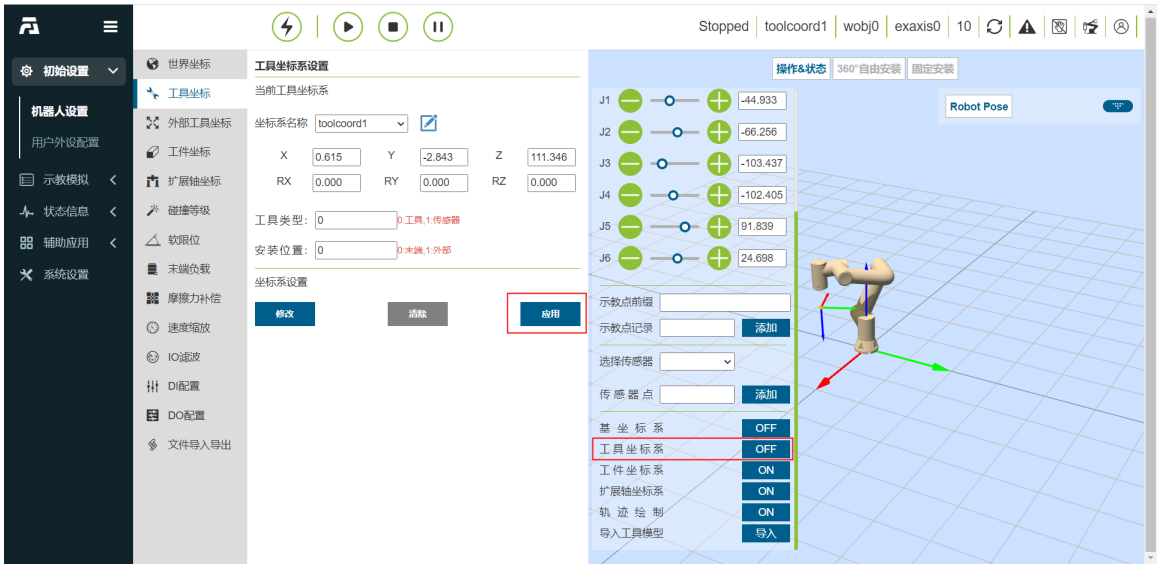
基坐标系：基坐标系 WebAPP 中系统机器人三维虚拟区域中进行默认开启展示，固定标记在机器人基座底部中心。三维虚拟基坐标系可进行手动关闭展示。



图表 3.3-2 基坐标系显示关闭与打开

工具坐标系：工具坐标系默认开启展示, 可手动关闭。在 WebAPP 启动并且用户登录成功后，获取当前应用的工具坐标系名称和对应参数数据，初始化当前工具坐标系。

使用的过程中应用其他工具坐标系时，当应用工具坐标系指令成功后，先将机器人三维虚拟区域中已有的工具坐标系清除，再将新应用的工具坐标系参数数据传入三维坐标系生成 API 进行工具坐标系生成，生成后完成在机器人三维虚拟区域中进行对应展示。



图表 3.3-3 工具坐标系显示

工件坐标系：工件坐标系默认关闭，可以进行手动开启展示。流程与工具坐标系一致。

图表 3.3-4 工件坐标系显示

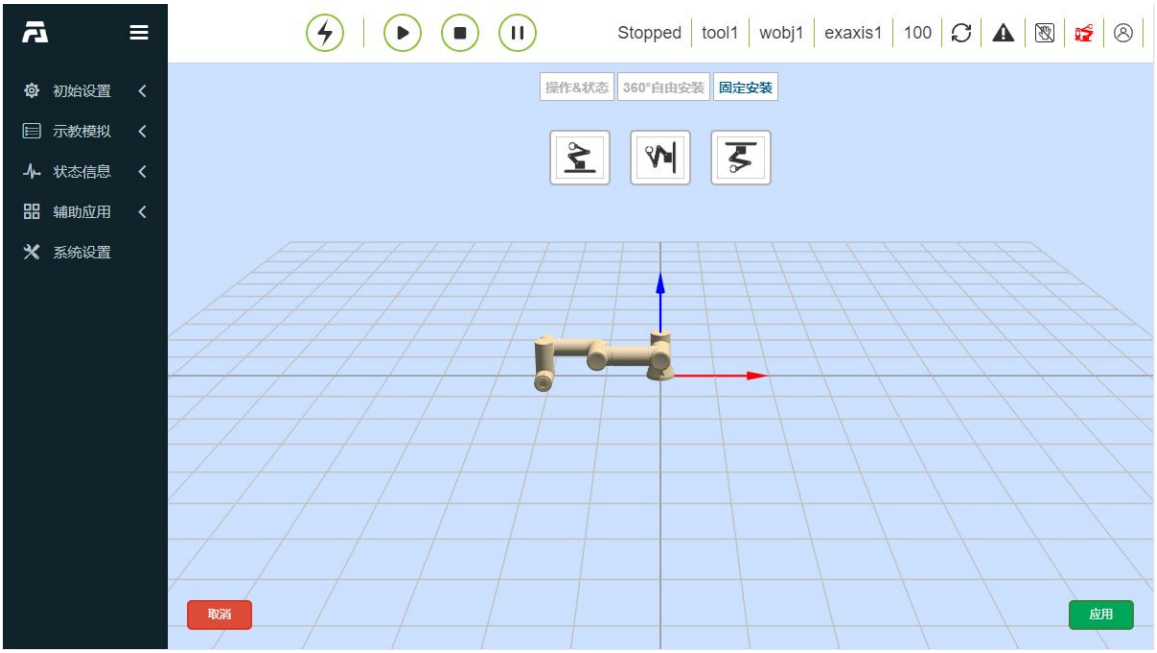
外部轴坐标系：外部轴坐标系默认关闭，可以进行手动开启展示。流程与工具坐标系一致。

图表 3.3-5 外部轴坐标系显示

1.3.5.3.3 机器人安装方式设置和展示

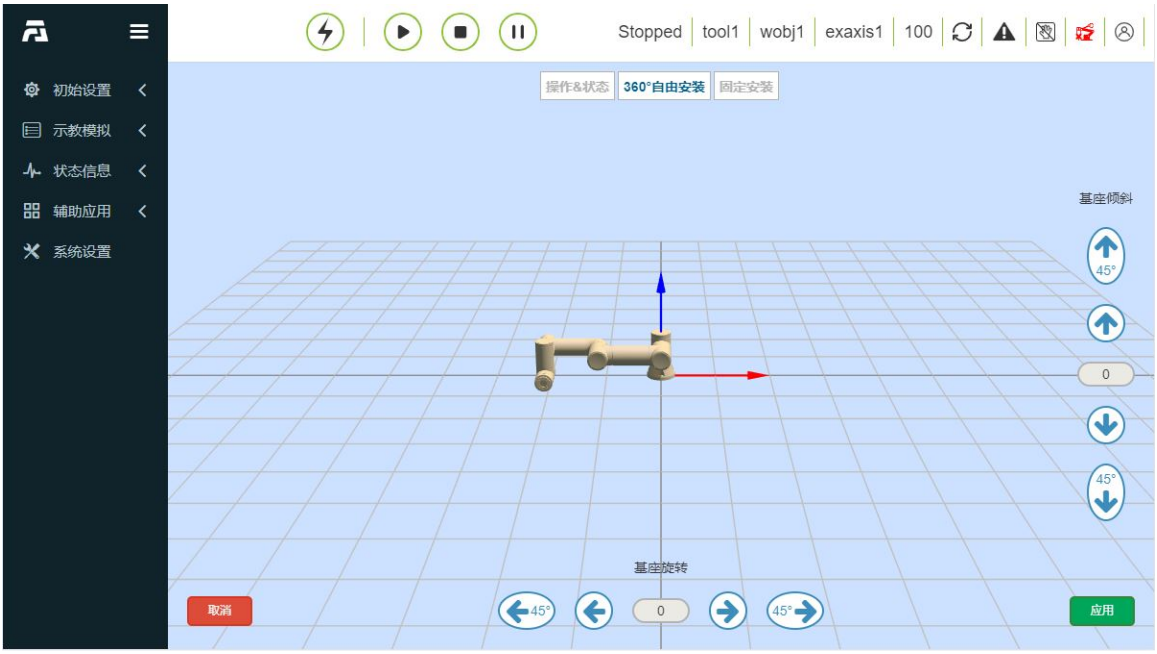
机器人默认安装方式为水平安装，当机器人安装方式更改时，需及时在此页面设置机器人的实际安装方式，以保证机器人正常工作。

用户点击机器人三维虚拟展示区域中的“固定安装”选项卡，进入机器人固定安装方式设置页面，选择“正装”、“倒装”或者“侧装”，点击点击“应用”按钮完成机器人安装方式设置。



图表 3.3-6 固定安装

考虑到更加灵活丰富的机器人部署场景，我们提供了自由安装功能，用户点击机器人三维虚拟展示区域中的“360 度自由安装”选项卡，进入机器人自由安装方式设置页面。手动调整“基座倾斜”和“基座旋转”角度，三维模型会对应展示安装效果。修改后点击“应用”按钮即可完成机器人安装方式设置。



图表 3.3-7 360 度自由安装

重要： 机器人安装完成后，必须正确设置机器人的安装方式，否则会影响机器人的拖动功能以及碰撞检测功能使用。

1.3.5.4 机器人设置

1.3.5.4.1 工具坐标

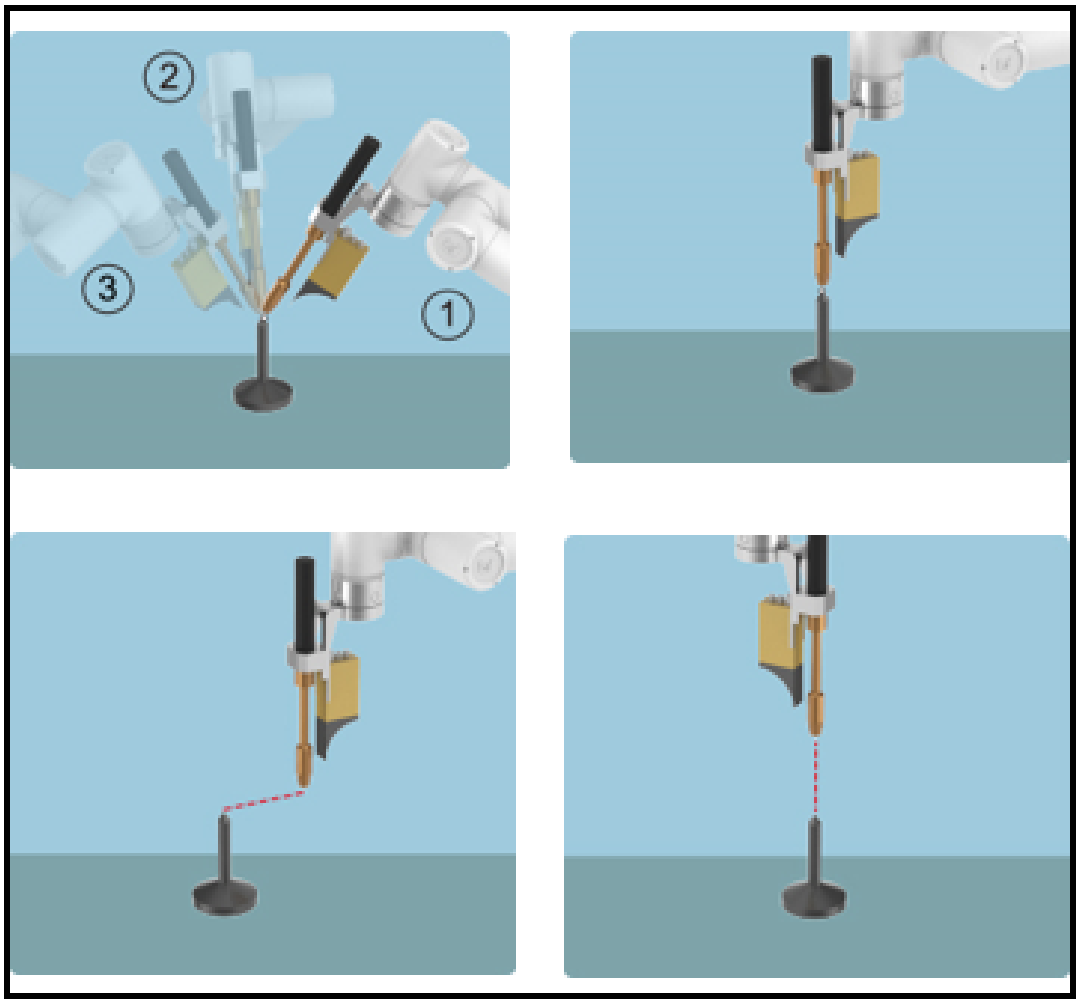
在“初始设置”中的“机器人设置”的菜单栏下，点击“工具坐标”进入工具坐标界面。工具坐标可实现工具坐标的修改、清空与应用。工具坐标系的下拉列表中共有 15 个编号，选择对应的坐标系（坐标系名称可自定义）后会在下方显示对应坐标值，工具类型以及安装位置（仅在传感器类型工具下显示），选择某一坐标系后点击“应用”按钮，当前使用的工具坐标系变为所选择的坐标，如图表 3.4-1 所示。

图表 3.4-1 设置工具坐标

点击“修改”可根据提示对该编号的工具坐标系进行重新设置。工具标定方法分为四点法和六点法，四点法只标定工具 TCP，即工具中心点的位置，其姿态默认与末端姿态一致，六点法则在四点法的基础上增加了两点，用于标定工具的姿态，这里我们以六点法为例进行讲解。

图表 3.4-2 设置工具坐标

在机器人空间选择一个固定的点，将工具以三个不同的姿态移至固定点，依次设置 1-3 点。如图 3.4-3 左上方所示。将工具垂直移至固定点设置点 4，如图 3.4-3 右上方所示。保持该姿态不变，利用基坐标移动，在水平方向移动一段距离，设定点 5，该方向即设定的工具坐标系 X 轴正方向。回到固定点，垂直往上移动一段距离，设定点 6，该方向即工具坐标系 Z 轴正方向，工具坐标系 Y 正方向则通过右手定则确定。点击计算按钮计算工具位姿，若需重新设置，点击取消按修改钮重新进行新建工具坐标系步骤。



图表 3.4-3 六点法示意图

完成最后步骤后，点击“完成”可返回工具坐标界面，点击“保存”即可存储刚才建立的工具坐标系。

重要:

1. 末端安装工具后，必须要进行工具坐标系的标定及应用，否则会导致机器人执行运动指令时工具中心点的位置和姿态不符合预期值。
2. 工具坐标系一般使用 toolcoord1~toolcoord14, 应用 toolcoord0 代表工具 TCP 的位置中心在末端法兰中心，在进行工具坐标系标定时，首先需将工具坐标系应用至 toolcoord0，然后选择其他工具坐标系进行标定及应用。

1.3.5.4.2 外部工具坐标

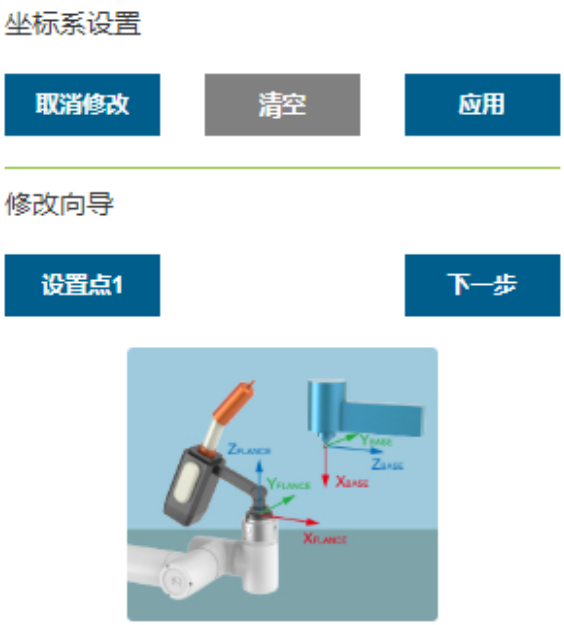
在“初始设置”中的“机器人设置”的菜单栏下，点击“外部工具坐标系”进入外部工具坐标系界面。

外部工具坐标系设置界面中可实现外部工具坐标的修改、清空与应用。

外部工具坐标系的下拉列表中共有 15 个编号，从 etoolcoord0~etoolcoord14，选择对应的坐标系后会在下方显示对应坐标值，选择某一坐标系后点击“应用”按钮，当前使用的工具坐标系变为所选择的坐标，如图 3.4-4 所示。

图表 3.4-4 外部工具坐标

点击“修改”可根据提示对该编号的工具坐标系进行重新设置，如图 3.4-5 所示。



图表 3.4-5 六点法示意图

1. 三点法确定外部 TCP

1. **设置点 1**：已测量工具的 TCP 移动至外部 TCP，点击设置点 1 按钮；
2. **设置点 2**：由点 1 沿外部 TCF 坐标系 X 轴移动一段距离，点击设置点 2 按钮；
3. **设置点 3**：回到点 1，由点 1 沿外部 TCF 坐标系 Z 轴移动一段距离，点击设置点 3 按钮；
4. **计算**：点击计算按钮得到外部 TCF；

2. 六点法确定工具 TCF

1. **设置点 1-4**：在机器人空间选择一个固定的点，将工具从四个不同的角度移至所选的点上，依次设置 1-4 点；
2. **设置点 5**：回到固定的点沿工具 TCF 坐标系 X 轴移动一段距离，点击设置点 5 按钮；
3. **设置点 6**：回到固定的点沿工具 TCF 坐标系 Y 轴移动一段距离，点击设置点 6 按钮；
4. **计算**：点击计算按钮得到工具 TCF；

若需重新设置，点击取消按钮重新进去新建工具坐标系步骤。

完成最后步骤后，点击“完成”可返回工具坐标界面，点击“保存”即可存储刚才建立的工具坐标系。

重要:

1. 使用外部工具必须要进行外部工具坐标系的标定及应用，否则会导致机器人执行运动指令时工具中心点的位置和姿态不符合预期值。
2. 外部工具坐标系一般使用 etoolcoord1~etoolcoord14, 应用 etoolcoord0 代表外部工具 TCP 的中心位置在末端法兰中心，在进行工具坐标系标定时，首先需将工具坐标系应用至 etoolcoord0，然后选择其他工具坐

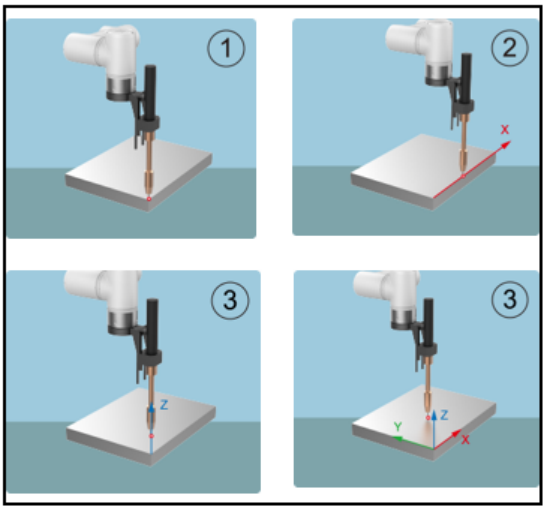
标系进行标定。

1.3.5.4.3 工件坐标

在“初始设置”中的“机器人设置”的菜单栏下，点击“工件坐标”进入工件坐标界面。工件坐标可实现工件坐标的修改、清空与应用。工件坐标系的下拉列表中共有 15 个编号，选择对应的坐标系（wobjcoord0~wobjcoord14），后会在下方的“坐标系坐标”中显示对应坐标值，选择某一坐标系后点击“应用”按钮，当前使用的工件坐标系变为所选择的坐标，如图表 3.4-6 所示。

图表 3.4-6 设置工件坐标

工件坐标系一般是基于工具基础上进行标定的，需要在已建立工具坐标系的基础上进行工件坐标系的建立。点击“修改”可根据提示对该编号的工件坐标系进行重新设置。固定好工件，选择标定方法“原点-X 轴-Z 轴”或“原点-X 轴-XY+ 平面”，两种标定方法前两点的选取都是一致的，第三点有所区别，选第一种方法标定的是工件坐标系的 Z 方向，选第二种方法标定的是 XY+ 平面上一点，根据图示标定即可。点击计算按钮计算工件位姿，若需重新设置，点击取消按修改钮重新进行新建工件坐标系步骤。



图表 3.4-7 三点法示意图

完成最后步骤后，点击“完成”可返回工件坐标界面，点击“保存”即可存储刚才建立的工件坐标系。

重要:

1. 工件坐标系是基于工具基础上进行标定的，需要在已建立工具坐标系的基础上进行工件坐标系的建立。
2. 工件坐标系一般使用 wobjcoord1~wobjcoord14, 应用 wobjcoord0 代表工件坐标系原点在基坐标原点，在进行工件坐标系标定时，首先需将工件坐标系应用至 wobjcoord0，然后选择其他工件坐标系进行标定及应用。

1.3.5.4.4 扩展轴坐标

在“初始设置”中的“机器人设置”的菜单栏下，点击“扩展轴坐标系”进入扩展轴坐标系界面。扩展轴坐标系设置界面中可实现扩展轴坐标的修改、清空与应用。

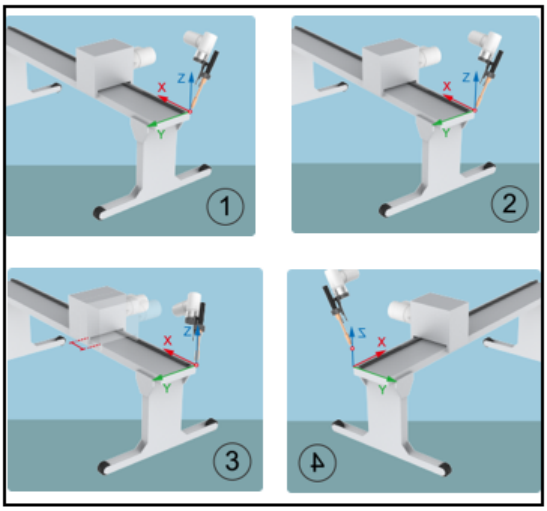
扩展轴坐标系的下拉列表中共有 5 个编号，从 eaxis0~eaxis4，选择对应的坐标系后会在下方显示对应坐标值，选择某一坐标系后点击“应用”按钮，当前使用的扩展轴坐标系变为所选择的坐标，如图 3.4-8 所示。

图表 3.4-8 扩展轴坐标

点击“修改”可根据提示对该编号的扩展轴标系进行重新设置，如图 3.4-9 所示。标定之前先清空需要标定的扩展轴坐标系，应用此扩展轴坐标系。先看第一种扩展轴方案-直线导轨的标定方法。选择扩展轴的编号，获取信息可以获取对应扩展轴的驱动器信息，我们可以根据该信息进行参数配置。配置完后设置 DH 参数，直线导轨方案默认为 0。设置机器人相对扩展轴位置，直线导轨为扩展轴上。若不标定，点击保存即可，此时扩展轴只能异步运动。

图表 3.4-9 直线导轨配置

若需跟机器人同步运动，点击标定，进入标定界面，在扩展轴零点处，点击操作区 Eaxis 使能扩展轴，将机器人末端中心（应用工具坐标系下用工具末端点）以两个不同姿势对准扩展轴上固定一点，分别设定点 1 和点 2。去除使能，将扩展轴移动一段距离，使能后，同样将机器人末端中心点对准之前固定点，设定点 3。去除使能，将扩展轴移至零点，使能扩展轴。将机器人末端中心点移至固定点垂直往上空间一点，设定点 4，计算坐标系并保存。

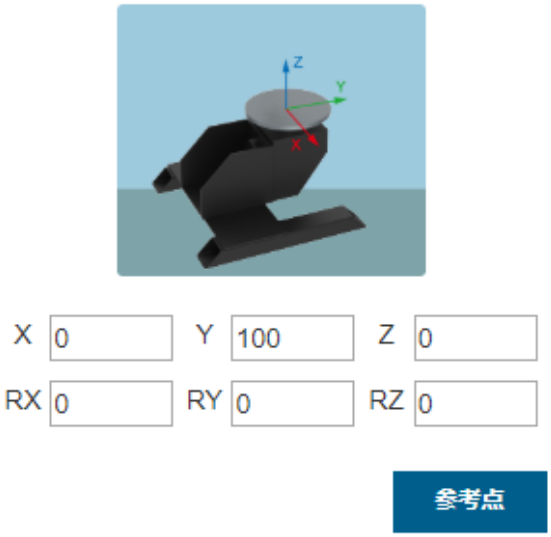


图表 3.4-10 直线导轨标定

接下来看第二种扩展轴方案-变位机的标定方法。变位机由两个扩展轴组成，选择扩展轴的编号，获取信息可以获取对应扩展轴的驱动器信息，我们可以根据该信息进行参数配置。配置完后设置 DH 参数，根据图示测量出变位机的 DH 参数，输入到输入框中。设置机器人相对扩展轴位置，变位机为扩展轴外。若不标定，点击保存即可，此时扩展轴只能异步运动。

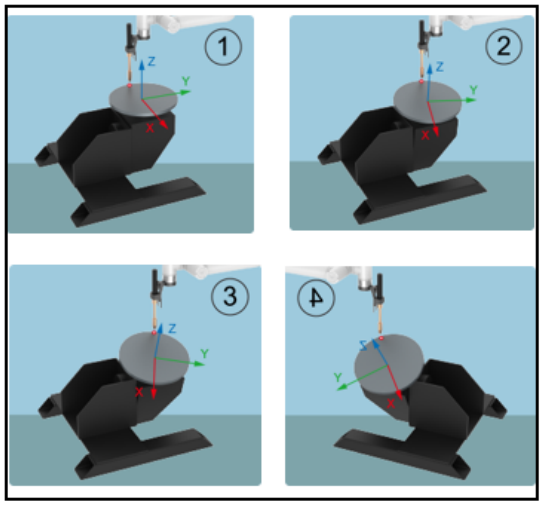
图表 3.4-11 变位机配置

若需跟机器人同步运动，点击标定，进入标定界面，在扩展轴零点处，点击操作区 Eaxis 使能扩展轴，在变位机上建立坐标系，选择一点，输入该点在该坐标系下的笛卡尔位姿，比如选择 Y 正向一点，测出 Y 为 100mm，则输入如图所示数值，点击参考点，即可设定参考点。后续四个标定点都需将机器人末端中心（应用工具坐标系下用工具末端点）对准该参考点。



图表 3.4-12 变位机参考点配置

将机器人末端中心（应用工具坐标系下用工具末端点）对准该参考点，设定点 1，点击操作区 Eaxis 点动两个轴一小段距离，将机器人末端中心对准参考点，设定点 2，继续点动两个轴，机器人末端中心对准参考点，设定点 3，最后继续点动两个轴，将机器人末端中心对准参考点，设定点 4，点击计算，得到坐标系结果，点击保存，应用即可。



图表 3.4-13 变位机标定

接下来看第三种扩展轴方案-单轴变位机的标定方法。该变位机由一个旋转扩展轴组成，选择扩展轴的编号，获取信息可以获取对应扩展轴的驱动器信息，我们可以根据该信息进行参数配置。DH 参数设置为 0。设置机器人相对扩展轴位置，变位机为扩展轴外。若不标定，点击保存即可，此时扩展轴只能异步运动。

图表 3.4-14 单轴变位机配置

若需跟机器人同步运动，点击标定，进入标定界面，在扩展轴零点处，点击操作区 Eaxis 使能扩展轴，在变位机上建立坐标系，选择一点，输入该点在该坐标系下的笛卡尔位姿，点击“参考点”，即可设定参考点。后续四个标定点都需将机器人末端中心（应用工具坐标系下用工具末端点）对准该参考点。将机器人末端中心（应用工具坐标系下用工具末端点）对准该参考点，设定点 1，点击操作区 Eaxis 点动旋转轴一小段距离，将机器人末端中心对准参考点，设定点 2，继续点动旋转轴，机器人末端中心对准参考点，设定点 3，最后继续点动旋转轴，将机器人末端中心对准参考点，设定点 4，点击计算，得到坐标系结果，点击保存，应用即可。

图表 3.4-15 单轴变位机标定

重要:

1. 扩展轴坐标系是基于工具基础上进行标定的，需要在已建立工具坐标系的基础上进行扩展轴坐标系的建立。
2. 扩展轴系一般使用 exaxis1~ exaxis4, 应用 exaxis0 代表无扩展轴坐标系，在进行扩展轴坐标系标定时，首先需将扩展轴坐标系应用至 exaxis0，然后选择其他扩展轴坐标系进行标定及应用。

1.3.5.4.5 碰撞等级

在“初始设置”中的“机器人设置”的菜单栏下，点击“碰撞等级”进入碰撞等级界面。

碰撞等级分为一到十级，一到三级检测比较灵敏，机器人需要在推荐速度下运行。同时可以选择自定义百分比设置，100% 即对应十级。碰撞策略可以设置机器人碰撞后的处理方式，分为报错停止和继续运动，用户可以根据具体使用需求来设定。如图表 3.4-16。

图表 3.4-16 碰撞等级示意图

1.3.5.4.6 软限位

在“初始设置”中的“机器人设置”的菜单栏下，点击“软限位”进入软限位界面。

机器人行程内可能存在其它设备，限位角度可对机器人进行软限位，使机器人运动不超过某个坐标值，防止机器人碰撞。触发软限位机器人停止为机器人自动触发，无停止距离。

管理员可使用默认值也可输入角度值。输入角度值，可分别对机器人关节正负角度进行限位，当输入值超出表 1.4-1 机器人基本参数节所列出的机器人关节软限位角度值，会将限位角度调整为所能设定最大值。当机器人报出超出指令超限时，需要进入拖动模式，将机器人关节拖动至限位角度之内。界面如 3.4-17 所示。

图表 3.4-17 机器人限位示意图

1.3.5.4.7 末端负载

在“初始设置”中的“机器人设置”的菜单栏下，点击“末端负载”进入末端负载界面。

在配置末端负载时请将所使用的末端工具的质量以及对应的质心坐标分别输入“负载质量”和“负载质心坐标 X、Y 和 Z”输入框中并应用。

重要： 负载质量不可超过机器人的最大负载范围。具体机器人型号对应的负载范围请参考 2.1. 基本参数, 质心坐标设置范围为 0-1000，单位 mm。

图表 3.4-18 负载设定示意图

重要：机器人末端安装负载后，必须正确设置末端负载重量以及质心坐标，否则会影响机器人的拖动功能以及碰撞检测功能使用。

用户对工具质量或质心不确定的情况下，可以通过点击“自动辨识”进入负载辨识功能对工具数据测定。

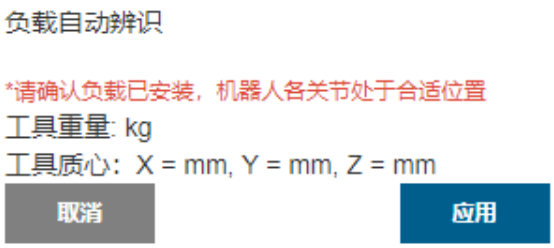
在进行测定之前，确保负载已安装后选择版本。点击“工具数据测定”按键，进入负载运动测试界面。

图表 3.4-19 负载辨识关节设置

点击“负载辨识启动”进行测试，如遇紧急情况请及时停止运动。

图表 3.4-20 负载辨识启动

运动结束后，点击“获取辨识结果”按键，获取计算出的工具数据，并显示在页面上，如需应用到负载数据中，点击应用即可



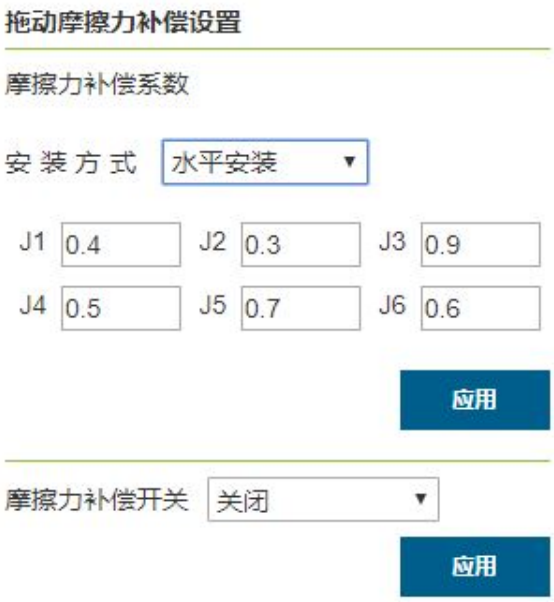
图表 3.4-21 负载辨识结果

1.3.5.4.8 摩擦力补偿

在“初始设置”中的“机器人设置”的菜单栏下，点击“摩擦力补偿”进入摩擦力补偿设置界面。

摩擦力补偿系数：摩擦力补偿所针对的使用场景仅在拖动模式下，摩擦力补偿系数可设置范围为 0~1，数值越高，拖动时补偿的力就越大。摩擦力补偿系数根据安装方式的不同需要单独设置每个轴的补偿系数。

摩擦力补偿开关：用户可根据实际机器人及使用习惯开启或关闭摩擦力补偿。



图表 3.4-22 摩擦力补偿设置

重要： 机器人摩擦力补偿功能需要谨慎使用，根据实际情况，设置合理的补偿系数，一般推荐中间值 0.5 左右。

1.3.5.4.9 速度缩放

在“初始设置”中的“机器人设置”的菜单栏下，点击“速度缩放设置”进入速度缩放设置界面。

该功能是设置手动/自动下机器人运行的速度，若当前为自动运行模式，则设置的速度为机器人自动运行速度，若当前为手动运行模式，则设置的速度为机器人手动运行速度。设置数值为机器人标准速度百分比，若设置 100，即标准速度的百分之百（标准速度请翻阅表格 1.4-1 机器人基本参数）。

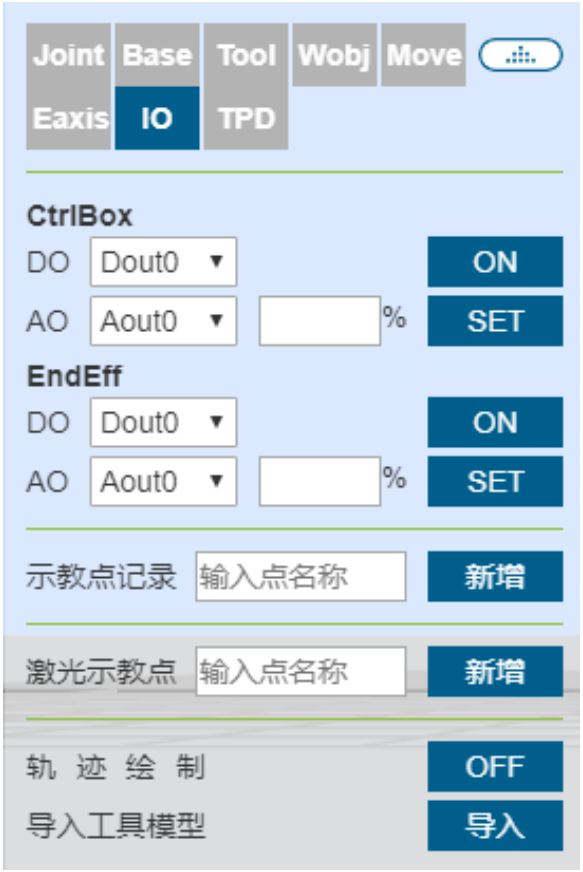
图表 3.4-23 速度缩放设置

速度设置成功后，相应的速度状态栏会更改为设置的数值，速度值设置的范围是 0~100。

1.3.5.5 控制箱 I/O

1.3.5.5.1 I/O 设置

点击三维模型左侧操作区“IO”按钮可进入 IO 设置界面，如图表 3.5-1 所示，该界面中可实现对机器人控制箱中数字输出、模拟输出（0-10v）和末端工具数字输出、模拟输出（0-10v）进行手动控制：

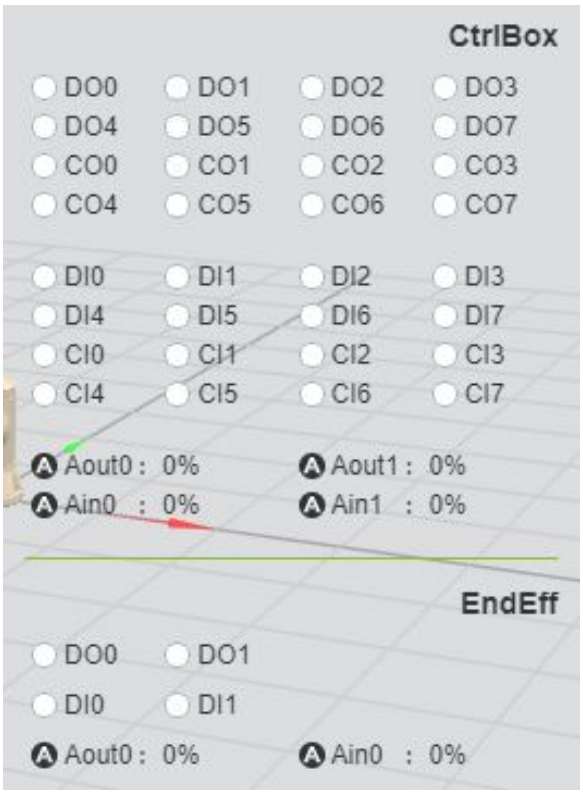


图表 3.5-1 I/O 设置界面

- DO 操作：选择端口号，若该 DO 为低电平，则右侧操作按键显示 ON，点击按键即设置该 DO 为高电平。
- AO 操作：选择端口号，右侧输入框输入值（0-100），该数值为百分比，设置 100 即表示设置该 AO 端口为 10v。

1.3.5.5.2 I/O 状态显示

三维模型右侧状态显示区会显示当前 IO 的状态，数字输入与数字输出中，若该端口电平为高，则该点显示为绿色，若为低，则显示为白色；模拟输入和模拟输出显示值为 0-100，100 即表示 10v。



图表 3.5-2 状态显示界面

1.3.5.5.3 I/O 滤波

点击左侧菜单栏“初始设置”中“机器人设置”，点击“IO 滤波”子菜单进入 IO 滤波时间设置界面，滤波时间设置界面包括：控制箱 DI 滤波时间，末端板 DI 滤波时间、控制箱 AI0 滤波时间、控制箱 AI1 滤波时间、末端板 AI0 滤波时间，如图表 3.5-3 所示。用户可以根据自己的需求来设定对应的参数，点击相应的设置按钮即可。

图表 3.5-3 滤波界面

重要： I/O 滤波时间范围为 [0~200]，单位 ms。

1.3.5.5.4 I/O 配置

点击左侧菜单栏“初始设置”中“机器人设置”，分别点击“DI 配置”和“DO 配置”子菜单进入 DI 和 DO 配置界面。其中控制箱 CI0-CI7 和 CO0-CO7 可配置，末端 DI0 和 DI1 可配置。在生产中协作机器人需要连接外设时或因故障或者其它因素突然停止，需要输出 DO 信号，实现声光报警提示，输入可配置功能如表格 3.5-1 控制箱输入可配置功能所示。

功能编号	功能名称
0	无
1	起弧成功信号
2	焊机准备信号
3	传送带检测
4	暂停
5	恢复
6	启动
7	停止
8	暂停/恢复
9	启动/停止
10	脚踏拖动开关
11	移至作业原点
12	手自动切换
13	焊丝寻位成功
14	运动中断
15	启动主程序
16	启动倒带
17	启动确认
18	激光检测信号 X
19	激光检测信号 Y

表格 3.5-1 控制箱输入可配置功能

输出可配置功能如表 3.5-2 和表 3.5-3 所示。

功能编号	功能名称
0	无
1	报错
2	运动
3	喷涂启停
4	喷涂清枪
5	起弧
6	送气
7	正向送丝
8	反向送丝
9	JOB 输入口 1
10	JOB 输入口 2
11	JOB 输入口 3
12	传送带启停
13	暂停
14	到达作业原点
15	进入干涉区
16	焊丝寻位启停控制
17	机器人启动完成
18	程序启动停止
19	自动手动模式

表格 3.5-2 控制箱输出可配置功能

功能编号	功能名称
0	无
1	拖动模式
2	示教点记录
3	手自动切换
4	TPD 轨迹记录启动/停止
5	暂停
6	恢复
7	启动
8	停止
9	暂停/恢复
10	启动/停止

表格 3.5-3 末端输入可配置功能

其中控制箱默认配置：CO0 为 1-机器人报错，CO1 为 2-机器人运动中。

图表 3.5-4 控制箱 DI 和 DO 配置

末端 DI 默认配置：DI0 拖动示教，DI1 示教点记录。

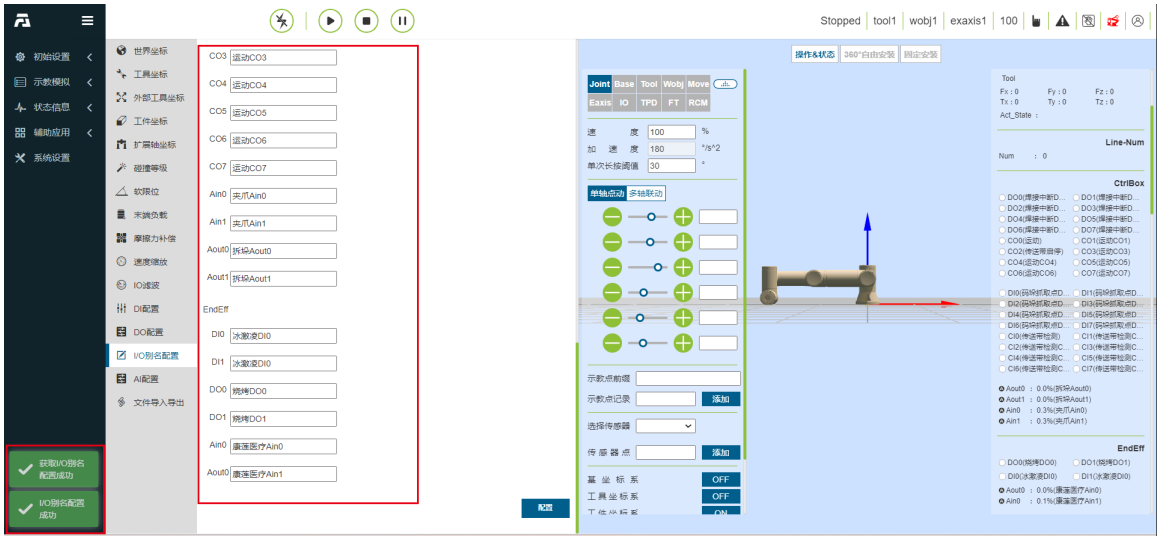
图表 3.5-5 末端 DI 配置

配置完成后，可在对应状态下，于控制箱 I/O 页面中查看相应的输出 DO 状态。（注意：已配置 DI、DO 不能出现于示教编程页面）

重要： 已配置 DI、DO 禁止在示教编程中使用。

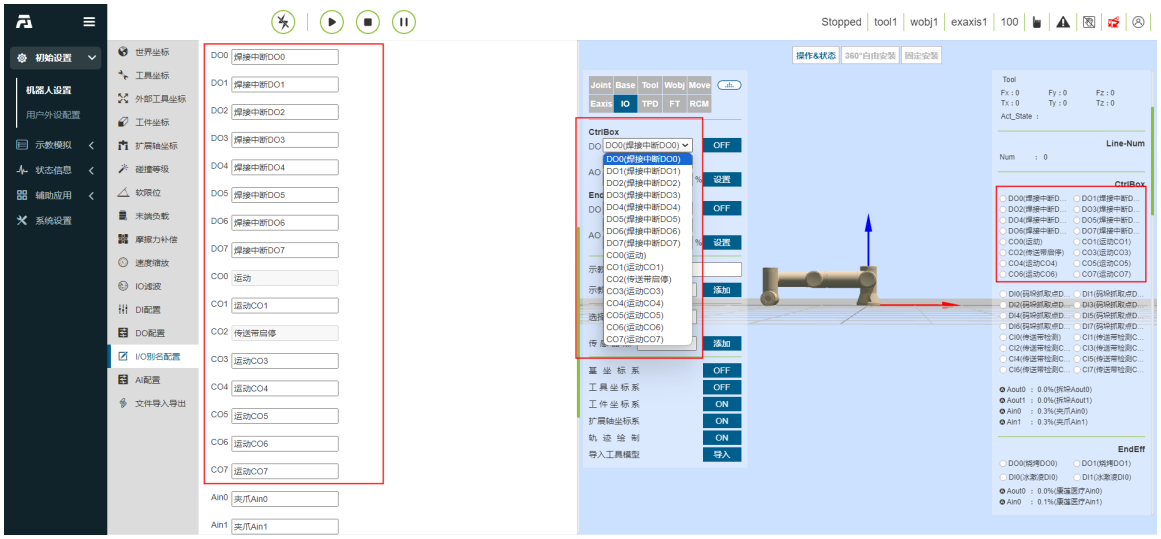
1.3.5.5.5 I/O 别名配置

点击左侧菜单栏“初始设置”中“机器人设置”，点击“I/O 别名配置”子菜单进入配置界面，根据实际使用场景配置控制箱和末端 IO 信号的给定含义名称。配置成功后，有关 IO 信号内容的模块显示对应别名，模块如下：

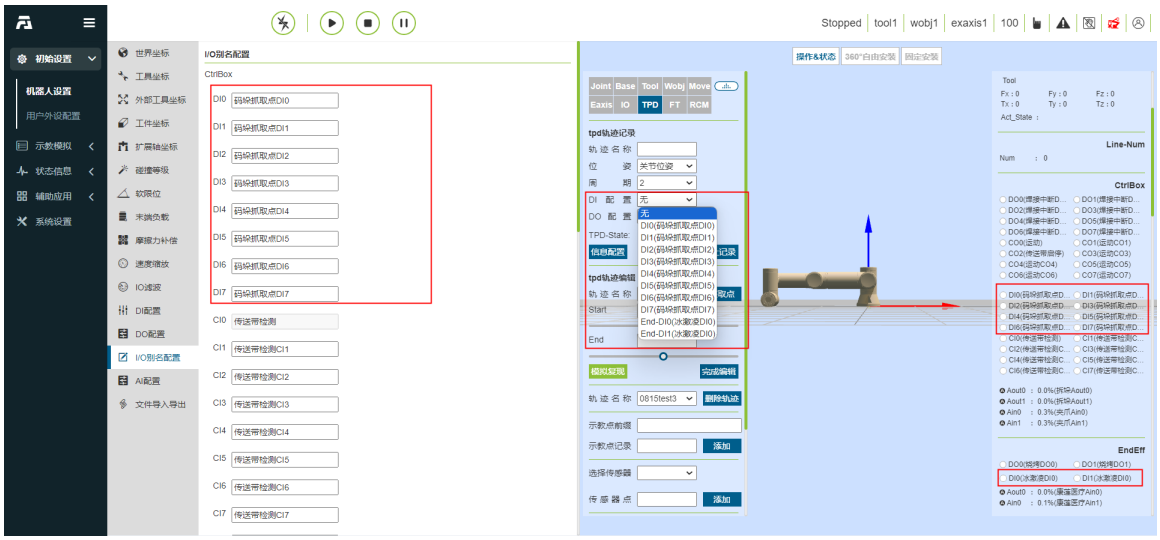


图表 3.5-6 IO 别名配置

机器人操作：IO 和 TPD 控制箱（CtrlBox）和末端（End）的 IO 信号选择框；

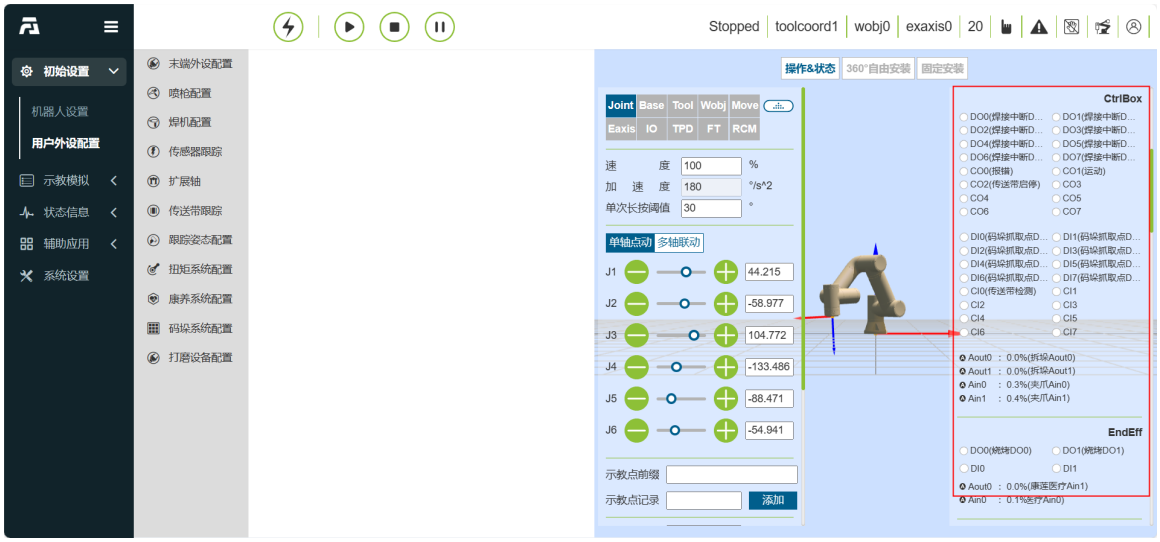


图表 3.5-7 机器人操作 IO



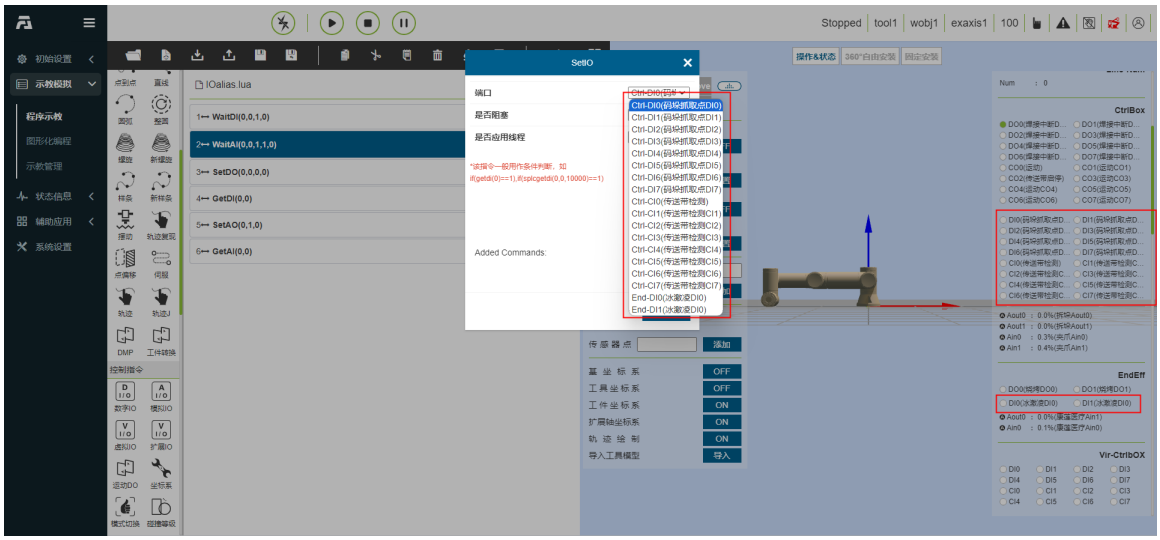
图表 3.5-8 机器人操作 TPD

机器人状态：Robot 的控制箱（CtrlBox）和末端（End）的 IO 信号状态；

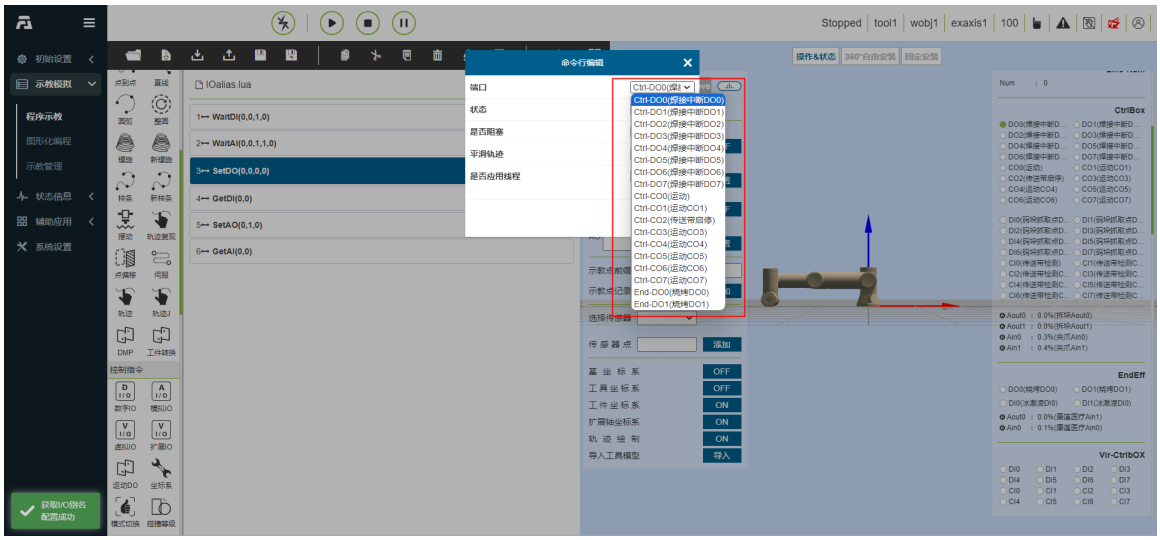


图表 3.5-9 机器人状态 IO

示教模拟——程序示教：“等待”、“数字 IO”和“模拟 IO”添加程序命令，以及选中程序命令行点击“编辑”图标，弹出框内容中 IO 选择框；

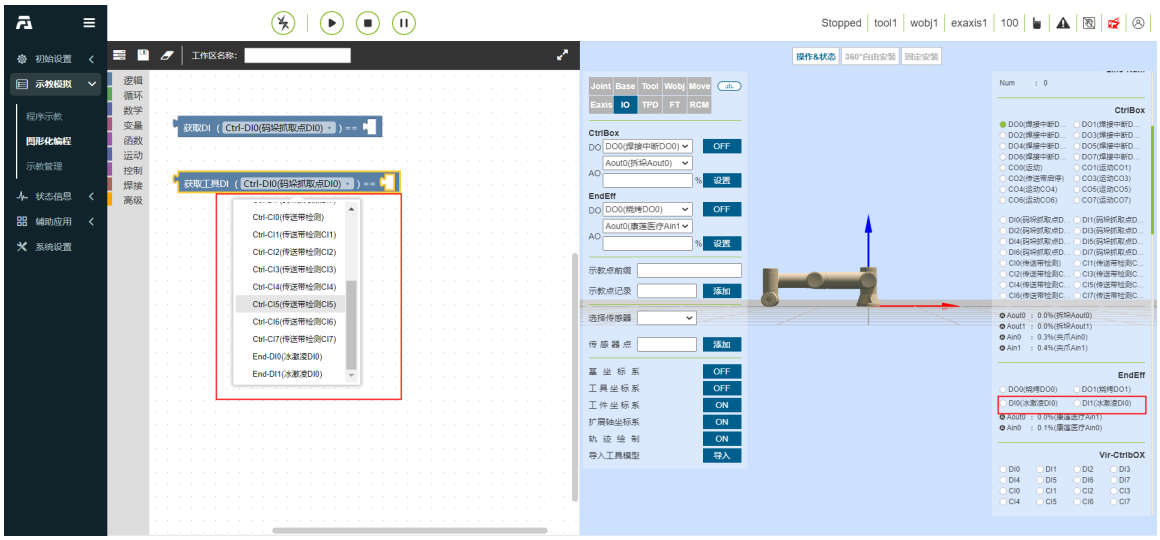


图表 3.5-10 程序示教添加 IO



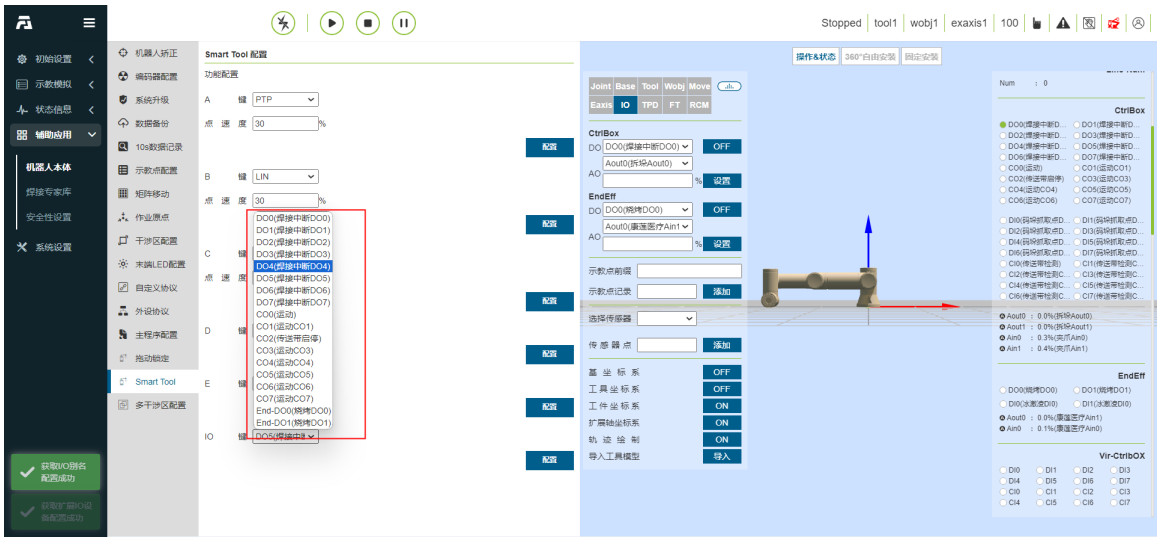
图表 3.5-11 程序示教编辑 IO

示教模拟——图形化编程：“逻辑”和“控制”模块的 IO 下拉选择项；



图表 3.5-12 图形化编程 IO

辅助应用——机器人本体：“smart Tool”中的 IO 键选择框；



图表 3.5-13 smart Tool IO

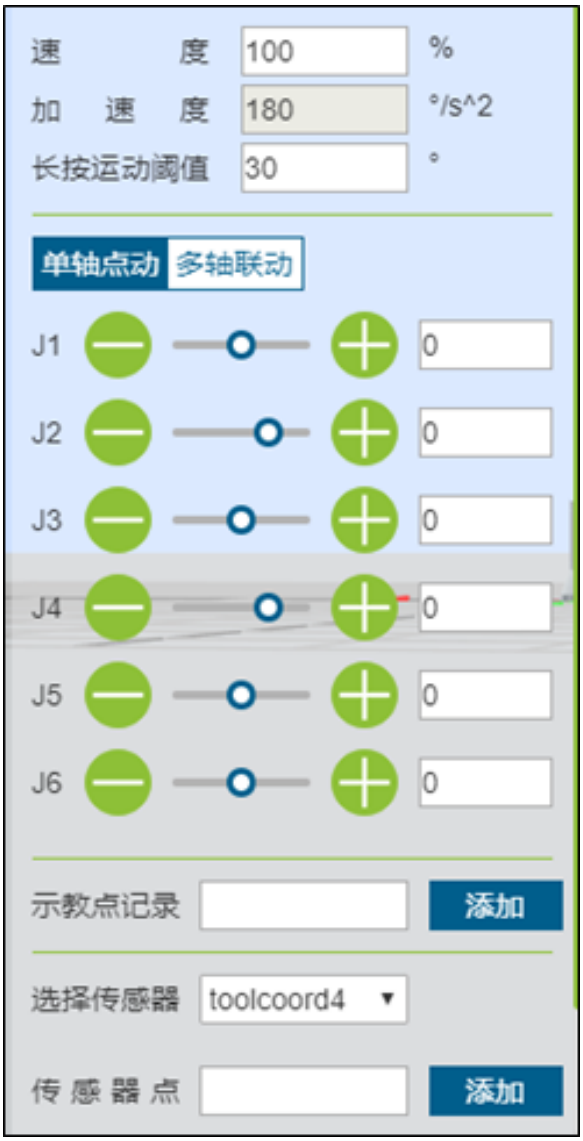
1.3.5.6 机器人操作

1.3.5.6.1 示教点记录

手动示教控制区主要是在示教模式中对考坐标系进行设定，并实时显示机器人各轴角度与坐标值，并可对示教点进行命名保存。

保存示教点时，该示教点的坐标系为当前机器人应用的坐标系。在该操作区上方可以对示教点速度，加速度设置，设置数值为机器人标准速度百分比，若设置 100，即标准速度的百分之百（标准速度请翻阅表 1.4-1 机器人基本参数）。

传感器示教点，选择已经标定的传感器类型工具，输入点名称，点击添加，保存的点的位置为传感器识别到点的位置。



图表 3.6-1 手动操作区示意图

重要： 第一次使用时，请设置 30 这样较小的速度值，熟悉机器人运动，以免发生意外情况。

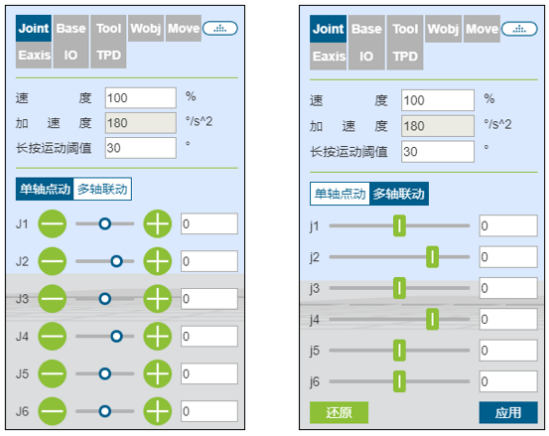
1.3.5.6.2 Joint 运动

Joint 运作下，中间的 6 个滑块条分别表示对应轴的角度，joint 运动分单轴点动和多轴联动

单轴点动：用户可通过操作左右两边圆形按钮来控制机器人运动，如图表 3.6-2。在手动模式和关节坐标系下，对机器人某一关节进行转动操作。当机器人超出运动范围（软限位）而停止时，可以利用单轴点动进行手动操作，将机器人移出超限位置。单轴点动在进行粗略定位和较大幅度移动时，会比其他操作模式更快捷方便。

设置“长按运动阈值”（长按按钮时，机器人运行的最大距离，输入值得范围 0~300）参数，长按圆形按钮控制机器人运行，若在机器人运行中松开按钮，机器人会立即停止运动，若一直按住不松开按钮，机器人会运行长按运动阈值所设置的值后停止运动。

多轴联动：用户可操作中间六个滑块来调整机器人相应的目标位置，如图表 3.6-3，可通过观察三维虚拟机器人来确定目标位置，若调整的位置不符合自己的预期，点击“还原”按钮，使得三维虚拟机器人回到初始的位置。当用户确定目标位置后，可点击“应用”按钮，实体机器人便会进行相应的运动。

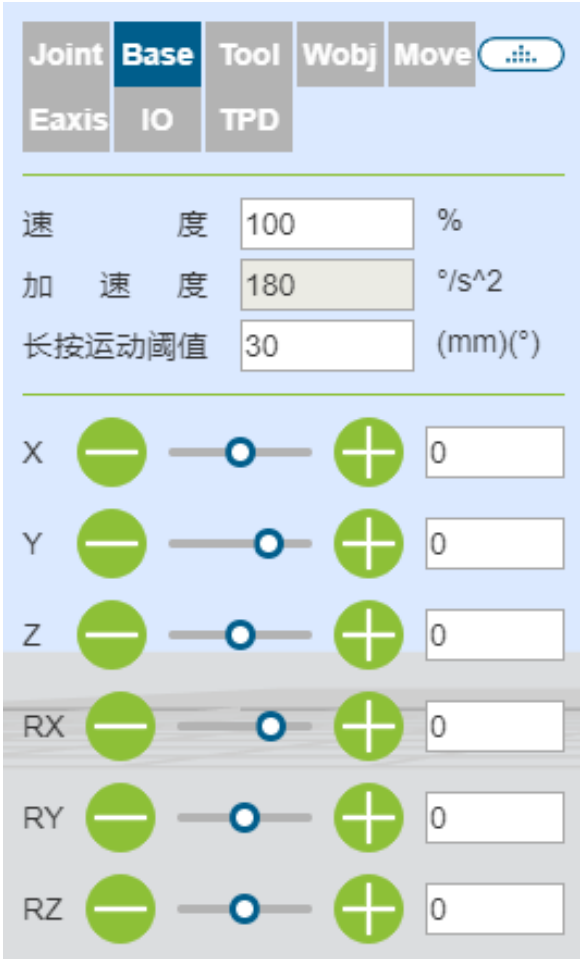


图表 3.6-2 单轴点动和多轴联动示意图

重要： 多轴联动中，第 5 个关节 j5 的设置值不能小于 0.01 度，若期望值小于 0.01 度，则可以先设置为 0.011 度，然后通过单轴点动微调第 5 个关节 j5。

1.3.5.6.3 Base 点动

在基坐标系下，可以操作左右两边圆形按钮控制机器人，在 X，Y，Z 轴上直线移动或绕着 RX，RY，RZ 旋转，中间的 6 个滑块条分别表示在对应坐标轴上的位置与运动范围，如图表 3.6-3。Base 点动的功能与 Joint 运动中单轴点动的功能相似。

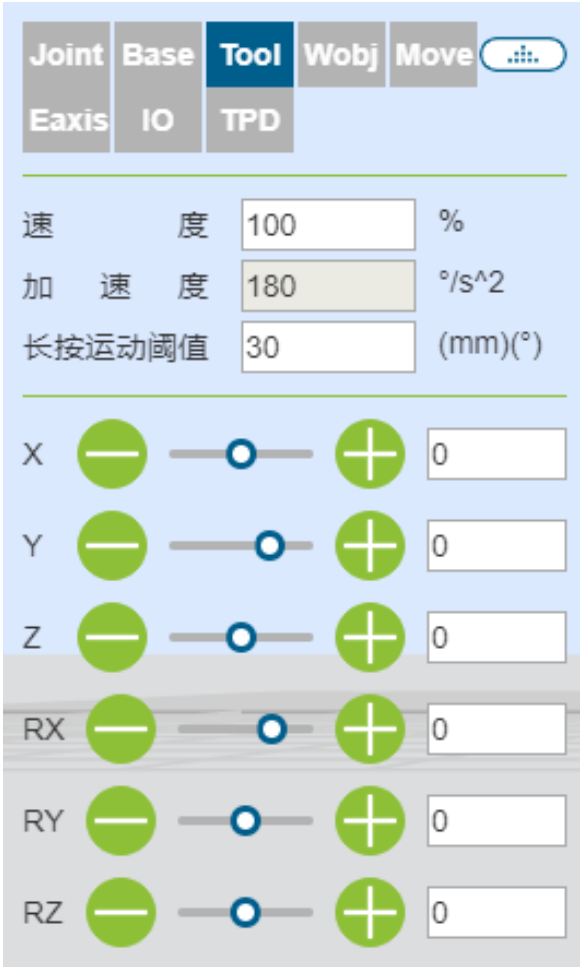


图表 3.6-3 Base 点动示意图

重要： 可随时释放该按钮，使机器人停止运动。在必要情况下，按急停按钮使机器人停止。

1.3.5.6.4 Tool 点动

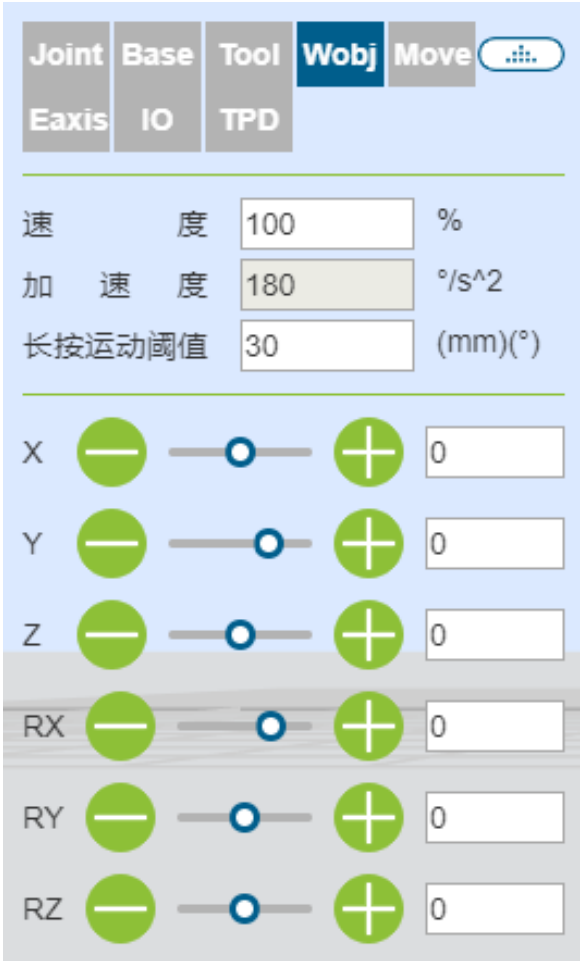
选择工具坐标系，可以操作左右两边圆形按钮控制机器人，在 X，Y，Z 轴上直线移动或绕着 RX，RY，RZ 旋转，中间的 6 个滑块条分别表示在对应坐标轴上的位置与运动范围，如图表 3.6-4。Tool 点动的功能与 Joint 运动中单轴点动的功能相似。



图表 3.6-4 Tool 点动示意图

1.3.5.6.5 Wobj 点动

选择工件点动，可以操作左右两边圆形按钮控制机器人，在工件坐标系下，沿着 X，Y，Z 轴上直线移动或绕着 RX，RY，RZ 旋转，中间的 6 个滑块条分别表示在对应坐标轴上的位置与运动范围，如图表 3.6-5。Wobj 点动的功能与 Joint 运动中单轴点动的功能相似。



图表 3.6-5 Wobj 点动示意图

1.3.5.6.6 Move 移动

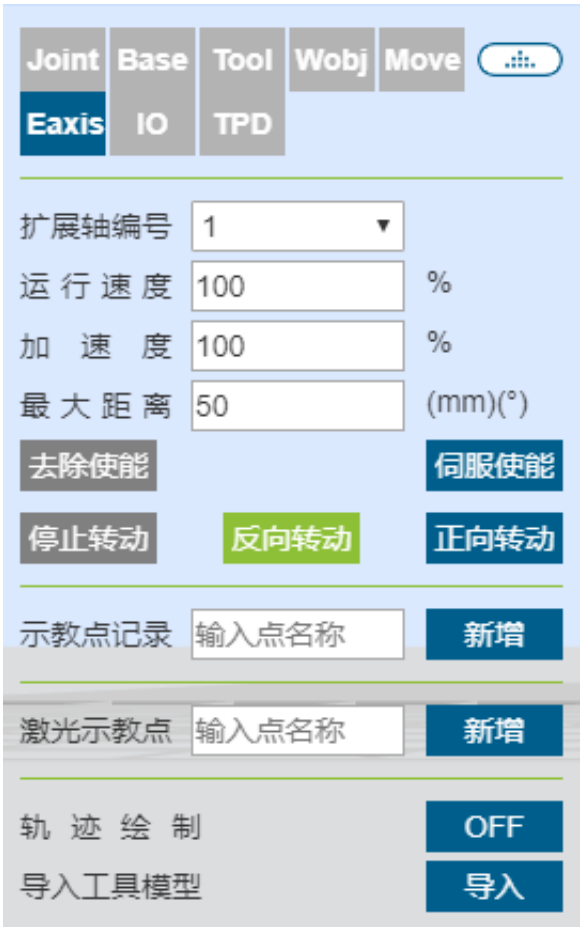
选择 Move 移动，可以直接输入笛卡尔坐标值，点击“计算关节位置”，关节位置显示为计算后结果，确认无危险，可以点击“移至该点”控制机器人运动至输入的笛卡尔位姿。

图表 3.6-6 Move 移动示意图

重要： 当出现给定位姿无法到达时，首先检查笛卡尔空间位姿是否超过机器人工作范围，然后检查当前位姿到目标位姿过程中是否存在奇异位姿，若存在奇异位置则调整下当前姿态或过程中插入一个新的位姿以避开奇异位姿。

1.3.5.6.7 Eaxis 移动

选择 Eaxis 移动，该功能为扩展轴的点动功能，需要在配置好扩展轴的前提下，使用该点动功能控制扩展轴，详见“第四章机-器人外设-扩展轴外设配置”。



图表 3.6-7 Eaxis 移动示意图

1.3.5.6.8 TPD（示教编程）

示教编程（TPD）功能操作步骤如下：

- **Step1 记录初始位置**：进入三维模型左侧操作区，记录机器人当前位置。在编辑框内设定好点的名称，点击“保存”按钮，若保存成功，则提示“保存点成功”；
- **Step2 配置轨迹记录参数**：点击 TPD 进入“TPD”功能项配置轨迹记录参数，设定好轨迹文件的名称、位姿类型以及采样周期，配置 DI 和 DO，可以在记录 TPD 轨迹的过程中，通过触发 DI 来记录对应需要输出的 DO，如图表 3.6-8；

图表 3.6-8 TPD 轨迹记录

- **Step3 检查机器人模式**：检查机器人模式是否处于手动模式下，若不处于则切换至手动模式，在手动模式下可通过两种方式切换到托动示教模式，一种是长按末端按钮，一种是界面拖动模式切换按键，在 TPD 记录是推荐从界面切换机器人进入托动示教模式。如图表 3.6-9 所示；

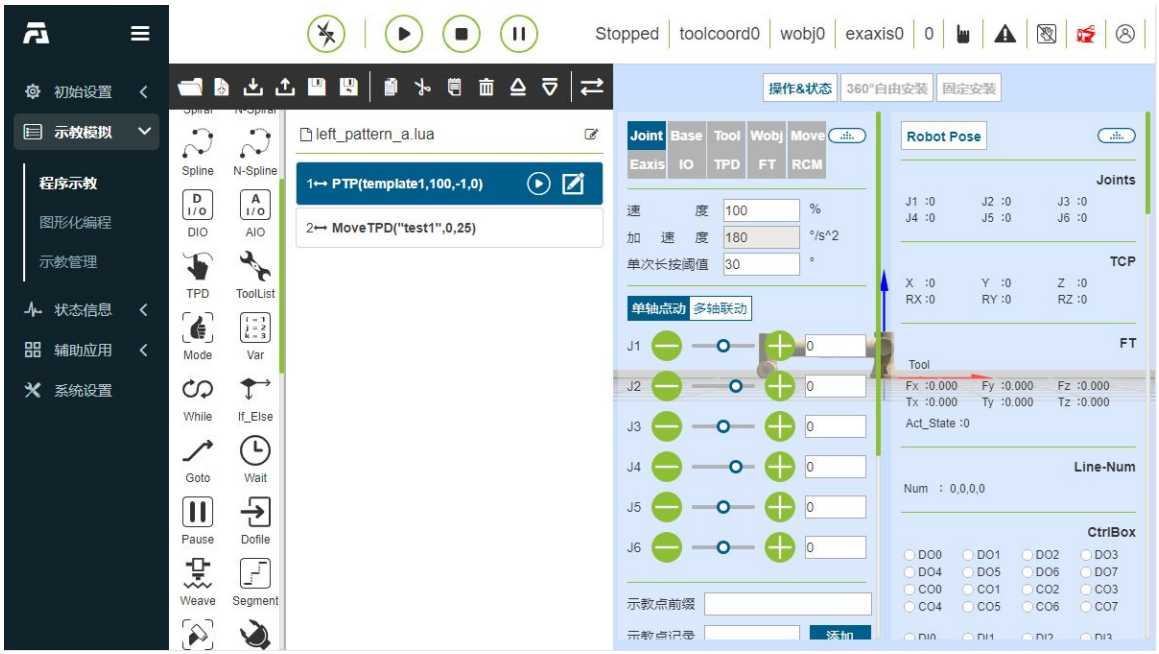
图表 3.6-9 机器人模式

重要： 从界面切入拖动模式时，先确认末端工具负载以及质心是否设置正确、摩檫力补偿系数是否设置合理，然后通过长按末端按钮确认拖动是否正常，确认无误后从界面切入拖动模式。

- **Step4 开始记录**：点击“开始记录”按钮开始轨迹记录，拖动机器人进行动作示教。此外，末端 DI 配置中有“TPD 记录启动/停止”功能配置项，通过配置此功能，用户可以通过外部信号触发“开始记录”

轨迹功能，需要注意的是，通过外部信号开始记录轨迹，首先得在页面先进行 TPD 轨迹的信息配置。

- **Step5 停止记录**：动作示教完成后，点击“停止记录”按钮，停止轨迹记录，然后通过拖动示教切换按键使机器人退出拖动示教模式。示教器接收到“停止轨迹记录成功”即表示轨迹记录成功。同步骤 4，在配置“TPD 记录启动/停止”功能后，可以通过外部信号触发停止记录。
- **Step6 示教编程**：点击新建，选择空白模板，点击进入 PTP 功能编程项，选择刚保存的初始位置点，点击“添加”按钮，应用完成后，在程序文件中会显示一条 PTP 指令；然后点击进入 TPD 功能编程项，选择刚刚记录的轨迹，设定是否平滑以及速度缩放比例，点击“添加”按钮，应用完成后，在程序文件中会显示一条 MoveTPD 指令，如图表 3.6-10 所示；



图表 3.6-10 TPD 编程

- **Step7 轨迹复现**：示教程序编辑完成后，切换至自动运行模式，点击界面上方”开始运行”图标开始运行程序，机器人开始复现示教的动作。
- **Step8 轨迹编辑**：TPD 轨迹编辑区可对轨迹可视化展示和编辑裁切，以达到 TPD 轨迹预分析和精简。选择对应轨迹获取点，那么用户记录的轨迹点会展示在机器人三维空间内，其次用户可以拖动“Start”和“End”滚动条对轨迹的起点和终点进行模拟复现和剪辑。

TPD 文件删除与异常处理：

- **轨迹文件删除**：点击进入 TPD 功能项，选择需要删除的轨迹文件，点击”删除轨迹”按钮，若删除成功，则会收到删除成功提示。
- **异常处理：**
  - **指令点数超限**：一条轨迹最多可记录 2 万个点数，当超过 2 万个点时，控制器不再记录超过的点数，并向示教器发出“指令点数超限”告警提示，此时需点击停止记录；
  - **TPD 指令间隔过大**：若示教器报错 TPD 指令间隔过大，则应检查机器人是否回到了记录前的初始位置，若机器人回到了初始位置依然报错 TPD 指令间隔过大，则删除当前轨迹重新记录一条新的

轨迹；

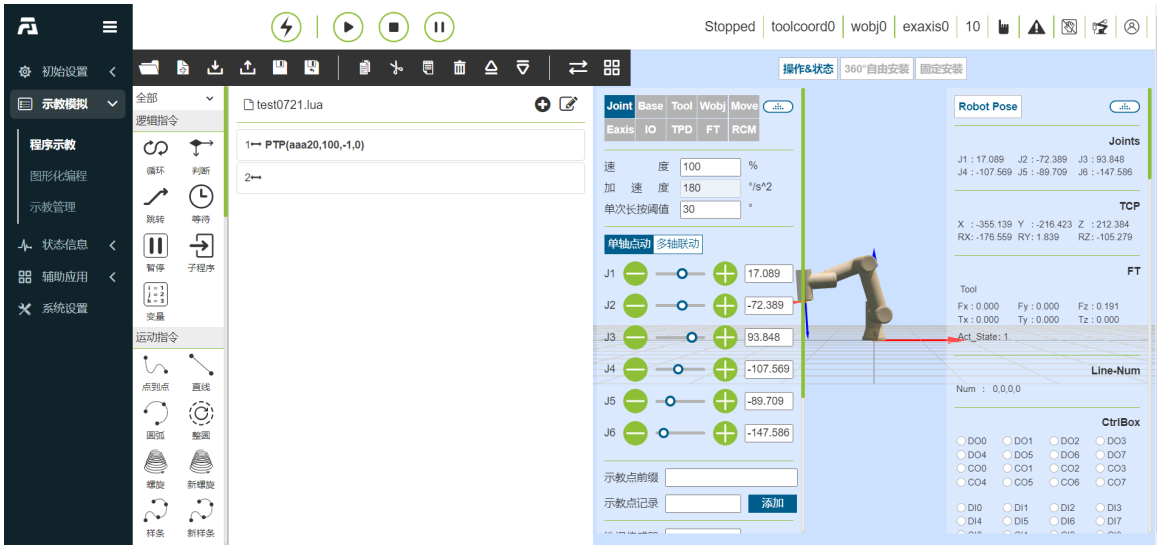
- TPD 操作过程中若出现其他异常情况，则应通过示教器或急停按钮立即停止机器人操作，检查原因。

重要： TPD 功能操作过程中应严格按照示教器上相应的提示进行操作。

1.3.5.7 示教模拟

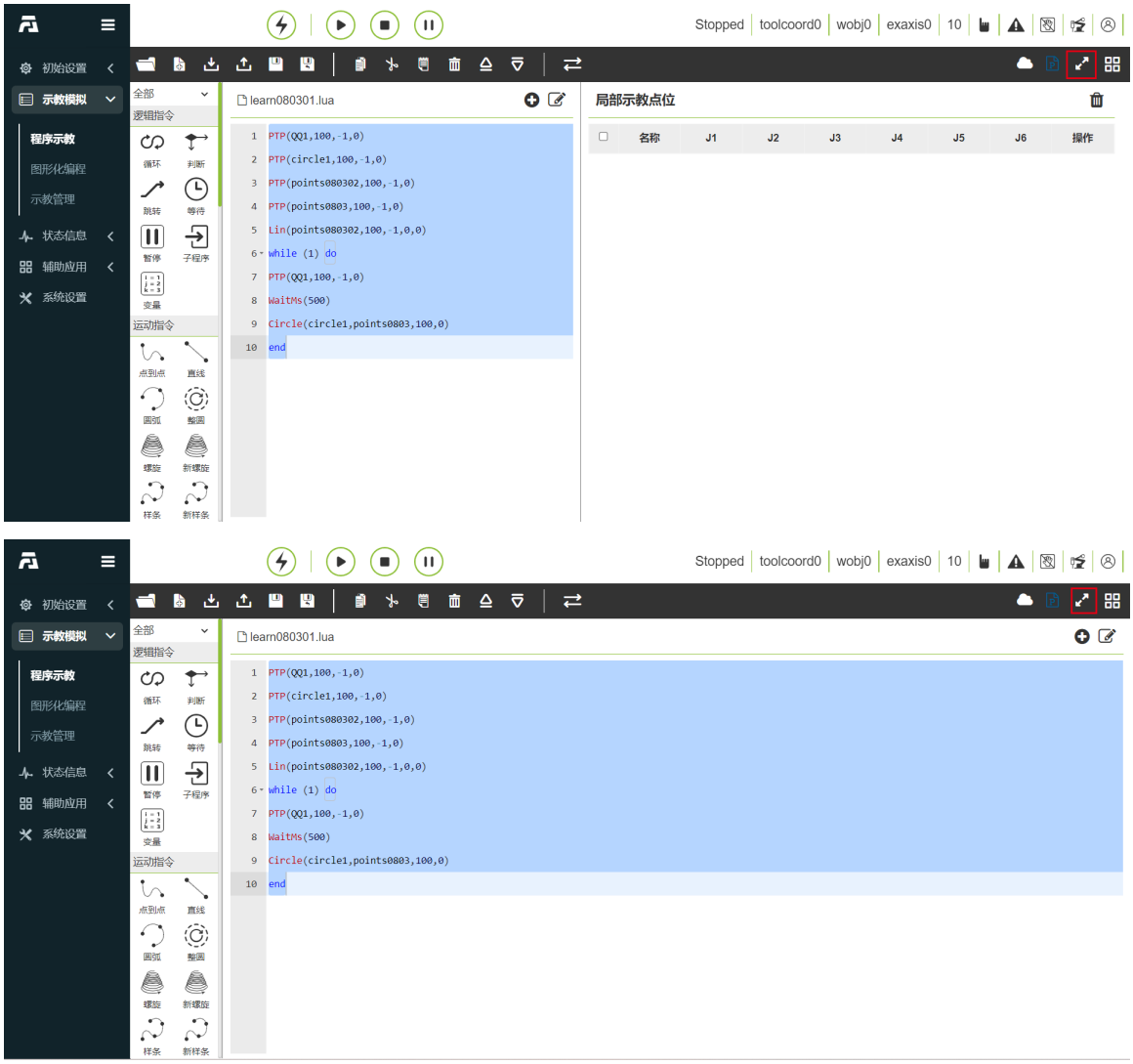
1.3.5.7.1 简介

点击左侧命令可以向程序树添加程序节点。程序运行时，当前执行的程序节点绿色高亮显示。在手动模式下，点击节点右侧第一个图标可以使机器人单独执行该指令，第二个图标为编辑该节点内容



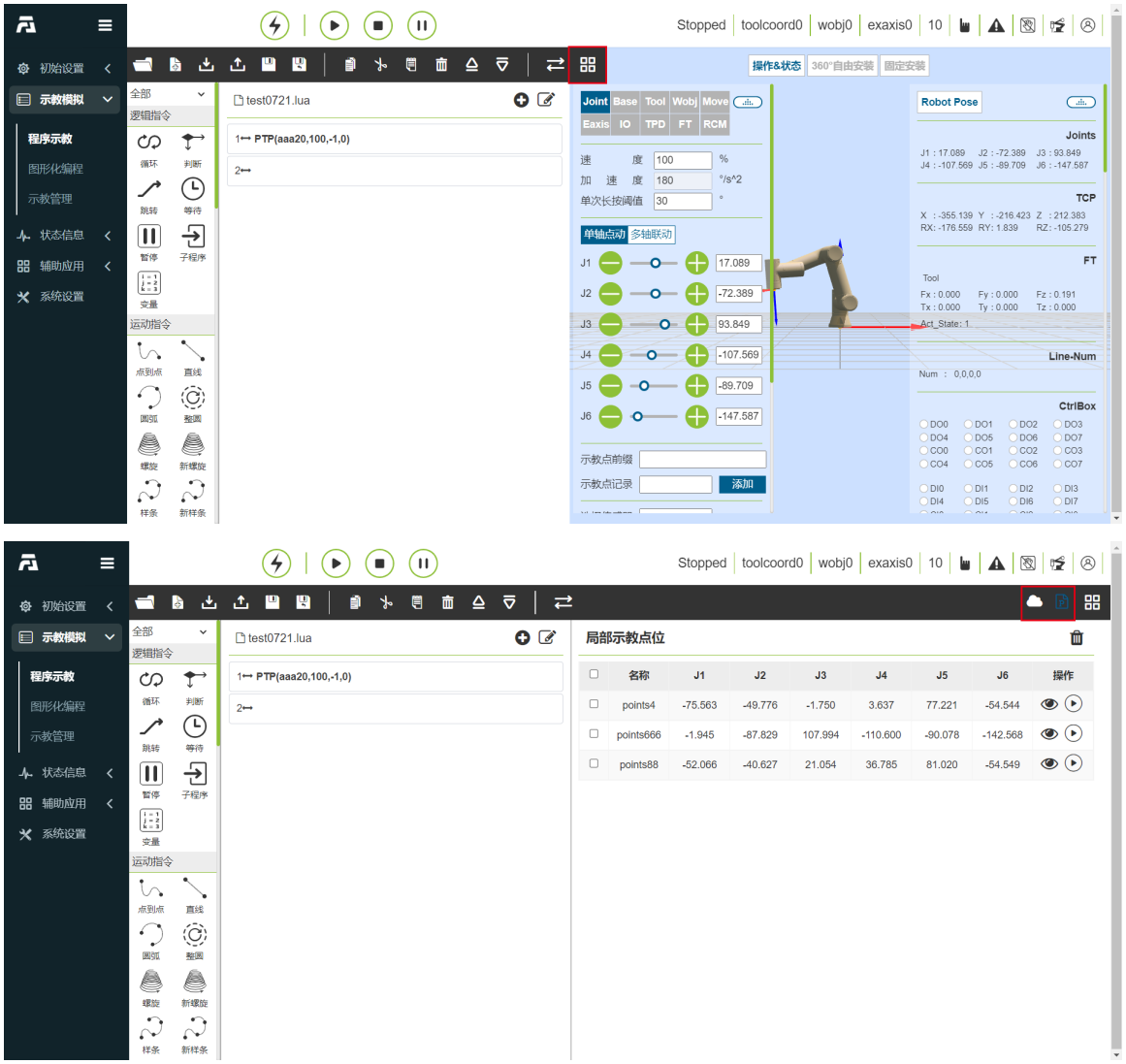
图表 3.7-1 程序树界面

点击“⇄”切换模式，可以将示教程序文本变为编辑状态，编辑状态下可以展开和收起编辑区域。



图表 3.7-2 示教程序编辑状态

点击“当前程序右侧内容弹出/隐藏”按钮，可以展开或隐藏局部示教点位和当前程序备份内容。右侧内容展开后，点击“局部示教点位”和“当前程序备份”图标展示相对应的内容。



图表 3.7-3 当前程序右侧内容

1.3.5.7.2 工具栏

使用程序树底部的工具栏修改程序树。

备注:

名称：**打开**

作用：打开用户程序文件

备注:

名称：**新建**

作用：选择模板新建程序文件

备注:

名称：**导入**

作用：导入文件到用户程序文件夹中

备注:

名称：**导出**

作用：导出用户程序文件到本地点。

备注:

名称：**保存**

作用：保存文件编辑内容

备注:

名称：**另存为**

作用：给文件重命名存放到用户程序或模板程序文件夹中。

备注:

名称：**复制**

作用：复制一个节点，并允许将其用于其他操作（例如：将其粘贴到程序树的其他位置）。

备注:

名称：**粘贴**

作用：允许您粘贴之前剪切或复制的节点。

备注:

名称：**剪切**

作用：剪切一个节点，并允许将其用于其他操作（例如：将其粘贴到程序树的其他位置）。

备注:

名称：**删除**

作用：从程序树中删除一个节点。

备注:

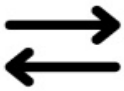
名称：**上移**

作用：向上移动该节点。

备注:

名称：**下移**

作用：向下移动该节点。



备注:

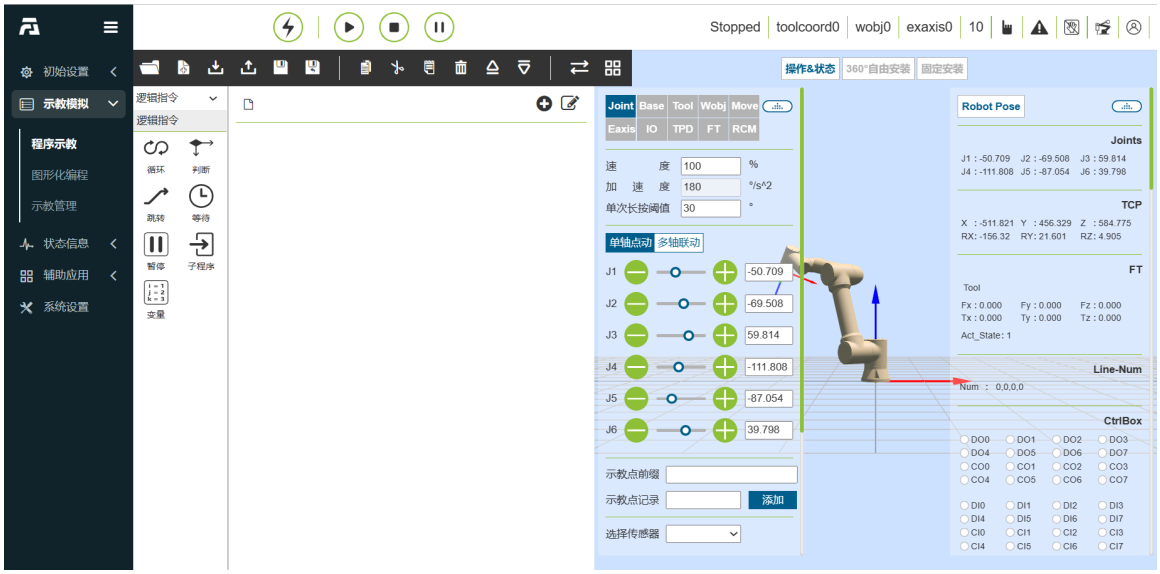
名称：**切换编辑模式**

作用：程序树模式和 lua 编辑模式互相切换。

1.3.5.7.3 程序命令

左侧主要是程序命令的添加，点击各关键字上方图标进入详细界面，程序命令添加到文件中的操作主要分为两种，一种方式打开相关指令点击应用按键即可将该指令添加到程序中，另一种方式为先点击“添加”按键，此时命令并未保存到程序文件中，需要再点击“应用”方可将命令保存到文件中。第二种方式多出现在同类型指令多条下发的情况，我们对该类型命令增加添加按键和显示已添加指令内容功能，点击添加按键可添加一条指令，已添加指令显示所有已添加的指令，点击“应用”即可将添加的指令保存到右侧已打开的文件中。

1.3.5.7.4 逻辑指令界面

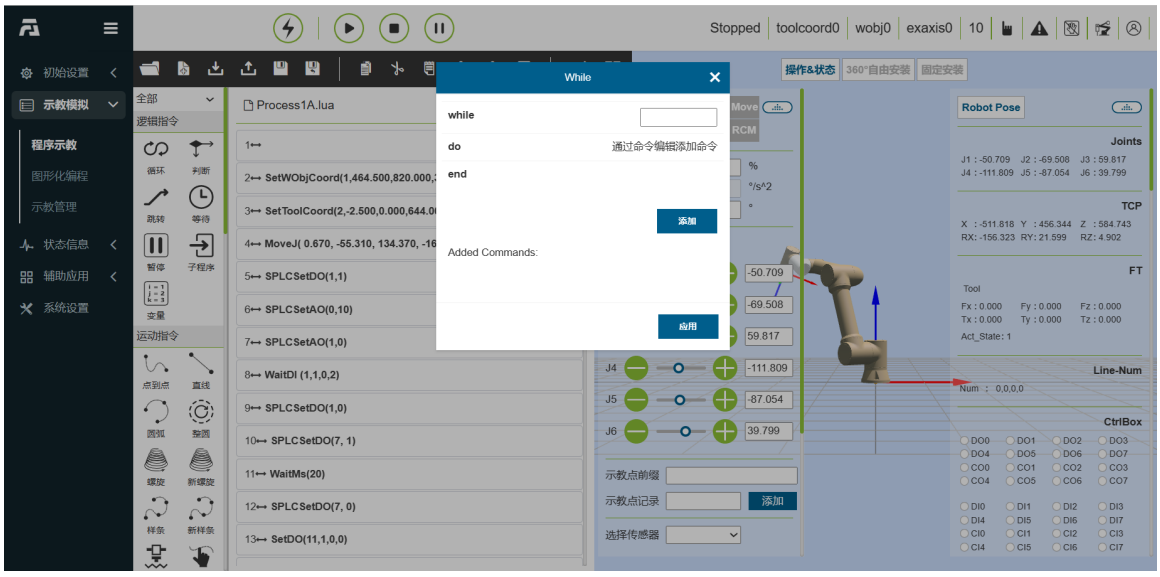


图表 3.7-4 逻辑指令界面

1.3.5.7.4.1 循环命令

点击“循环”图标进入 While 命令编辑界面

在 While 后方的输入框中输入等待条件，在 do 后方的输入框中输入循环期间的动作指令，点击保存即可。(为方便操作，可任意输入 do 内容，在程序中编辑其他指令插入代替)

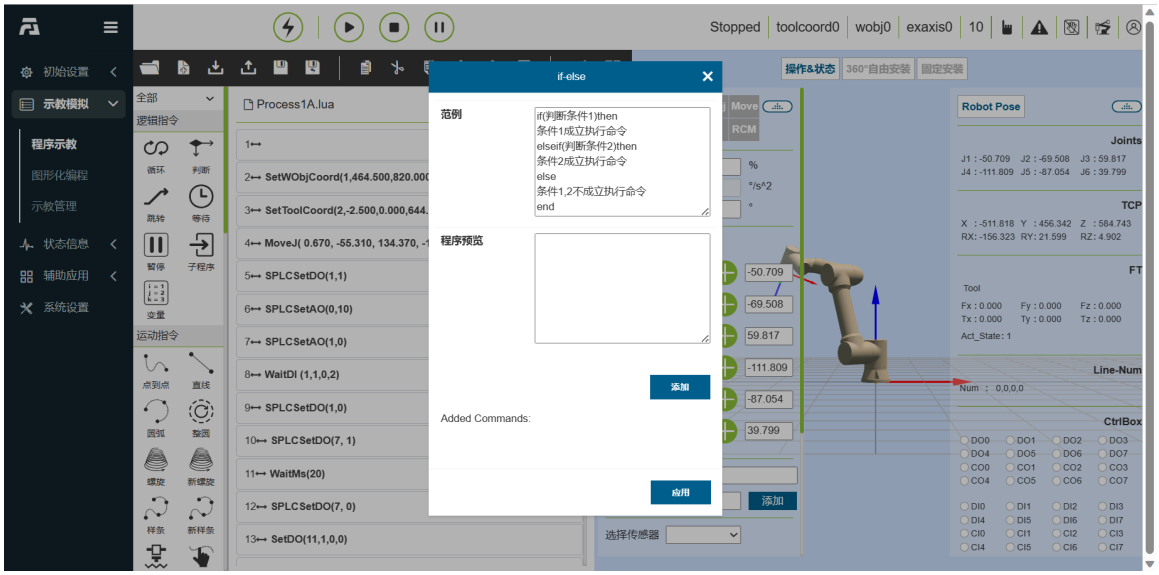


图表 3.7-4-1 While 指令界面

1.3.5.7.4.2 判断命令

点击“判断”按钮进入 if…else 命令编辑界面

在右侧输入框中输入语句，编辑完毕后点击“添加”、“应用”即可。(该指令需要一定编程基础，如需帮助，请联系我们)

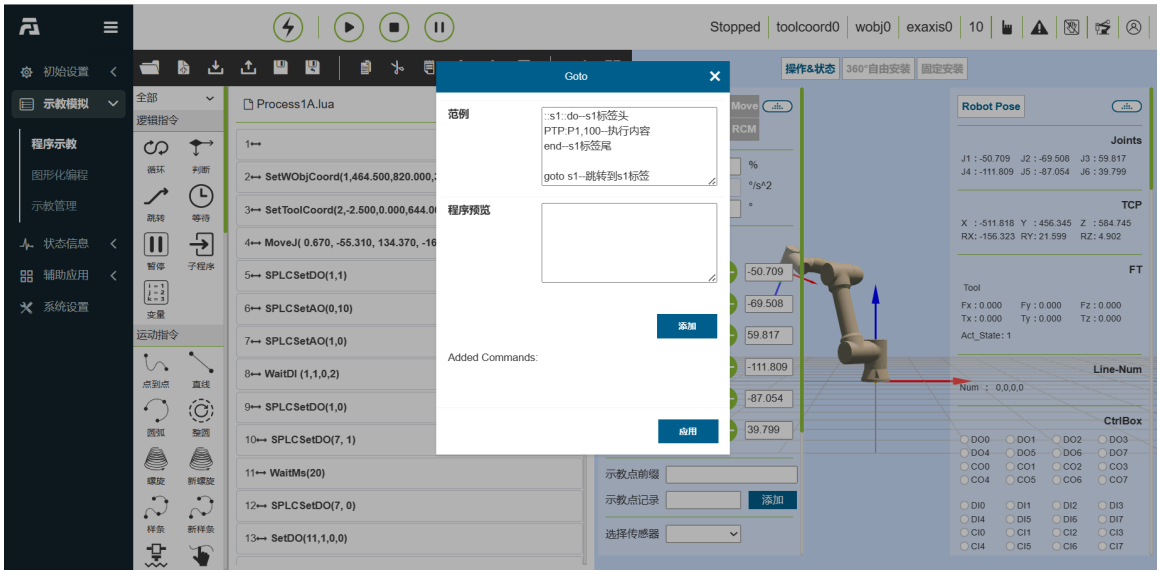


图表 3.7-4-2 if…else 指令界面

1.3.5.7.4.3 跳转命令

点击“跳转”按钮进入 Goto 命令编辑界面

Goto 指令为跳转指令，在右侧输入框中输入语句，编辑完毕后点击“添加”、“应用”即可。(该指令需要一定编程基础，如需帮助，请联系我们)



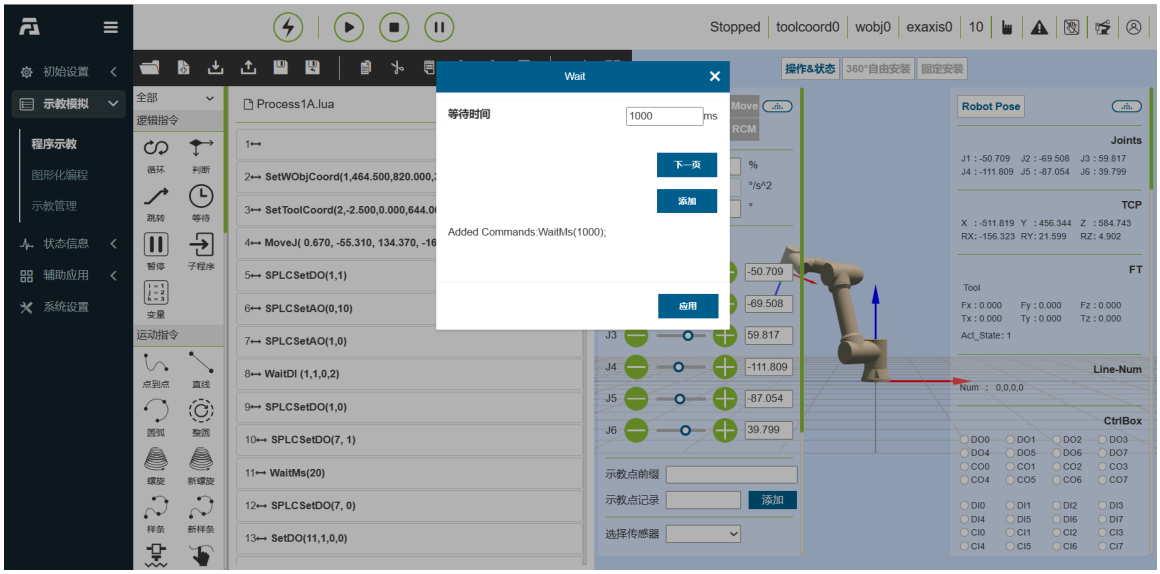
图表 3.7-4-3 Goto 指令界面

1.3.5.7.4.4 等待命令

点击“等待”图标进入 Wait 命令编辑界面

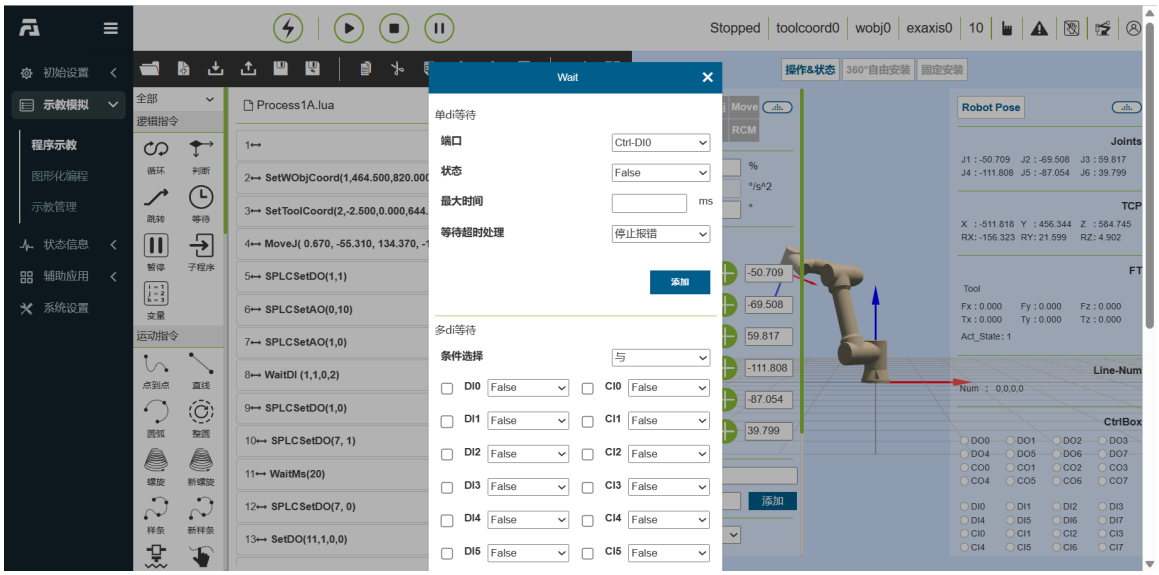
该指令为延时指令，分为“WaitMs”、“WaitDI”和“WaitAI”三部分。

“WaitTime”指令延时等待时间单位为毫秒，输入需要等待的毫秒数，点击“添加”、“应用”即可。



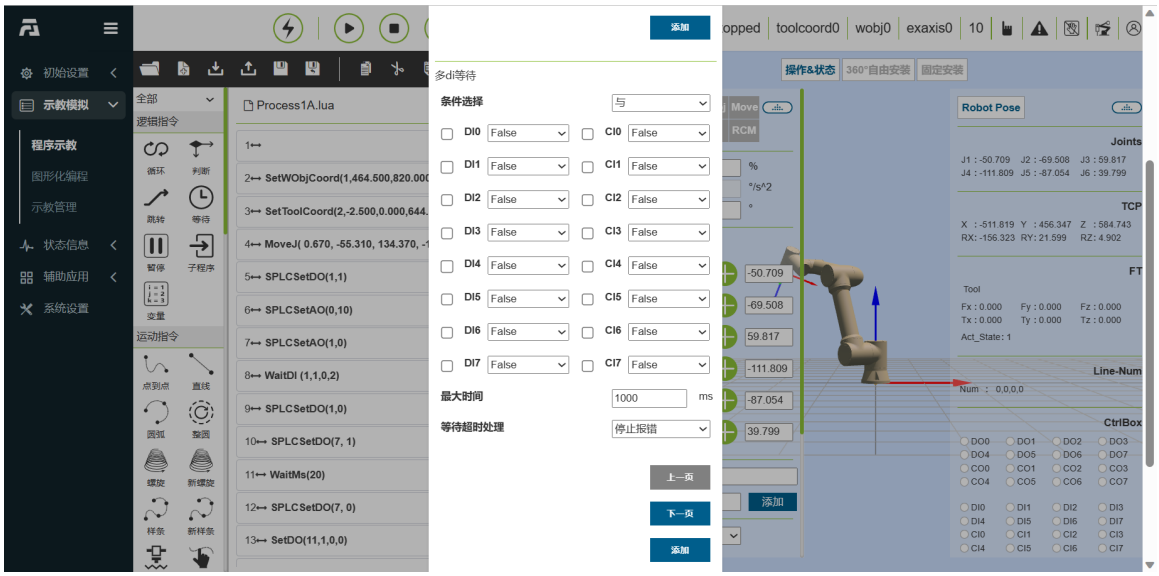
图表 3.7-4-4 WaitTime 指令界面

“WaitDI”指令，即单 DI 等待，选择需要等待的 IO 端口号、等待状态、等待最大时间和等待超时处理方式，点击“添加”、“应用”即可。



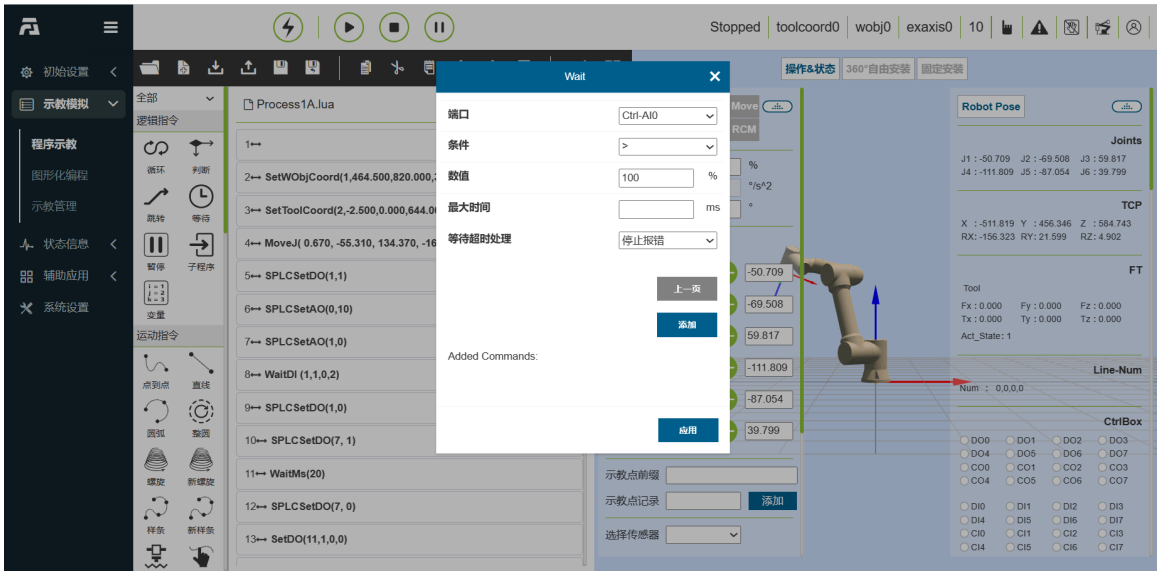
图表 3.7-4-5 WaitDI 指令界面

“WaitMultiDI”指令，即多 DI 等待，首先选择多 DI 成立条件，其次勾选需要等待的 DI 端口和状态，最后设置等待最大时间和等待超时处理方式，点击“添加”、“应用”即可。



图表 3.7-4-6 WaitMultiDI 指令界面

“WaitAI”指令，选择需要等待的模拟量、数值、等待的最大时间以及等待超时处理方式，点击“添加”、“应用”即可。

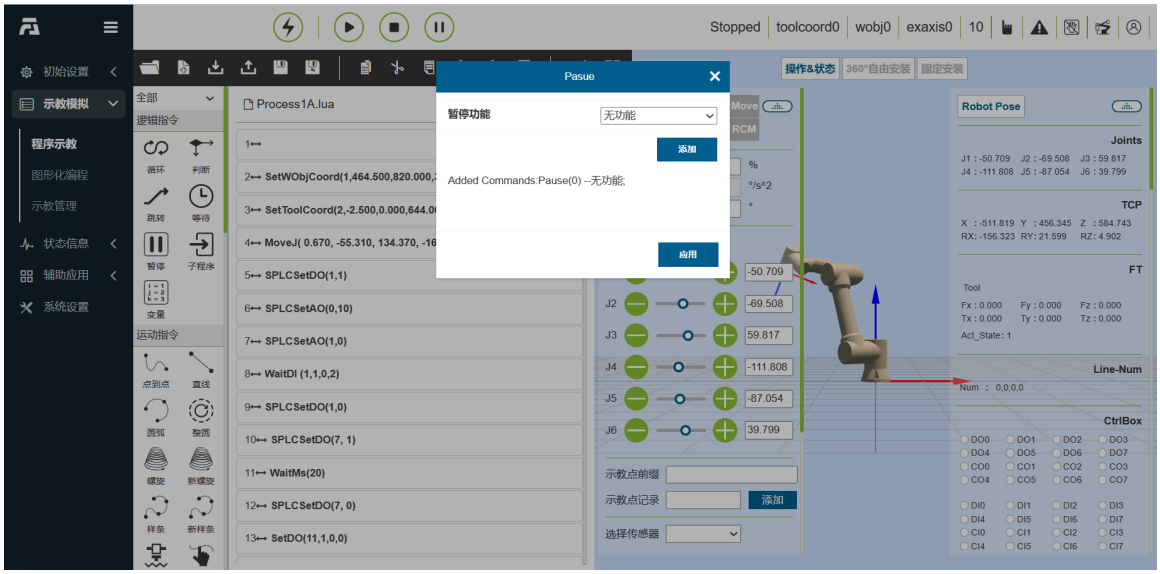


图表 3.7-4-7 WaitAI 指令界面

1.3.5.7.4.5 暂停命令

点击“暂停”图标进入 Pause 命令编辑界面

该指令为暂停指令，在程序中插入该指令，当程序执行到该指令时，机器人会处于暂停状态，若想继续运行，点击控制区“暂停/恢复”按键即可。

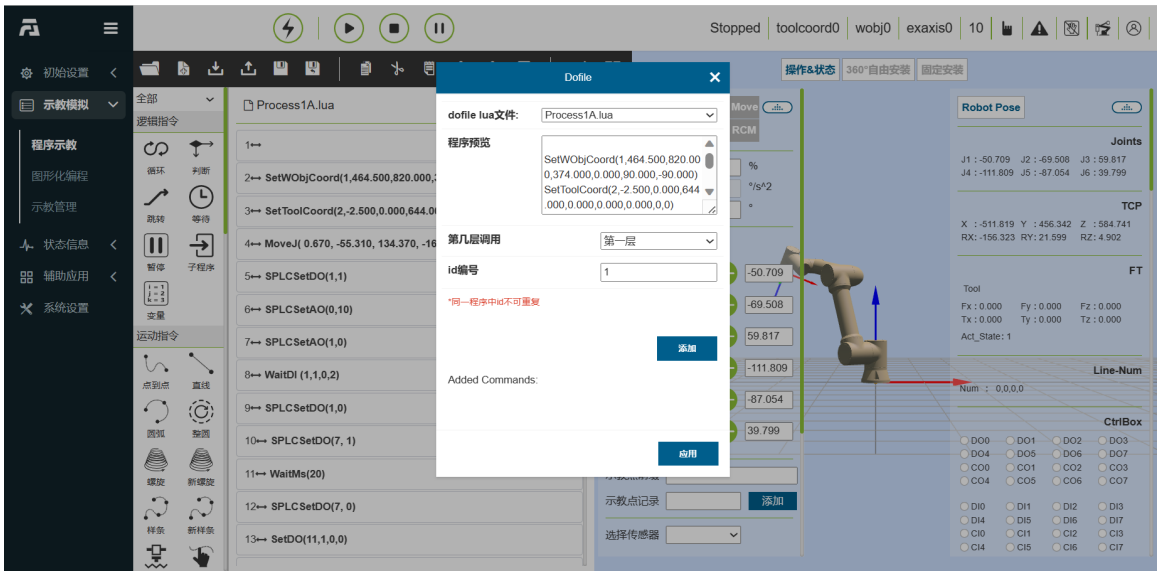


图表 3.7-4-8 Pause 指令界面

1.3.5.7.4.6 子程序命令

点击“子程序”图标进入 Dofile 命令编辑界面

Dofile 指令调用的是控制器内部程序，使用 Dofile 指令需要保存被调用的子程序，而主程序若未改变可不用再次保存。Dofile 指令支持二级调用，需要注意两个参数设置，一是该调用处于第几层，二是该调用的 ID 编号，ID 编号原则上同一程序不能出现相同 ID。

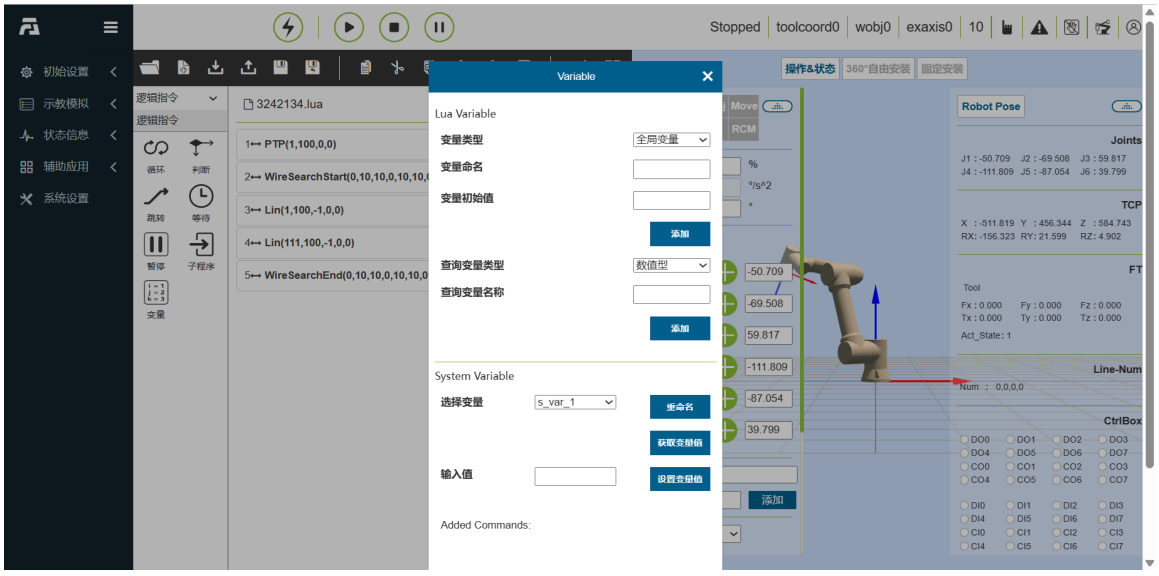


图表 3.7-4-9 Dofile 指令界面

1.3.5.7.4.7 变量命令

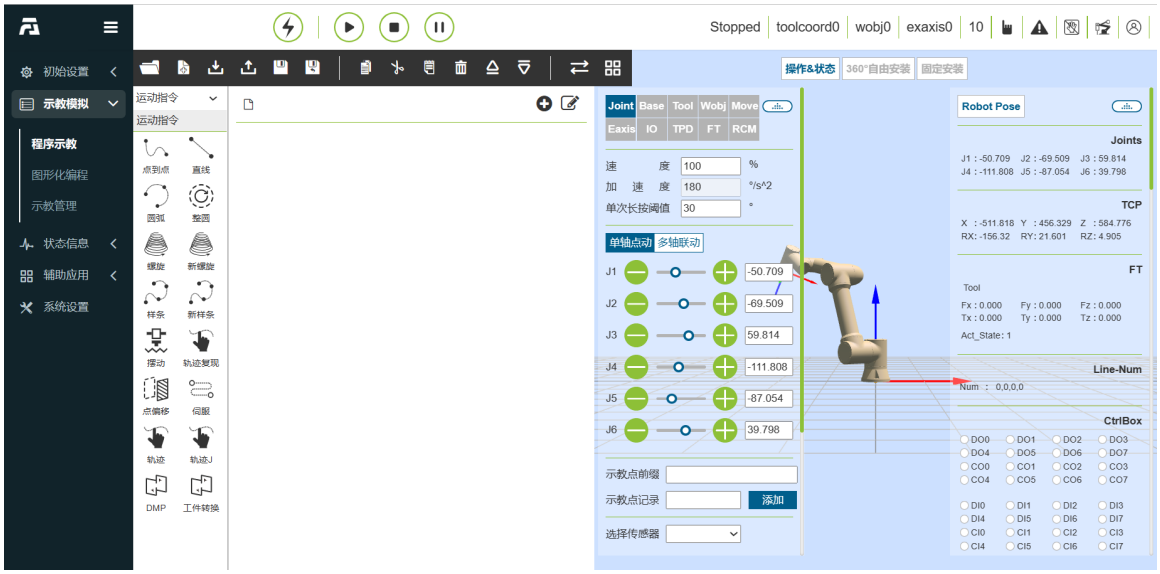
点击“变量”图标进入 Var 命令编辑界面

该指令为变量系统指令，分为 Lua 变量定义，变量查询和 Sys 变量重命名，获取值，设置值两部分，Lua 变量定义可以声明一个变量并赋予初始值，与 while，if-else 等指令配合使用，Lua 变量查询指令可以实时查询输入的变量名称的值，显示在状态栏。Sys 变量个数是固定的，可以对其重命名，获取变量值以及设置变量值，该变量保存的值不随系统关机而清零。



图表 3.7-4-10 Var 指令界面

1.3.5.7.5 运动指令界面

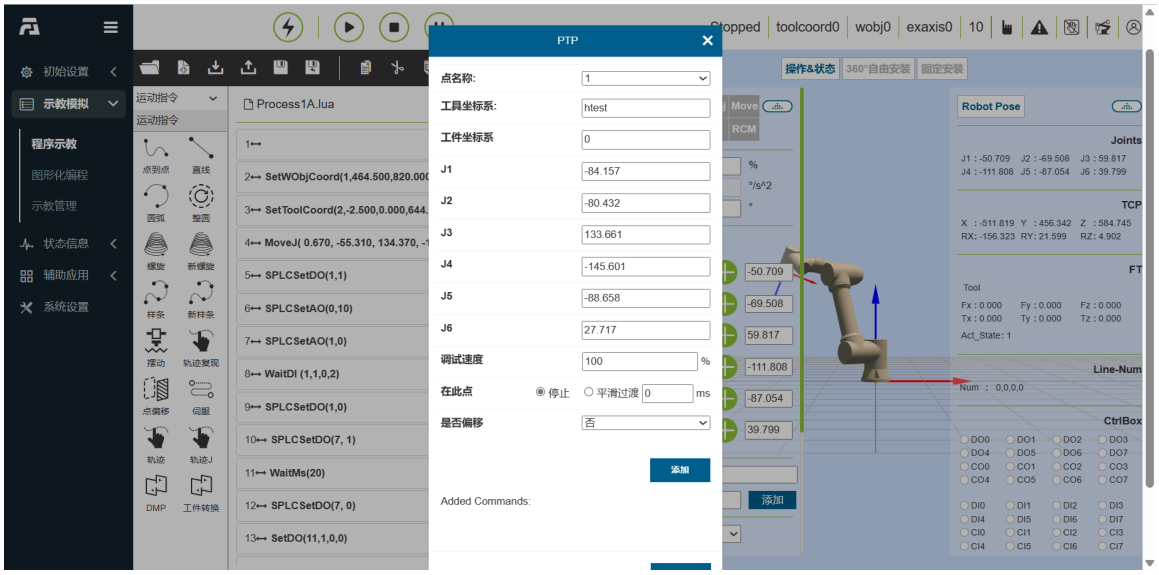


图表 3.7-5 运动指令界面

1.3.5.7.5.1 点到点命令

点击“点到点”图标进入 PTP 命令编辑界面。

可以选择需要到达的点，平滑过渡时间设置可以实现该点到下一点的运动是连续的，是否偏移设置，可以选择基于基坐标系偏移和基于工具坐标偏移，并弹出 x,y,z,rx,ry,rz 偏移量设置，PTP 具体路径为运动控制器自动规划的最优路径，点击“添加”、“应用”后可保存该条指令。

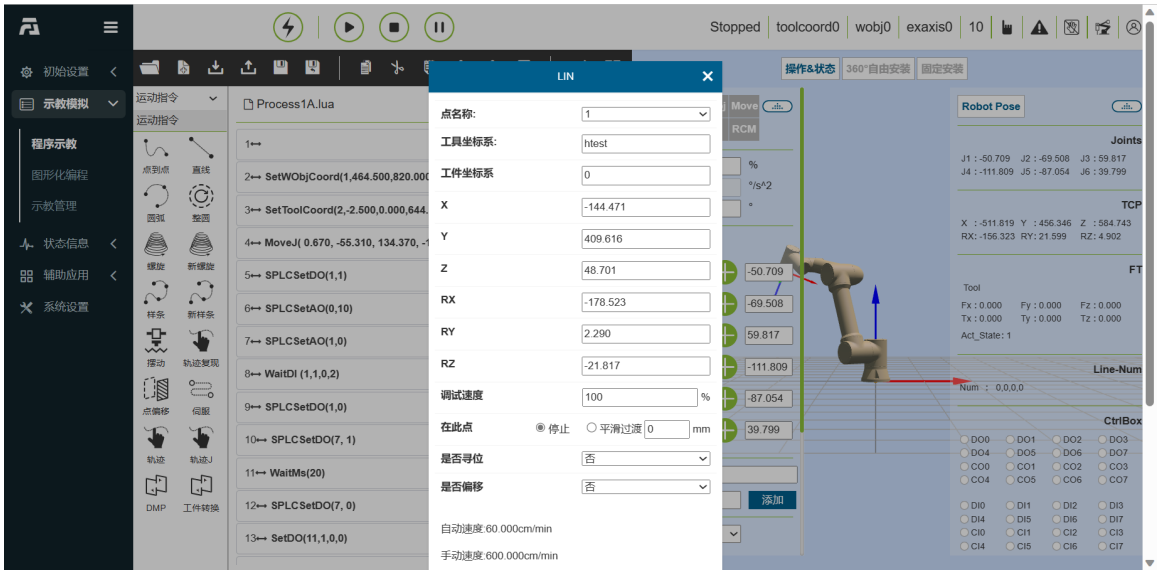


图表 3.7-5-1 PTP 指令界面

1.3.5.7.5.2 直线命令

点击“直线”图标进入 Lin 命令编辑界面。

该指令功能与“PTP”指令相似，但该指令所到达点的路径为直线。

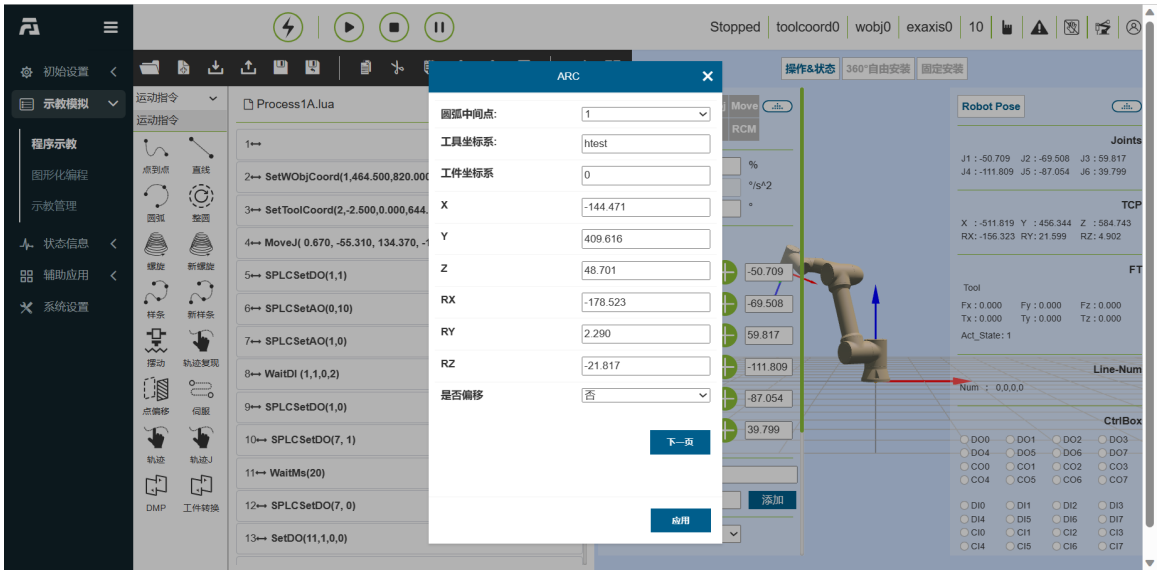


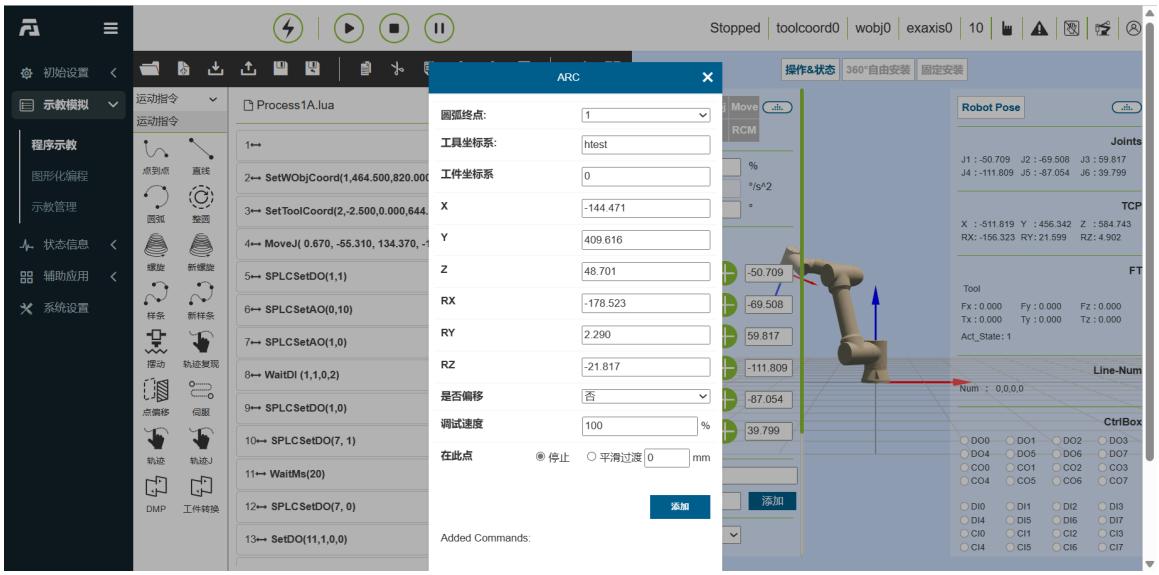
图表 3.7-5-2 Lin 指令界面

1.3.5.7.5.3 圆弧命令

点击“圆弧”图标进入 Arc 命令编辑界面。

“Arc”指令为圆弧运动，包含两个点，第一点为圆弧中间过渡点，第二点为终点，过渡点和终点都可以对是否偏移进行设置，可以选择基于基坐标系偏移和基于工具坐标偏移，并弹出 x,y,z,rx,ry,rz 偏移量设置，终点可以设置平滑过渡半径，实现运动连续效果。



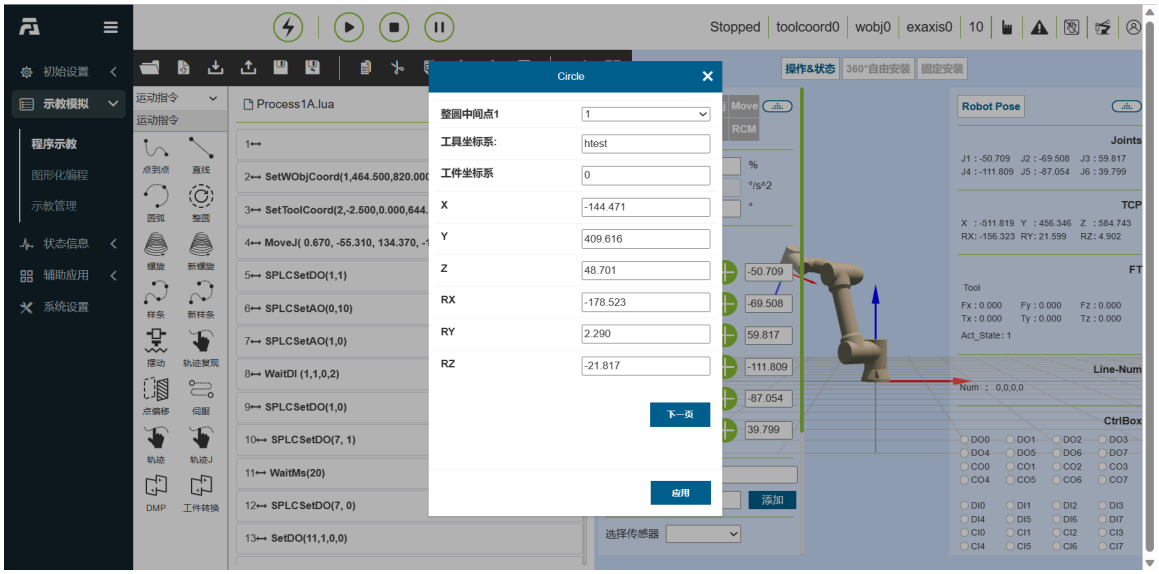


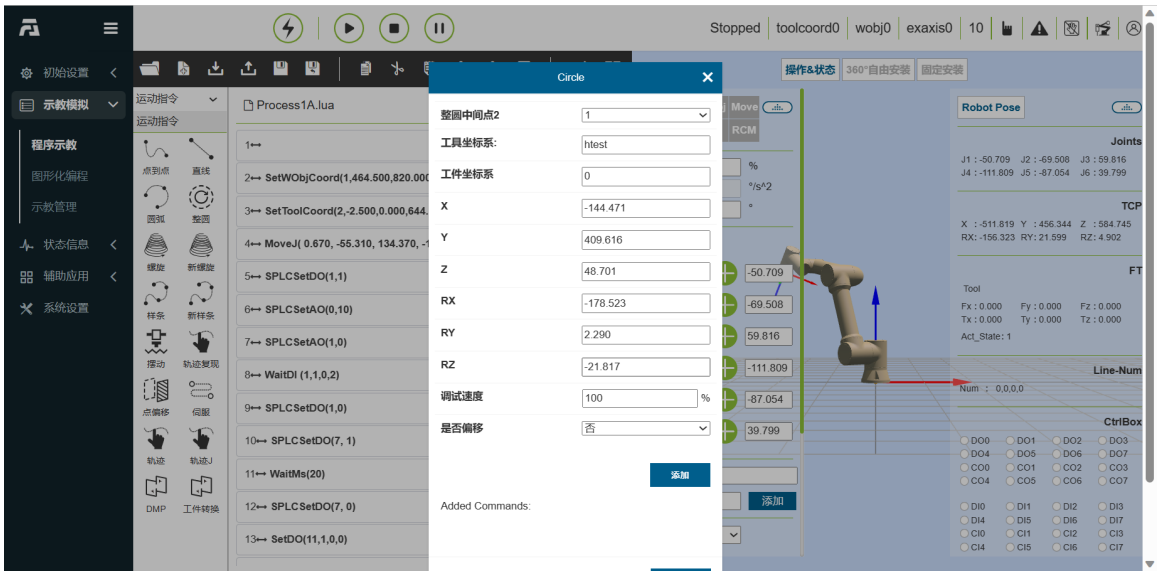
图表 3.7-5-3 Arc 指令界面

1.3.5.7.5.4 整圆命令

点击“整圆”图标进入 Circle 命令编辑界面。

“Circle”指令为整圆运动，包含两个点，第一点为整圆中间过渡点 1，第二点为整圆中间过渡点 2，过渡点 2 可以设置是否偏移，该偏移量同时生效于过渡点 1 和过渡点 2。



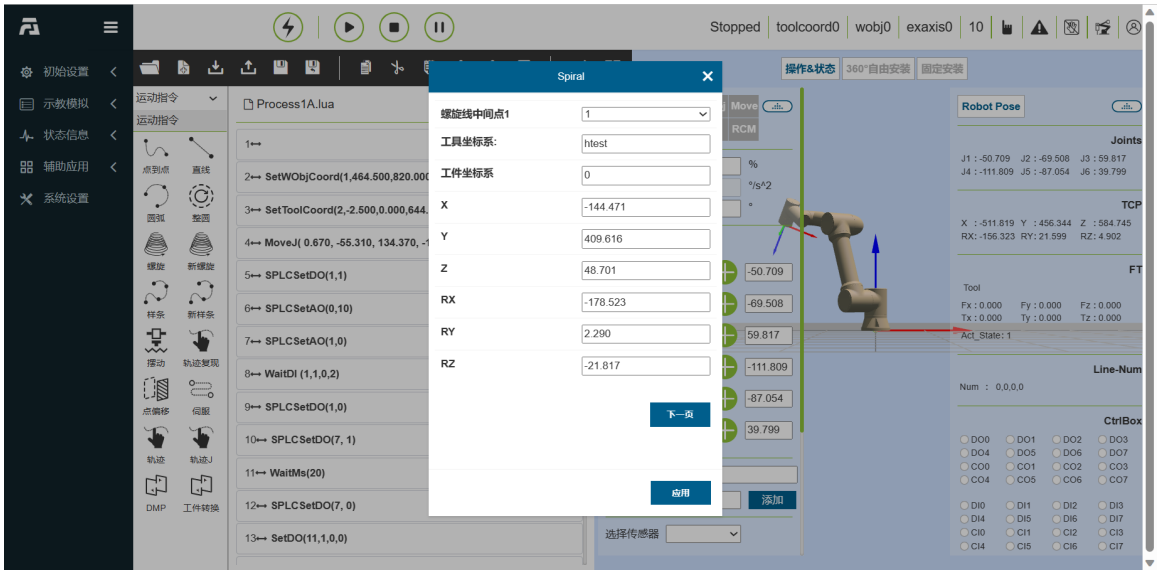


图表 3.7-5-4 Circle 指令界面

1.3.5.7.5.5 螺旋命令

点击“螺旋”图标进入 Spiral 命令编辑界面

“Spiral”指令为螺旋线运动，包含三个点，该三个点组成一个圆，在第三点设置页面，包含螺旋圈数，姿态修正角，半径增量和转轴方向增量这几个参数设置，螺旋圈数即该螺旋线的运动圈数，姿态修正角修正的是螺旋线结束时的姿态与螺旋线第一点的姿态，半径增量即每一圈半径的增量，转轴方向增量即螺旋轴方向的增量。设置是否偏移，该偏移量生效于整个螺旋线的轨迹。

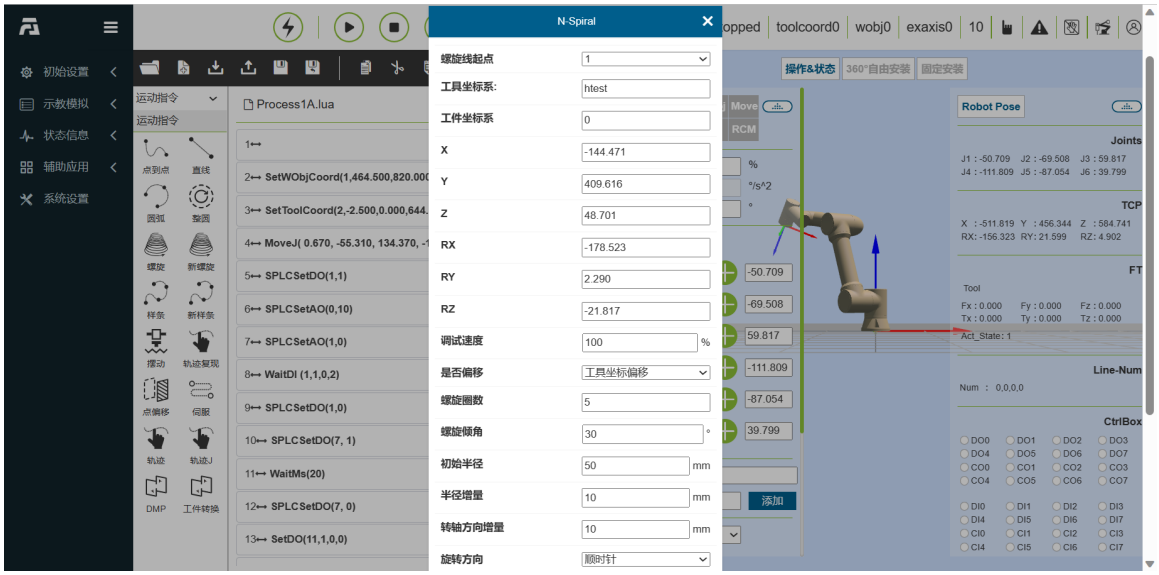


图表 3.7-5-5 Spiral 指令界面

1.3.5.7.5.6 新螺旋命令

点击“新螺旋”图标进入 N-Spiral 命令令编辑界面

“N-Spiral”指令为优化版螺旋线运动，该指令只需要一个点加各参数的配置实现螺旋线运动。机器人以当前位置作为起点，用户设置调试速度，是否偏移，螺旋圈数，螺旋倾角，初始半径，半径增量，转轴方向增量和旋转方向这几个参数，螺旋圈数即该螺旋线的运动圈数，螺旋倾角即工具 Z 轴与水平方向的夹角，姿态修正角修正的是螺旋线结束时的姿态与螺旋线第一点的姿态，初始半径即第一圈半径大小，半径增量即每一圈半径的增量，转轴方向增量即螺旋轴方向的增量，旋转方向即顺时针和逆时针。

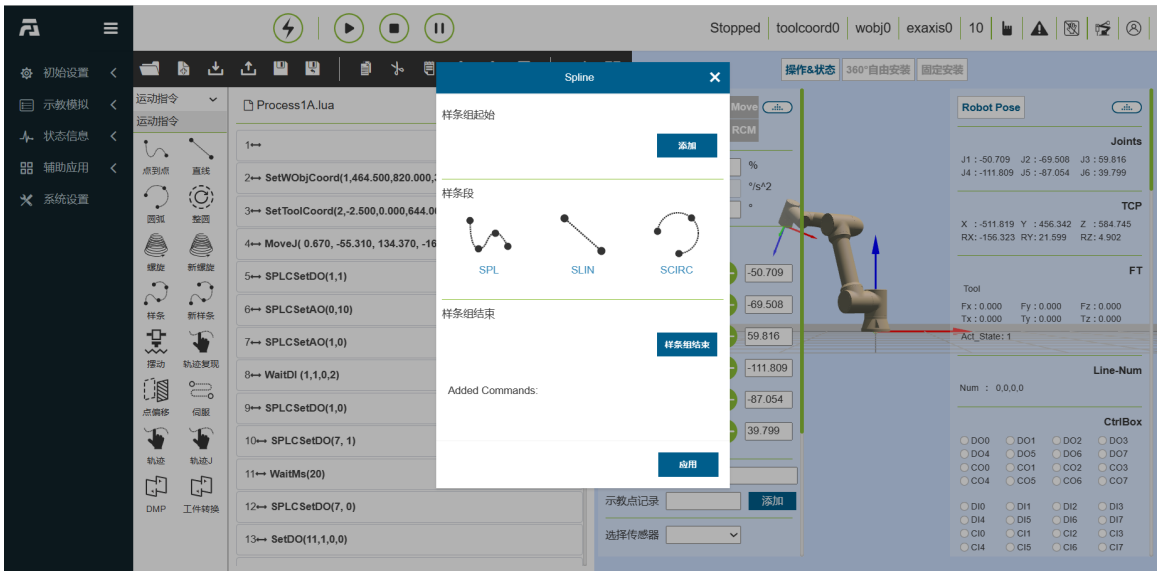


图表 3.7-5-6 N-Spiral 指令界面

1.3.5.7.5.7 样条命令

点击“样条”图标进入 Spline 命令编辑界面

该指令分为样条组起始，样条段和样条组结束三部分，样条组开始是样条运动的起始标志，样条段包含 SPL、SLIN 和 SCIRC 段，点击对应图标进入指令添加界面，样条组结束是样条运动的结束标志。

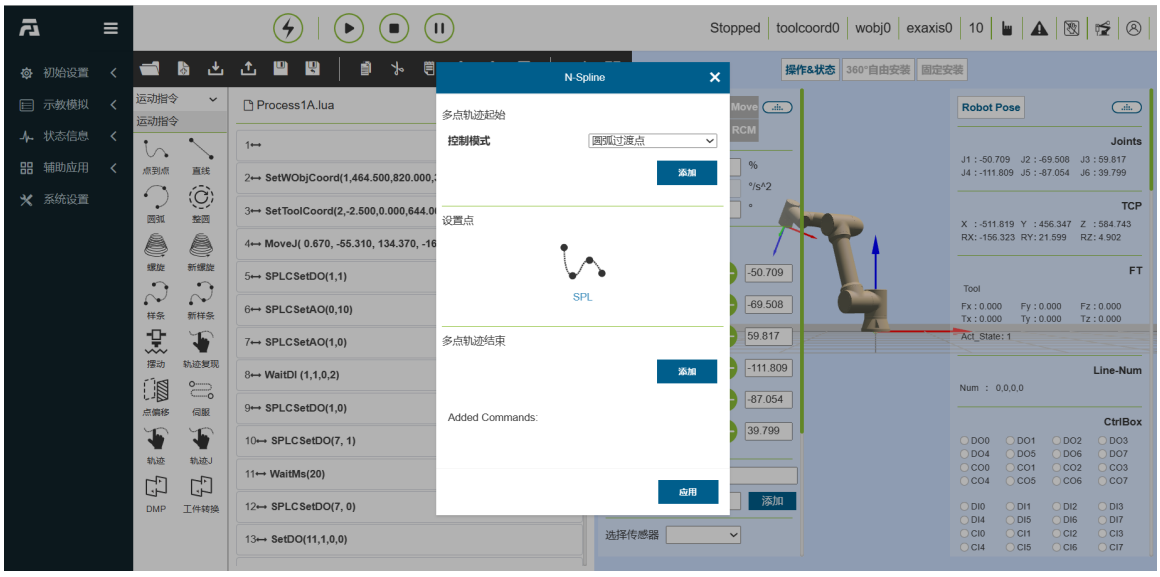


图表 3.7-5-7 Spline 指令界面

1.3.5.7.5.8 新样条命令

点击“新样条”图标进入 N-Spline 命令编辑界面

该指令为 Spline 指令算法优化指令，后续会替代现有的 Spline 指令，该指令分为多点轨迹起始，多点轨迹段和多点轨迹结束三部分，多点轨迹开始是多点轨迹运动的起始标志，多点轨迹段即设置各个轨迹点，点击图标进入点位添加界面，多点轨迹结束是多点轨迹运动的结束标志，在此可以设置控制模式和调试速度，控制模式分为给定控制点和给定路径点。

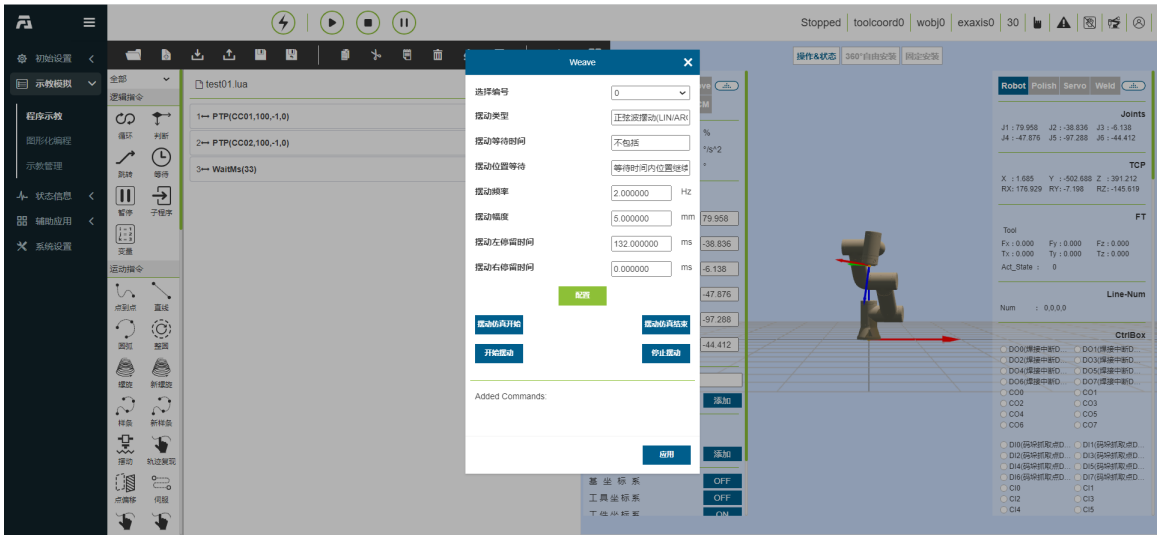


图表 3.7-5-8 N-Spline 指令界面

1.3.5.7.5.9 摆动命令

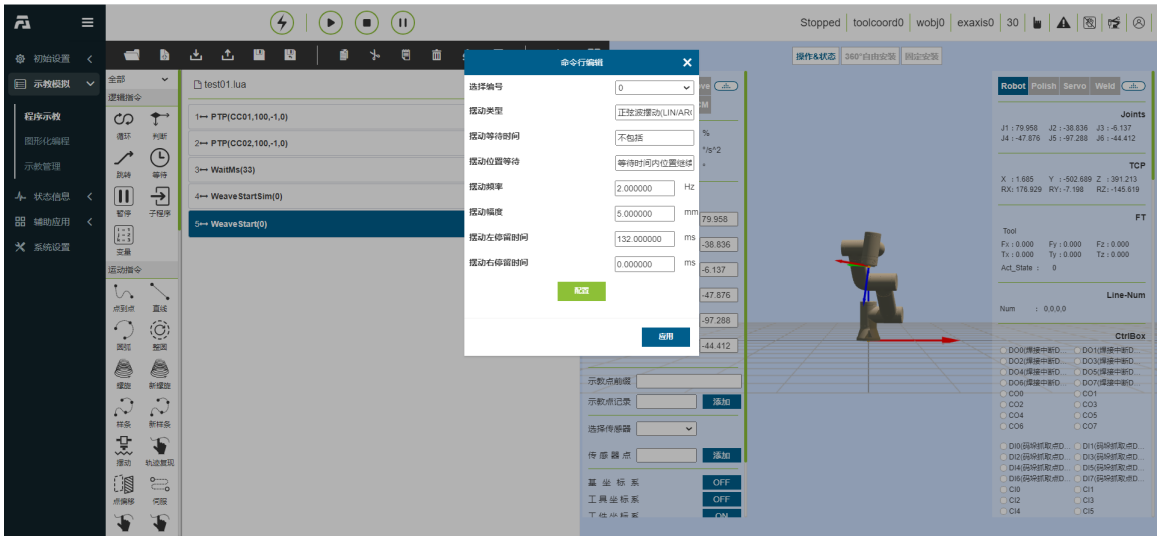
点击“摆动”图标进入 Weave 命令编辑界面

“Weave”指令包含两部分，第一部分选择配置好参数的摆焊编号，点击“开始摆焊”和“停止摆焊”并应用可将相关指令添加到程序中。



图表 3.7-5-9 Weave 指令界面

点击“配置与测试”，可以对摆焊的参数进行配置，配置完成后可通过开始摆焊测试和停止摆焊测试按键测试该摆焊轨迹。



图表 3.7-5-10 Weave 配置与测试指令界面

1.3.5.7.5.10 轨迹复现命令

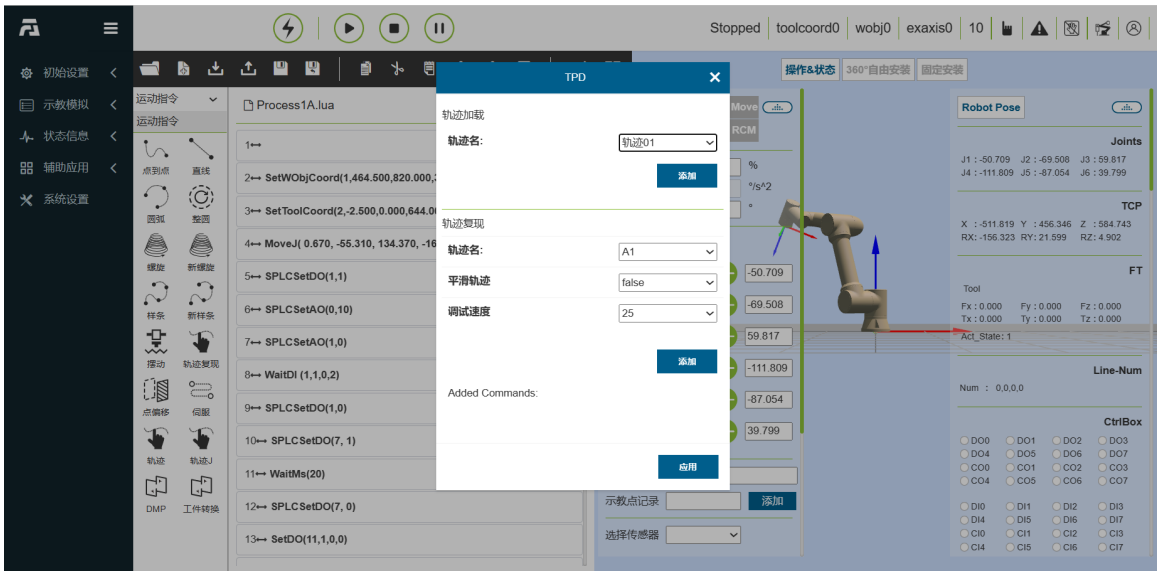
点击“轨迹复现”按钮进入 TPD 命令编辑界面

在该指令中，用户首先需要有记录好的轨迹。

关于轨迹记录：在准备记录轨迹之前，先保存下轨迹的起始点。在机器人处于拖动模式下，输入文件名，选择周期（假设数值为 x，即每隔 x 毫秒记录一个点，推荐 4 毫秒记录一个点），点开始记录，用户可以根据需求拖动机器人进行指定运动，记录完成后，点击停止记录，即可保存之前机器人的运动轨迹。当一条运动无法完全记录，会提示记录点数超限提示，用户需要将运动分几次进行记录。

进行程序编程时，首先用 PTP 指令到达对应轨迹起始点，然后在 TPD 轨迹复现指令中选择轨迹，选择是否平滑，设置调试速度，依次点击“添加”、“应用”，即可插入程序。轨迹加载指令主要用于预先读取轨迹文件，提取成轨迹指令，更好的应用于传送带跟踪场景。

备注： 关于 TPD 详细操作可见示教编程（TPD）功能操作说明模块。

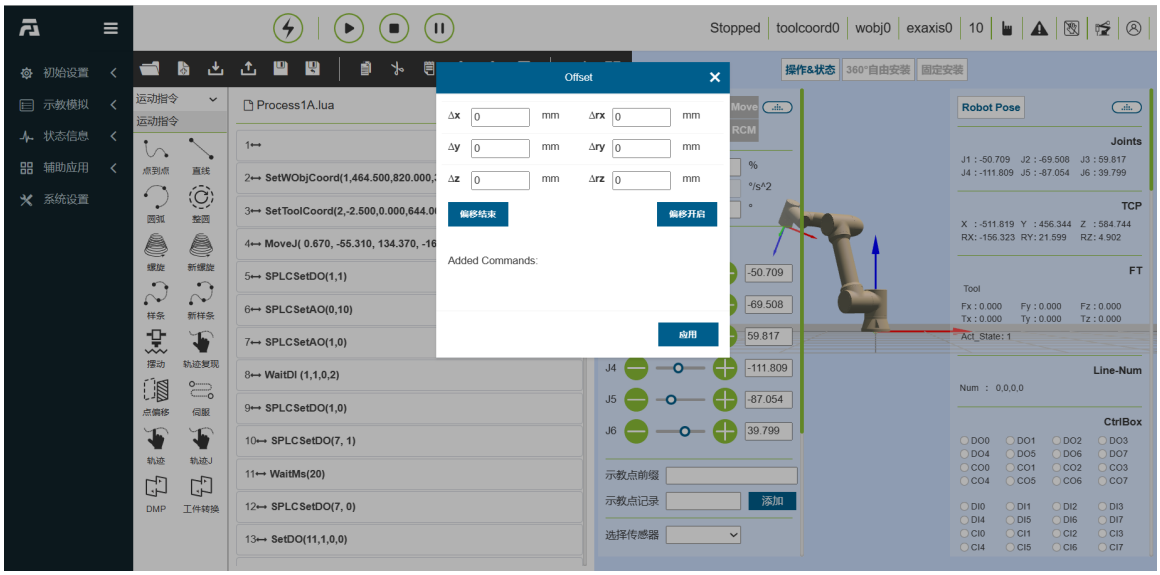


图表 3.7-5-11 TPD 指令界面

1.3.5.7.5.11 点偏移命令

点击“点偏移”图标进入 Offset 命令编辑界面

该指令为整体偏移指令，输入各个偏移量，将开启指令和关闭指令添加到程序中，在开始和关闭中间的运动指令会基于基坐标（或工件坐标）进行偏移。

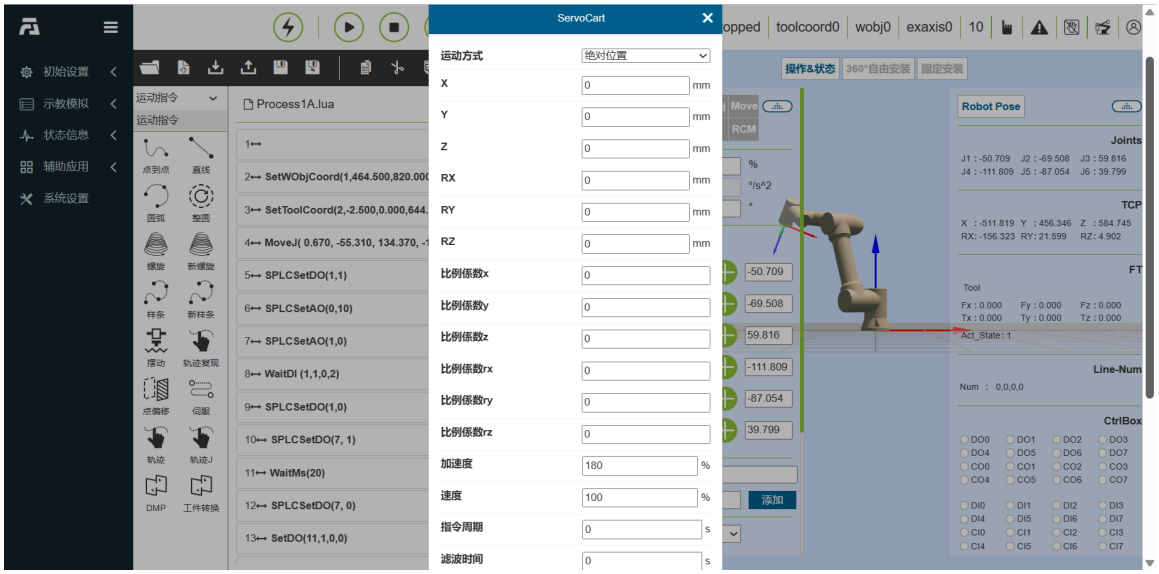


图表 3.7-5-12 Offset 指令界面

1.3.5.7.5.12 伺服命令

点击“伺服”图标进入 ServoCart 命令编辑界面

ServoCart 伺服控制（笛卡尔空间运动）指令，该指令可以通过绝对位姿控制或基于当前位姿偏移来控制机器人运动。



图表 3.7-5-13 ServoCart 指令界面

绝对位姿控制程序实例：

```
PTP(p1,100,0,0)
x,y,z,rx,ry,rz = GetActualTCPPose()
while 1 do
    if type(x) == "number" then
        x = x+1
        WaitMs(8)
        ServoCart(0,x,y,z,rx,ry,rz,0,0,0,0,0,0,180,100,0.008,0,0)
    end
end
```

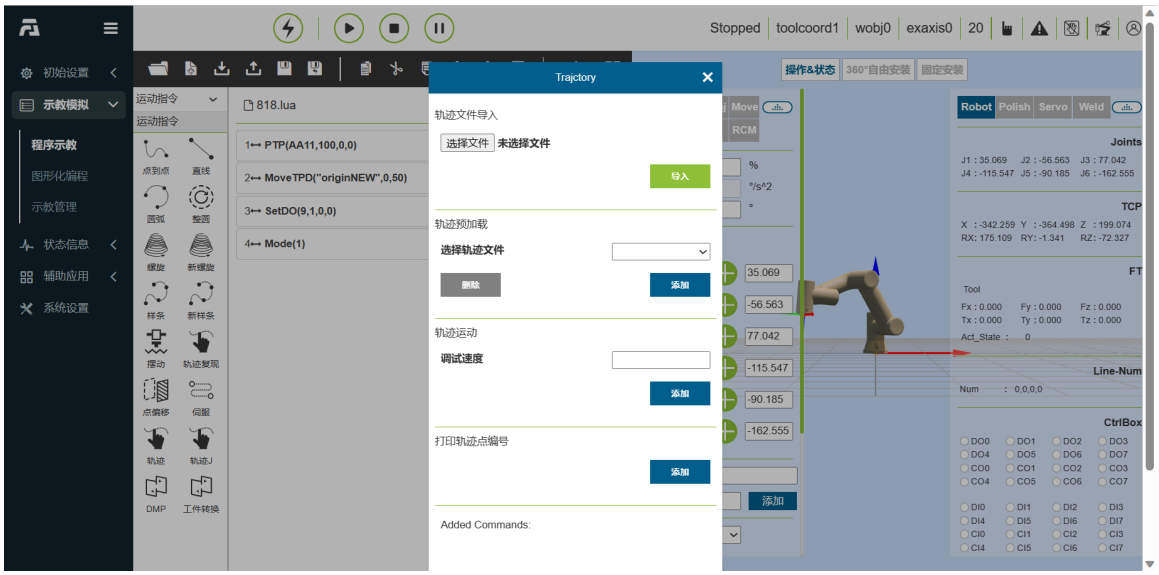
此例中，x,y,z,rx,ry,rz（笛卡尔位置）是获取的机器人当前位置，此外，用户可以通过读取轨迹数据文件，socket 通讯发送轨迹数据等方式，控制机器人运动。

基于当前位姿偏移（基坐标偏移）控制程序实例：

```
PTP(p1,100,0,0)
while 1 do
    WaitMs(8)
    ServoCart(1,0,1,-1,0,0,0,0.4,0.4,0.4,0.2,0.2,0.2,180,100,0.008,0,0)
end
```

1.3.5.7.5.13 轨迹命令

点击“轨迹”图标进入 Trajctory 命令编辑界面



图表 3.7-5-14 Trajctory 指令界面

1.3.5.7.5.14 轨迹 J 命令

点击“轨迹 J”图标进入 TrajctoryJ 命令编辑界面

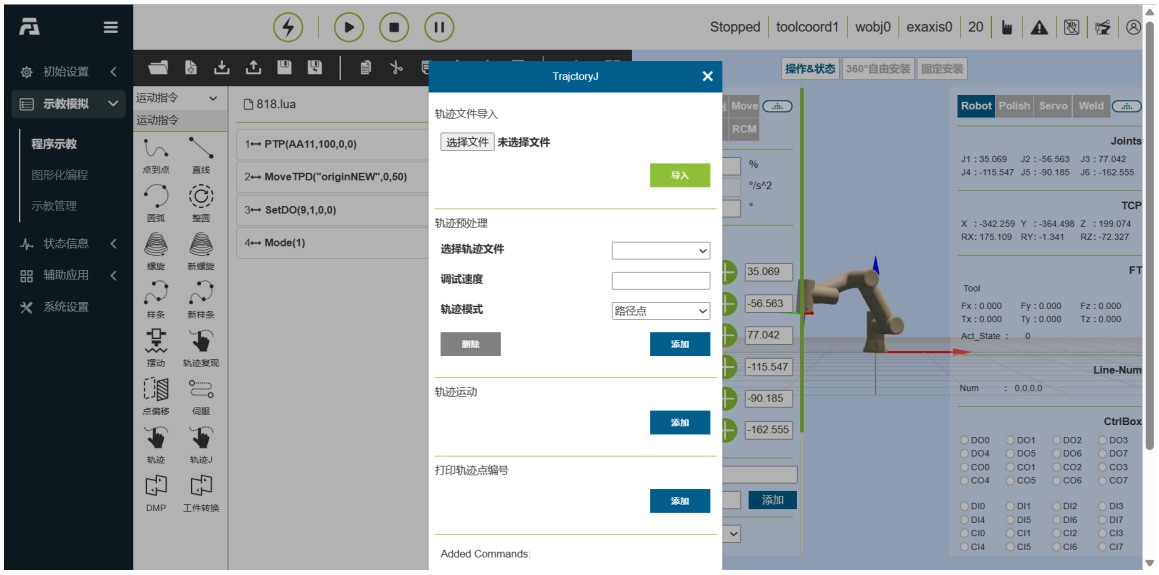
Trajctory 指令和 TrajctoryJ 指令适用于相机直接给定轨迹的通用接口，满足在已有固定格式的离散的轨迹点文件时，可导入系统使得机器人按照导入文件的轨迹进行运动。

1. 轨迹文件导入功能：选择本地计算机文件导入机器人控制系统

2. 轨迹预加载：选择已导入的轨迹文件通过指令加载

3. 轨迹运动：通过预加载的轨迹文件和选择的调试速度组合指令下发机器人运动

4. 打印轨迹点编号：在机器人运行轨迹的过程中打印轨迹点编号，以便查看当前运动的进度

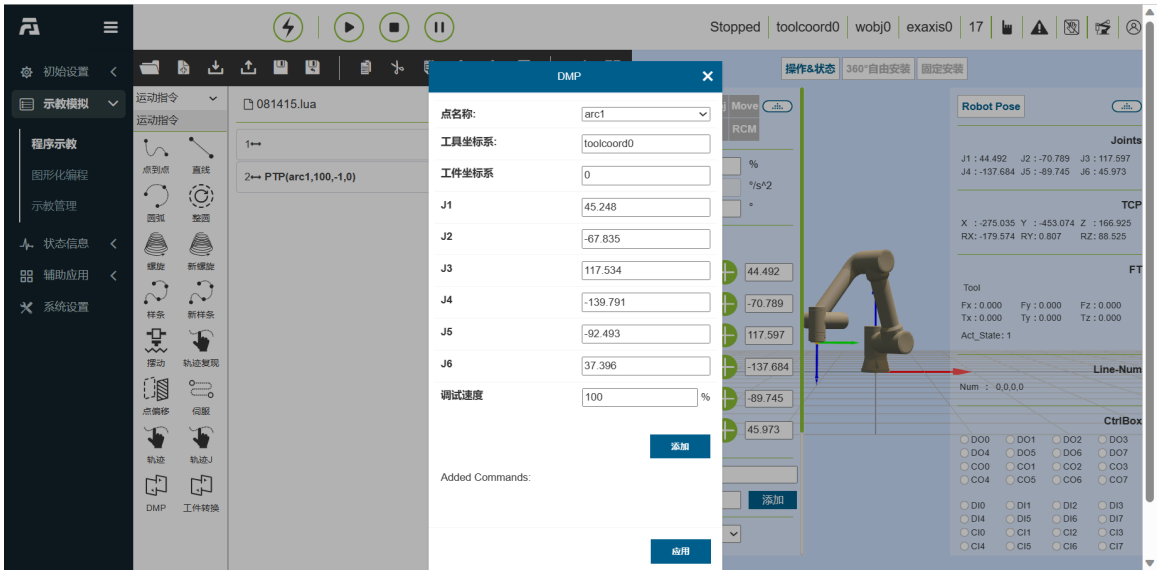


图表 3.7-5-15 TrajctoryJ 指令界面

1.3.5.7.5.15 DMP 命令

点击“DMP”图标进入 DMP 命令编辑界面

DMP 是一种轨迹模仿学习的方法，需要事先规划参考轨迹。在命令编辑界面，选择示教点作为新的起点，点击“添加”、“应用”后可保存该指令。DMP 具体路径为以新的起点模仿参考轨迹的新轨迹。

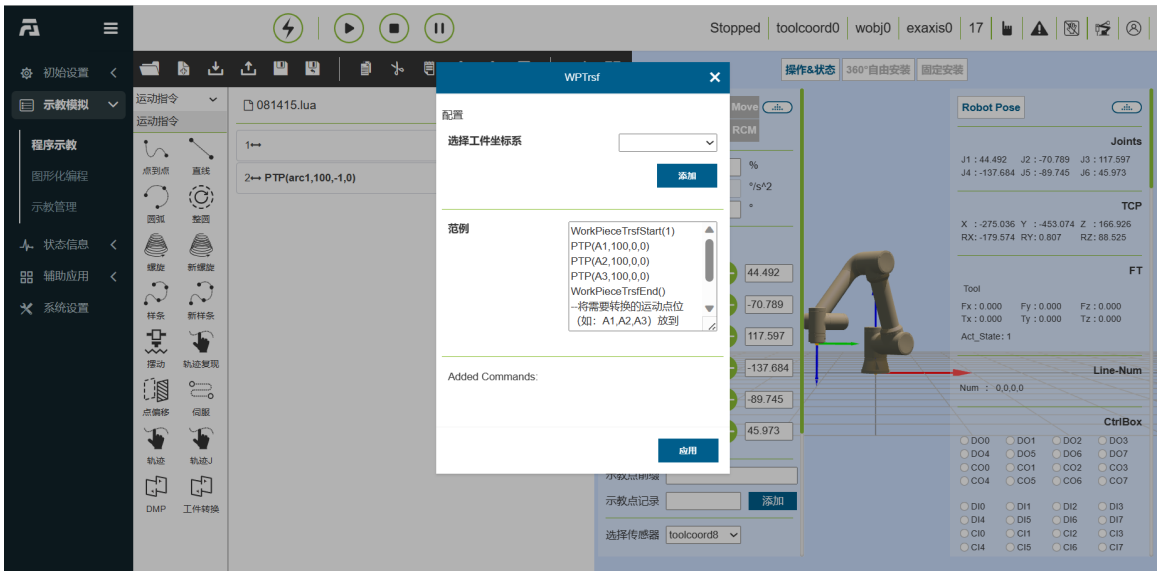


图表 3.7-5-16 DMP 指令界面

1.3.5.7.5.16 工件转换命令

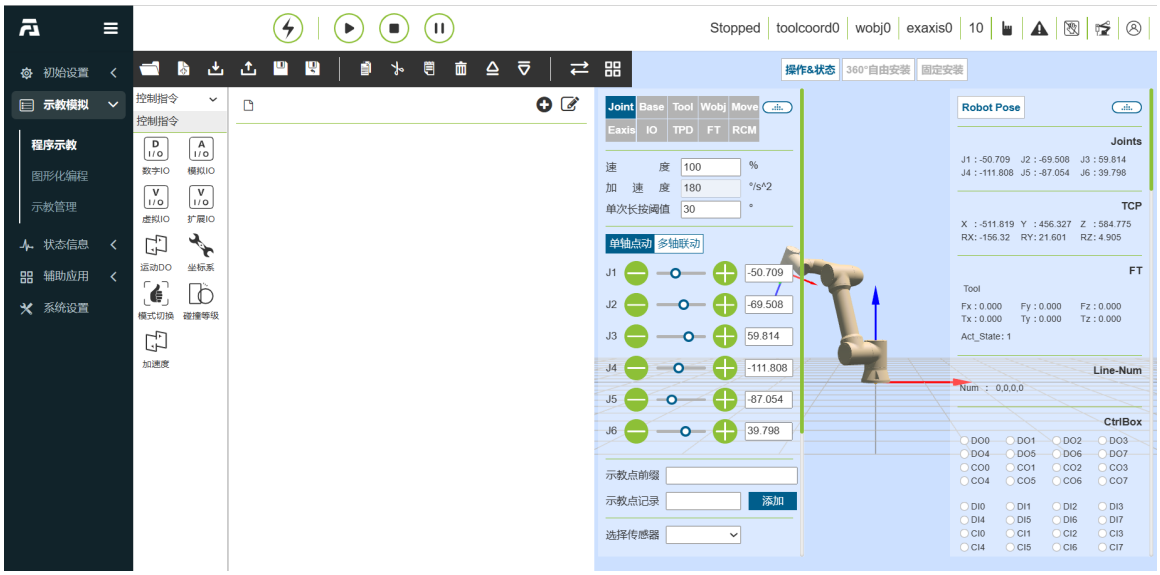
点击“工件转换”图标进入 WPTrsf 命令编辑界面

选择所要进行自动转换的工件坐标系，点击“添加”、“应用”后可保存该指令，该指令实现在执行内部的 PTP、LIN 指令时，工件坐标系下点位自动转换。使用示例区域展示并提示了指令的正确使用方式组合，具体指令在添加后可依据实际场景自行调整组合。



图表 3.7-5-17 WPTrsf 指令界面

1.3.5.7.6 控制指令界面



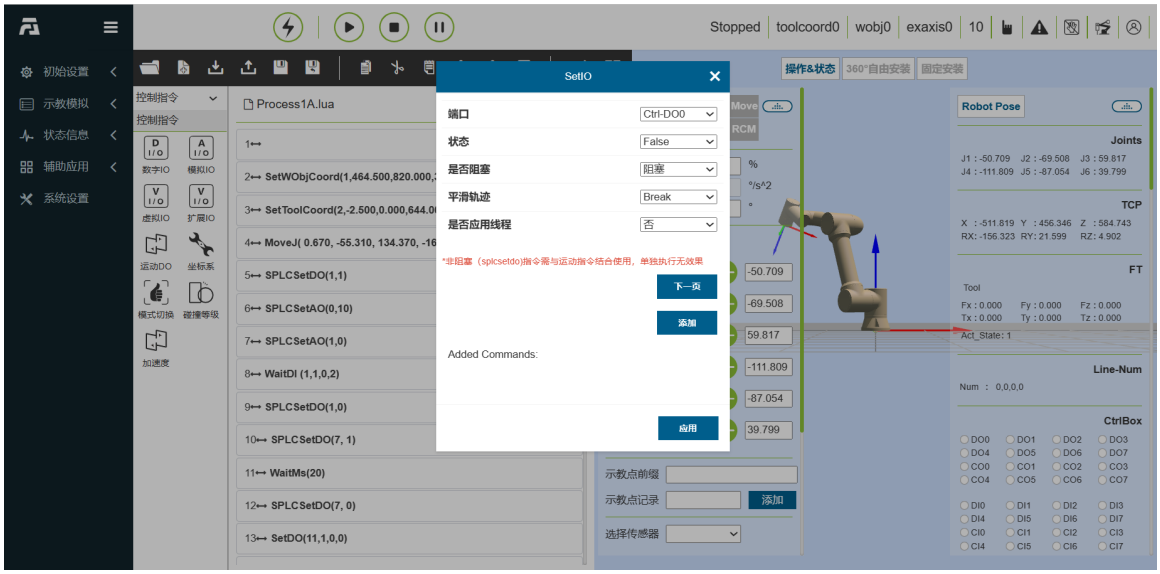
图表 3.7-6 控制指令界面

1.3.5.7.6.1 数字 IO 命令

点击“数字 IO”图标进入 IO 命令编辑界面

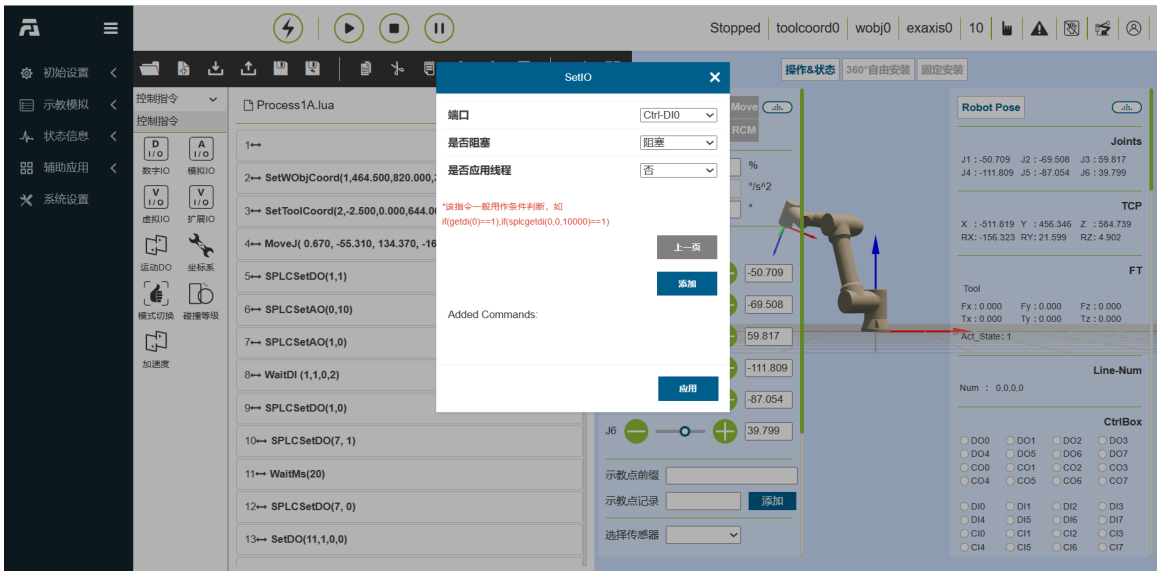
“IO”指令分为设置 IO（SetDO/SPLCSetDO）和获取 IO（GetDI/SPLCGetDI）两部分。

“SetDO/SPLCSetDO”该指令可设定指定的输出 DO 状态，包括 16 路控制箱数字输出和 2 路工具数字输出，状态选项“False”为闭，“True”为开，是否阻塞选项选择“阻塞”表示运动停止后设置 DO 状态，选择“非阻塞”选项表示在上一条运动过程中设置 DO 状态。平滑轨迹选项选择“Break”表示在平滑过渡半径结束后设置 DO 状态，选择“Serious”表示在平滑过渡半径运动过程中设置 DO 状态。当该指令是添加在辅助线程中，是否应用线程需要选择是，其他地方使用该指令选择否。点击“添加”、“应用”即可。



图表 3.7-6-1 SetDO 指令界面

在“GetDI/SPLCGetDI”指令中，选择想要获取端口号的数值，是否阻塞选项选择“阻塞”表示运动停止后获取 DI 状态，选择“非阻塞”选项表示在上一条运动过程中获取 DI 状态。当该指令是添加在辅助线程中，是否应用线程需要选择是，其他地方使用该指令选择否。选择完毕后点击“添加”、“应用”按钮即可。



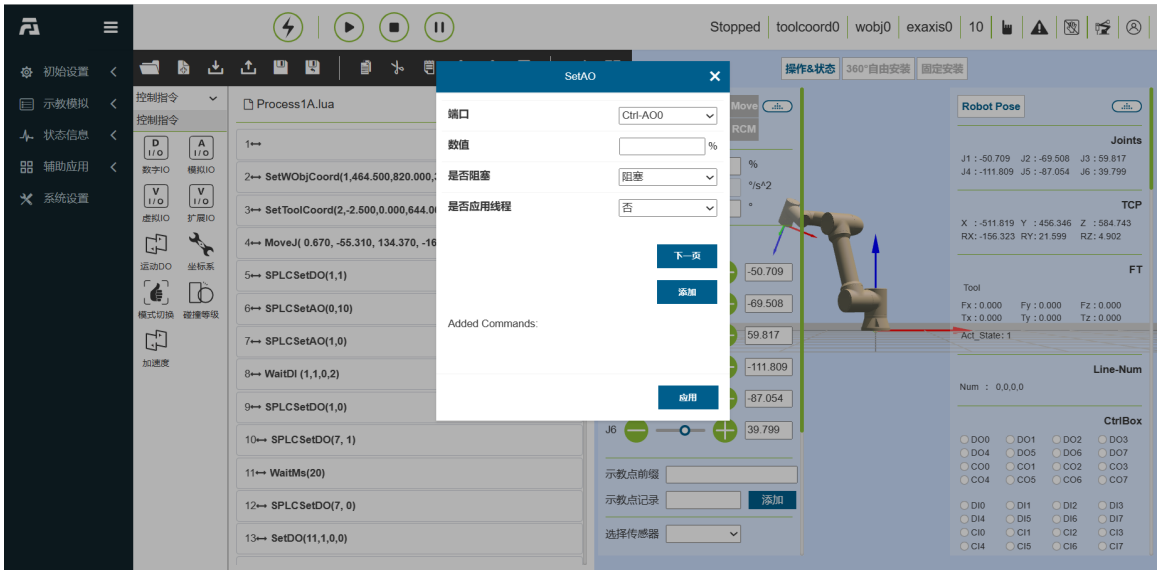
图表 3.7-6-2 GetDI 指令界面

1.3.5.7.6.2 模拟 AI 命令

点击“模拟 AI”图标进入 AI 命令编辑界面

在该指令中，分为设置模拟输出（SetAO/SPLCSetAO）和获取模拟输入（GetAI/SPLCGetAI）两部分功能。

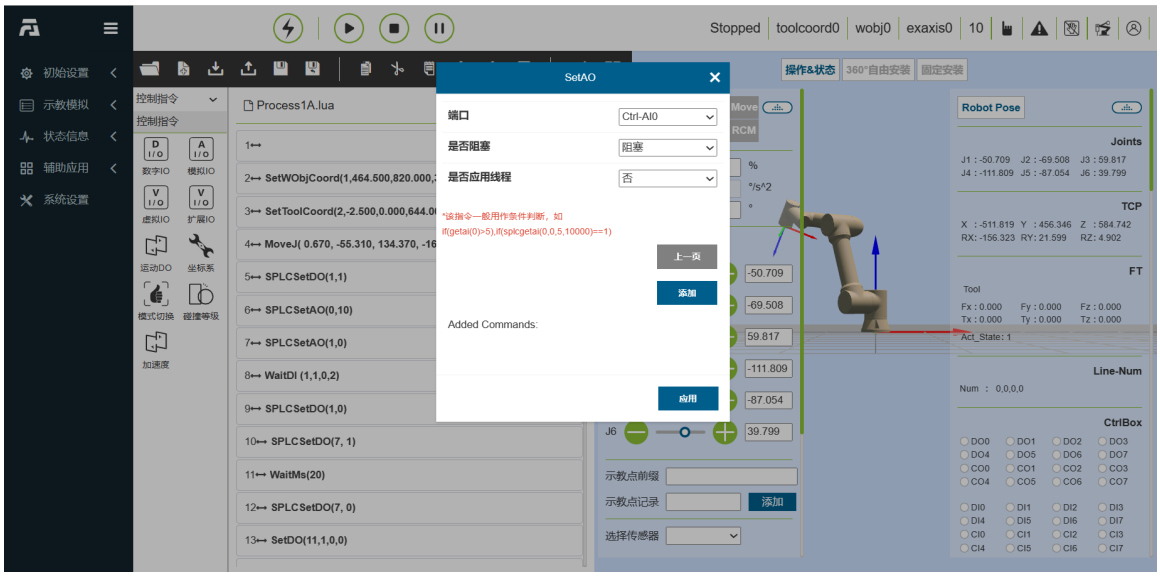
“SetAO/SPLCSetAO”选择需要设置的模拟输出，输入需要设置的值，范围为 0-10，是否阻塞选项选择“阻塞”表示运动停止后设置 AO 状态，选择“非阻塞”选项表示在上一条运动过程中设置 AO 状态。当该指令是添加在辅助线程中，是否应用线程需要选择是，其他地方使用该指令选择否。点击“添加”、“应用”即可。



图表 3.7-6-3 SetAO 指令界面

“GetAI/SPLCGetAI”选择需要获取的模拟输入，是否阻塞选项选择“阻塞”表示运动停止后获取 AI 状态，

选择“非阻塞”选项表示在上一条运动过程中获取 AI 状态。当该指令是添加在辅助线程中，是否应用线程需要选择是，其他地方使用该指令选择否。点击“添加”、“应用”即可。

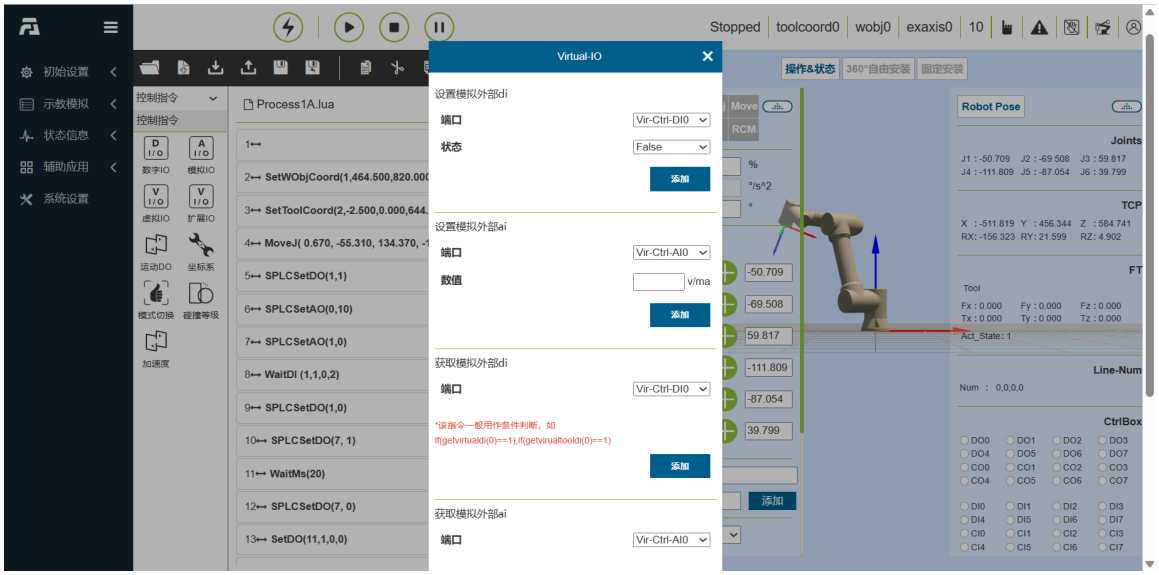


图表 3.7-6-4 GetAI 指令界面

1.3.5.7.6.3 虚拟 IO 命令

点击“虚拟 IO”图标进入 Vir-IO 命令编辑界面

该指令虚拟的 IO 控制指令，可以实现设置模拟外部 DI 和 AI 状态，获取模拟 DI 和 AI 状态。

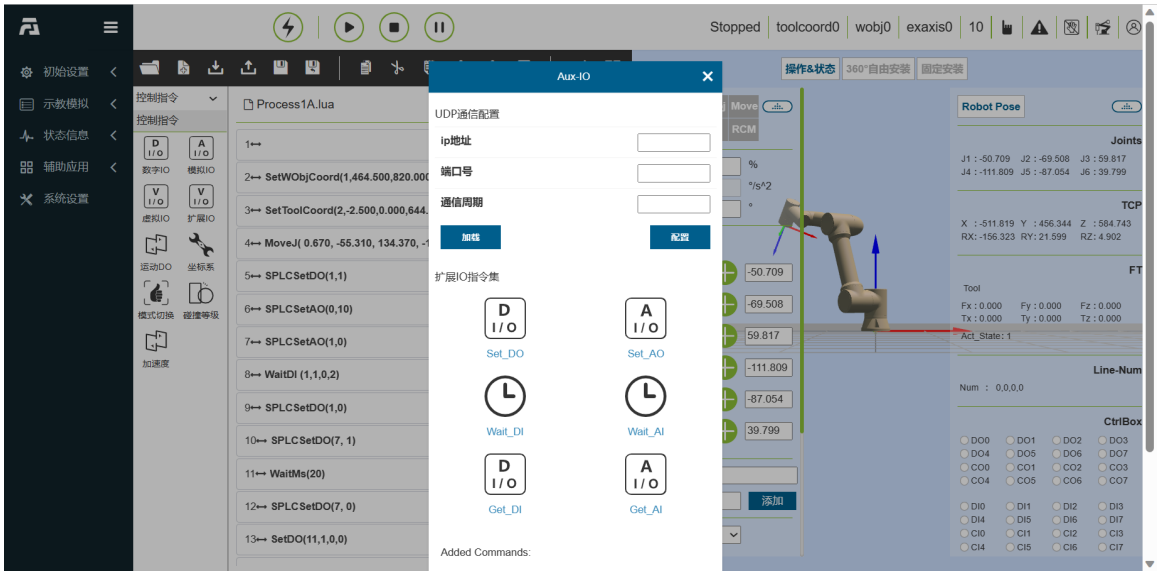


图表 3.7-6-5 Vir-IO 指令界面

1.3.5.7.6.4 扩展 IO 命令

点击“扩展 IO”图标进入 Aux-IO 命令编辑界面

Aux-IO 是机器人与 PLC 通讯控制外部扩展 IO 的指令功能，需要机器人与 PLC 建立 UDP 通讯，在原有的 16 路输入输出基础上，可以扩展 128 路输入输出，该指令用法与前文所讲的通用 IO 用法类似。使用此功能，有一定技术难度，前请联系我们咨询。

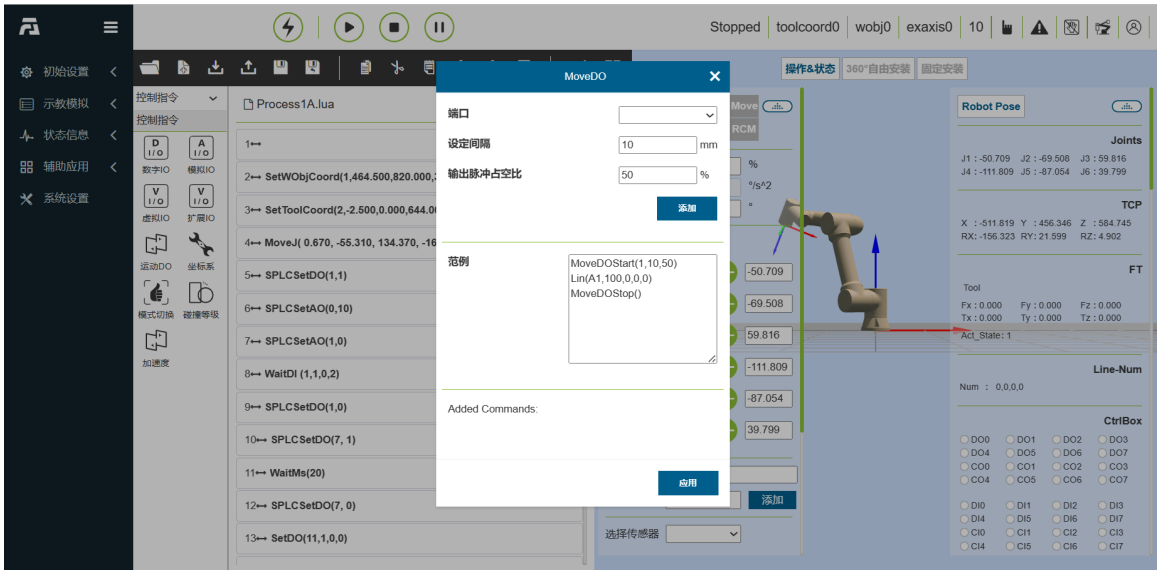


图表 3.7-6-6 Aux-IO 指令界面

1.3.5.7.6.5 运动 DO 命令

点击“运动 DO”图标进入 MoveDO 命令编辑界面

该指令实现直线运动过程中，根据设定的间隔，连续输出 DO 信号功能。

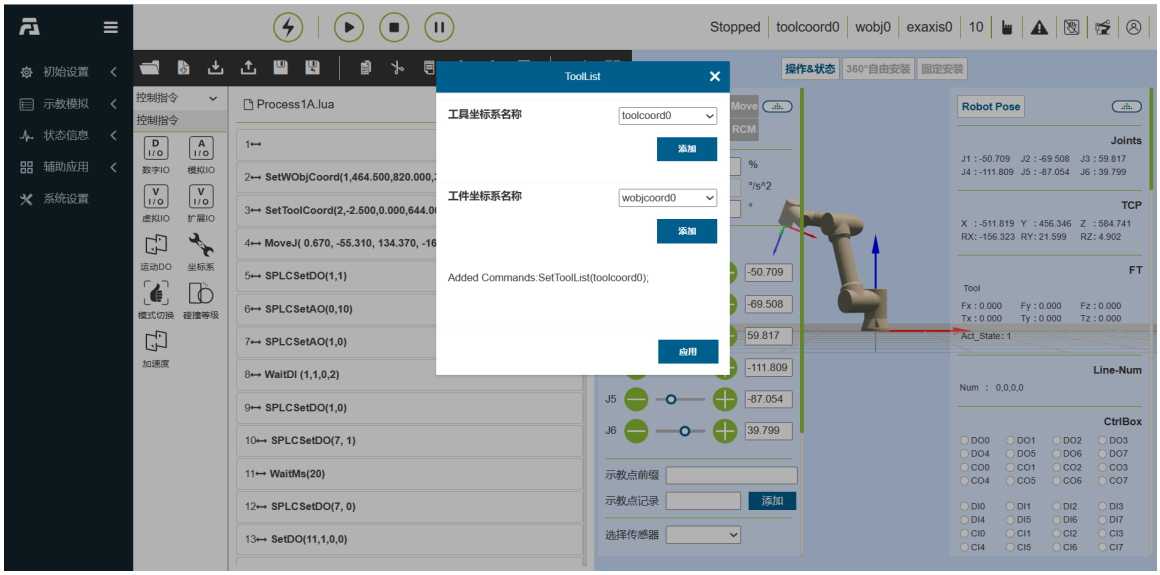


图表 3.7-6-7 MoveDO 指令界面

1.3.5.7.6.6 坐标系命令

点击“坐标系”图标进入 ToolList 命令编辑界面

选择工具坐标系名称，点击“应用”添加该指令到程序中，当程序运行该语句，会设定机器人的工具坐标系。

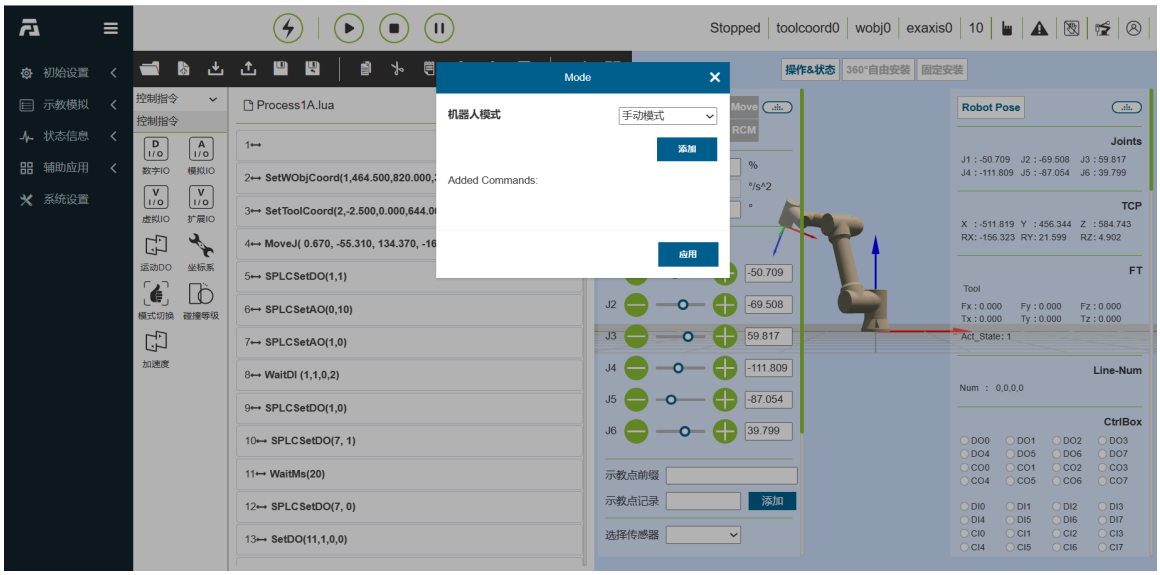


图表 3.7-6-8 ToolList 指令界面

1.3.5.7.6.7 模式切换命令

点击“模式切换”图标进入 Mode 命令编辑界面

该指令可切换机器人到手动模式，通常在一个程序结尾处添加，以便用户在程序运行结束后，使机器人自动切换到手动模式，拖动机器人。

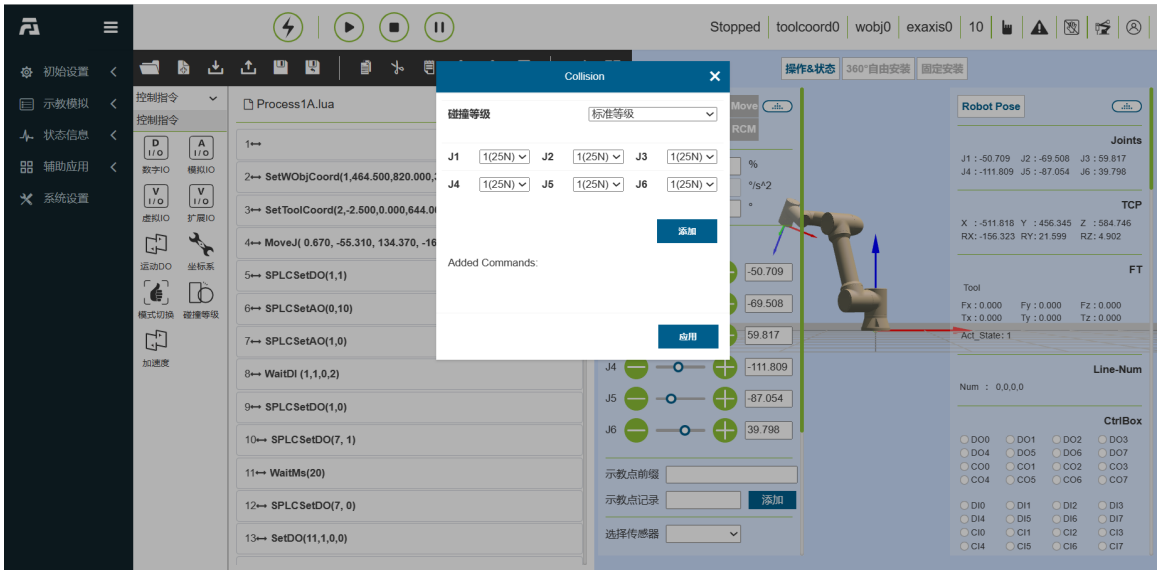


图表 3.7-6-9 Mode 指令界面

1.3.5.7.6.8 碰撞等级命令

点击“碰撞等级”图标进入 Collision 命令编辑界面

该指令碰撞等级设置，通过该指令可以在程序运行中实时调节各轴碰撞等级，更灵活的部署应用场景。

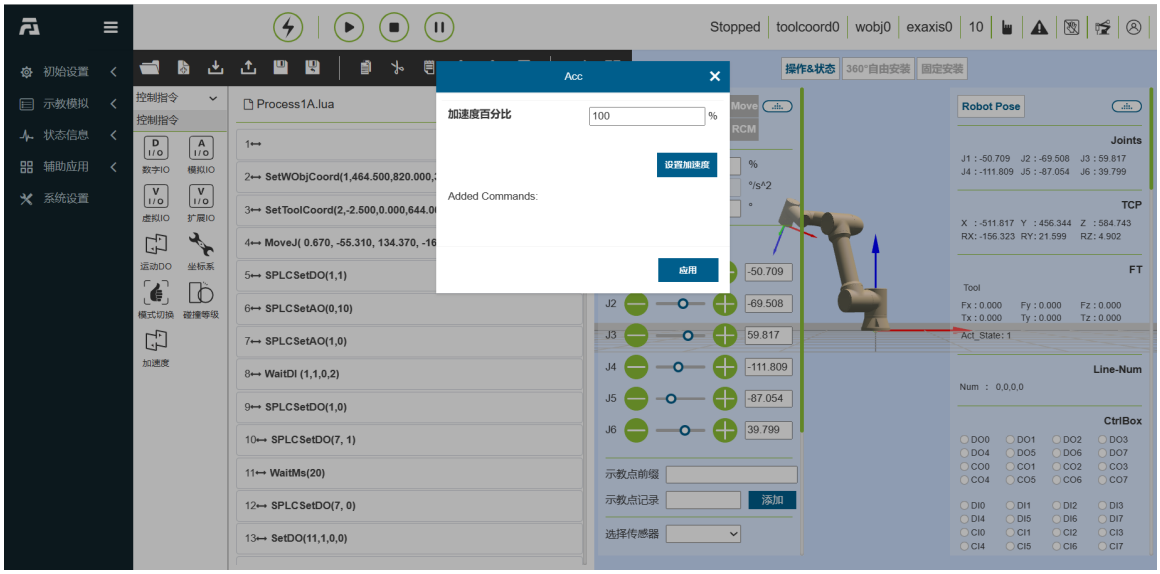


图表 3.7-6-10 Collision 指令界面

1.3.5.7.6.9 加速度命令

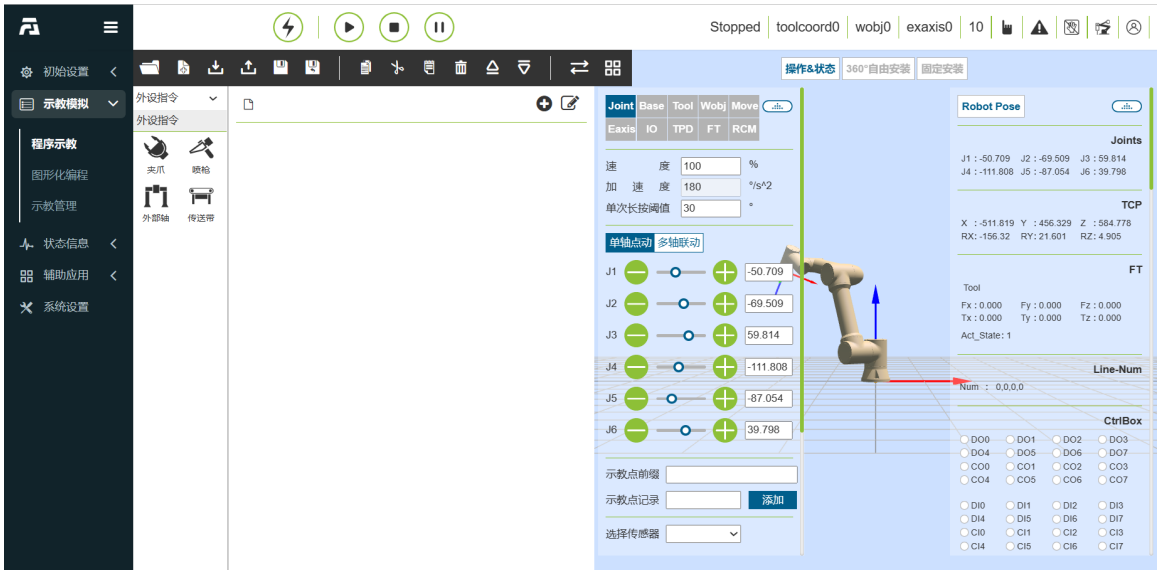
点击“加速度”图标进入 Acc 命令编辑界面

Acc 指令是实现机器人加速度可单独设置功能，通过调节运动指令加速度缩放因子，可以增加或减小加减速时间，实现机器人动作节拍时间可调。



图表 3.7-6-11 Acc 指令界面

1.3.5.7.7 外设指令界面

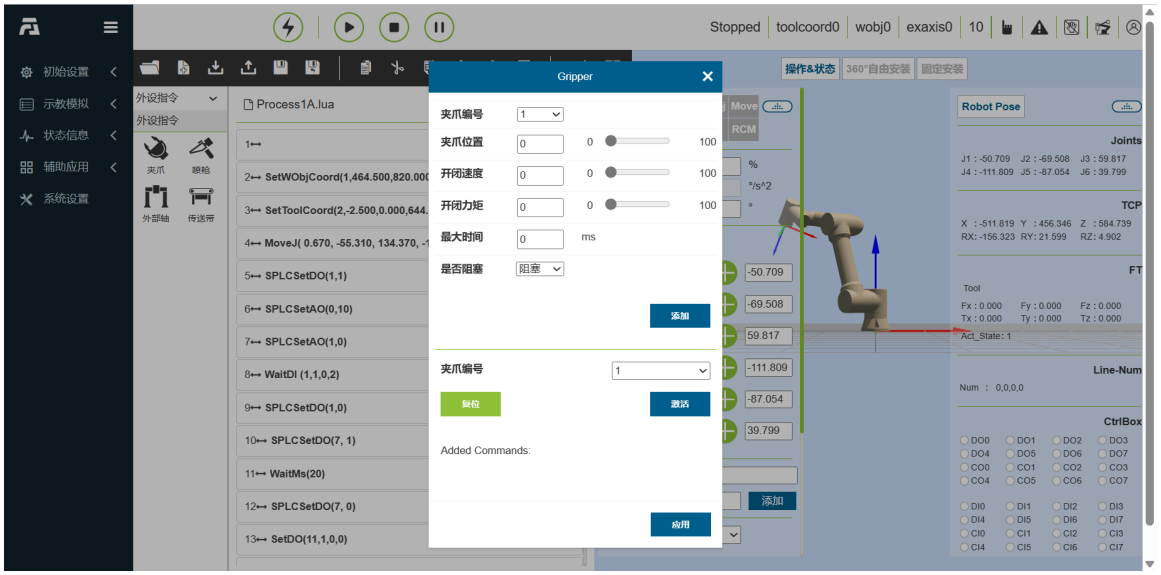


图表 3.7-7 外设指令界面

1.3.5.7.7.1 夹爪命令

点击“夹爪”图标进入 Gripper 命令编辑界面

在该指令中，分为夹爪运动控制指令和夹爪激活/复位指令，夹爪控制指令中，显示完成配置并且已被激活的夹爪编号，用户可以通过编辑框编辑，或者滑动条滑动至所需的值来完成对夹爪开闭、开闭速度和开闭力矩的设置，数值为百分比，是否阻塞功能选项，选择阻塞即夹爪运动需等待上一条运动指令执行完才执行，选择非阻塞即夹爪运动与上一条运动指令并行。点击“添加”、“应用”按钮，即可将设置的值保存至示教文件中。夹爪复位/激活指令，显示已经配置的夹爪编号，可以添加复位/激活指令到程序中。

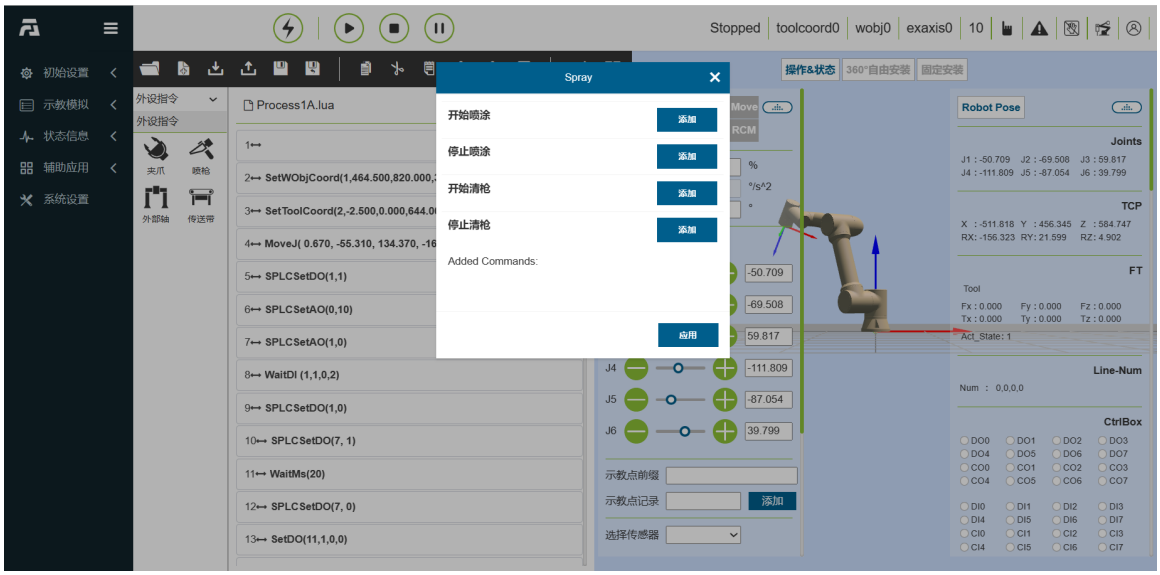


图表 3.7-7-1 Gripper 指令界面

1.3.5.7.7.2 喷枪命令

点击“喷枪”图标进入 Spray 命令编辑界面

该指令为喷涂相关指令，控制喷枪“开始喷涂”、“停止喷涂”、“开始清枪”和“停止轻枪”。在编辑该程序命令时，需确认已经配置好喷枪外设，详见机器人外设章节。

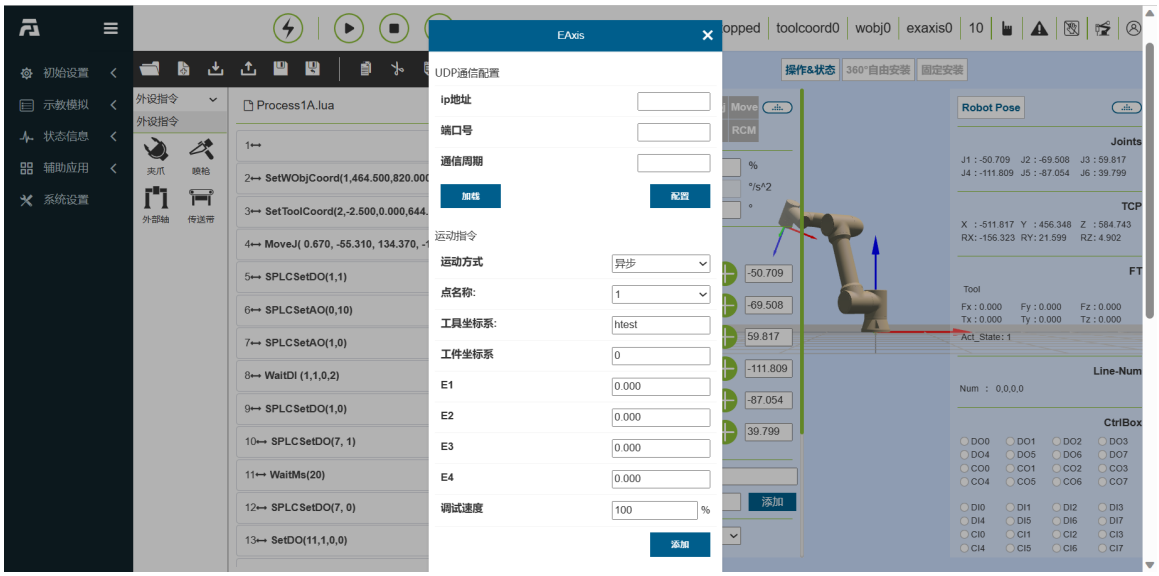


图表 3.7-7-2 Spray 指令界面

1.3.5.7.7.3 外部轴命令

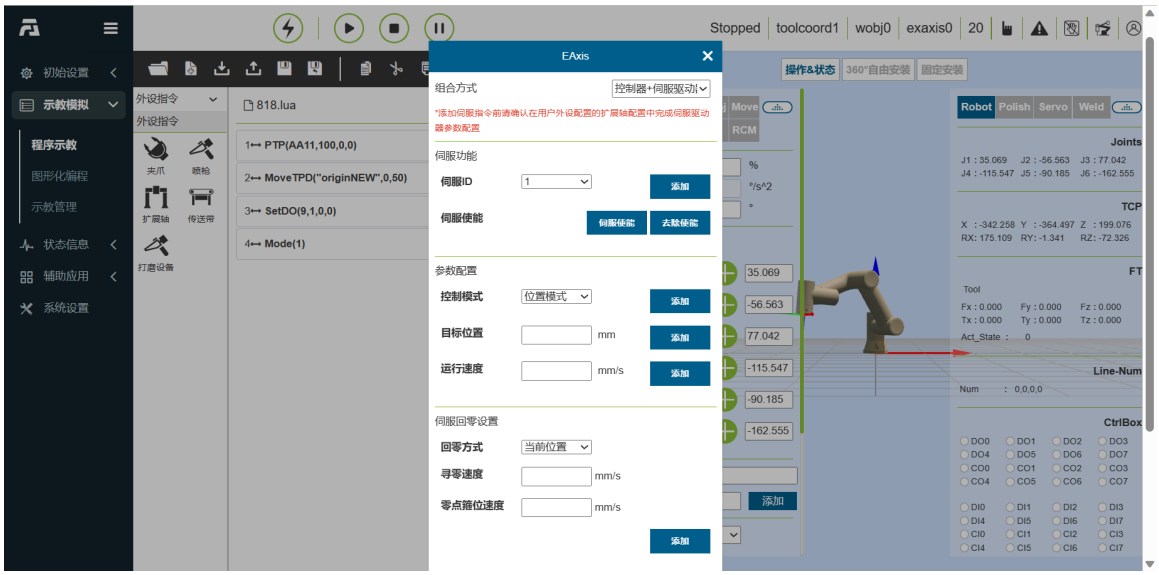
点击“外部轴”图标进入 EAxis 命令编辑界面，选择组合模式：控制器 + 伺服驱动器 (485)/控制器 +PLC(UDP)

选择控制器 +PLC(UDP), 该指令针对使用外部轴的场景，与 PTP 指令组合使用，可将空间上一点 X 轴方向上的移动分解到外部轴运动。选择外部轴编号，运动方式选同步，选择需要到达的点，点击“添加”、“应用”后可保存该条指令。



图表 3.7-7-3 EAxis 指令界面

选择控制器 + 伺服驱动器 (485), 该指令可对扩展轴参数进行配置。根据不同的控制模式设置不同的参数。已配置好的扩展轴，可对其零点设定。

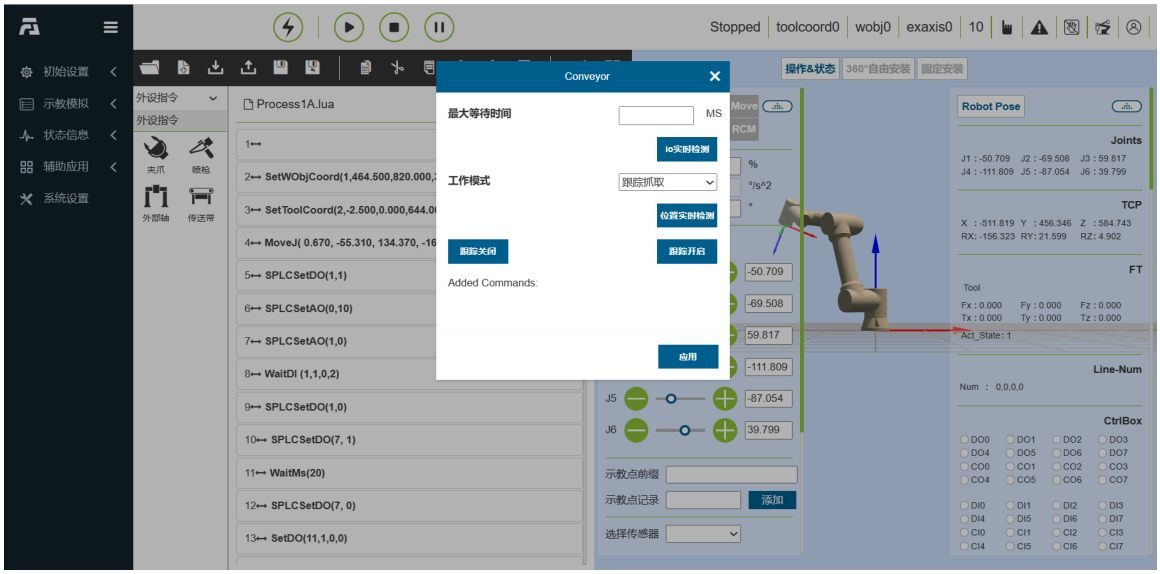


图表 3.7-7-4 扩展轴指令界面

1.3.5.7.7.4 传送带命令

点击“传送带”图标进入 Convey 命令编辑界面

该指令包含位置实时检测，IO 实时检测，跟踪开启和跟踪关闭四条命令。详见机器人外设章节。

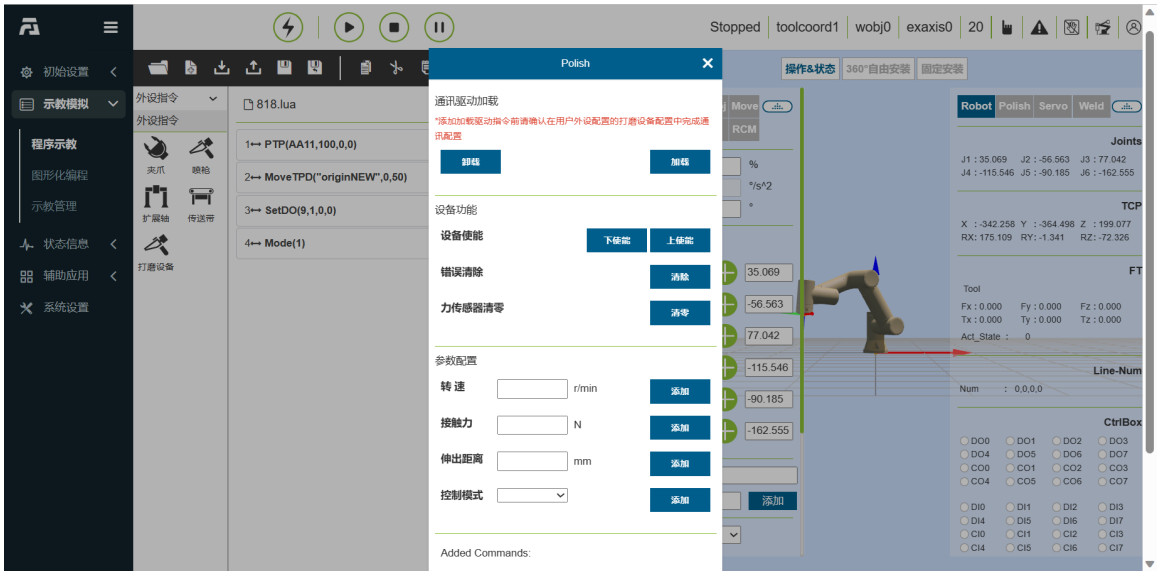


图表 3.7-7-5 Conveyor 指令界面

1.3.5.7.7.5 打磨设备命令

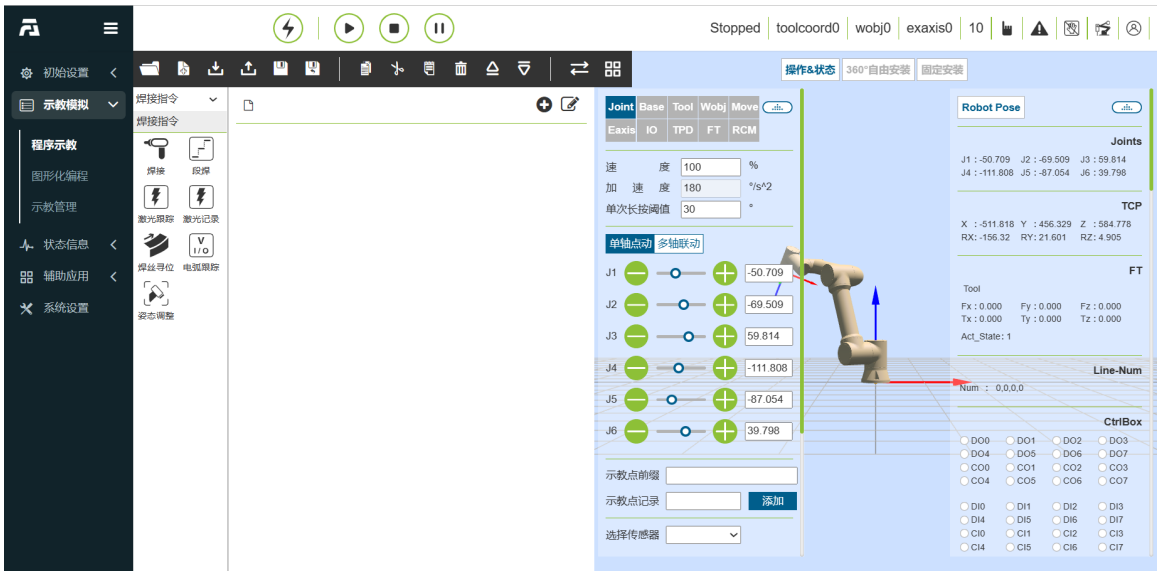
点击“打磨设备”图标进入 Polish 命令编辑界面

该指令可设置打磨设备的转速、接触力、伸出距离和控制模式。



图表 3.7-7-6 Polish 命令界面

1.3.5.7.8 焊接指令界面

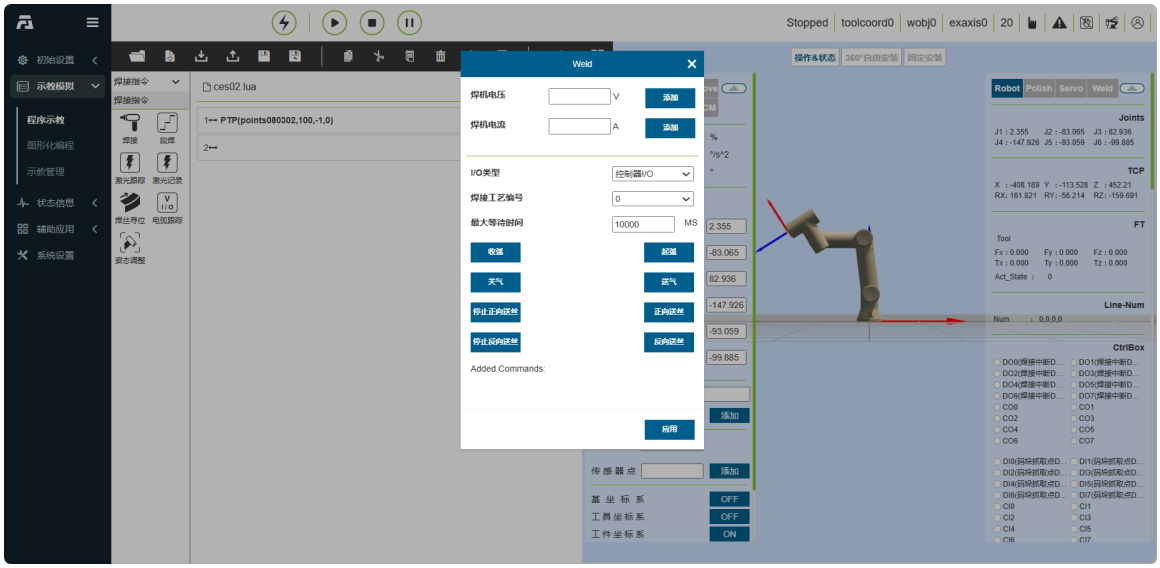


图表 3.7-8 焊接指令界面

1.3.5.7.8.1 焊接命令

点击“焊接”图标进入 Weld 命令编辑界面

该指令主要用于焊机外设，在添加该指令前请确认在用户外设中焊机配置是否完成，详见机器人外设章节。

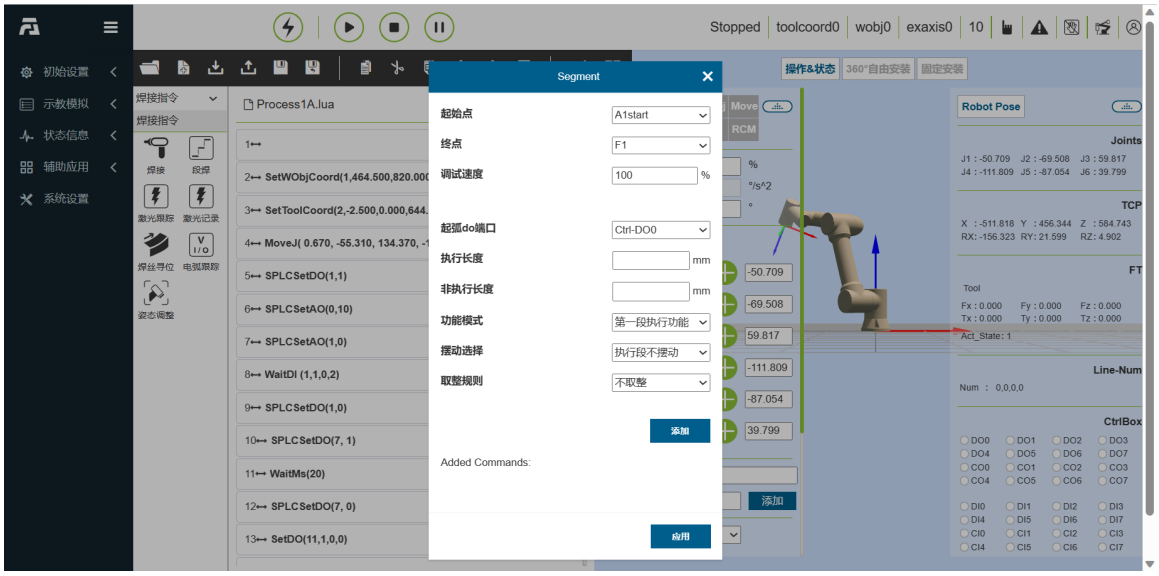


图表 3.7-8-1 Weld 指令界面

1.3.5.7.8.2 段焊命令

点击“段焊”图标进入 Segment 命令编辑界面

该指令为焊接专用指令，主要用于一段焊，一段不焊的循环间断焊接场景。在起点与终点之间，使用该指令，选择起点和终点，设置调试速度，设置起弧的 DO 端口，执行长度，非执行长度，根据实际应用场景设置功能模式，摆动选择和取整规则即可实现段焊功能。

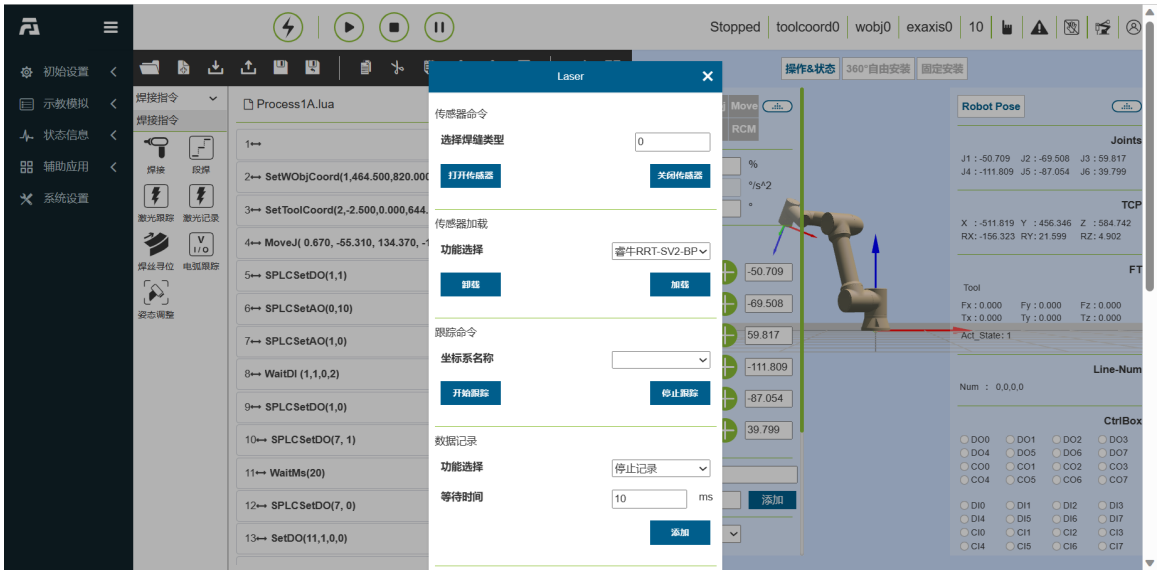


图表 3.7-8-2 Segment 指令界面

1.3.5.7.8.3 激光跟踪命令

点击“激光跟踪”图标进入 Laser 命令编辑界面

该指令包含激光命令、跟踪命令和寻位命令三部分，在添加该指令前，请确认用户外设中激光跟踪传感器是否已经配置成功。详见机器人外设章节。



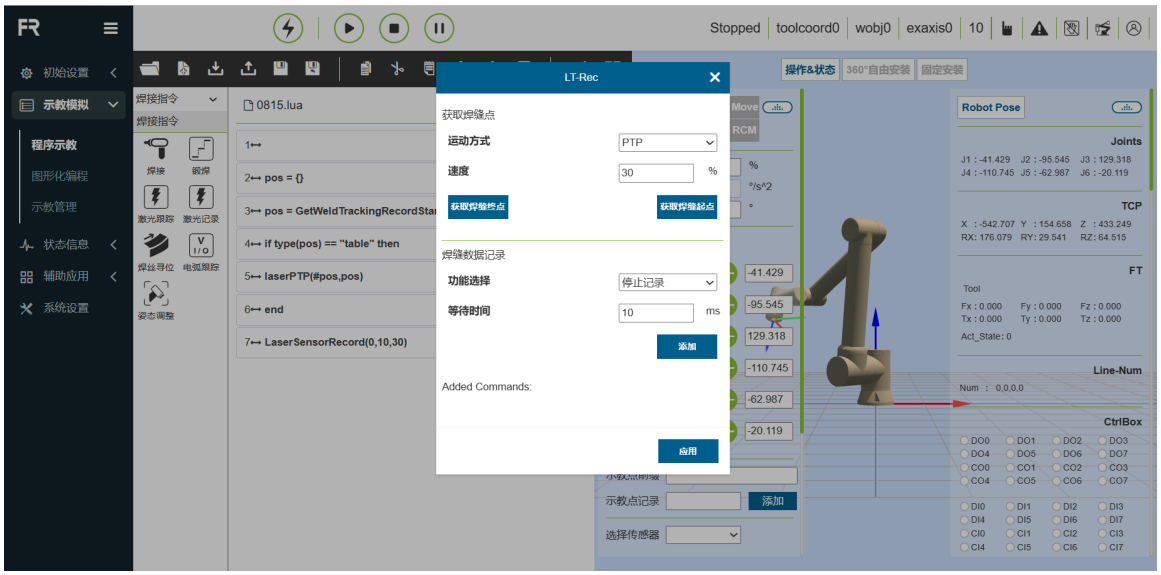
图表 3.7-8-3 Laser 指令界面

1.3.5.7.8.4 激光记录命令

点击“激光记录”图标进入 LT-Rec 命令编辑界面

该指令实现激光跟踪记录起点、终点取出功能，使机器人可以自动运动到起点位置，适用于从工件外部开始运动并进行激光跟踪记录的场合，同时上位机可获取记录数据中起点、终点的信息，用于后续运动。

实现激光跟踪复现速度可调功能，使机器人可以用一个很快的速度进行记录，然后按照正常焊接速度进行复现，可以提高作业效率。

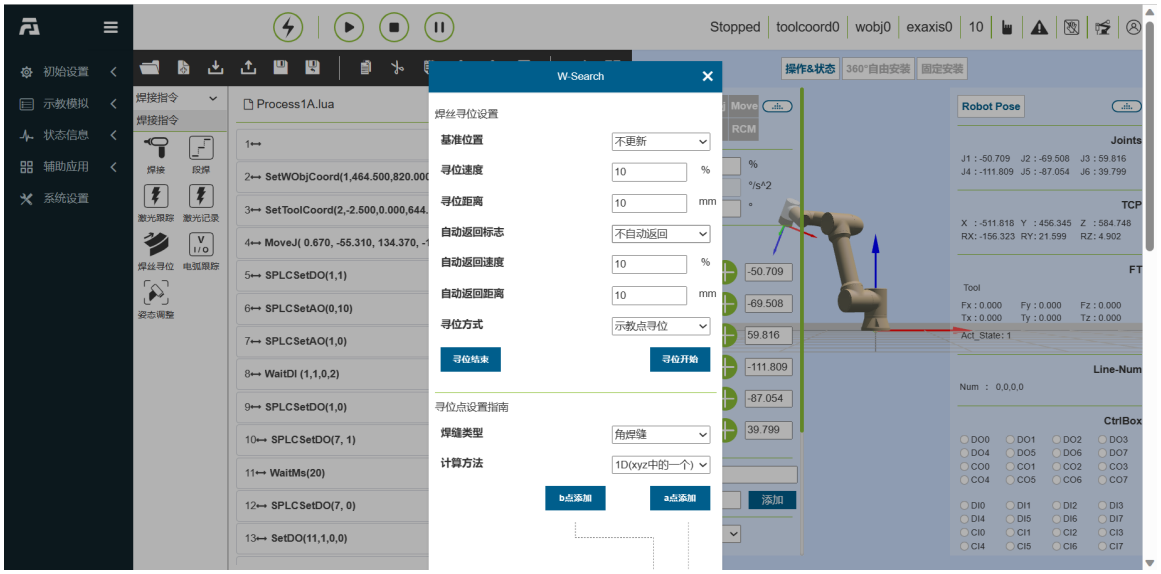


图表 3.7-8-4 LT-Rec 指令界面

1.3.5.7.8.5 焊丝寻位命令

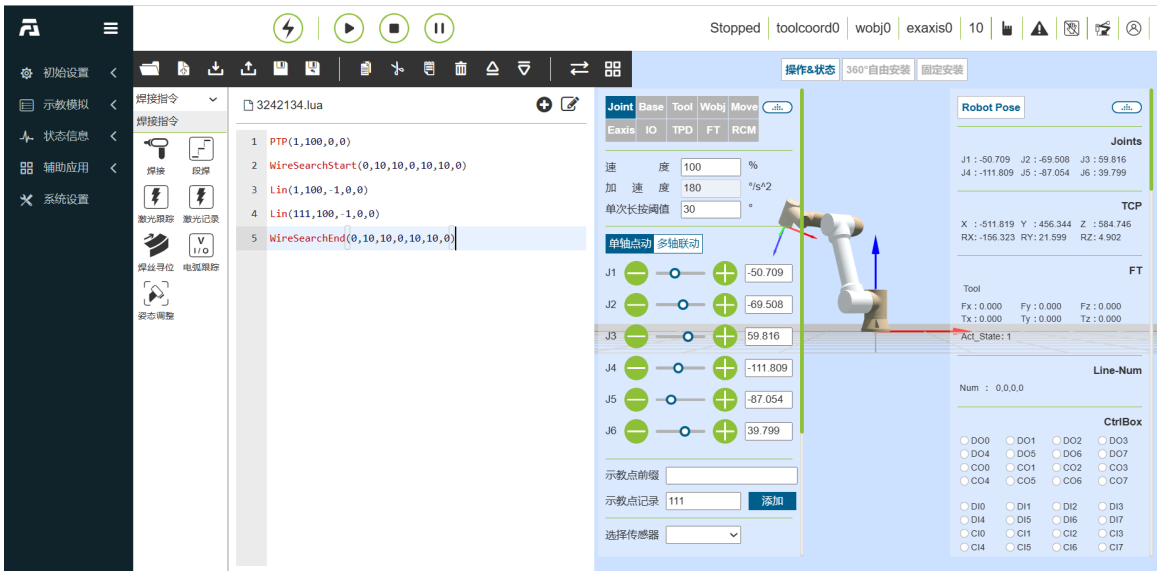
点击“焊丝寻位”图标进入 W-Search 命令编辑界面

该指令为焊丝寻位指令，包含寻位开始，寻位结束和计算偏移量三个指令，该指令一般应用于焊接场景中，需要焊机与机器人 IO 和运动指令相结合使用。



图表 3.7-8-5 W-Search 指令界面

在编写程序中，通常先设置寻位开始指令，之后添加两条 LIN 指令，确定寻位的方向，寻位成功后，获取计算出来的偏移量，将该偏移量通过整体偏移指令，生效到真正的焊接运动指令中，程序示例如下。



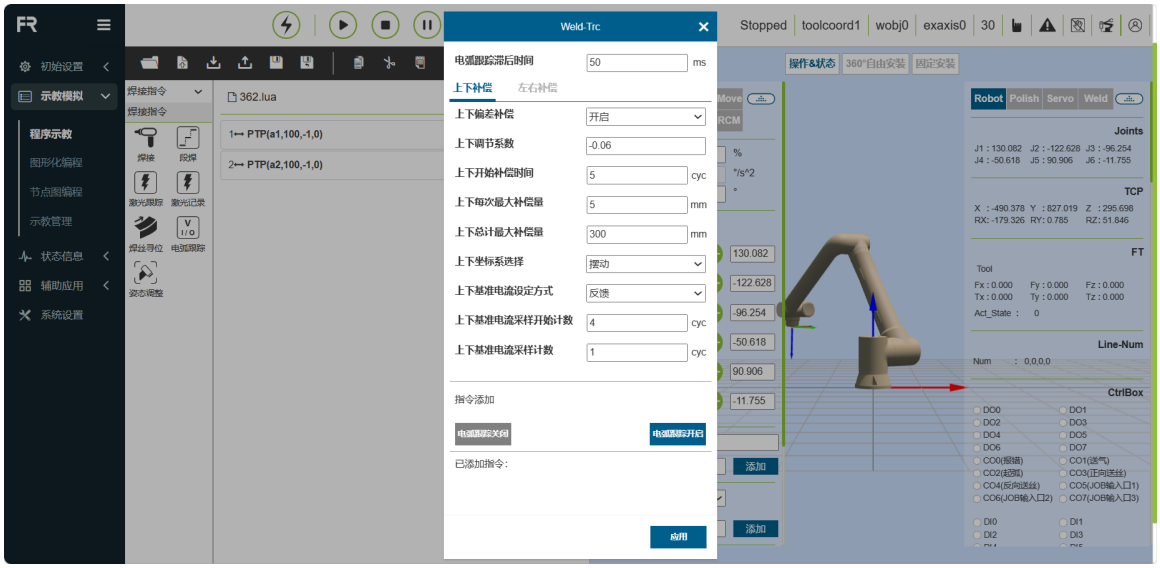
图表 3.7-8-6 W-Search 示例（1D）

1.3.5.7.8.6 电弧跟踪命令

点击“电弧跟踪”图标进入 Weld-Trc 命令编辑界面

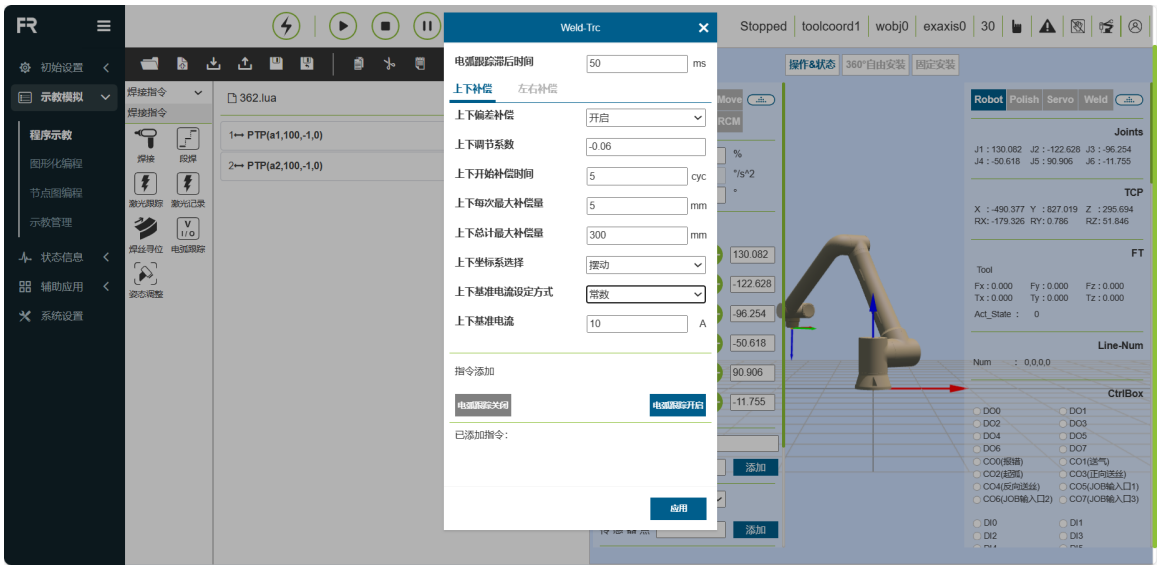
该指令实现机器人焊缝跟踪利用焊缝的偏差检测进行补偿轨迹，可以使用电弧传感器来检测焊缝偏差。

Step1：上下补偿基准电流设定方式：反馈，设置上下基准电流开始计数和上下基准电流计数



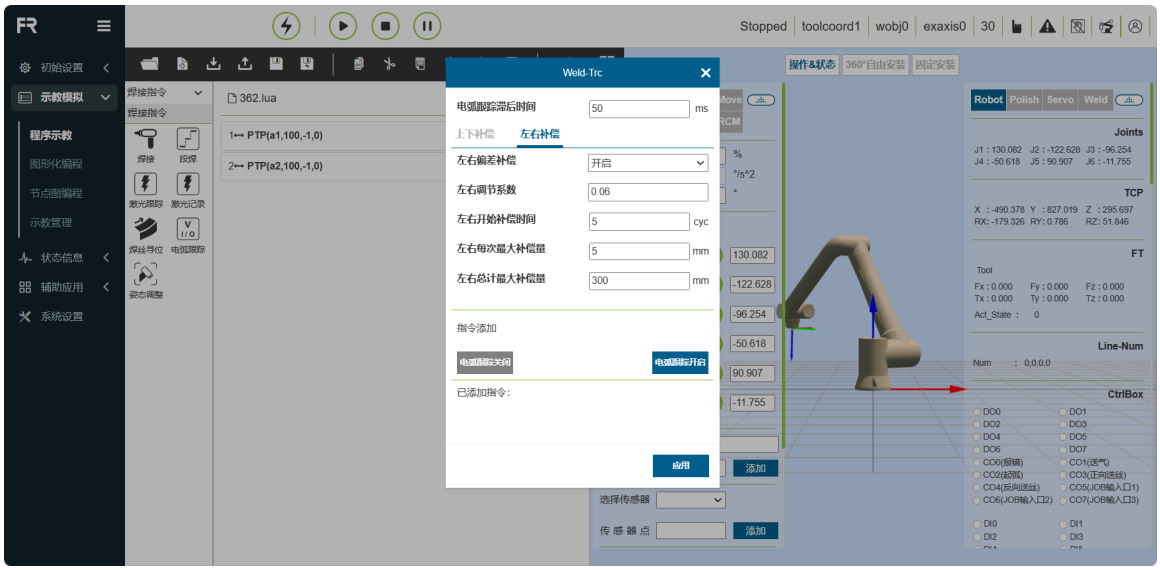
图表 3.7-8-7-1 Weld-Trc 指令界面-反馈

Step2：上下补偿基准电流设定方式：常数，设置上下基准电流



图表 3.7-8-7-2 Weld-Trc 指令界面-常数

Step3：左右补偿参数交互页面

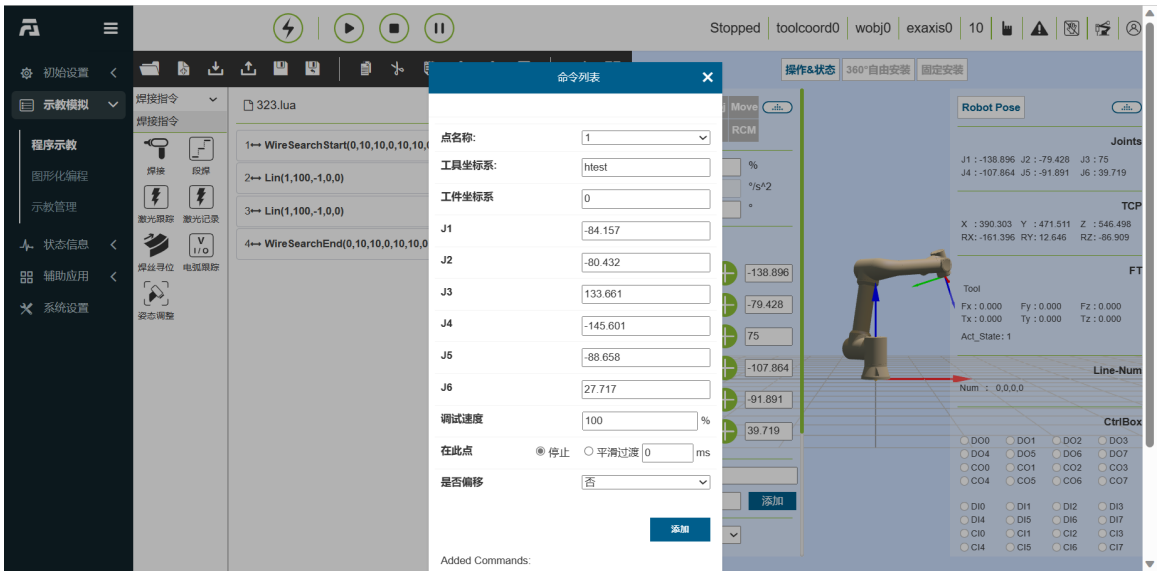


图表 3.7-8-7-3 Weld-Trc 指令界面-左右补偿参数

1.3.5.7.8.7 姿态调整命令

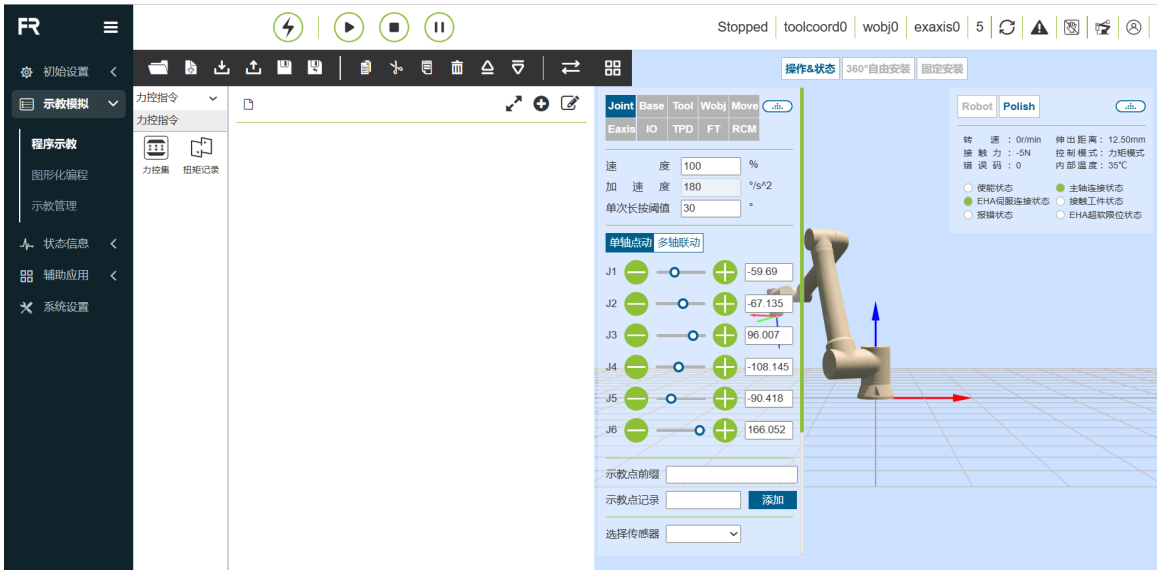
点击“姿态调整”图标进入 Adjust 命令编辑界面

该指令针对焊接跟踪自适应调整焊枪姿态场景，记录好三个对应的姿态点后，根据机器人实际运动方向，添加姿态自适应调整指令。详见机器人外设章节。



图表 3.7-8-8 Adjust 指令界面

1.3.5.7.9 力控指令界面

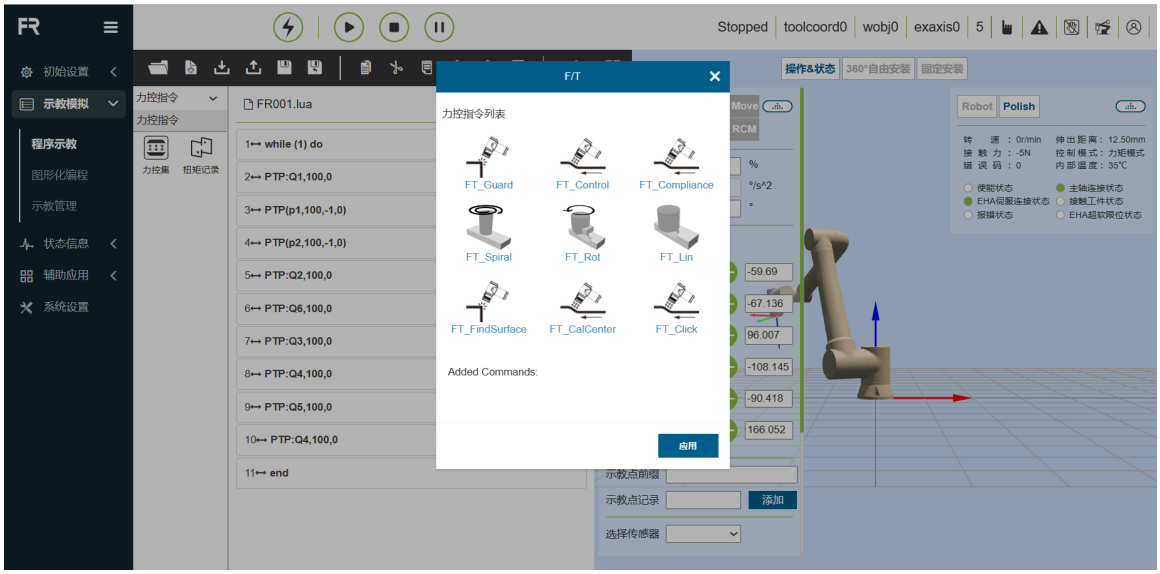


图表 3.7-9 力控指令界面

1.3.5.7.9.1 力控集命令

点击“力控集”图标进入 F/T 命令编辑界面

该指令包含 FT_Guard(碰撞检测)，FT_Control(恒力控制)，FT_Spiral(螺旋插入)，FT_Rot(旋转插入)，FT_Lin(直线插入)，FT_FindSurface(表面定位)，FT_CalCenter(中心定位) 七个指令，详见机器人外设章节。

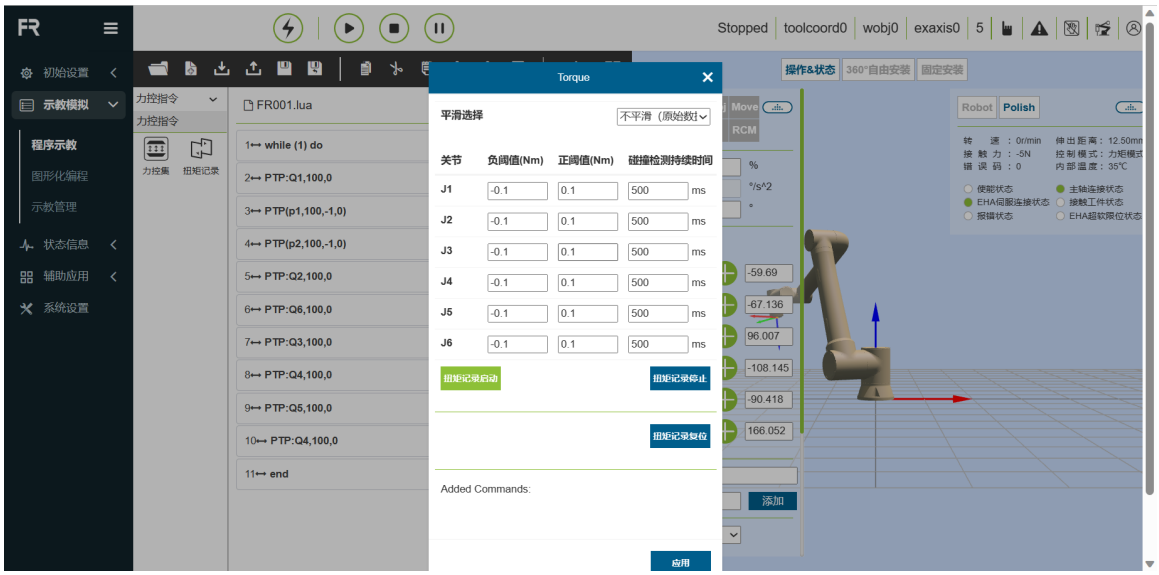


图表 3.7-9-1 F/T 指令界面

1.3.5.7.9.2 扭矩记录命令

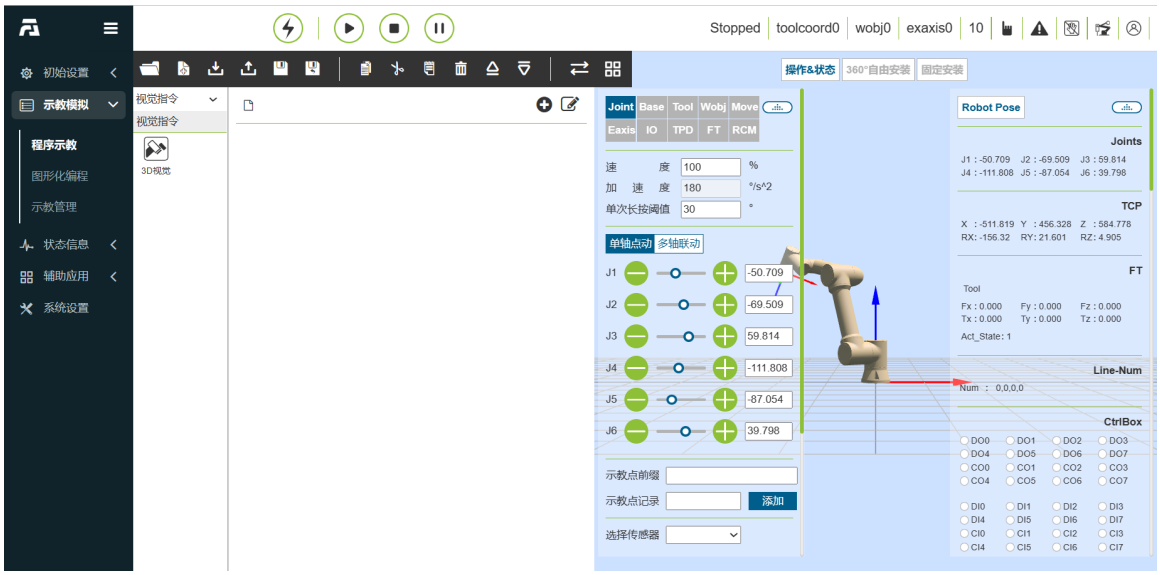
点击“扭矩记录”图标进入 Torque 命令编辑界面

该指令为扭矩记录指令，实现扭矩实时记录碰撞检测功能。点击“扭矩记录启动”按钮，持续记录运动指令运行过程中的碰撞情况，记录的实时扭矩作为碰撞检测判断的理论值，以减少误报错概率。当超出设定阈值范围时，记录碰撞检测持续时间。点击“扭矩记录停止”按钮，停止记录。点击“扭矩记录复位”，状态恢复默认状态。



图表 3.7-9-2 Torque 指令界面

1.3.5.7.10 视觉指令界面

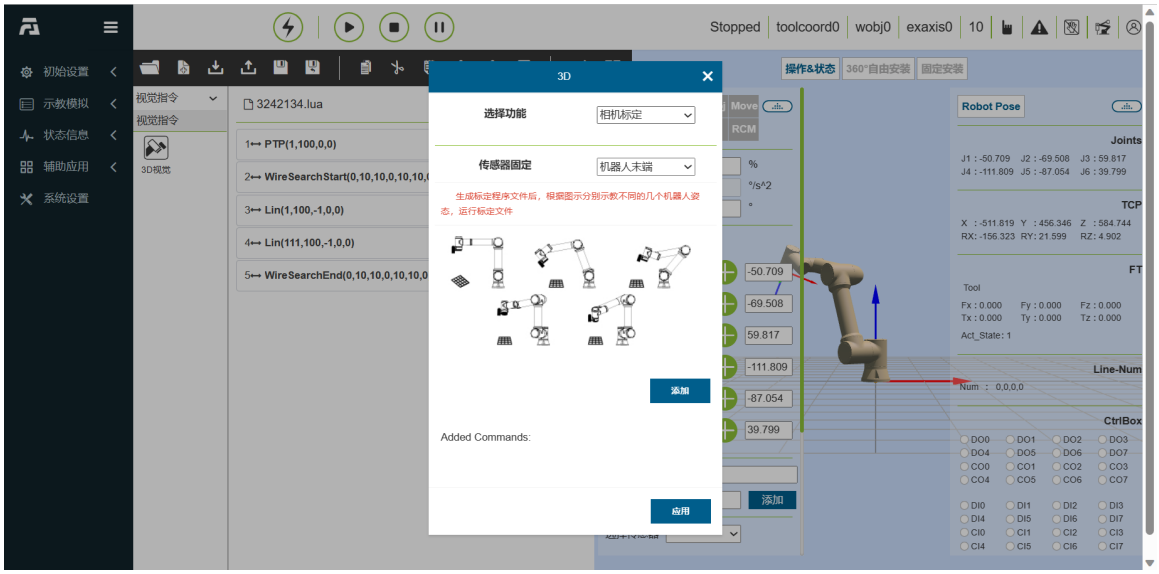


图表 3.7-10 视觉指令界面

1.3.5.7.10.1 3D 视觉命令

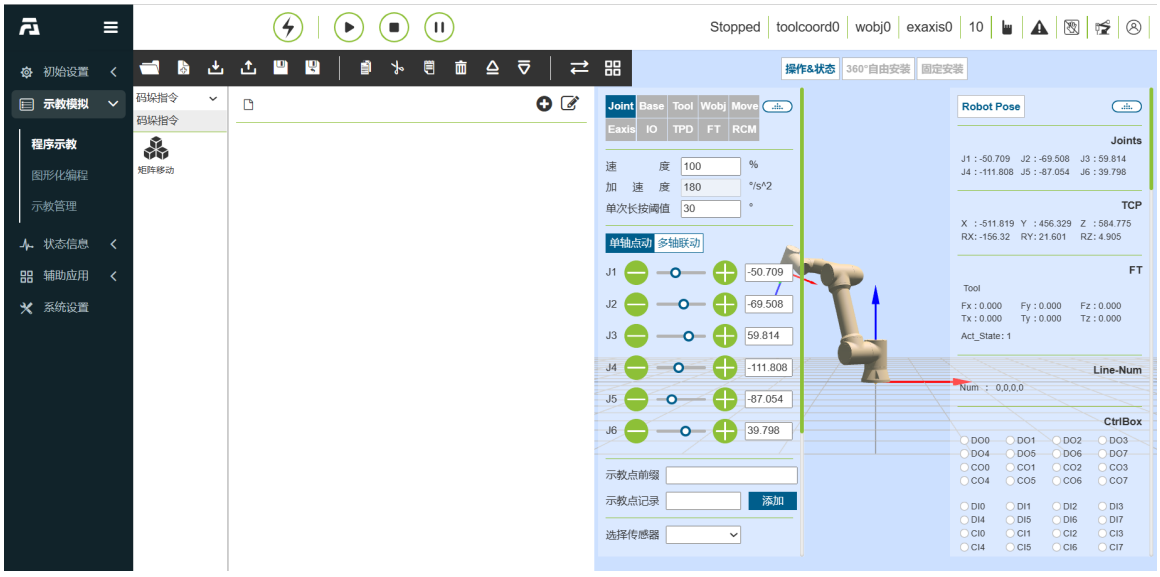
点击“3D 视觉”图标进入 3D 命令编辑界面

该指令为 3D 视觉程序实例生成指令，用户可以根据生成的程序进行参考，与其他视觉设备进行通讯工作，包含相机标定和相机抓取两个程序案例参考。



图表 3.7-10-1 3D 指令界面

1.3.5.7.11 码垛指令界面

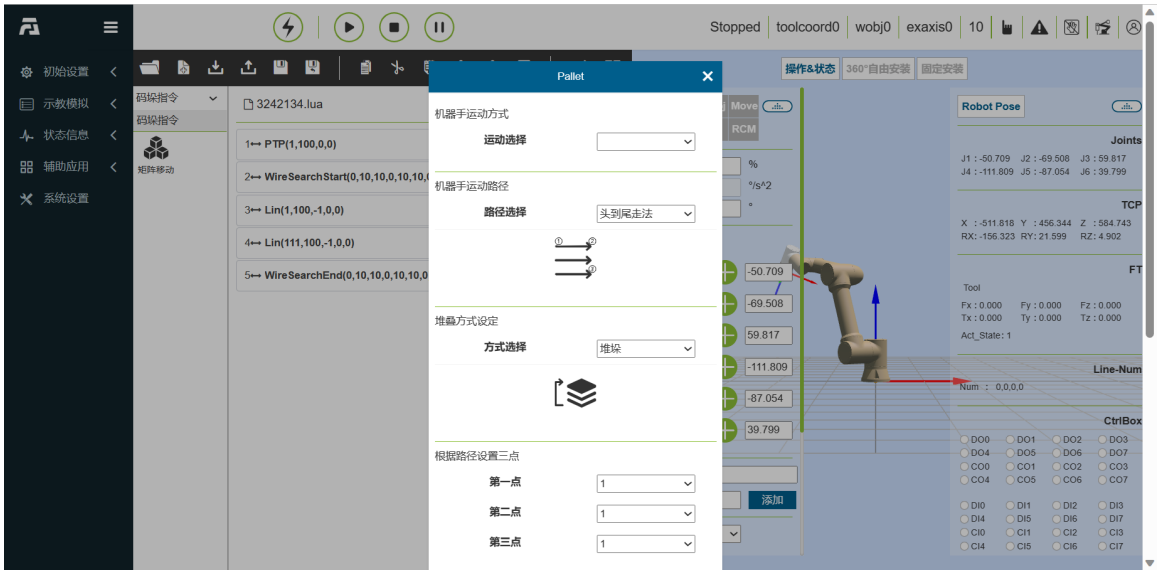


图表 3.7-11 码垛指令界面

1.3.5.7.11.1 矩阵移动命令

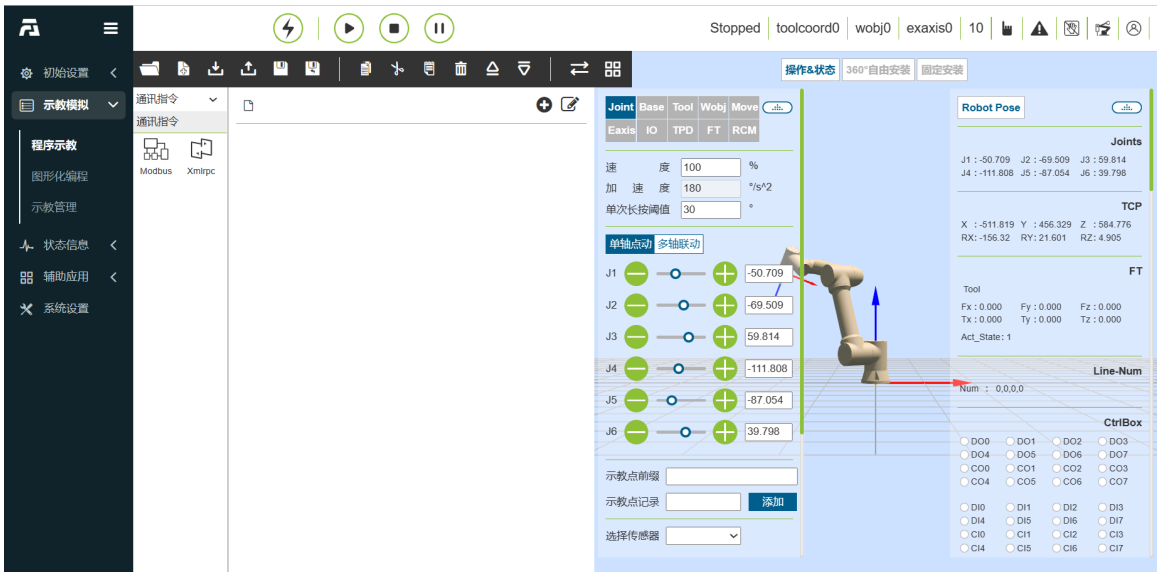
点击“矩阵移动”图标进入 Pallet 命令编辑界面

该指令为码垛程序生成指令，与 3.9.6 节矩阵移动功能是一致的，详细说明参考那一章节。



图表 3.7-11-1 Pallet 指令界面

1.3.5.7.12 通讯指令界面



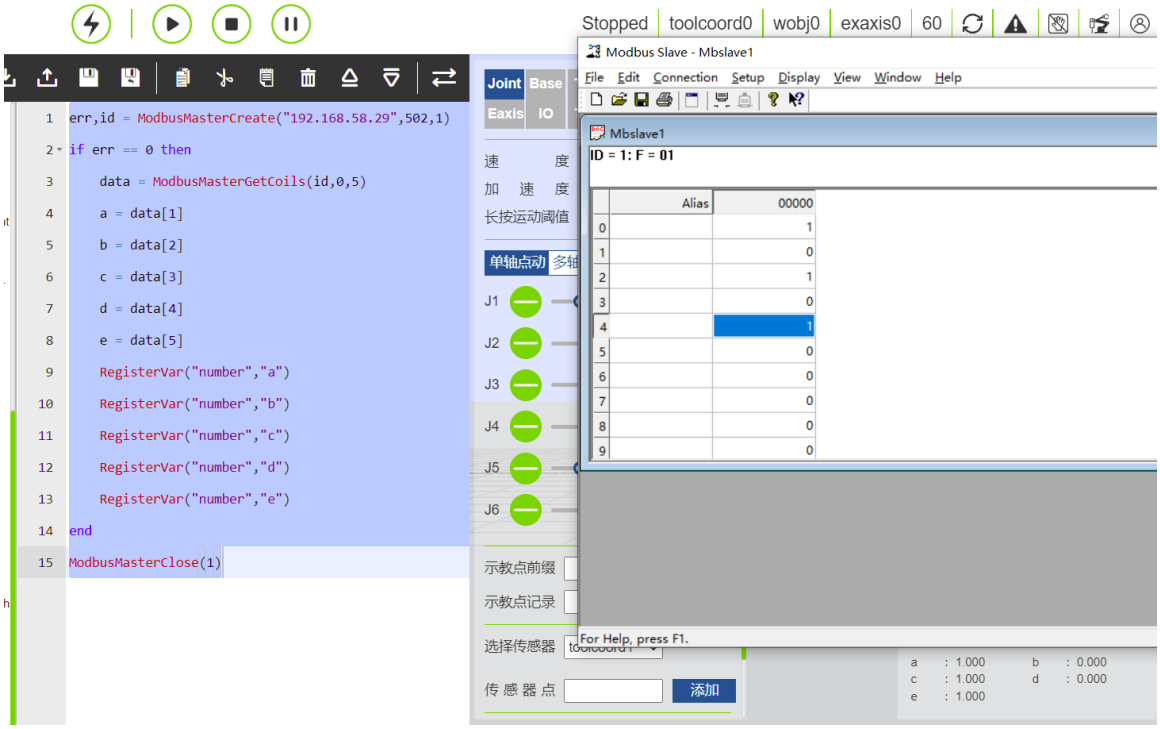
图表 3.7-12 通讯指令界面

1.3.5.7.12.1 Modbus 命令

点击“Mobus”图标进入 Modbus 命令编辑界面

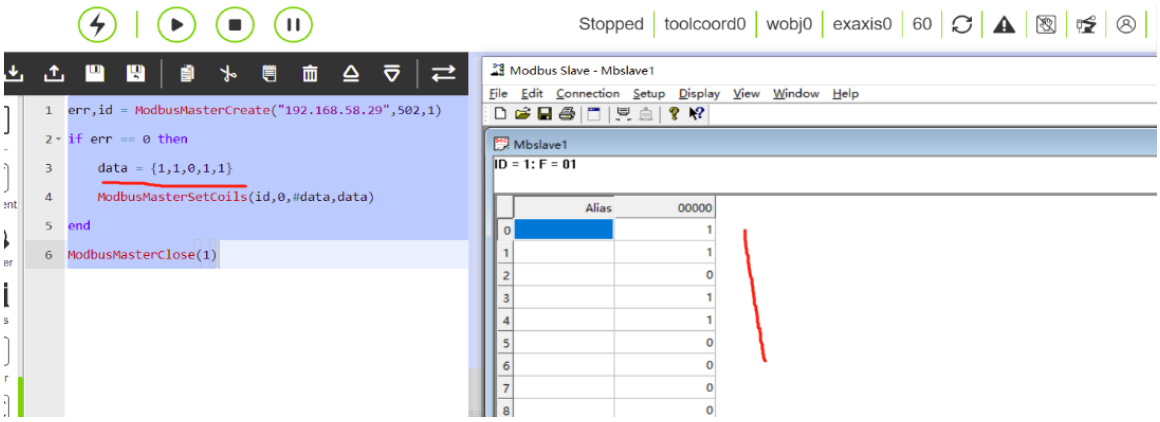
该指令功能为基于 ModbusTCP 协议的总线功能，用户可以通过相关指令控制机器人与 ModbusTCP client 或 server 通讯（主站与从站通讯），对线圈，离散量，寄存器进行读写操作。

modbus 主站读线圈实例：



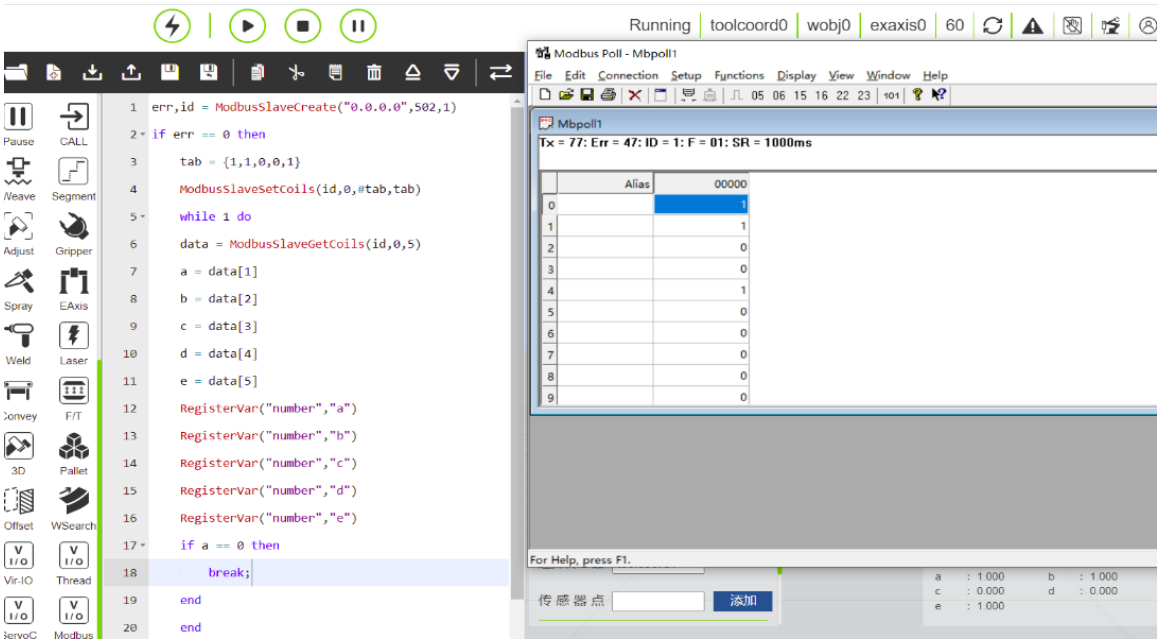

```
err,id = ModbusMasterCreate("192.168.58.29",502,1)
if err == 0 then
    data = ModbusMasterGetCoils(id,0,5)
    a = data[1]
    b = data[2]
    c = data[3]
    d = data[4]
    e = data[5]
    RegisterVar("number","a")
    RegisterVar("number","b")
    RegisterVar("number","c")
    RegisterVar("number","d")
    RegisterVar("number","e")
end
ModbusMasterClose(1)
```

modbus 主站写线圈实例：




```
err,id = ModbusMasterCreate("192.168.58.29",502,1)
if err == 0 then
    data = {1,1,0,1,1}
    ModbusMasterSetCoils(id,0,#data,data)
end
ModbusMasterClose(1)
```

Modbus 从站读写线圈实例：

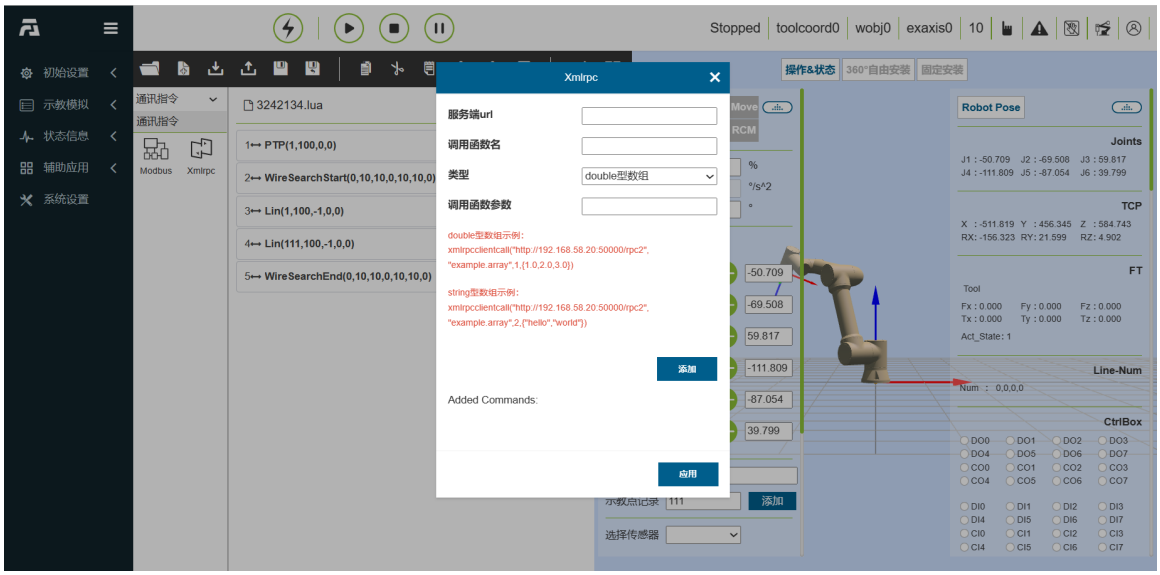


关于 ModbusTCP 更多操作功能，前请联系我们咨询。

1.3.5.7.12.2 Xmlrpc 命令

点击“Xmlroc”图标进入 Xmlrpc 命令编辑界面

XML-RPC 是一种通过 sockets 使用 xml 在程序之间传输数据的远程过程调用方法。通过这种方法，机器人控制器可以在远端的程序/服务调用功能函数（可带参数）并获取返回的结构性数据。机器人控制器负责处理编写 XML-RPC 客户端消息以及处理数据类型与 XML 之间转换的所有细节。



图表 3.7-12-1 Xmlrpc 指令界面

重要:

1) 控制器作为客户端连接远端自定义端口;

2) 控制器作为客户端调用远端功能函数;

3) 支持调用远端不同功能函数;

4) 支持字符串数组参数传入与字符数组结果返回，数组元素个数可自定义;

支持 double 型数组参数传入与 double 型数组结果返回，数组元素个数个数可自定义;

```
XmlrpcClientCall(serverUrl,methodName,tableType,param)

serverUrl 服务端url，例如："http://192.168.58.29:50000/RPC2"

methodName 调用函数名，“example.add”

tableType 1-double型数组，2-string型数组

param 调用函数参数
```

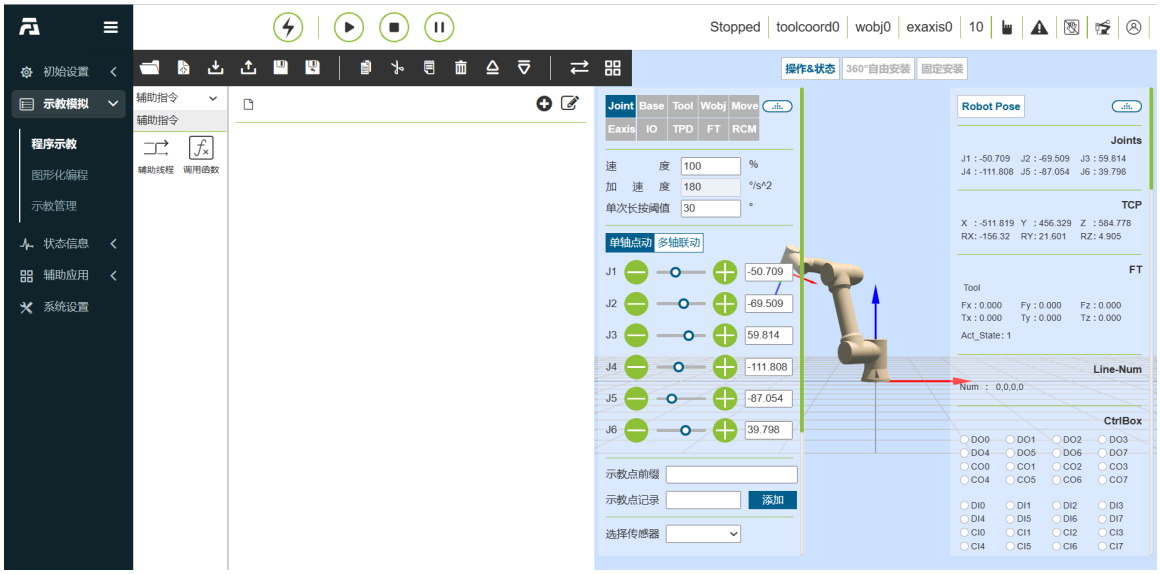
```
XmlrpcClientCall(error, result)

error 0-无错误，1-错误

result 若参数传入为double型数组，则result为double型数组，

若参数传入为string型数组，则result为string型数组
```

1.3.5.7.13 辅助指令界面

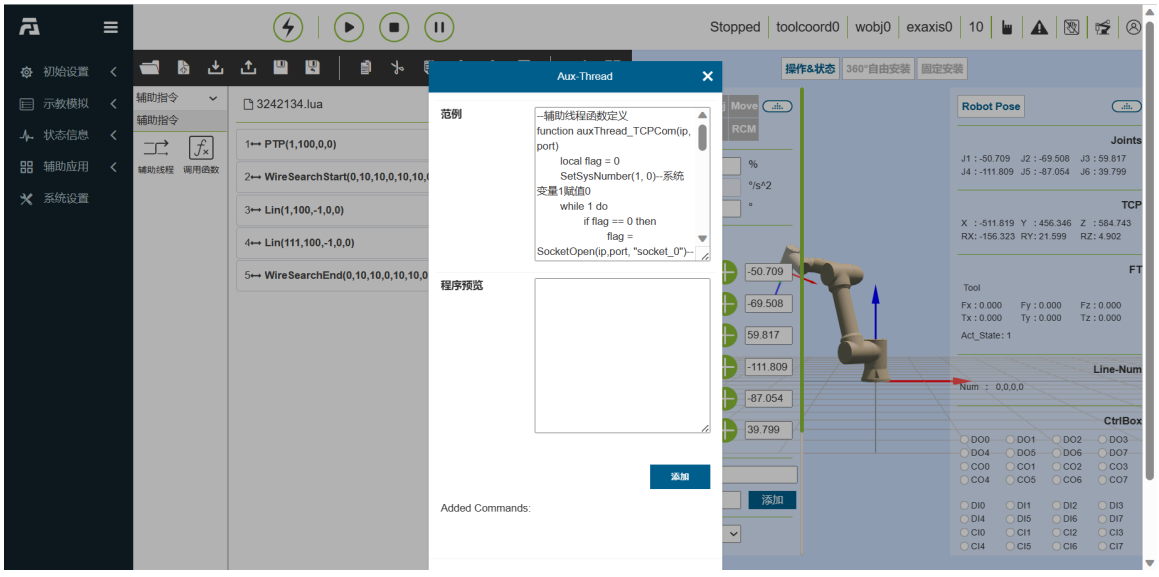


图表 3.7-13 辅助指令界面

1.3.5.7.13.1 辅助线程命令

点击“辅助线程”图标进入 Thread 命令编辑界面

Thread 命令为辅助线程功能，用户可以定义一个辅助线程与主线程同时运行，辅助线程主要与外部设备进行数据交互，支持 socket 通信，机器人 DI 状态获取，机器人 DO 状态设置，机器人状态信息获取，与主线程数据交互，主线程通过辅助线程获取的数据用于控制机器人运动逻辑的判断，用户程序示例截图：

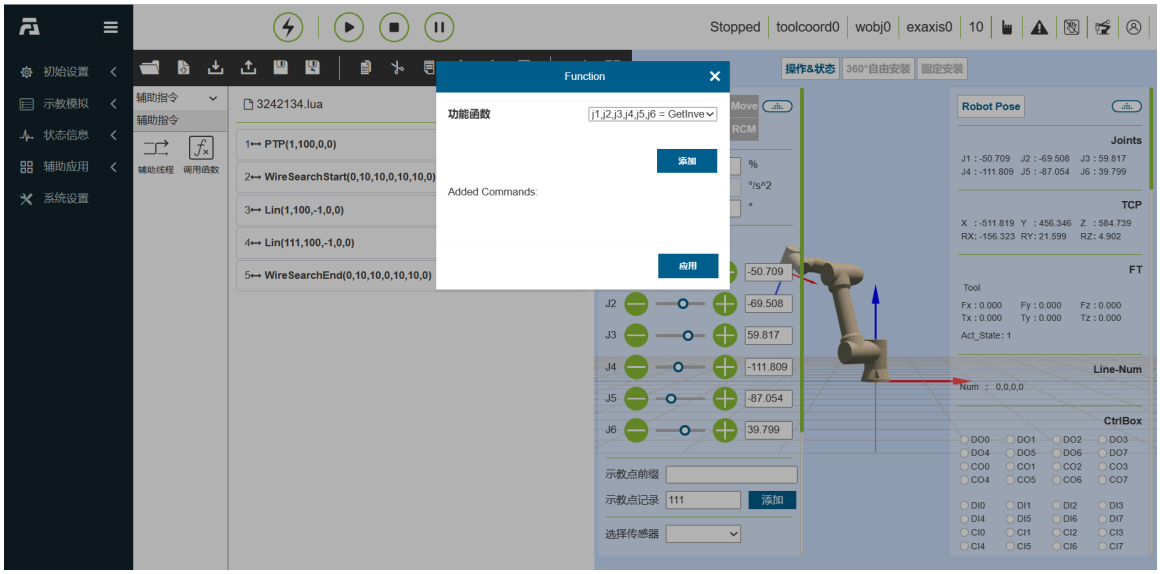


图表 3.7-13-1 Thread 程序示例

1.3.5.7.13.2 调用函数命令

点击“调用函数”图标进入 Function 命令编辑界面

该指令为调用函数接口功能，将机器人接口函数提供给客户选择，并提示客户该函数所需要的参数，方便客户编写脚本指令，更多函数陆续添加中。

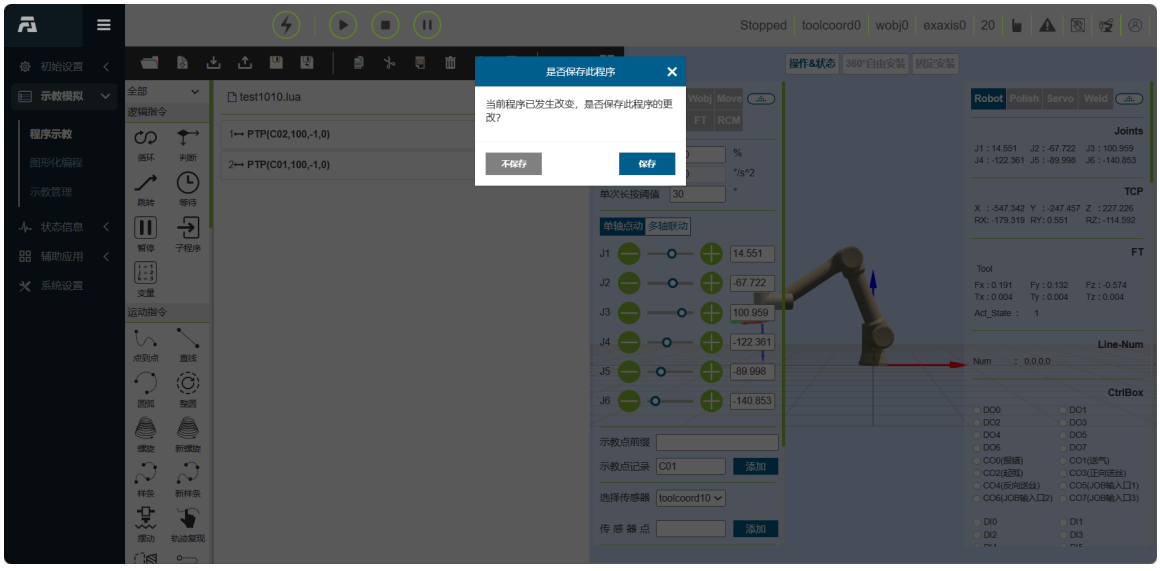


图表 3.7-13-2 Function 指令界面

1.3.5.7.14 示教程序未保存验证

在程序示教页面，打开/新建程序后，若示教程序发生改动未保存程序。

若点击“打开”、“新建”、“导出”、“重命名”等相关文件操作，则触发“是否保存此程序”弹出框，提示“当前程序已发生改变，是否保存此程序的更改?”，如下图。

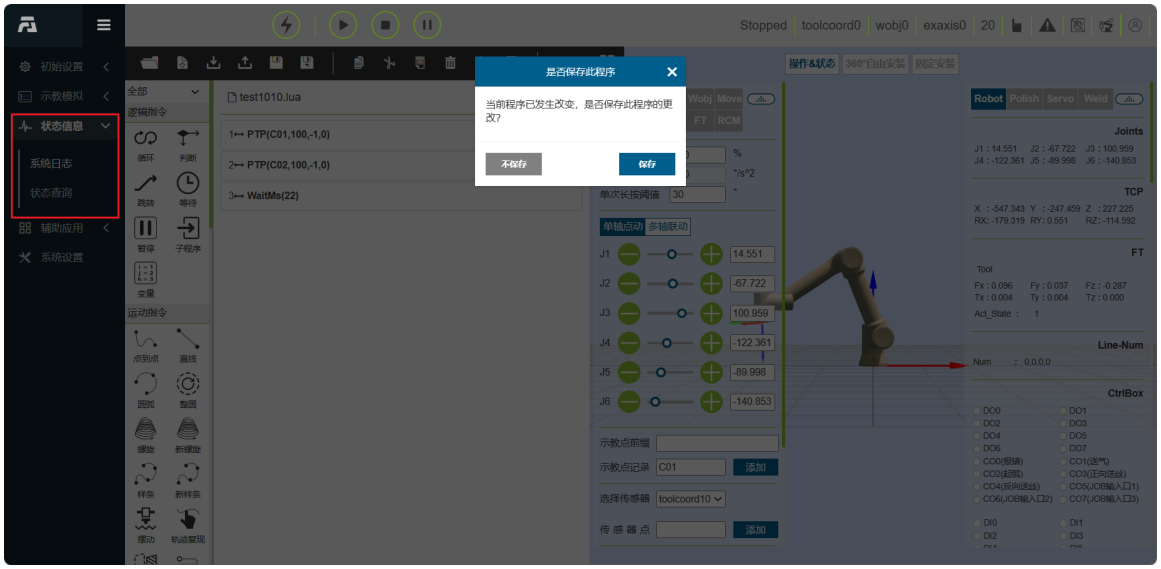


图表 3.7-14 当前页面程序未保存验证

Step1：点击“不保存”按钮，程序恢复未修改之前数据, 并继续执行之前的相关文件操作。

Step2：点击“保存”按钮，未保存的 lua 程序保存成功，并继续执行之前的相关文件操作。

若离开程序示教页面，切换到其他页面时，同样触发“是否保存此程序”提示，且仍然停留在当前示教程序页面，如下图。



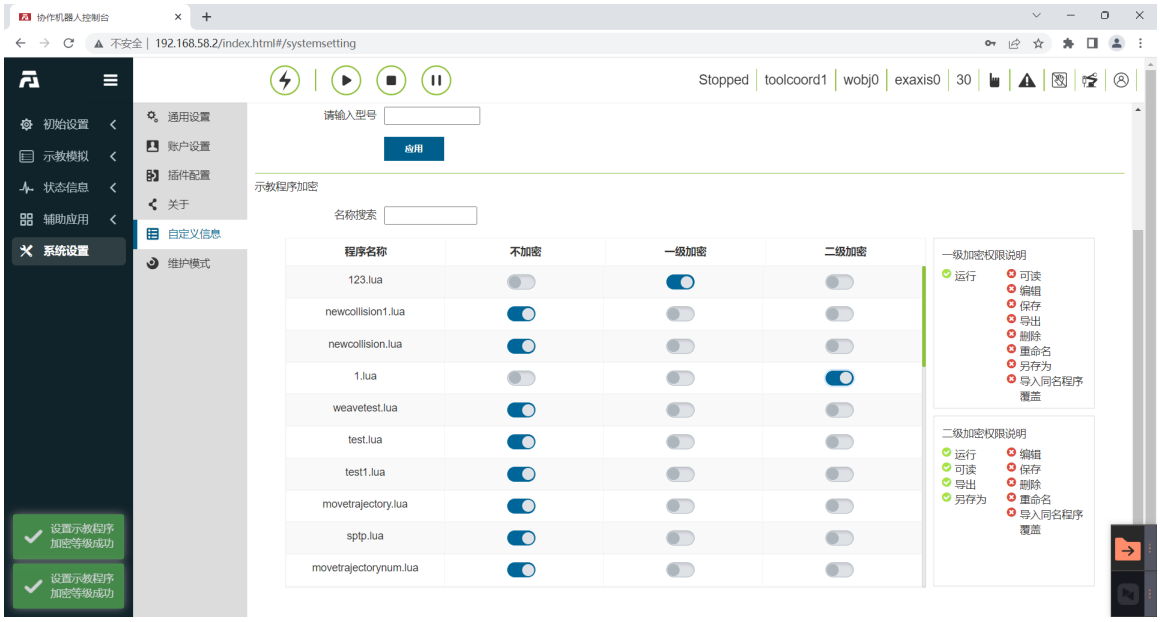
图表 3.7-15 切换页面程序未保存验证

Step1：点击“不保存”按钮，跳转到之前选择的页面。

Step2：点击“保存”按钮，未保存的 lua 程序保存成功，并跳转到之前选择的页面。

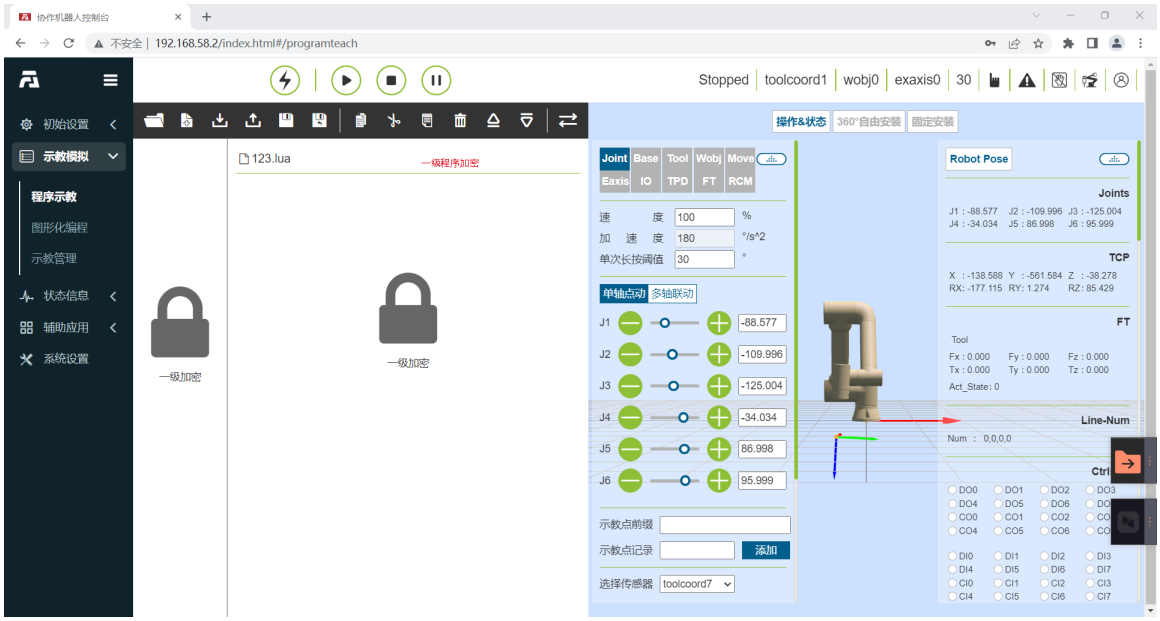
1.3.5.7.15 示教程序加密

示教程序分为加密和不加密的状态。加密级别分为一级加密和二级加密，其中一级加密保护程度最高，二级次之。所有示教程序在“系统设置–自定义信息”中以表格形式进行程序加密信息展示与设置。表格右侧配有加密级别说明。



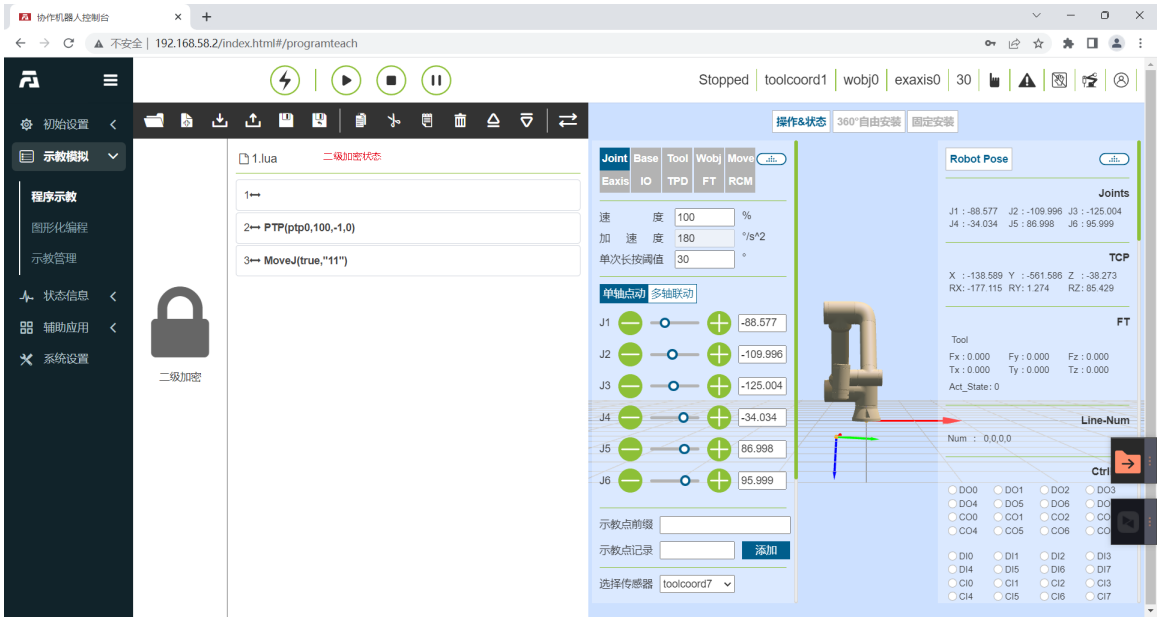
图表 3.7-16 示教程序加密

当程序为一级加密状态时，打开该程序后：操作栏中对应的“导出”、“保存”、“另存为”、“复制”、“剪切”、“粘贴”、“删除”、“上移”、“下移”和“编辑模式切换”等按钮图标都会变灰，点击图标无效并会提示当前程序处于加密状态。程序“重命名”图标将会隐藏。添加指令栏和程序编辑区域都会不可见且提示已处于一级加密锁定。



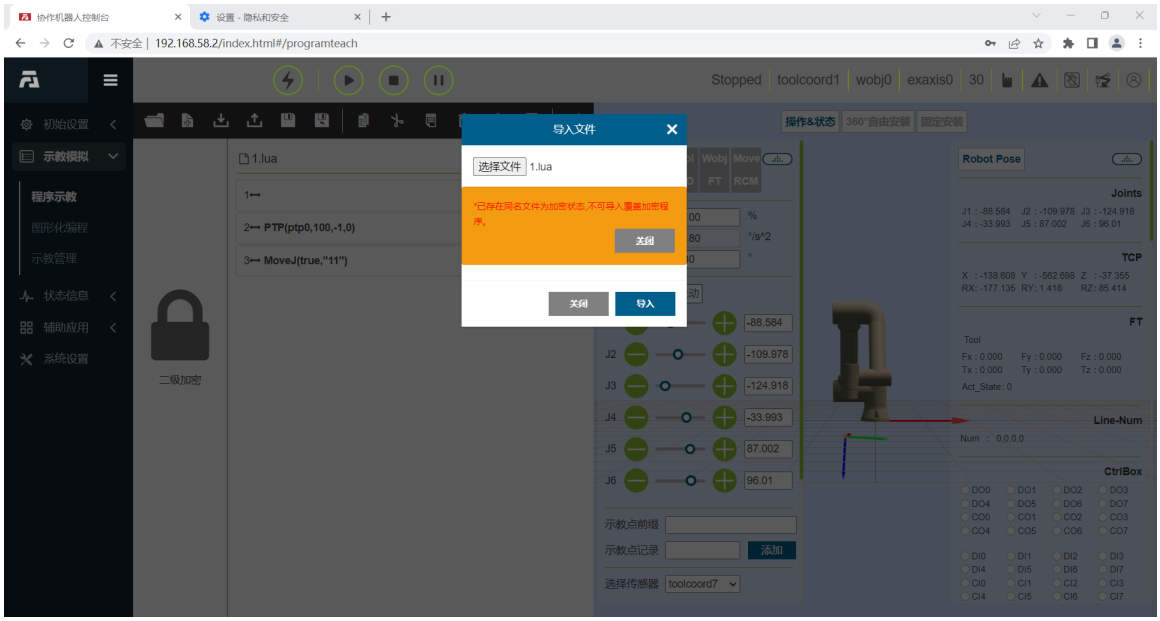
图表 3.7-17 程序一级加密界面

当程序为二级加密时，在“程序示教”页面打开该程序后：操作栏中对应的“保存”、“复制”、“剪切”、“粘贴”、“删除”、“上移”和“下移”等按钮图标都会变灰。点击图标无效并会提示当前程序处于加密状态。程序“重命名”图标将会隐藏。添加指令栏不可见且提示已处于二级加密锁定。程序编辑区域可正常浏览阅读程序。



图表 3.7-18 程序二级加密界面

一级加密和二级加密都可以使用“导出”功能，在导入时会进行验证操作，如果存在同名程序为加密文件，则会中断导入操作并提示不可导入覆盖加密程序。

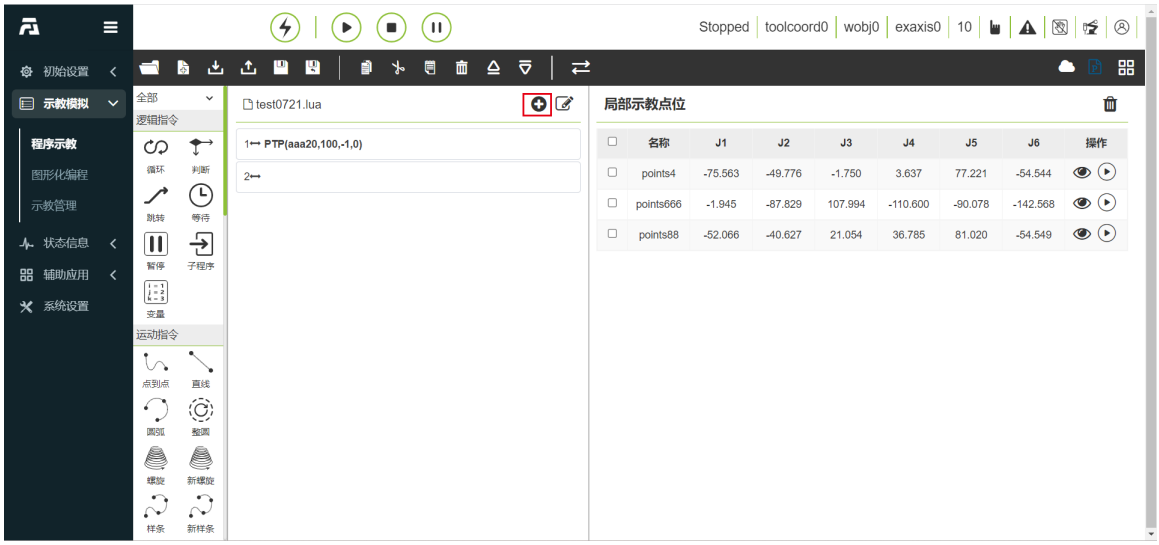


图表 3.7-19 程序导入

1.3.5.7.16 局部示教点位

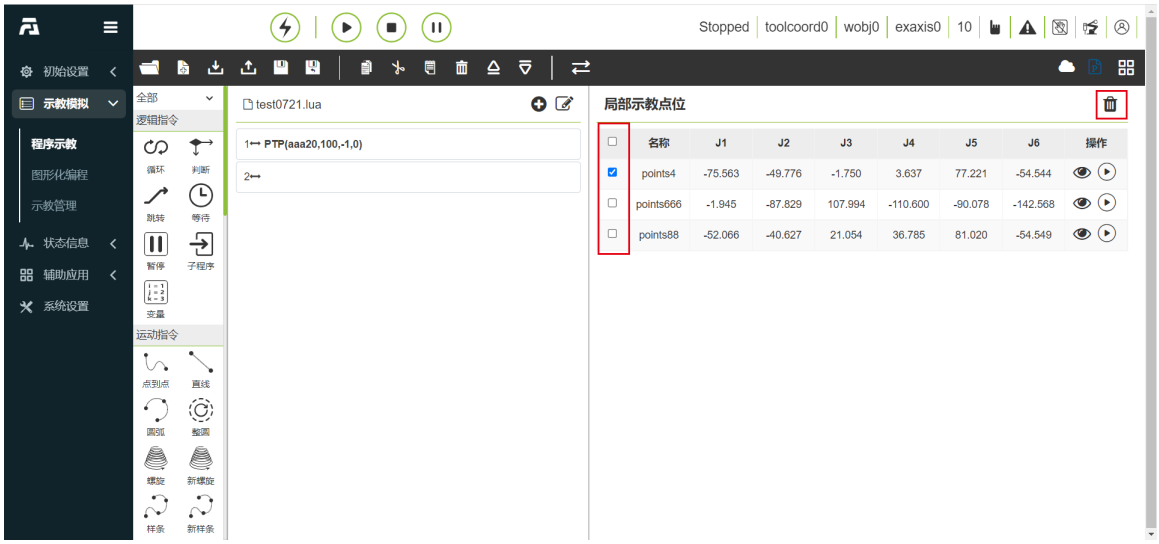
局部示教点位和当前示教程序绑定。添加程序命令时，只能应用于当前示教程序，不可用于其它示教程序。

新增：点击程序文件名最右侧的“新增局部示教点”图标，进行局部示教点的添加。(局部示教点位详情记录请翻阅机器人操作中的示教点记录)



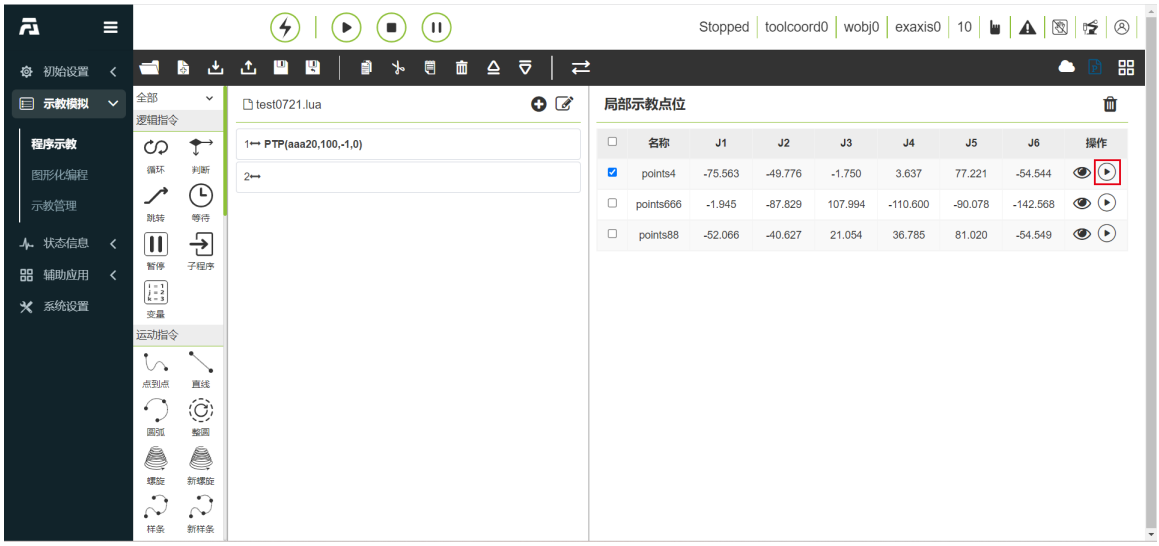
图表 3.7-20 新增局部示教点

删除：点击表格序号栏选择需要删除的局部示教点后，点击局部示教点位标题右上角的“删除”图标，进行局部示教点的删除。



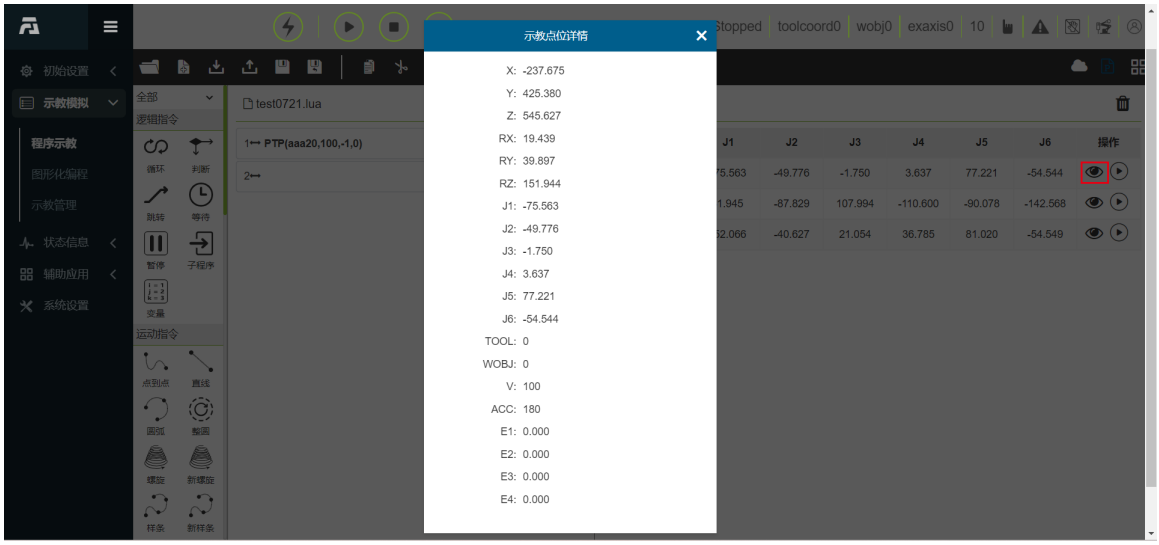
图表 3.7-21 删除局部示教点

运行：点击局部示教点位表格数据操作栏中的“开始运行”图标，进行局部示教点的单点运行，将机器人移动到该点的位置。



图表 3.7-22 运行局部示教点

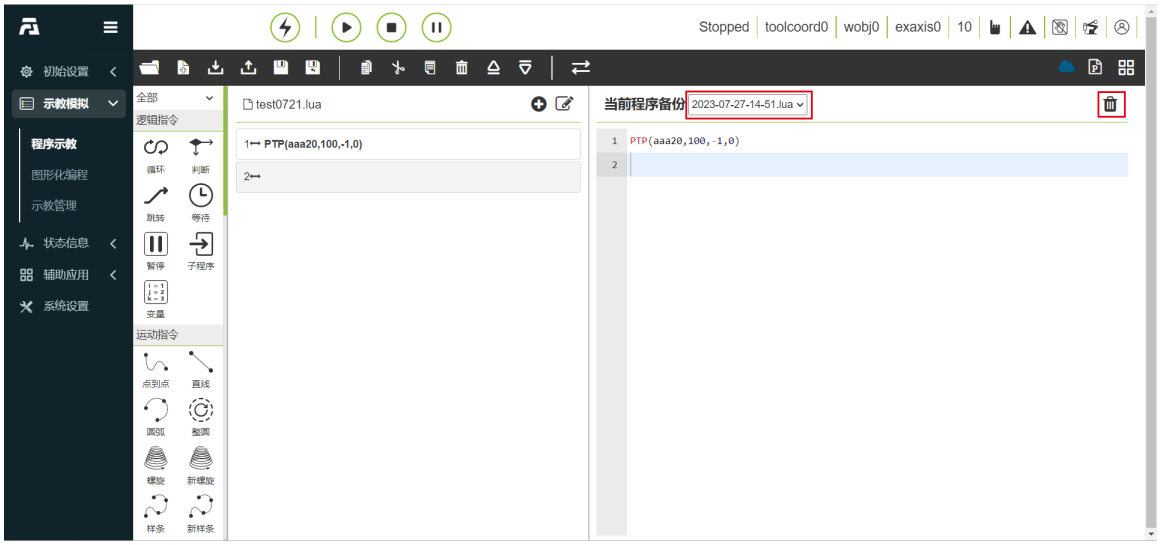
详情：点击局部示教点位表格数据操作栏中的“详情”图标，查看局部示教点的详情。



图表 3.7-23 局部示教点详情

1.3.5.7.17 当前程序备份

用户修改示教程序点击保存后，触发当前程序的“备份”功能(备份时间为 1 年)，将当前程序的初始内容进行保存展示在右侧，方便用户对比修改的内容。用户选择日期可以查看对应的程序备份内容，点击右上角“删除”图标可以删除当前程序备份内容。当前程序备份的内容只可查看，不可修改。

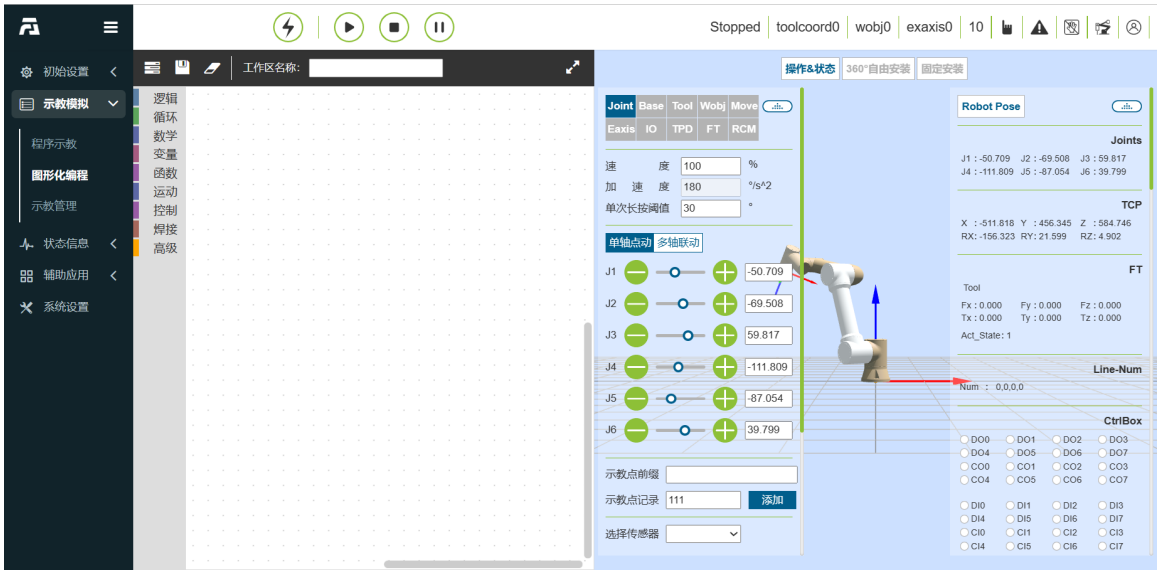


图表 3.7-24 当前程序备份

1.3.5.7.18 图形化编程

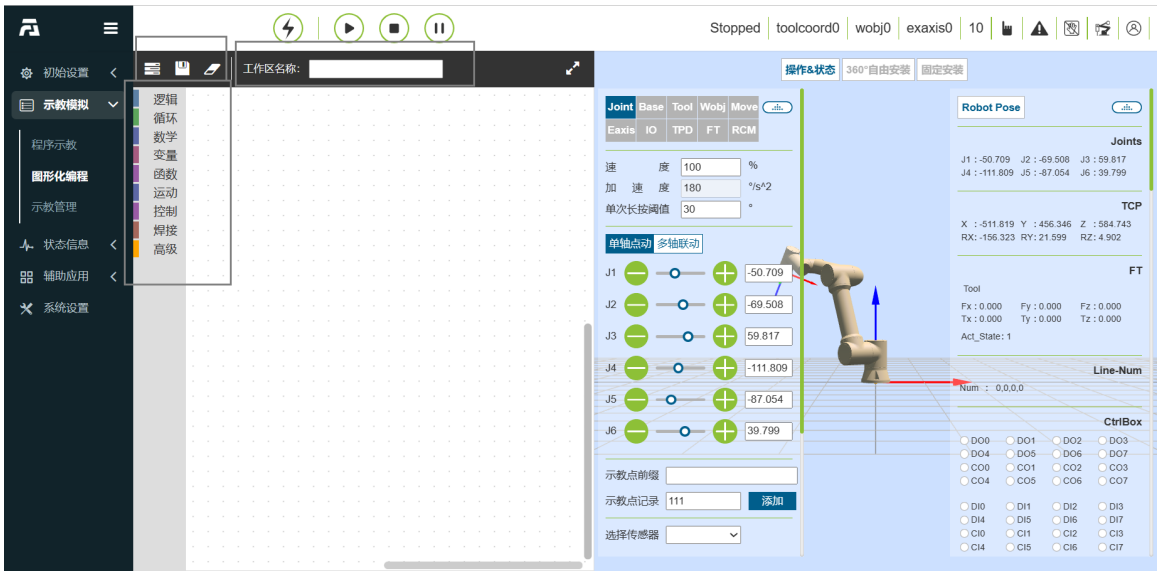
由于示教器一般不会外接键鼠等外设，在示教器端访问机器人 WebAPP 时，用户可以通过图形化编程功能进行机器人示教程序编辑。功能标准化函数实现使用 Blockly 库，可以集成在 WebAPP 系统中，根据需要实现自定义代码块，并且拖拽编程完成后转换为 LUA 程序通过现有指令协议下发运行。

通过使用图形编程，能够做到简单易懂，易操作，语言汉化操作。



图表 3.7-25 图形化编程界面

页面分为三个区域：“操作栏”、“toolbox 工具栏”和“workspace 代码编辑区”，整体的布局设计如图如下



图表 3.7-26 图形化编程页面布局整体设计

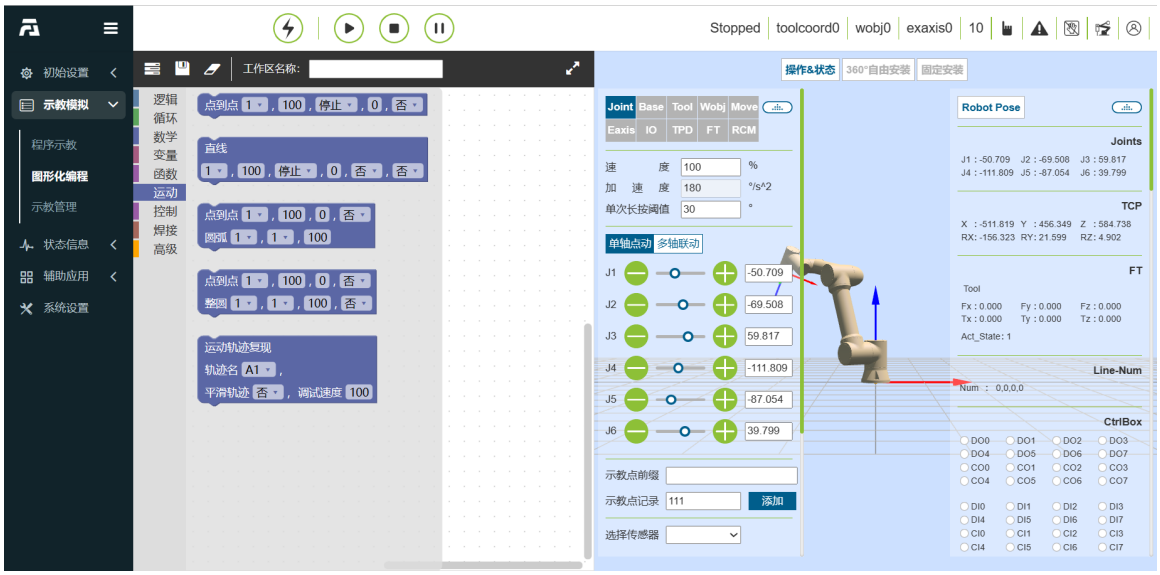
操作栏：“加载”按钮负责 workspace 的重新加载，“保存”按钮功能为代码块编辑完成后保存为对应的示教程序，“清空按钮”负责快速清空代码编辑区；

Toolbox：包含所有指令和逻辑代码的代码块，可以拖动到 workspace 创建代码块并编辑；Toolbox 工具栏部分会根据指令类型进一步分类。逻辑类指令：if-else，while，print 等；基础运动类指令：PTP，LIN，ARC 等；依据应用场景指令分类：涂胶，焊接，传送带等。在使用的过程中可以方便地找到所需代码块。

Workspace：在代码编辑区中可以编辑和展示图形化的代码块。

1.3.5.7.18.1 运动类图形化编程命令

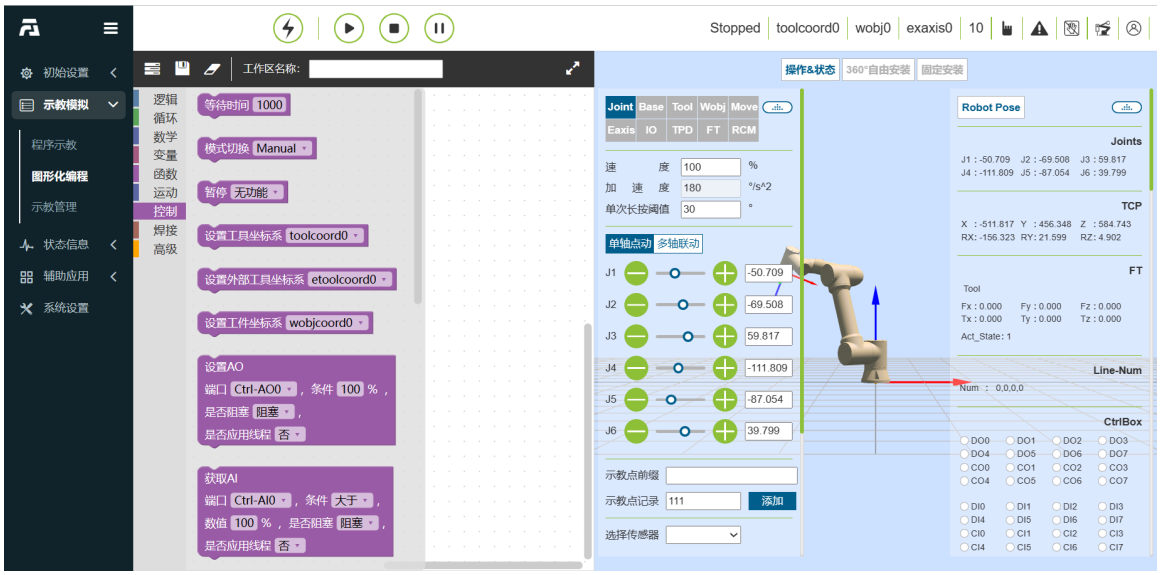
运动类图形化编程命令包含 PTP、Lin、ARC等运动命令。



图表 3.7-27 运动类图形化编程

1.3.5.7.18.2 控制类图形化编程命令

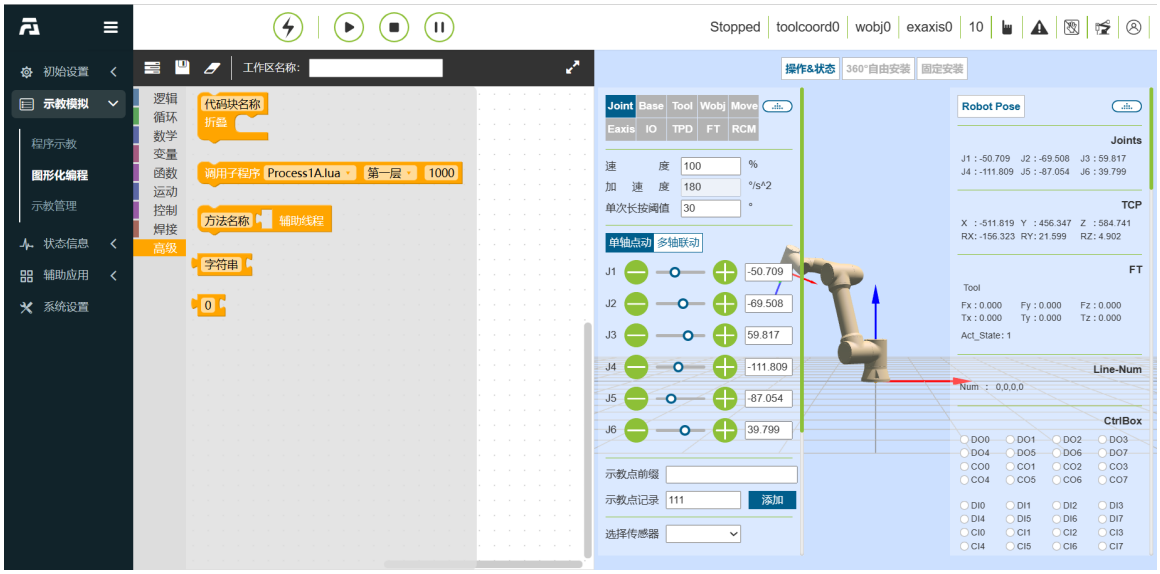
控制类图形化编程命令包含Wait、IO等控制命令。



图表 3.7-28 控制类图形化编程命令

1.3.5.7.18.3 高级类图形化编程命令

高级类图形化编程命令包含dofile 调用子程序、thread 多线程、折叠指令等高级命令。

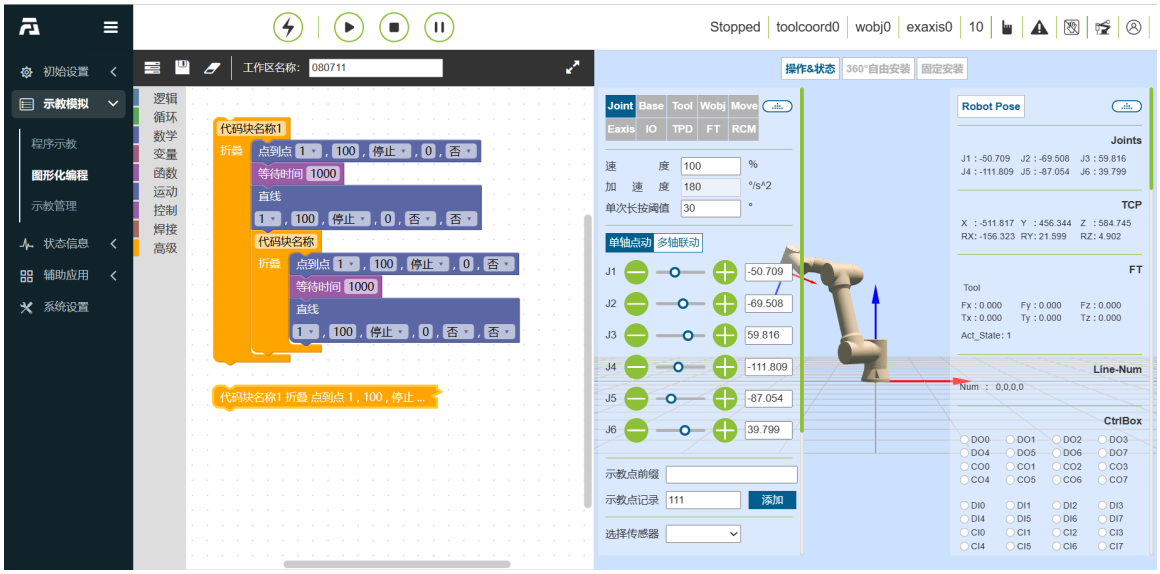


图表 3.7-29 高级类图形化编程命令

1.3.5.7.18.4 图形化编程命令使用示例

选择图形化编程类型后，点击需要使用的图形代码块，即可在工作区进行拖拽和拼接操作。

例如选择 PTP 和 Lin 运动指令以及控制指令 Waitms 进行拼接，外层可嵌套一个折叠高级指令并输入注释名称，则可实现代码块折叠操作。其中点击下拉框可选择指令参数类型，输入框可填入指令参数数据。图形化编程命令示例如下：

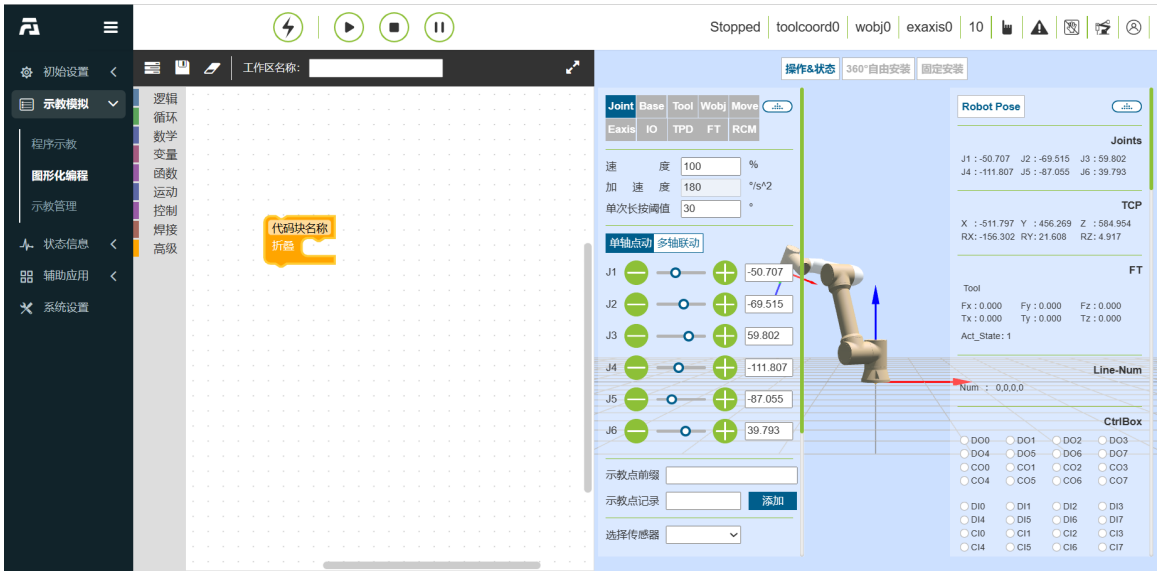


图表 3.7-30 图形化编程命令示例

图形化编程指令拼接和参数填入完成后，填写工作区名称，点击“保存”图标即可保存本次程序。选择编写完成的“工作区”，点击开始运行，即可执行本段程序。

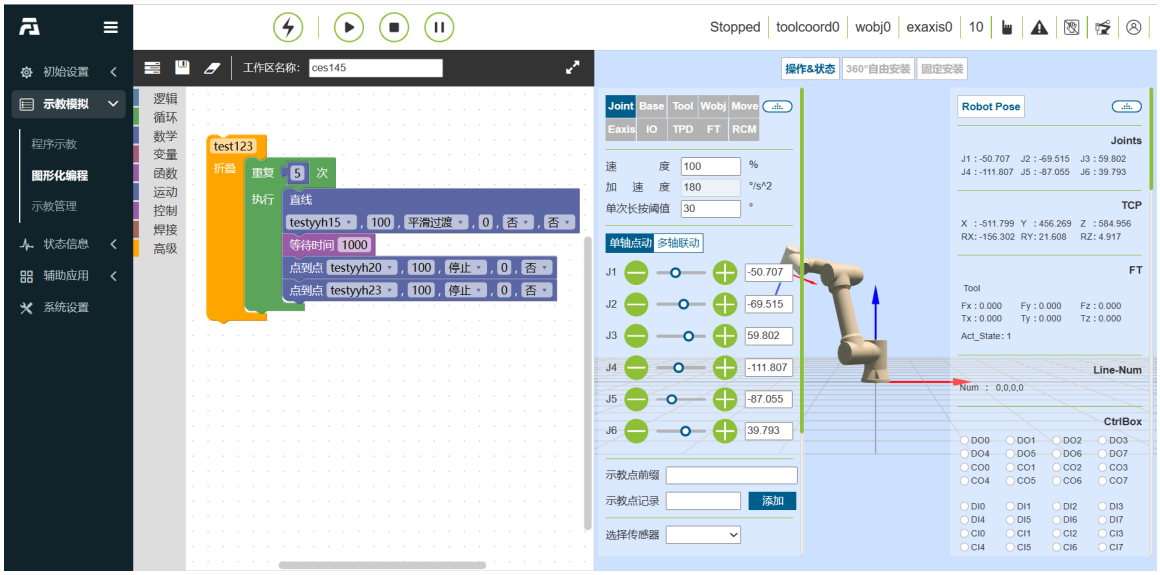
1.3.5.7.18.5 图形化编程代码块模块化

为了提高图形化编程代码可阅读性，增加了图形化编程代码块模块化功能，即高级指令：折叠指令代码块。



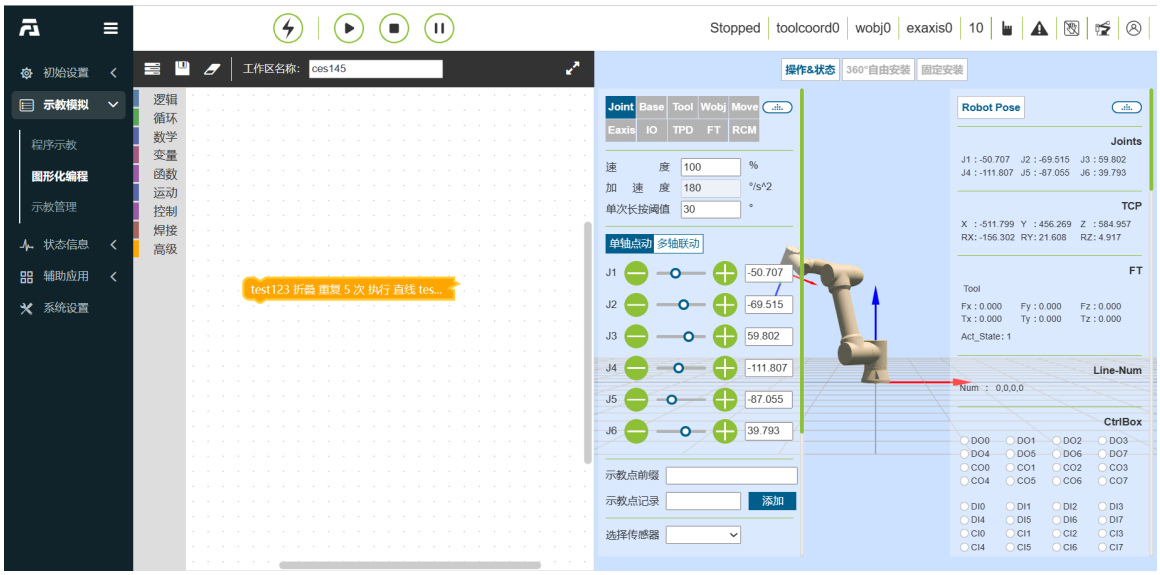
图表 3.7-31 折叠指令代码块

1. 编写一段代码块指令，在外层添加折叠指令代码块，在输入框内编写该段指令的备注。



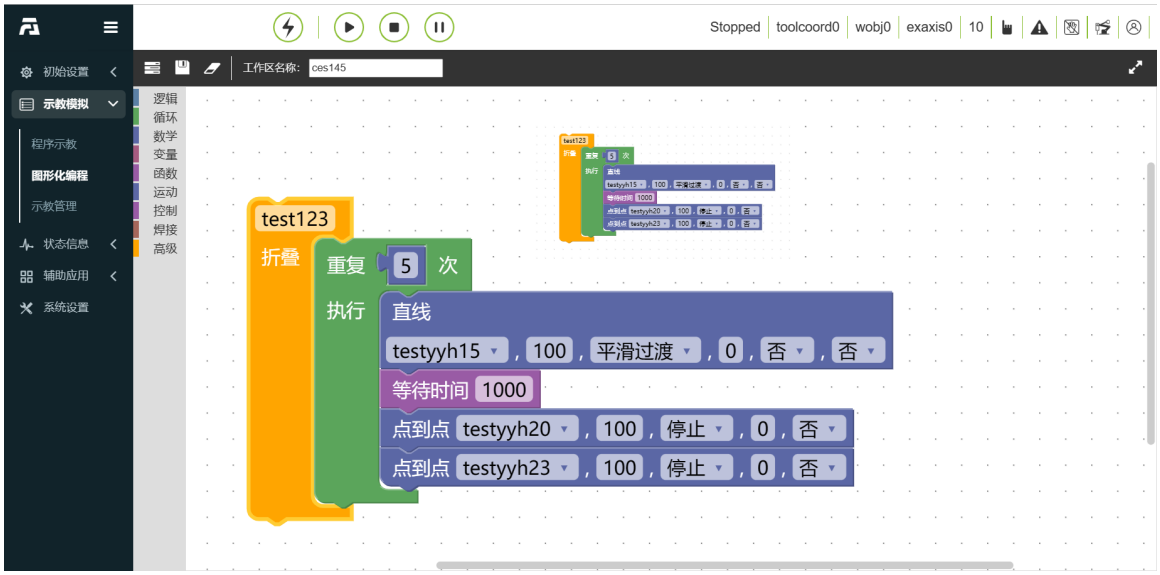
图表 3.7-32 折叠指令效果图

2. 右键操作栏右击”折叠块”，该段指令代码块折叠，该代码块折叠成一行显示，且折叠下可正确执行程序



图表 3.7-33 折叠后效果图

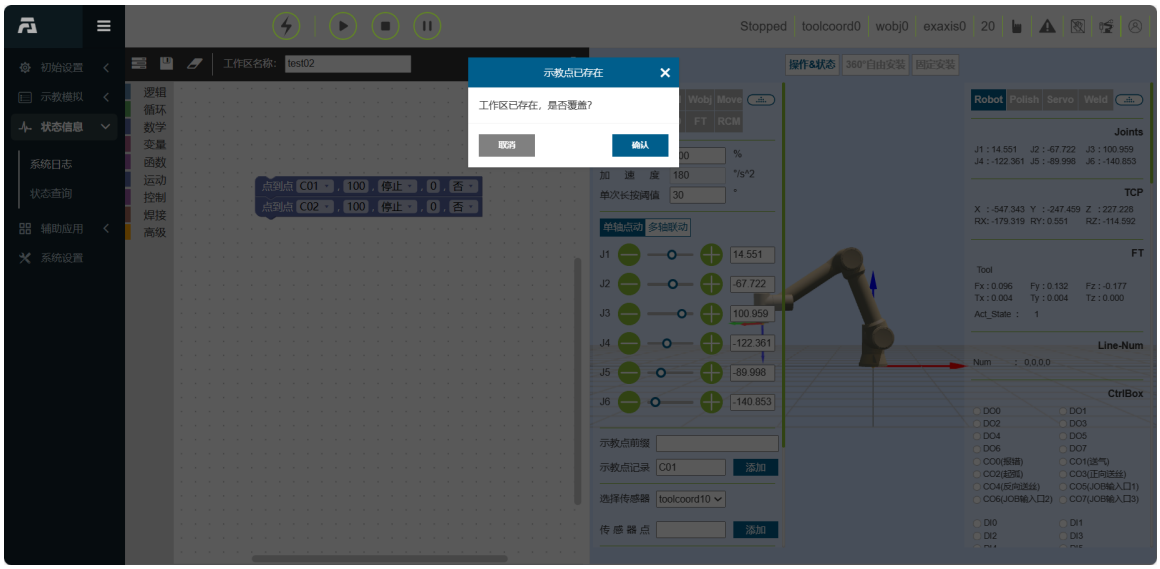
3. 滚动鼠标，可实现页面缩放功能，具体效果如下：



图表 3.7-34 页面缩放功能效果图

1.3.5.7.18.6 图形化编程同名覆盖

在图形化编程页面，新建/加载文件后，更改工作区名称后点击保存。若更改的工作区名称文件已存在，则触发“示教点已存在”弹出框，如下图。



图表 3.7-35 图形化编程程序覆盖

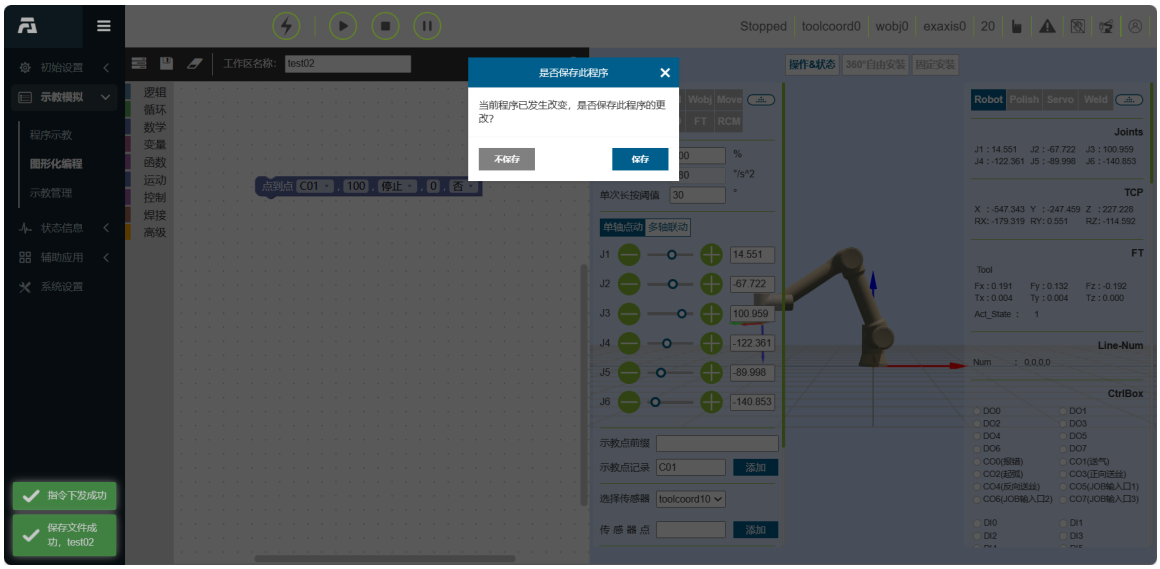
Step1：点击“取消”按钮，继续执行之前的操作。

Step2：点击“同步更新示教程序”复选框，再点击“覆盖”按钮，则当前图形化编程页面的 lua 程序覆盖更改后工作区文件名的 lua 程序。

1.3.5.7.18.7 图形化编程程序未保存验证

在图形化编程页面，打开/新建程序后，若图形化编程程序发生改动未保存程序。

若点击“打开”文件操作，则触发“是否保存此程序”弹出框，提示“当前程序已发生改变，是否保存此程序的更改?”，如下图。

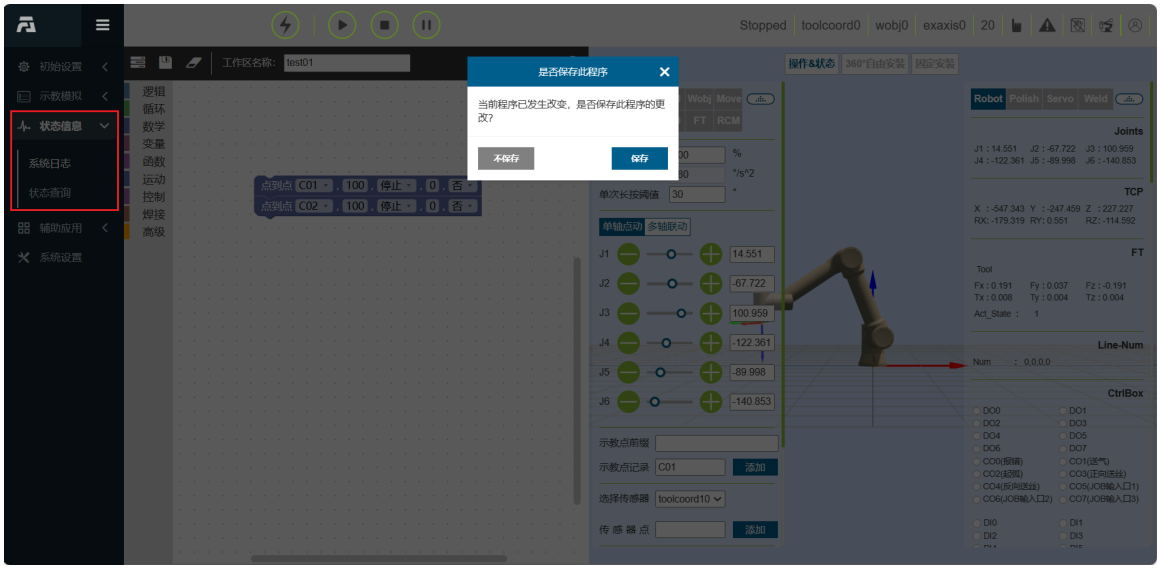


图表 3.7-36 当前页面程序未保存验证

Step1：点击“不保存”按钮，继续执行之前的“打开”文件操作。

Step2：点击“保存”按钮，未保存的 lua 程序保存成功，并继续执行之前的“打开”文件操作。

若离开图形化编程页面，切换到其他页面时，同样触发“是否保存此程序”提示，且仍然停留在当前图形化编程页面，如下图。



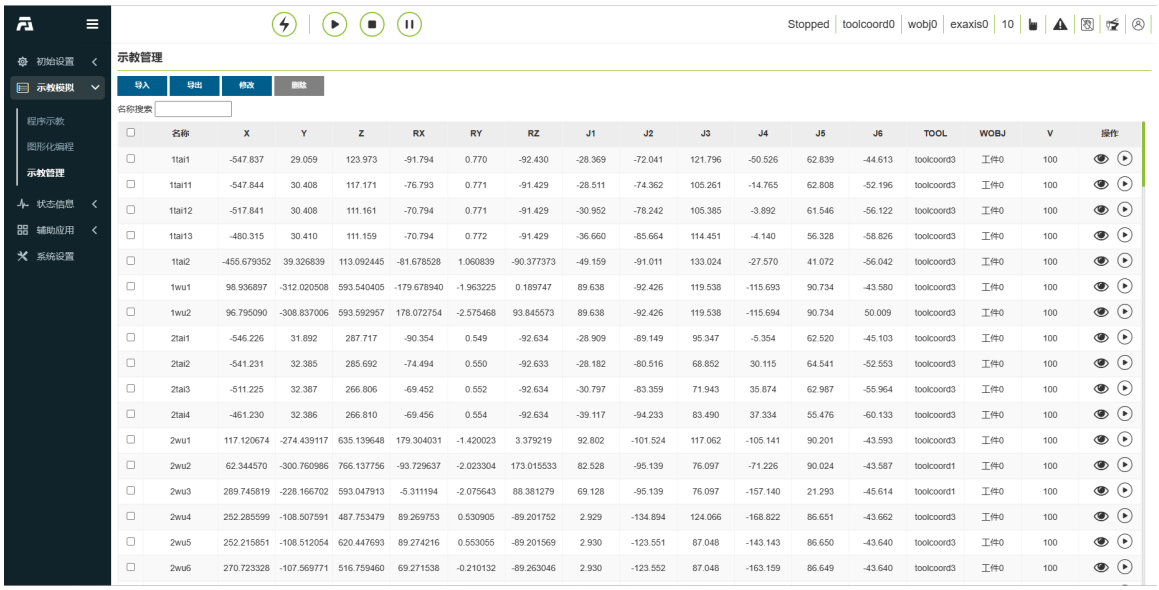
图表 3.7-37 切换页面程序未保存验证

Step1：点击“不保存”按钮，跳转到之前选择的页面。

Step2：点击“保存”按钮，未保存的 lua 程序保存成功，并跳转到之前选择的页面。若保存的程序名称已存在，提示示教点位已存在，是否覆盖。进行取消/覆盖操作后，跳转到之前选择的页面。

1.3.5.7.19 示教管理

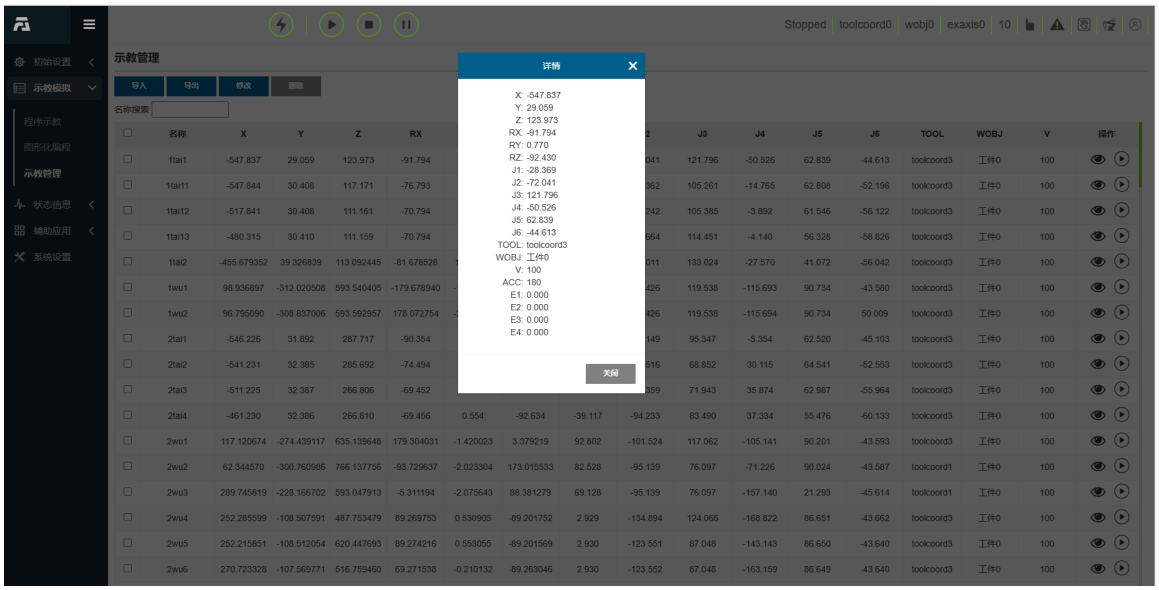
点击“示教管理”可显示所有保存的示教点信息，在该界面中可对示教点文件导入和导出，选中一个示教点后点击“删除”按钮即可将该点信息删除，示教点 x,y,z,rx,ry,rz 和 v 数值可进行修改，输入修改值，勾选左侧勾选兰，点击上方修改即可修改示教点信息。此外，用户可以通过名称搜索示教点。



图表 3.7-38 示教管理界面

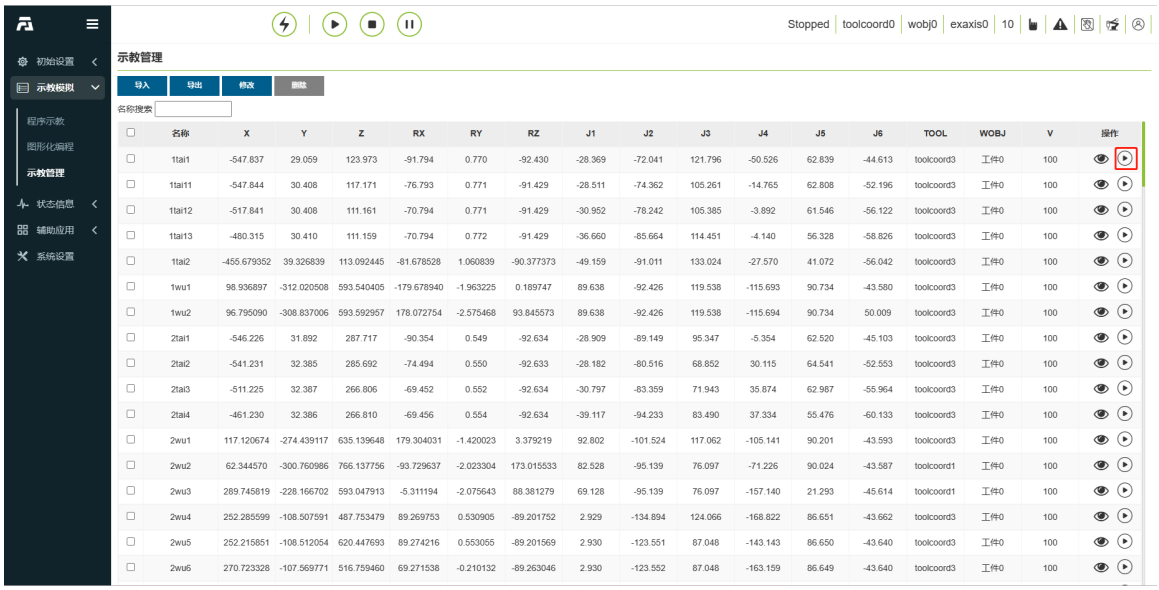
重要： 示教点 x,y,z,rx,ry,rz 的修改值不应超过机器人的工作范围。

详情：点击“详情”按钮，查看示教点的详情。



图表 3.7-39 示教点详情

运行：点击“开始运行”按钮，进行局部示教点的单点运行，将机器人移动到该点的位置。

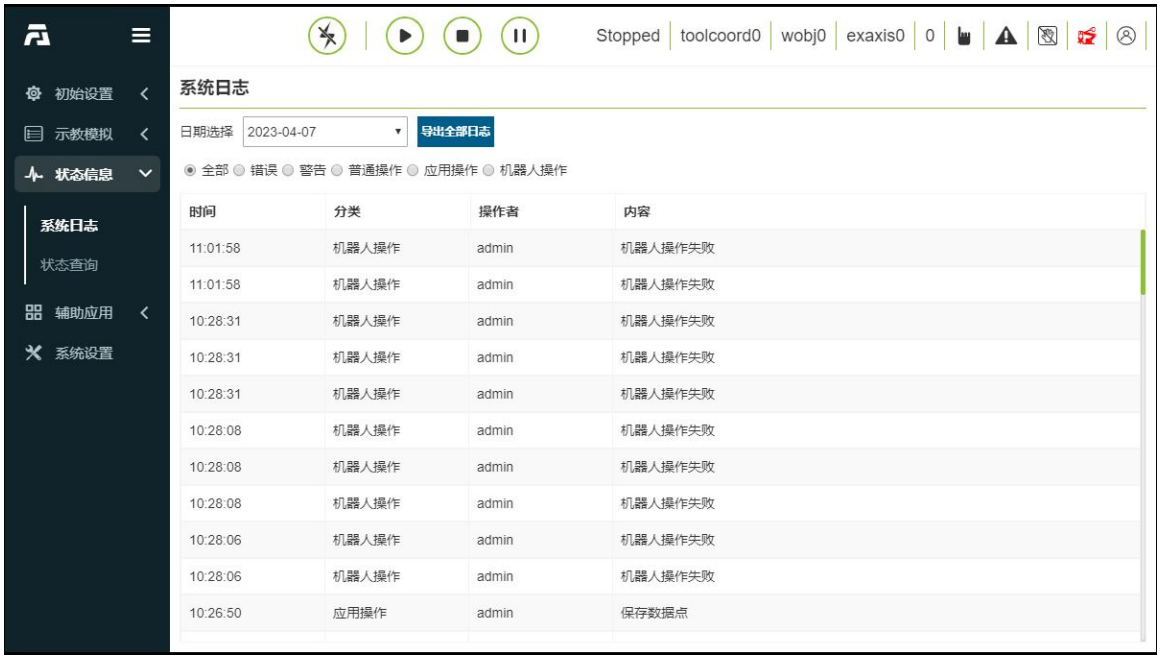


图表 3.7-40 运行示教点

1.3.5.8 状态信息

1.3.5.8.1 系统日志

点击左侧菜单栏“状态信息”中“系统日志”按钮，进入日志显示界面。日志中记录着示教器的一些重要操作记录，如登录者、示教点增减等，点击进入后默认显示当天的日志记录，若要查询以前的记录，则在“日期选择”中选择目标日期，下方会实时显示当天日志记录，日志记录份数设置详见系统设置中。用户日志信息过多时，用户可以根据类型去查找相关日志信息。



图表 3.8-1 系统日志界面

1.3.5.8.2 状态查询

点击左侧菜单栏“状态信息”中“状态查询”菜单进入状态查询界面，如图表 3.8-2 状态查询。

图表 3.8-2 状态查询

状态查询操作步骤：

Step1：点击“图表”按钮弹出图表设置弹出框如图表 3.8-3 图表设置所示，查询类型选择图表展示查询，在图表设置中选择所需查询的参数以及参数放入的图表，点击“右移”按钮即可将参数配置到图表中。点击“设置”则下发设置图表指令。目前只能支持一张表格中最多包含四个待查参数并且最多设置一张图表；

图表 3.8-3 图表设置

Step2：触发功能暂时不需要设置，点击“查询”按钮即可查询数据。

1.3.5.9 辅助应用

1.3.5.9.1 机器人打包

在“辅助应用”中的“机器人本体”的菜单栏下，点击“机器人打包”按钮，进入机器人一键打包界面。

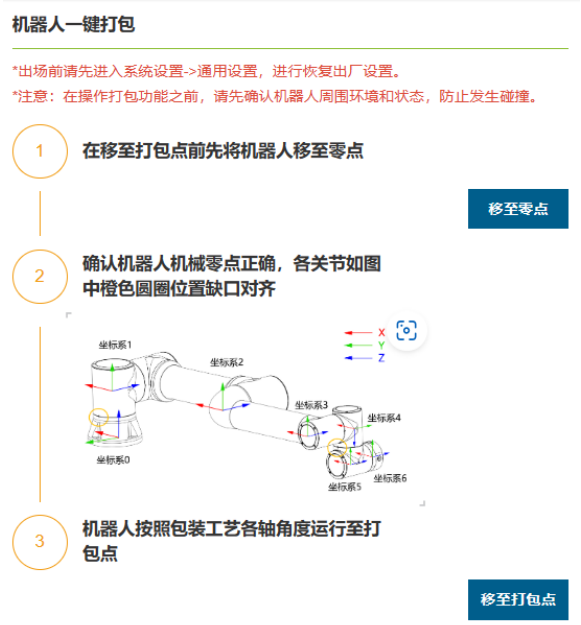
重要： 在操作打包功能之前，请先确认机器人周围环境和状态，防止发生碰撞。

若出厂，则出厂前先进去系统设置-通用设置，进行恢复出厂设置。

Step1：在移至打包点前先将机器人移至零点

Step2：点击“移至零点”按钮，确认机器人机械零点正确，各关节如图中橙色圆圈位置缺口对齐。

Step3：点击“移至打包点”按钮，机器人按照包装工艺各轴角度运行至打包点。

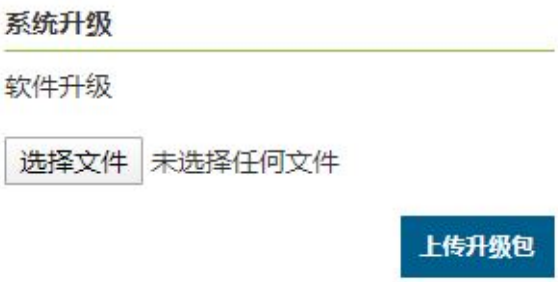


图表 3.9-1 机器人一键打包

1.3.5.9.2 系统升级

在“辅助应用”中的“机器人本体”的菜单栏下，点击“系统升级”按钮，进入系统升级界面。系统升级分为软件升级、驱动器升级和系统关机。

软件升级：在软件升级下点击“上传文件”，选择 U 盘中的 software.tar.gz 升级包，点击上传升级包，升级按钮旁显示“上传中…上传百分比”。待后台文件下载完成，界面显示“上传完成，正在升级中”，进行文件 MD5 和版本号检测，通过后，解密解压升级文件，并提示”升级成功，请重新启动控制箱!”，如果其中检测，解压或发生其他错误，升级按钮旁显示“升级失败”。



图表 3.9-2 系统升级

重要： 软件升级包名为确定的 software.tar.gz，如果升级包名与之不一致，那么会出现升级失败，修改为确定的升级包名称即可。

固件升级：机器人进入 BOOT 模式后，上传升级压缩包，选择需要升级的从站（控制箱从站，本体驱动器从站 1~6，末端从站），进行升级操作，并显示升级状态。

图表 3.9-3 固件升级

从站配置文件升级：机器人去使能后，上传升级文件，选择需要升级的从站（控制箱从站，本体驱动器从站1~6，末端从站），进行升级操作，并显示升级状态。

图表 3.9-4 从站配置文件升级

1.3.5.9.3 数据备份

在“辅助应用”中的“机器人本体”的菜单栏下，点击“数据备份”进入数据备份界面，如 3.9-5 所示。

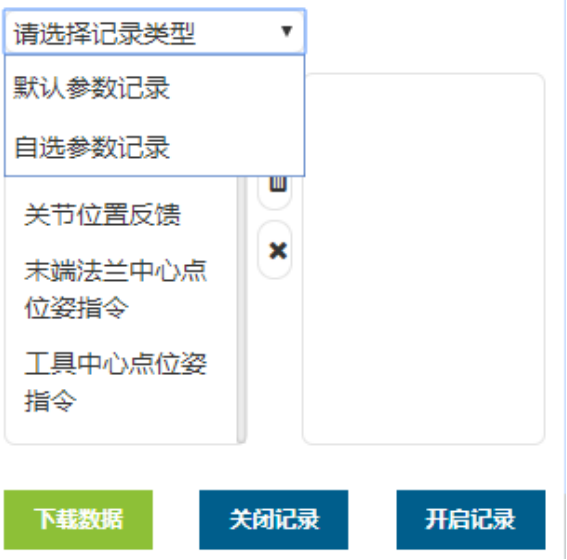
备份包数据中包含工具坐标系数据，系统配置文件，示教点数据，用户程序，模板程序和用户配置文件，当用户需要将本机器人相关数据移到另一台机器人上使用时，可通过此功能快速实现。

图表 3.9-5 数据备份界面

1.3.5.9.4 10s 数据记录

在“辅助应用”中的“机器人本体”的菜单栏下，点击“10s 数据记录”进入 10s 数据记录功能界面。

首先选择记录类型，分为默认参数记录和自选参数记录，默认参数记录为系统自动设置记录的数据，自选参数记录用户可自行选择需要记录的参数数据，参数个数最多为 15 个。选定参数列表后，选择记录参数，点击“右移”按钮即可将参数配置到参数列表中。点击“开始记录”机器人开始记录数据，点击“停止记录”机器人停止记录数，点击“下载数据”可下载最后 10s 的数据。



图表 3.9-6 10s 数据记录

1.3.5.9.5 示教点配置

在“辅助应用”中的“机器人本体”的菜单栏下，点击“示教点配置”进入示教点配置功能界面。

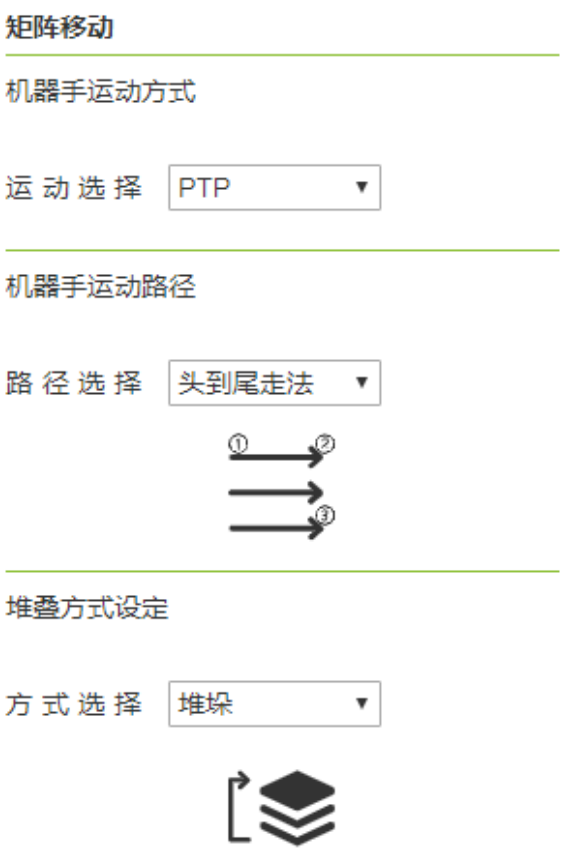
用户在使用按钮盒或其它 IO 信号记录示教点功能前，首先对示教点名称前缀，编号上限和示教方法进行配置，名称前缀支持自定义前缀和以当前程序名作为前缀两种模式。例如，自定义名称前缀“P”，编号上限“3”，示教方法“机器人示教”，记录机器人当前末端（工具）点依次为：P1、P2、P3，再次记录将覆盖之前记录点。

图表 3.9-7 示教点配置

1.3.5.9.6 矩阵移动

在“辅助应用”中的“机器人本体”的菜单栏下，点击“矩阵移动”进入矩阵移动配置功能界面。

此功能通过设定三点坐标及行列层和层高等数值，来控制机器手规则移动，适用于常见的码垛应用。第一步选择机器人运动方式，“PTP”或者“Line”，第二步设定机器人运动路径，“头到尾走法”或“弓字形走法”，第三步设定堆叠方式，“堆垛”或“卸垛”。



图表 3.9-8 矩阵移动

第四步根据路径示教三个点，第一点为第一排起点，整个运动过程手臂姿态由该点决定，第二点为第一排终点，第三点为最后一排终点。第五步设点行数和列数，第六步设定层数和每一层高度，最后一步，命名该矩阵运动程序文件，一个矩阵移动程序生成成功。

图表 3.9-9 矩阵移动

1.3.5.9.7 作业原点

在“辅助应用”中的“机器人本体”的菜单栏下，点击“作业原点”进入作业原点配置功能界面。

该页面显示作业原点的名称和关节位置信息，作业原点命名为固定名 pHome，点击“设置”以当前机器人位姿作为作业原点，点击“移至该点”机器人会运动到作业原点。此外 DI 配置中增加移动至作业原点可配置选项，DO 配置中增加到达作业原点可配置选项。

图表 3.9-10 作业原点

1.3.5.9.8 干涉区配置

在“辅助应用”中的“机器人本体”的菜单栏下，点击“干涉区配置”进入干涉区配置功能界面。

首先我们需要对干涉方式和进入干涉区操作进行配置，干涉方式分为“轴干涉”和“立方体干涉”，当开启后，会显示激活标志。首先进行进入干涉区运动配置“继续运动”或者“停止”。

图表 3.9-11 干涉区配置

接下来设置进入干涉区拖动配置，用户可以根据需求设置拖动模式下进入干涉区后策略，不限制拖动，阻抗回调和切换回手动模式。

图表 3.9-12 干涉区拖动配置

选择轴干涉，需要对轴干涉的参数进行配置，检测方法分为“指令位置”和“反馈位置”两种，干涉区模式分为“范围内干涉”和“范围外干涉”两种，接下来设置每个关节的范围以及各个关节范围是否使能，可以输入数值，也可以通过“机器人示教”按钮将当前机器人的位置记录到当中，最后点击应用即可。

图表 3.9-13 轴干涉配置

选择立方体干涉，需要对立方体干涉的参数进行配置，检测方法分为“指令位置”和“反馈位置”两种，干涉区模式分为“范围内干涉”和“范围外干涉”两种，参考坐标系分为“基坐标”和“工件坐标”，根据实际使用选择设置。接下来设进行范围设置，范围设置分为两种方法，首先看第一种方法“两点法”，即立方体的两个对角的顶点组成，我们可以通过输入或者机器人示教记录位置。最后点击应用即可。

图表 3.9-14 立方体干涉配置

接下来看第二种方法“中心点 + 边长”，即立方体的中心位置点和立方体的边长构成干涉区，我们可以通过输入或者机器人示教记录位置。最后点击应用即可。

图表 3.9-15 立方体干涉配置

1.3.5.9.9 末端 LED 配置

在“辅助应用”中的“机器人本体”的菜单栏下，点击“末端 LED 配置”进入末端 LED 颜色配置功能界面。

可配置 LED 颜色为绿色，蓝色和白青色，用户可以根据需求对自动模式，手动模式和拖动模式的 LED 颜色进行配置，不同模式不可配置同一种颜色。

图表 3.9-16 末端 LED 配置

1.3.5.9.10 外设协议

在“辅助应用”中的“机器人本体”的菜单栏下，点击“外设协议”进入外设协议配置功能界面。

该页面是对外设协议的配置页面，用户可以根据当前使用的外设进行协议配置。

图表 3.9-17 外设协议配置

在程序示教中增加基于 Modbus-rtu 通讯的读写寄存器 lua 接口，输入寄存器地址 0x1000 寄存器数量为 50 个，共 100 字节数据内容；保持寄存器地址 0x2000，寄存器数量为 50 个，共 100 字节数据内容。

```
ModbusRegRead（fun_code，reg_add，reg_num）：读寄存器；

fun_code：  功能码，0x03-保持寄存器，0x04-输入寄存器

reg_add：  寄存器地址

reg_num：  寄存器数量
```

```
ModbusRegWrite（fun_code，reg_add，reg_num，reg_value）：写寄存器；

fun_code 功能码，0x06-单个寄存器，0x10-多个寄存器

reg_add:  寄存器地址

reg_num:  寄存器数量

reg_value:  字节数组
```

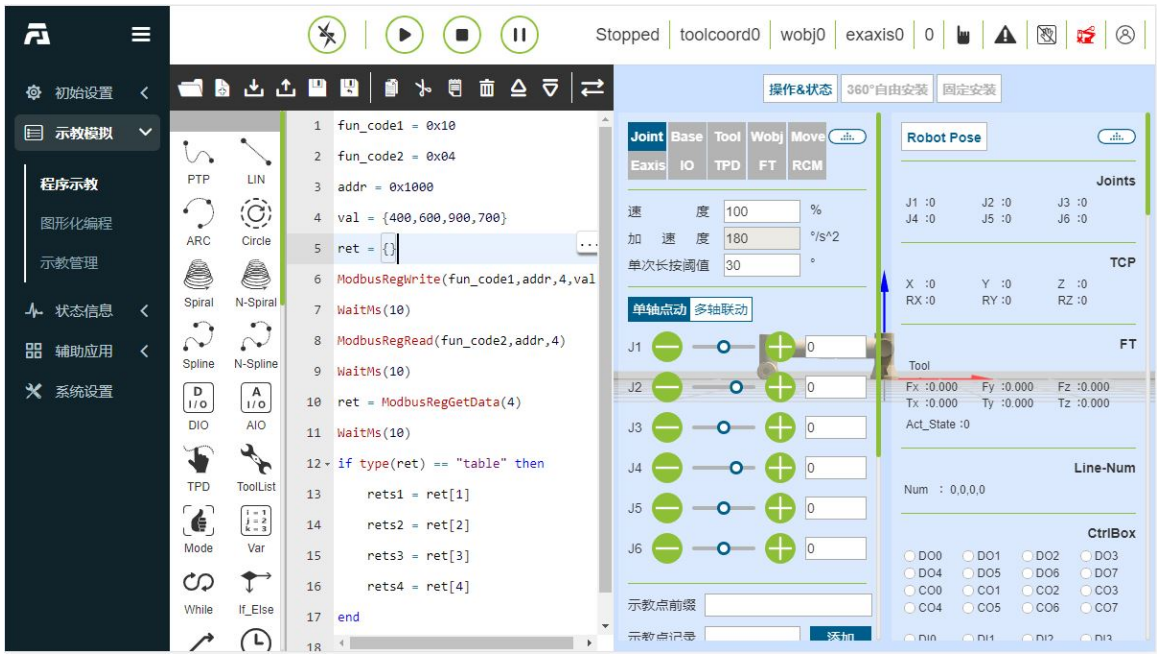
```
ModbusRegGetData（reg_num）：获取寄存器数据；

reg_num:  寄存器数量

返回值说明：

reg_value: 数组变量
```

程序示例截图：



图表 3.9-18 Modbus-rtu 通讯 lua 程序示例

1.3.5.9.11 主程序配置

在“辅助应用”中的“机器人本体”的菜单栏下，点击“主程序配置”进入主程序配置功能界面。

配置主程序可以与 DI 配置主程序启动配合使用，配置的主程序需要先试运行以确保安全，在机器人设置中配置对应 DI 为启动主程序信号功能后，用户可以控制该 DI 信号实现运行主程序。

图表 3.9-19 主程序配置

1.3.5.9.12 拖动锁定

在“辅助应用”中的“机器人本体”的菜单栏下，点击“拖动锁定”进入拖动示教锁定配置功能界面。

针对拖动示教增加了锁定自由度功能，当拖动示教功能开关设置为使能状态时，各自由度参数在用户拖动机器人时生效。例如，当参数设置为 X:10，Y:0，Z:10，RX:10，RY:10，RZ:10 时，在拖动模式下拖拽机器人，可以限制机器人只移动 Y 方向，假如需要在拖动时保持机器人姿态不变，只移动 X，Y，Z 方向，可以将 X，Y，Z 设置为 0，RX，RY，RZ 设置为 10。

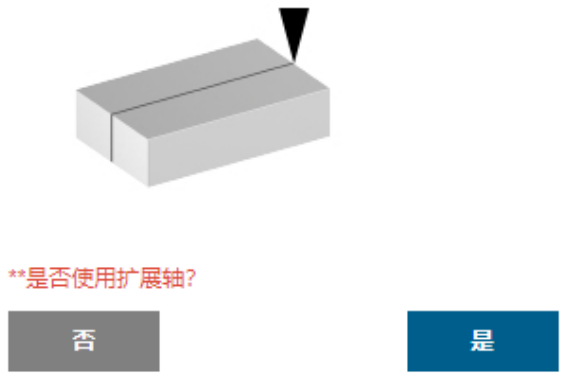
图表 3.9-20 拖动示教锁定配置

1.3.5.9.13 焊接专家库

点击“辅助应用”中的“焊接专家库”的菜单栏，进入焊接专家库功能界面。焊接专家库分为焊件形状，零件设计，夹具结构和配置四部分功能。

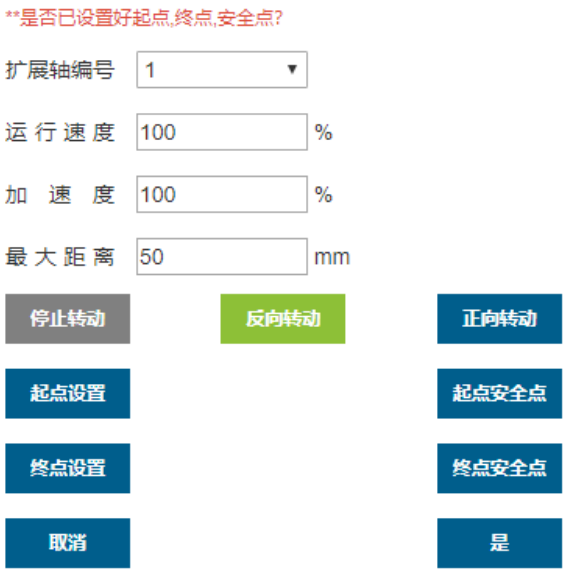
点击“焊件形状”下的“直焊”，进入直焊指导界面。在各项机器人基础设置配置完成的基础上，我们可以通过几个简单的步骤快速生成焊接示教程序。主要包含以下五个步骤，由于功能间存在互斥，所以实际生成一个焊接示教程序的步骤少于五步。

步骤一，是否使用扩展轴，如果使用扩展轴需要配置好扩展轴相关坐标系以及使能扩展轴。



图表 3.9-21 扩展轴配置

步骤二，标定起点，起点安全点，终点，终点安全点。若第一步选择了扩展轴，会加载扩展轴移动功能，配合相关点的标定。



图表 3.9-22 标定相关点

步骤三，选择是否需要激光，如果是的话，需要编辑激光寻位指令的参数。

图表 3.9-23 激光寻位配置

步骤四，选择是否需要摆焊，如果需要摆焊，需要编辑摆焊相关参数。

图表 3.9-24 摆焊配置

步骤五，给程序命名，并在程序示教界面中自动打开该程序。

图表 3.9-25 保存程序

点击“焊件形状”下的“圆弧焊”，进入圆弧焊指导界面。在各项机器人基础设置配置完成的基础上，我们可以通过两个简单的步骤快速生成焊接示教程序。主要包含以下两个步骤。

步骤一，标定起点，起点安全点，圆弧过渡点，终点和终点安全点。

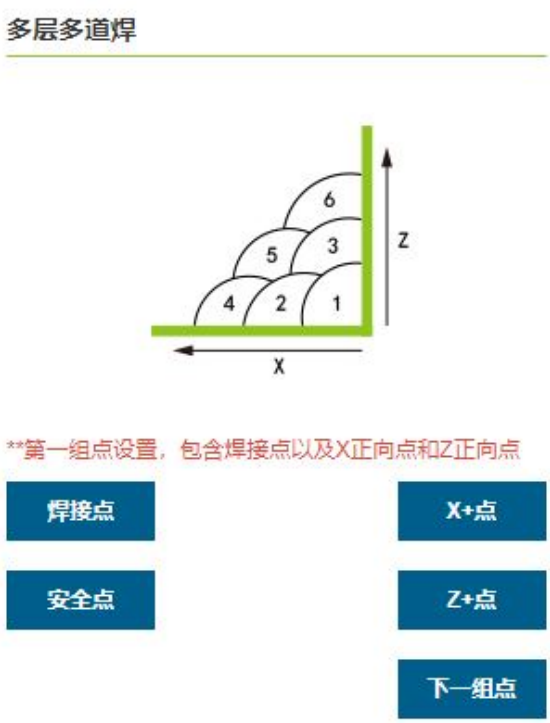
图表 3.9-26 标定点

步骤二，给程序命名，并在程序示教界面中自动打开该程序。

图表 3.9-27 保存程序

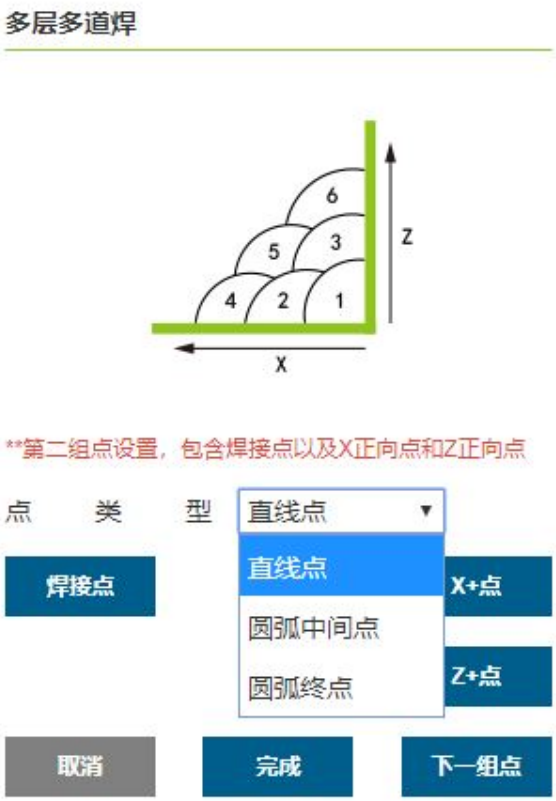
点击“焊件形状”下的“多层多道焊”，进入多层多道焊指导界面。在各项机器人基础设置配置完成的基础上，我们可以通过四个简单的步骤快速生成焊接示教程序。主要包含以下五个步骤。

步骤一，根据提示设置第一组点，即焊接点，X+ 点，Z+ 点以及安全点。



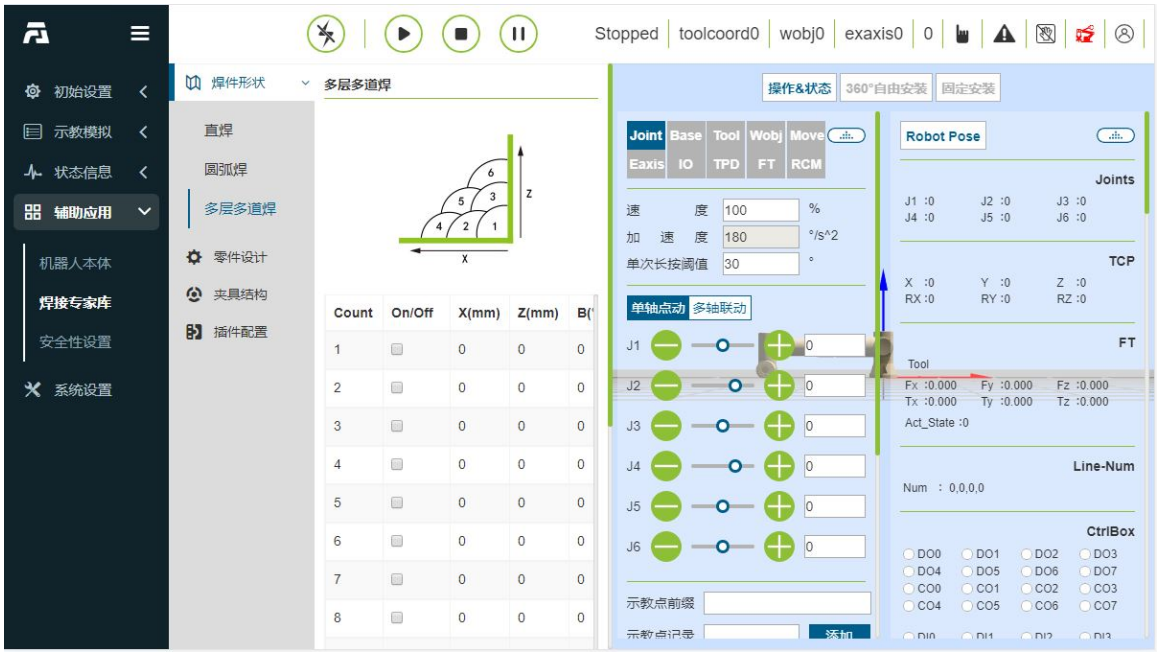
图表 3.9-28 第一组点设置

步骤二，第二组点设置，可以设置路径点的类型，支持直线和圆弧路径，包括焊接点，X+ 点和 Z+ 点。



图表 3.9-29 第二组点设置

步骤三，所有组点设置完成后，点击“完成”进入各个焊道偏移量设置功能页面，依次设置所需焊道的偏移量，界面如下图所示。



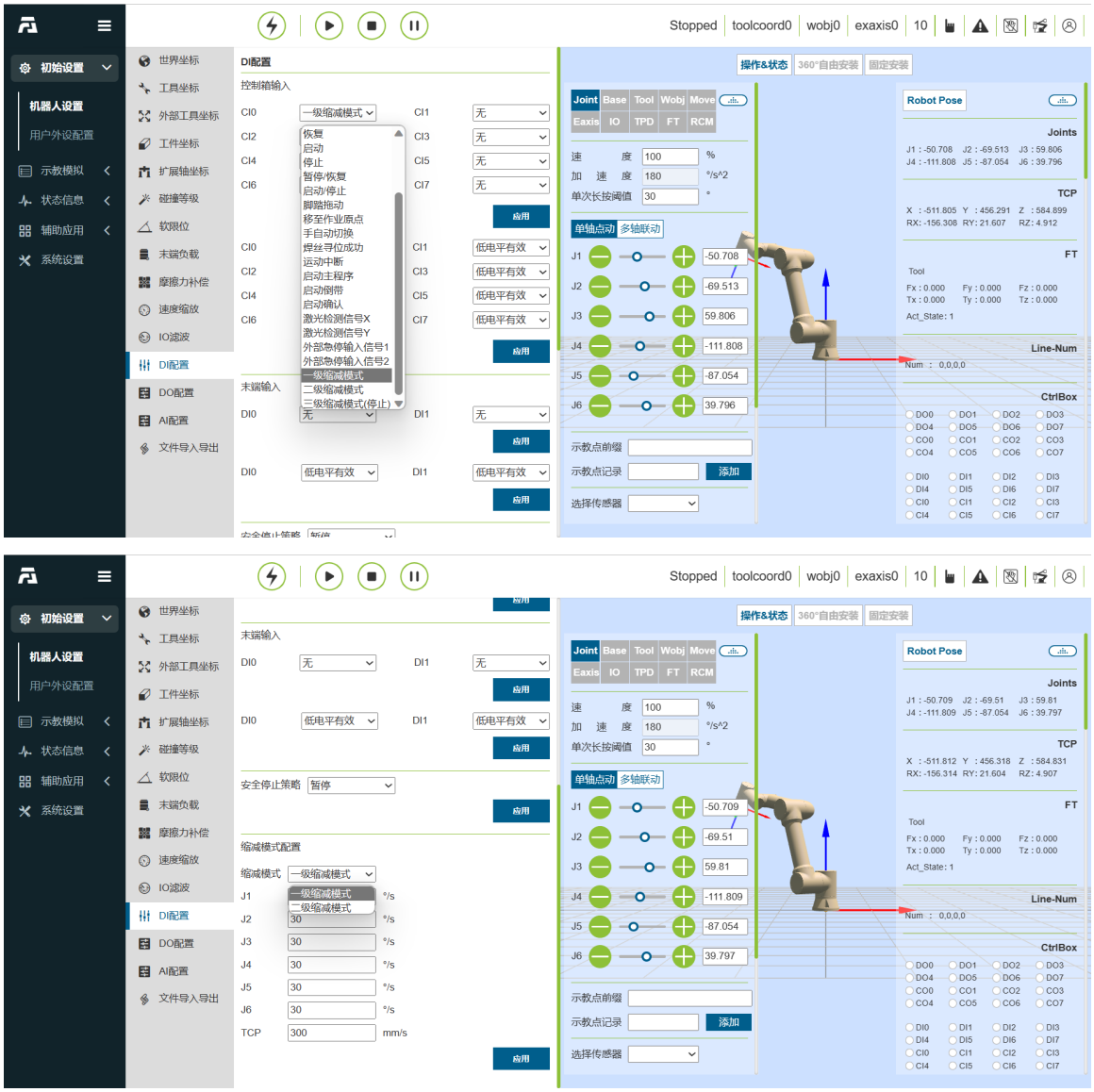
图表 3.9-30 焊道偏移量设置

步骤四，当所需设置焊道参数都设置完成后，点击“完成”跳转到程序生成页面，输入文件名，即可生成该多层多道焊程序，之后用户可在程序示教中打开该程序，进行调试，界面如下图所示。

图表 3.9-31 保存程序

1.3.5.9.14 安全速度设置

在“机器人设置”中的“DI 配置”下，点击不同的 DI 下拉框，可以配置缩减模式（一级、二级、三级）一级和二级缩减模式可以配置关节速度和末端 TCP 速度，三级缩减模式是停止可以不用配置速度。

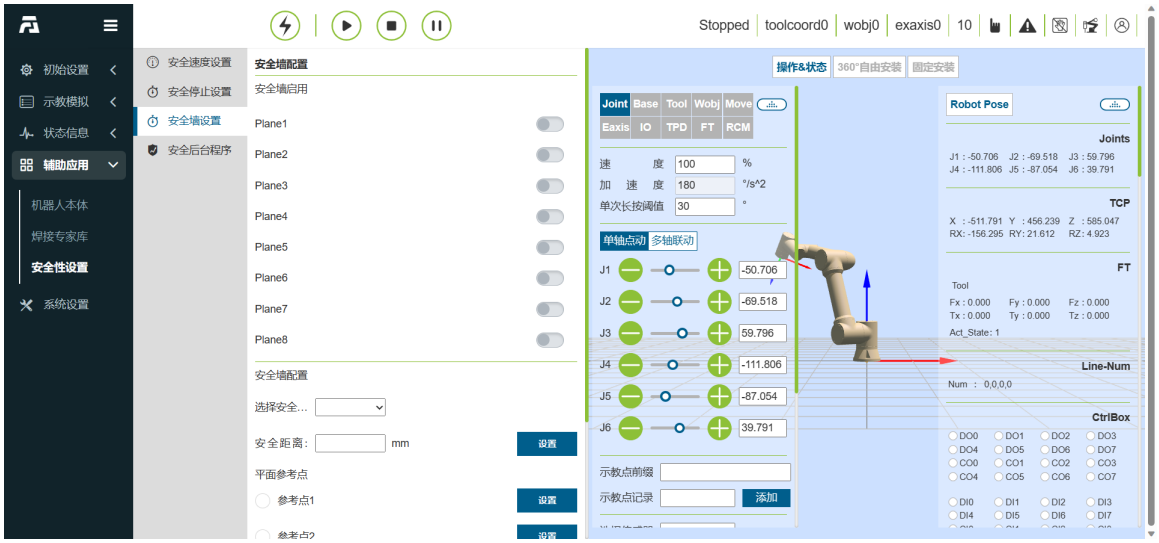


图表 3.9-32 缩减模式配置

1.3.5.9.15 安全墙配置

在“辅助应用”中的“安全性设置”的菜单栏下，点击“安全墙配置”进入安全墙配置功能界面。

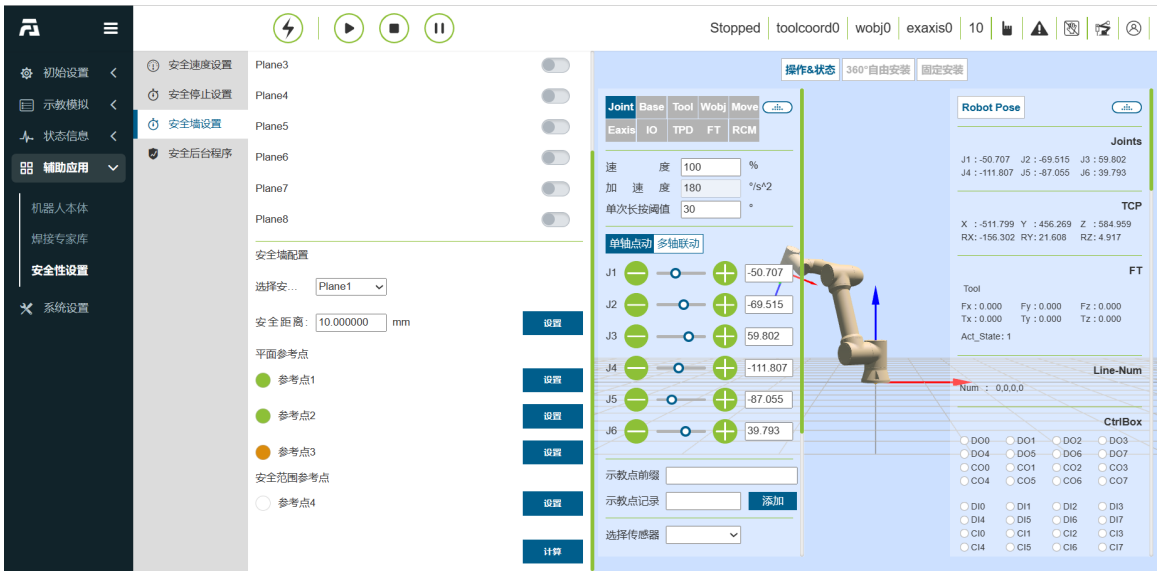
- **安全墙配置**：点击启用按钮，可启用的相应的安全墙。当安全墙未配置安全范围时，会提示报错。点击下拉框，选择想要设定的安全墙，自动带出安全距离 (可不设置，默认为 0), 再点击“设置”按钮，设置成功。



图表 3.9-33 安全墙配置

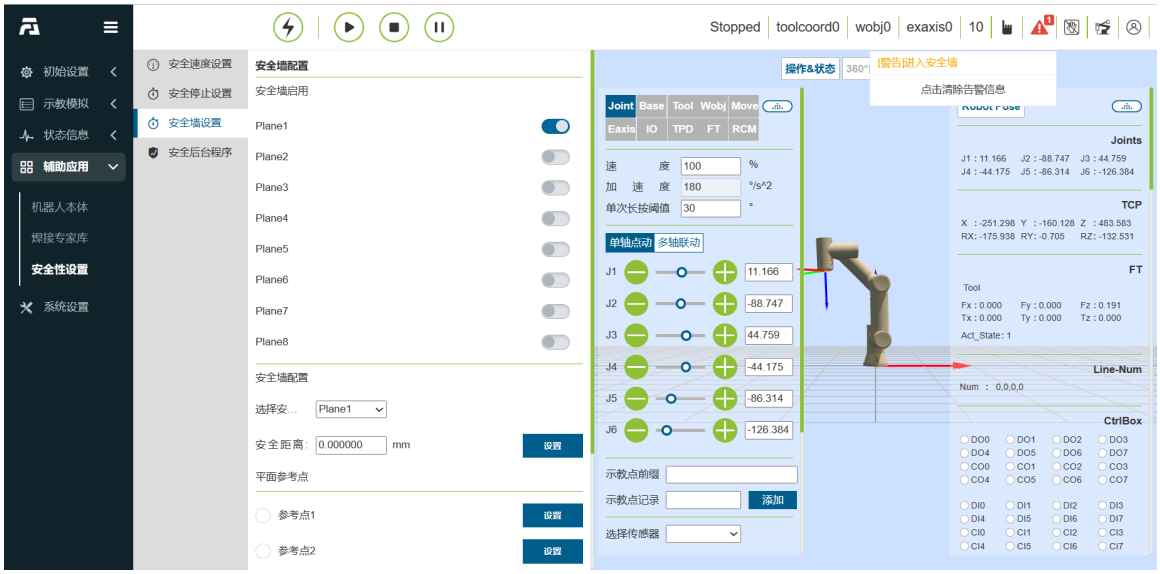
- **安全墙参考点配置**：选择安全墙后，可设置四个参考点。前三个点为平面参考点，用来确认设置的安全墙的平面。第四个点为安全范围参考点，用来确认设置的安全墙的安全范围。

重要： 若参考点设置成功，则亮绿灯。反之，则亮黄灯。直到参考点设置成功后转为绿灯。当四个参考点都设置成功后，可计算其安全范围，计算成功后安全范围参数点状态恢复默认。



图表 3.9-34 安全范围参考点设置

- 应用效果：启用配置成功的安全墙。拖动机器人，若机器人末端 TCP 处在设定安全范围内，则系统正常。若处在设定安全范围之外，则提示报错。



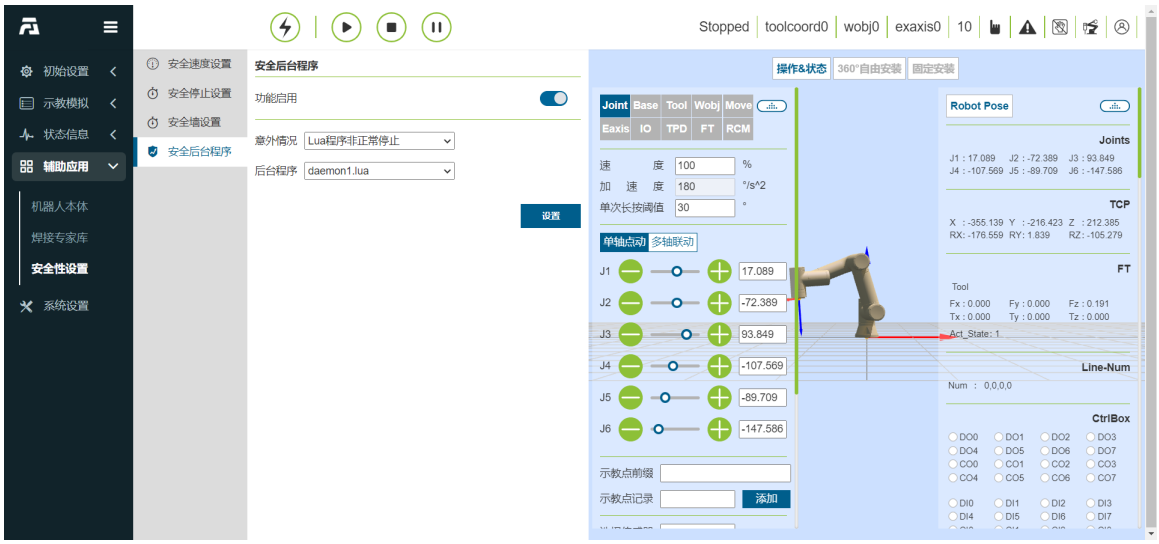
图表 3.9-35 安全范围设置成功后效果图

1.3.5.9.16 安全后台程序

在“辅助应用”中的“安全性设置”的菜单栏下，点击“安全后台程序”进入安全后台程序功能界面。

用户点击“功能启用”按钮打开或者关闭安全后台程序的设置。选择”意外情况“和”后台程序”，点击“设置”按钮配置意外情况处理逻辑的参数。

启用安全后台程序并设置意外情况场景和后台程序，当用户开始运行程序，发生的意外情况场景与设置的意外情况匹配时，机器人会执行相对应的后台程序，起到安全防护的作用。



图表 3.9-36 安全后台程序

1.3.5.10 系统设置

1.3.5.10.1 通用设置

点击左侧菜单栏“系统设置”，点击二级菜单栏“通用设置”，进入通用设置界面。通用设置可以根据当前电脑时间更新机器人系统时间，以便记录日志内容时间准确。

网络设置可以设置控制器 IP，子网掩码，默认网关，DNS 服务器和示教器 IP(使用我们的 FR-HMI 示教器情况下该 IP 有效，在使用 FR-HMI 示教器情况下需要配置示教器启用状态为启用)，方便客户使用场景。

客户可以根据需求选择示教器语言为中文，英文，法语或日文。此外，语言可支持用户自定义，用户可以通过导出的中文语言文件，进行翻译，将翻译后语言文件导入进去，即可选择使用导入后的语言文件。

用户可以对日志保留数进行设置和系统配置文件的导入导出，日志保留数最大为 30，系统配置文件记录着该设置数值。

系统恢复下恢复出厂设置可以清除用户数据，使机器人恢复到出厂配置。

从站日志生成和控制器日志导出功能为下载控制器一些重要的状态或报错的记录文件，方便排查机器人问题。

1.3.5.10.1.1 网络设置

图表 3.10-1 网络设置示意图

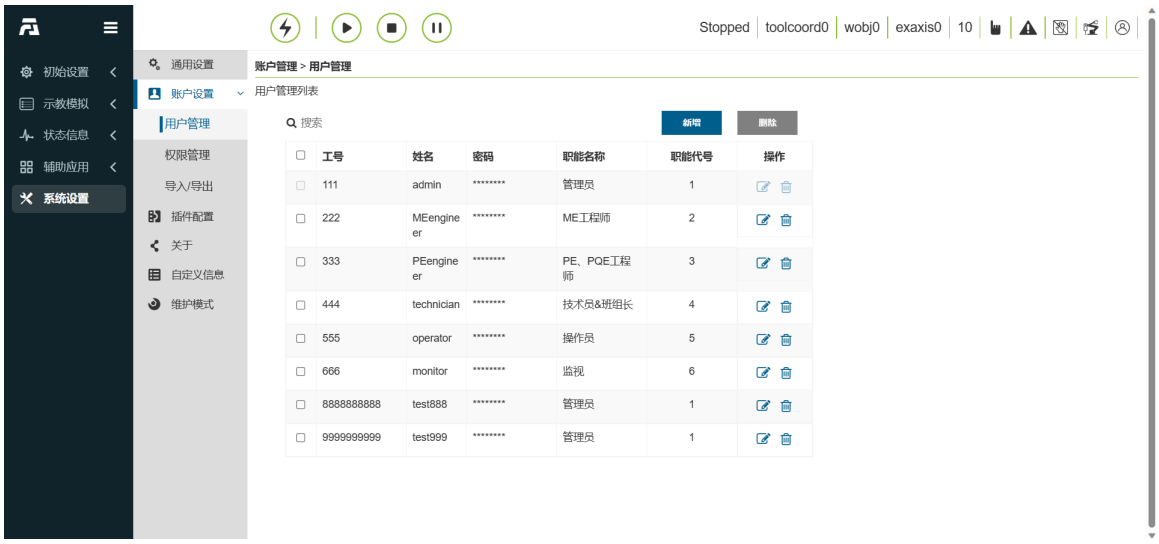
- **设置网卡**：输入需要通信的网卡 IP、子网掩码（与 IP 联动，自动填写）、默认网关、DNS 服务器。网卡 0 网口出厂默认 IP：192.168.57.2，网卡 1 网口出厂默认 IP：192.168.58.2。
- **示教器启用**：控制是否启用示教器。默认示教器关闭，无法使用示教器操作设备。点击滑动开关按钮，则启用示教器操作设备。
- **访问 IP**：选择 WebAPP 和 WebRecovery 关联的网卡，示教器启用时，WebAPP 默认选择网卡 1，网卡 0 不可选。
- **设置网络**：点击“设置网络”按钮，提示正在配置中。配置完成后，需要重启设备。

1.3.5.10.2 账户设置

点击二级菜单栏账户设置，进入账户设置界面。账户管理功能仅限管理员可使用。功能分以下三个模块：

1.3.5.10.2.1 用户管理

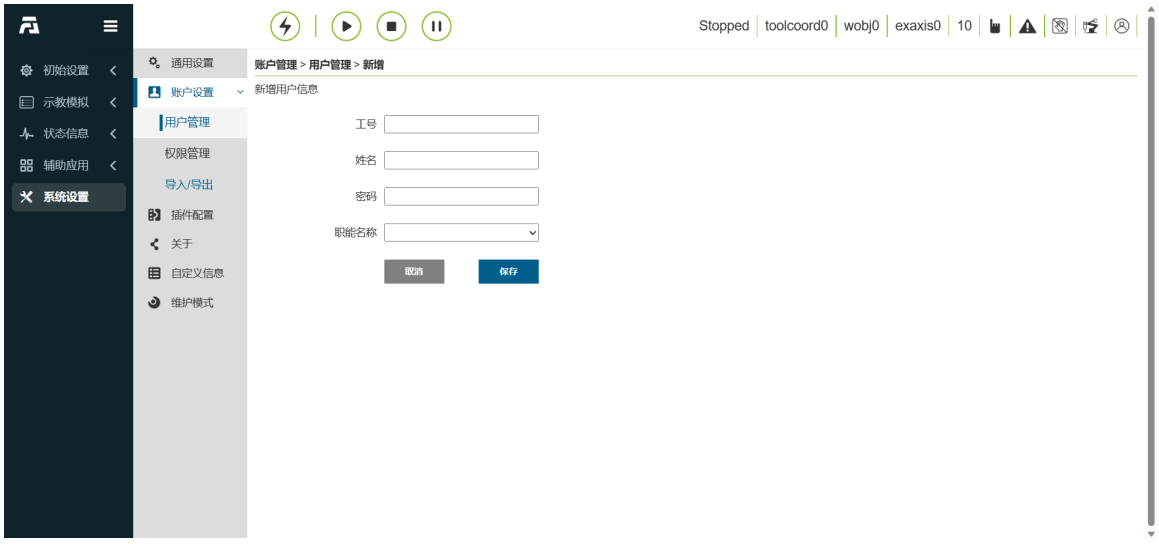
用户管理页面，用来保存用户信息，可以添加用户的工号、职能等。用户可通过输入用户列表中已有的用户名和密码进行登录操作。



图表 3.10-2 用户管理

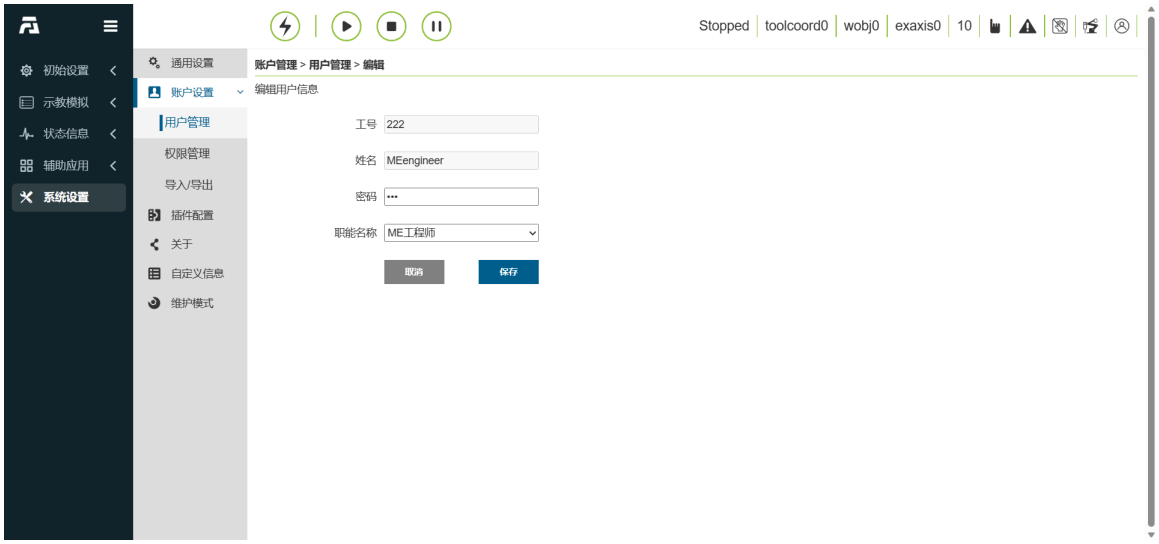
- **新增用户**：点击“新增”按钮，输入工号、姓名、密码并选择职能。

重要： 工号最大为 10 位整数型，工号和密码都有唯一性校验，且密码通过盲文显示。用户新增成功后，可以输入姓名和密码进行重新登录。



图表 3.10-3 新增管理

- **编辑用户**：当存在用户列表时，点击右侧“编辑”按钮，工号和姓名无法修改，可修改密码和职能，密码同样需要唯一性校验。



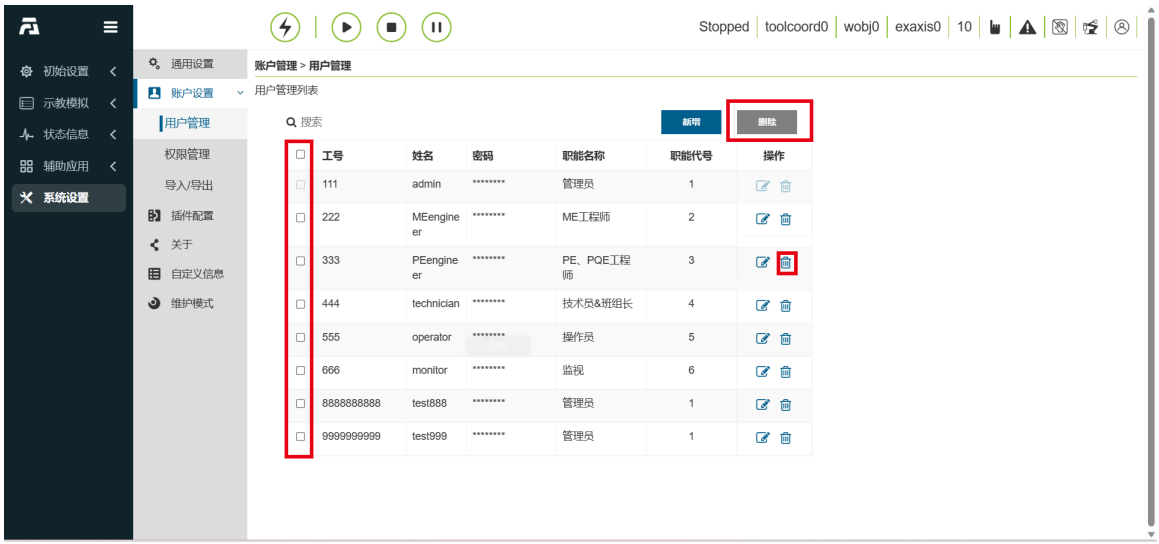
图表 3.10-4 编辑用户

- **删除用户**：删除方法分为单条删除和批量删除。

  1. 点击列表右侧单条“删除”按钮，提示“请再次点击删除按钮以确认删除”，再次点击该列表删除成功。

  2. 点击左侧复选框，选择需要删除的用户，再点击列表上方批量“删除”按钮两次后删除。

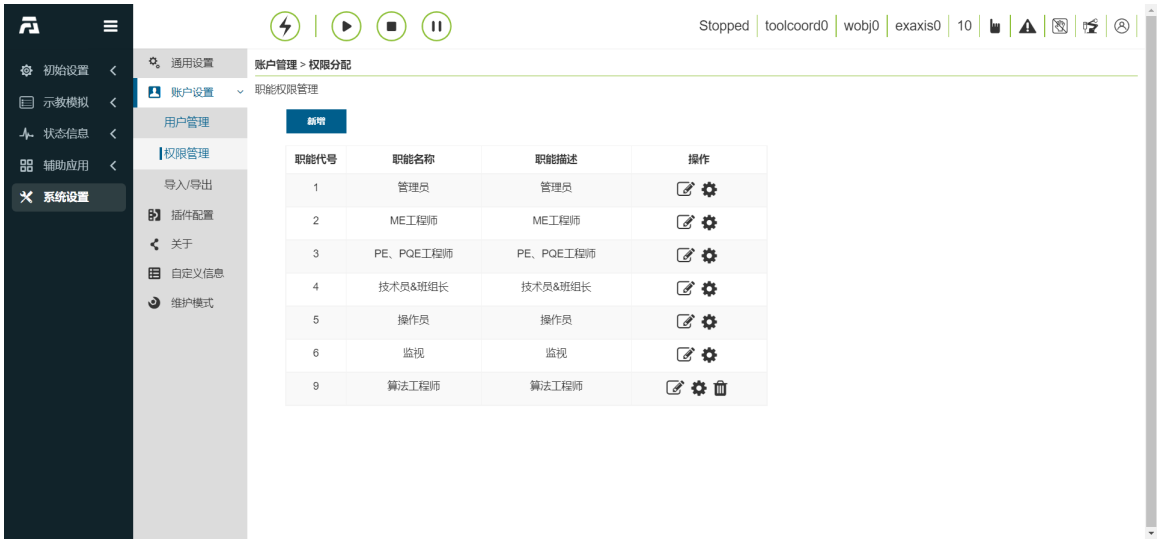
重要： 初始用户 111 以及当前登录用户无法删除。



图表 3.10-5 删除用户

1.3.5.10.2.2 权限管理

重要： 默认的职能数据（职能代号为 1-6）不可以删除，不可修改职能代号，可以修改职能名称和职能描述以及设置职能的权限。



图表 3.10-6 权限管理

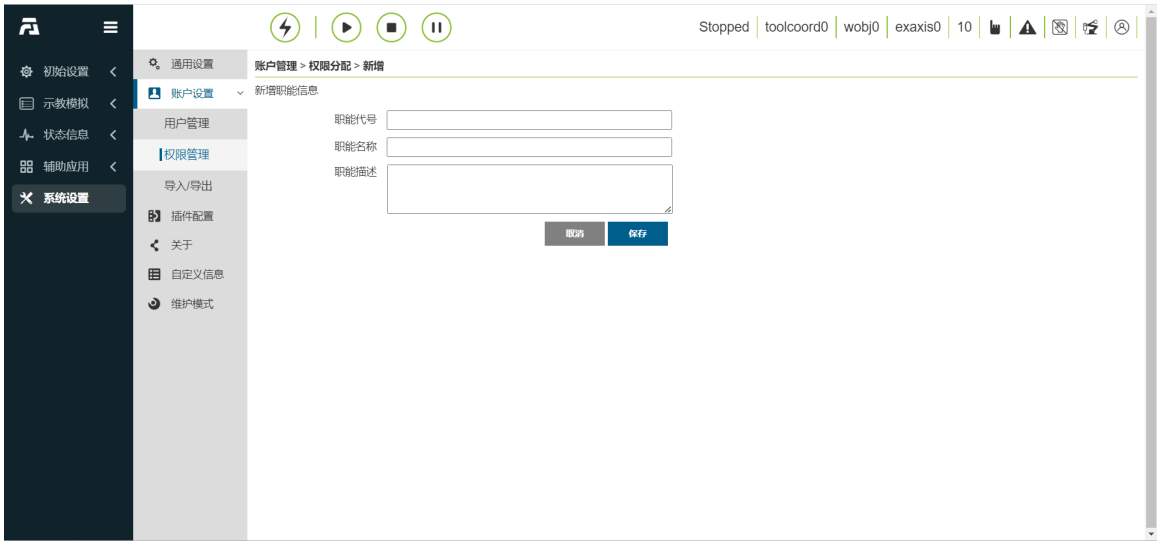
默认六个职能，管理员无功能限制，操作员和监视员少部分功能可以使用，ME 工程师、PE&PQE 工程师和技术员 & 班组长部分功能限制，管理员无功能限制，具体默认权限如下表所示：

重要： 默认权限可修改

类别	权限项	管理员	工程师1(ME)	工程师2(PE,PQE)	技术员&班组长	操作员	监视
可访问视图	机器人设置	有权限	有权限	有权限	有权限	有权限	无权限
	外设配置	有权限	有权限	有权限	有权限	无权限	无权限
	示教编程	有权限	有权限	有权限	有权限	有权限	有权限
	图形化编程	有权限	有权限	有权限	有权限	有权限	有权限
	示教管理	有权限	有权限	有权限	有权限	有权限	有权限
	系统日志	有权限	有权限	有权限	有权限	有权限	有权限
	状态查询	有权限	有权限	有权限	有权限	无权限	无权限
	辅助应用	有权限	有权限	有权限	有权限	无权限	无权限
	焊接专家库	有权限	有权限	有权限	无权限	无权限	无权限
	安全设置	有权限	有权限	有权限	无权限	无权限	无权限
	通用设置	有权限	有权限	有权限	有权限	无权限	无权限
	插件设置	有权限	有权限	有权限	无权限	无权限	无权限
	自定义信息	有权限	有权限	有权限	无权限	无权限	无权限
	维护模式	有权限	无权限	无权限	无权限	无权限	无权限
可操作功能	启动/停止/暂停&恢复	有权限	有权限	有权限	有权限	有权限	无权限
	速度缩放	有权限	有权限	有权限	有权限	无权限	无权限
	手/自动切换	有权限	有权限	有权限	有权限	有权限	无权限
	拖动模式切换	有权限	有权限	有权限	有权限	无权限	无权限
	自由安装	有权限	有权限	有权限	无权限	无权限	无权限
	固定安装	有权限	有权限	有权限	有权限	无权限	无权限
	Joint	有权限	有权限	有权限	有权限	有权限	无权限
	Base	有权限	有权限	有权限	有权限	有权限	无权限
	Tool	有权限	有权限	有权限	有权限	有权限	无权限
	Wobj	有权限	有权限	有权限	有权限	有权限	无权限
	Move	有权限	有权限	有权限	有权限	有权限	无权限
	EAxis	有权限	有权限	有权限	有权限	无权限	无权限
	I/O	有权限	有权限	有权限	有权限	有权限	无权限
	TPD	有权限	有权限	有权限	有权限	有权限	无权限
	FT	有权限	有权限	有权限	有权限	无权限	无权限
	RCM	有权限	有权限	有权限	有权限	无权限	无权限
	示教点记录	有权限	有权限	有权限	有权限	有权限	无权限
	传感器点记录	有权限	有权限	有权限	有权限	有权限	无权限
	基坐标系3D展示	有权限	有权限	有权限	有权限	有权限	有权限
	工具坐标系3D展示	有权限	有权限	有权限	有权限	有权限	有权限
	工件坐标系3D展示	有权限	有权限	有权限	有权限	有权限	有权限
	扩展轴坐标系3D展示	有权限	有权限	有权限	有权限	有权限	有权限
	轨迹绘制	有权限	有权限	有权限	有权限	有权限	无权限
	导入工具模型	有权限	有权限	有权限	有权限	无权限	无权限
	世界坐标	有权限	无权限	无权限	无权限	无权限	无权限
	工具坐标	有权限	有权限	有权限	有权限	有权限	无权限
	外部工具坐标	有权限	有权限	有权限	有权限	有权限	无权限
	工件坐标	有权限	有权限	有权限	有权限	有权限	无权限
	扩展轴坐标	有权限	有权限	有权限	有权限	有权限	无权限
	碰撞等级	有权限	有权限	有权限	有权限	有权限	无权限
	软限位	有权限	有权限	有权限	无权限	无权限	无权限
	末端负载	有权限	有权限	有权限	有权限	有权限	无权限
	摩擦力补偿	有权限	有权限	有权限	有权限	无权限	无权限
	速度缩放	有权限	有权限	有权限	有权限	有权限	无权限
	I/O滤波	有权限	有权限	有权限	有权限	无权限	无权限
	DI配置	有权限	有权限	有权限	有权限	无权限	无权限
	DO配置	有权限	有权限	有权限	有权限	无权限	无权限
	AI配置	有权限	有权限	有权限	有权限	无权限	无权限
	文件导入导出	有权限	有权限	有权限	有权限	无权限	无权限
	末端外设配置	有权限	有权限	有权限	有权限	无权限	无权限
	喷枪配置	有权限	有权限	有权限	有权限	无权限	无权限
	焊机配置	有权限	有权限	有权限	有权限	无权限	无权限
	传感器跟踪	有权限	有权限	有权限	有权限	无权限	无权限
	扩展轴	有权限	有权限	有权限	有权限	无权限	无权限
	传送带跟踪	有权限	有权限	有权限	有权限	无权限	无权限
	跟踪姿态	有权限	有权限	有权限	有权限	无权限	无权限
	扭矩系统	有权限	有权限	有权限	有权限	无权限	无权限
	康养系统	有权限	有权限	有权限	有权限	无权限	无权限
	码垛系统	有权限	有权限	有权限	有权限	无权限	无权限
	打磨设备配置	有权限	有权限	有权限	有权限	无权限	无权限
	导入	有权限	有权限	有权限	有权限	无权限	无权限
	导出	有权限	有权限	有权限	有权限	无权限	无权限
	修改示教点	有权限	有权限	有权限	无权限	无权限	无权限
	删除示教点	有权限	有权限	有权限	有权限	无权限	无权限
	导出日志	有权限	有权限	有权限	有权限	无权限	无权限
	机器人矫正	有权限	无权限	无权限	无权限	无权限	无权限
	编码器配置	有权限	无权限	无权限	无权限	无权限	无权限
	系统升级	有权限	无权限	无权限	无权限	无权限	无权限
	数据备份	有权限	有权限	有权限	有权限	无权限	无权限
	10s数据记录	有权限	有权限	有权限	有权限	无权限	无权限
	示教点配置	有权限	有权限	有权限	有权限	无权限	无权限
	矩阵移动	有权限	有权限	有权限	有权限	无权限	无权限
	作业原点	有权限	有权限	有权限	有权限	无权限	无权限
	干涉区配置	有权限	有权限	有权限	有权限	无权限	无权限
	末端LED配置	有权限	有权限	有权限	有权限	无权限	无权限
	自定义协议	有权限	无权限	无权限	无权限	无权限	无权限
	外设协议	有权限	有权限	有权限	无权限	无权限	无权限
	主程序配置	有权限	有权限	有权限	有权限	无权限	无权限
	拖动锁定	有权限	有权限	有权限	有权限	无权限	无权限
	Smart Tool	有权限	有权限	有权限	有权限	无权限	无权限
	多干涉区配置	有权限	有权限	有权限	有权限	无权限	无权限
	安全速度设置	有权限	有权限	有权限	无权限	无权限	无权限
	安全停止设置	有权限	有权限	有权限	无权限	无权限	无权限
	安全墙	有权限	有权限	有权限	无权限	无权限	无权限
	安全后台程序	有权限	有权限	有权限	无权限	无权限	无权限
	时间设置	有权限	有权限	有权限	有权限	无权限	无权限
	网络设置	有权限	有权限	有权限	有权限	无权限	无权限
	示教器设置	有权限	有权限	有权限	有权限	无权限	无权限
	系统语言	有权限	有权限	有权限	有权限	无权限	无权限
	日志管理	有权限	有权限	有权限	有权限	无权限	无权限
	超时登出时间	有权限	有权限	有权限	有权限	无权限	无权限
	系统文件导出	有权限	有权限	有权限	有权限	无权限	无权限
	信息包上传	有权限	无权限	无权限	无权限	无权限	无权限
	机器人型号名称修改	有权限	无权限	无权限	无权限	无权限	无权限
	示教程序加密	有权限	有权限	有权限	无权限	无权限	无权限

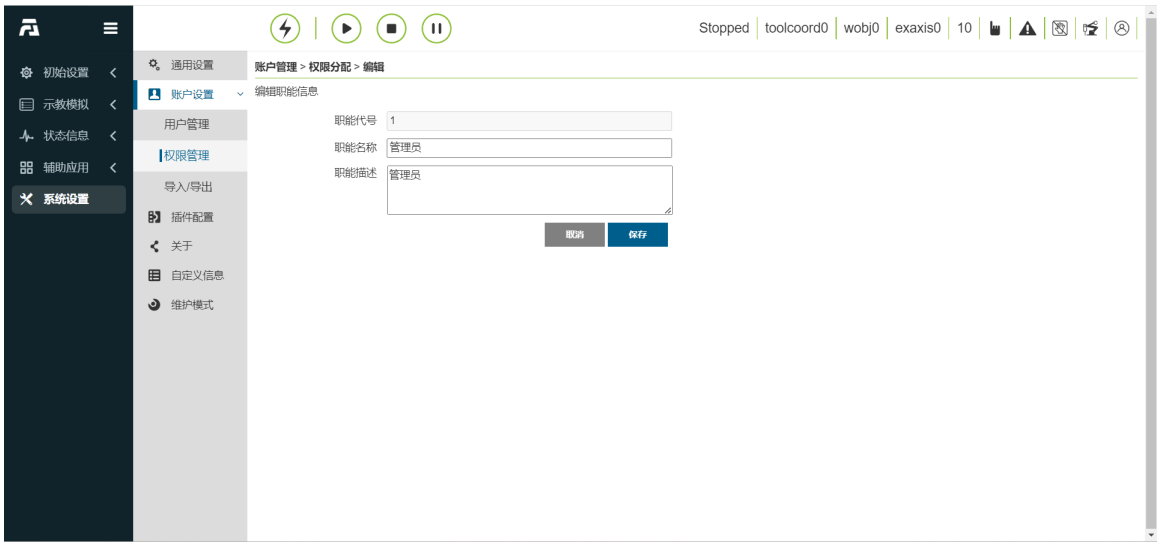
图表 3.10-2 权限详情

- **新增职能**：点击“新增”按钮，输入职能代号、职能名称和职能描述，点击”保存”按钮，成功后返回列表页面。其中职能代号只能为大于 0 的整数并且不能和已经存在的职能代号相同，输入项全部为必填。



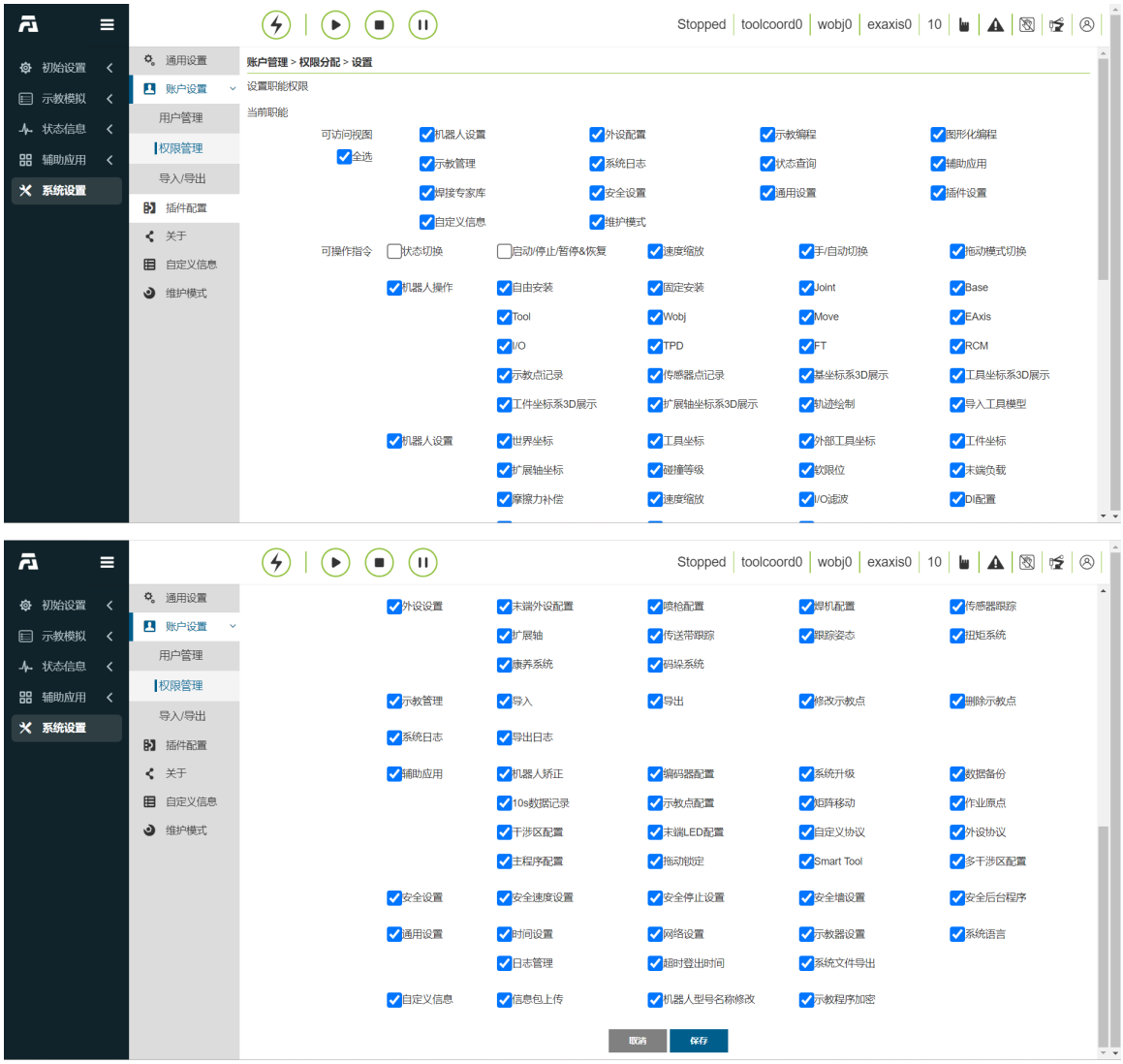
图表 3.10-7 新增职能

- **编辑职能名称和描述**：点击表格操作栏中的“编辑”图标，可以修改当前职能的职能名称和职能描述，修改完成后点击下方”保存”按钮确认修改。



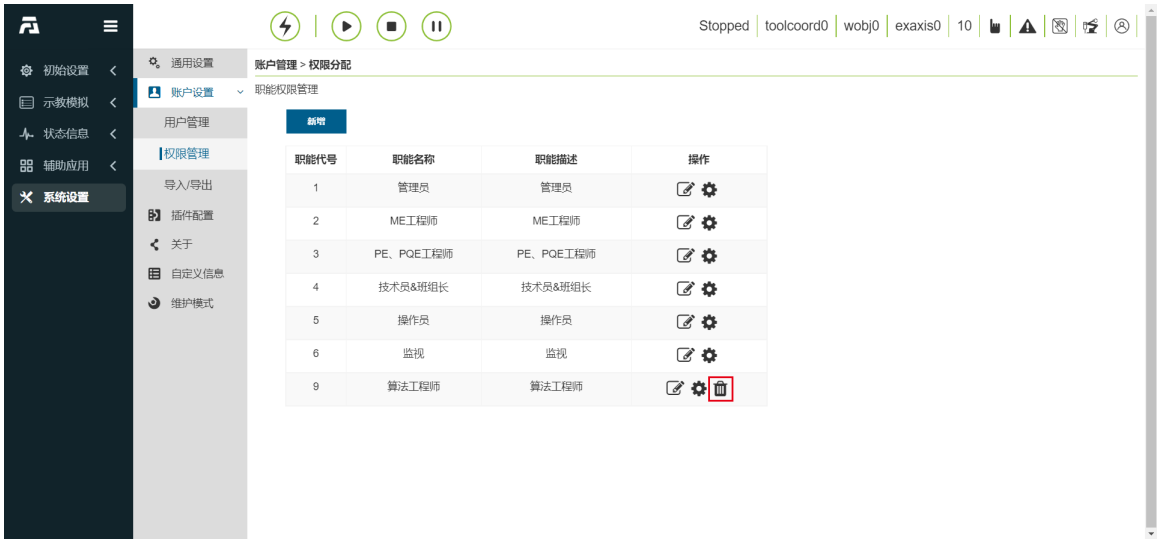
图表 3.10-8 编辑职能

- **设置职能权限**：点击表格操作栏中的“设置”图标，可以设置当前职能的权限，设置完成后点击下方”保存”按钮确实设置。



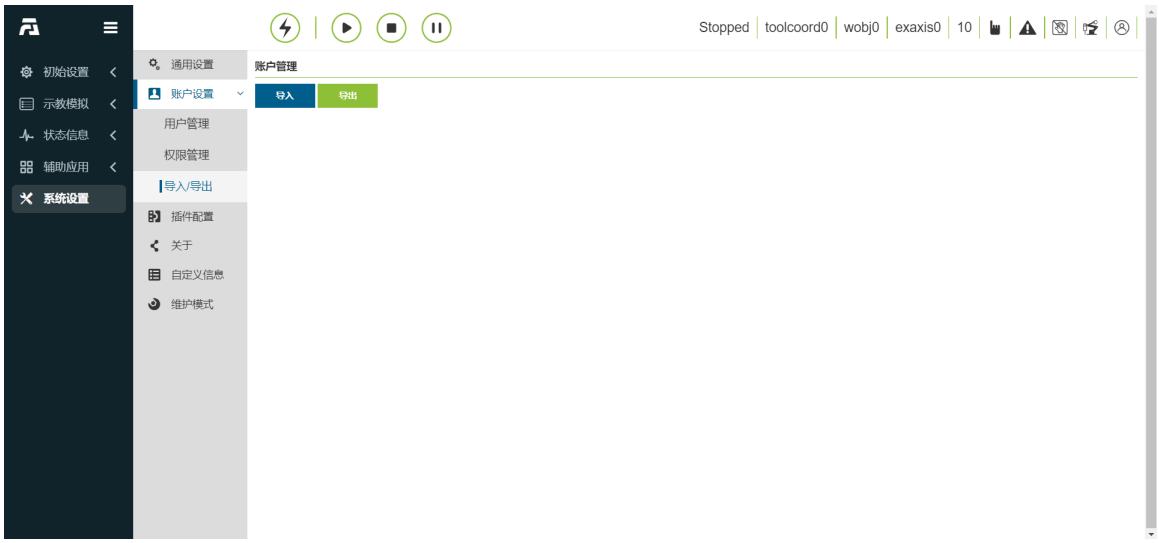
图表 3.10-9 设置职能权限

- **删除职能**：点击表格操作栏中的“删除”图标，首先会校验当前职能是否有用户使用，没有用户使用则可以删除当前职能，反之不可以删除。



图表 3.10-10 删除职能

1.3.5.10.2.3 导入/导出



图表 3.10-11 账户设置导入/导出

- **导入**：点击“导入”按钮，可以批量导入用户管理和权限管理的数据。
- **导出**：点击“导出”按钮，可以批量导出用户管理和权限管理的数据。

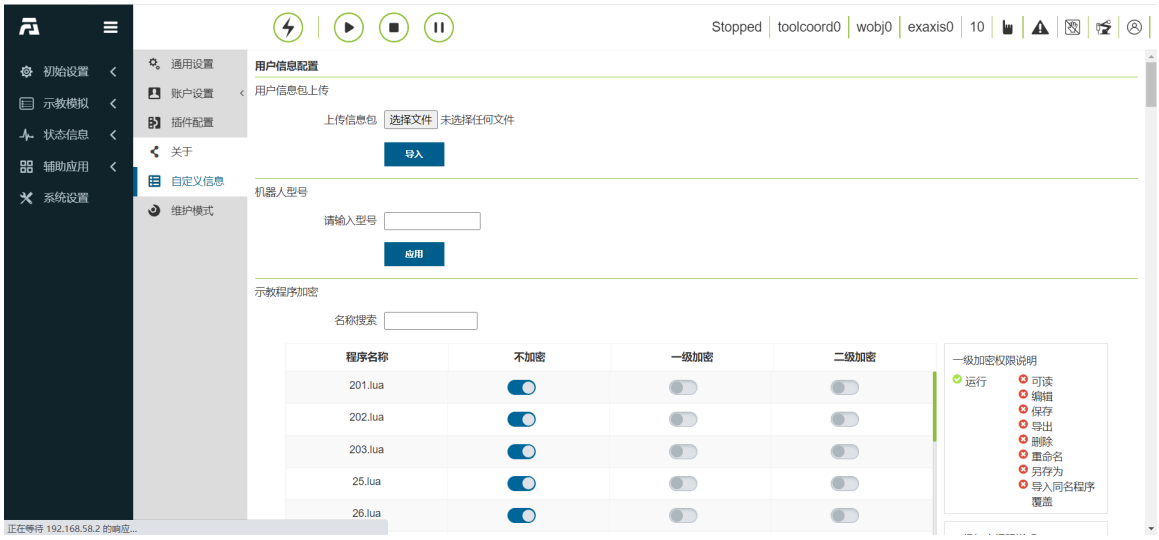
1.3.5.10.3 关于

点击二级菜单栏关于，进入关于界面。该页面展示了机器人的型号和序列号，机器人运行使用的 Web 版本和控制箱版本，硬件版本和固件版本。

图表 3.10-12 关于示意图

1.3.5.10.4 自定义信息

点击二级菜单栏自定义信息，进入自定义信息界面。自定义信息功能仅限管理员可使用。该页面可以上传用户信息包、自定义机器人型号和设置示教程序加密状态。



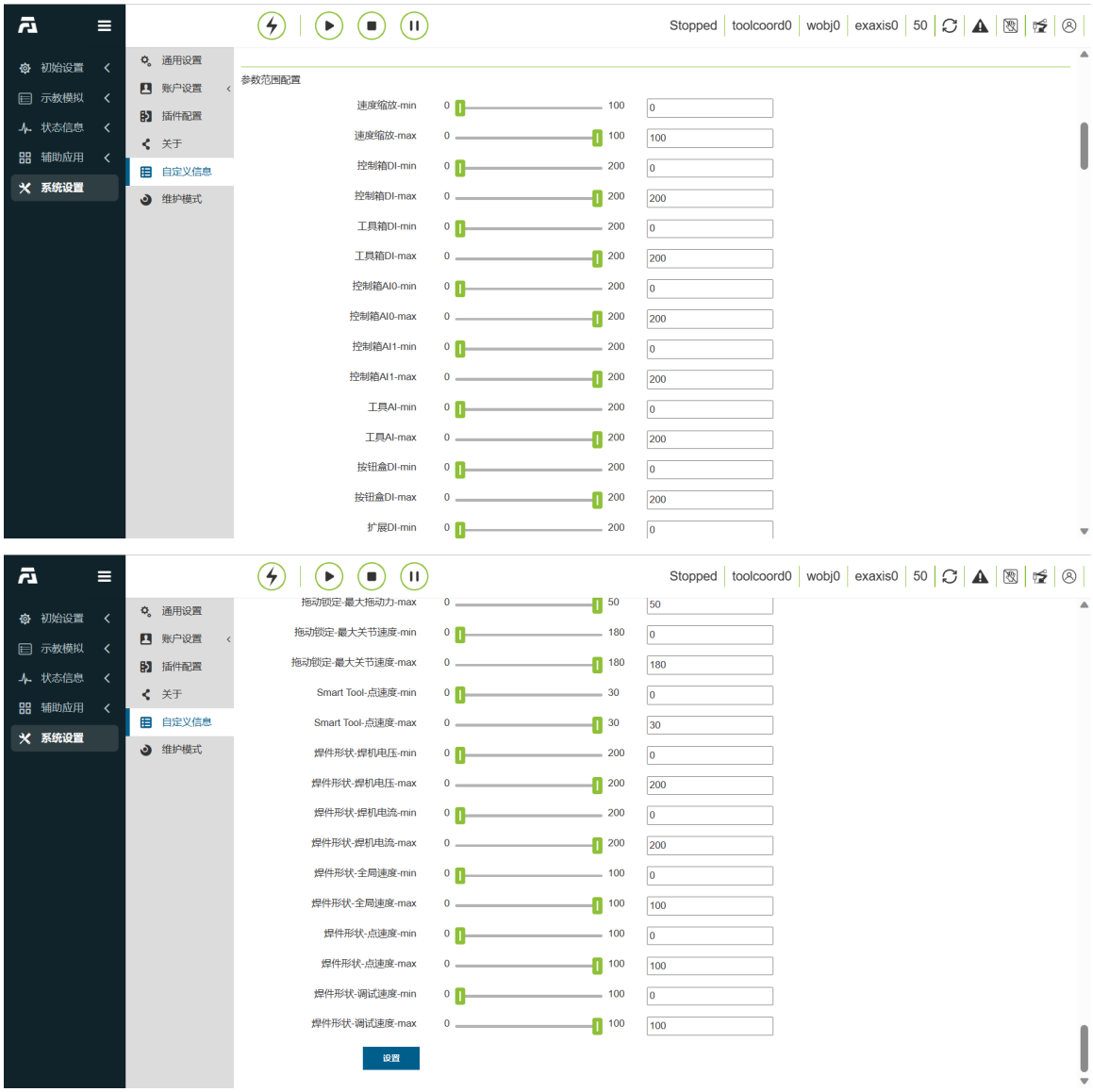
图表 3.10-13 自定义信息示意图

1.3.5.10.4.1 参数范围配置

参数范围配置，只有管理员可进行参数范围的调节，其他权限成员的参数只可在管理员设定的参数范围之内设定。

参数设定方式分为两种：滑块拖动和手动输入。

重要： 参数范围最大值必须大于最小值。参数范围配置成功 3 秒后，自动跳转到登录页，需重新登陆。



图表 3.10-14 参数范围配置示意图

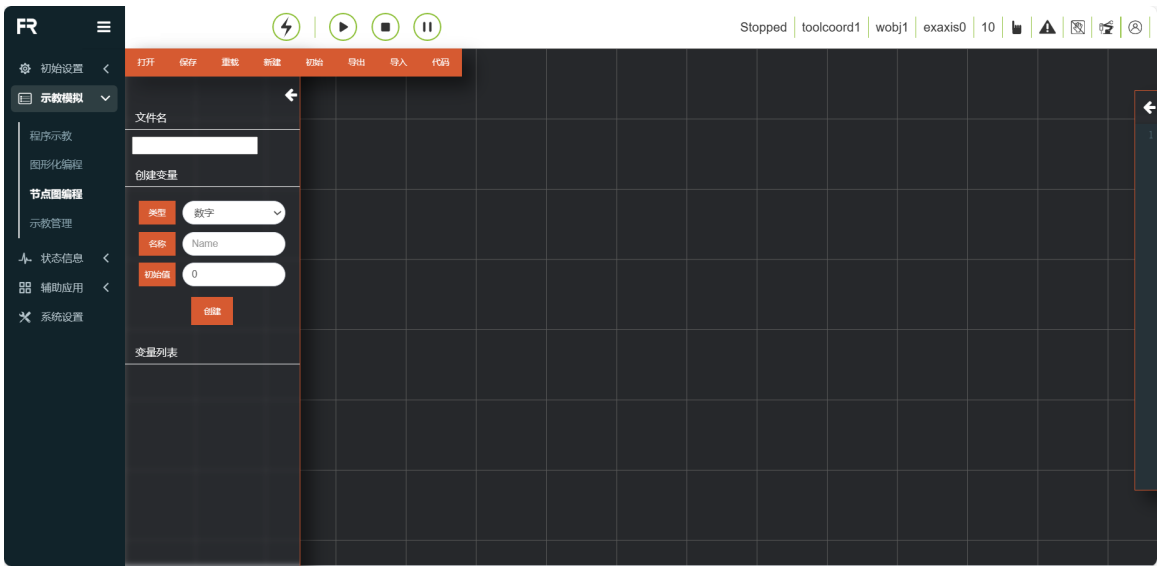
1.3.6 节点图编程

1.3.6.1 基础信息

1.3.6.1.1 系统简介

节点图编程是针对机器人开发的编程软件，其主要功能和技术特点如下：

- 节点之间的连线较好的呈现程序的上下文逻辑关系；
- 通过创建节点、连线节点和编辑节点参数等操作，只需拖动操作和少量的参数输入即可完成机器人程序编写；
- 有助于更好地可视化代码，并更快地编写复杂和重复任务的脚本；



图表 6.1-1 节点图编程界面

1.3.6.1.2 工具栏

使用节点图编程页面左侧顶部的工具栏。

图表 6.1-2 操作工具栏

备注： 名称：**打开**

作用：打开用户程序文件

备注： 名称：**保存**

作用：保存节点图编辑内容

备注： 名称：**重载**

作用：重新加载上次操作的节点图内容到本地。

备注： 名称：**新建**

作用：新建节点图编程文件

备注： 名称：**初始**

作用：加载初始设置的节点图内容。

备注： 名称：**导出**

作用：新建/打开节点图编程文件后，点击“导出”按钮，生成导出文件（json 格式）。

备注： 名称：**导入**

作用：点击“导入”按钮，弹出导入提示框。选择需要导入的文件，点击导入后，文件内容展示到节点图编程工作区。

备注： 名称：**代码**

作用：节点图连接后，生成 Lua 代码。

1.3.6.2 节点图操作

1.3.6.2.1 节点程序

节点程序需要在空白处点击鼠标右键，打开节点程序选择栏。程序指令主要分为逻辑指令、运动指令、力控指令、控制指令、Modbus 指令、扩展轴等指令。

节点程序选择栏上方输入框，可进行模糊搜索，快速定位所需节点指令。

具体节点程序操作流程如下：

- 点击“Begin”开始节点，创建开始节点编程位置；
- 点击选择的程序指令节点，对应节点图展示到工作区，可对其指令参数进行下拉框选择、输入操作；
- 指令节点右侧箭头作用：1. 单个箭头图标连接下一个节点 2. 多个箭头图标，第一个“Body”箭头图标连接内容节点，第二个“Completed”图标连接下一个节点；
- 将“Begin”开始节点与完成编写的节点程序相连, 则结束节点编程操作；

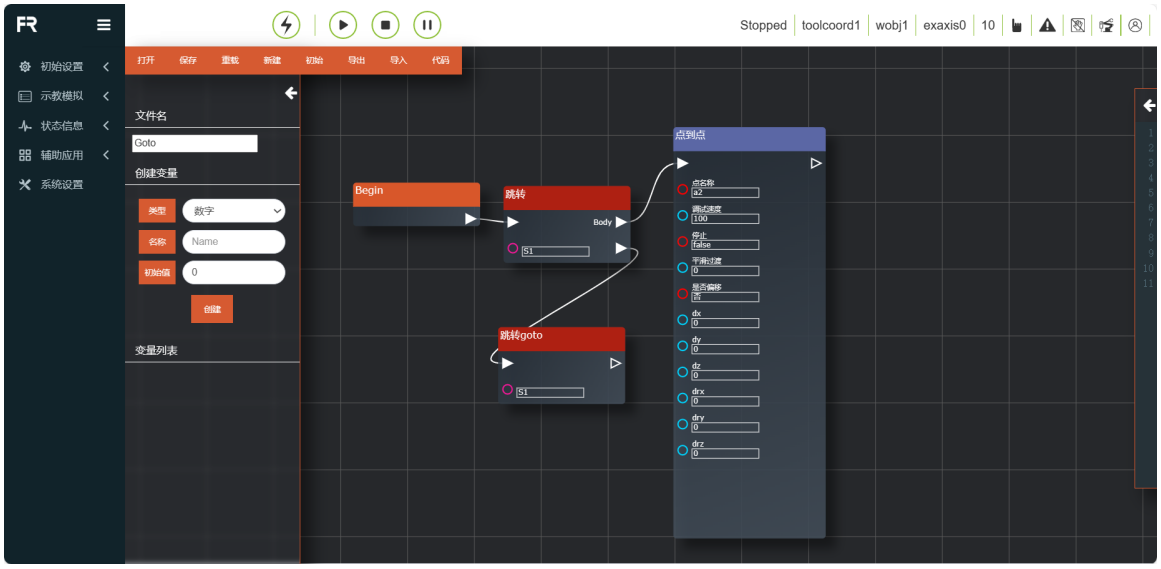
1.3.6.2.2 逻辑指令节点

1.3.6.2.2.1 跳转指令

点击“跳转”相关指令节点, 进入节点图编辑界面

“跳转”指令，第一个“Body”箭头图标连接主体内容节点，第二个箭头图标连接后续跳转位置 goto 指令节点。(该指令需要一定编程基础，如需帮助，请联系我们)

- 跳转名称: 输入跳转名称，来确定跳转位置



图表 6.2.1 “跳转”指令节点界面

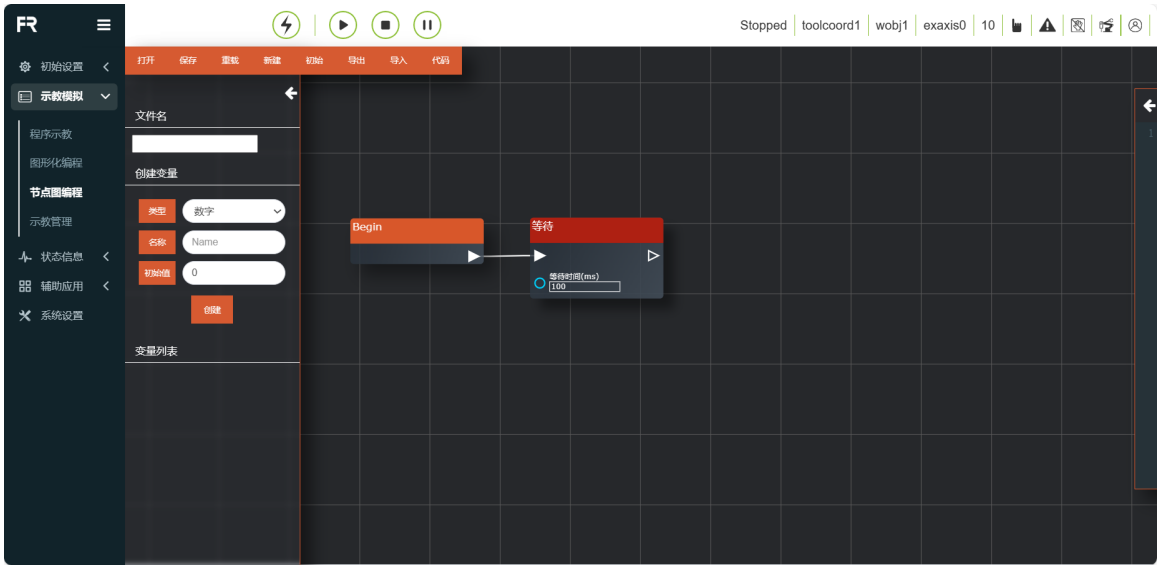
1.3.6.2.2.2 等待指令

点击“等待”相关指令节点, 进入节点图编辑界面

该指令为延时指令，分为“WaitMs”、“WaitDI”、“WaitMultiDI”和“WaitAI”四部分。

1. “等待”指令节点, 参数:

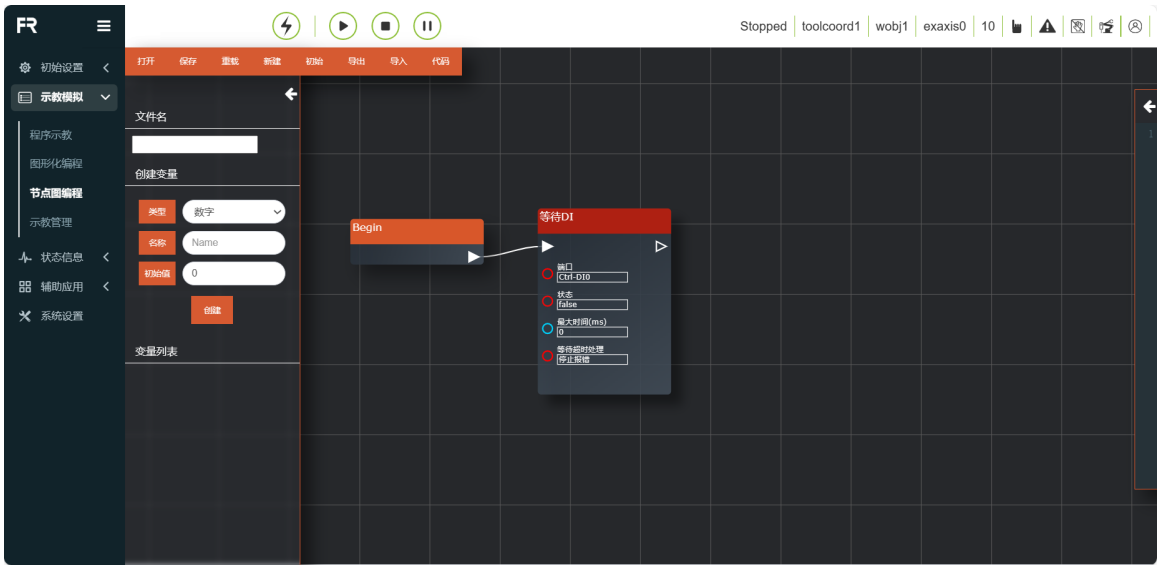
- 等待时间 (ms): 延时等待时间单位为毫秒，输入需要等待的毫秒数



图表 6.2.2 “等待”指令节点界面

2. “等待 DI”指令节点，参数:

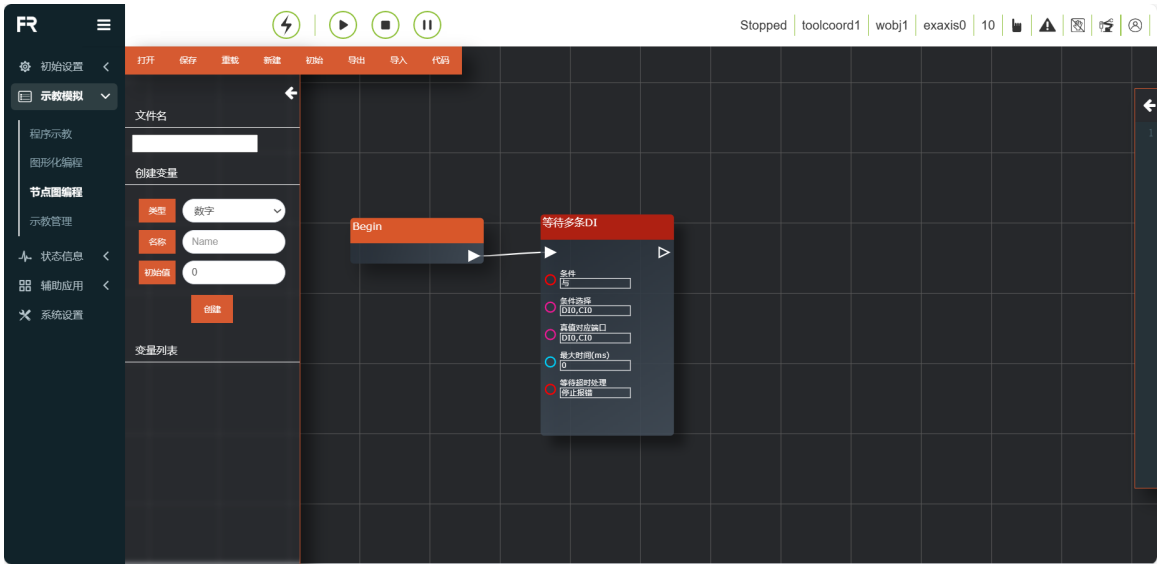
- DI 端口号：Ctrl-DI0 ~ Ctrl-CI7(WaitDI,[0~15]), End-DI0 ~ End-DI1(WaitToolDI,[0~1])
- 状态：false/true
- 最大时间 (ms)：0 ~ 10000
- 等待超时处理：停止报错/继续执行/一直等待



图表 6.2.3 “等待 DI”指令节点界面

3. “等待多条 DI”指令节点，参数:

- 条件：与/或
- 条件选择：选择位的状态开启的端口号，以逗号隔开，例 DI0,DI1
- 真值对应端口：选择真值的端口号，以逗号隔开，例 DI0,DI1
- 最大时间 (ms)：0 ~ 10000, 最大等待时间
- 等待超时处理：停止报错/继续执行/一直等待

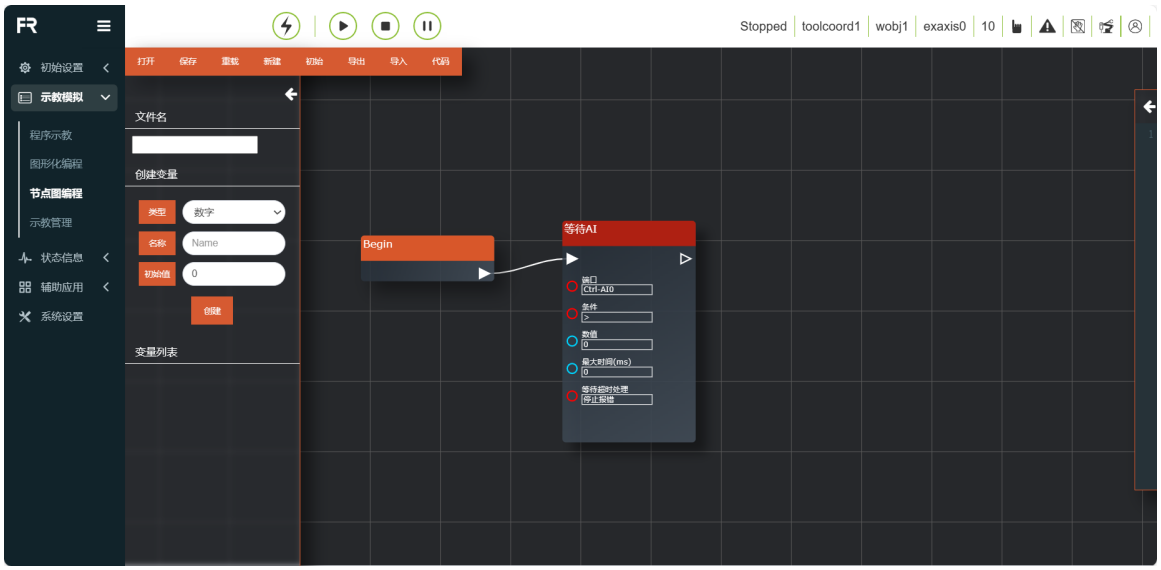


图表 6.2.4 “等待多条 DI”指令节点界面

4. “等待 AI”指令节点，参数:

- 条件：与/或

- AI 端口号：Ctrl-AI0 ~ Ctrl-AI1(WaitAI,[0~1]), End-AI0(WaitToolAI,[0])
- 条件：大于/小于
- 数值 (%)：1 ~ 100
- 最大时间 (ms)：0 ~ 10000
- 等待超时处理：停止报错/继续执行/一直等待, 等待超时处理一直等待时，最大时间默认为 0



图表 6.2.5 “等待 AI”指令节点界面

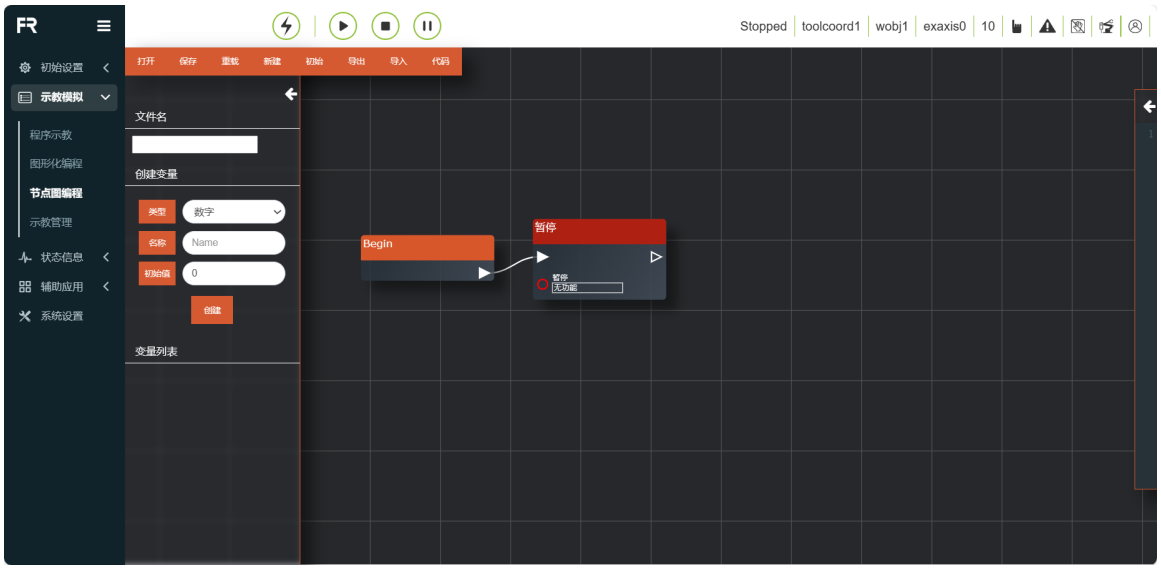
1.3.6.2.2.3 暂停指令

点击“暂停”指令节点, 进入节点图编辑界面

该指令为暂停指令，在程序中插入该指令，当程序执行到该指令时，机器人会处于暂停状态，若想继续运行，点击控制区“暂停/恢复”按键即可。

“暂停”指令节点, 参数:

- 暂停类型：无功能、气缸未到位等



图表 6.2.6 “暂停”指令节点界面

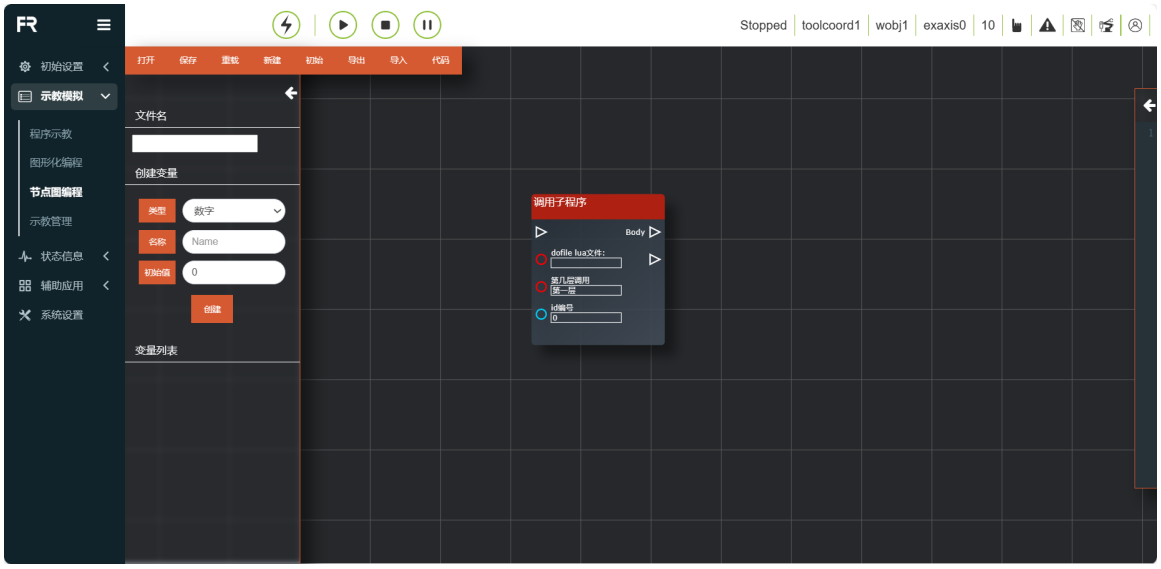
1.3.6.2.2.4 调用子程序指令

点击“调用子程序”指令节点, 进入节点图编辑界面

该指令为暂停指令，在程序中插入该指令，当程序执行到该指令时，机器人会处于暂停状态，若想继续运行，点击控制区“暂停/恢复”按键即可。

“调用子程序”指令节点, 参数:

- dofile 文件：创建生成的文件名
- 第几层调用：第一层/第二层
- id 编号：所属层级对应位置 id



图表 6.2.7 “调用子程序”指令节点界面

1.3.6.2.3 运动指令节点

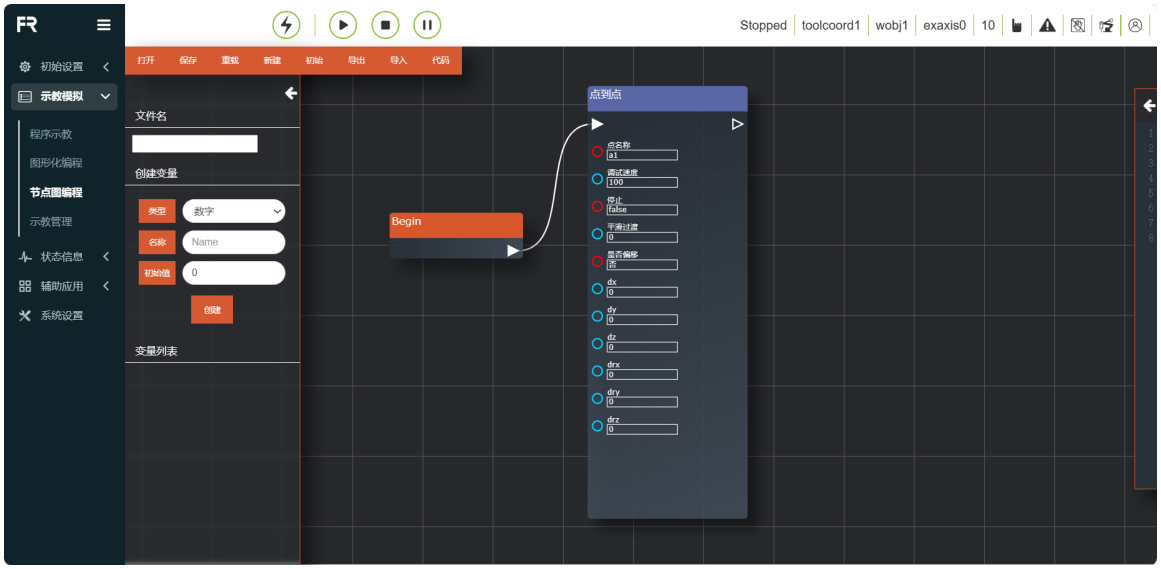
1.3.6.2.3.1 点到点指令

点击“点到点”指令节点, 进入节点图编辑界面

可以选择需要到达的点，平滑过渡时间设置可以实现该点到下一点的运动是连续的，是否偏移设置，可以选择基于基坐标系偏移和基于工具坐标偏移，并弹出 x,y,z,rx,ry,rz 偏移量设置，PTP 具体路径为运动控制器自动规划的最优路径

“点到点”指令节点, 参数:

- 点名称：示教点位
- 调试速度 (%)：0 ~ 100
- 停止：false/true
- 平滑过渡 (ms)：平滑过渡时间 0 ~ 500
- 是否偏移否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量



图表 6.2.8 “点到点”指令节点界面

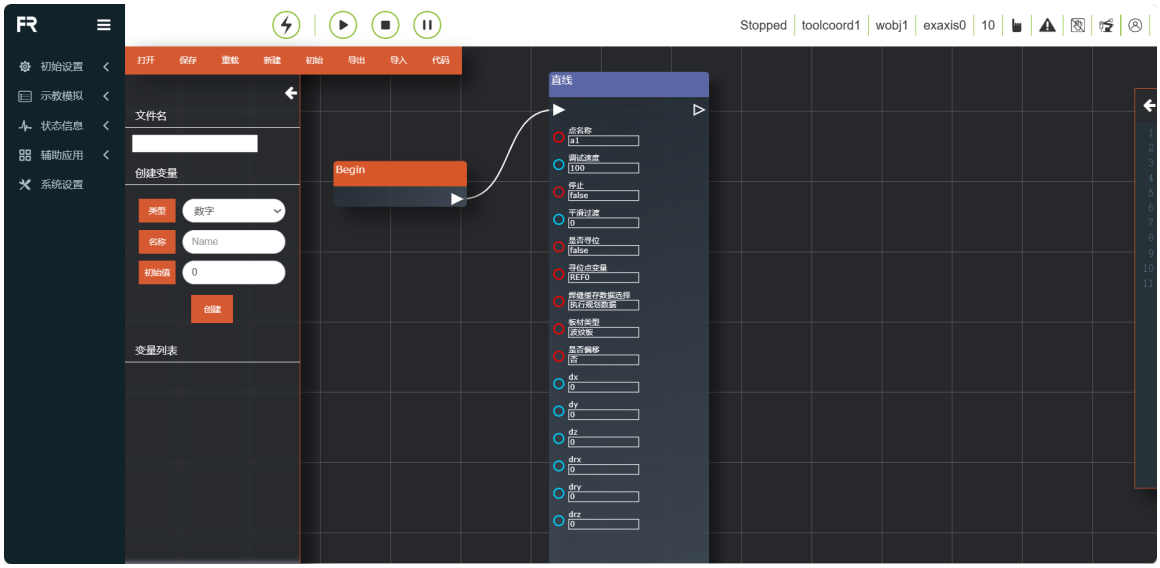
1.3.6.2.3.2 直线指令

点击“直线”指令节点, 进入节点图编辑界面

该指令功能与“点到点”指令相似，但该指令所到达点的路径为直线

“直线”指令节点, 参数:

- 点名称：示教点位
- 调试速度 (%)：0 ~ 100
- 停止：false/true，选择 true 时，平滑过渡参数值不生效
- 平滑过渡 (mm)：平滑过渡半径 0 ~ 1000
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- 是否寻位：false/true
- 寻位点变量：REF0~99/RES0~99，是否寻位选择 false 时，参数不生效；
- 焊缝缓存数据选择：执行规划数据/执行记录数据，选择点名称为“seamPos”时，参数生效，否则不生效；
- 板材类型：波纹板/瓦楞板/围栏板/油桶/波纹甲壳钢，选择点名称为“seamPos”时，参数生效，否则不生效；



图表 6.2.9 “直线”指令节点界面

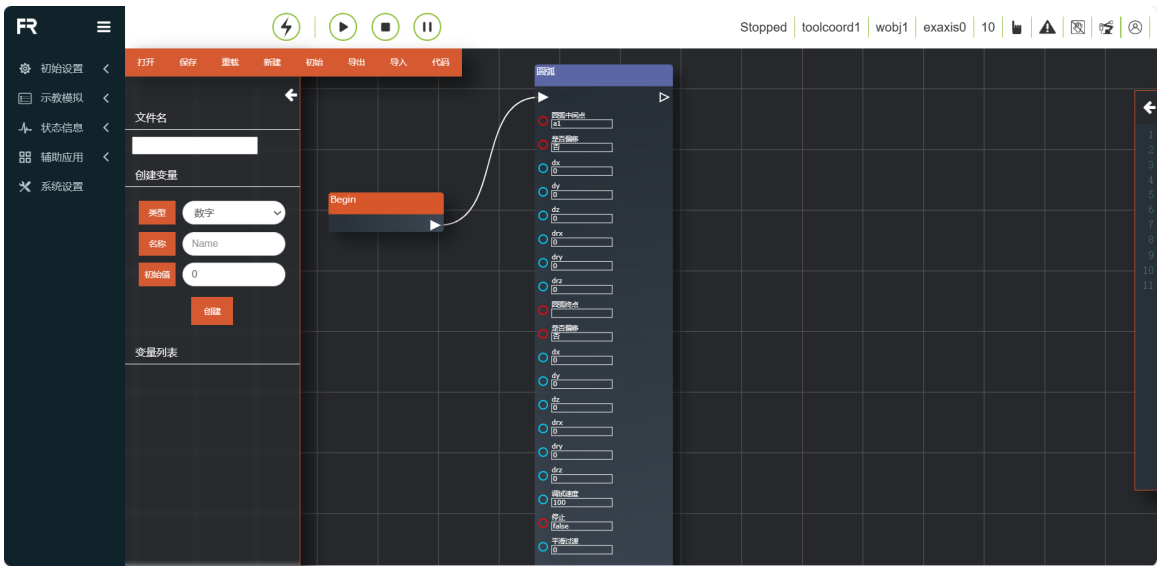
1.3.6.2.3.3 圆弧指令

点击“圆弧”指令节点, 进入节点图编辑界面

圆弧运动包含两个点，第一点为圆弧中间过渡点，第二点为终点，过渡点和终点都可以对是否偏移进行设置，可以选择基于基坐标系偏移和基于工具坐标偏移，设置 x,y,z,rx,ry,rz 偏移量，终点可以设置平滑过渡半径，实现运动连续效果

“圆弧”指令节点, 参数:

- 圆弧中间点：示教点位
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量
- 圆弧终点：示教点位
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量
- 调试速度 (%)：0 ~ 100
- 停止：false/true，选择 true 时，平滑过渡参数值不生效
- 平滑过渡 (mm)：平滑过渡半径 0 ~ 1000



图表 6.2.10 “圆弧”指令节点界面

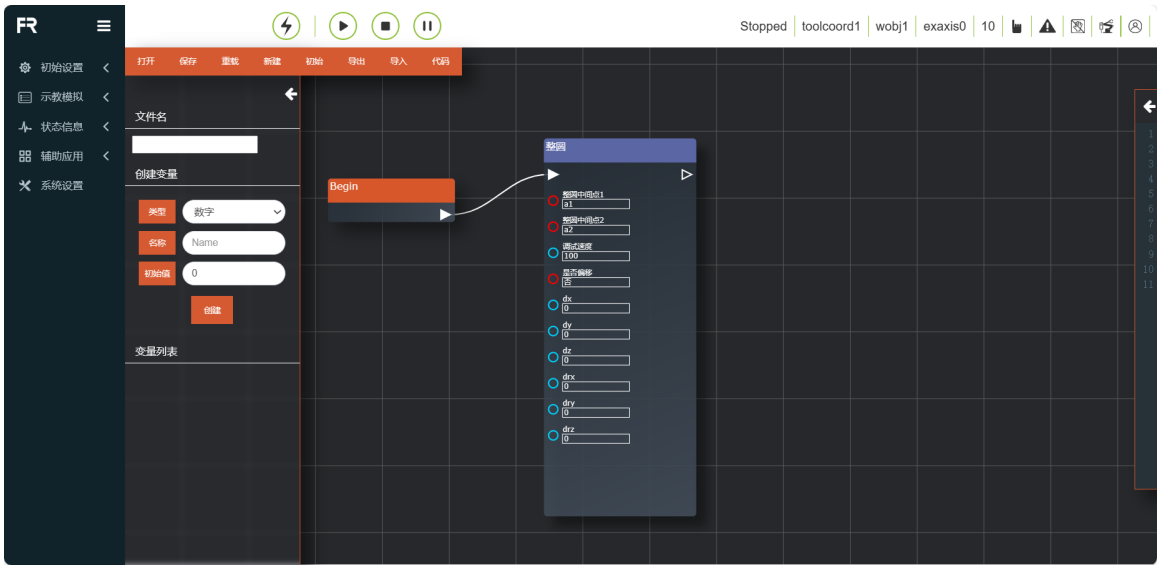
1.3.6.2.3.4 整圆指令

点击“整圆”指令节点, 进入节点图编辑界面

整圆运动包含两个点，第一点为整圆中间过渡点 1，第二点为整圆中间过渡点 2，过渡点 2 可以设置是否偏移，该偏移量同时生效于过渡点 1 和过渡点 2

“整圆”指令节点, 参数:

- 整圆中间点 1：示教点位
- 整圆中间点 2：示教点位
- 调试速度 (%)：0 ~ 100
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量



图表 6.2.11 “整圆”指令节点界面

1.3.6.2.3.5 螺旋指令

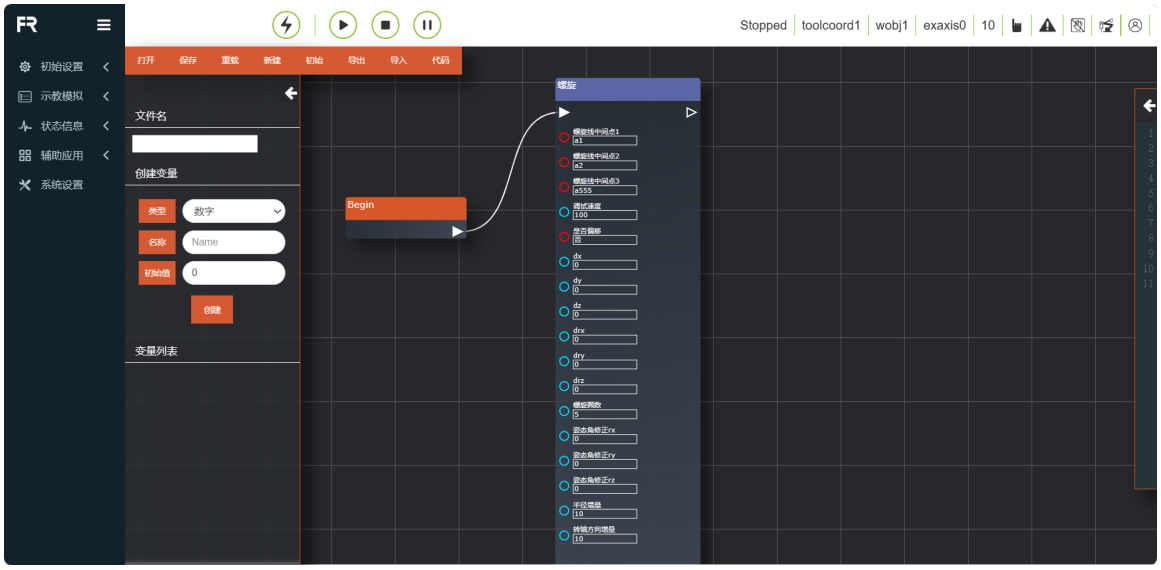
点击“螺旋”指令节点, 进入节点图编辑界面

螺旋线运动包含三个点，该三个点组成一个圆，在第三点设置页面，包含螺旋圈数，姿态修正角，半径增量和转轴方向增量这几个参数设置，螺旋圈数即该螺旋线的运动圈数，姿态修正角修正的是螺旋线结束时的姿态与螺旋线第一点的姿态，半径增量即每一圈半径的增量，转轴方向增量即螺旋轴方向的增量。设置是否偏移，该偏移量生效于整个螺旋线的轨迹。

“螺旋”指令节点, 参数:

- 螺旋线中间点 1：示教点位

- 螺旋线中间点 2：示教点位
- 螺旋线中间点 3：示教点位
- 调试速度 (%)：0 ~ 100
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量
- 螺旋圈数：0 ~ 100
- 姿态角修正 rx(°)：-1000 ~ 1000
- 姿态角修正 ry(°)：-1000 ~ 1000
- 姿态角修正 rz(°)：-1000 ~ 1000
- 半径增量 (mm)：-100 ~ 100
- 转轴方向增量 (mm)：-100 ~ 100



图表 6.2.12 “螺旋”指令节点界面

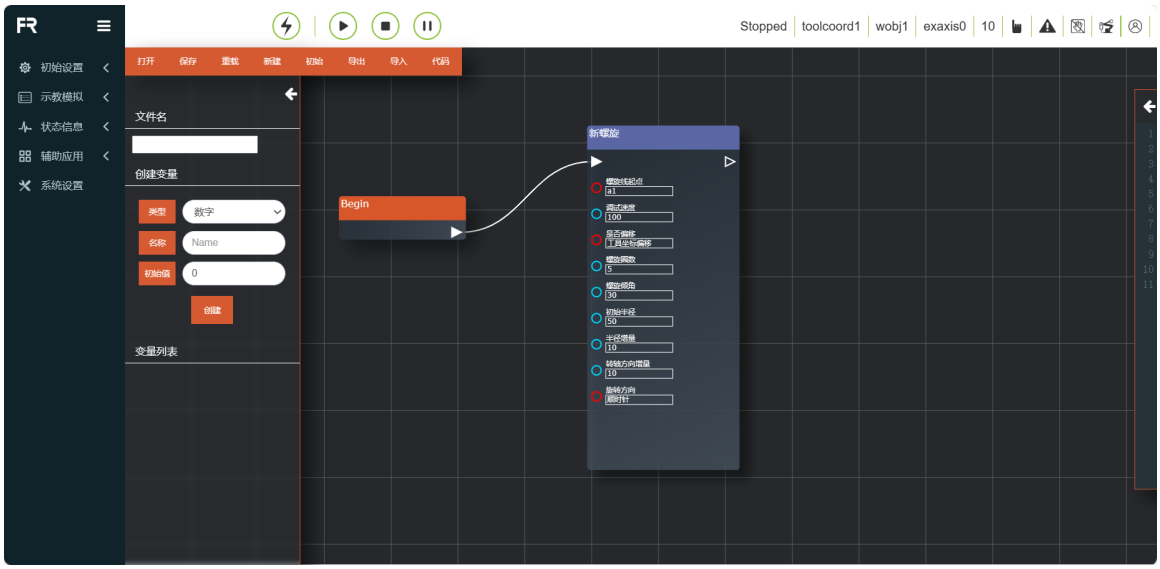
1.3.6.2.3.6 新螺旋指令

点击“新螺旋”指令节点, 进入节点图编辑界面

新螺旋运动为优化版螺旋线运动，该指令只需要一个点加各参数的配置实现螺旋线运动。机器人以当前位置作为起点，用户设置调试速度，是否偏移，螺旋圈数，螺旋倾角，初始半径，半径增量，转轴方向增量和旋转方向这几个参数，螺旋圈数即该螺旋线的运动圈数，螺旋倾角即工具 Z 轴与水平方向的夹角，姿态修正角修正的是螺旋线结束时的姿态与螺旋线第一点的姿态，初始半径即第一圈半径大小，半径增量即每一圈半径的增量，转轴方向增量即螺旋轴方向的增量，旋转方向即顺时针和逆时针。

“新螺旋”指令节点, 参数:

- 螺旋线起点：示教点位
- 调试速度 (%)：0 ~ 100
- 是否偏移：否/基坐标偏移/工具坐标偏移选择否时，dx~drz 参数值不生效
- dx~drz：偏移量
- 螺旋圈数：0 ~ 100
- 螺旋倾角 (°)：-100 ~ 100
- 初始半径：0 ~ 100
- 半径增量 (mm)：-100 ~ 100
- 转轴方向增量 (mm)：-100 ~ 100
- 旋转方向：顺时针/逆时针



图表 6.2.13 “新螺旋”指令节点界面

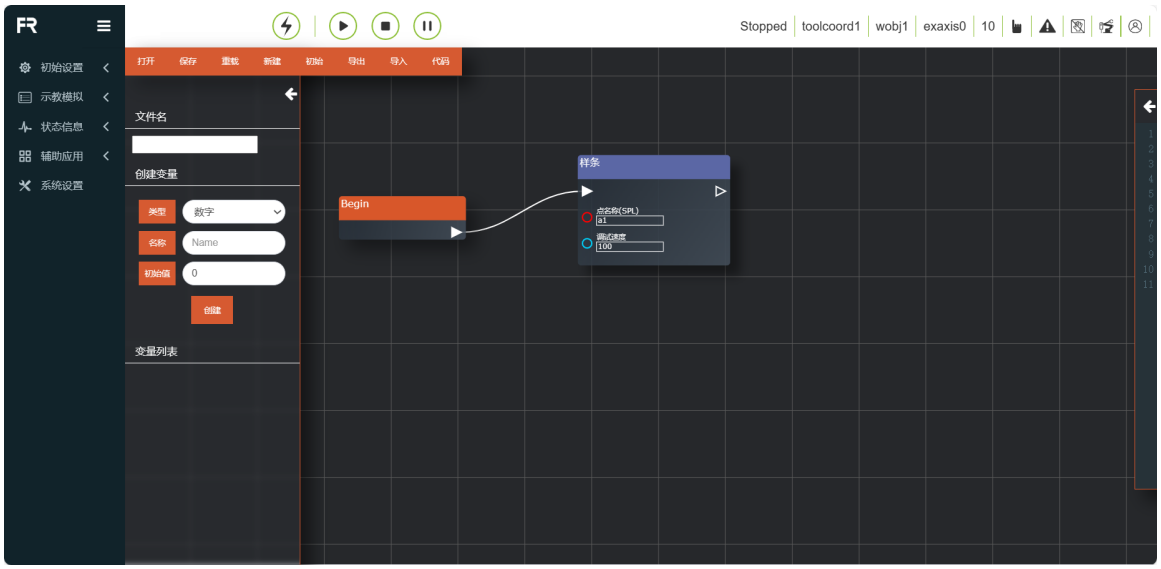
1.3.6.2.3.7 样条指令

点击“样条”指令节点, 进入节点图编辑界面

该指令分为样条组起始，样条段和样条组结束三部分，样条组开始是样条运动的起始标志，样条段目前节点图只包含 SPL 段，样条组结束是样条运动的结束标志。

“样条”指令节点, 参数:

- 点名称：示教点位
- 调试速度 (%)：0 ~ 100



图表 6.2.14 “样条”指令节点界面

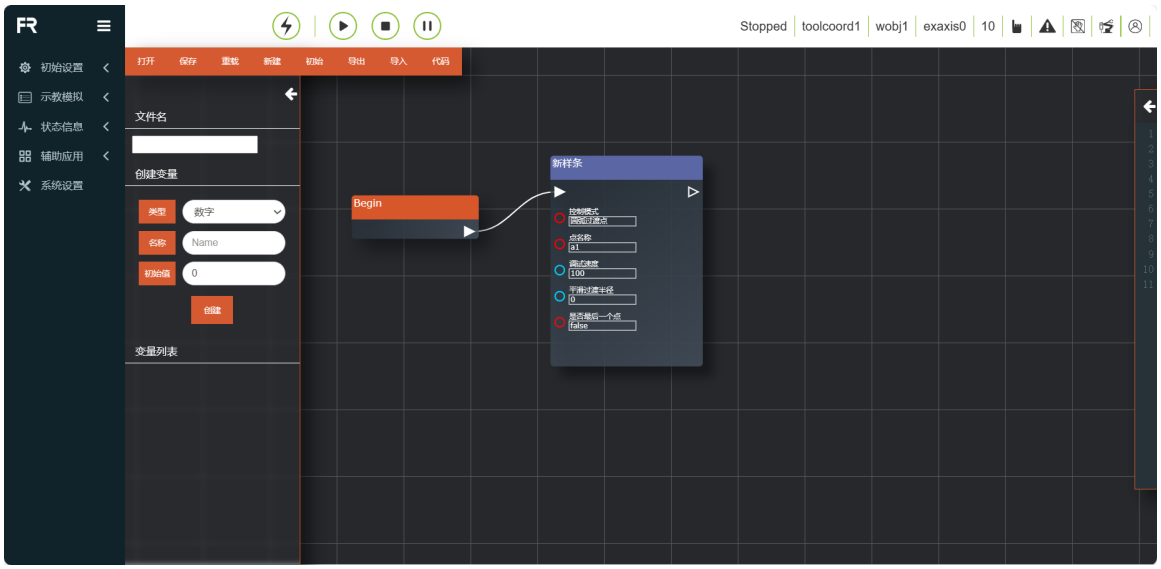
1.3.6.2.3.8 新样条指令

点击“新样条”指令节点, 进入节点图编辑界面

该指令为样条指令算法优化指令，后续会替代现有的样条指令，该指令分为多点轨迹起始，多点轨迹段和多点轨迹结束三部分，多点轨迹开始是多点轨迹运动的起始标志，多点轨迹段即设置各个轨迹点，点击图标进入点位添加界面，多点轨迹结束是多点轨迹运动的结束标志，在此可以设置控制模式和调试速度，控制模式分为给定控制点和给定路径点。

“新样条”指令节点, 参数:

- 控制模式：示教点位
- 点名称：示教点位
- 调试速度 (%)：0 ~ 100
- 平滑过渡半径：0 ~ 1000
- 是否最后一个点：否/是



图表 6.2.15 “新样条”指令节点界面

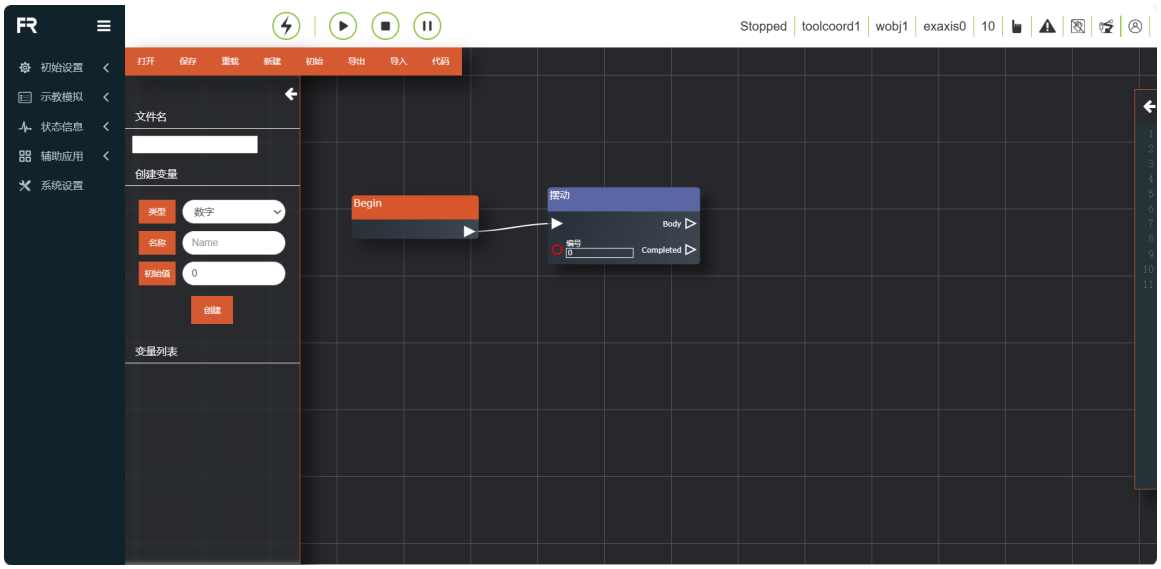
1.3.6.2.3.9 摆动指令

点击“摆动”指令节点, 进入节点图编辑界面

该指令包含两部分，第一部分选择配置好参数的摆焊编号，连接 Body 代表连接节点的程序在“开始摆焊”和“停止摆焊”中间执行。

“摆动”指令节点, 参数:

- 编号：0~7



图表 6.2.16 “摆动”指令节点界面

1.3.6.2.3.10 轨迹复现指令

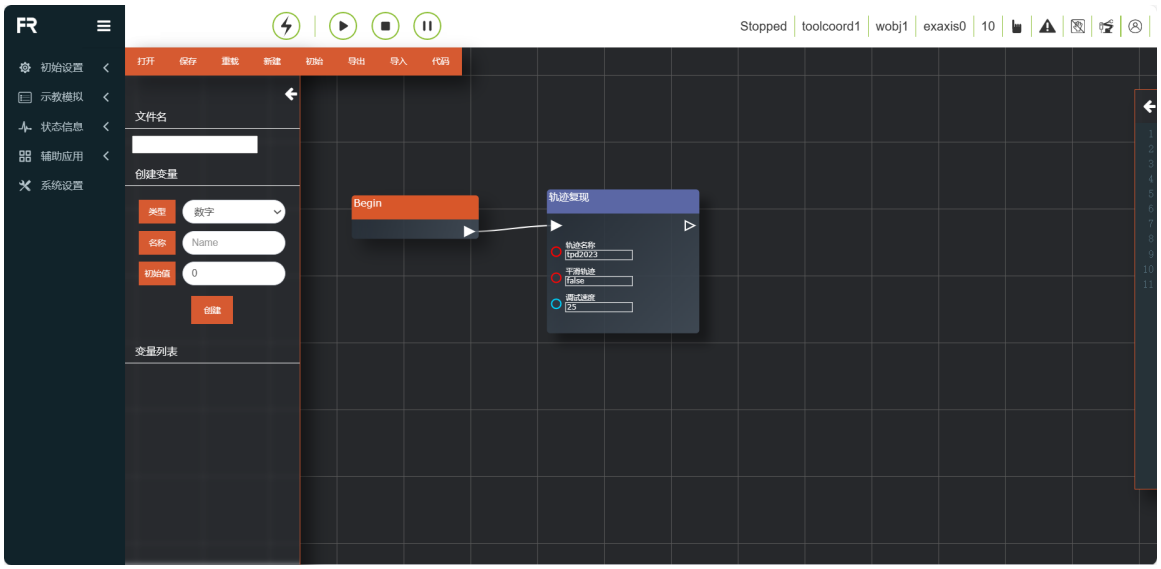
点击“轨迹复现”指令节点, 进入节点图编辑界面

在该指令中，用户首先需要有记录好的轨迹。

进行程序编程时，首先用点到点指令到达对应轨迹起始点，然后在轨迹复现指令中选择轨迹，选择平滑轨迹，设置调试速度。轨迹加载指令主要用于预先读取轨迹文件，提取成轨迹指令，更好的应用于传送带跟踪场景。

“轨迹复现”指令节点, 参数:

- 轨迹名称：记录好的轨迹
- 平滑轨迹：否/是
- 调试速度 (%)：0 ~ 100，默认值为 25



图表 6.2.17 “轨迹复现”指令节点界面

1.3.6.2.3.11 点偏移指令

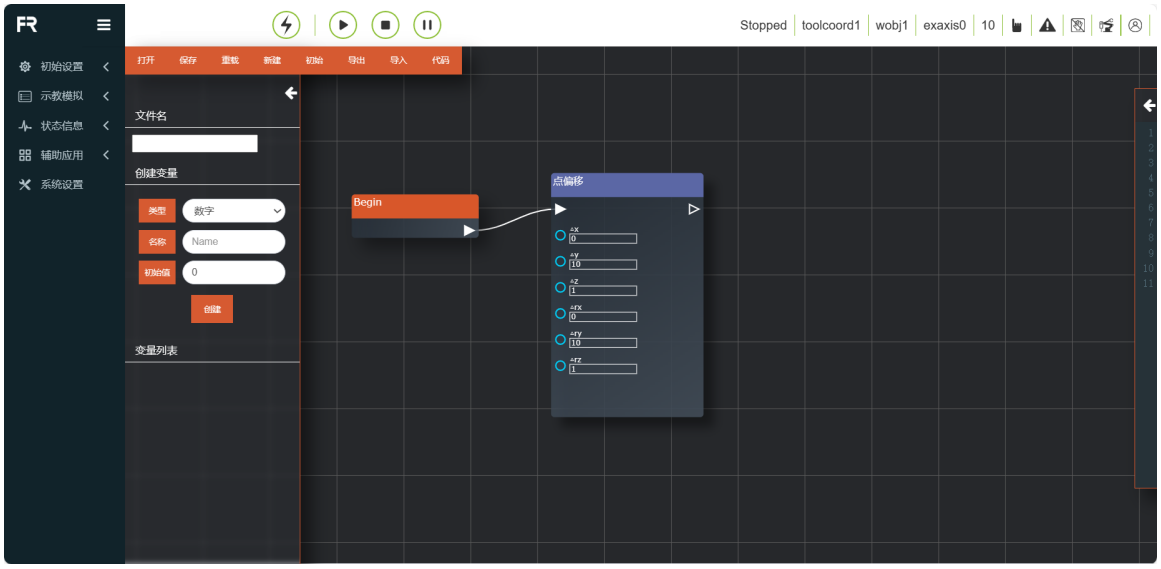
点击“点偏移”指令节点, 进入节点图编辑界面

该指令为整体偏移指令，输入各个偏移量，连接 Body 代表连接节点的程序在开始和关闭之间执行，中间的运动指令会基于基坐标（或工件坐标）进行偏移。

“点偏移”指令节点, 参数:

- Δx：偏移量，-300~300
- Δy：偏移量，-300~300
- Δz：偏移量，-300~300
- Δrx：偏移量，-300~300

- Δry：偏移量，-300~300
- Δrz：偏移量，-300~300



图表 6.2.18 “点偏移”指令节点界面

1.3.6.2.3.12 伺服指令

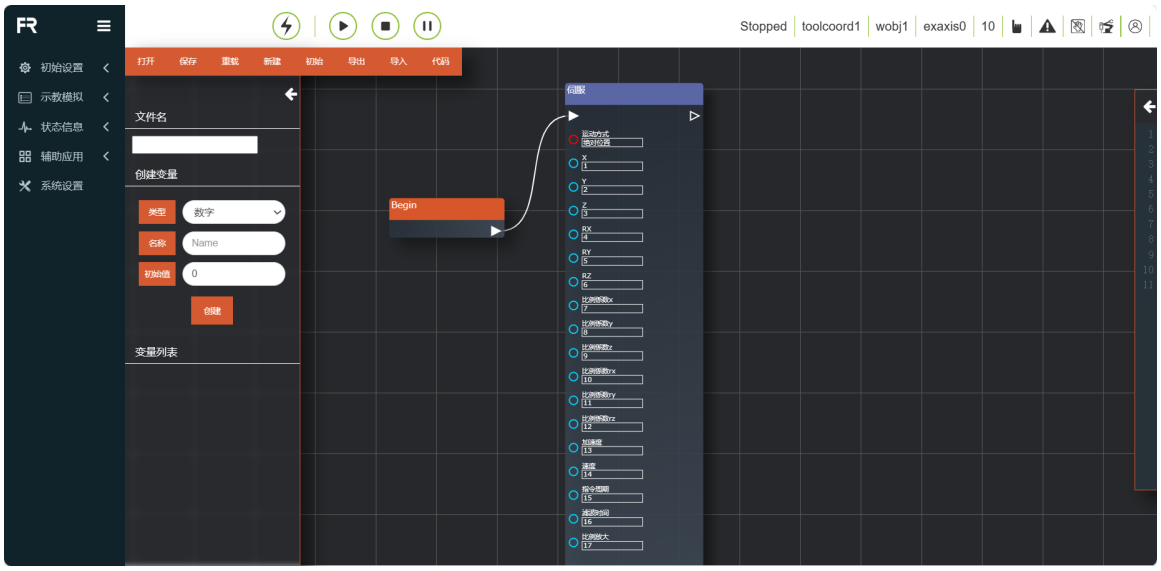
点击“伺服”指令节点, 进入节点图编辑界面

伺服控制（笛卡尔空间运动）指令，该指令可以通过绝对位姿控制或基于当前位姿偏移来控制机器人运动。

“伺服”指令节点, 参数:

- 运动方式：绝对位置/基坐标偏移/工具坐标偏移
- x：偏移量，-300~300
- y：偏移量，-300~300
- z：偏移量，-300~300
- rx：偏移量，-300~300
- ry：偏移量，-300~300
- rz：偏移量，-300~300
- 比例系数 x：0~1
- 比例系数 y：0~1
- 比例系数 z：0~1
- 比例系数 rx：0~1
- 比例系数 ry：0~1

- 比例系数 rz：0~1
- 加速度 (%)：0~100
- 速度 (%)：0~100
- 指令周期 (s)：0.001~0.016
- 滤波时间 (s)：0~1
- 比例放大：0~100



图表 6.2.19 “伺服”指令节点界面

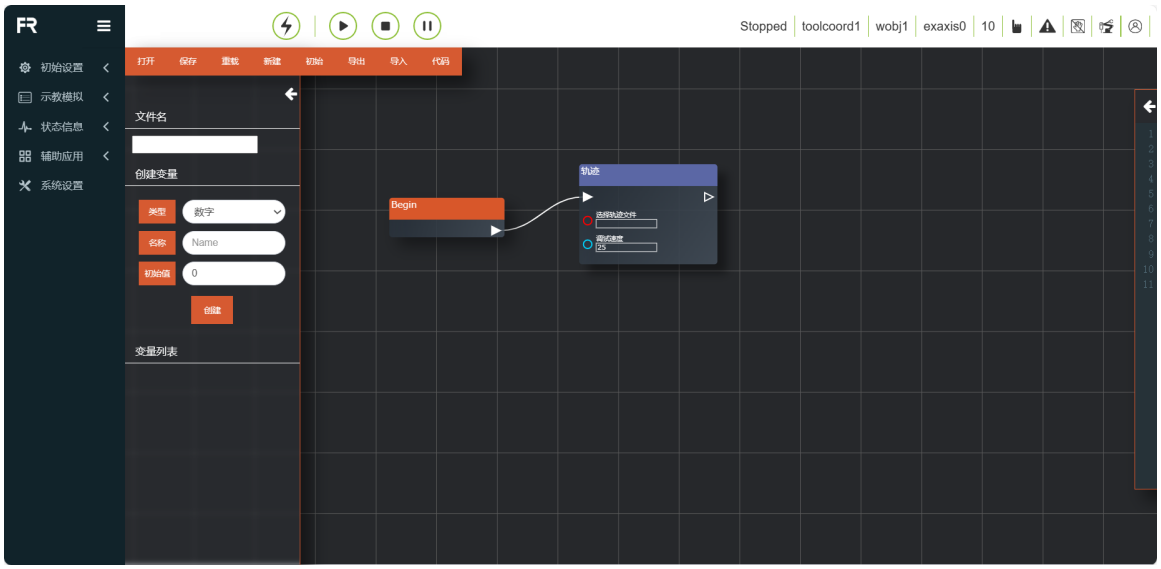
1.3.6.2.3.13 轨迹指令

点击“轨迹”指令节点, 进入节点图编辑界面

在该指令中，用户首先需要有记录好的轨迹。

“轨迹”指令节点, 参数:

- 选择轨迹文件：记录好的轨迹
- 调试速度 (%)：0 ~ 100，默认值为 25



图表 6.2.20 “轨迹”指令节点界面

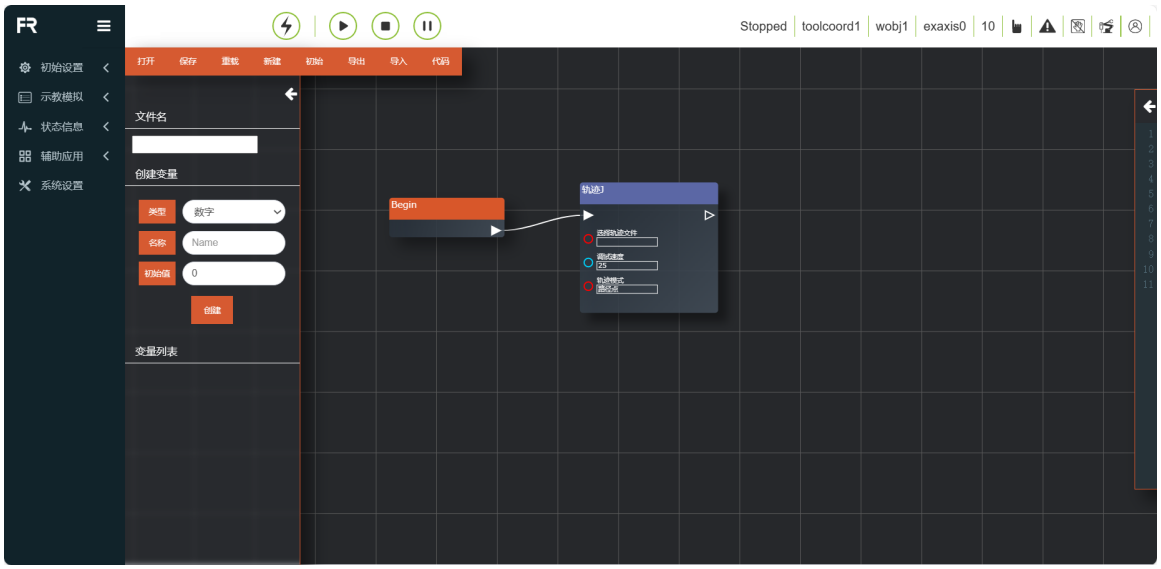
1.3.6.2.3.14 轨迹 J 指令

点击“轨迹 J”指令节点, 进入节点图编辑界面

在该指令中，用户首先需要有记录好的轨迹，可以在示教程序界面预先导入轨迹文件。轨迹指令和轨迹 J 指令适用于相机直接给定轨迹的通用接口，满足在已有固定格式的离散的轨迹点文件时，可导入系统使得机器人按照导入文件的轨迹进行运动。

“轨迹 J”指令节点, 参数:

- 选择轨迹文件：记录好的轨迹
- 调试速度 (%)：0 ~ 100，默认值为 25
- 轨迹模式：路径点/控制点



图表 6.2.21 “轨迹 J”指令节点界面

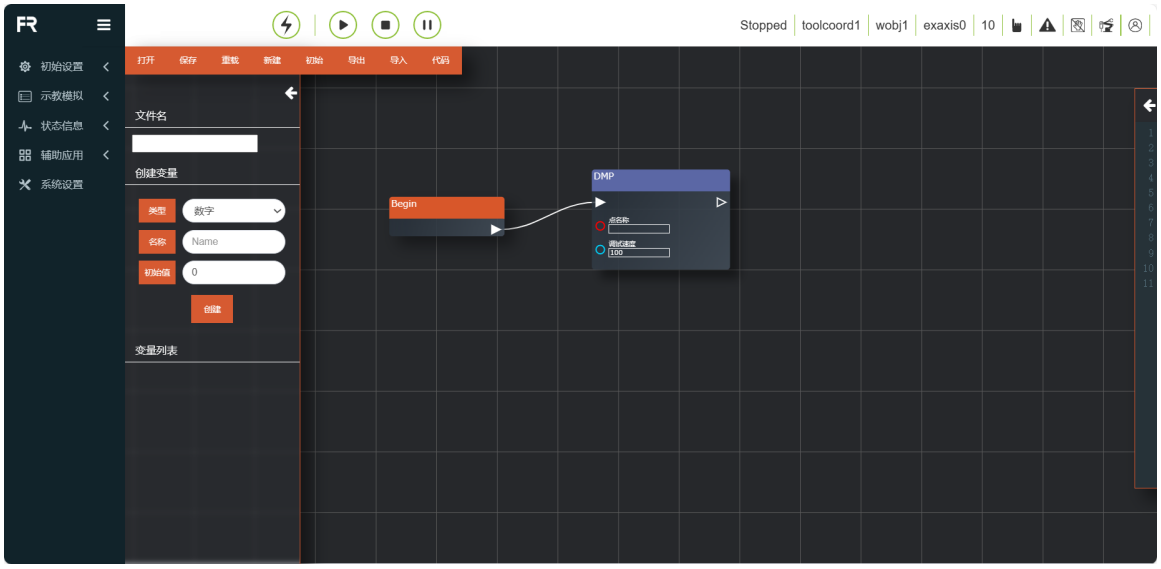
1.3.6.2.3.15 DMP 指令

点击“DMP”指令节点, 进入节点图编辑界面

DMP 是一种轨迹模仿学习的方法，需要事先规划参考轨迹。在命令编辑界面，选择示教点作为新的起点，点击“添加”、“应用”后可保存该指令。DMP 具体路径为以新的起点模仿参考轨迹的新轨迹。

“DMP”指令节点, 参数:

- 点名称：示教点
- 调试速度 (%)：0 ~ 100，默认值为 100



图表 6.2.22 “DMP”指令节点界面

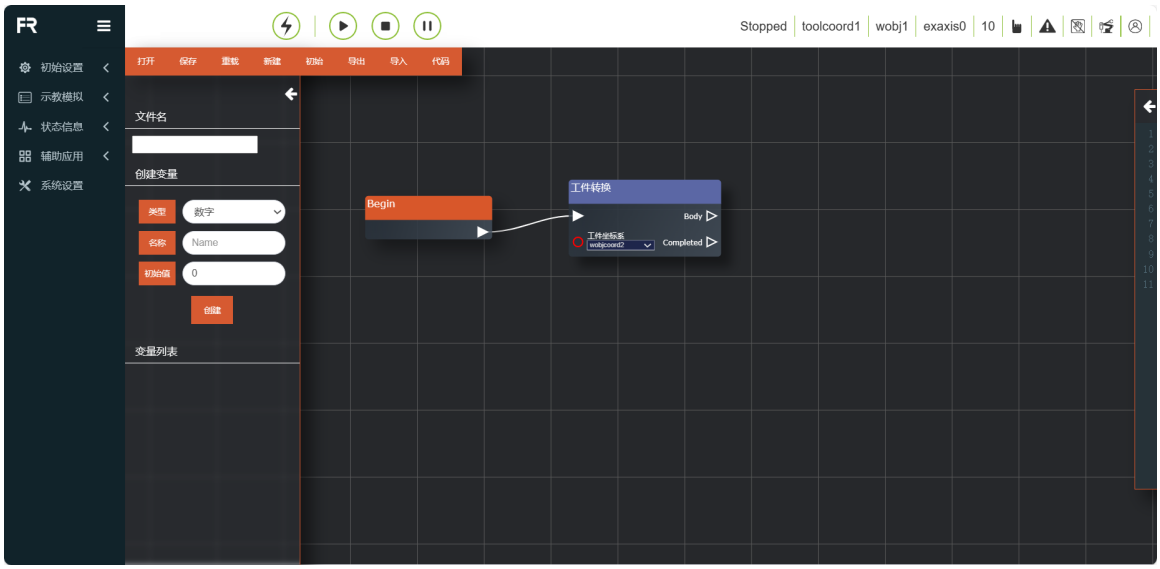
1.3.6.2.3.16 工件转换指令

点击“工件转换”指令节点, 进入节点图编辑界面

选择所要进行自动转换的工件坐标系，点击“添加”、“应用”后可保存该指令，添加 PTP、LIN 指令时与 Body 相连可实现在该指令内部执行，工件坐标系下点位自动转换。

“工件转换”指令节点, 参数:

- 工件坐标系：工件坐标系列表



图表 6.2.23 “工件转换”指令节点界面

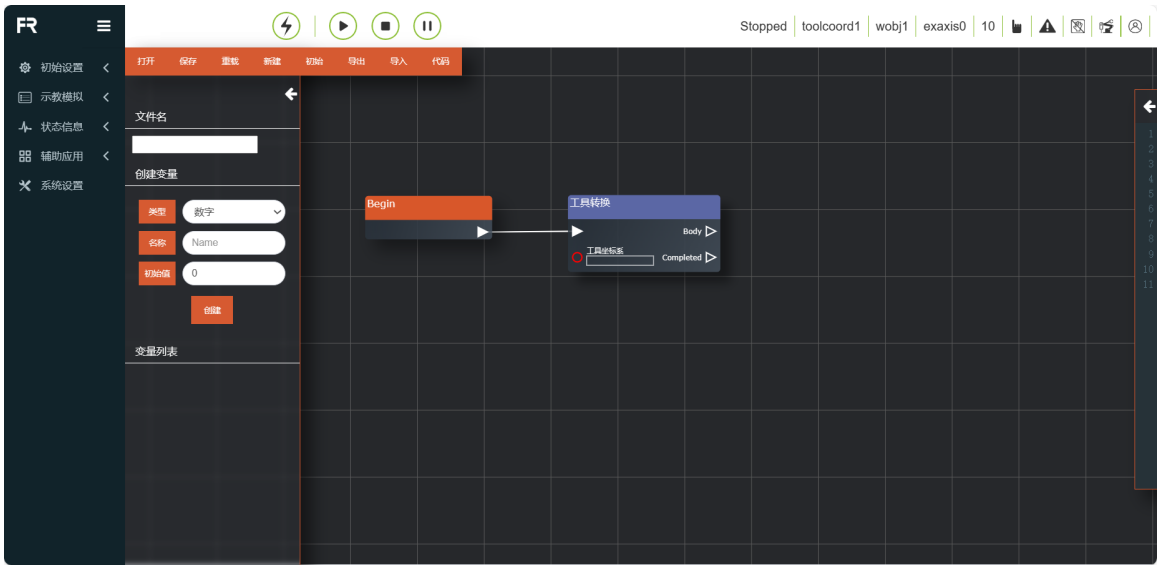
1.3.6.2.3.17 工具转换指令

点击“工具转换”指令节点, 进入节点图编辑界面

选择所要进行自动转换的工具坐标系，点击“添加”、“应用”后可保存该指令，添加 PTP、LIN 指令时与 Body 相连可实现在该指令内部执行，工具坐标系下点位自动转换。

“工具转换”指令节点, 参数:

- 工具坐标系：工具坐标系列表



图表 6.2.24 “工具转换”指令节点界面

1.3.6.2.4 控制指令界面

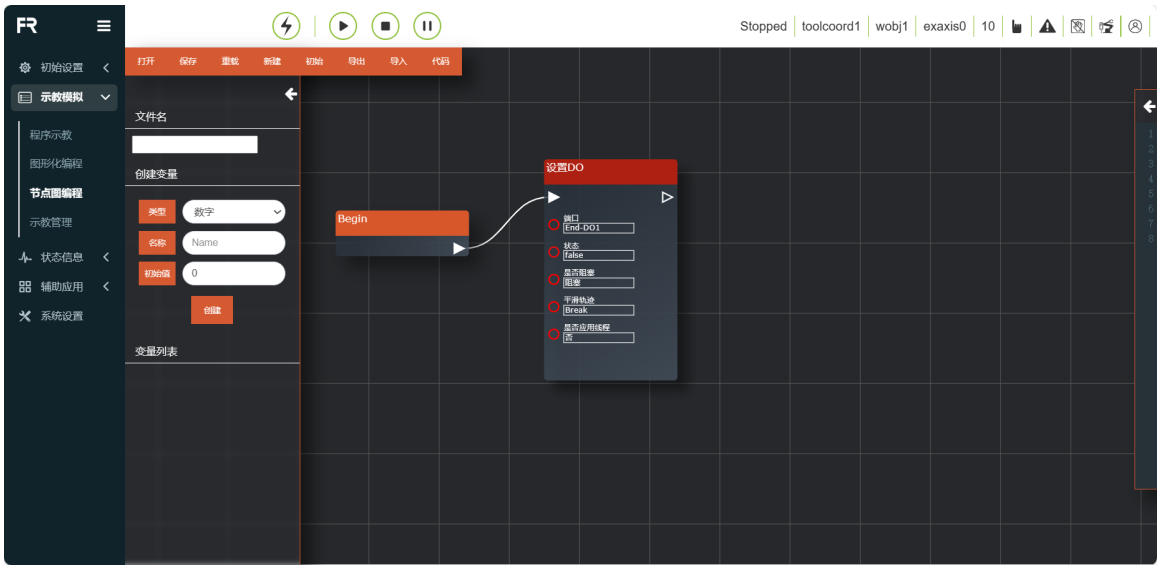
1.3.6.2.4.1 数字 IO 指令节点

点击“设置 DO”/“获取 DI”指令节点, 进入节点图编辑界面

该指令为 IO 指令，分为设置 IO（SetDO/SPLCSetDO）和获取 IO（GetDI/SPLCGetDI）两部分。

1. “设置 DO”指令节点, 参数:

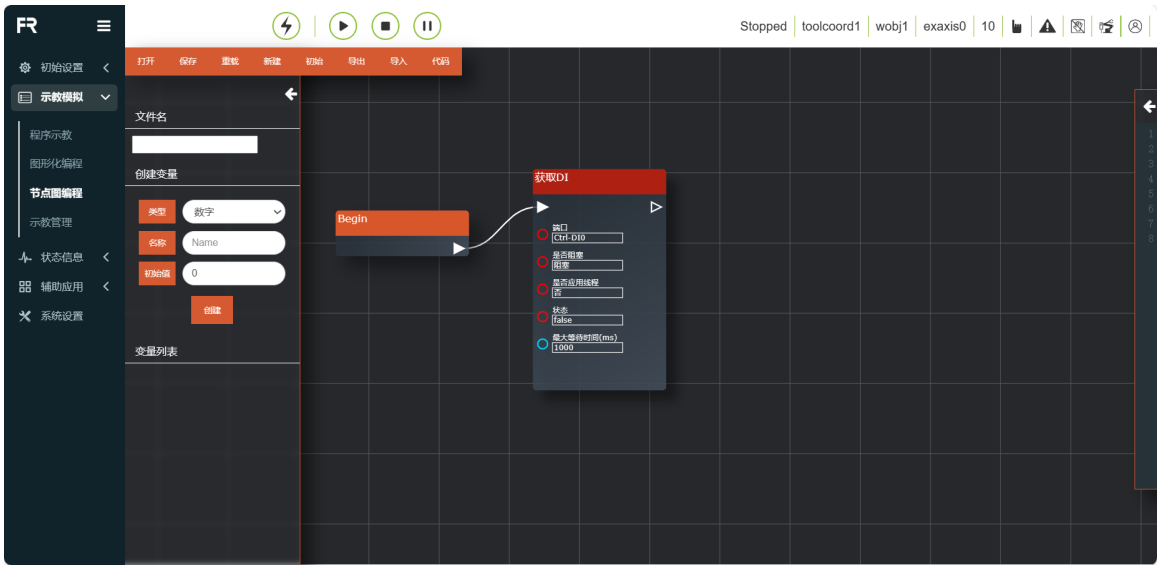
- 端口：Ctrl-DO0 ~ Ctrl-CO7(阻塞:SetDO, 非阻塞:SPLCSetDO,[0~15]), End-DO0 ~ End-DO1(阻塞:SetToolDO, 非阻塞:SPLCSetToolDO,[0~1])
- 状态：false/true
- 是否阻塞：阻塞/非阻塞
- 平滑轨迹：Break/Serious
- 是否应用线程：否/是



图表 6.2.25 “设置 DO”指令节点界面

2. “获取 DI”指令节点, 参数:

- 端口：Ctrl-DI0 ~ Ctrl-CI7(阻塞:GetDI, 非阻塞:SPLCGetDI,[0~15]), End-DI0 ~ End-DI1(阻塞:GetToolDI, 非阻塞:SPLCGetToolDI,[0~1])
- 是否阻塞：阻塞/非阻塞
- 状态：false/true
- 最大等待时间 (ms)：0 ~ 10000
- 是否应用线程：否/是



图表 6.2.26 “获取 DI”指令节点界面

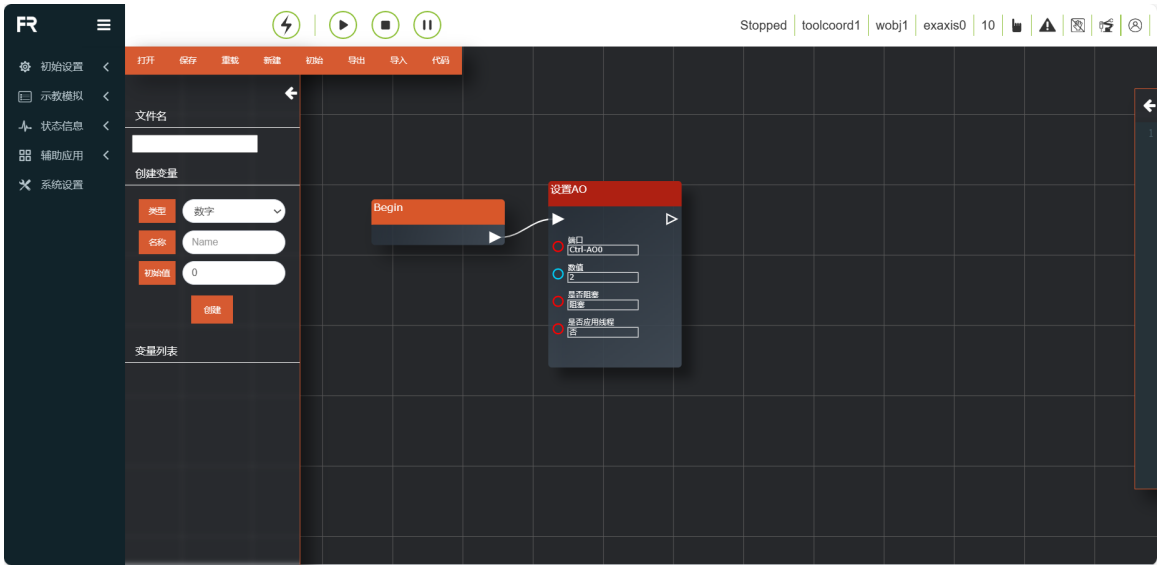
1.3.6.2.4.2 模拟 AI 命令

点击“设置 AO”/“获取 AI”指令节点，进入节点图编辑界面

在该指令中，分为设置模拟输出（SetAO/SPLCSetAO）和获取模拟输入（GetAI/SPLCGetAI）两部分功能。

1. “设置 AO”指令节点, 参数:

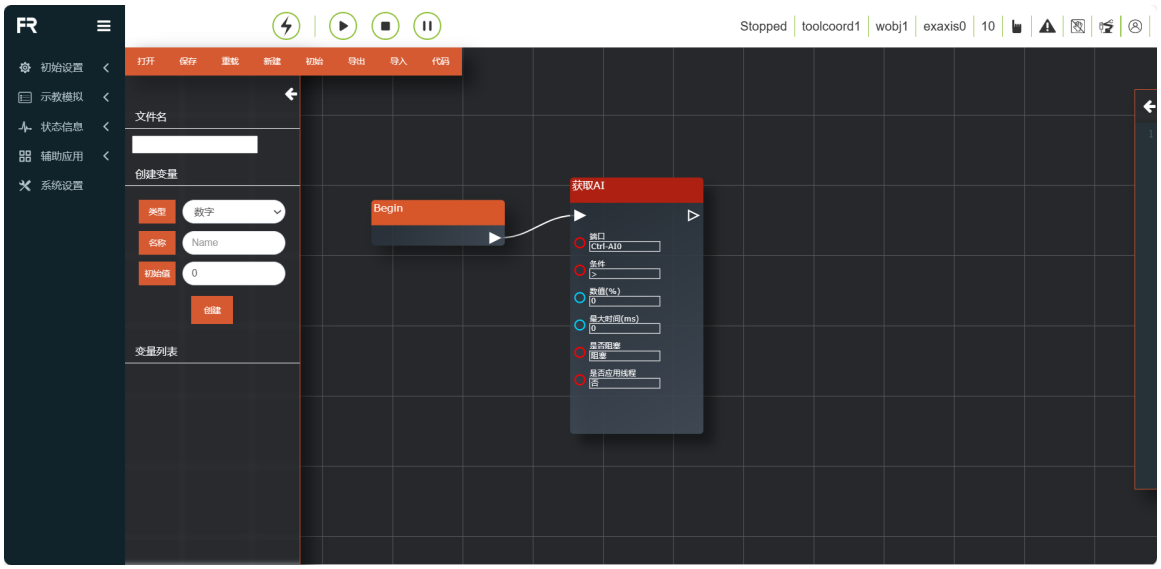
- 端口：Ctrl-AO0 ~ Ctrl-AO1(阻塞:SetAO，非阻塞:SPLCSetAO,[0~1])，End-AO0(阻塞:SetToolAO，非阻塞:SPLCSetToolAO,[0])
- 数值 (%)：0 ~ 100
- 是否阻塞：阻塞/非阻塞
- 是否应用线程：否/是



图表 6.2.27 “设置 AO”指令节点界面

2. “获取 AI”指令节点, 参数:

- 端口：Ctrl-AI0 ~ Ctrl-DI1(阻塞:GetAI，非阻塞:SPLCGetAI,[0~1])，End-AI0(阻塞:GetToolAI，非阻塞:SPLCGetToolAI,[0])
- 条件：大于/小于
- 数值 (%)：0 ~ 100
- 最大时间 (ms)：0 ~ 10000
- 是否阻塞：阻塞/非阻塞
- 是否应用线程：否/是



图表 6.2.28 “获取 AI”指令节点界面

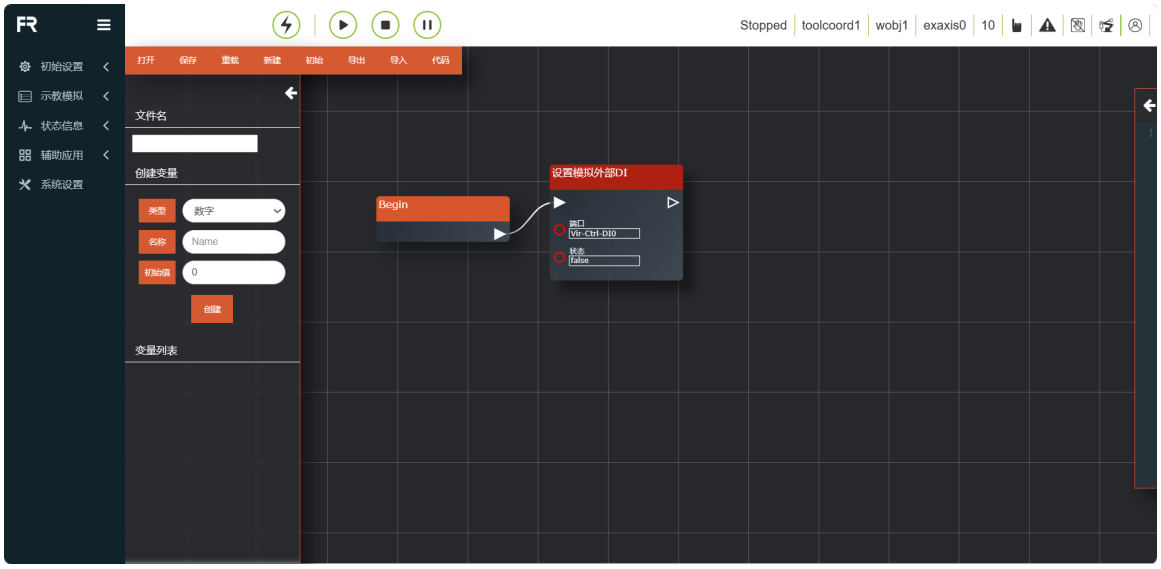
1.3.6.2.4.3 虚拟 IO 命令节点

点击“设置模拟外部 DI”/“设置模拟外部 AI”指令节点, 进入节点图编辑界面

该指令虚拟的 IO 控制指令，可以实现设置模拟外部 DI 和 AI 状态，获取模拟 DI 和 AI 状态。

1. “设置模拟外部 DI”指令节点, 参数:

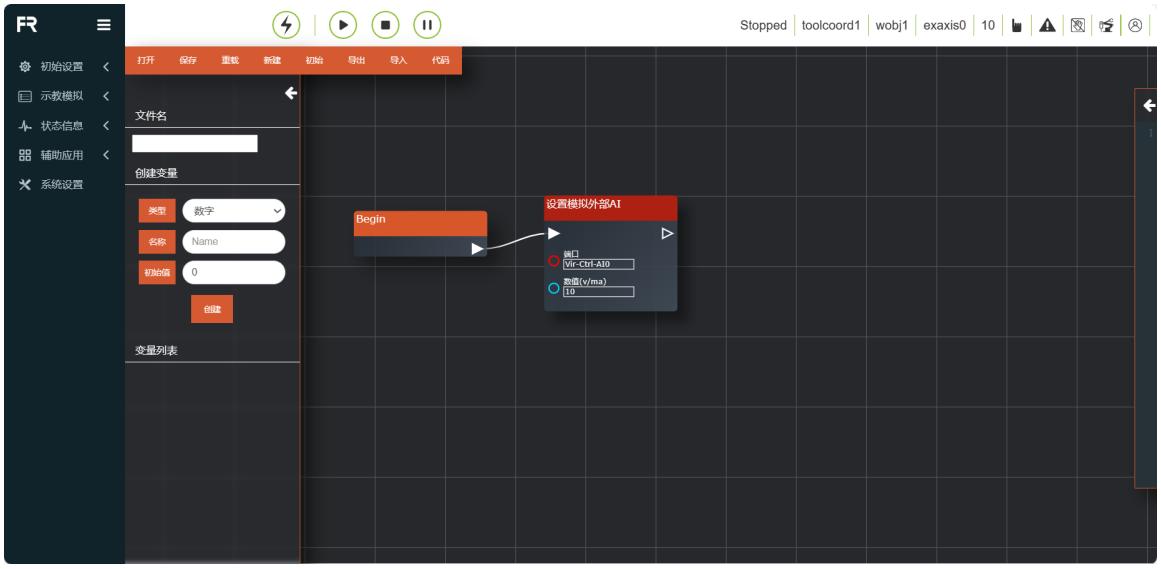
- 端口：Vir-Ctrl-DI0 ~ Vir-Ctrl-DI15(SetVirtualDI,[0~15]), Vir-End-DI0 ~ Vir-End-DI1(SetVirtualToolDI,[1~2])
- 状态：false/true



图表 6.2.29 “设置模拟外部 DI”指令节点界面

2. “设置模拟外部 AI”指令节点, 参数:

- 端口： Vir-Ctrl-AI0 ~ Vir-Ctrl-AI0(SetVirtualAI,[0~1]), Vir-End-AI0(SetVirtualToolAI,[0])
- 数值 (v/ma)： 0 ~ 20



图表 6.2.30 “设置模拟外部 AI”指令节点界面

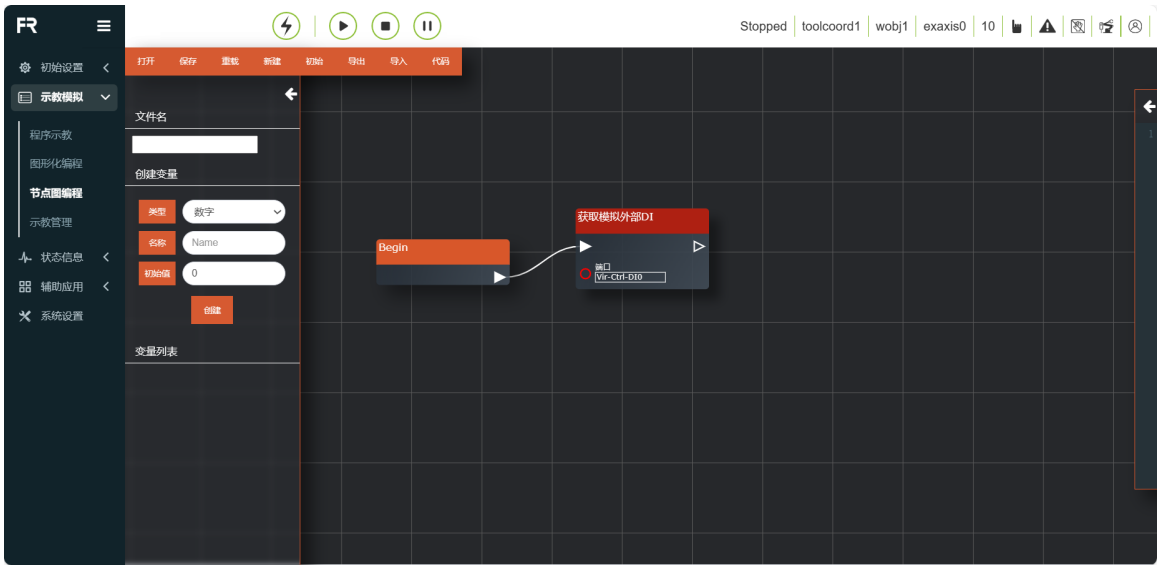
1.3.6.2.4.4 扩展 IO 命令节点

点击“获取模拟外部 DI” / “获取模拟外部 AI”指令节点, 进入节点图编辑界面

Aux-IO 是机器人与 PLC 通讯控制外部扩展 IO 的指令功能，需要机器人与 PLC 建立 UDP 通讯。

1. “获取模拟外部 DI”指令节点, 参数:

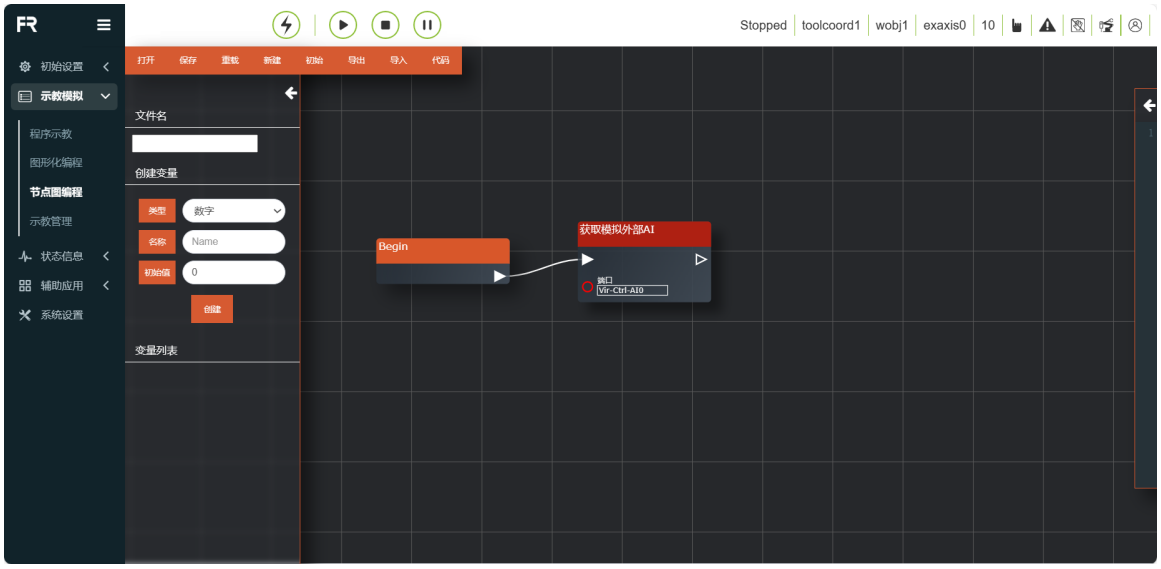
- 端口 : Vir-Ctrl-DI0 ~ Vir-Ctrl-DI15(GetVirtualDI,[0~15]), Vir-End-DI0 ~ Vir-End-DI1(GetVirtualToolDI,[1~2])



图表 6.2.31 “获取模拟外部 DI”指令节点界面

2. “设置模拟外部 AI”指令节点, 参数:

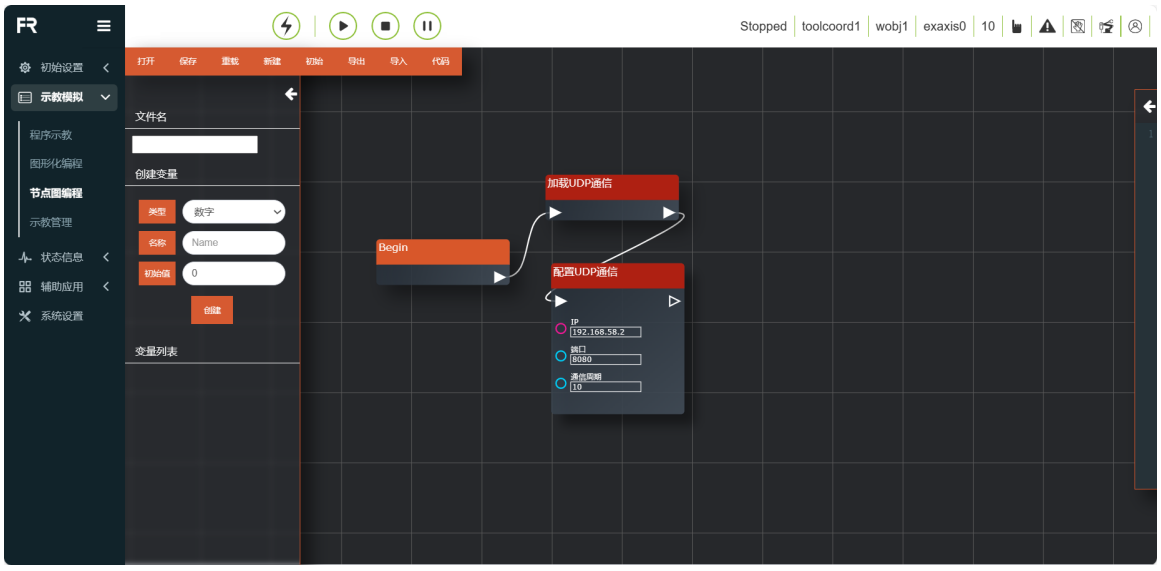
   - 端口：Vir-Ctrl-AI0 ~ Vir-Ctrl-AI0(GetVirtualAI,[0~1]), Vir-End-AI0(GetVirtualToolAI,[0])



图表 6.2.32 “设置模拟外部 AI”指令节点界面

3. “配置 UDP 通信”指令节点, 参数:

   - ip：ip 地址
   - 端口：端口号
   - 通信周期 (ms)：0 ~ 10000



图表 6.2.33 “配置 UDP 通信”指令节点界面

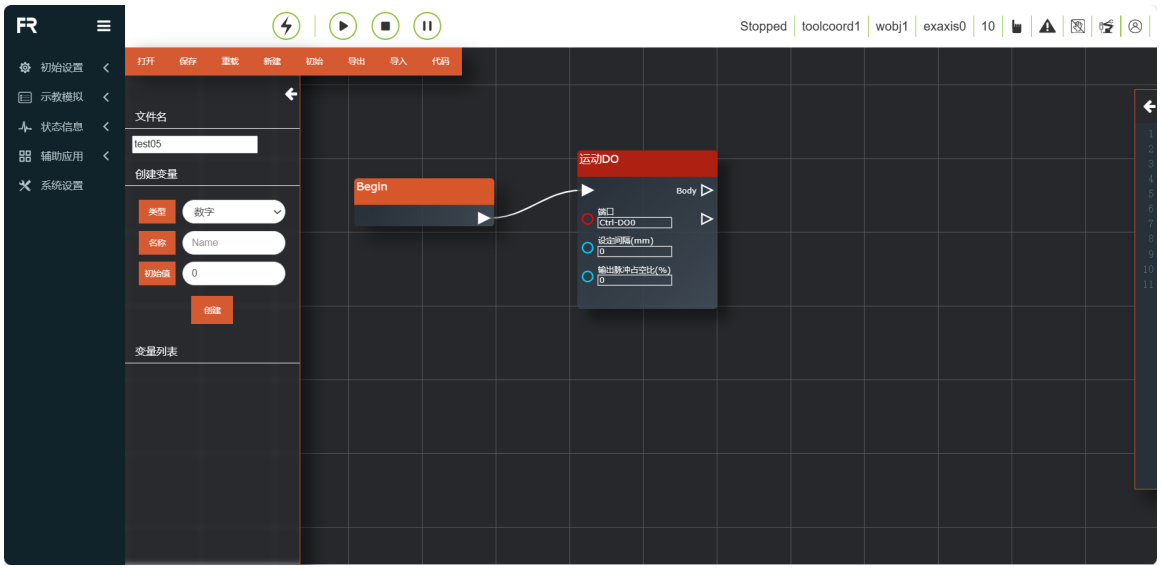
1.3.6.2.4.5 运动 DO 命令

点击“运动 DO”指令节点, 进入节点图编辑界面

该指令实现直线运动过程中，根据设定的间隔，连续输出 DO 信号功能。

“运动 DO”指令节点, 参数:

- 端口：Ctrl-DO0 ~ Ctrl-DO0(MoveDOStart,[0~15]), End-DO1(MoveDOStart,[0~1])
- 设定间隔 (mm): 0 ~ 500
- 输出脉冲占空比 (%): 0 ~ 99



图表 6.2.34 “运动 DO”指令节点界面

1.3.6.2.4.6 坐标系命令

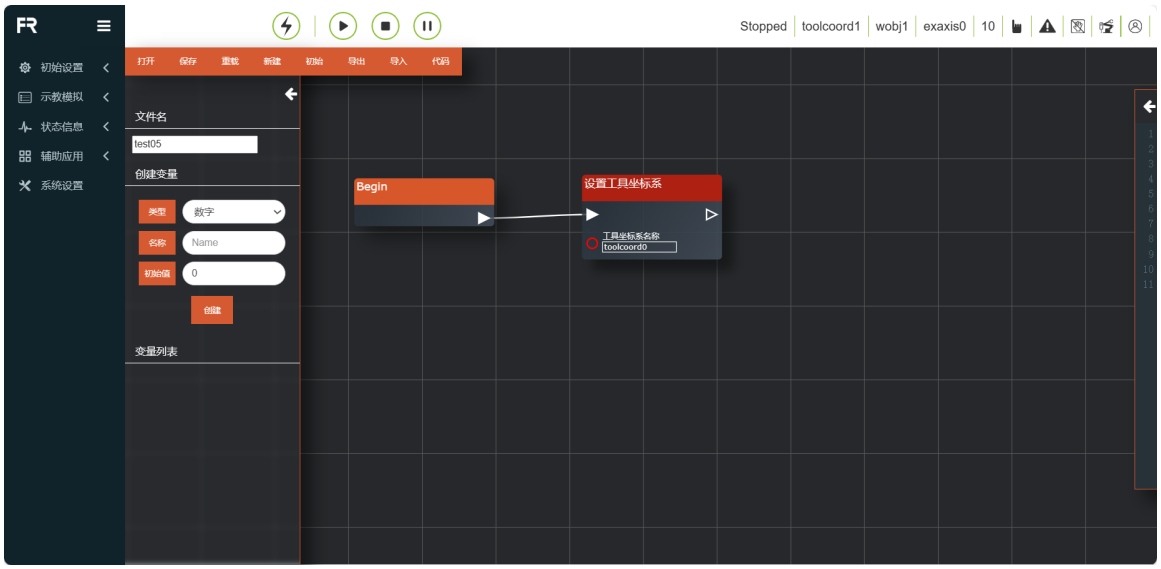
点击“设置工具坐标系”/“设置工件坐标系”相关指令节点, 进入节点图编辑界面

在该指令中，分为“设置工具坐标系”和“设置工件坐标系”两部分功能。

选择工具坐标系名称，点击“应用”添加该指令到程序中，当程序运行该语句，会设定机器人的工具坐标系。

1. “设置工具坐标系”指令节点, 参数:

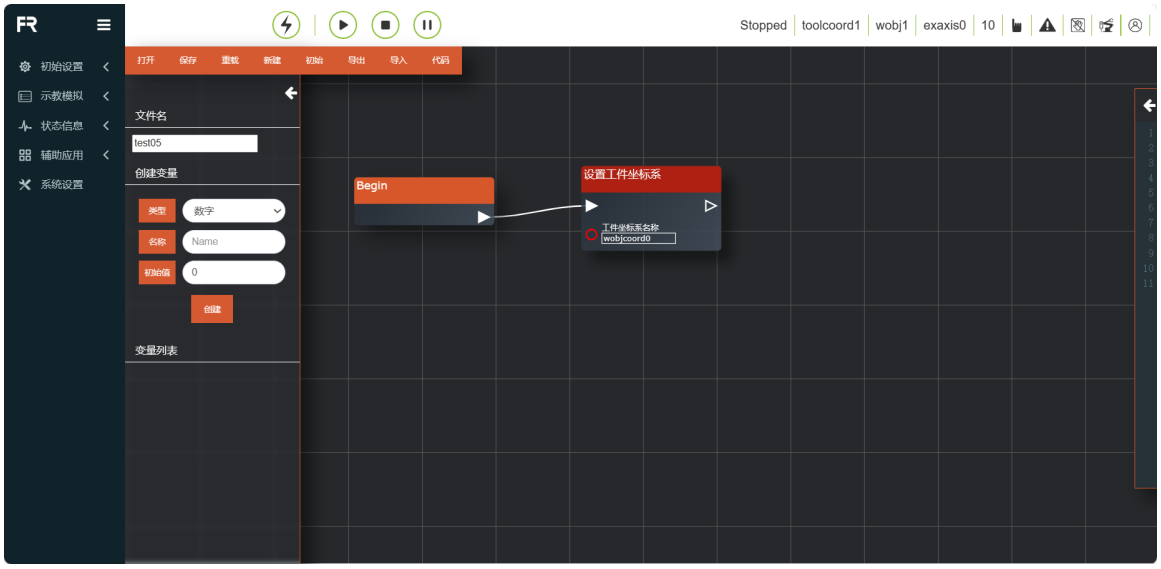
- 工具坐标系名称：toolcoord1 ~ toolcoord14(SetToolList,[0~14]), etoolcoord0 ~ etoolcoord14(SetExToolList, [0~14])



图表 6.2.35 “设置工具坐标系”指令节点界面

2. “设置工件坐标系”指令节点, 参数:

- 工件坐标系名称：wobjcoord1 ~ wobjcoord14



图表 6.2.36 “设置工件坐标系”指令节点界面

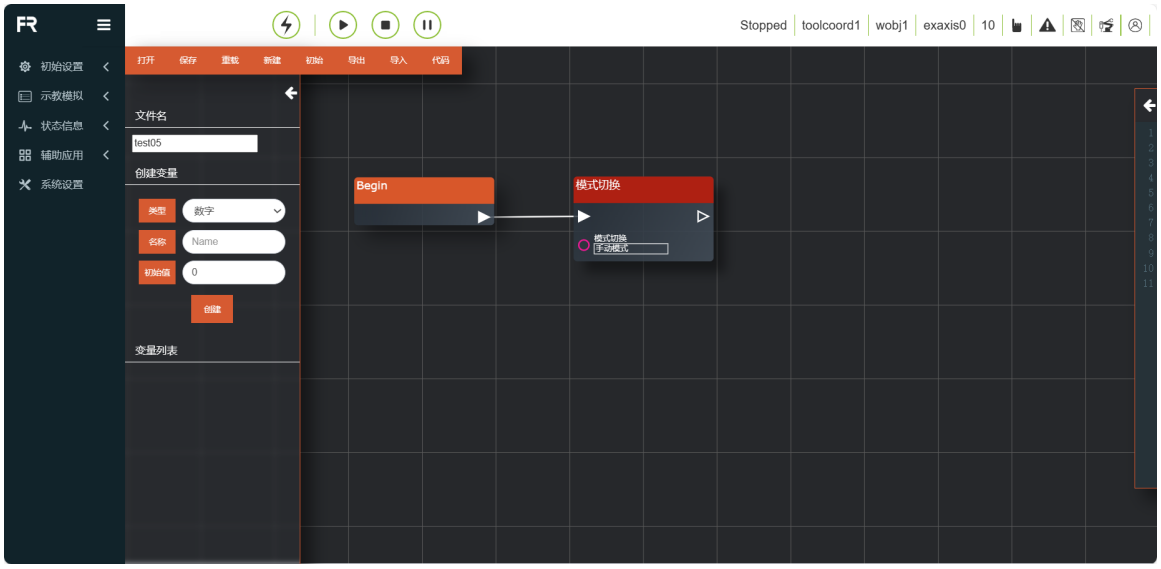
1.3.6.2.4.7 “模式切换”命令

点击“模式切换”指令节点，进入节点图编程界面

该指令可切换机器人到手动模式，通常在一个程序结尾处添加，以便用户在程序运行结束后，使机器人自动切换到手动模式，拖动机器人。

“模式切换”指令节点, 参数:

- 模式切换：手动模式



图表 6.2.37 “模式切换”指令节点界面

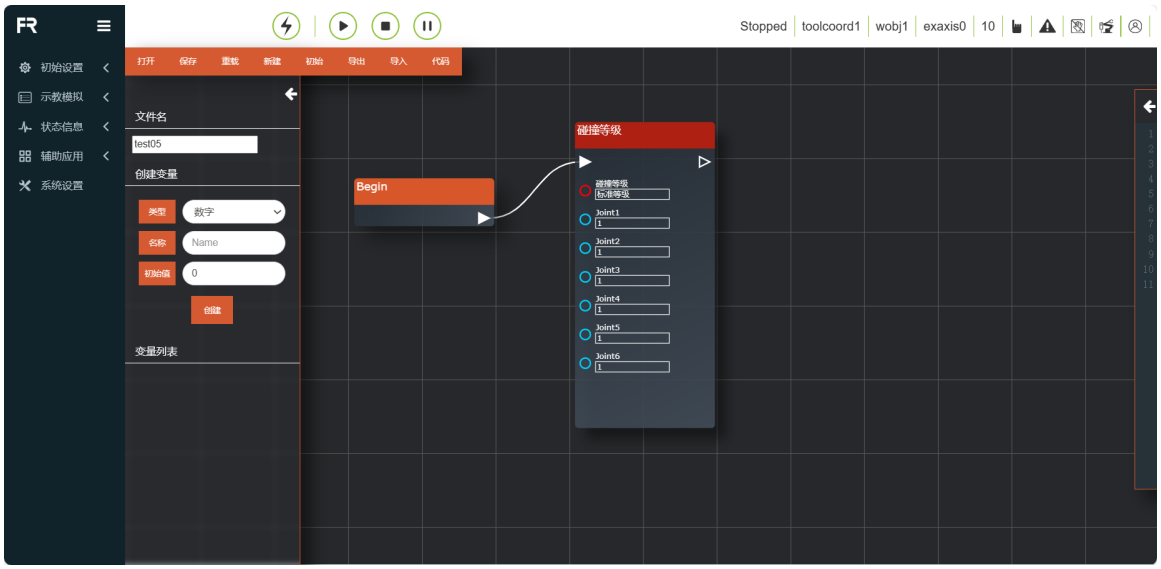
1.3.6.2.4.8 “碰撞等级”命令

点击“碰撞等级”指令节点，进入节点图编程界面

该指令碰撞等级设置，通过该指令可以在程序运行中实时调节各轴碰撞等级，更灵活的部署应用场景。

“碰撞等级”指令节点, 参数:

- 标准等级：标准等级/自定义百分比
- joint1-joint6(N): 0 ~ 100，碰撞阈值，数组型



图表 6.2.38 “碰撞等级”指令节点界面

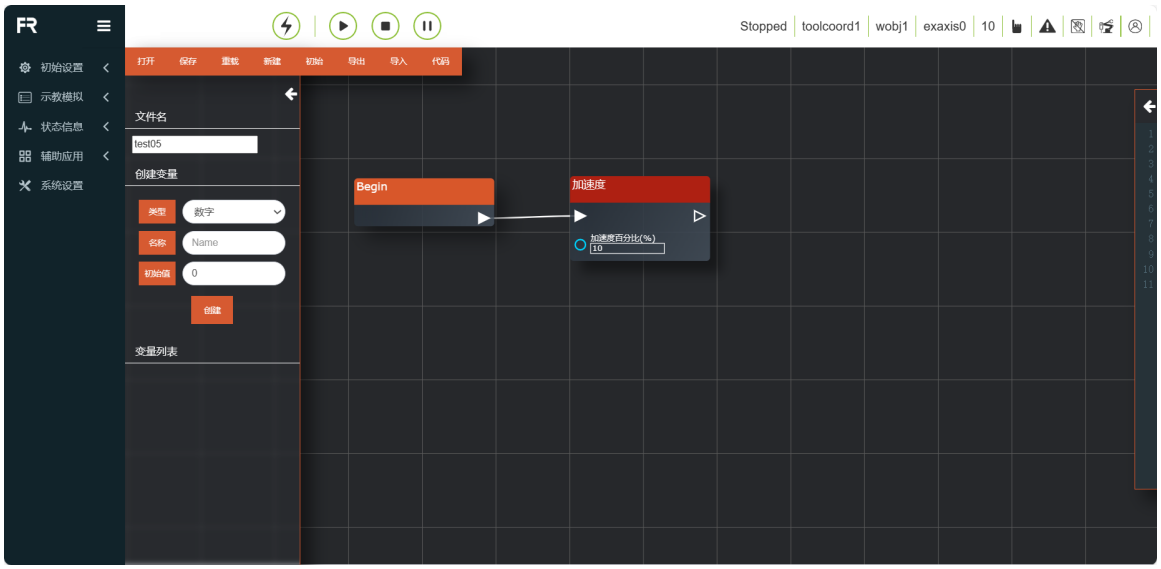
1.3.6.2.4.9 “加速度”命令

点击“加速度”指令节点，进入节点图编程界面

“加速度”命令是实现机器人加速度可单独设置功能，通过调节运动指令加速度缩放因子，可以增加或减小加减速时间，实现机器人动作节拍时间可调。

“加速度”指令节点, 参数:

- 加速度百分比 (%)：0 ~ 100



图表 6.2.39 “加速度”指令节点界面

1.3.6.2.5 “外设”指令界面

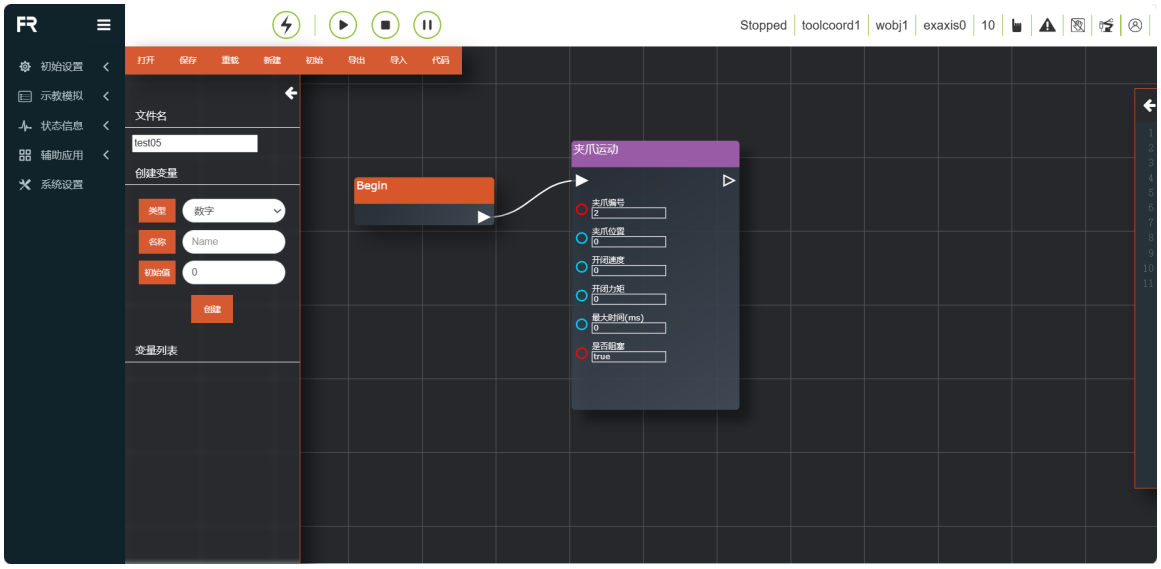
1.3.6.2.5.1 夹爪指令

该指令分为“夹爪运动”、“夹爪激活”和“夹爪复位”。

指令中，显示完成配置并且已被激活的夹爪编号，对夹爪开闭、开闭速度和开闭力矩的设置，数值为百分比，是否阻塞功能选项，选择阻塞即夹爪运动需等待上一条运动指令执行完才执行，选择非阻塞即夹爪运动与上一条运动指令并行。

“夹爪运动”节点, 参数:

- 夹爪编号：已被激活的夹爪编号
- 夹爪位置：0~100
- 开闭速度：0~100
- 开闭力矩：0~100
- 最大时间 (ms)：0~30000
- 是否阻塞：false/true

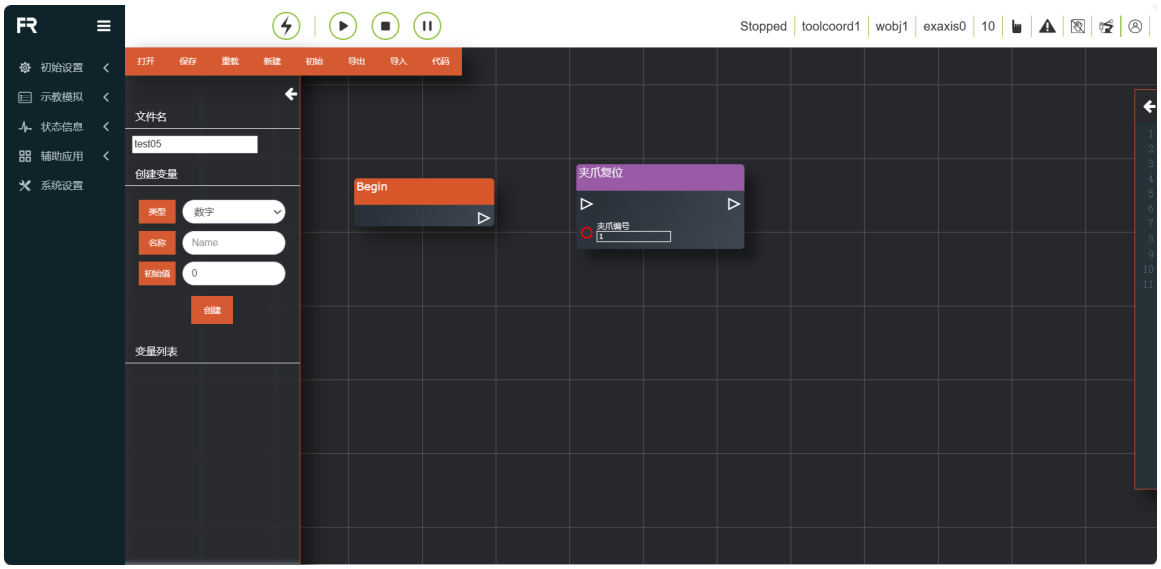


图表 6.2.40 “夹爪运动”节点界面

夹爪复位指令，显示已经配置的夹爪编号，可以添加夹爪复位指令到程序中。

“夹爪复位”节点, 参数:

- 夹爪编号：已被激活的夹爪编号

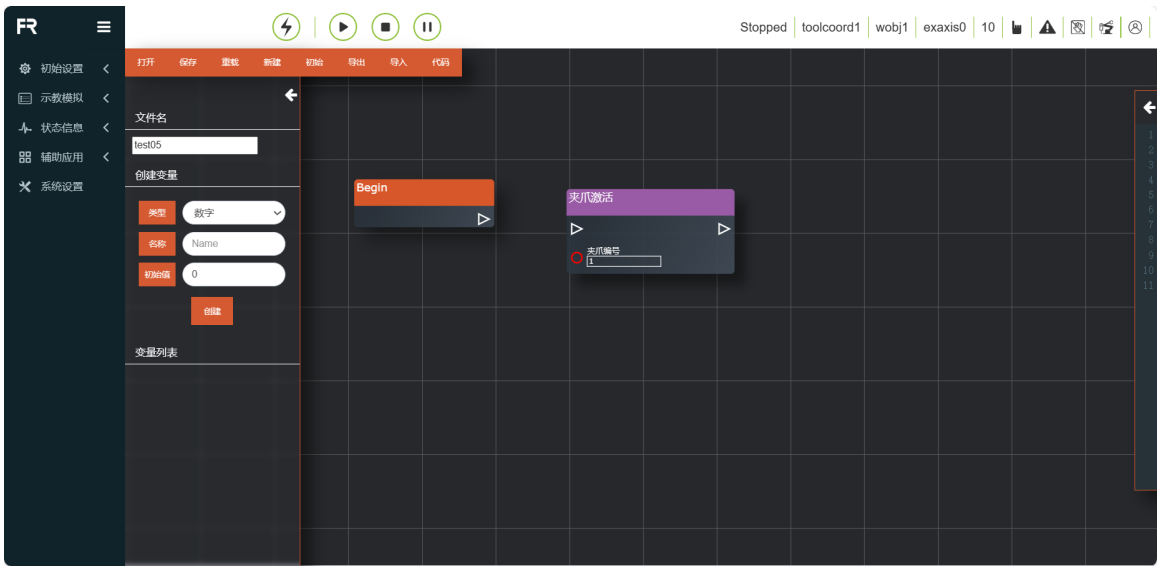


图表 6.2.41 “夹爪复位”节点界面

夹爪激活指令，显示已经配置的夹爪编号，可以添加夹爪激活指令到程序中。

“夹爪激活”节点, 参数:

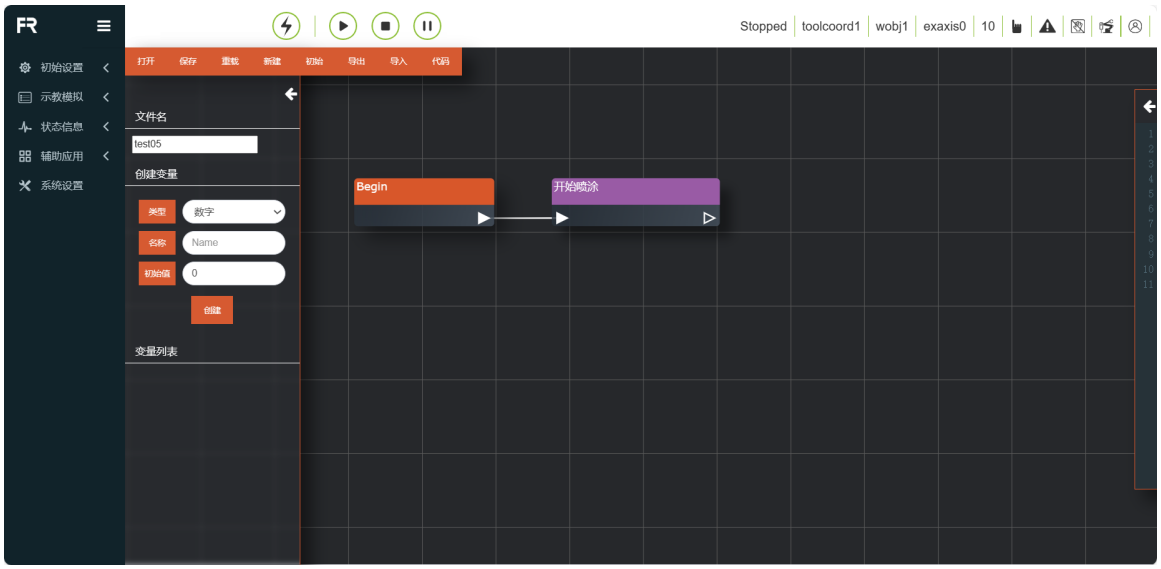
- 夹爪编号：已被激活的夹爪编号



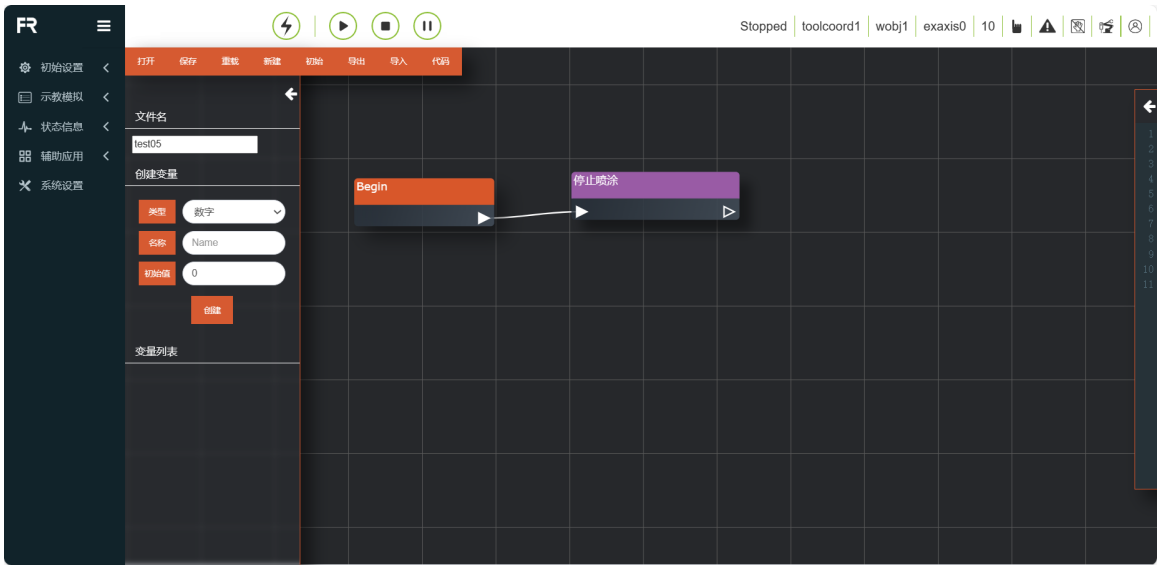
图表 6.2.42 “夹爪激活”节点界面

1.3.6.2.5.2 喷枪指令

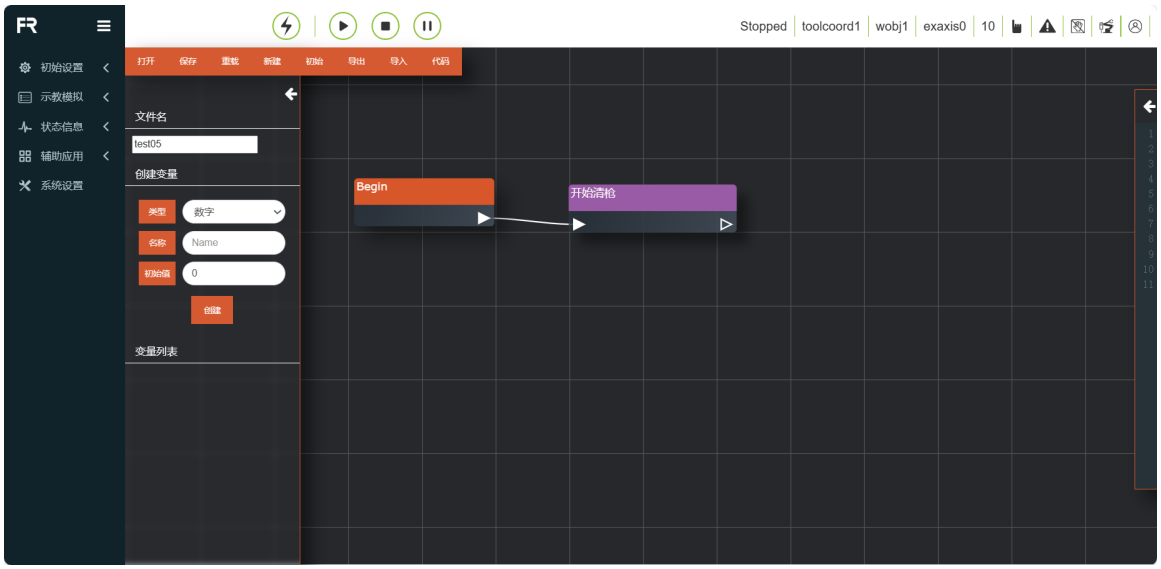
该指令为喷涂相关指令，控制喷枪“开始喷涂”、“停止喷涂”、“开始清枪”和“停止清枪”。在编辑该程序相关节点时，需确认已经配置好喷枪外设，否则无法保存。详见机器人外设章节。



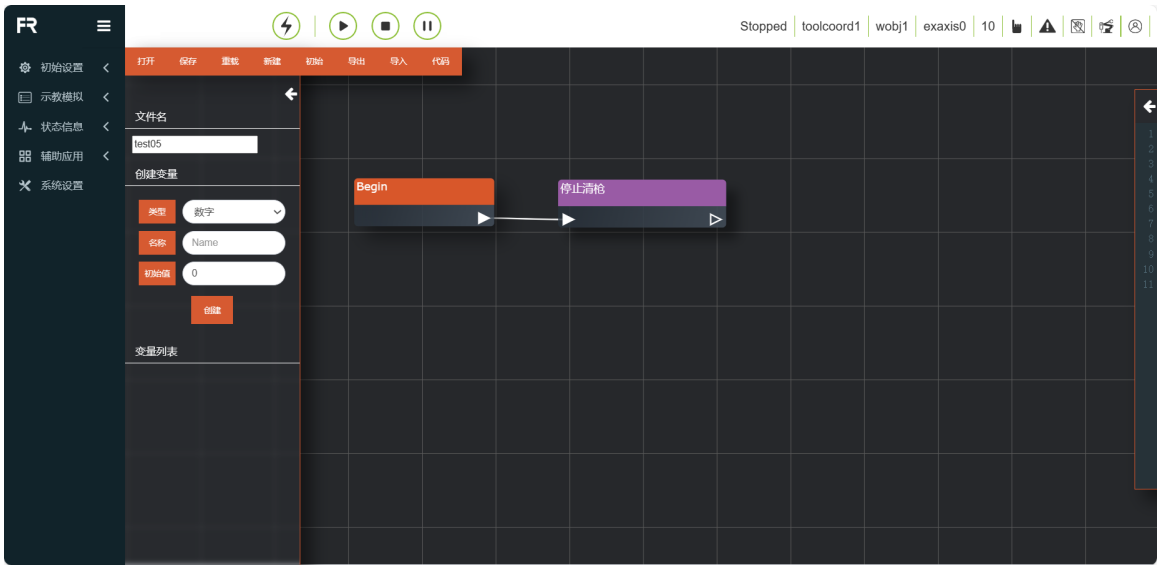
图表 6.2.43 “开始喷涂”指令节点界面



图表 6.2.44 “停止喷涂”指令节点界面



图表 6.2.45 “开始清枪”指令节点界面



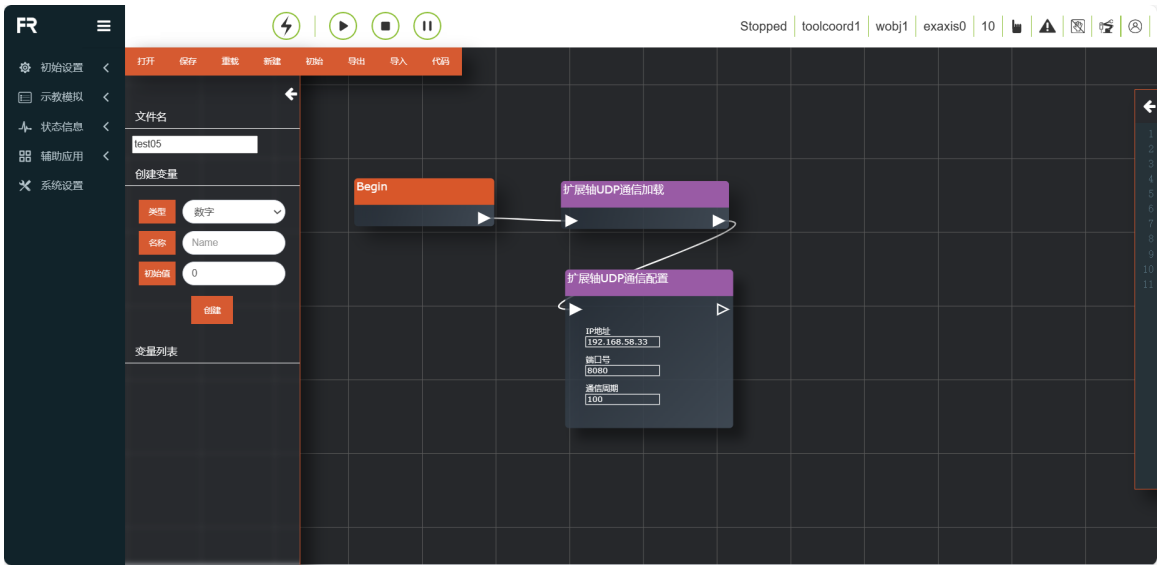
图表 6.2.46 “停止清枪”指令节点界面

1.3.6.2.5.3 扩展轴指令（控制器 +PLC）

该指令针对使用外部轴的场景，与 PTP 指令组合使用，可将空间上一点 X 轴方向上的移动分解到外部轴运动。选择外部轴编号，运动方式选同步，选择需要到达的点。

分为 UDP 通信加载/配置、异步运动、同步 PTP/LIN 运动、同步 ARC 运动、回零指令和使能指令。

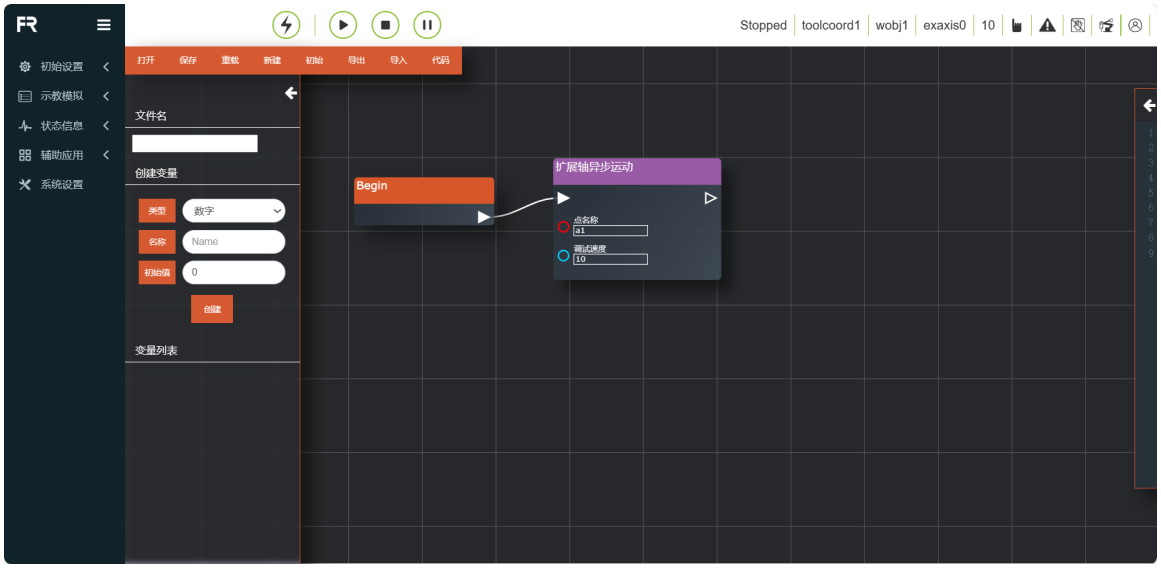
“UDP 通信配置”指令节点, 输入 IP 地址、端口号和通信周期；



图表 6.2.47 “UDP 通信配置”指令节点界面

“异步运动”指令节点, 参数:

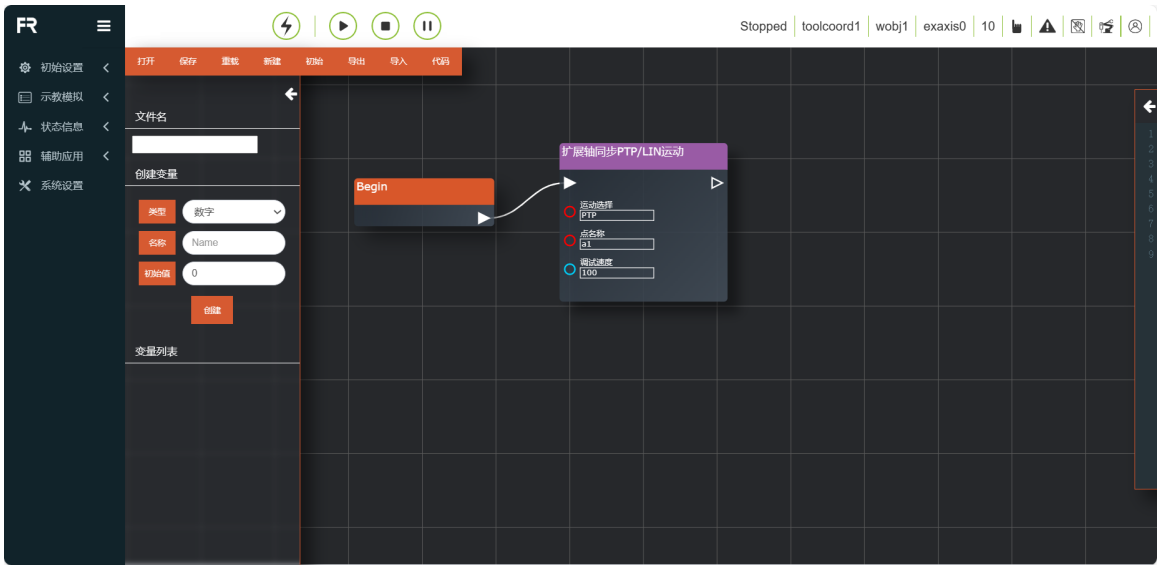
- 点名称：示教点位
- 调试速度 (%)：0~100



图表 6.2.48 “异步运动”指令节点界面

“同步 PTP/LIN 运动”指令节点, 参数:

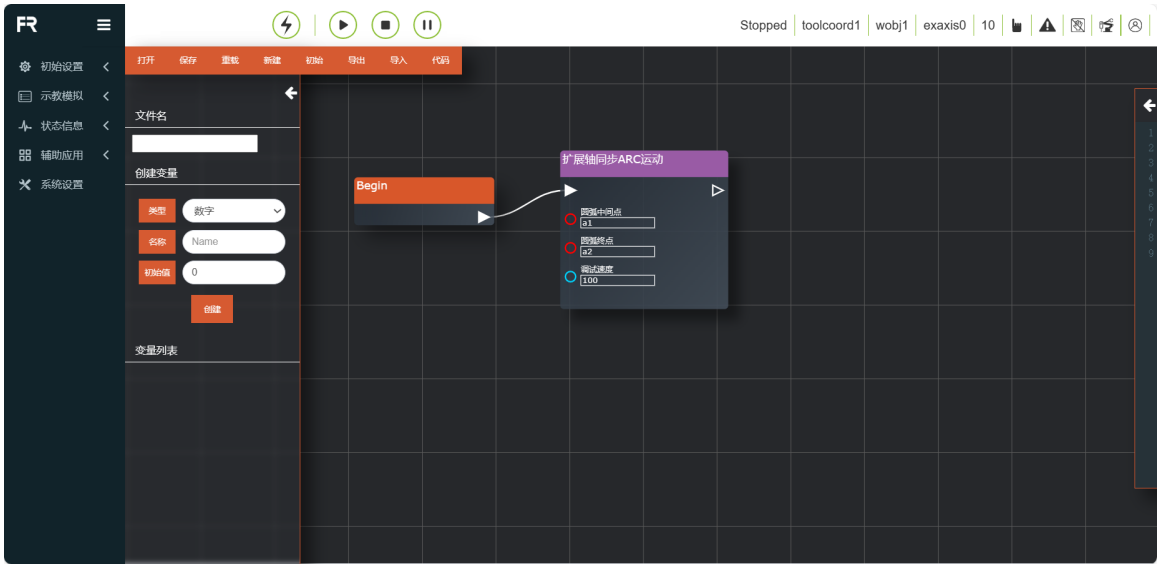
- 运动选择：PTP/LIN
- 点名称：示教点位
- 调试速度 (%)：0~100



图表 6.2.49 “同步 PTP/LIN 运动”指令节点界面

“同步 ARC 运动”指令节点, 默认运动方式为 ARC, 参数:

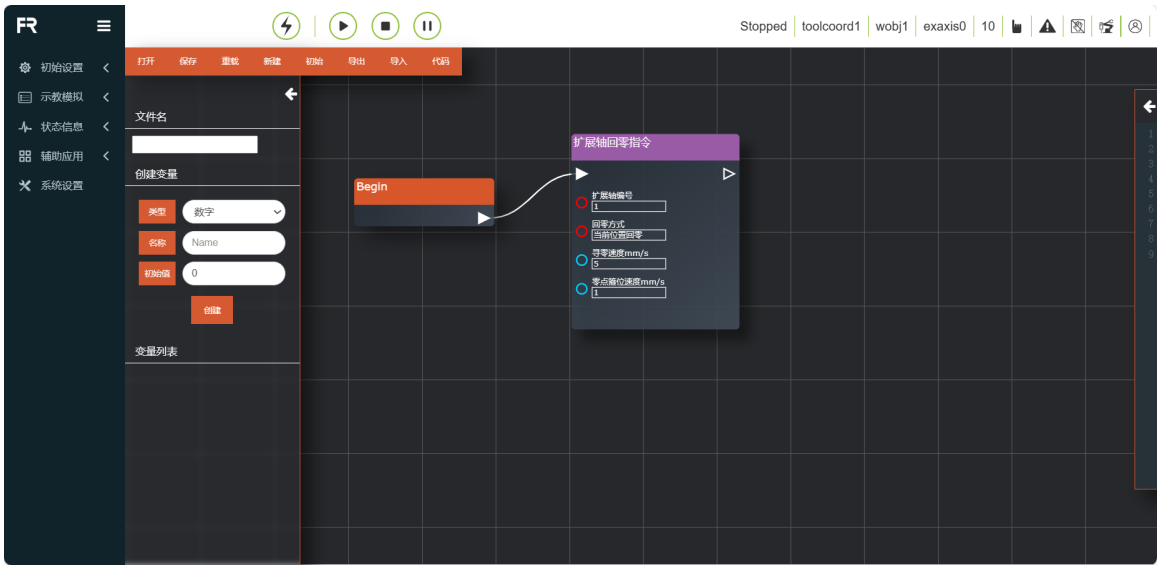
- 点名称：示教点位
- 调试速度 (%)：0~100



图表 6.2.50 “同步 ARC 运动”指令节点界面

“回零”指令节点,, 参数:

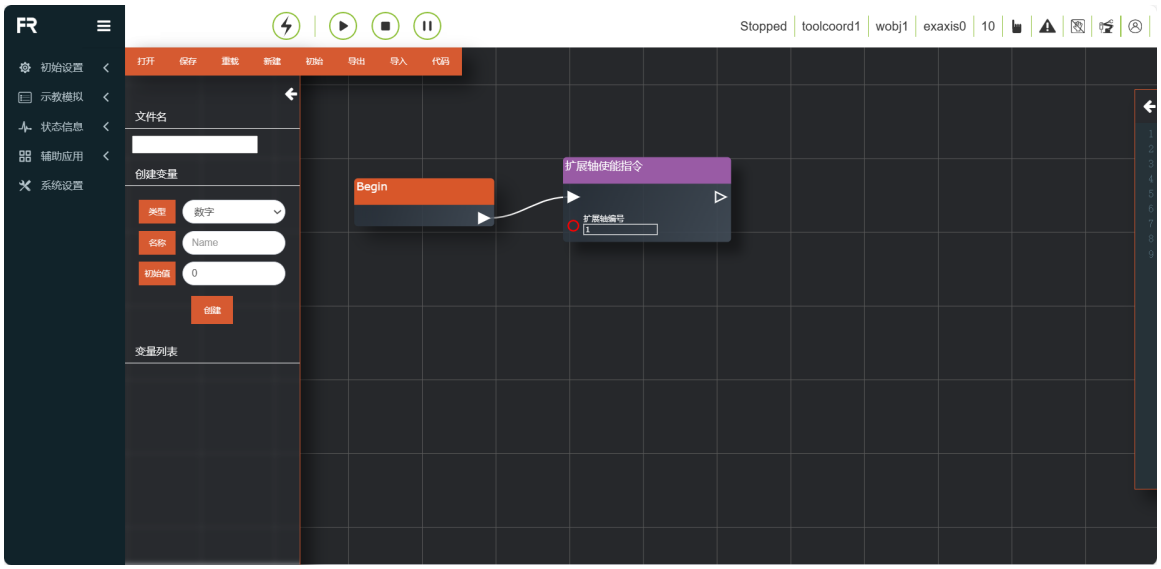
- 扩展轴编号：1~4
- 回零方式：当前位置回零/负限位回零/正限位回零
- 寻零速度：0~2000，默认位 5
- 零点箍位速度：0~2000，默认为 1



图表 6.2.51 “回零”指令节点界面

“使能”指令节点„ 参数:

- 扩展轴编号：1~4



图表 6.2.52 “使能”指令节点界面

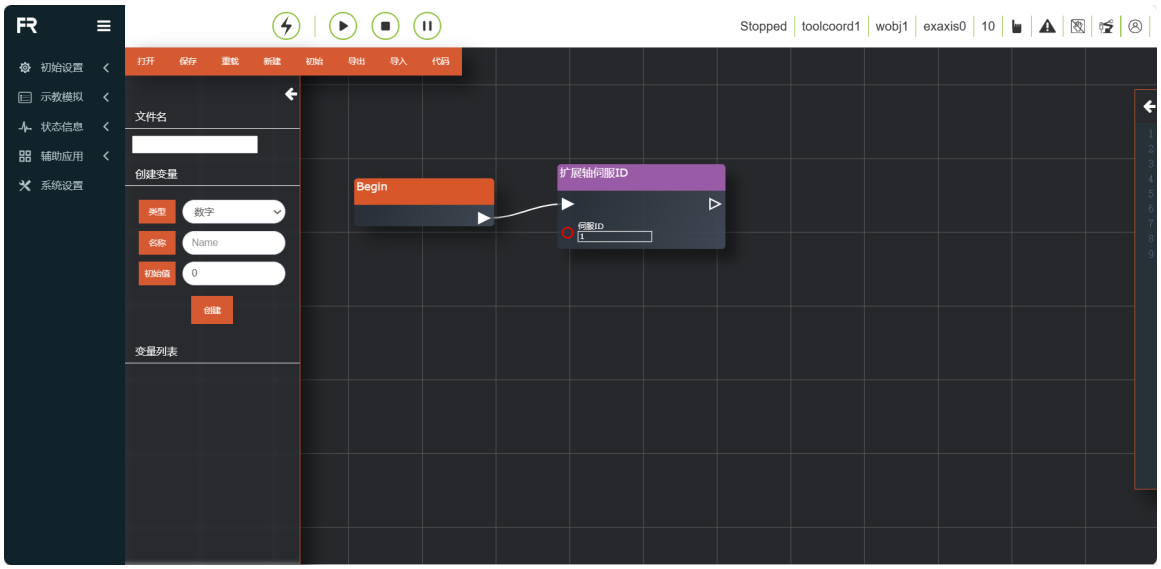
1.3.6.2.5.4 扩展轴指令（控制器 + 伺服驱动器）

该指令可对扩展轴参数进行配置。根据不同的控制模式设置不同的参数。已配置好的扩展轴，可对其零点设定。

分为伺服 ID、控制模式、伺服使能和伺服回零；控制模式中又分为位置模式和速度模式，这两个节点需要结合控制模式使用，否则单独添加无法生效。

“伺服 ID”指令节点„ 参数：

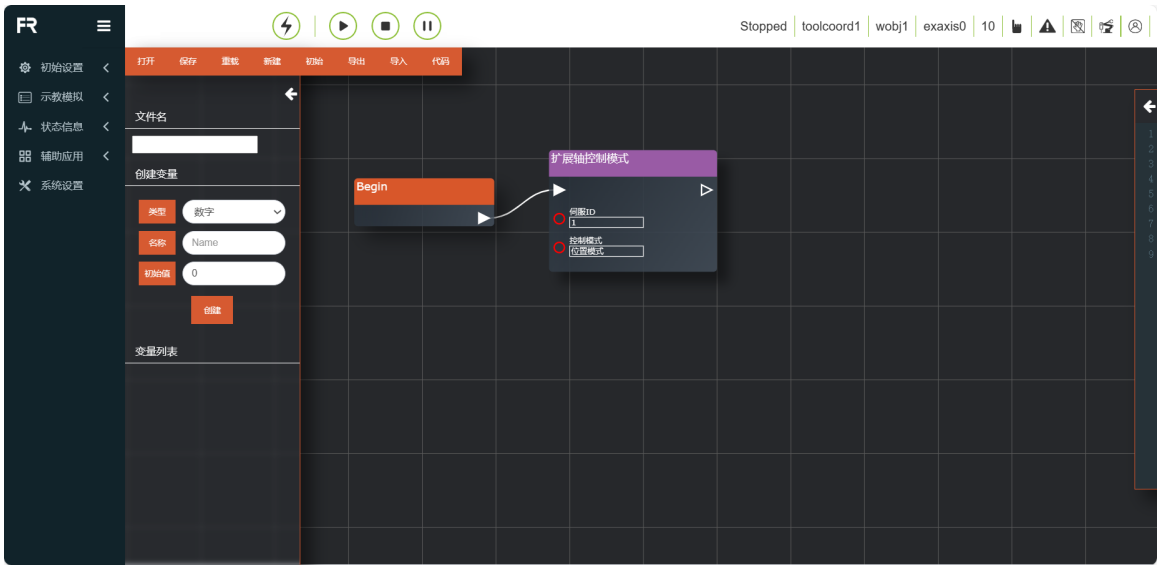
- 伺服 ID：1~15



图表 6.2.53 “伺服 ID”指令节点界面

“控制模式”指令节点„ 参数：

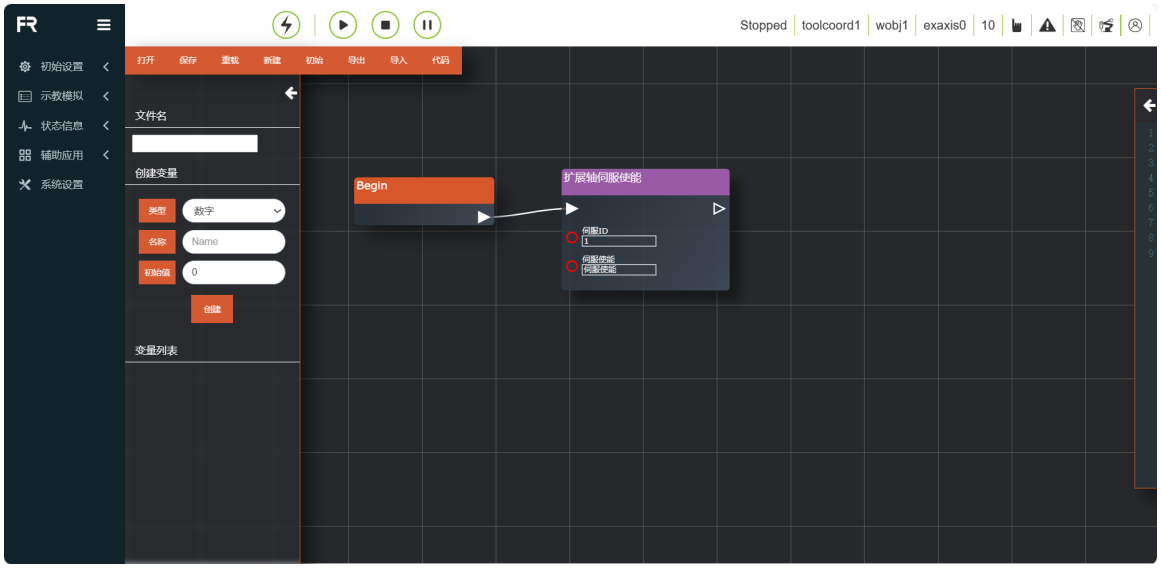
- 伺服 ID：1~15
- 控制模式：位置模式/速度模式



图表 6.2.54 “控制模式”指令节点界面

“伺服使能”指令节点„ 参数:

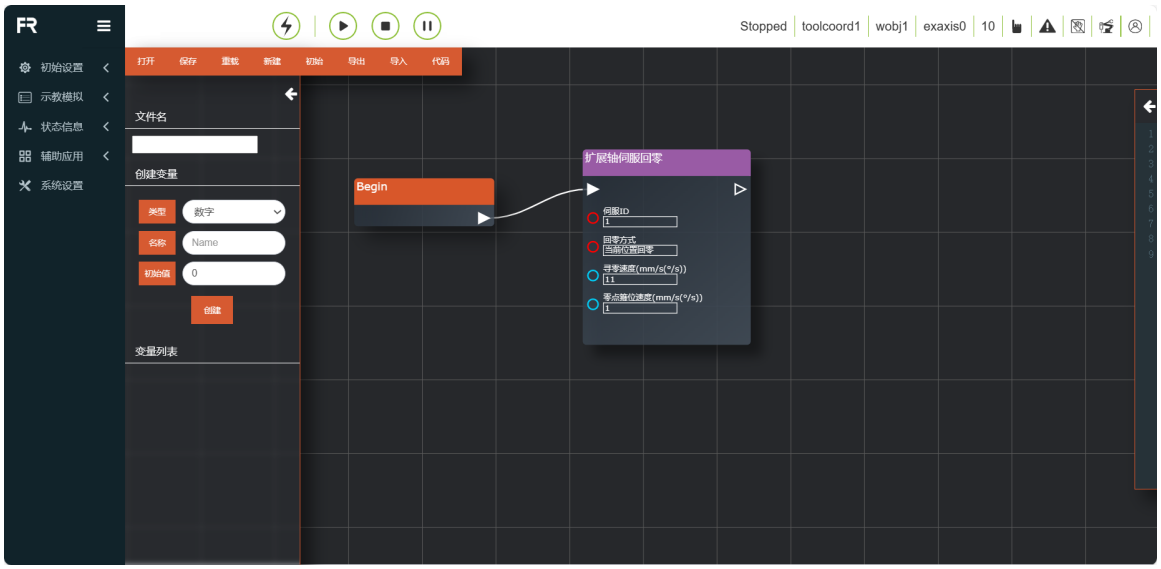
- 伺服 ID：1~15
- 伺服使能：伺服使能/去除使能



图表 6.2.55 “伺服使能”指令节点界面

“伺服回零”指令节点„ 参数:

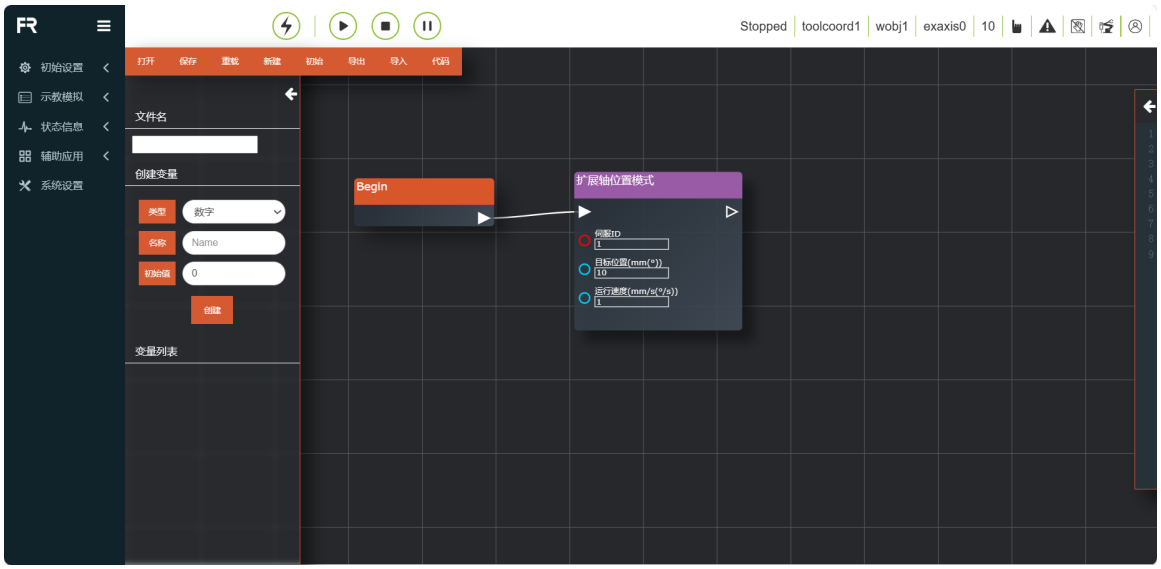
- 伺服 ID：1~15
- 回零方式：当前位置回零/负限位回零/正限位回零
- 寻零速度：0~2000，默认位 5
- 零点箍位速度：0~2000，默认为 1



图表 6.2.56 “伺服回零”指令节点界面

“位置模式”指令节点, 参数:

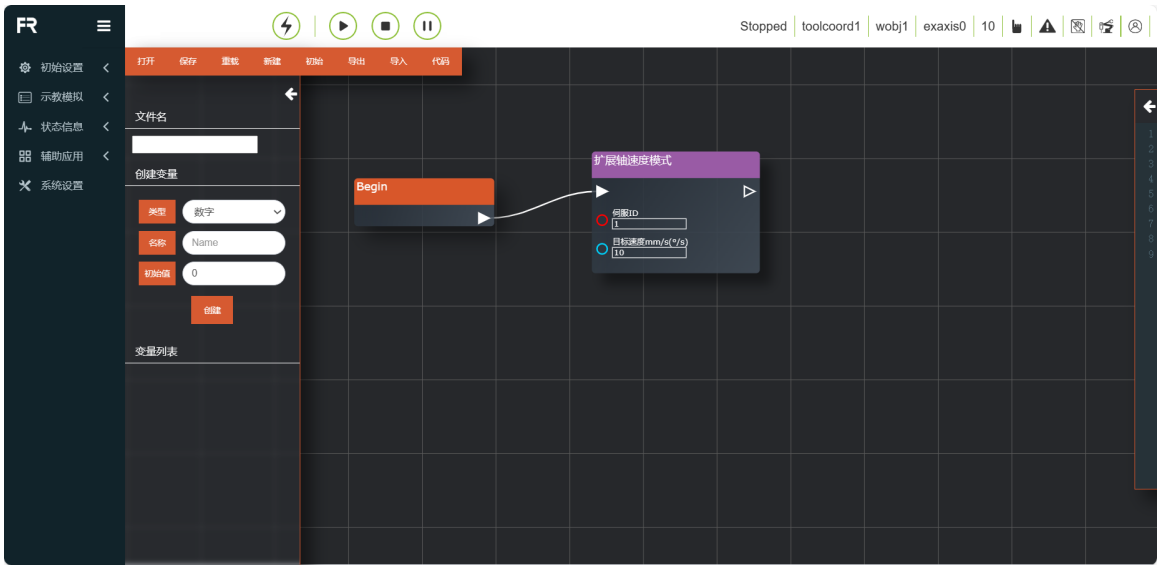
- 伺服 ID：1~15
- 目标位置：无限制
- 寻零速度：无限制



图表 6.2.57 “位置模式”指令节点界面

“速度模式”指令节点, 参数:

- 伺服 ID：1~15
- 目标速度：无限制



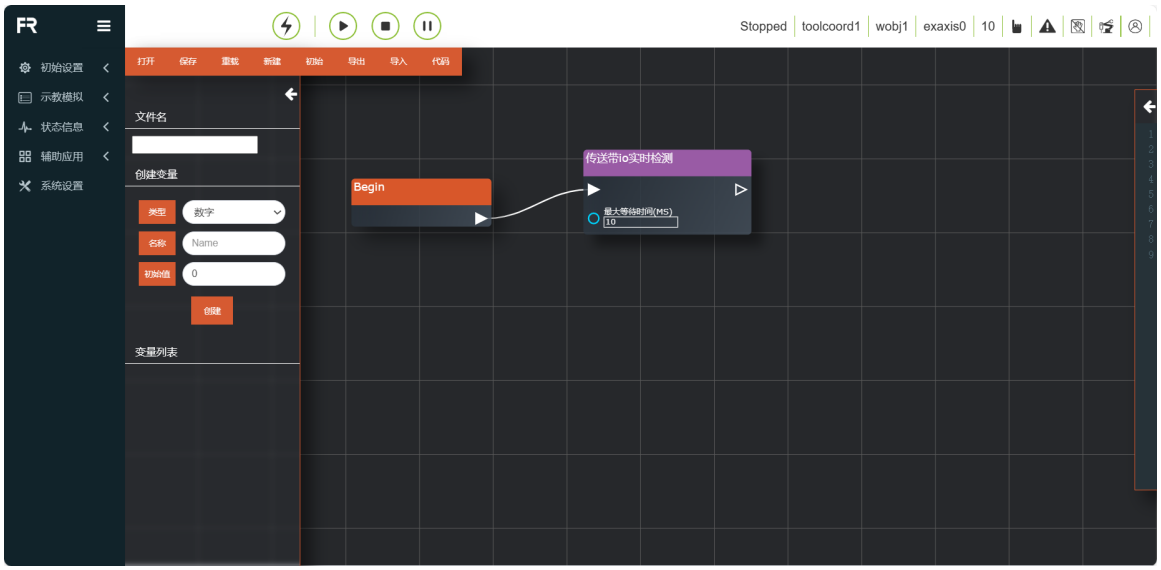
图表 6.2.58 “速度模式”指令节点界面

1.3.6.2.5.5 传送带指令

该指令包含 IO 实时检测，位置实时检测，跟踪开启和跟踪关闭四条命令。详见机器人外设章节

“IO 实时检测”指令节点, 参数:

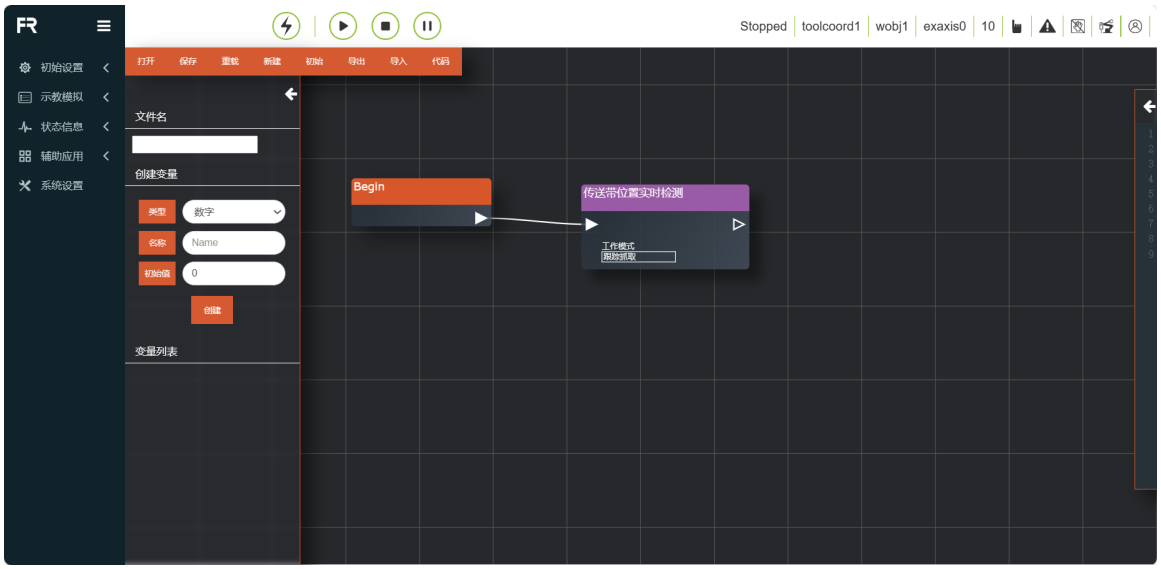
- 最大等待时间：0~10000



图表 6.2.59 “IO 实时检测”指令节点界面

“位置实时检测”指令节点, 参数:

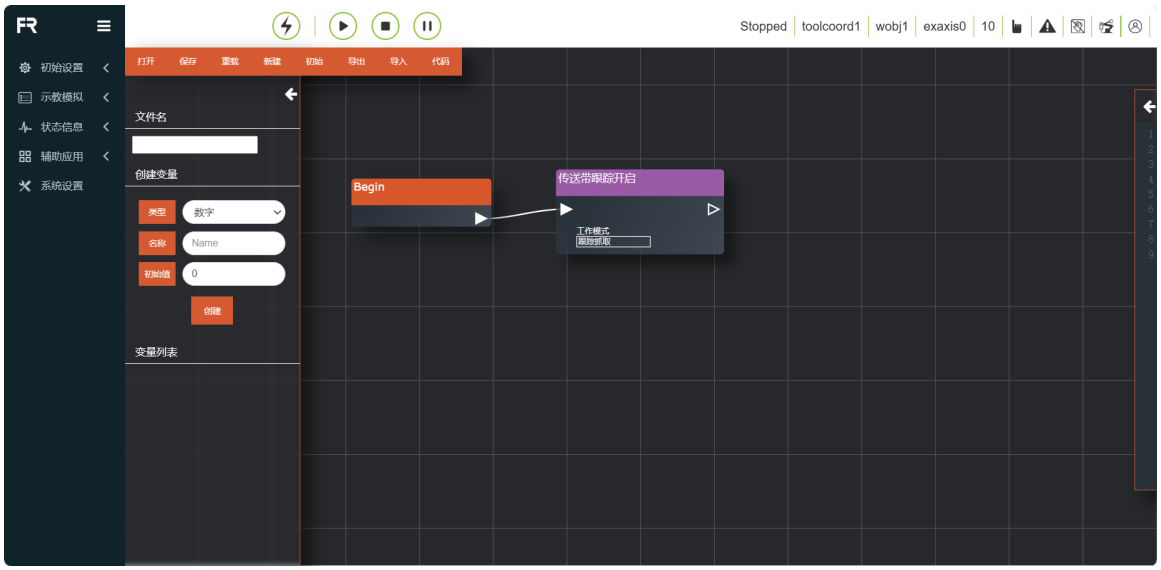
- 工作模式：跟踪抓取/跟踪运动/TPD 跟踪



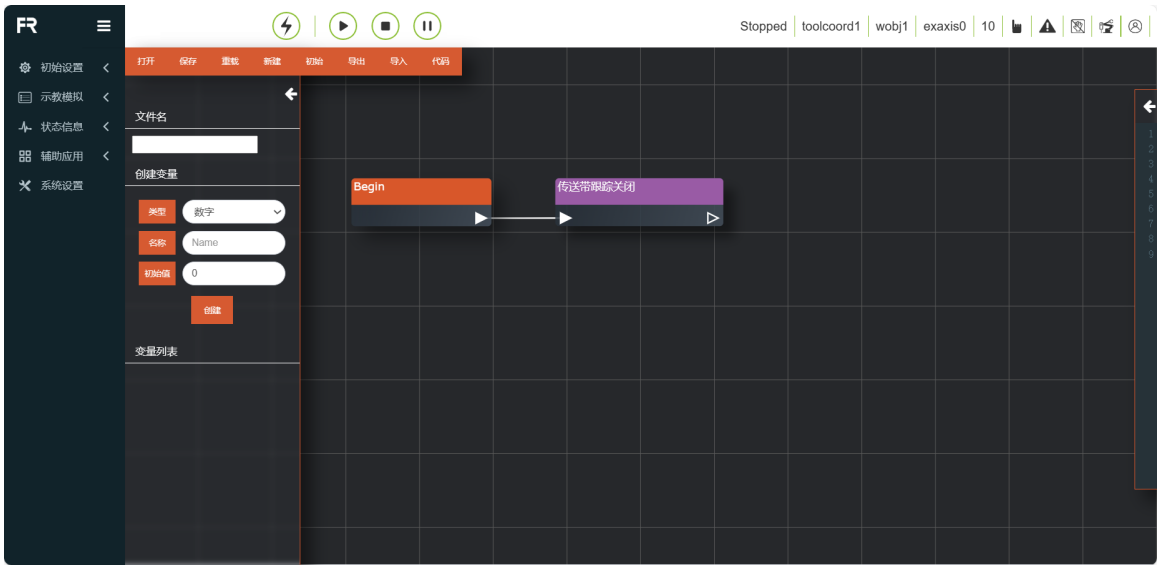
图表 6.2.60 “位置实时检测”指令节点界面

“跟踪开启”指令节点, 参数:

- 工作模式：跟踪抓取/跟踪运动/TPD 跟踪



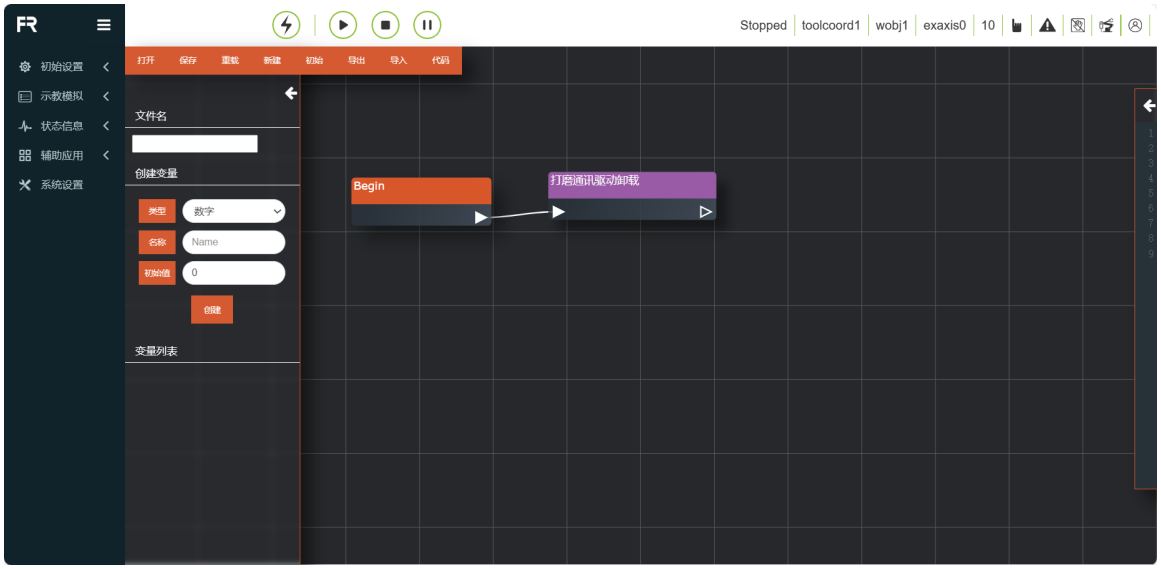
图表 6.2.61 “跟踪开启”指令节点界面



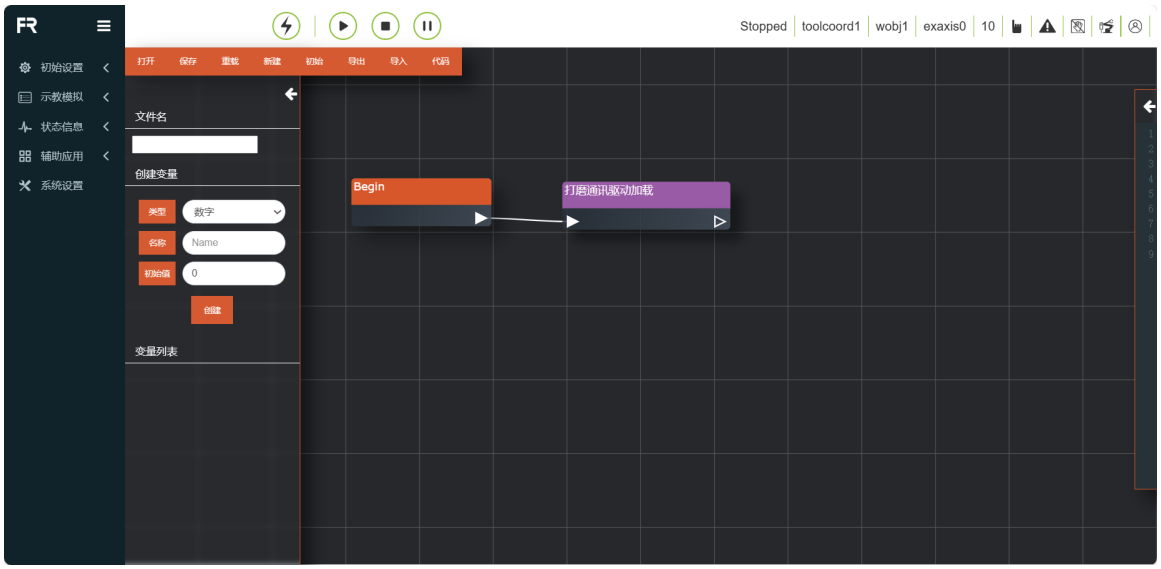
图表 6.2.62 “跟踪关闭”指令节点界面

1.3.6.2.5.6 打磨指令

该指令用于打磨场景的使用，使用时需要先卸载驱动再加载驱动，然后设置打磨设备使能。进而设置打磨设备的转速、接触力、伸出距离和控制模式，同时可以对打磨设备错误清除和设备力传感器清零。



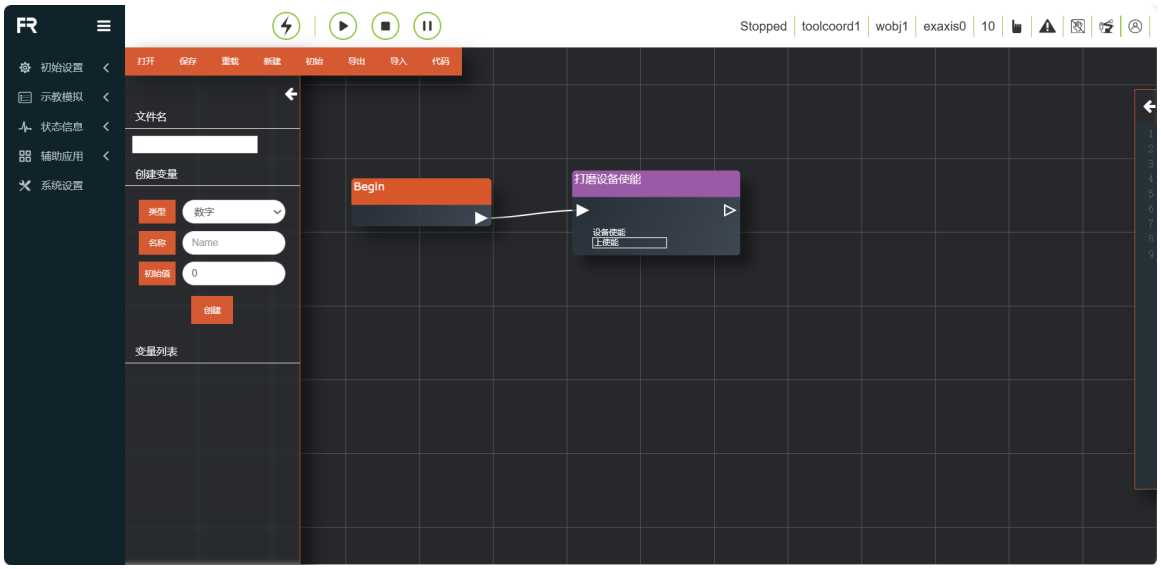
图表 6.2.63 “通讯驱动卸载”指令节点界面



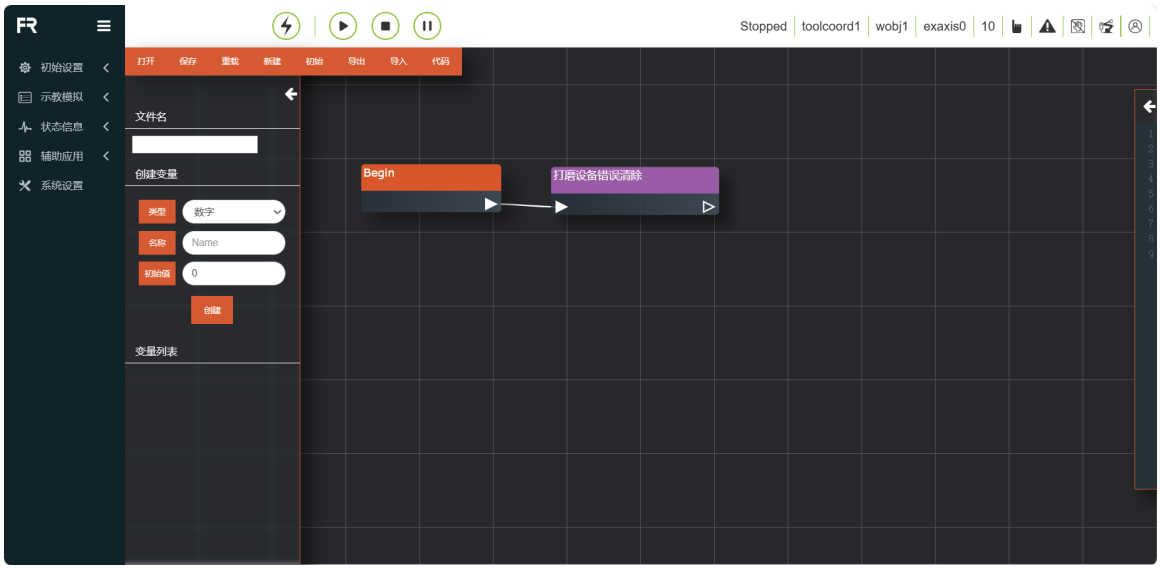
图表 6.2.64 “通讯驱动加载”指令节点界面

“设备使能”指令节点, 参数:

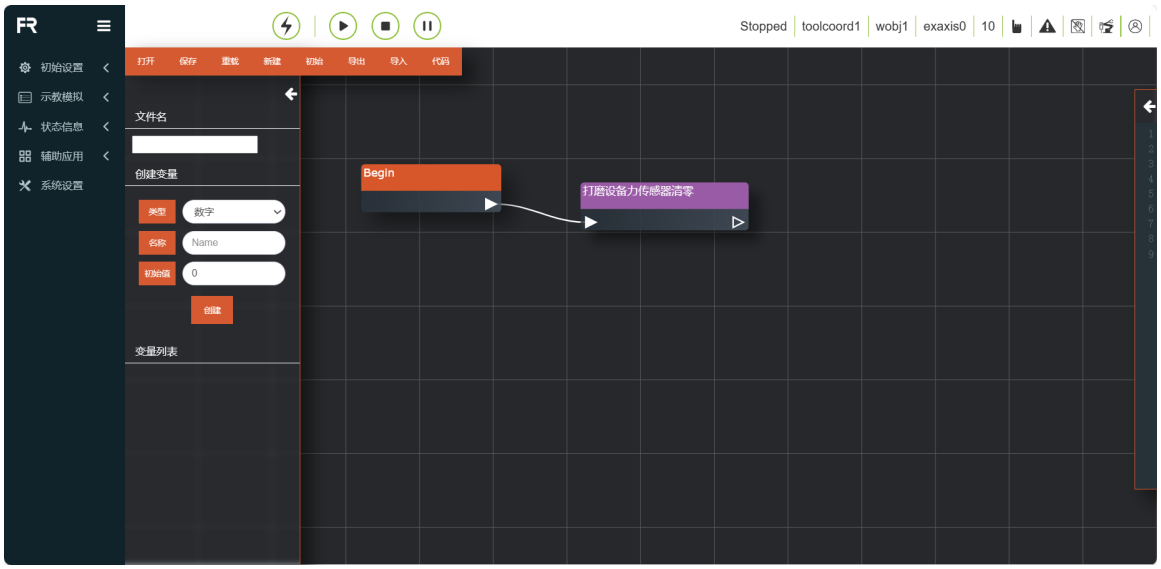
- 设备使能：上使能/下使能



图表 6.2.65 “设备使能”指令节点界面



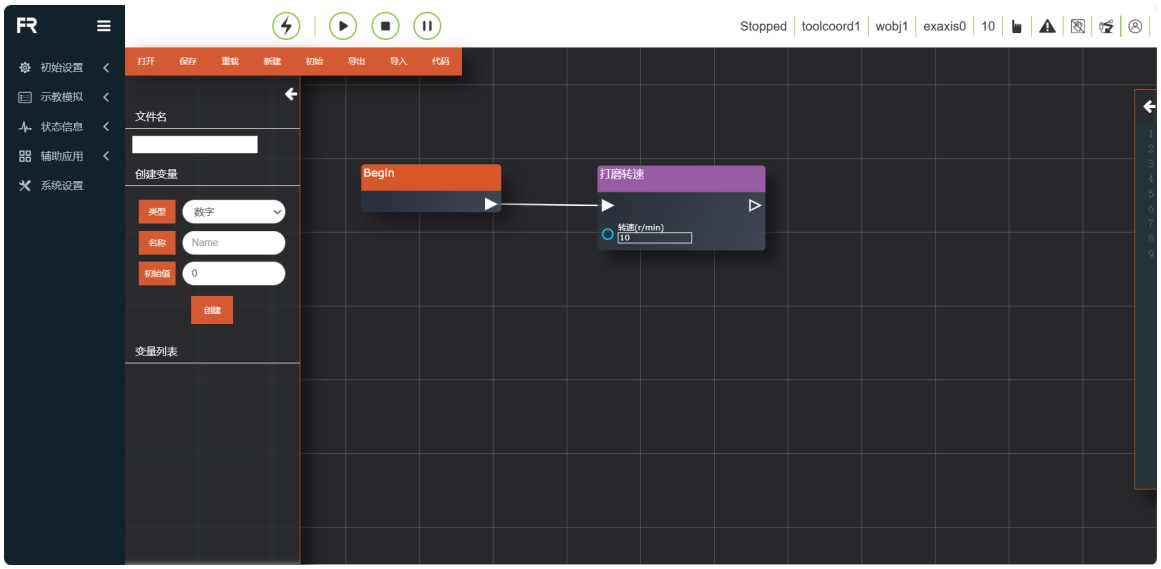
图表 6.2.66 “设备错误清除”指令节点界面



图表 6.2.67 “设备力传感器清零”指令节点界面

“转速”指令节点, 参数:

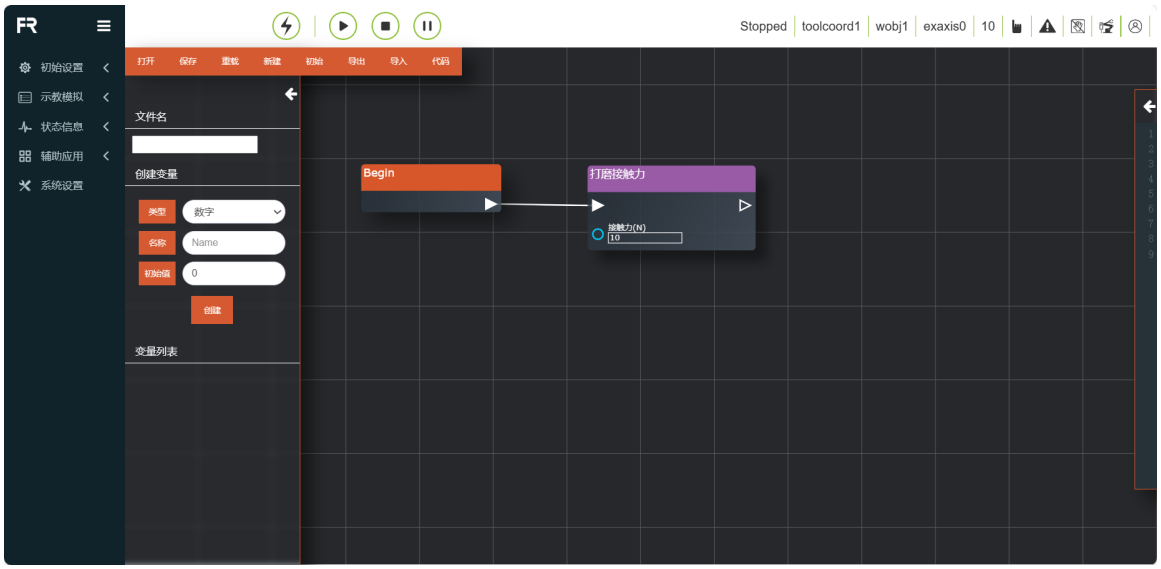
- 转速：0~5500



图表 6.2.68 “转速”指令节点界面

“接触力”指令节点, 参数:

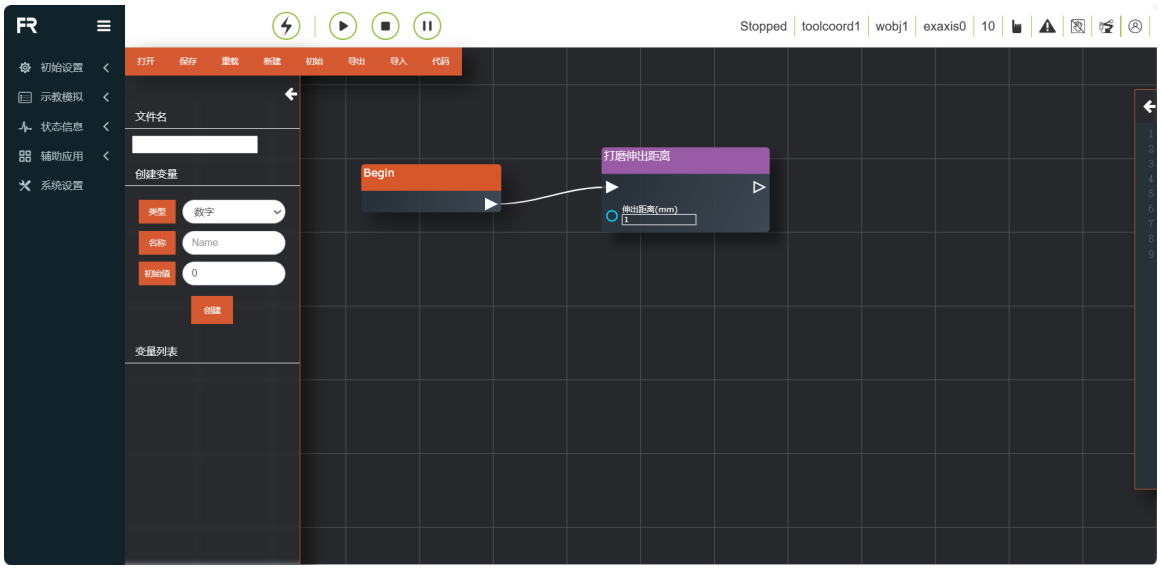
- 接触力：0~200



图表 6.2.69 “接触力”指令节点界面

“伸出距离”指令节点, 参数:

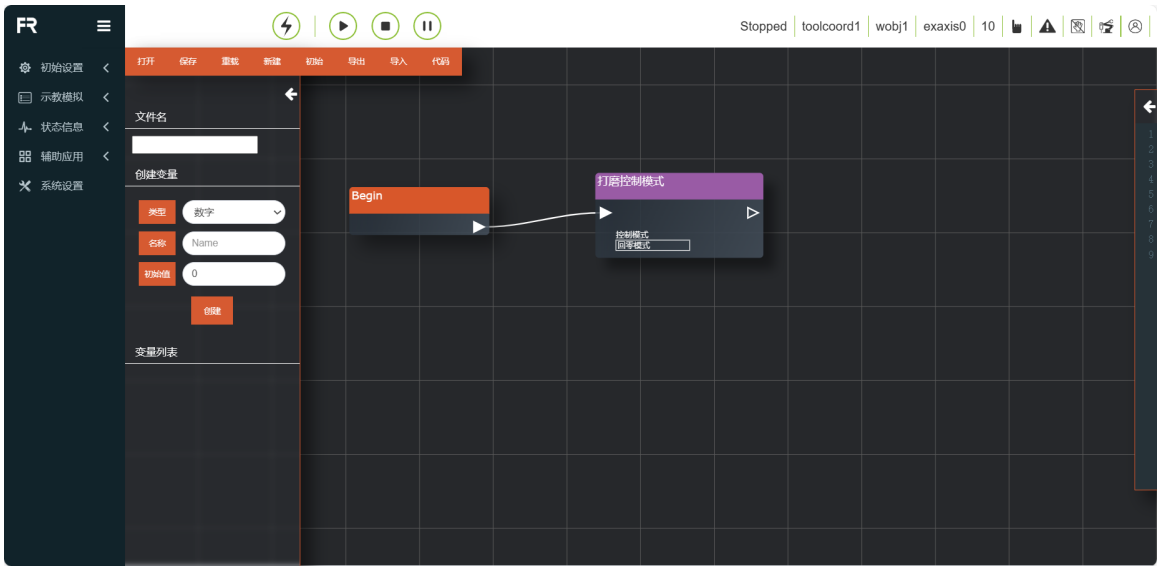
- 伸出距离：0~12



图表 6.2.70 “伸出距离”指令节点界面

“控制模式”指令节点, 参数:

- 控制模式：回零模式/位置模式/力矩模式



图表 6.2.71 “控制模式”指令节点界面

1.3.6.2.6 “焊接”指令界面

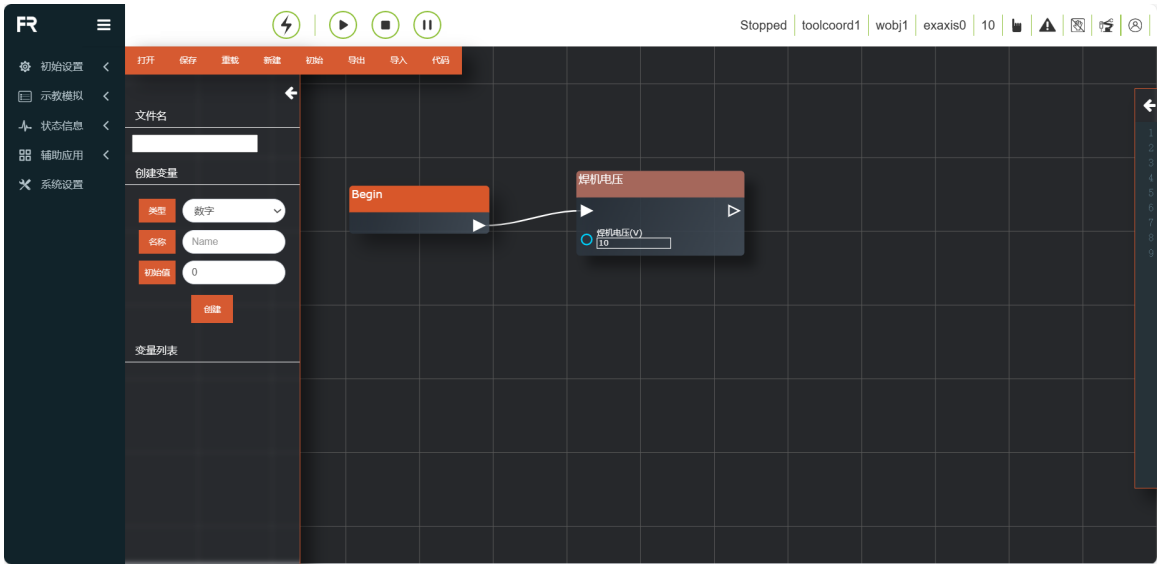
1.3.6.2.6.1 焊接指令

点击“焊接相关指令节点，进入节点图编程界面

该指令主要用于焊机外设，在添加该指令前请确认在用户外设中焊机配置是否完成，详见机器人外设章节。

1. “焊机电压”指令节点, 参数:

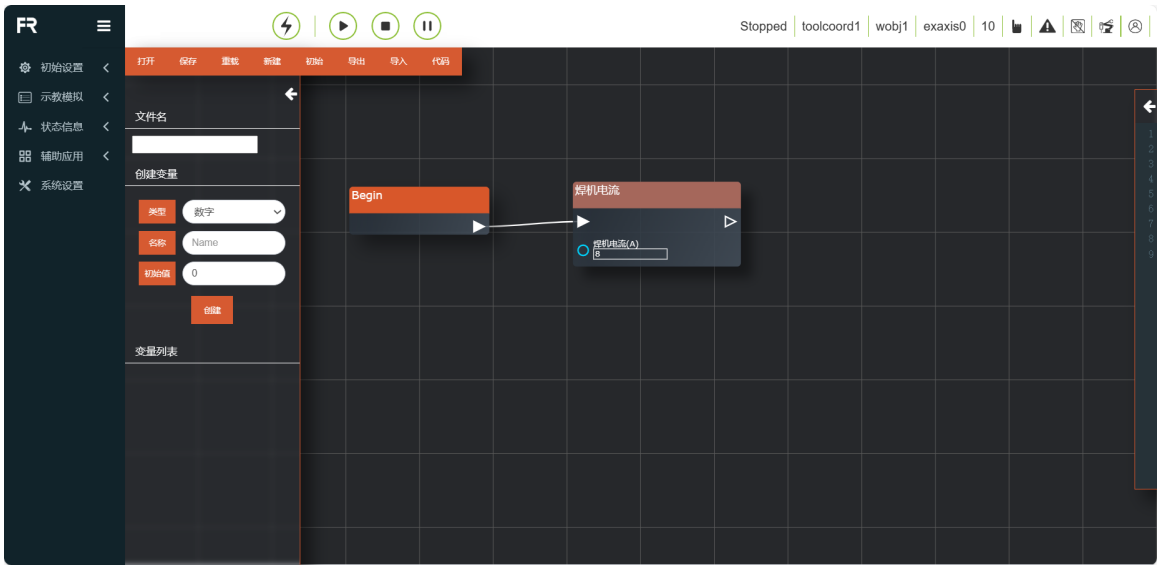
   - 焊机电压：最小值为 0



图表 6.2.72 “焊机电压”指令节点界面

2. “焊机电流”指令节点, 参数:

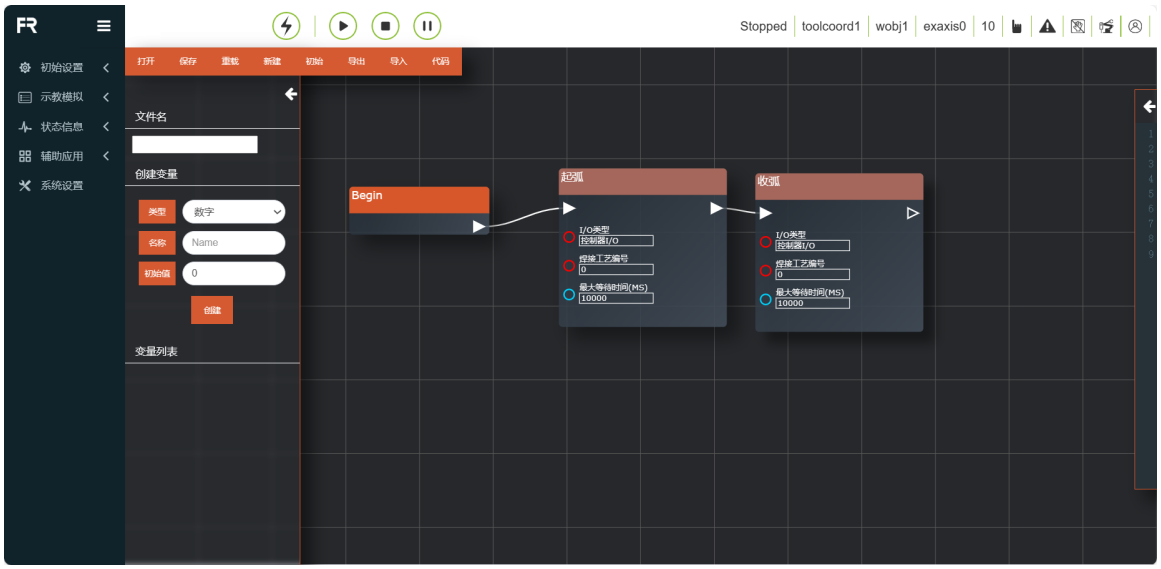
   - 焊机电流：最小值为 0



图表 6.2.73 “焊机电流”指令节点界面

3. “收弧/起弧”指令节点, 参数:

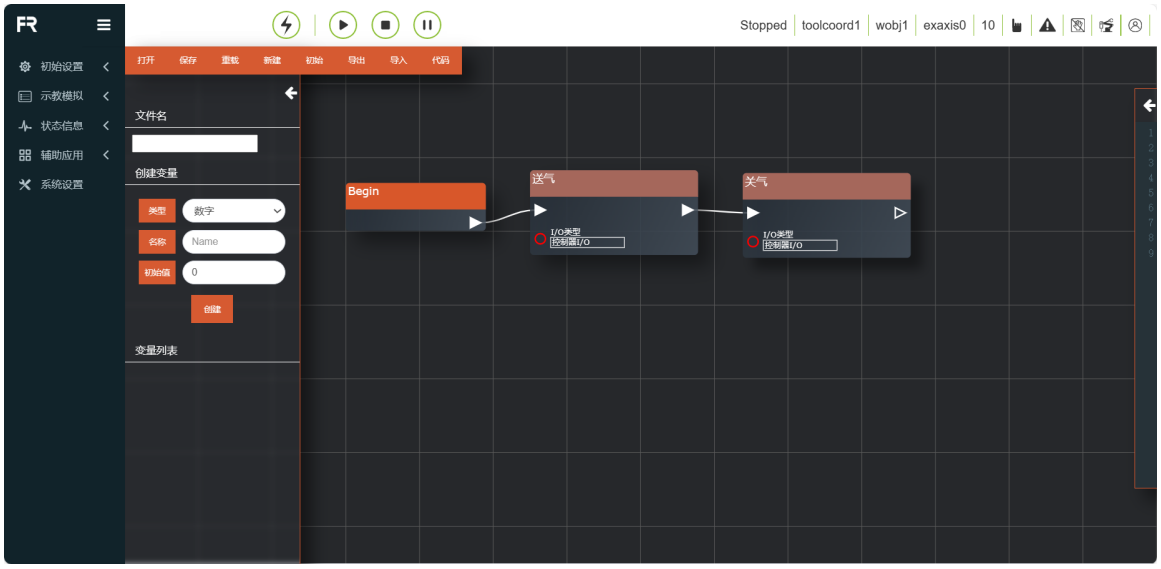
   - I/O 类型：控制器 IO/扩展 IO
   - 焊接工艺编号：0 ~ 7
   - 最大等待时间 (ms): 0 ~ 10000



图表 6.2.74 “收弧/起弧”指令节点界面

4. “送气/关气”指令节点, 参数:

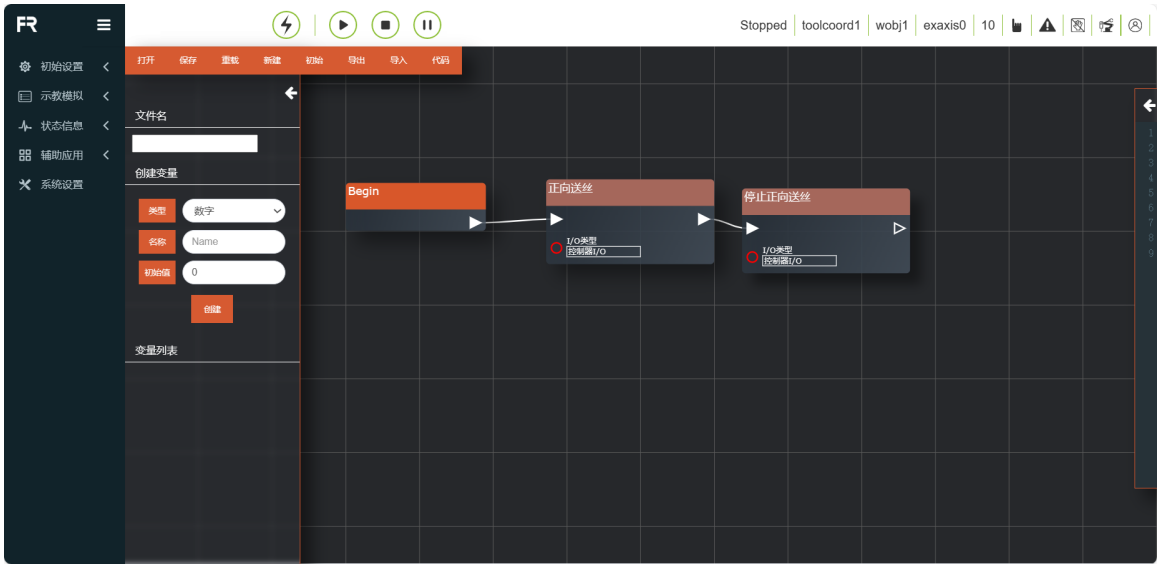
   - I/O 类型：控制器 IO/扩展 IO



图表 6.2.75 “送气/关气”指令节点界面

5. “正向送丝/停止正向送丝”指令节点, 参数:

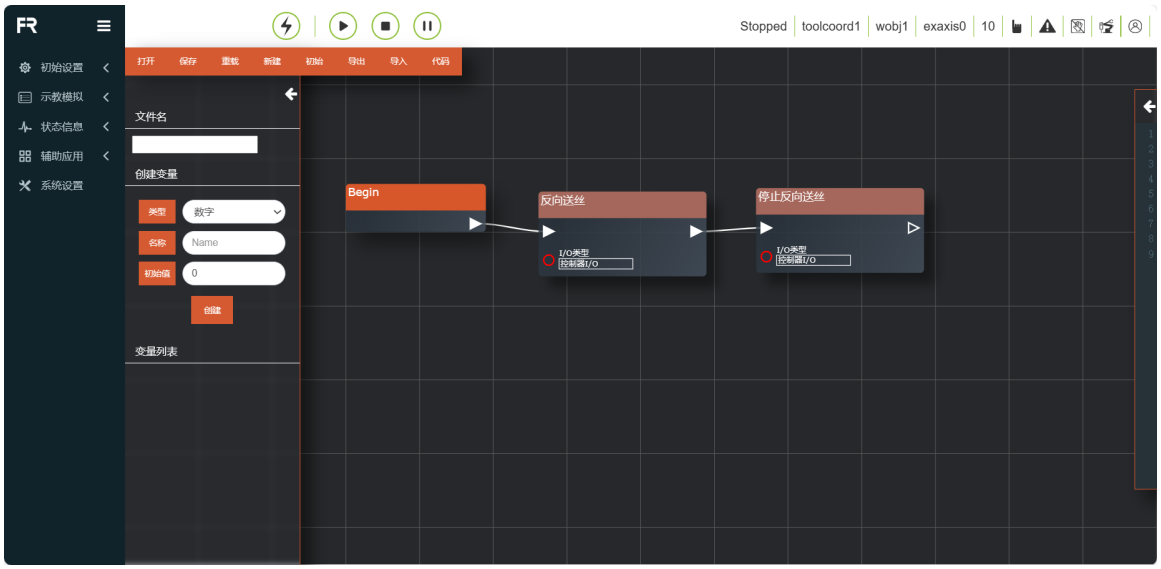
- I/O 类型：控制器 IO/扩展 IO



图表 6.2.76 “正向送丝/停止正向送丝”指令节点界面

5. “反向送丝/停止反向送丝”指令节点, 参数:

- I/O 类型：控制器 IO/扩展 IO



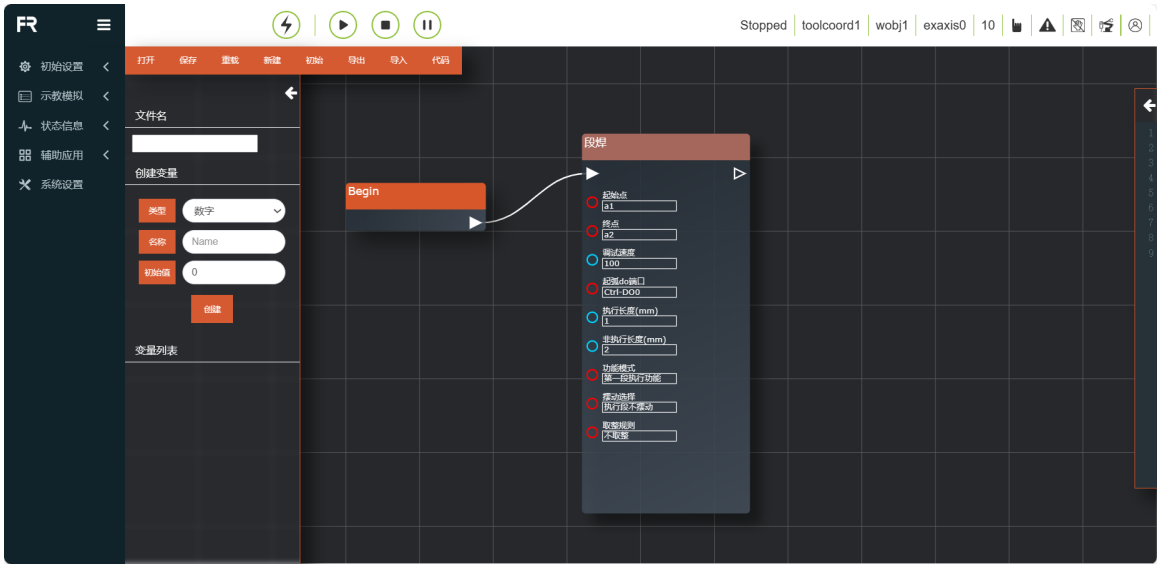
图表 6.2.77 “反向送丝/停止反向送丝”指令节点界面

1.3.6.2.6.2 段焊指令

该指令为焊接专用指令，主要用于一段焊，一段不焊的循环间断焊接场景。在起点与终点之间，使用该指令，选择起点和终点，设置调试速度，设置起弧的 DO 端口，执行长度，非执行长度，根据实际应用场景设置功能模式，摆动选择和取整规则即可实现段焊功能。

“段焊”指令节点, 参数:

- 起始点：示教点位
- 终点：示教点位
- 调试速度 (%)：0~100，默认为 100
- 起弧 do 端口：Ctrl-DO0~7，Ctrl-CO0~7
- 执行长度：0~1000
- 非执行长度：0~1000
- 功能模式：0~100，默认为 100
- 摆动选择：执行段不摆动/执行段摆动
- 取整规则：不取整/循环取整/单段取整



图表 6.2.78 “段焊”指令节点界面

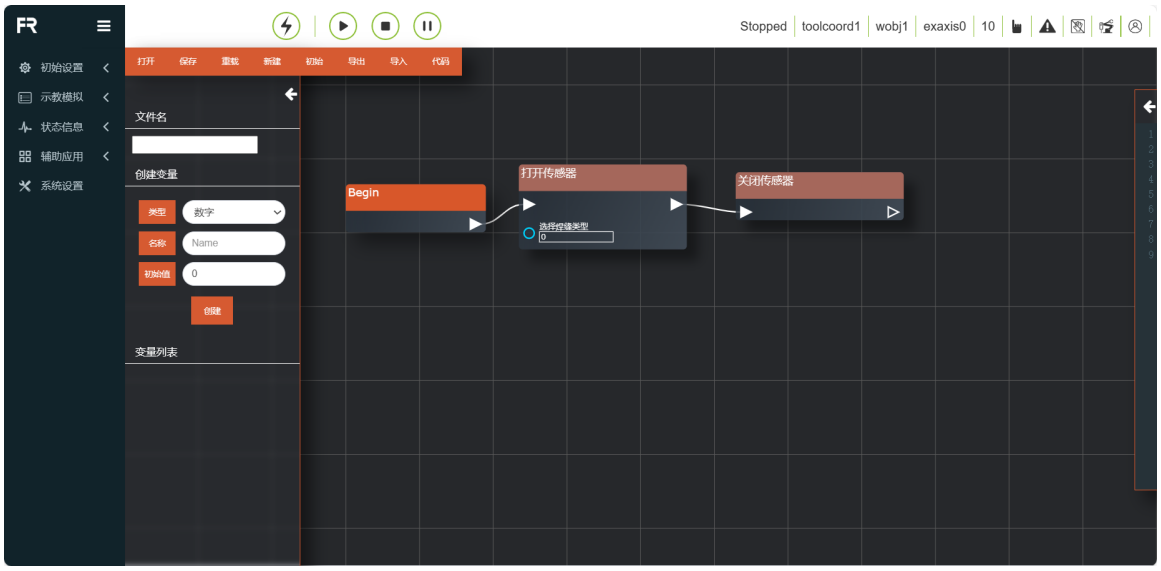
1.3.6.2.6.3 激光跟踪指令

点击“激光跟踪”指令节点，进入节点图编程界面

该指令包含激光命令、跟踪命令和寻位命令三部分，在添加该指令前，请确认用户外设中激光跟踪传感器是否已经配置成功。详见机器人外设章节。

1. “打开/关闭传感器”指令节点，参数

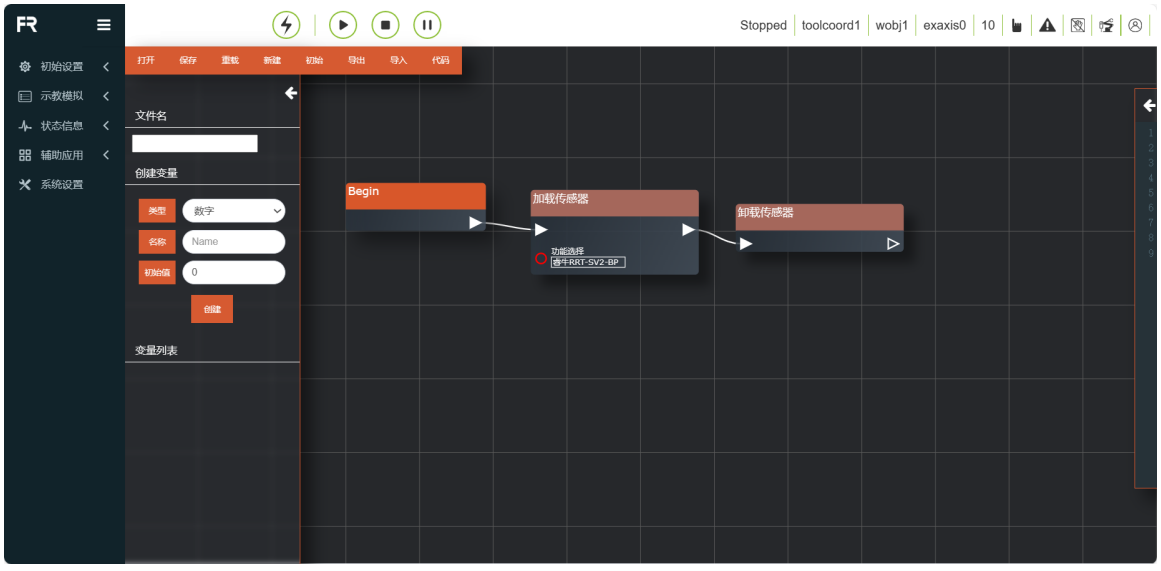
   - 选择焊缝类型：0 ~ 49



图表 6.2.79 “打开/关闭传感器”指令节点界面

2. “加载/卸载传感器”指令节点，参数

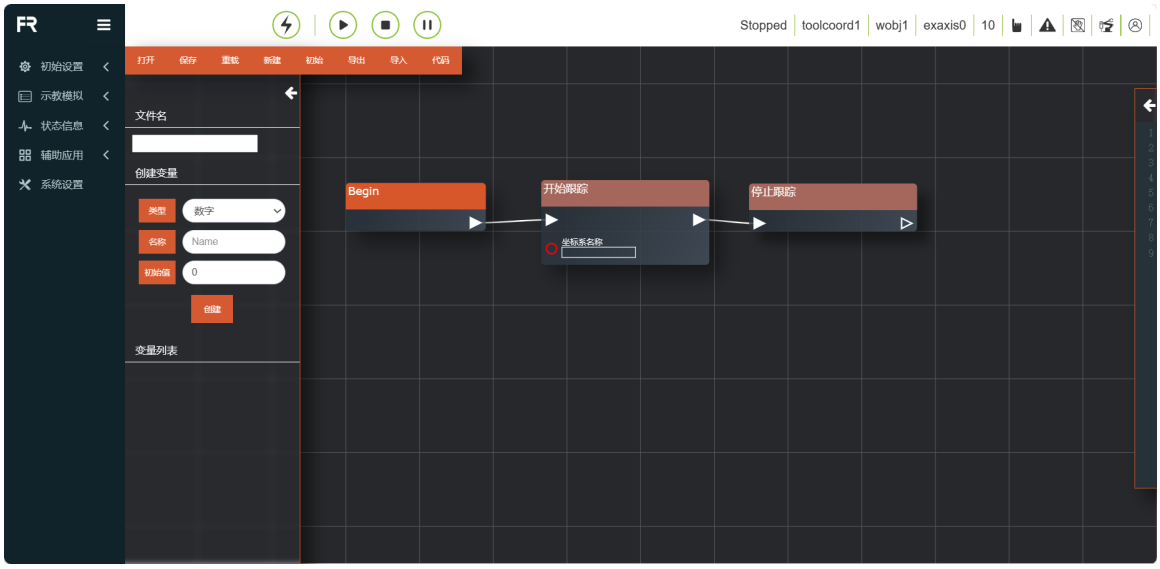
   - 功能选择：睿牛 RRT-SV2-BP/创想 CXZK-RBTA4L



图表 6.2.80 “加载/卸载传感器”指令节点界面

3. “开始/停止跟踪”指令节点，参数

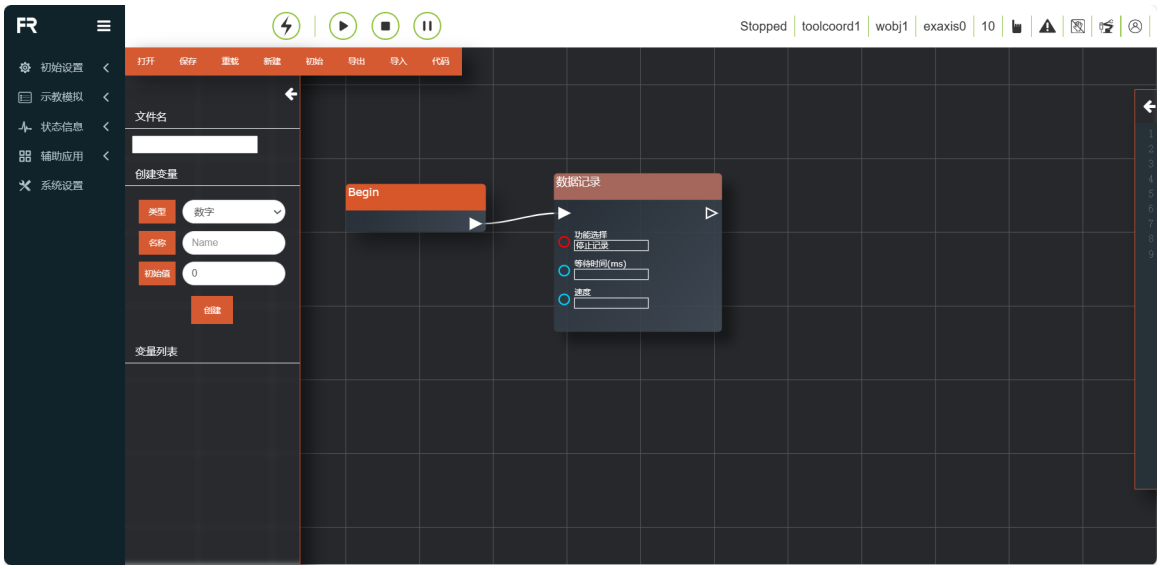
   - 坐标系名称：自定义配置坐标系



图表 6.2.81 “开始/停止跟踪”指令节点界面

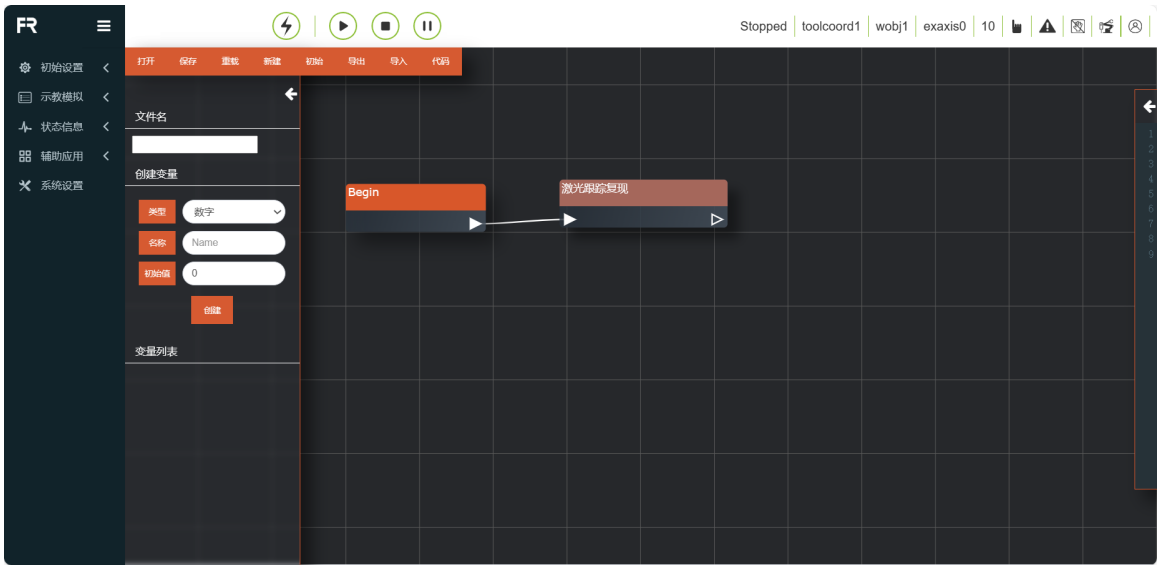
4. “数据记录”指令节点，参数

   - 功能选择：停止记录/实时跟踪/开始记录/轨迹复现
   - 等待时间 (ms)：0 ~ 10000



图表 6.2.82 “数据记录”指令节点界面

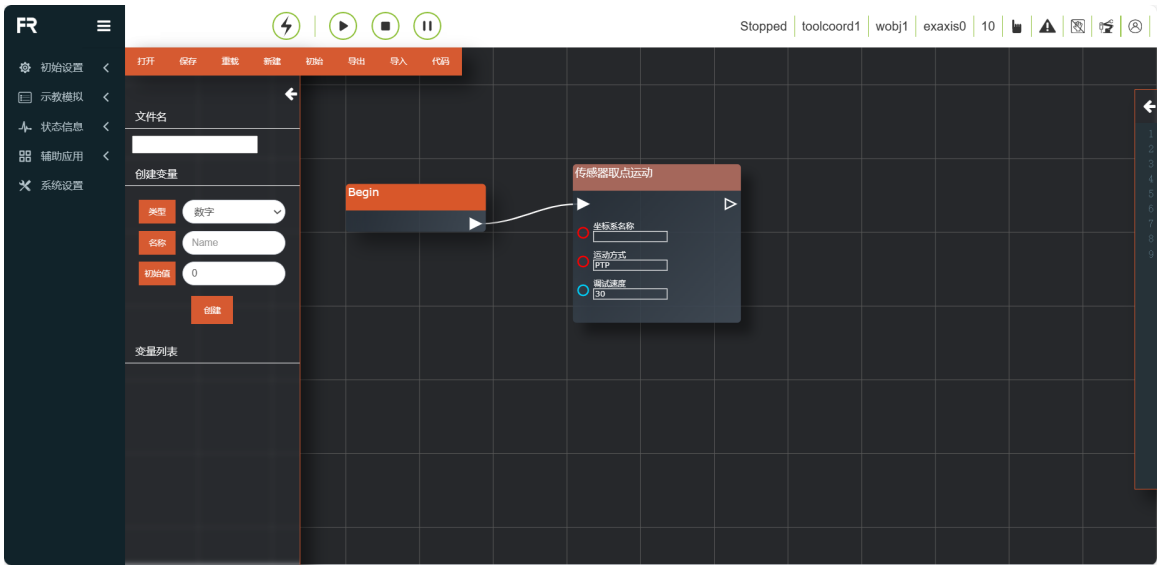
5. “激光跟踪复现”指令节点，参数



图表 6.2.83 “激光跟踪复现”指令节点界面

6. “传感器取点运动”指令节点，参数

- 坐标系名称：自定义配置坐标系
- 运动方式：PTP/Lin
- 调试速度 (%)：0 ~ 100

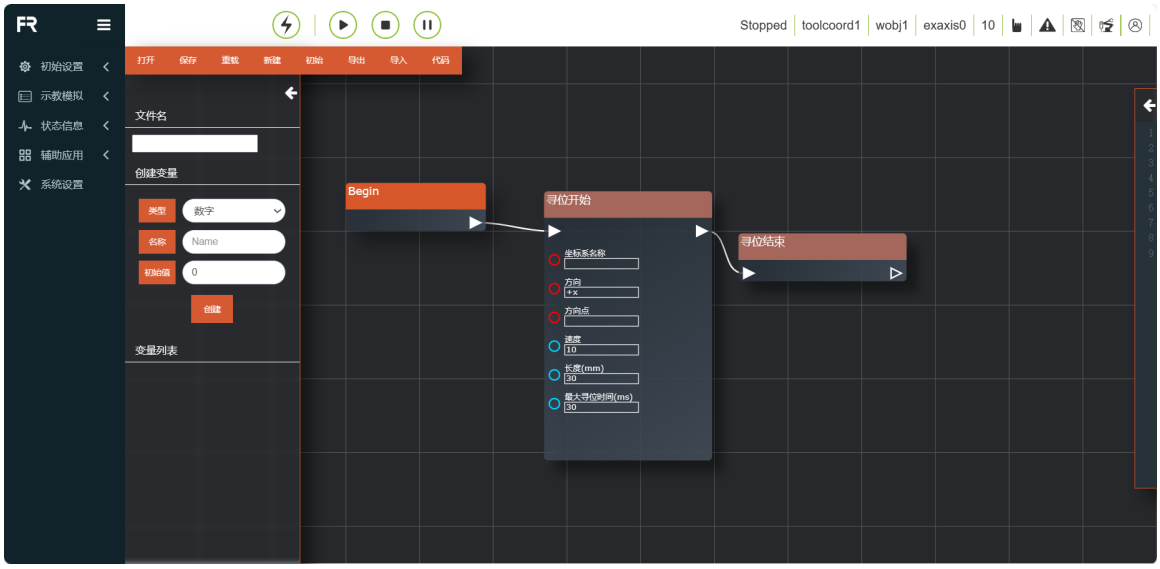


图表 6.2.84 “传感器取点运动”指令节点界面

7. “寻位开始/结束”指令节点，参数

- 坐标系名称：自定义配置坐标系
- 方向：+x/-x/+y/-y/+z/-z/指定方向
- 方向点：未选择“指定方向”时，参数失效

- 速度 (%): 0 ~ 100
- 长度 (mm): 0 ~ 1000
- 最大寻位时间 (ms): 0 ~ 10000



图表 6.2.85 “寻位开始/结束”指令节点界面

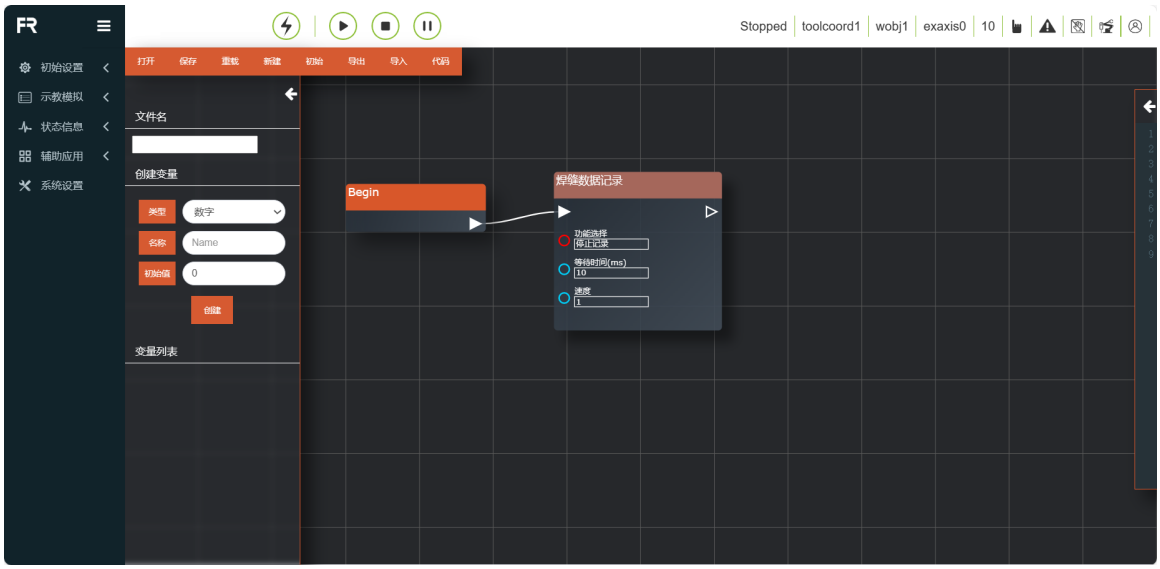
1.3.6.2.6.4 激光记录指令

该指令实现激光跟踪记录起点、终点取出功能，使机器人可以自动运动到起点位置，适用于从工件外部开始运动并进行激光跟踪记录的场合，同时上位机可获取记录数据中起点、终点的信息，用于后续运动。

实现激光跟踪复现速度可调功能，使机器人可以用一个很快的速度进行记录，然后按照正常焊接速度进行复现，可以提高作业效率。

“焊缝数据记录”指令节点, 参数:

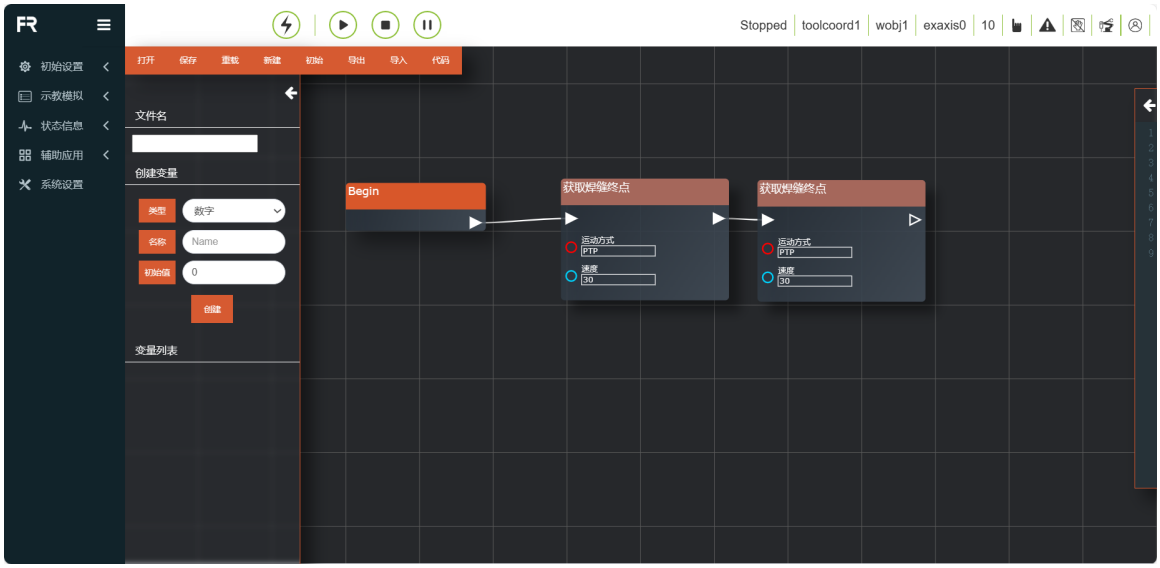
- 功能选择：停止记录/实时跟踪/开始记录/轨迹复现
- 等待时间 (ms)：0~10000，默认为 10
- 速度 (%)：0~100，默认为 30，选择轨迹复现时，该参数生效



图表 6.2.86 “焊缝数据记录”指令节点界面

“获取焊缝起点/终点”指令节点, 参数:

- 运动方式：PTP/LIN
- 速度 (%)：0~100，默认为 30



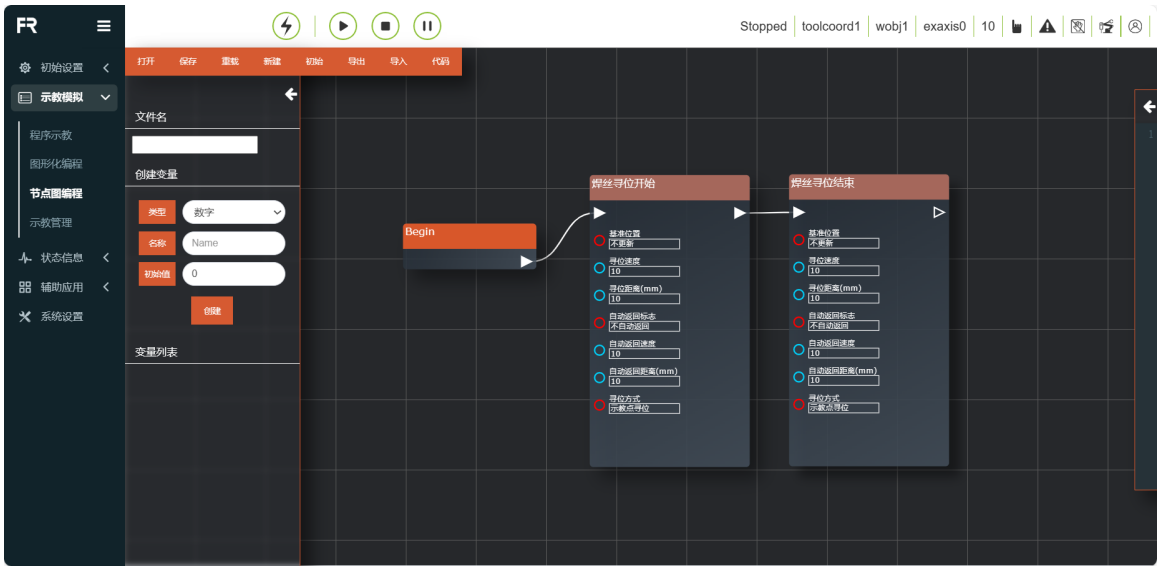
图表 6.2.87 “获取焊缝起点/终点”指令节点界面

1.3.6.2.6.5 焊丝寻位指令

该指令一般应用于焊接场景中，需要焊机与机器人 IO 和运动指令相结合使用。分为寻位开始、寻位结束、寻位点设置、计算偏移量和接触点数据写入。

“焊丝寻位开始/结束”指令节点, 参数：

- 基准位置：不更新/更新
- 寻位速度：0~100
- 寻位距离：0~1000
- 自动返回标志：不自动返回/自动返回
- 自动返回速度：0~100
- 自动返回距离：0~1000
- 寻位方式：示教点寻位/带偏移量寻位

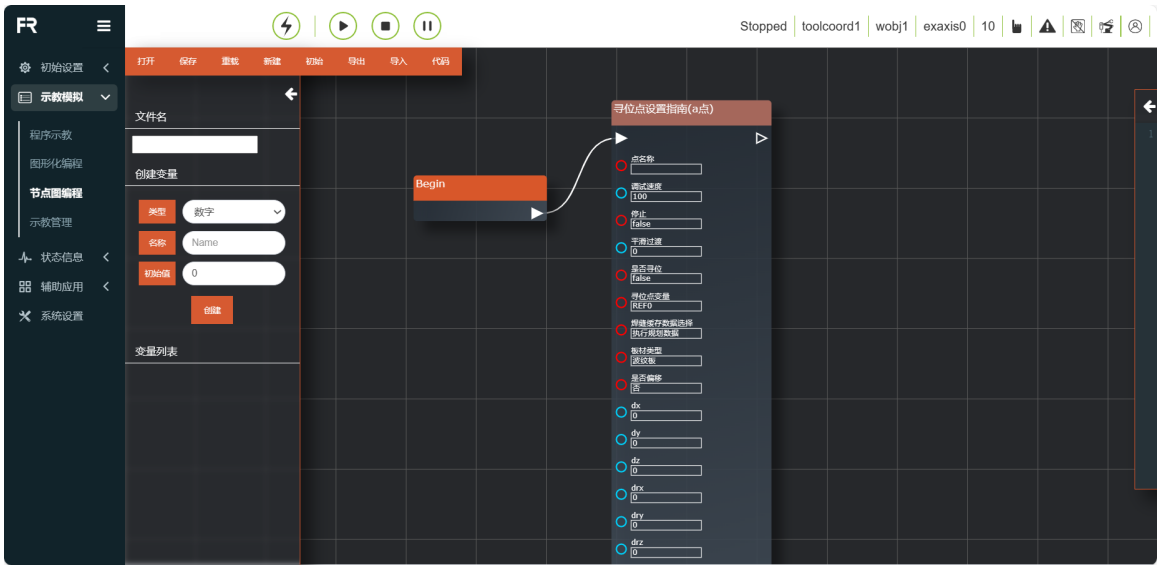


图表 6.2.88 “焊丝寻位开始/结束”指令节点界面

寻位点设置根据焊缝类型和计算方法添加点位。

- 当类型为角焊缝，计算方法为 1D（xyz 中的一个）时，点位添加从 a 点、b 点中选择；
- 当类型为角焊缝，计算方法为 2D（xyz 中的两个）时，点位添加从 a 点、b 点、e 点、f 点中选择；
- 当类型为角焊缝，计算方法为 3D（xyz）时，点位添加从 a 点、b 点、c 点、d 点、e 点、f 点中选择；
- 当类型为角焊缝，计算方法为 2D+（xyz 中的两个，rxryrz 中的一个）时，点位添加从 a 点、b 点、c 点、d 点、e 点、f 点中选择；
- 当类型为内外径，计算方法为 2D2D（xyz 中的两个）时，点位添加从 a 点、b 点中选择；
- 当类型为点，计算方法为 3D（xyz）时，点位添加从 a 点、b 点、c 点、d 点、e 点、f 点中选择；

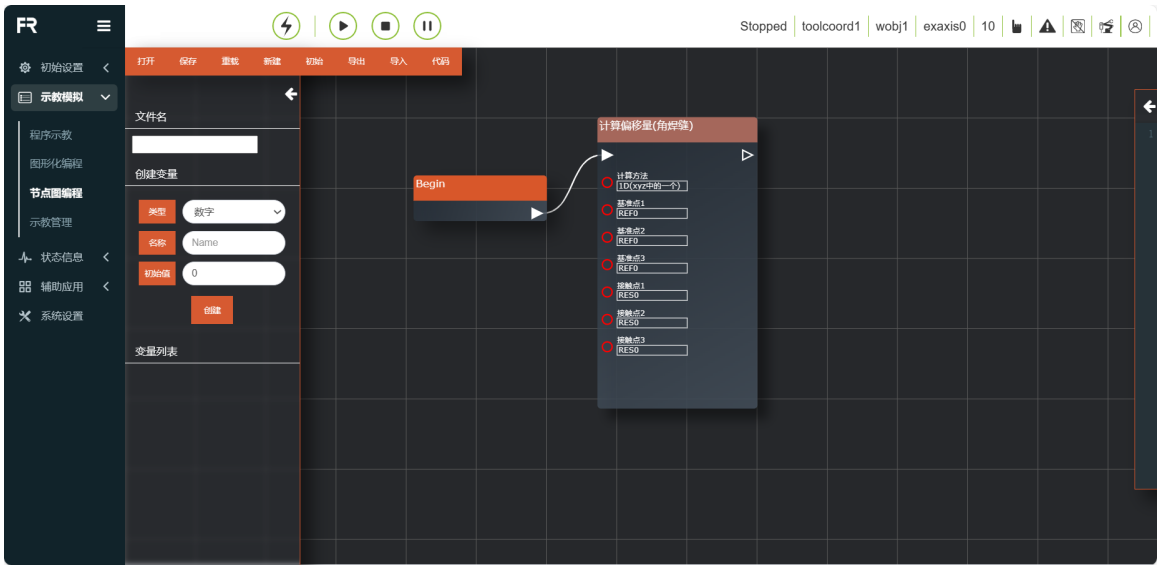
- 当类型为相机，计算方法为 3D+（xyzrxryrz）时，点位添加从 a 点、b 点中选择；
- 当类型为面，计算方法为 3D+（xyzrxryrz）时，点位添加从 a 点、b 点中选择；



图表 6.2.89 “寻位点设置”指令节点界面

计算偏移量根据焊缝类型和计算方法设置基准点和接触点。

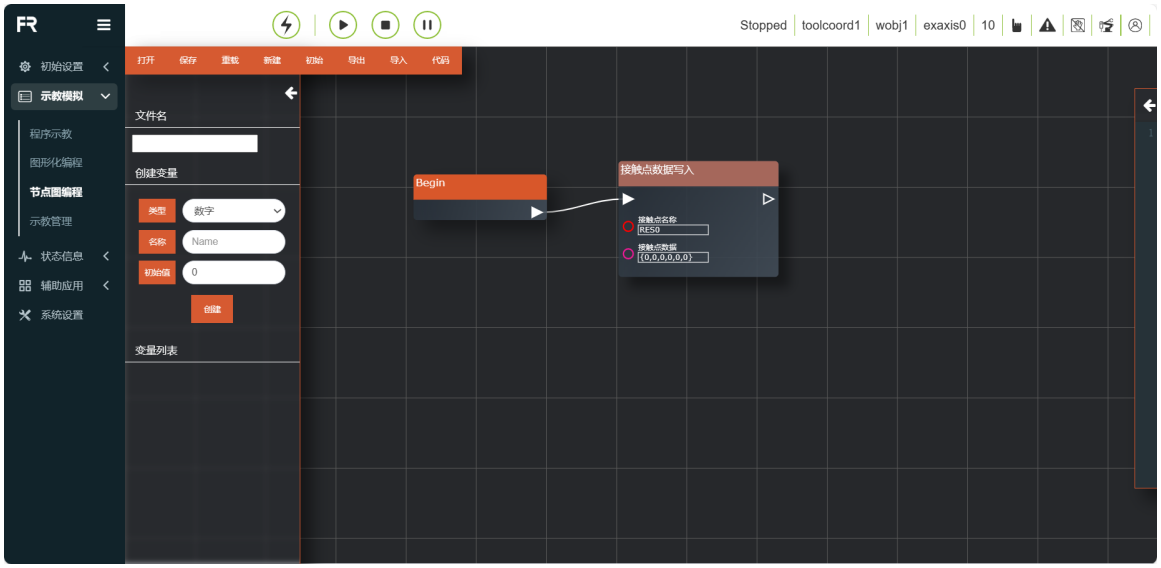
- 当类型为角焊缝，计算方法为 1D（xyz 中的一个）时，设置基准点 1、接触点 1；
- 当类型为角焊缝，计算方法为 2D（xyz 中的两个）时，设置基准点 1、基准点 2、接触点 1、接触点 2；
- 当类型为角焊缝，计算方法为 3D（xyz）时，设置基准点 1、基准点 2、基准点 3、接触点 1、接触点 2、接触点 3；
- 当类型为角焊缝，计算方法为 2D+（xyz 中的两个，rxryrz 中的一个）时，设置基准点 1、基准点 2、基准点 3、接触点 1、接触点 2、接触点 3；
- 当类型为内外径，计算方法为 2D2D（xyz 中的两个）时，设置基准点 1、基准点 2、基准点 3、接触点 1、接触点 2、接触点 3；
- 当类型为点，计算方法为 3D（xyz）时，设置接触点 1、接触点 2；
- 当类型为相机，计算方法为 3D+（xyzrxryrz）时，设置接触点 1、接触点 2；
- 当类型为面，计算方法为 3D+（xyzrxryrz）时，设置接触点 1、接触点 2、接触点 3、接触点 4、接触点 5、接触点 6；



图表 6.2.90 “计算偏移量”指令节点界面

“接触点数据写入”指令节点, 参数:

- 接触点名称: RES0~99
- 接触点名称：数据格式为 {0,0,0,0,0,0};



图表 6.2.91 “接触点数据写入”指令节点界面

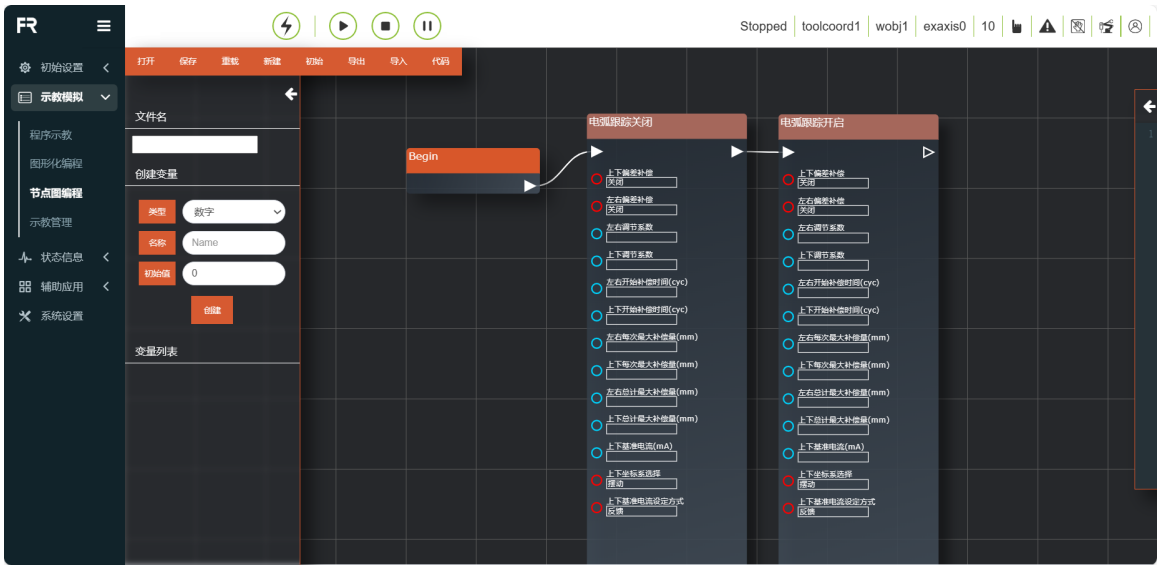
1.3.6.2.6.6 电弧跟踪指令

点击“电弧跟踪”指令节点，进入节点图编程界面

该指令实现机器人焊缝跟踪利用焊缝的偏差检测进行补偿轨迹，可以使用电弧传感器来检测焊缝偏差。

“电弧跟踪开启/关闭”指令节点, 参数:

- 电弧跟踪滞后时间 (ms): 参考值 50
- 偏差补偿: 关闭/开启
- 调节系数: 0 ~ 300
- 补偿时间 (cyc): 0 ~ 300
- 每次最大补偿量 (mm): 0 ~ 300
- 总计最大补偿量 (mm): 0 ~ 300
- 上下坐标系选择: 摆动
- 上下基准电流设定方式: 反馈/常数
- 上下基准电流 (A): 0 ~ 300



图表 6.2.92 “电弧跟踪开启/关闭”指令节点界面

1.3.6.2.6.7 姿态调整指令

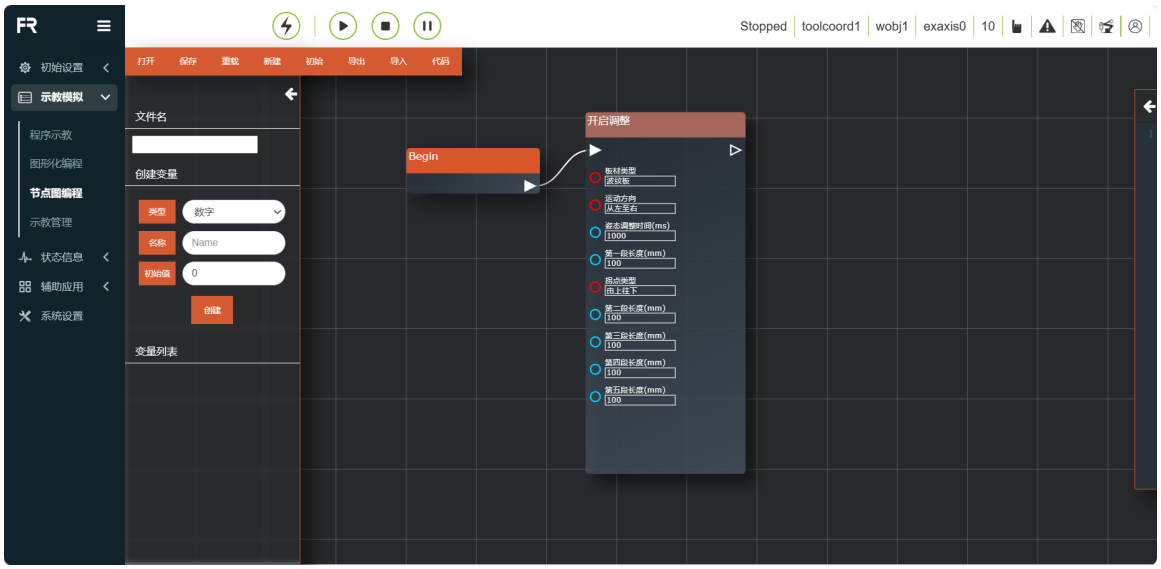
点击“姿态调整”相关指令节点，进入节点图编程界面

该指令针对焊接跟踪自适应调整焊枪姿态场景，需要先示教 PosA、PosB、PosC 三个点位，否则无法添加节点。

记录好三个对应的姿态点后，根据机器人实际运动方向，添加姿态自适应调整指令。详见机器人外设章节。

“开启姿态调整”指令节点, 参数：

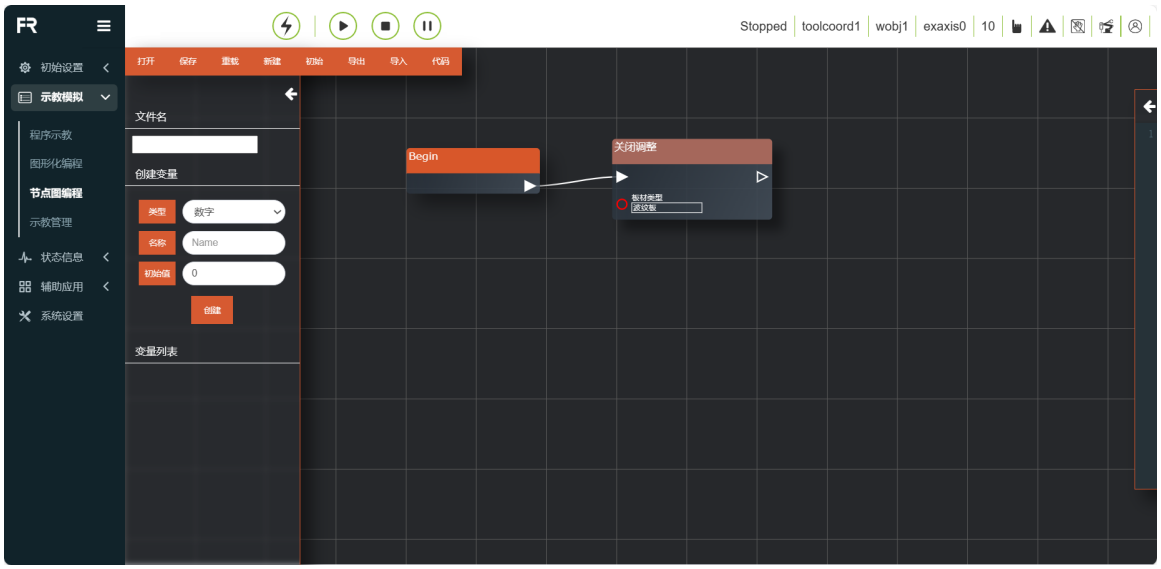
- 板材类型：波纹板/瓦楞板/围栏板/波纹甲壳钢
- 运动方向：从左至右/从右至左
- 姿态调整时间 (ms): 0 ~ 1000
- 第一段长度 (mm):
- 拐点类型：由上往下/由下往上
- 第二段长度 (mm):
- 第三段长度 (mm):
- 第四段长度 (mm):
- 第五段长度 (mm):



图表 6.2.93 “开启姿态调整”指令节点界面

“关闭姿态调整”指令节点, 参数：

- 板材类型：波纹板/瓦楞板/围栏板/波纹甲壳钢



图表 6.2.94 “关闭姿态调整”指令节点界面

1.3.6.2.7 “力控”指令界面

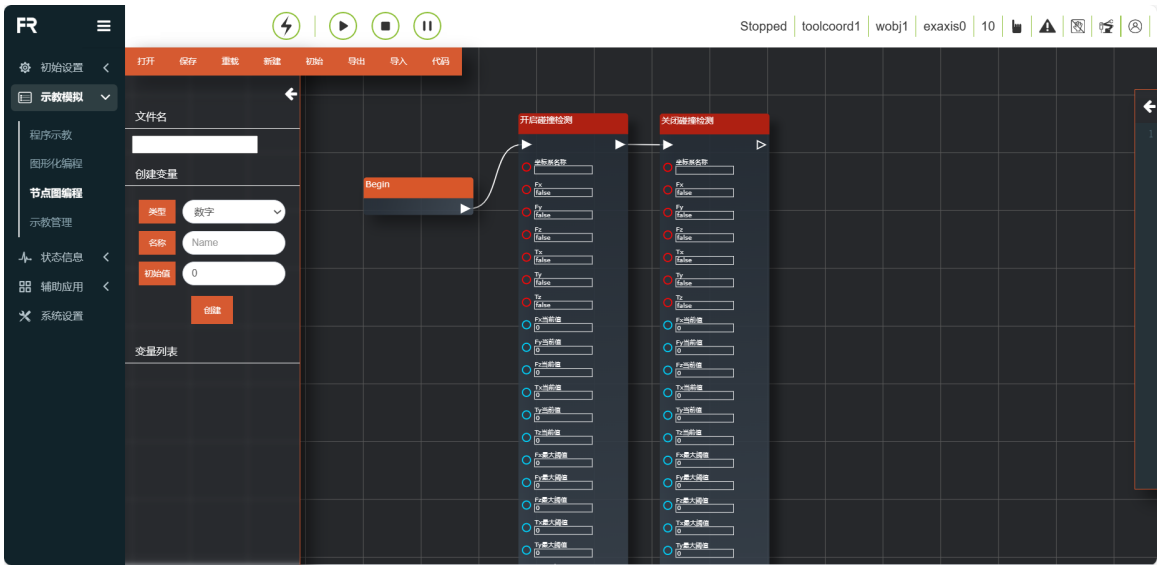
1.3.6.2.7.1 “力控”命令

点击“力控”指令相关指令节点，进入节点图编程界面

该指令包含 FT_Guard(碰撞检测)，FT_Control(恒力控制)，FT_Compliance(柔顺控制)，FT_Spiral(螺旋插入)，FT_Rot(旋转插入)，FT_Lin(直线插入)，FT_FindSurface(表面定位)，FT_CalCenter(中心定位)，FT_Click(点按力探测) 九个指令，详见机器人外设章节。

1. “开启/关闭碰撞检测”指令节点, 参数:

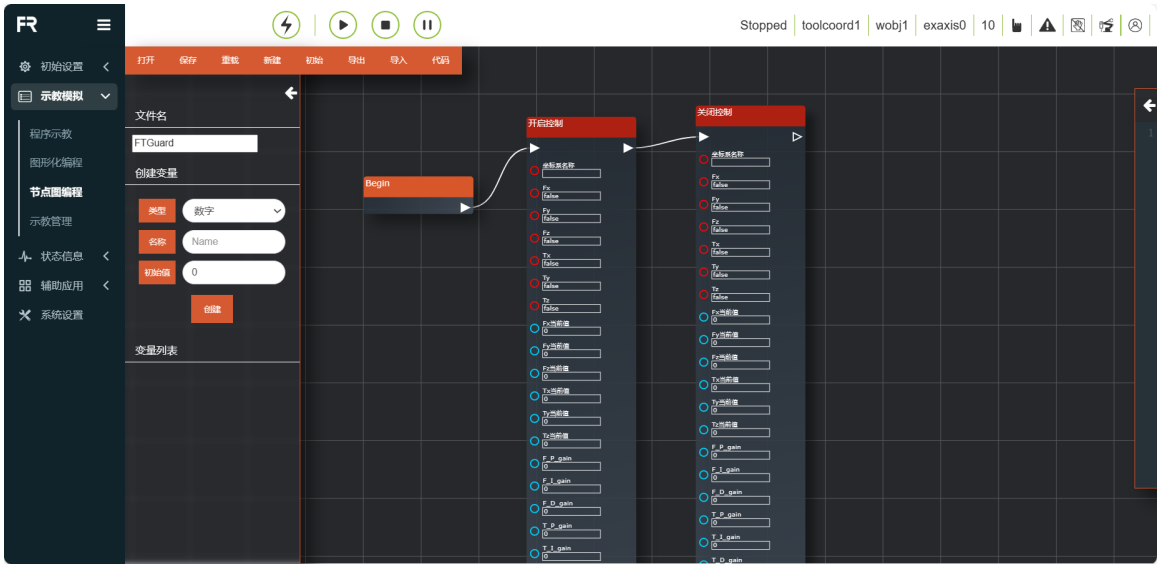
- 坐标系名称：自定义配置的坐标系
- Fx-Tx 真值：true/false
- Fx-Tx 当前值：根据实际情况输入
- Fx-Tx 最大阈值：根据实际情况输入
- Fx-Tx 最小阈值：根据实际情况输入



图表 6.2.95 “开启/关闭碰撞检测”指令节点界面

2. “开启/关闭控制”指令节点, 参数:

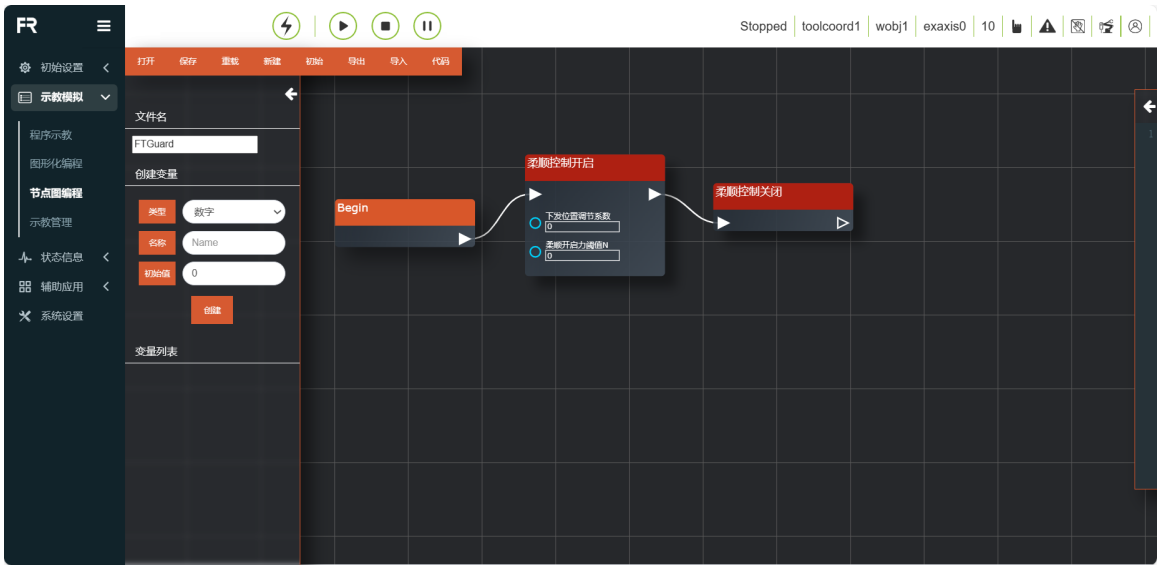
- 坐标系名称：自定义配置的坐标系
- Fx-Tx 真值：true/false
- Fx-Tx 当前值：根据实际情况调整
- F_P_gain - F_D_gain：根据实际情况调整，不能为 0
- 自适应启停状态：停止/开启
- ILC 控制启停状态：停止/训练/实操
- 最大调整距离 (mm): 0 ~ 1000
- 最大调整角度 (°): 0 ~ 1000



图表 6.2.96 “开启/关闭控制”指令节点界面

3. “开启/关闭柔顺控制”指令节点, 参数:

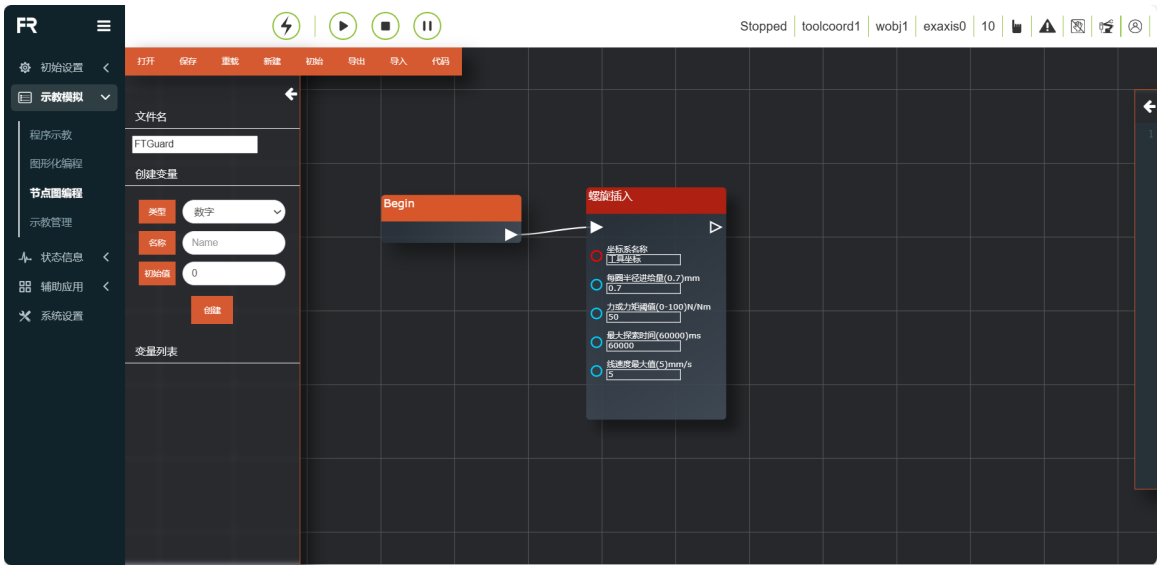
   - 下发位置调节系数: 0 ~ 1
   - 柔顺开启力阈值 (N): 0 ~ 100



图表 6.2.97 “开启/关闭柔顺控制”指令节点界面

4. “螺旋插入”指令节点, 参数:

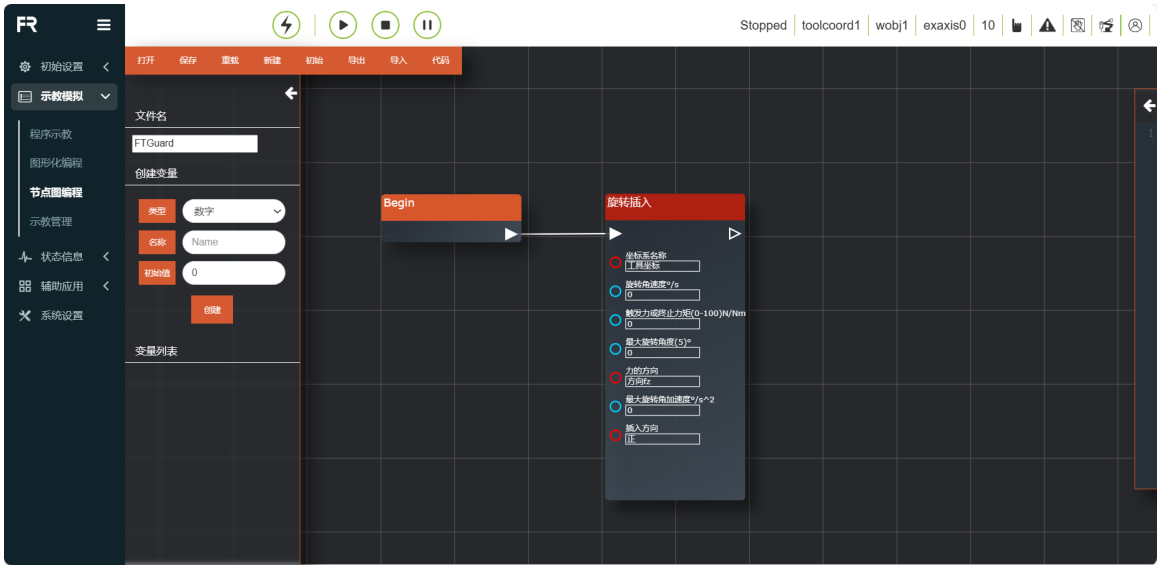
   - 坐标系名称：工具坐标系/基坐标
   - 每圈半径进给量 (mm)：0 ~ 100, 参考值：0.7
   - 力或力矩阈值 (N/Nm)：0 ~ 100, 参考值：50
   - 最大探索时间 (ms)：0 ~ 60000, 参考值：60000
   - 线速度最大值 (mm/s): 0 ~ 100，参考值：5



图表 6.2.98 “螺旋插入”指令节点界面

5. “旋转插入”指令节点, 参数:

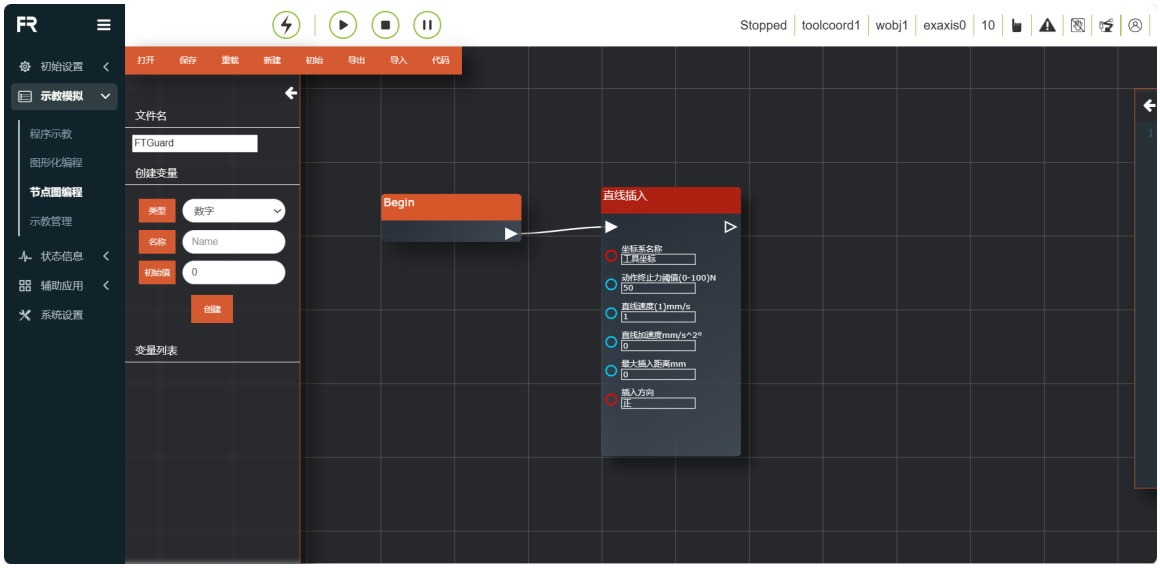
- 坐标系名称：工具坐标系/基坐标
- 旋转角速度 (°/s)：0 ~ 100, 参考值：0.7
- 触发力或终止力矩 (N/Nm)：0 ~ 100, 参考值：50
- 最大旋转角度 (°)：0 ~ 100, 参考值：5
- 力的方向：方向 z/方向 mz
- 最大旋转角加速度 (°/s^2)：0 ~ 100
- 插入方向: 正/负



图表 6.2.99 “旋转插入”指令节点界面

6. “直线插入”指令节点, 参数:

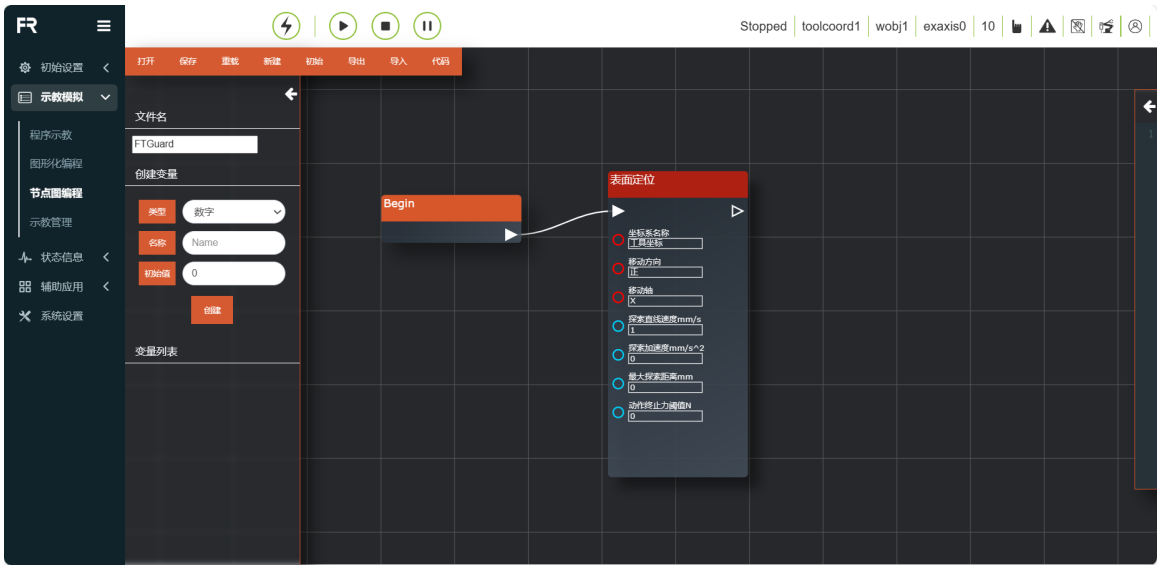
- 坐标系名称：工具坐标系/基坐标
- 动作终止力阈值 (N)：0 ~ 100
- 直线速度 (mm/s)：0 ~ 100, 参考值：1
- 直线加速度 (°/s^2)：0 ~ 100
- 最大插入距离 (mm)：0 ~ 100
- 插入方向: 正/负



图表 6.2.100 “直线插入”指令节点界面

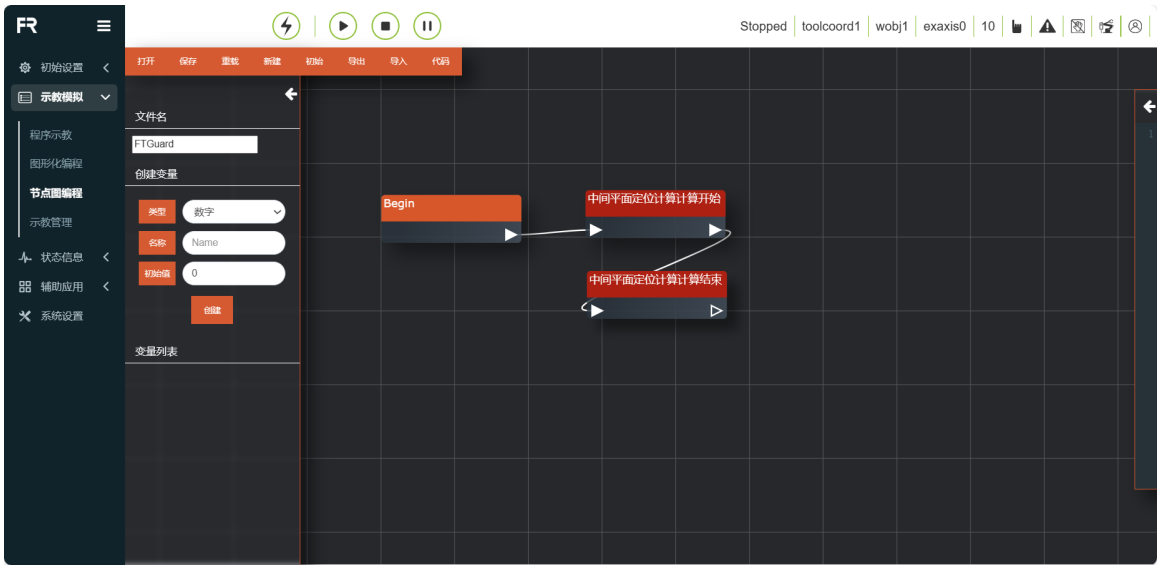
7. “表面定位”指令节点, 参数:

- 坐标系名称：工具坐标系/基坐标
- 移动方向：正/负
- 移动轴：X/Y/Z
- 探索直线速度 (mm/s)：0 ~ 100
- 探索加速度 (mm/s^2)：0 ~ 100
- 最大探索距离 (mm)：0 ~ 100
- 动作终止力阈值 (N)：0 ~ 100



图表 6.2.101 “表面定位”指令节点界面

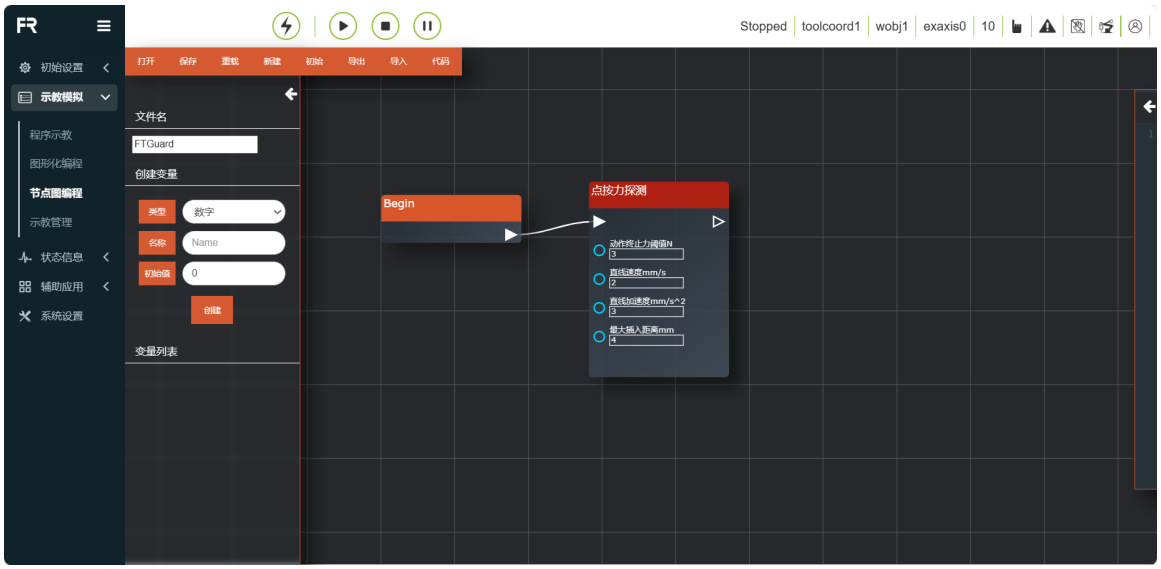
8. “中间平面开始/结束计算”指令节点



图表 6.2.102 “中间平面开始/结束计算”指令节点界面

9. “点按力探测”指令节点, 参数:

- 动作终止力阈值 (N)：0 ~ 100
- 直线速度 (mm/s)：0 ~ 100
- 直线加速度 (mm/s^2)：0 ~ 100
- 最大插入距离 (mm)：0 ~ 100



图表 6.2.103 “点按力探测”指令节点界面

1.3.6.2.7.2 扭矩记录命令

点击“扭矩记录”相关指令节点, 进入节点图编程界面

该指令为扭矩记录指令，包含“扭矩记录开始/“扭矩记录停止”和“扭矩记录复位”三种指令。

实现扭矩实时记录碰撞检测功能。

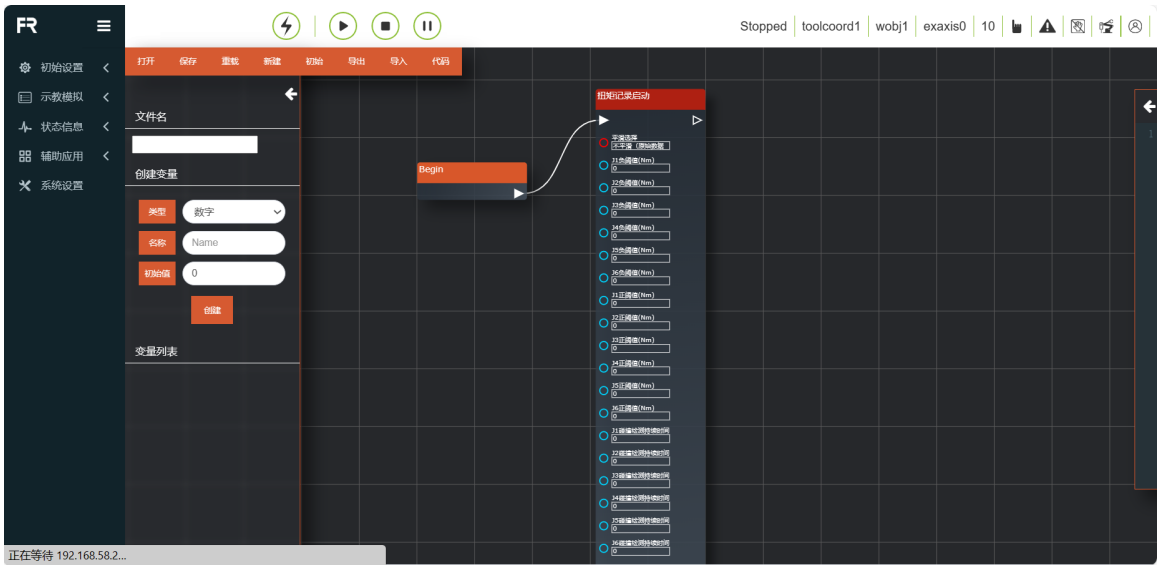
点击“扭矩记录启动”按钮，持续记录运动指令运行过程中的碰撞情况，记录的实时扭矩作为碰撞检测判断的理论值，以减少误报错概率。

当超出设定阈值范围时，记录碰撞检测持续时间。

点击“扭矩记录停止”按钮，停止记录。点击“扭矩记录复位”，状态恢复默认状态。

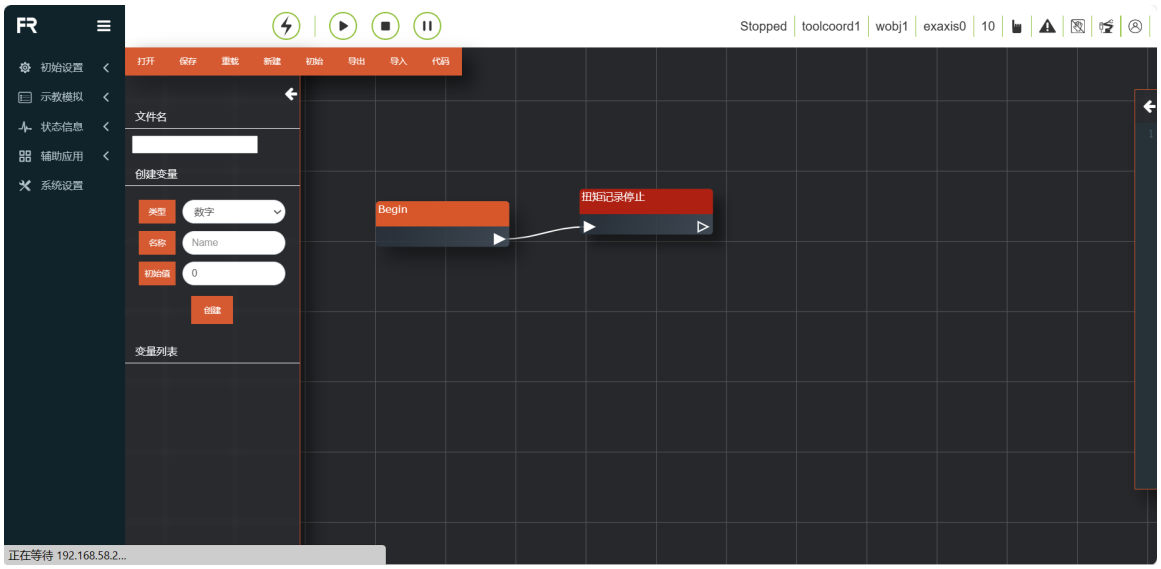
1. “扭矩记录开始”指令节点, 参数:

   - 平滑选择：不平滑 (原始数据)/平滑 (平滑后数据)
   - 关节负阈值 (Nm)：-100 ~ 0
   - 关节正阈值 (Nm)：0 ~ 100
   - 关节持续检测碰撞时间 (ms)：0 ~ 1000



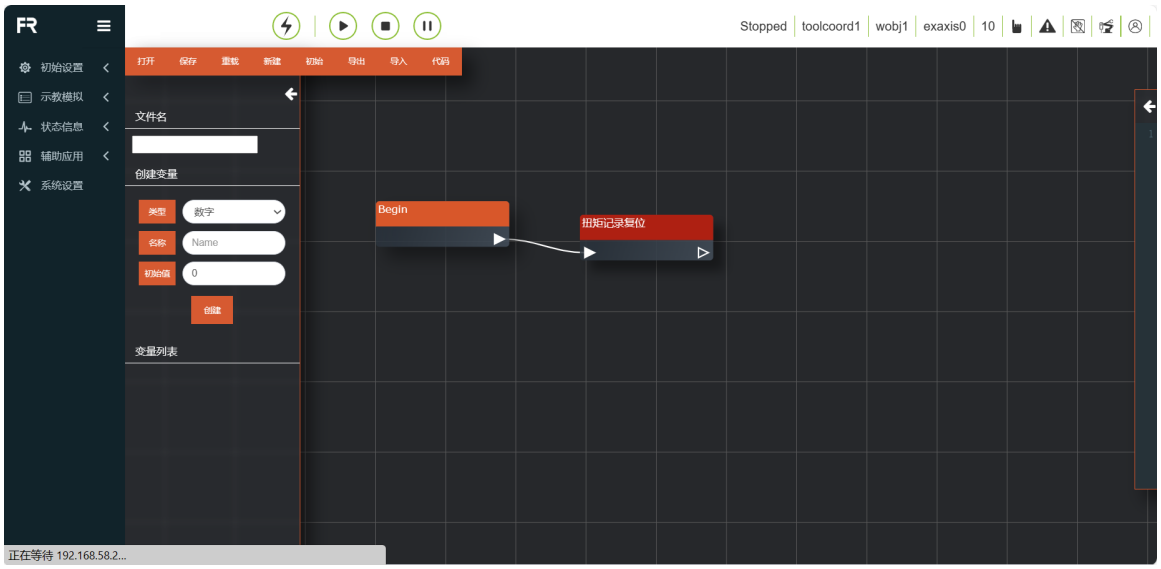
图表 6.2.104 “扭矩记录开始”指令节点界面

2. “扭矩记录结束”指令节点



图表 6.2.105 “扭矩记录结束”指令节点界面

3. “扭矩记录复位”指令节点



图表 6.2.106 “扭矩记录复位”指令节点界面

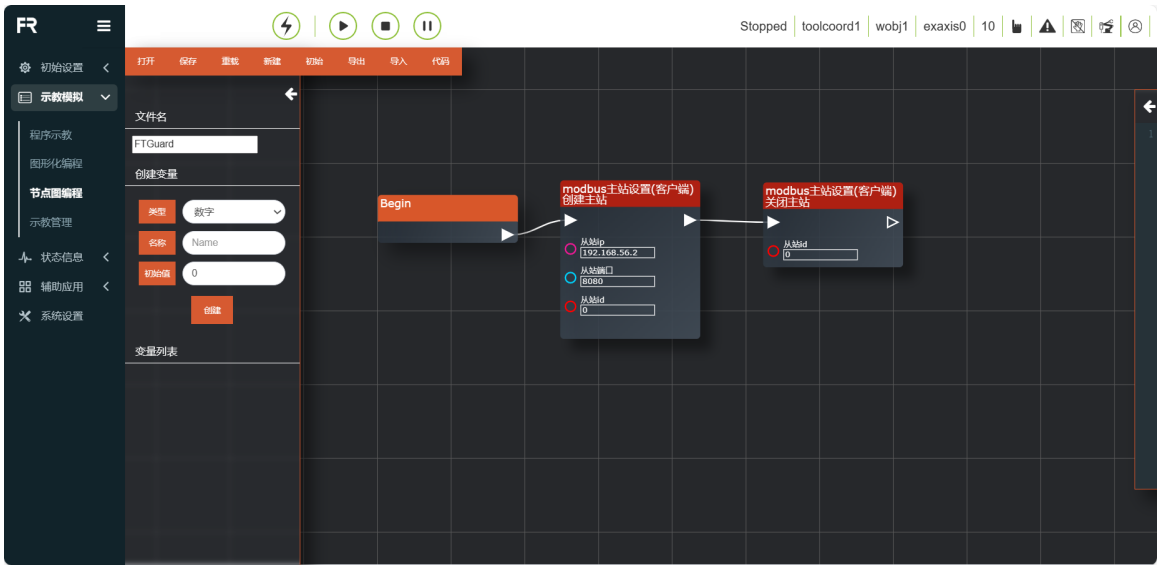
1.3.6.2.7.3 Modbus 命令

点击“Mobus”相关指令节点, 进入节点图编程界面

该指令功能为基于 ModbusTCP 协议的总线功能，用户可以通过相关指令控制机器人与 ModbusTCP client 或 server 通讯（主站与从站通讯），对线圈，离散量，寄存器进行读写操作。关于 ModbusTCP 更多操作功能，前请联系我们咨询。

1. “创建/关闭主站”指令节点, 创建/关闭 modbus-tcp 通信，机器人作为主站（客户端）, 参数:

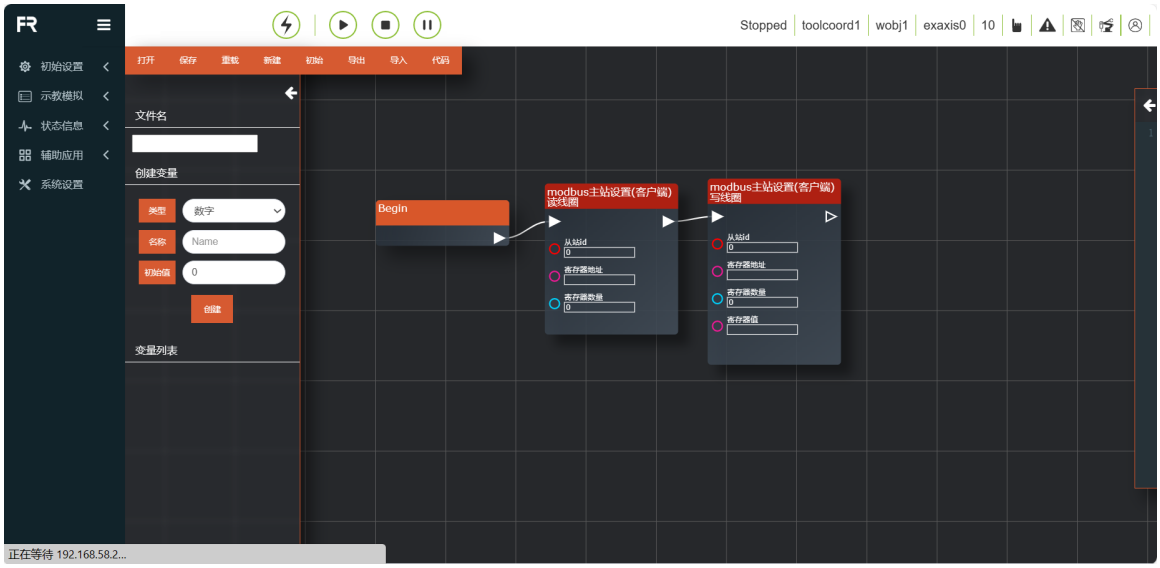
   - 从站 ip: ip 地址
   - 从站端口：根据实际情况输入，默认为 502；
   - 从站 id: 0 ~ 4



图表 6.2.107 “创建/关闭主站”指令节点界面

2. 主站线圈设置, 参数:

- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 寄存器数量：整数型
- 寄存器值：根据寄存器数量来定，可输入多个数值。例如数量为 3，值为 {1,0,1}

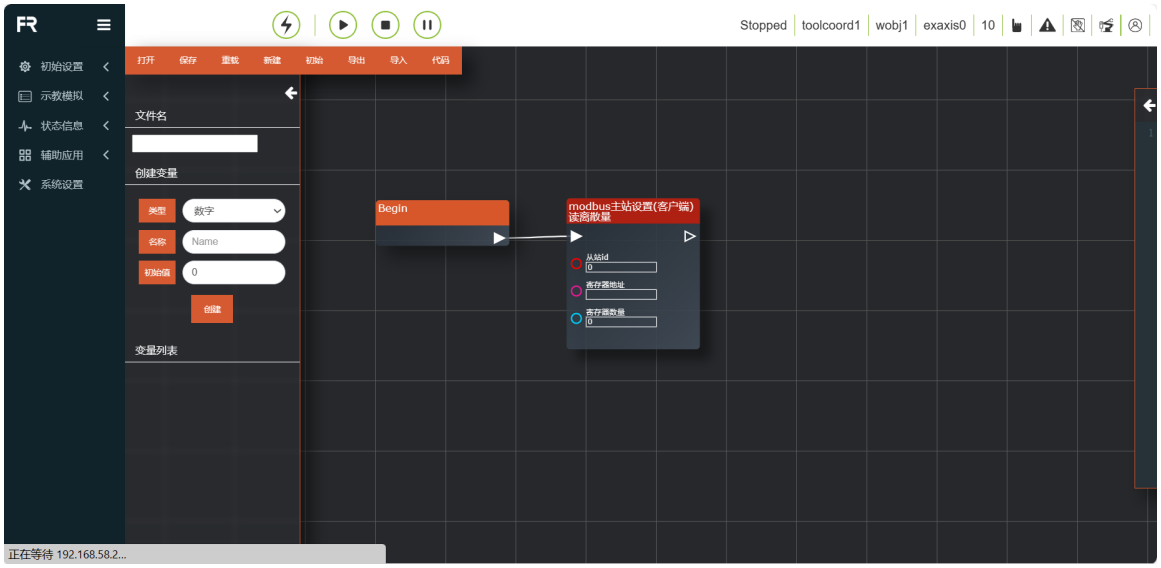


图表 6.2.108 主站“读/写线圈”指令节点界面

3. 主站离散量设置, 参数:

- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入

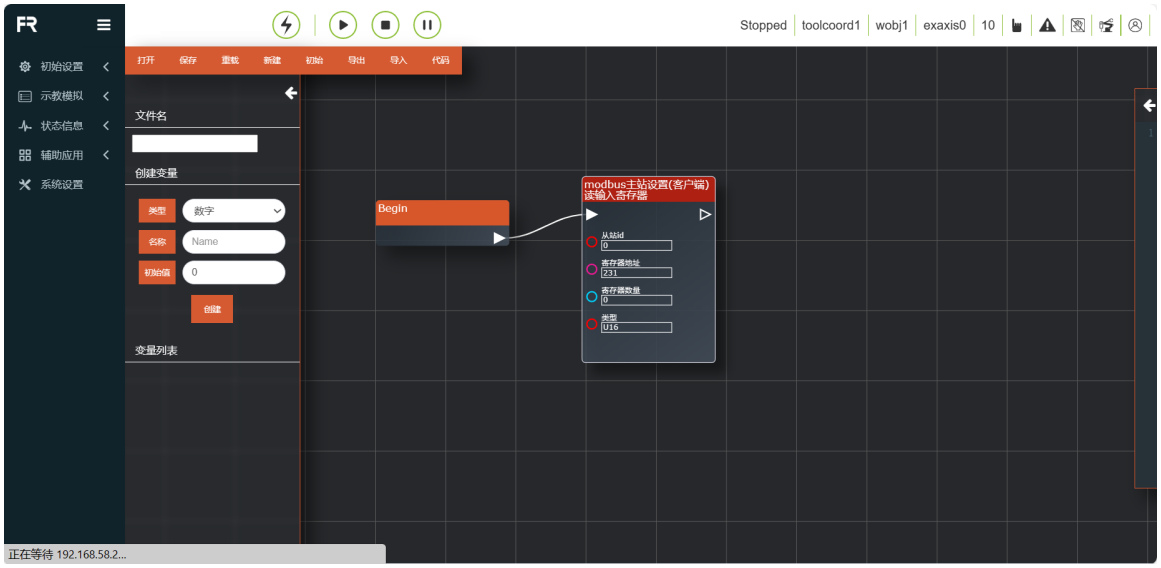
- 寄存器数量：整数型
- 类型：U16/U32/F32/F64，-16 位无符号整数（占 1 个寄存器），“U32” -32 位无符号整数（占 2 个寄存器），“F32” -32 位单精度浮点数（占 2 个寄存器），“F64” -64 位双精度浮点数（占 4 个寄存器）



图表 6.2.109 主站“读离散量”指令节点界面

4. 主站输入寄存器设置, 参数:

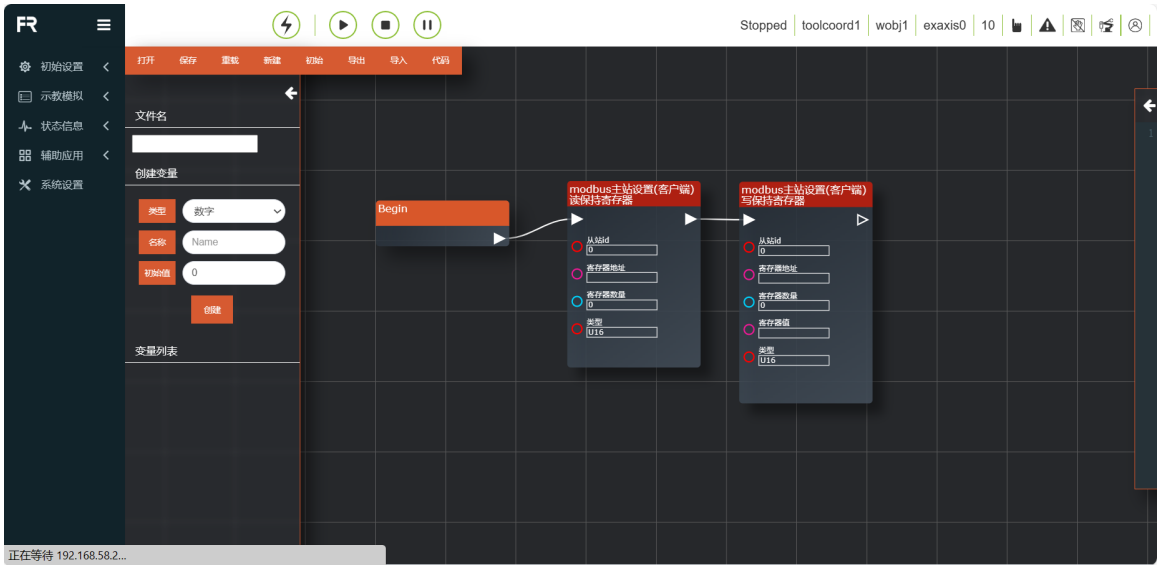
- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 类型：U16/U32/F32/F64，-16 位无符号整数（占 1 个寄存器），“U32” -32 位无符号整数（占 2 个寄存器），“F32” -32 位单精度浮点数（占 2 个寄存器），“F64” -64 位双精度浮点数（占 4 个寄存器）



图表 6.2.110 主站“读输入寄存器”指令节点界面

5. 主站保持寄存器设置, 参数:

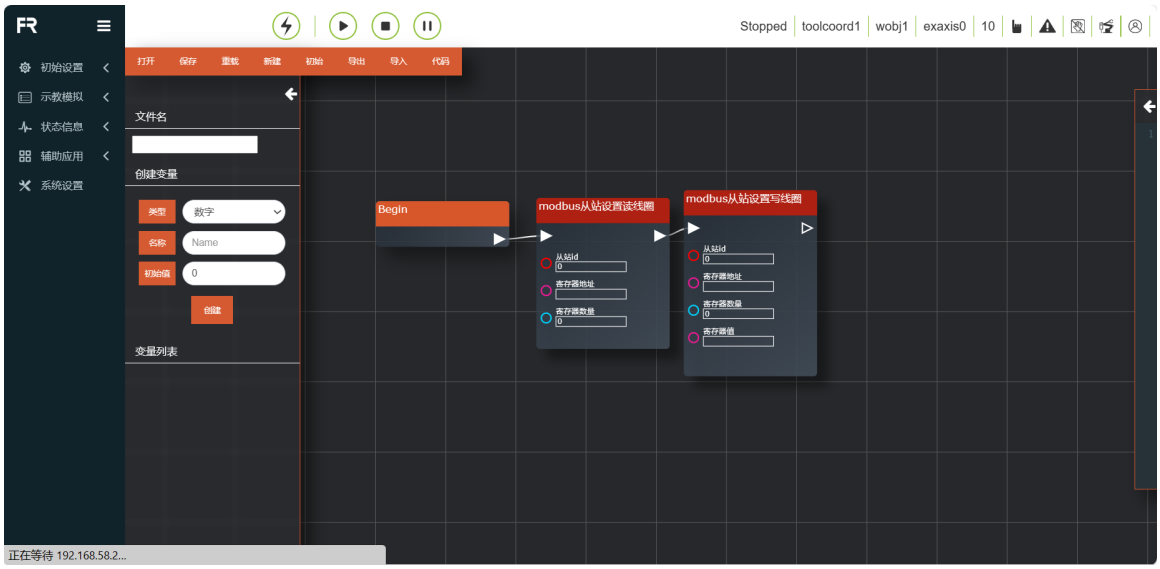
- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 寄存器数量：整数型
- 寄存器值：根据寄存器数量来定，可输入多个数值。例如数量为 3，值为 {1,0,1}
- 类型：U16/U32/F32/F64，-16 位无符号整数（占 1 个寄存器），“U32” -32 位无符号整数（占 2 个寄存器），“F32” -32 位单精度浮点数（占 2 个寄存器），“F64” -64 位双精度浮点数（占 4 个寄存器）



图表 6.2.111 主站“读/写保持寄存器”指令节点界面

6. 从站线圈设置, 参数:

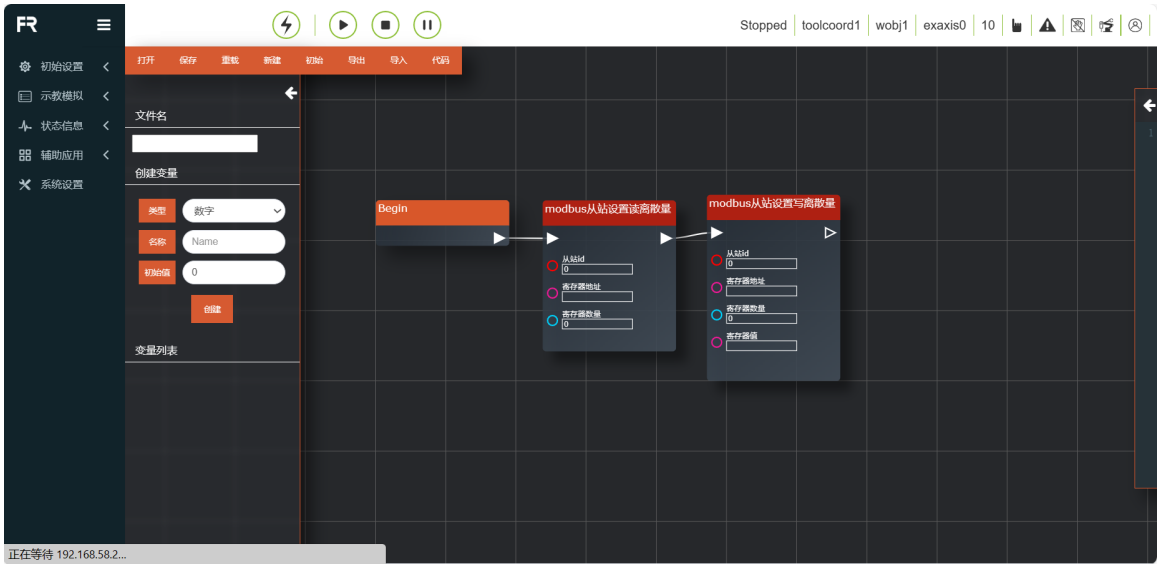
- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 寄存器数量：整数型



图表 6.2.112 从站“读/写线圈”指令节点界面

7. 从站离散量设置, 参数:

- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 寄存器数量：整数型
- 寄存器值：根据寄存器数量来定，可输入多个数值。例如数量为 3，值为 {1,0,1}

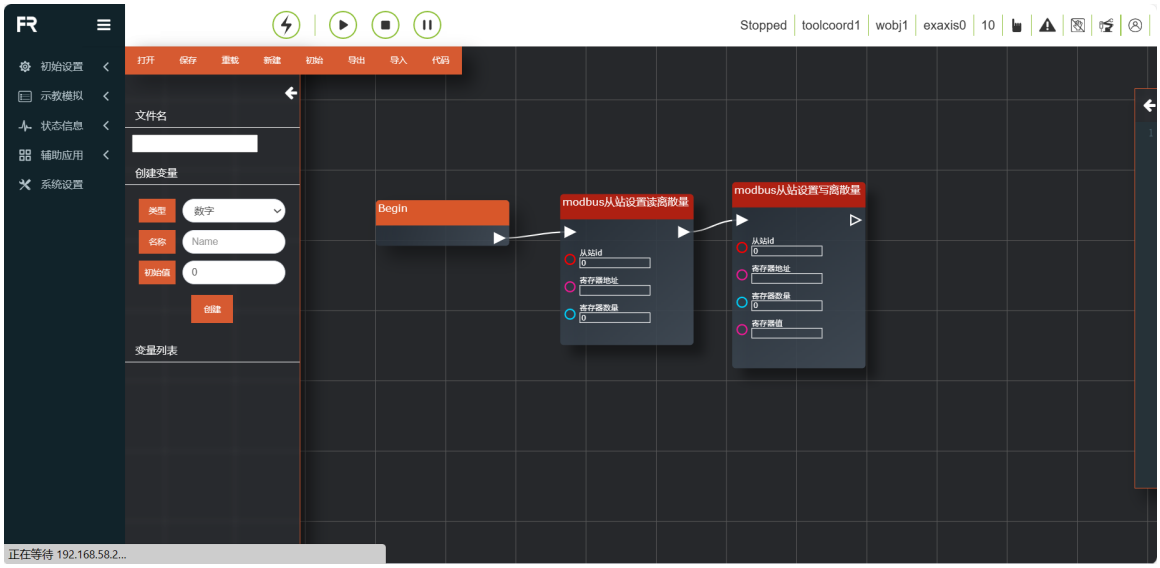


图表 6.2.113 从站“读/写离散量”指令节点界面

8. 从站输入寄存器设置, 参数:

- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入

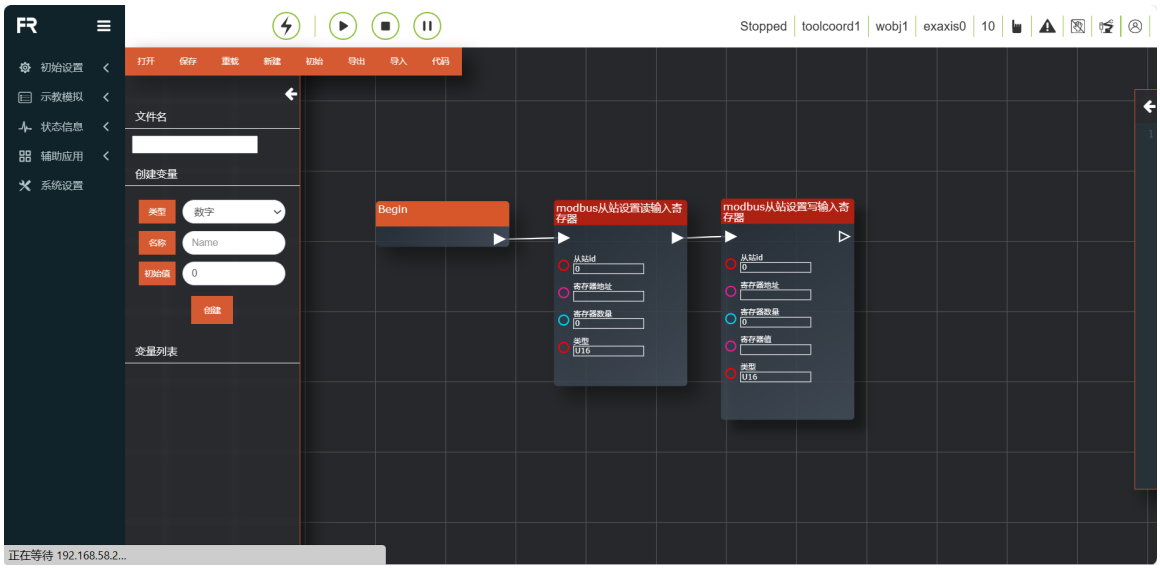
- 寄存器数量：整数型
- 寄存器值：根据寄存器数量来定，可输入多个数值。例如数量为 3，值为 {1,0,1}
- 类型：U16/U32/F32/F64，-16 位无符号整数（占 1 个寄存器），“U32” -32 位无符号整数（占 2 个寄存器），“F32” -32 位单精度浮点数（占 2 个寄存器），“F64” -64 位双精度浮点数（占 4 个寄存器）



图表 6.2.114 从站“读/写输入寄存器”指令节点界面

9. 从站保持寄存器设置, 参数:

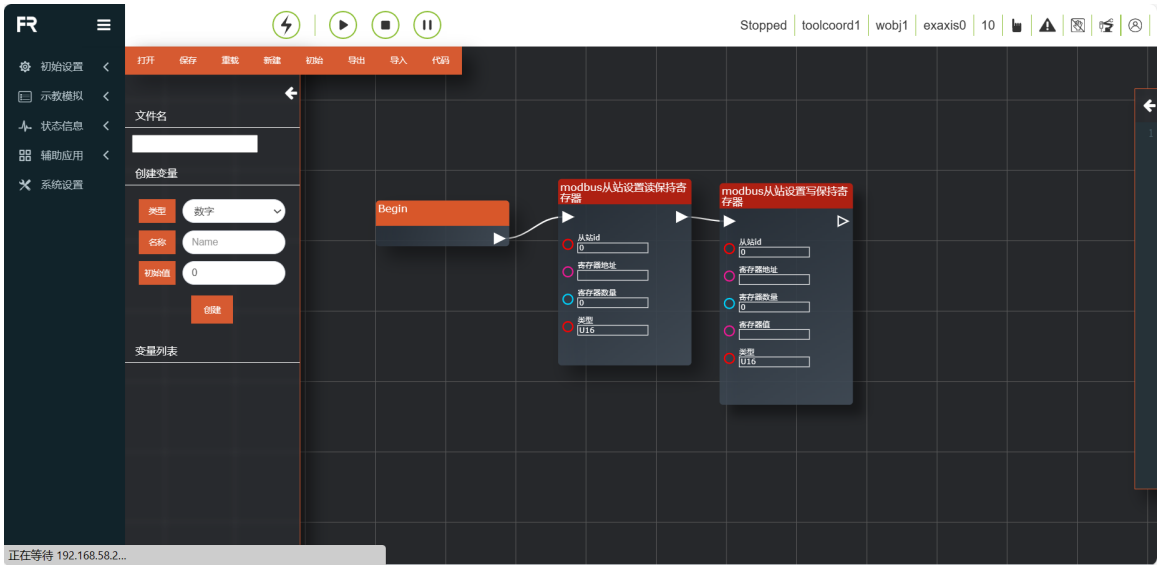
- 从站 id: 0 ~ 4
- 寄存器地址：根据实际情况输入
- 寄存器数量：整数型
- 寄存器值：根据寄存器数量来定，可输入多个数值。例如数量为 3，值为 {1,0,1}
- 类型：U16/U32/F32/F64，-16 位无符号整数（占 1 个寄存器），“U32” -32 位无符号整数（占 2 个寄存器），“F32” -32 位单精度浮点数（占 2 个寄存器），“F64” -64 位双精度浮点数（占 4 个寄存器）



图表 6.2.115 从站“读/写保持寄存器”指令节点界面

10. 读寄存器指令, 参数:

- 功能码：0x01-线圈/0x02-离散量/0x03-保持寄存器/0x04-输入寄存器
- 寄存器、线圈、离散量地址：根据实际情况输入
- 寄存器、线圈、离散量数量：0 ~ 255
- 地址：根据实际情况输入
- 是否应用线程：否/是

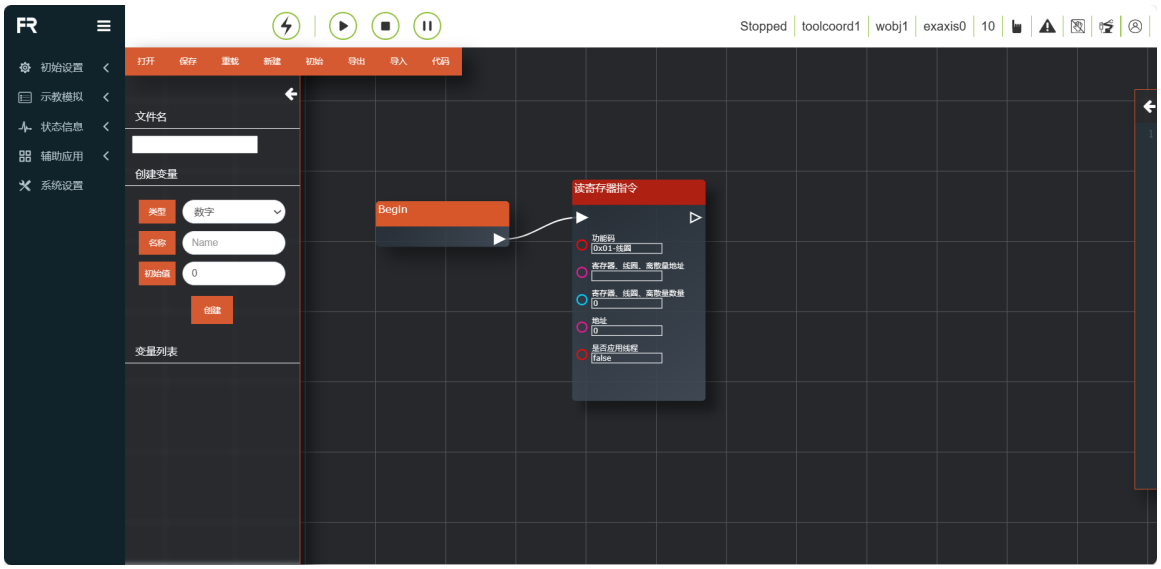


图表 6.2.116 “读寄存器指令”节点界面

11. 读寄存器数据指令, 参数:

- 寄存器、线圈、离散量数量：0 ~ 255

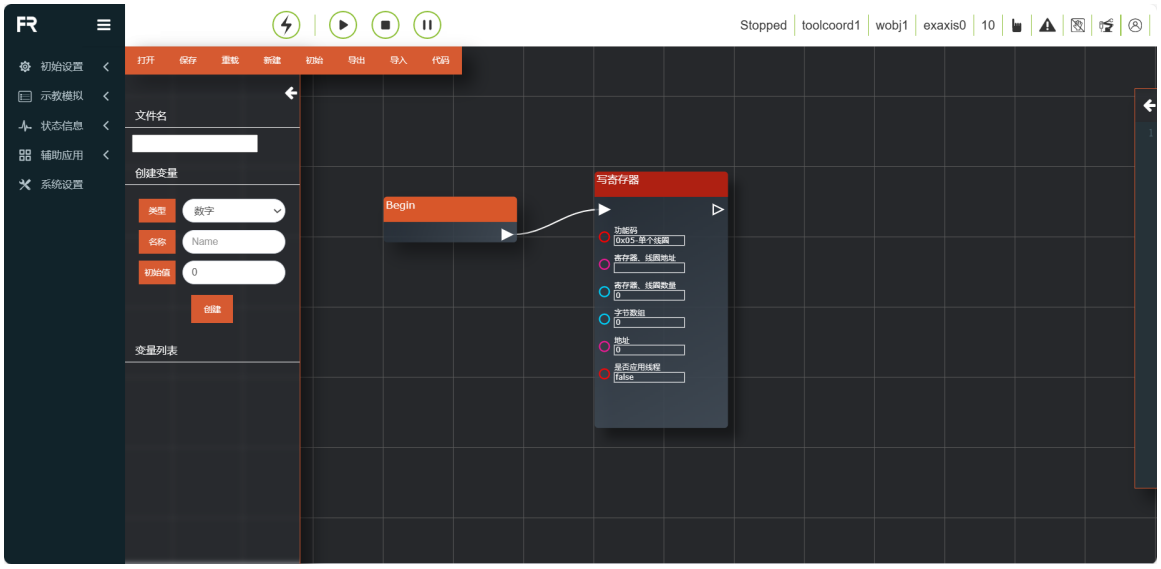
- 是否应用线程：否/是



图表 6.2.117 “读寄存器数据指令”节点界面

12. 写寄存器指令, 参数:

- 功能码：0x01-线圈/0x02-离散量/0x03-保持寄存器/0x04-输入寄存器
- 寄存器、线圈地址：根据实际情况输入
- 寄存器、线圈数量：0 ~ 255
- 字节数组：根据实际情况输入
- 地址：根据实际情况输入
- 是否应用线程：否/是

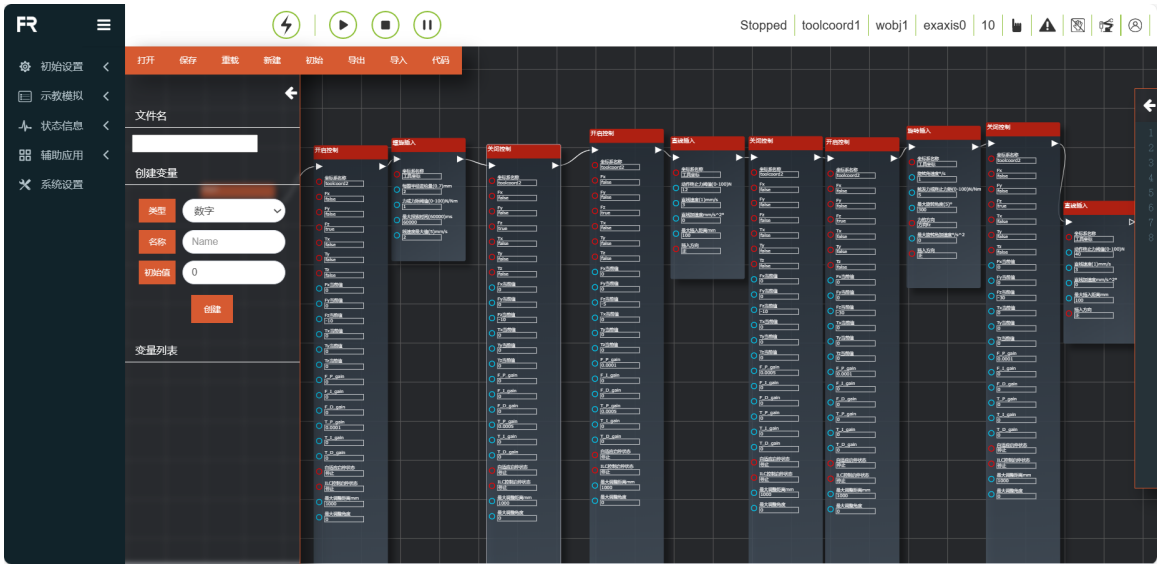


图表 6.2.118 “写寄存器指令”节点界面

1.3.6.3 应用场景使用示例

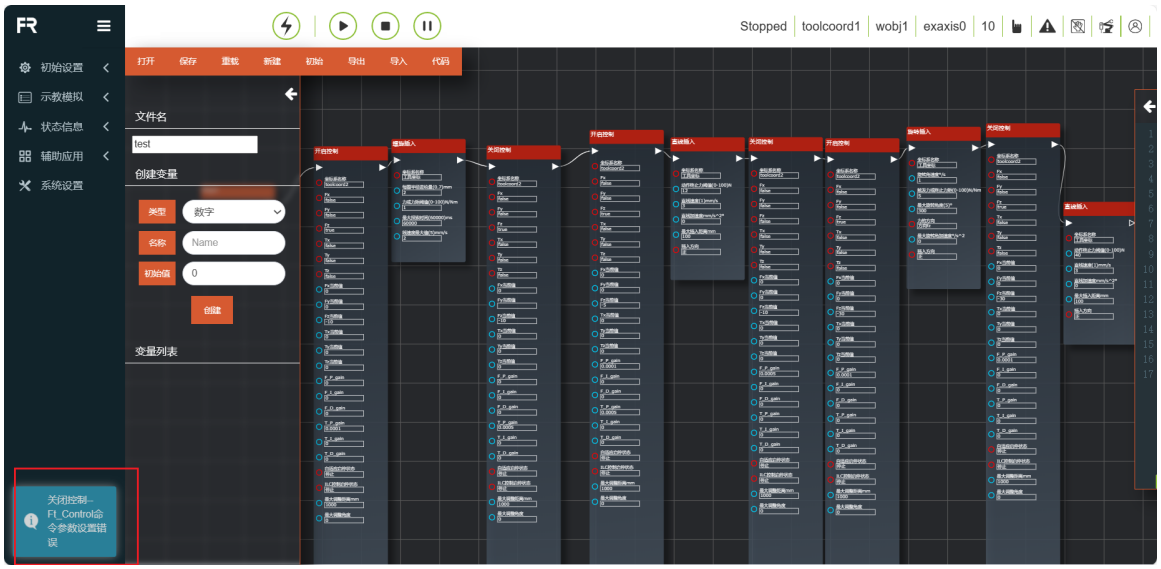
例如给机器人末端装上尖端，拖动到托盘孔位附近位置，想要进行力传感器螺旋、旋转和直线插入操作。

- 首先，右击鼠标键，选择” Begin”、” 开始/结束控制”、” 螺旋插入”、” 旋转插入”、” 直线插入” 指令节点；
- 依次按如下位置连接，并配置相关参数。



图表 6.3.1 “力控” 指令节点应用配置界面

- 输入文件名，若未输入正确参数，则保存失败，提示指令节点参数配置错误。



图表 6.3.2 指令节点参数配置错误界面

- 点击运行后，机器人会以螺旋形加直线的运动进行探索。当探索到正确孔位位置后，以直线加旋转插入运动，直至正确插入孔位。

1.3.7 机器人外设

1.3.7.1 夹爪外设配置

1.3.7.1.1 夹爪程序示教步骤

Step1：在用户外设配置界面中选择“末端外设配置”按钮，设备类型选择“夹爪设备”，夹爪的配置信息分为夹爪厂商、夹爪类型、软件版本和挂载位置，用户可根据具体的生产需求来配置相应的夹爪信息。若用户需要更改配置，可先选择相应的夹爪编号，点击“清除”按钮，来清除相应的按钮，并重新根据需求配置；

图表 4.1-1 夹爪配置

重要： 点击清除配置前，相应的夹爪应处于未激活状态。

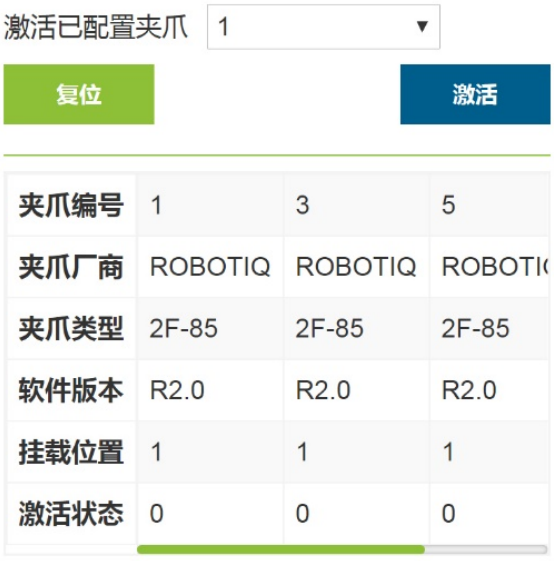
Step2：夹爪配置完成后，用户可在页面下方的夹爪信息表中查看相应的夹爪信息，若发现配置错误，可点击“清除”按钮，重新配置夹爪；

图表 4.1-2 夹爪配置信息

Step3：选择配置完成的夹爪，点击“复位”按钮，页面弹出命令发送成功后，再点击“激活”按钮，可查看夹爪信息表中的激活状态，来判断是否激活成功；

重要： 激活夹爪时，夹爪不可有夹持物

Step4：程序示教命令界面中选择“Gripper”命令。在夹爪命令界面中，用户可以选择想要控制的夹爪编号（已经完成配置并且被激活的夹爪），设置相应的开闭状态、开闭速度、开闭力矩已经等待夹爪动作的最大时间。完成设置后点击添加应用即可。此外还可以添加夹爪激活和复位指令，以便于在运行程序时去激活/复位夹爪。


激活已配置夹爪
1
复位
激活
夹爪编号 1 3 5
夹爪厂商 ROBOTIQ ROBOTIQ ROBOTI
夹爪类型 2F-85 2F-85 2F-85
软件版本 R2.0 R2.0 R2.0
挂载位置 1 1 1
激活状态 0 0 0


Gripper
夹爪编号 1
夹爪开闭 0 开 闭
开闭速度 0 慢 快
开闭力矩 0 小 大
最大时间 0 ms
是否阻塞 阻塞
添加
夹爪编号 1
复位
激活
已添加指令:
应用

图表 4.1-3 夹爪指令编辑

1.3.7.1.2 夹爪程序示教

序号	指令格式	注释
1	PTP(template2,100,-1,0)	**#等待夹取点**
2	PTP(template1,100,-1,0)	**#夹取点**
3	MoveGripper(1,255,255,0,1000,0)	**#夹爪闭合**
4	PTP(template2,100, -1,0)	/
5	PTP(template3,100, -1,0)	**#等待放件点**
6	PTP(template3,100, -1,0)	**#放件点**
7	MoveGripper(1,0,255,0,1000,0)	**#夹爪开启**

1.3.7.2 喷枪外设配置

1.3.7.2.1 喷枪外设配置步骤

Step1：在用户外设配置界面中选择“喷枪配置”按钮，用户可以通过喷涂功能一键配置按键，对喷涂所需 DO 进行快速配置（默认配置 DO10 为喷涂启停，DO11 为喷涂清枪）。用户也可以根据自己的需求在“IO 配置”界面，自定义配置 DO；

重要： 使用喷涂功能之前，需要先建立相应的工具坐标系，并在程序示教时应用建立好的工具坐标系。

Step2：配置完成后，点击“开始喷涂”、“停止喷涂”、“开始清枪”和“停止清枪”四个按钮，进行喷枪调试；

图表 4.2-1 喷枪配置

Step3：在程序示教命令界面选择“spray”命令。根据具体的程序示教需求，在相应的地方添加应用“开始喷涂”、“停止喷涂”、“开始清枪”和“停止清枪”四个指令。

图表 4.2-2 喷枪指令编辑


喷枪配置
IO接口配置
配置喷枪IO
喷枪调试
停止喷涂
开始喷涂
停止清枪
开始清枪


Spray
开始喷涂
添加
停止喷涂
添加
开始清枪
添加
停止清枪
添加
已添加指令：
应用

1.3.7.2.2 喷涂程序示教

序号	指令格式	注释
1	Lin(template1,100,-1,0,0)	#开始喷涂点
2	SprayStart()	#开始喷涂
3	Lin(template2,100,-1,0,0)	#喷涂路径
4	Lin(template3,100,-1,0,0)	#停止喷涂点
5	SprayStop()	#停止喷涂
6	Lin(template4,100,-1,0,0)	#清枪点
7	PowerCleanStart()	#开始清枪
8	WaitTime(5000)	#清枪时间 ms
9	PowerCleanStop()	#停止清枪

1.3.7.3 焊机外设配置

1.3.7.3.1 焊机外设配置步骤

Step1：在用户外设配置界面中选择“焊机配置”按钮，用户可以通过选择 IO 类型，选择“控制器 I/O”配置焊机 IO 按键，对焊机所需 DI 和 DO 进行快速配置（默认配置 DI12 起弧成功信号，DO9 送气信号，DO10 起弧信号，DO11 点动送丝，DO12 反向送丝，DO13 JOB 选择 1，DO14 JOB 选择 2，DO15 JOB 选择 3）。用户也可以根据自己的需求在“IO 配置”界面，自定义配置；或者选择“扩展 I/O”后，配置扩展 DI 的“焊机准备”和“起弧成功”，DO 的“焊机起弧”、“气体检测”、“正向送丝”和“反向送丝”；

图表 4.3-1 IO 配置（控制器 IO）

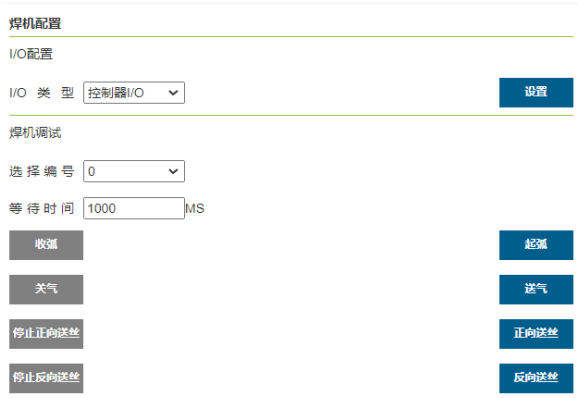
图表 4.3-2 IO 配置（扩展 IO）

重要： 使用焊机功能之前，需要先建立相应的工具坐标系，并在程序示教时应用建立好的工具坐标系。焊机功能通常与激光跟踪传感器配合使用。


焊机配置
I/O配置
I/O 类 型
控制器I/O
设置
焊机调试
选 择 编 号
0
等 待 时 间
1000
MS
收弧
起弧
关气
送气
停止正向送丝
正向送丝
停止反向送丝
反向送丝


焊机配置
I/O配置
I/O 类 型
扩展I/O
扩展I/O功能配置
DI
焊 机 准 备
-1
起 弧 成 功
-1
DO
焊 机 起 弧
13
气 体 检 测
12
正 向 送 丝
15
反 向 送 丝
14
设置
焊机调试
选 择 编 号
0
等 待 时 间
1000
MS
收弧
起弧
关气
送气
停止正向送丝
正向送丝
停止反向送丝
反向送丝

Step2：配置完成后，选择编号，设定等待时间，点击“收弧”、“起弧”、“送气”、“关气”、“正向送丝”和“反向送丝”六个按钮，进行焊机调试；



图表 4.3-3 焊机配置

Step3：在程序示教命令界面选择“Weld”命令。根据具体的程序示教需求，在相应的地方添加应用“起弧”、“收弧”、“关气”、“送气”、“停止正向送丝”、“正向送丝”、“停止反向送丝”和“反向送丝”指令。

图表 4.3-4 焊机指令编辑

1.3.7.3.2 焊机程序示教

序号	指令格式	注释
1	ARCEnd(0,0,10000)	收弧
2	ARCStart(0,0,10000)	起弧
3	SetAspirated(0,0)	关气
4	SetAspirated(0,1)	送气
5	SetForwardWireFeed(0,0)	停止正向送丝
6	SetForwardWireFeed(0,1)	正向送丝
7	SetReverseWireFeed(0,0)	停止反向送丝
8	SetReverseWireFeed(0,1)	反向送丝

1.3.7.3.3 电弧中断参数配置

电弧中断参数配置，可获取和设置电弧中断检测使能和确认时长。

焊接中断再恢复参数配置，可获取和设置焊接中断再重连使能、焊道重叠距离、从当前位置返回再起弧点速度和运行方式。

图表 4.3-5 电弧中断参数配置

当发生焊接中断，警告提示当前焊接已中断，可进行“退出中断”和“中断恢复”操作。


Weld
焊机电压
V
添加
焊机电流
A
添加
I/O类型
控制器I/O
焊接工艺编号
0
最大等待时间
10000
MS
收弧
起弧
关气
送气
停止正向送丝
正向送丝
停止反向送丝
反向送丝
Added Commands:
应用


检测电弧中断参数配置
使 能
确 认 时 长
50
ms
配置
焊接中断再恢复参数配置
使 能
重 叠 距 离
0
mm
速 度
0
%
运 动 方 式
PTP
配置

1.3.7.4 传感器外设配置

1.3.7.4.1 传感器外设配置步骤

Step1：在用户外设配置界面中选择“传感器配置”按钮，本小节以机器人末端为例说明，用户首先对最大差值进行设置，传感器扫描偏差点数最大差值建议默认设置为 4，数据处理根据实际使用场景选择原始数据或者 YZ 数据。控制器 IP 默认为 192.168.57.2，传感器 IP 配置为同一网段即可，端口为 5020，采样周期建议值 25，通信协议目前适配为睿牛通信协议，加载对应协议即可。加载完成后，可通过“打开传感器”和“关闭传感器”按键测试传感器。

图表 4.4-1 激光跟踪传感器 IP 配置

重要:

1. 使用传感器功能之前，需要先建立相应的工具坐标系，并在程序示教时应用建立好的工具坐标系。焊机功能通常与传感器配合使用。

2. 传感器最大差值即为激光扫描焊缝位置上一时刻与当前时刻的最大偏差，范围为 [0~10], 单位为 mm，推荐值为 4。

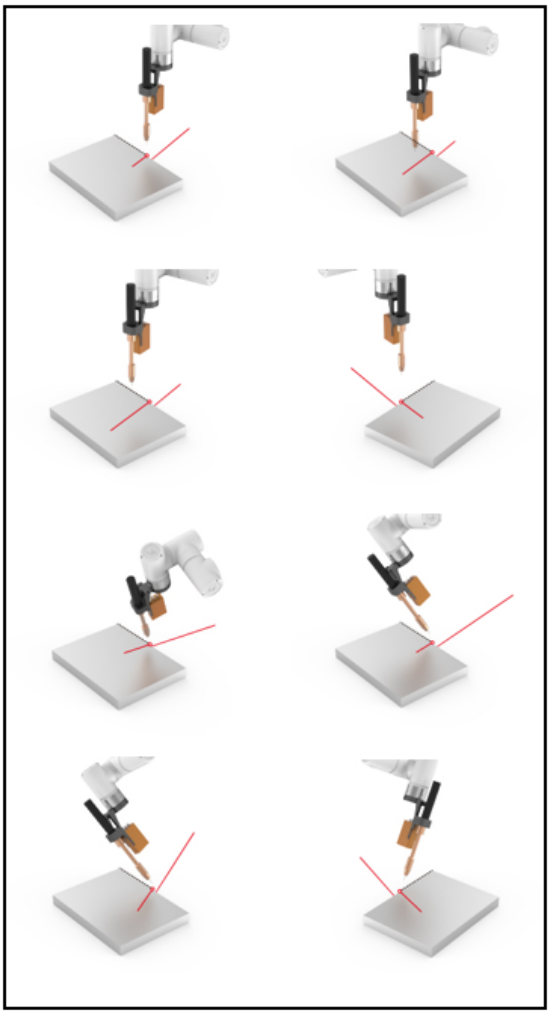
Step2：标定传感器参考点。

在工具坐标系设置功能中，我们标定传感器类型工具，使用六点法配置传感器坐标系。在机器人工作空间中选择一个固定的点，将传感器中心点从三个不同的角度移至所选点上，分别设置点 1，2，3。将传感器中心点垂直移至所选点正上方，记录点 4。将传感器中心点由固定点移至传感器坐标系的 X 轴方向上一点，设置点 5。回到固定点，垂直向上移动，将传感器中心点由固定点移至传感器坐标系的 Z 轴方向上一点，设置为点 6。点击计算，得到传感器工具的位姿，点击应用，即可完成。

图表 4.4-2 参考点配置-六点法

八点法：在工具坐标系设置功能中，我们标定传感器类型工具，使用八点法配置传感器坐标系，选择八点法，移动传感器激光线使其与标定板上的标定线重合，尽量使得传感器与标定线保持较近距离且识别到标定点，记录点 1，移动-y/+y 方向 20mm 左右, 调整机器人使激光识别到标定点，记录点 2，移动-x/+x 方向 20mm 左右，调整机器人使激光识别到标定点，记录点 3，移动-y/+y 方向 20mm 左右，调整机器人使激光识别到标定点，记录点 4，移动-rx 方向 5mm 左右，调整机器人使激光识别到标定点，记录点 5。移动-ry 方向 5mm 左右，调整机器人使激光识别到标定点，记录点 6，移动-rz 方向 5mm 左右，调整机器人使传感器识别到标定

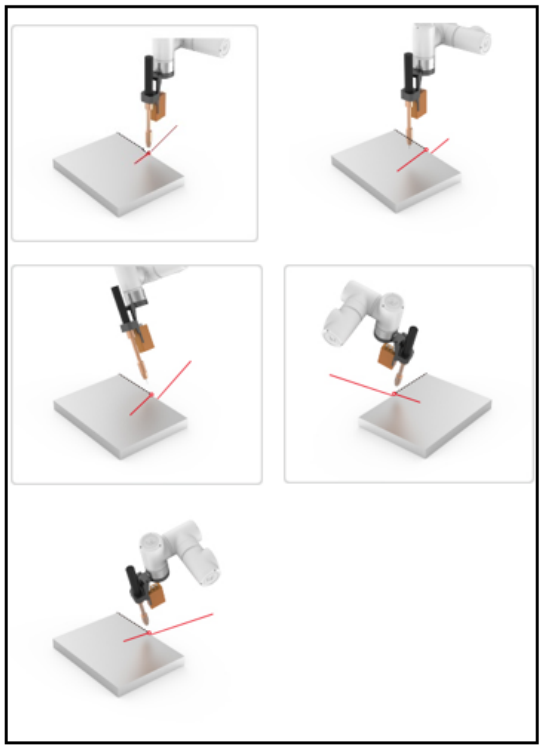
点，记录点 7，移动-rz 方向 5mm 左右，调整机器人使激光识别到标定点，记录点 8。点击计算，得到传感器位姿，点击应用，即可完成。



图表 4.4-3 参考点配置-八点法

五点法：在工具坐标系设置功能中，我们标定传感器类型工具，使用五点法配置传感器坐标系，首先确定一个固定点，将工具末端对准该点，记录点 1，接着调整机器人姿态，使得激光识别到记录的固定点，分别记录点 2 至点 5，注意姿态变化需要尽可能大。点击计算，得到传感器位姿，点击应用，即可完成。

图表 4.4-4 参考点配置-五点法



1.3.7.4.2 激光传感器跟踪功能

指令说明：在程序示教命令界面选择“Laser”命令。里面集成了激光的相关指令，根据具体的程序示教需求，在相应的地方添加指令，参考下方的程序示例。

图表 4.4-5 激光跟踪指令编辑

程序示例：

1.3.7.4.3 激光传感器轨迹复现功能

指令说明：在程序示教命令界面选择“LT-Rec”命令。该指令主要用于激光识别路径起点终点获取以及轨迹复现，根据具体的程序示教需求，在相应的地方添加指令，参考下方的程序示例。

图表 4.4-6 轨迹复现指令编辑

程序示例：


Laser
传感器命令
选择焊缝类型
0
打开传感器
关闭传感器
传感器加载
功能选择
睿牛RRT-SV2-BF
卸载
加载
跟踪命令
坐标系名称
开始跟踪
停止跟踪
数据记录
功能选择
停止记录
等待时间
10
ms
添加

序号	指令格式	注释
1	LTLaserOn(2)	#打开传感器
2	PTP(template1,100,-1,0)	#传感器起始点
3	LTSearchStart(1,20,100,10000,2)	#开始寻位
4	LTSearchStop()	#停止寻位
5	Lin(seamPos,20,-1,0,0,0)	#焊缝起点
6	LTTrackOn(2)	#激光跟踪
7	ARCStart(0,10000)	#焊机起弧
8	Lin(SeamEnd11,10-1,0,0)	#焊缝终点
9	ARCEnd(0,10000)	#焊机收弧
10	LTTrackOff()	#传感器跟踪关闭
11	LTLaserOff()	#关闭传感器


LT-Rec
获取焊缝点
运动方式
PTP
速度
30
%
获取焊缝终点
获取焊缝起点
焊缝数据记录
功能选择
轨迹复现
等待时间
10
ms
速度
30
%
添加
跟踪复现
激光跟踪复现
Added Commands:
应用

序号	指令格式	注释
1	PTP(template1,100,-1,0)	#运动至起点
2	LaserSensorRecord(2,0,30)	#传感器开始记录
3	Lin(template2,100,-1,0,0)	#运动至终点
4	LaserSensorRecord(0,0,30)	#停止记录
5	pos={}	#初始化数组
6	pos=GetWeldTrackingRecordStartPos(0,30)	#获取激光记录的起点
7	If type(pos) == “table” then	#判断数据类型
8	LaserPTP(#pos,pos)	#运动至激光轨迹起点
9	end	
10	LaserSensorRecord(3,0,30)	#设置复现轨迹
11	MoveLTR()	#开始复现轨迹
12	LaserSensorRecord(0,0,30)	#结束

1.3.7.5 扩展轴外设配置

在用户外设配置界面中选择“扩展轴”按钮，进入扩展轴界面，选择组合方式进行对应的扩展轴外设配置。组合方式分为：控制器 +PLC（UDP）和控制器 + 伺服驱动器（485）。

1.3.7.5.1 控制器 +PLC（UDP）

1.3.7.5.1.1 配置步骤

Step1：首先对扩展轴 UDP 通信进行配置。设置 IP 地址、端口号和通信周期。

图表 4.5-3 扩展轴通信配置

Step2：选择扩展轴编号 1，点击“参数配置”按键进入右侧界面。设置轴类型，轴方向，运行速度，加速度，正方向限位，反方向限位，导程，编码器分辨率，起点偏置，厂家，型号和模式，点击配置即可配置完成。

图表 4.5-1 扩展轴参数配置

重要： 使用扩展轴功能之前，需要先建立相应的扩展轴标系，并在程序示教时应用建立好的工具坐标系。扩展轴功能主要与焊机功能和激光跟踪传感器功能配合使用。


扩展轴配置
轴类型
直线导轨
轴方向
正
运行速度
1000
mm/s
加速度
2000
mm/s2
正方向限位
1000
反方向限位
-1000
导程
14.130
编码器分辨率
10000
起点偏置
200
mm
厂家
禾川
型号
SV-XD3EA040L-E
模式
增量系统
*分辨率设置值，0：开环，其他值：有反馈。

Step3：使点击“零点设置”按键进入零点设置弹窗，如右侧图片所示。设定回零方式，寻零速度，零点箍位速度和轴方向，点击“设置”按键，扩展轴开始回零，回零状态会显示在轴方向下方空白处，当出现“回零已完成”提示表明扩展轴零点设置成功。

图表 4.5-2 扩展轴零点设置

Step4：选择已经配置好参数的扩展轴编号，点击“伺服使能”后，设置运行速度，加速度和单次运行最大距离，可以进行正向转动和反向转动测试扩展轴。

图表 4.5-3 扩展轴测试

Step5：扩展轴通常于激光传感器配合使用，此时激光传感器通常采用外部安装方式，传感器参考点配置需要采用三点法标定，而不是之前使用的六点法标定。将工具中心对准右侧横截面底部中间点（靠近相机那一侧）设定点 1，将工具中心点对准另一截面即左侧横截面底部中间点，设定点 2，将工具中心点移至传感器右侧横截面上边缘中间点，设定点 3，计算并保存，点击应用完成三点法标定。

图表 4.5-4 传感器三点法标定

Step6：在程序示教命令界面选择“EAxis”命令。根据具体的程序示教需求，在相应的地方添加指令。

图表 4.5-5 扩展轴指令编辑

扩展轴配置

组合方式 控制器 + PLC（UDP）

UDP通信配置

IP 地 址

端 口 号

通信周期

配置 加载

扩展轴参数配置

扩展轴编号 1

获取信息

零点设置 参数配置

扩展轴测试

扩展轴编号 1

运行速度 100 %

加 速 度 100 %

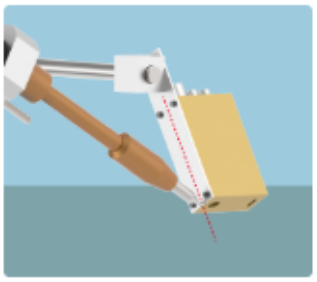
传感器参考点配置

X Y Z

RX RY RZ

取消修改 应用

六点法 八点法 十点法 三点法

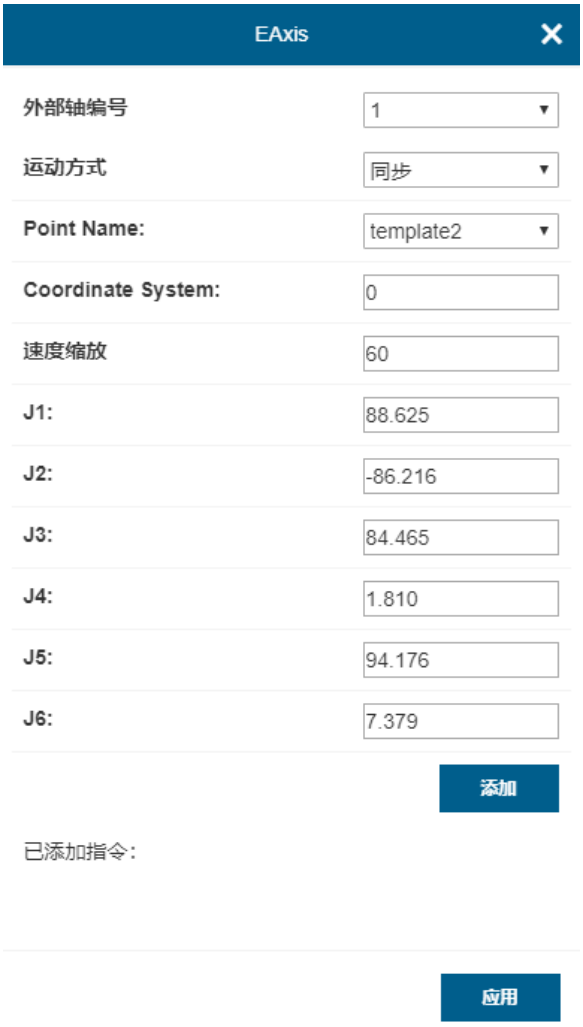


设置点1

设置点2

设置点3

计算


EAxis
外部轴编号
1
运动方式
同步
Point Name:
template2
Coordinate System:
0
速度缩放
60
J1:
88.625
J2:
-86.216
J3:
84.465
J4:
1.810
J5:
94.176
J6:
7.379
添加
已添加指令:
应用

1.3.7.5.2 控制器 + 伺服驱动器（485）

Step1：首先对伺服驱动器参数进行配置。设置伺服驱动器编号、伺服驱动器厂商、伺服驱动器型号、伺服驱动器软件版本、编码器分辨率和机械传动比。点击“清除”按钮清空当前伺服驱动器编号的配置。配置成功后获取伺服驱动器配置列表数据。

图表 4.5-5 配置伺服驱动器参数

Step2：进行伺服驱动器调试前，务必先设置伺服驱动器编号，设置成功后进行控制模式设置。设置回零方式、寻零速度和零点箍位速度。选择“位置模式”时，设置“目标位置”和“运行速度”；选择“速度模式”设置“目标速度”。

重要： 设置控制模式前，若伺服已使能，请先去使能，否则设置无法生效。

图表 4.5-5 伺服驱动器调试

Step3：在程序示教命令界面选择“扩展轴”命令。根据具体的程序示教需求，在相应的地方添加指令。

图表 4.5-5 扩展轴指令编辑

1.3.7.5.3 扩展轴配合激光跟踪焊接示教程序

1.3.7.6 传送带跟踪配置

1.3.7.6.1 传送带跟踪配置步骤

Step1：在用户外设配置界面中选择“传送带跟踪”按钮，进入传送带跟踪配置界面，点击“配置传送带 IO”按键快速配置传送带功能所需 IO，之后根据实际使用功能情况配置对应的参数，此处以无视觉跟踪抓取功能为例，需要配置传送带编码器通道，分辨率，导程，视觉搭配选择否，点击配置。

图表 4.6-1 传送带配置

Step2：接下来设置抓取点补偿值，为 X,Y,Z 三个方向上的补偿距离，可在调试过程中根据实际情况设置。

图表 4.6-2 传送带抓取点补偿配置

Step3：开启传送带，将标定的物体移动到定义的 A 点位置，停止传送带。移动机器人，将机器人末端的标定杆尖点与所标定的物体尖点对齐，点击起始点 A 按键，跳出对话框，显示当前编码器值和机器人位姿，点击标定完成起始点 A 标定。

图表 4.6-3 起始点 A 配置

Step4：点击参考点按键，进入参考点标定，记录参考点时记录机器人抓取时的高度和姿态，每次跟踪时都会以记录参考点的高度和姿态区跟踪抓取，可以和 AB 点不在一个高度，点击标定完成参考点标定。

图表 4.6-4 参考点配置


EAxis
组合方式
控制器+伺服驱动
*添加伺服指令前请确认在用户外设配置的扩展轴配置中完成伺服驱动器参数配置
伺服功能
伺服ID
1
添加
控制模式
位置模式
添加
*添加控制模式设置指令会在该指令前后固定搭配伺服使能和去使能指令
伺服使能
伺服使能
去除使能
伺服回零设置
回零方式
当前位置
寻零速度
mm/s(°/s)
零点箍位速度
mm/s(°/s)
添加
参数配置
当前控制模式 位置模式
目标位置
mm(°)
运行速度
mm/s(°/s)
添加
Added Commands:
应用

序号	指令格式	注释
1	EXT_AXIS_PTP(1,1,laserstart)	#外部轴运动激光传感器起始点
2	PTP(laserstart,10,-1,0)	#机器人运动激光传感器起始点
3	LTSearchStart(3,20,10,10000)	#开始寻位
4	LTSearchStop()	#停止寻位
5	EXT_AXIS_PTP(1,1,seamPos)	#外部轴运动焊缝起点
6	Lin(seamPos,20,-1,0,0,0)	#机器人运动焊缝起点
7	LTTrackOn()	#激光跟踪
8	ARCStart(0,10000)	#焊机起弧
9	EXT_AXIS_PTP(1,1,laserend)	#外部轴运动焊缝终点
10	Lin(laserend,10,-1,0,0)	#机器人运动焊缝终点
11	ARCEnd(0,10000)	#焊机收弧
12	LTTrackOff	#激光跟踪关闭


传送带跟踪配置
IO配置
配置传送带IO
功能配置
功 能 选 择
跟踪抓取
参数配置
编码器通道
通道1
分 辨 率
10000
导 程
200
mm
视 觉 搭 配
否
配置
抓取点补偿
X:
mm
Y:
mm
Z:
mm
配置


起始点A
X
0
Y
0
Z
0
RX
0
RY
0
RZ
0
编码器值
0
标定
机器人參考点
X
0
Y
0
Z
0
RX
0
RY
0
RZ
0
标定

Step5：开启传送带，将标定的物体移动到定义的 B 点位置，停止传送带。移动机器人，将机器人末端的标定杆尖点与所标定的物体尖点对齐，点击终点 B 按键，弹出对话框，显示当前编码器值和机器人位姿，点击标定完成终点 B 标定。

图表 4.6-5 终点 B 配置

1.3.7.6.2 传送带跟踪示教程序

1.3.7.7 姿态自适应配置

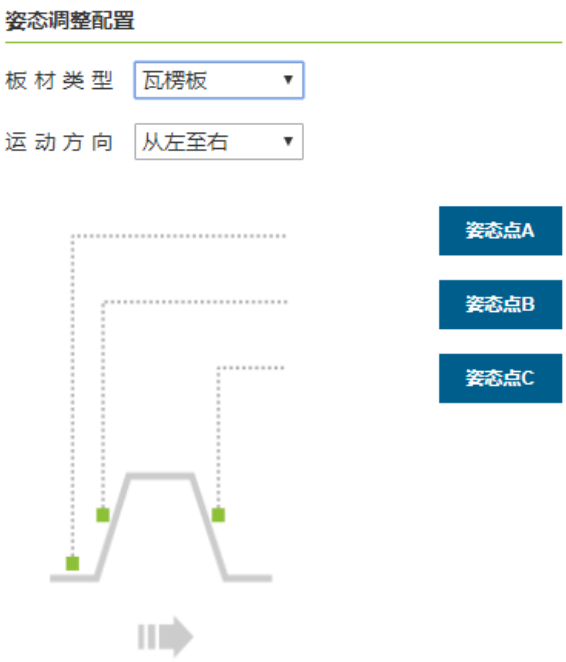
1.3.7.7.1 姿态自适应配置步骤

Step1：在用户外设配置界面中选择“跟踪姿态配置”按钮，进入姿态调整配置界面，选择板材类型和机器人实际工作运动方向，调整机器人姿态，分别设置姿态点 A，姿态点 B 和姿态点 C，通常 A 为平面姿态点，B 为上升沿姿态点，C 为下降沿姿态点。

图表 4.7-1 姿态调整配置

重要： A 姿态和 B 姿态，A 姿态和 C 姿态之间的姿态变化在满足应用需求下姿态变化越小越好。姿态自适应

序号	指令格式	注释
1	PTP(conveyorstart,30,-1,0)	#机器人抓取起点
2	While(1) do	#循环抓取
3	ConveyorIODetect(10000)	#IO 实时检测物体
4	ConveyorGetTrackData(1)	#物体位置获取
5	ConveyorTrackStart(1)	#传送带跟踪开始
6	Lin(cvrCatchPoint,10,-1,0,0)	#机器人到达抓取点
7	MoveGripper(1,255,255,0,10000)	#夹爪抓取物体
8	Lin(cvrRaisePoint,10,-1,0,0)	#机器人提起
9	ConveyorTrackEnd()	#传送带跟踪结束
10	PTP(conveyorraise,30,-1,0)	#机器人到达等待点
11	PTP(conveyorend,30,-1,0)	#机器人到达放置点
12	MoveGripper(1,0,255,0,10000)	#夹爪松开
13	PTP(conveyorstart,50,-1,0)	#机器人再次回到抓取起点，等待下一次抓取
14	end	#结束



功能为辅助应用功能，通常配合焊缝跟踪使用。

Step2：在程序示教命令界面选择“Adjust”命令。根据具体的程序示教需求，在相应的地方添加指令。

图表 4.7-2 姿态调整指令编辑

1.3.7.7.2 姿态自适应配合扩展轴和激光跟踪焊接示教程序

1.3.7.8 力/扭矩传感器外设配置

1.3.7.8.1 力/扭矩传感器配置步骤

Step1：在用户外设配置界面中选择“末端外设配置”按钮，设备类型选择“力传感器设备”，力传感器配置信息分为厂商、类型、软件版本和挂载位置，用户可根据具体的生产需求来配置相应的力传感器信息。若用户需要更改配置，可先选择相应的编号，点击“清除”按钮，来清除相应的信息，并重新根据需求配置；

图表 4.8-1 力/矩传感器配置

重要： 点击清除配置前，相应的传感器应处于未激活状态。

Step2：力传感器配置完成后，用户可在页面下方的信息表中查看相应的力传感器信息，若发现配置错误，可点击“清除”按钮，重新配置。

图表 4.8-2 力/矩传感器配置信息


Adjust
板材类型
瓦楞板
运动方向
从左至右
姿态调整时间
1000
ms
姿态A
rx: 0.000
ry: 0.000
rz: 0.000
姿态B
rx: 0.000
ry: 0.000
rz: 0.000
姿态C
rx: 0.000
ry: 0.000
rz: 0.000
B
C
A
关闭调整
开启调整
已添加指令:
应用

序号	指令格式	注释
1	EXT_AXIS_PTP(1,1,laserstart)	#外部轴运动激光传感器起始点
2	PTP(laserstart,10,-1,0)	#机器人运动激光传感器起始点
3	LTSearchStart(3,20,10,10000)	#开始寻位
4	LTSearchStop()	#停止寻位
5	EXT_AXIS_PTP(1,1,seamPos)	#外部轴运动焊缝起点
6	Lin(seamPos,20,-1,0,0,0)	#机器人运动焊缝起点
7	LTTrackOn()	#激光跟踪
8	ARCStart(0,10000)	#焊机起弧
9	PostureAdjustOn(0,PosA,PosC,PosB,1000)	#姿态自适应调整开启
10	EXT_AXIS_PTP(1,1,laserend)	#外部轴运动焊缝终点
11	Lin(laserend,10,-1,0,0)	#机器人运动焊缝终点
12	ARCEnd(0,10000)	#焊机收弧
13	PostureAdjustOff(0)	#姿态自适应调整关闭
14	LTTrackOff	#激光跟踪关闭


末端外设配置
设备类型:
力传感器设备
厂 商 :
kunwei
类 型 :
KWR75B
软件版本:
1.0
挂载位置:
末端1号口
清除
配置


激活已配置力传感器
复位
激活
去除零点
零点矫正
编号 1 1 1 1 1 1 5
厂商 1 1 1 1 1 1 6
类型 1 1 1 10 1 1 7
软件版本 1 1 1 1 1 1 8
挂载位置 1 1 1 1 1 1 9
激活状态 1 1 1 1 1 1 10

Step3：选择配置完成的力传感器编号，点击“复位”按钮，页面弹出命令发送成功后，再点击“激活”按钮，可查看力传感器信息表中的激活状态，来判断是否激活成功；此外，力传感器会有初始值，用户根据使用需求选择“零点矫正”和“去除零点”。力传感器零点矫正需要确保力传感器水平垂直向下，且机器人未配置负载。

Step4：力传感器配置完成后，需要配置传感器类型工具坐标系，可根据传感器与末端工具中心的距离直接输入传感器工具坐标系值并应用。

1.3.7.8.2 力/扭矩传感器负载辨识

在机器人配置界面中选择“力/扭矩传感器负载”进行配置。

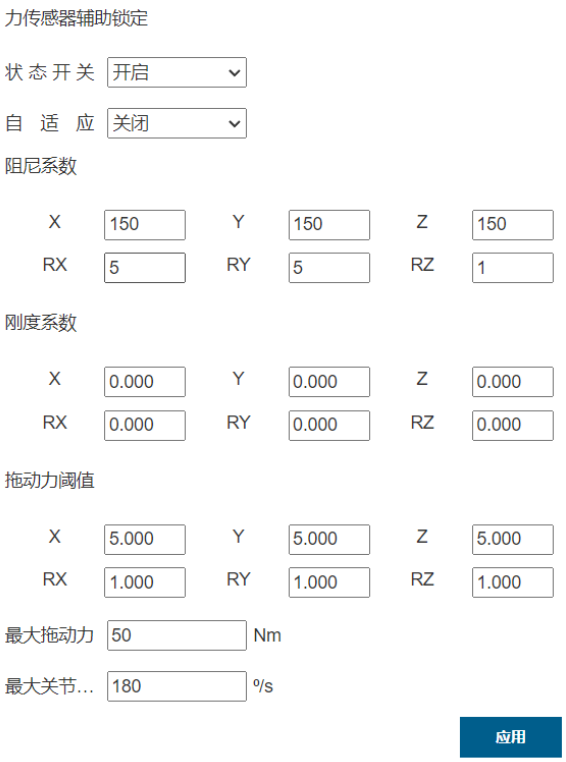
特定姿态辨识：清除末端负载数据，配置好力传感器后，建立传感器坐标系，将机器人末端姿态调整为垂直向下，进行“零点矫正”后安装末端负载。首先选择对应传感器工具坐标系，调整机器人，使得传感器及工具垂直向下，记录数据，计算质量。接着，调整机器人 3 个不同姿态，分别记录三组数据，计算出质心，确认无误后点击应用。

动态辨识：清除末端负载数据，配置好力传感器后，建立传感器坐标系，将机器人末端姿态调整为垂直向下，进行“零点矫正”后安装末端负载。点击“辨识开启”，拖动机器人进行运动，接着点击“辨识关闭”，即可自动将负载结果应用到机器人中。

图表 4.8-4 力/扭矩传感器负载辨识

1.3.7.8.3 力/扭矩传感器辅助拖动

配置好传感器后，可以搭配传感器对拖动机器人进行更好的辅助。第一次使用时可以按照右侧图片的数据进行配置，应用完成后，此时无需进入拖动模式，直接对末端力传感器进行拖拽，即可控制机器人在固定姿态进行移动。（如下图中的数据为参考标准）



图表 4.8-4 力/扭矩传感器拖动锁定

自适应选择：在需要装配时开启，开启后拖动变重；

阻尼参数：

- 平动方向：建议设置参数在 [100-200] 之间；
- 转动方向：建议设置参数在 [3-10] 之间，其中 RZ 方向设置范围在 [0.1-5]；
- 效果：借助传感器拖动时，增大阻尼会导致拖动困难，减小阻尼会导致拖动机器人过于轻松（建议不要太小）；
- 阻尼参数整体范围：平动 XYZ：[100-1000]；转动 RX、RY：[3-50],RZ:[2-10]；
- 最大拖动力为 50，最大拖动速度为 180。

刚度参数：均设为 0；

拖动力阈值：平动 XYZ 为 [5-10]；转动 RX、RY、RZ 为 [0.5-5]；

重要： 通过加大平动方向 XYZ 或转动方向 RX、RY、RZ 的力阈值来实现锁定的方式。

1.3.7.8.4 力/扭矩传感器碰撞检测

指令说明：“FT_Guard”指令为碰撞检测指令。选择对应的传感器坐标系，勾选生效的力矩方向检测，设置当前值，碰撞最大阈值和碰撞最小阈值三项，碰撞检测条件正常范围为（当前值-最小阈值，当前值 + 最大阈值），将“开启”和“关闭”指令加入到程序中在。

图表 4.8-5 FT_Guard 指令编辑

程序示例：

序号	指令格式	注释
1	FT_Guard(1,1,1,1,1,0,0,0,5,0,0,0,0,0,10,0,0,0,0,0,5,0,0,0,0,0)	#力/矩碰撞检测开启
2	PTP(template1,100,-1,0)	#运动指令
3	FT_Guard(0,1,1,1,1,1,0,0,0,5,0,0,0,0,0,10,0,0,0,0,0,5,0,0,0,0,0)	#力/矩碰撞检测关闭

1.3.7.8.5 力/扭矩传感器力控运动

指令说明：“FT_Control”指令为力控运动指令，可以使机器人在设定力的附近运动，常用于打磨场景中。选择对应的传感器坐标系，勾选生效的力矩方向检测，设置检测阈值，以及各个方向上 PID 比例系数 (一般设置 p 为 0.001)，设置最大调整距离（对应 X,Y,Z）和最大调整角度（对应 RX,RY,RZ)，将“开启”和“关闭”指令加入到程序中在。

图表 4.8-6 FT_Control 指令编辑

程序示例：

1.3.7.8.6 力/扭矩传感器螺旋插入

指令说明：“FT_Spiral”指令为螺旋线探索插入，一般用于圆柱轴的轴孔装配动作。在运行动作之前，需要将机器人末端拖动至孔位的大致位置，根据当前场景，设定指令的参数，添加到程序中，运行后，机器人会以螺旋形的运动进行探索。

图表 4.8-7 FT_Spiral 指令编辑

程序示例：

1.3.7.8.7 力/扭矩传感器旋转插入

指令说明：“FT_Rot”指令为旋转探索插入，一般用于承接螺旋线插入动作，用于键轴的轴孔装配。在运行动作之前，需要将机器人末端移动至螺旋线探索找到的孔位或者完全对齐的示教孔位，根据当前场景，设定指令的参数，添加到程序中，运行后，机器人会以缓慢的旋转进行探索。

图表 4.8-8 FT_Rot 指令编辑

程序示例：

序号	指令格式	注释
1	FT_Control(1,11,1,0,1,0,0,0,10,0,5,0,0,0,0.001,0,0,0,0,0,0,0,0,10,5)	#力/矩运动控制开启
2	Lin(template3,100,-1,0,0)	#运动指令
3	FT_Control(0,11,1,0,1,0,0,0,10,0,5,0,0,0,0.001,0,0,0,0,0,0,0,0,10,5)	#力/矩运动控制关闭

序号	指令格式	注释
1	FT_Control(1,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制开启
2	FT_SpiralSearch(0,0.7,0,60000,5)	#螺旋插入
3	FT_Control(0,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制关闭

序号	指令格式	注释
1	FT_Control(1,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制开启
2	FT_RotInsertion(0,3,0,5,1,0,1)	#旋转插入
3	FT_Control(0,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制关闭

1.3.7.8.8 力/扭矩传感器直线插入

指令说明：“FT_Lin”指令为旋转探索插入，一般用于承接螺旋线插入动作或旋转插入动作，用于键轴的轴孔装配。在运行动作之前，需要将机器人末端移动至螺旋线探索找到的孔位，旋转插入动作结束的位置或者完全对齐的示教孔位，根据当前场景，设定指令的参数，添加到程序中，运行后，机器人会以设定的方向进行直线运动。

图表 4.8-9 FT_Lin 指令编辑

程序示例：

1.3.7.8.9 力/扭矩传感器表面定位

指令说明：“FT_FindSurface”指令为表面定位，一般用于寻找物体表面。根据当前场景，设置对应坐标系，移动方向、移动轴、探索直线速度、探索直线加速度、最大探索距离、动作终止力阈值等参数，添加到程序中，运行程序，动作开始执行，机器人末端开始缓慢向表面所在方向移动。

图表 4.8-10 FT_FindSurface 指令编辑

程序示例：

序号	指令格式	注释
1	FT_Control(1,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制开启
2	FT_LinInsertion(0,50,1,0,100,1)	#直线插入
3	FT_Control(0,10,0,0,1,0,0,0,0,0,5,0,0,0,0,0.0005,0,0,0,0,0,0,10,0)	#力/矩运动控制关闭

序号	指令格式	注释
1	PTP(1,30,-1,0)	#初始位置
2	FT_FindSurface(0,1,3,1,0,100,5)	#平面定位

1.3.7.8.10 力/扭矩传感器中心定位

指令说明：“FT_CalCenter”指令为中心定位，一般用于寻找两表面的中间平面表面。根据当前场景，设置对应坐标系，移动方向、移动轴、探索直线速度、探索直线加速度、最大探索距离、动作终止力阈值等参数，分别寻找 A 平面和 B 平面，添加到程序中，运行程序，动作开始执行，机器人缓慢向表面 A 所在方向移动，定位到 A 面后，机器人缓慢向表面 B 所在方向移动，定位到 B 面后，即可算出中心平面位置。

图表 4.8-11 FT_CalCenter 指令编辑

程序示例：

1.3.7.8.11 力/扭矩传感器点按力探测

指令说明：“FT_Click”指令为点按力探测，点按力探测用于探测一个点按力，通常和表面定位动作配合使用。设置好参数后，添加到程序中，运行程序，末端开始沿工具坐标系 Z 方向向目标移动，当 Z 正方向上的力达到点按力数值，则点按力探测完成。

图表 4.8-12 FT_Click 指令编辑

程序示例：

序号	指令格式	注释
1	PTP(1,30,-1,0)	#初始位置
2	FT_CalCenterStart()	#表面定位开始
3	FT_Control(1,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)	#力/矩运动控制开启
4	FT_FindSurface(1,2,2,10,0,200,5)	#定位平面 A
5	FT_Control(0,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)	#力/矩运动控制关闭
6	PTP(1,30,-1,0)--初始位置	#初始位置
7	FT_Control(1,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)	#力/矩运动控制开启
8	FT_FindSurface(1,1,2,20,0,200,5)	#定位平面 B
9	FT_Control(0,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)	#力/矩运动控制关闭
10	pos = {}	#定义数组 pos
11	pos = FT_CalCenterEnd()	#获取定位中心笛卡尔位姿
12	MoveCart(pos,GetActualTCPNum(),GetActualWObjNum(),30,10,100,-1,0)	#运动至定位的中心位置

序号	指令格式	注释
1	PTP(1,30,-1,0)	#初始位置
2	FT_Click(0,5,5,0,100,0)	#点按力探测

1.3.7.9 扩展 IO 设备外设配置

1.3.7.9.1 扩展 IO 设备配置步骤

Step1：在用户外设配置界面中选择“末端外设配置”按钮，设备类型选择“扩展 IO 设备”，扩展 IO 设备配置信息分为厂商、类型、软件版本和挂载位置，用户可根据具体的生产需求来配置相应的设备信息。若用户需要更改配置，可先选择相应的编号，点击“清除”按钮，来清除相应的信息，并重新根据需求配置；

图表 4.9-1 扩展 IO 设备配置

重要： 点击清除配置前，相应的设备应处于未激活状态。

Step2：扩展 IO 设备配置完成后，用户可在辅助应用中点击“Smart Tool”功能菜单，进入此功能配置页面，用户可以对末端手柄上的各个按键功能进行自定义，包括（新建程序，保持程序，PTP，Lin，ARC，摆焊开始，摆焊结束，IO 端口）。

图表 4.9-2 扩展 IO 设备功能配置

1.3.7.10 码垛系统配置

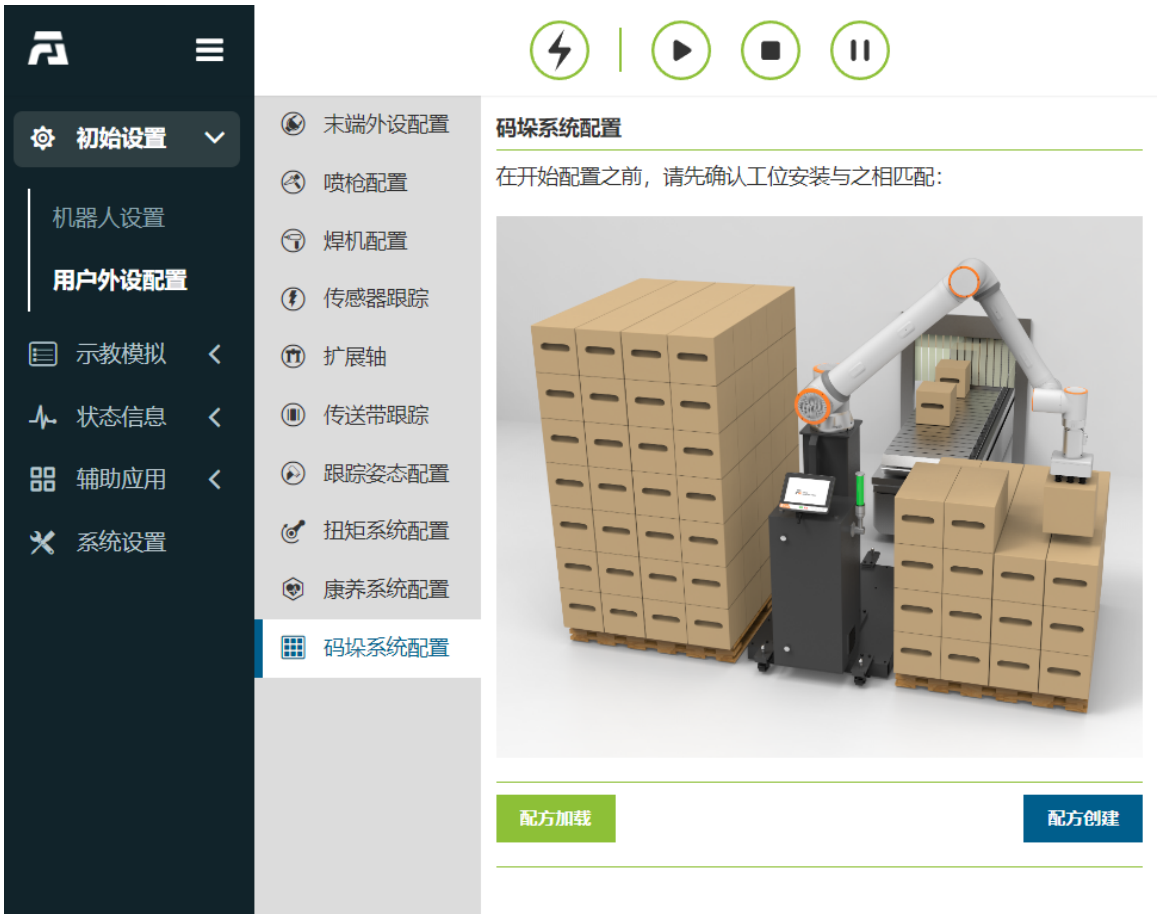
1.3.7.10.1 码垛系统配置步骤

Step1：在用户外设配置界面中选择“码垛系统配置”按钮，第一次使用，需要首先进行配方创建，点击“配方创建”，输入配方名称，点击“创建”，创建成功后点击“开始配置”进入码垛配置页面。

图表 4.10-1 码垛配方配置

Step2：在工件配置栏中点击“配置”进入工件配置弹窗，设置工件的“长度”，“宽度”，“高度”以及工件的抓取点，点击“确认配置”完成工件信息设置。


Smart Tool 配置
功能配置
A 键 : PTP
点 速 度 : 30 %
配置
B 键 : LIN
点 速 度 : 30 %
配置
C 键 : ARC
点 速 度 : 20 %
配置
D 键 : 新建程序
配置


初始设置
机器人设置
用户外设配置
示教模拟
状态信息
辅助应用
系统设置
末端外设配置
喷枪配置
焊机配置
传感器跟踪
扩展轴
传送带跟踪
跟踪姿态配置
扭矩系统配置
康养系统配置
码垛系统配置
码垛系统配置
在开始配置之前，请先确认工位安装与之相匹配:
配方加载
配方创建


工件配置
长度:
400
mm
宽度:
300
mm
高度:
300
mm
height
B
A
width
Length
工件抓取位置示教
设置点
确认配置

图表 4.10-2 码垛工件配置

Step3：在托盘配置栏中点击“配置”进入托盘配置弹窗，设置托盘“前边”，“侧边”和“高度”，接着设置工位和工位过渡点，点击“确认配置”完成托盘信息设置。

图表 4.10-3 码垛托盘配置

Step4：在码垛设备尺寸配置栏中点击“配置”进入码垛设备尺寸配置配置弹窗，设置设备“X”、“Y”、“Z”和“Angle”，点击“确认配置”完成码垛设备尺寸配置信息设置。

重要： X、Y、Z 为做托盘右上角或者右托盘左上角点相对于机器人基坐标系坐标值的绝对值，Angle 为机器人安装时的旋转角度，推荐安装时为 0。

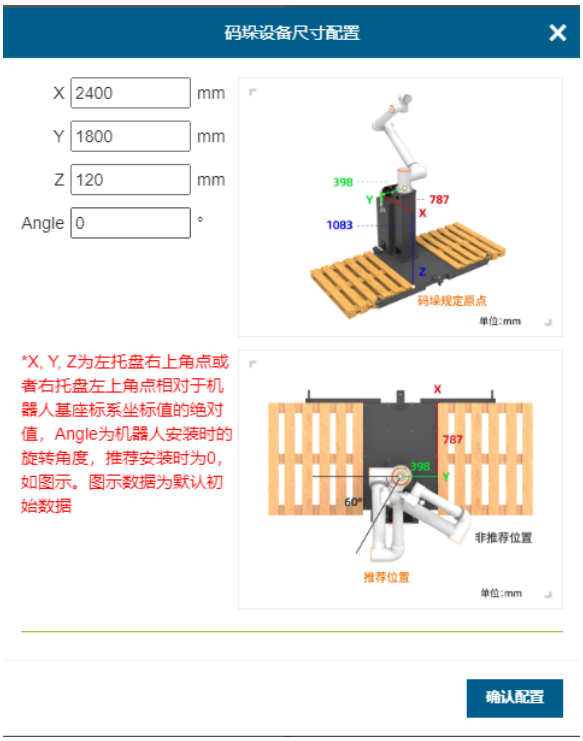
图表 4.10-4 码垛设备尺寸配置

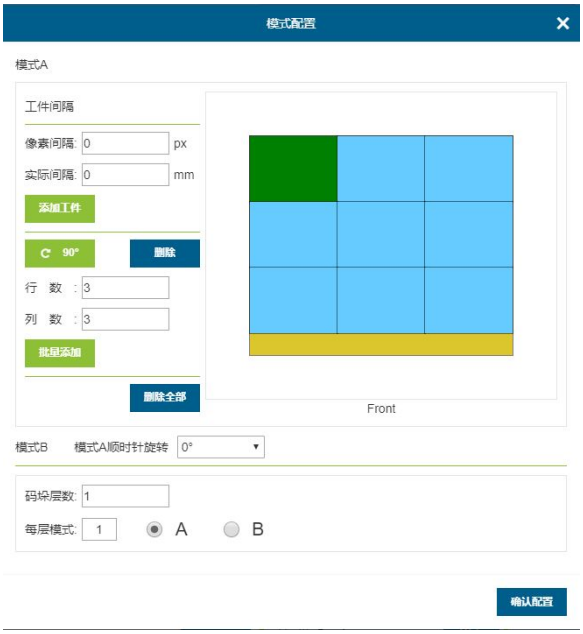
Step5：在模式配置栏中点击“配置”进入模式配置弹窗，设置工件间隔，右侧框框为模拟工件放置方式，可以单个添加也可以批量添加。接着设置码垛层数和各层的模式，点击“确认配置”完成模式信息设置。

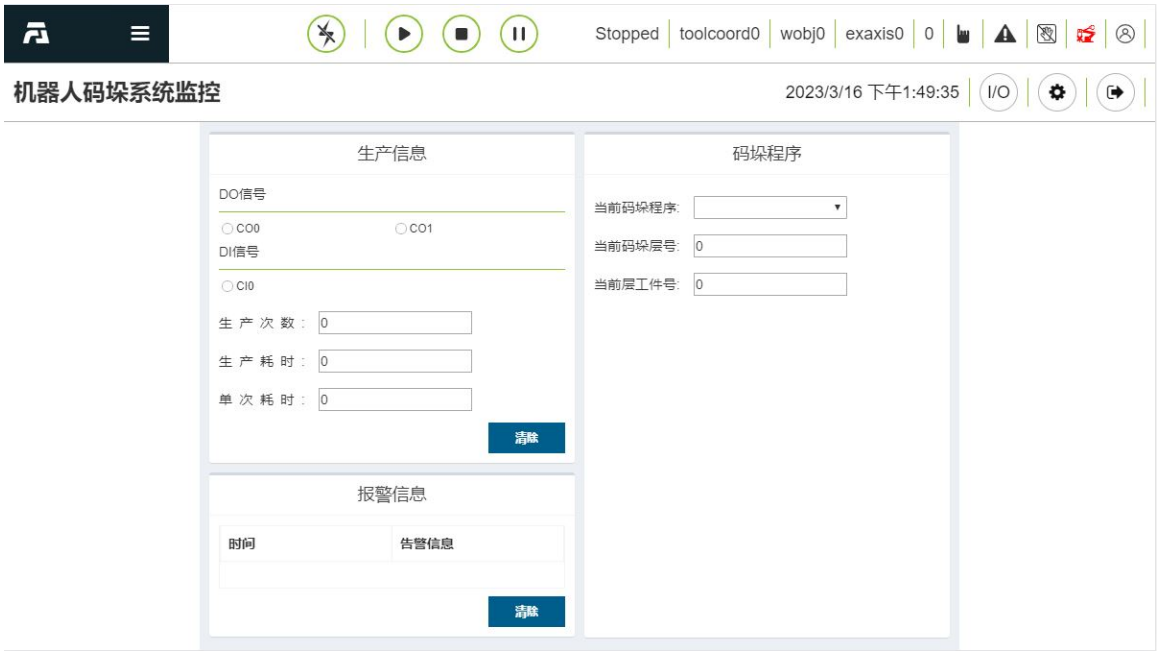
图表 4.10-5 码垛模式配置

Step6：在示教程序生成栏选择“方式选择”，点击“生成程序”，打开“码垛监控页”，在此页面可以对“生成信息”，“报警信息”和“码垛程序”显示和查看。

图表 4.10-6 码垛系统监控


码垛设备尺寸配置
X 2400 mm
Y 1800 mm
Z 120 mm
Angle 0 °
398
787
1083
码垛规定原点
单位:mm
*X, Y, Z为左托盘右上角点或者右托盘左上角点相对于机器人基座标系坐标值的绝对值，Angle为机器人安装时的旋转角度，推荐安装时为0，如图示。图示数据为默认初始数据
787
398
60°
非推荐位置
推荐位置
单位:mm
确认配置


模式配置
模式A
工件间隔
像素间隔: 0 px
实际间隔: 0 mm
添加工件
90°
删除
行 数 : 3
列 数 : 3
批量添加
删除全部
Front
模式B
模式A顺时针旋转 0°
码垛层数: 1
每层模式: 1 A B
确认配置



1.3.7.11 打磨设备配置

1.3.7.11.1 打磨设备配置步骤

Step1: 进入打磨设备配置页面，配置通讯信息，需要配置 IP 地址、端口、采样周期和通信协议。配置成功后，下次操作可自动显示。

Step2：完成通信配置后，可建立通信，通过加载/卸载打磨设备。

Step3：设备功能。可进行设备使能、错误清除和力传感器清零等操作。

Step4：参数配置。可设置打磨设备的转速、接触力、伸出距离和控制模式。设置成功后，可在右侧”Polish”状态反馈栏显示相应数据和状态。

图表 4.11-1 打磨状态配置页面

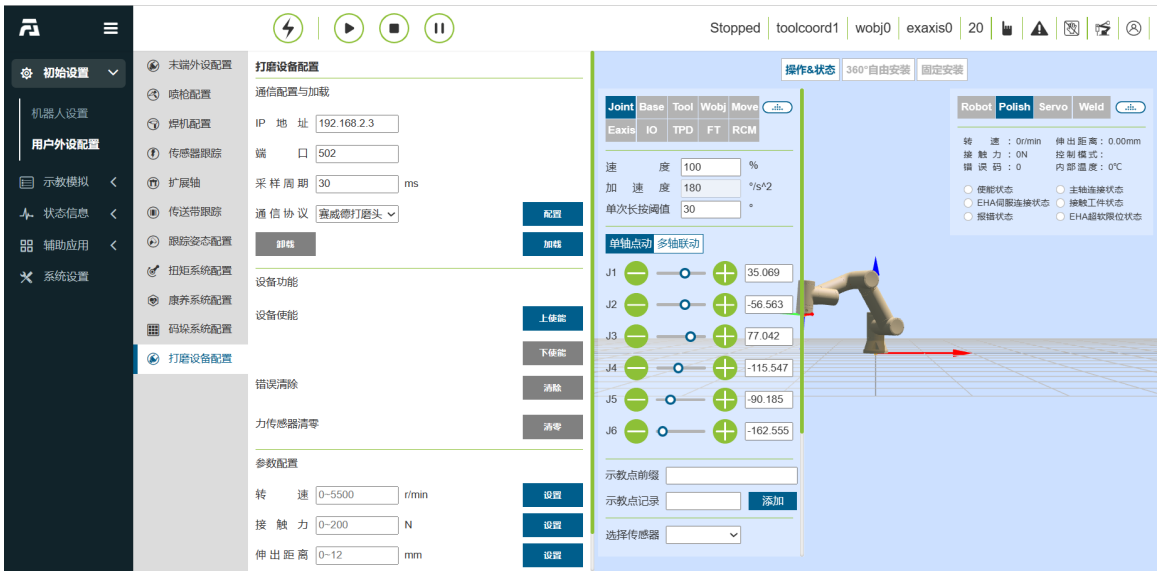
1.3.8 附录

1.3.8.1 附录 1：运动控制器错误及处理方式

1.3.8.2 附录 2：伺服驱动器故障代码表

1.3.8.3 附录 3：末端板 485 升级

现场使用时，有可能更新固件，满足新的要求，会提供新的升级文件（XX_XX_MAIN.bin），通过 485 接口对末端板进行升级（需要借助 USB 转 485 模块）。升级步骤如下：



Step1：485 接线，在机器人末端处有 5Pin 通信航空接头，航空接头 Pin 脚分布及其 pin 脚说明如图表 1 所示。将机器人末端的 485-A、485-B 与 USB 转 485 工具的 A、B 使用双绞线连接。

图表 3-1 航空接头 Pin 脚分布

Step2，硬件连接，将 USB 转 485 工具的 USB 端与 PC 连接，在 PC 设备管理器中，如果识别到 USB&485 工具，会出现如下界面。

图表 3-2 USB&485 端口识别说明

Step3：升级工具，在完成接线后，打开“法奥串口调试助手”，点击“末端板”按钮，在“串口参数设置”功能中选择上述识别的串口，波特率 115200，数据位 8 位，校验位无，停止位 1，然后打开串口，成功之后会出现“串口打开成功”的提示。

图表 3-3 串口参数设置

Step4：固件升级，选择“末端板”，点击“固件升级”，如图表所示：

图表 3-4 末端板固件升级

- 首先点击“Flash 擦除”，擦除成功之后，会在接收数据区提示擦除成功。
- 打开文件（待升级文件），选择存放的路径，如下所示，选择完成后，待升级文件名会出现在文件名显示框中。

图表 3-5 选择升级文件

- 点击“发送文件”，当进度条显示 100% 时，表示已经完成发送升级文件。

Step5：升级验证，系统重启上电，在“维护信息”栏，选择“查询末端板固件版本信息”，在“接收数据区”会显示固件版本信息，如果和升级的文件版本信息一致，说明升级成功，否则升级失败.

图表 3-6 查询固件版本信息

错误分类	错误名（示教器显示）	处理方式	处理后操作
指令点错误	关节指令点错误	1. 检查指令点是否有误； 2. 检查工具是否与记录指令点时一致； 3. 重新记录指令点，修改示教程序；	检查确认，修改程序后，重新点击 START 按钮即可执行新程序；
	直线目标点错误		
	圆弧中间点错误		
	圆弧目标点错误		
	圆弧指令点间距过小		
	整圆/螺旋线中间点 1 错误（包括工具不符）		
	整圆/螺旋线中间点 2 错误（包括工具不符）		
	整圆/螺旋线中间点 3 错误（包括工具不符）		
	整圆/螺旋线指令点间距过小		
	TPD 指令点错误		
	TPD 指令工具与当前工具不符		
	TPD 当前指令与下一指令起始点偏差过大	1. 检查是否记录 TPD 轨迹起点，示教程序中 TPD 指令前是否有运动指令(PTP/LIN)运动到 TPD 轨迹的起点； 2. 修改示教程序；	
	内外部工具切换错误	1. 重新记录指令点，修改示教程序；	

	PTP 关节指令超限	1. 运动过程指令超出软限位，需要修改程序；	反向点动，使机器人离开软限位区域或切换到拖动模式，拖动机器人离开软限位区域； 修改程序后，重新点击 START 按钮即可执行新程序；
	TPD 关节指令超限	1. 运动过程指令超出软限位，需要修改程序；	
	LIN\ARC 下发关节指令超限	1. 运动过程指令超出软限位，需要修改程序；	
	JOG 关节指令超限	1. 运动过程指令超出软限位；	反向点动，使机器人离开软限位区域或切换到拖动模式，拖动机器人离开软限位区域；
	笛卡尔空间内指令超速	1. 指令存在问题；	联系售后人员；
	关节空间内扭矩指令超限		
	轴 1-轴 6 关节空间内指令速度超限		
	下一指令关节配置发生变化（下一指令中存在奇异位姿）	1. 修改示教程序，运动停止时当前行的下一行程序修改为 PTP 指令；	
	当前指令关节配置发生变化（当前指令中存在奇异位姿）	1. 自动模式时，修改示教程序，运动停止时当前行程序修改为 PTP 指令； 2. 手动模式时，重新上电后切换关节坐标再进行点动；	
	指令错误，ARCSTART 和 ARCEND 间只允许 LIN 和		

	ARC 指令	1. 检查指令，修改示教程序；	
	指令错误，WEAVESTART 和 WEAVEEND 间只允许 LIN 指令		
	摆焊参数错误	1.检查摆焊参数，修改示教程序；	
	激光传感器指令偏差过大	1. 使用激光传感器配套软件，检查激光传感器数据是否有跳变；	
	激光传感器指令中断，焊缝跟踪提前结束	1. 使用激光传感器配套软件，检查激光传感器数据是否中断；	
	外部轴指令速度超限	1. 指令存在问题；	联系售后人员；
	外部轴指令与反馈偏差过大		
	传送带跟踪-起始点与参考点姿态变化过大	1. 重新标定传送带位姿点；	
	恒力控制-（X,Y,Z,RX,RY,RZ）方向超过最大调整距离	1. 调整恒力控制 PID 数值，修改示教程序；	
驱动器故障	1-6 轴驱动器故障	1. 驱动器发生故障，查看驱动器故障表，检查驱动器故障原因；	重新上电；
超出软限位	1-6 轴超出软限位故障	1. 运动过程超出软限位；	切换到拖动模式，拖动机器人离开软限位区域；
碰撞故障	1-6 轴碰撞故障	1. 运动过程发生碰撞；	检查碰撞原因，需要再开始运动时点击示教编程界面 RESUME 按

			钮；
文件错误	zbt 配置文件版本错误	1. zbt.config 配置文件版本错误；	联系售后人员；
	zbt 配置文件加载失败	1. zbt.config 配置文件加载错误；	重新上电，仍报错的话联系售后人员；
	user 配置文件版本错误	1. user.config 配置文件版本错误；	联系售后人员；
	user 配置文件加载失败	1. user.config 配置文件加载错误；	重新上电，仍报错的话联系售后人员；
	exaxis 配置文件版本错误	1. exaxis.config 配置文件版本错误；	联系售后人员；
	exaxis 配置文件加载失败	1. exaxis.config 配置文件加载错误；	重新上电，仍报错的话联系售后人员；
	机器人型号不一致，需要重新设置		联系售后人员；
IO 错误	通道错误	1. 检查程序中 IO 指令通道是否设置错误；	检查确认，修改程序后，重新点击 START 按钮即可执行新程序；
	数值错误	1. 检查程序中 IO 指令数值是否设置错误；	
	WaitDI 等待超时	1. 检查是否有有效的输入信号；	
	WaitAI 等待超时		
	WaitToolDI 等待超时		
	WaitToolAI 等待超时		
	通道已配置功能错误	1. 当前通道已配置功能，示教编程中不能使用，需更换为其他通道；	当前通道已配置功能，示教编程中不能使用，需更换为其他通

			道；
	起弧超时	1. 检查起弧成功信号；	
	收弧超时	1. 检查收弧成功信号；	
	寻位超时	1. 调整寻位参数，修改示教程序；	
	传送带 IO 检测超时，可复位	1. 检查 IO 触发信号；	
	起弧成功 DI 未配置	1. 检查 DI 配置；	
夹爪错误	夹爪运动超时错误	1. 等待夹爪运动完成信号超时；	联系售后人员；
	485 超时		
	指令格式错误		
	运动延迟、运动前必须先激活		
	运动时，激活位必须为激活		
	温度过高		
	电压过低		
	正在自动释放		
	内部故障		
	激活失败		
	过流		
	自动释放结束		
警告	肩关节配置变化	暂未实现；	
	肘关节配置变化		
	腕关节配置变化		
	RPY 初始化失败	1. RX、RY、RZ 初始化失	移动机器人后，

		败，移动机器人，使 RY 不等于±90°；	重新上电；
	1min 后切入普通示教模式警告		
活动从站数量错误	活动从站数量错误	1. 从站出现错误，检查 EtherCAT 线缆是否正常；	重新上电，若未解决，联系售后人员；
从站错误	从站掉线		
	从站状态与设置值不一致		
	从站未配置		
	从站配置错误		
	从站初始化错误		
	从站邮箱通信初始化错误		
安全门警告	安全门触发		
运动警告	LIN 指令姿态变化过大	暂未实现；	
干涉区警告	进入干涉区		
参数错误	工具号超限错误	1. 导出备份包；	联系售后人员
	定位完成阈值错误		
	碰撞等级错误		
	负载重量错误		
	负载质心（X,Y,Z）错误		
	DI 滤波时间错误		
	AxleDI 滤波时间错误		
	AI 滤波时间错误		
	AxleAI 滤波时间错误		
	DI 高低电平范围错误		
	DO 高低电平范围错误		

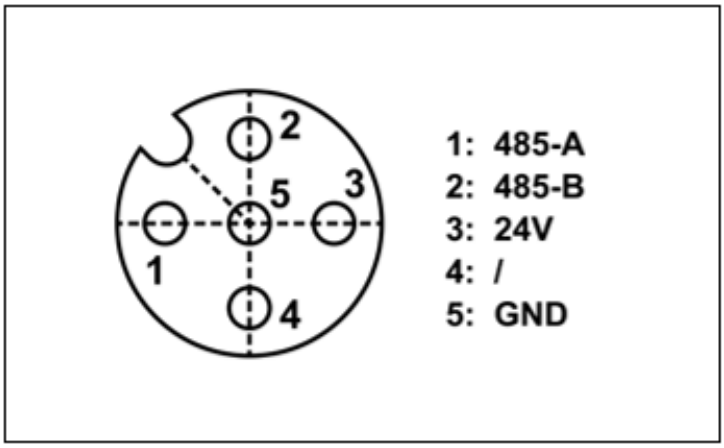
	工件号超限错误		
	外部轴号超限错误		
	传送带跟踪-编码器通道错误		
	传送带跟踪-工件轴号错误		
八点法	第 1 组数据姿态变化过大	1. 重新标定；	
	数据错误，计算失败		
按钮盒状态反馈	记录点信号		
	开始运行		
	停止运行		
外部轴超出软限位故障	外部轴 1-4 轴超出软限位故障	1. 运动过程超出软限位；	反向点动外部轴，离开软限位区域；
示教器（树莓派）通信	Webapp 与示教器（树莓派）通信失败	1.检查是否需要启用示教器（树莓派），若不需要启用，进入 Web 页面，系统设置—通用设置—示教器设置，关闭示教器启用； 2.检查示教器（树莓派）与控制器网线连接是否正常；	
原点错误	原点已发生改变，需要重新设置原点	1.进入 Web 页面，辅助应用—机器人本体—作业原点，重新设置作业原点；	

故障码	故障名称	处理方法
1	软件过流故障	1、检查关节负载或阻力是否变大或异常 2、若故障仍未排除，维修或更换驱动板
2	过压故障	降低机器人运行速度或加速度
3	欠压故障	1、检查控制箱 48V 电源电压输出是否异常 2、检查驱动板和关节外壳是否短路 3、若故障仍未排除，维修或更换驱动板
4	过热故障	减小机器人负载或降低机器人运行速度
5	过载故障	减小机器人负载或降低机器人运行速度
6	超速故障	1、检查磁编组件和电机轴固定顶丝是否松动 2、重新进行编码器校零 3、若故障仍未排除，维修或更换磁编组件
7	参数异常故障	维修或更换驱动板
8	飞车故障	1、检查磁编组件和电机轴固定顶丝是否松动 2、重新进行编码器校零 3、若故障仍未排除，维修或更换磁编组件
9	位置误差故障	1、检查关节负载或阻力是否变大或异常 2、若故障仍未排除，维修或更换驱动板
10	位置溢出故障	1、检查硬限位是否松动 2、重新进行机器人校零
11	硬件过流故障	维修或更换驱动板
12	驱动禁止故障	未启用
13	电机堵转故障	1、检查刹车电磁铁是否吸合 2、检查是否撞到硬限位

		3、若故障仍未排除，维修或更换驱动板
14	功率电源故障	未启用
15	STO 故障	未启用
16	相电流 AD 调零故障	维修或更换驱动板
17	EEPROM 故障	维修或更换驱动板
18	霍尔故障	1、检查霍尔线束是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换关节
19	编码器故障	维修或更换磁编组件
20	编码器调零故障	1、重新进行编码器校零 2、若故障仍未排除，维修或更换磁编组件
21	编码器 Z 相信号丢失故障	未启用
22	编码器计数故障	未启用
23	编码器多圈数据溢出故障	未启用
24	外部时钟故障	维修或更换驱动板
25	UVW 相序故障	未启用
26	FPGA 故障	未启用
27	回零故障	未启用
28	磁编码器故障	1、检查磁编组件和电机轴固定顶丝是否松动 2、若故障仍未排除，维修或更换磁编组件
29	电机动力线断线故障	1、检查电机动力线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板
30	EtherCAT 故障	1、检查网线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板

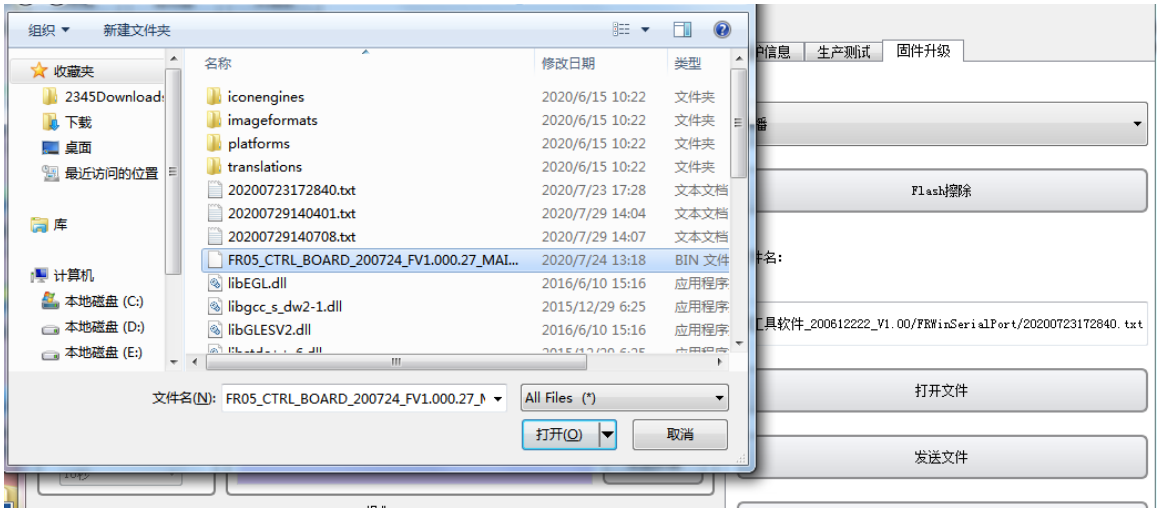
31	EtherCAT_SM_DOG 故障	1、检查网线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板
32	EtherCAT_FATALSYNC 故障	1、检查网线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板
33	EtherCAT_SYNC 故障	1、检查网线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板
34	EtherCAT_RFT 故障	1、检查网线是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换驱动板
35	驱动器轴地址故障	1、重新进行驱动器轴地址配置 2、若故障仍未排除，维修或更换驱动板
36	机器人校零故障	1、 重新进行机器人校零 2、 先使用 JLINK 擦除 FLASH，再重新下载程序并校零 3、若故障仍未排除，维修或更换驱动板
37	编码器通讯故障	1、检查编码器线束是否插接牢固，有无短路、断路 2、若故障仍未排除，维修或更换磁编组件
40	磁编模块故障-校零故障	1、 重新进行磁编组件校零 2、 若故障仍未排除，维修或更换磁编组件
41	磁编模块故障-多圈故障	1、 检查磁编组件和电机轴固定顶丝是否松动 2、 若故障仍未排除，维修或更换磁编组件
42	磁编模块故障-多圈小磁编故障	1、 检查多圈小磁编芯片是否异常 2、 若故障仍未排除，维修或更换磁编组件
43	磁编模块故障-多圈大磁编故障	1、 检查多圈大磁编芯片是否异常 2、 若故障仍未排除，维修或更换磁编组件
44	磁编模块故障-单圈磁编故障	1、 检查单圈磁编芯片是否异常 2、 若故障仍未排除，维修或更换磁编组件

45	磁编模块故障-光编故障	1、 检查光编码盘是否被污染或未粘牢 2、 若故障仍未排除，维修或更换磁编组件




串口参数设置
串口号:
COM2
波特率:
115200
数据位:
8
校验位:
无
停止位:
1
打开串口

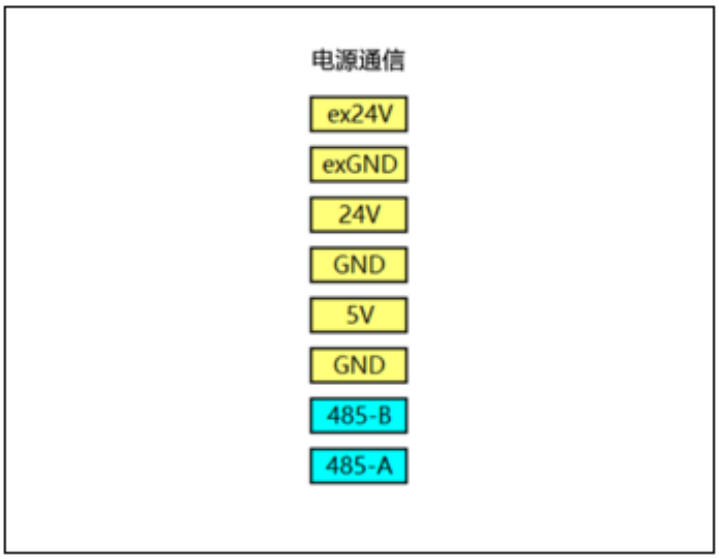

控制箱
驱动器
末端板
串口参数设置
串口号:
COM2
波特率:
115200
数据位:
8
校验位:
无
停止位:
1
关闭串口
数据设置
十六进制数据发送
十六进制数据接收
启用自动发送数据
10秒
启用自动保存数据
10秒
接收数据区
发送数据
发送数据
操作
发送计数清零
接收计数清零
停止显示
清空显示
手动保存数据
关于程序
维护信息
生产测试
固件升级
单播
Flash擦除
文件名:
打开文件
发送文件
跳转BOOT


组织
新建文件夹
收藏夹
2345Download
下载
桌面
最近访问的位置
库
计算机
本地磁盘 (C:)
本地磁盘 (D:)
本地磁盘 (E:)
名称
修改日期
类型
iconengines 2020/6/15 10:22 文件夹
imageformats 2020/6/15 10:22 文件夹
platforms 2020/6/15 10:22 文件夹
translations 2020/6/15 10:22 文件夹
20200723172840.txt 2020/7/23 17:28 文本文档
20200729140401.txt 2020/7/29 14:04 文本文档
20200729140708.txt 2020/7/29 14:07 文本文档
FR05_CTRL_BOARD_200724_FV1.000.27_MAI... 2020/7/24 13:18 BIN 文件
libEGL.dll 2016/6/10 15:16 应用程序
libgcc_s_dw2-1.dll 2015/12/29 6:25 应用程序
libGLESV2.dll 2016/6/10 15:16 应用程序
文件名(N): FR05_CTRL_BOARD_200724_FV1.000.27_M
All Files (*)
打开(O)
取消
生产测试
固件升级
Flash擦除
打开文件
发送文件


末端板
接收数据区
发送数据
发送数据
维护信息
生产测试
固件升级
单播
查询末端板固件版本信息
读取系统运行日志
查询Ecat链路信息

1.3.8.4 附录 4：控制箱 485 升级

在机器人控制箱板有“电源通信”接口，将 USB&485 工具 A、B 分别接入其接口的“485-A”、“485-B”。其升级过程操作同末端板，软件对应选择即可，此处不在赘述。



图表 4-1 电源通信接口

1.3.8.5 附录 5：备件、易损件清单

零件	编号	数量
M8*30 **螺钉**	4.0.08.2006185	4
圆柱销 A **型** 8*20	4.5.00.2013076	2
保险丝 5x20 6A		1

1.3.9 术语

停止类别：

- **0 类停机**：当机器人的电源被切断后，机器人立刻停止工作。这是不可控的停止，由于每个关节会以最快的速度制动，因此机器人可能偏离程序设定的路径。当超过安全评定极限，或当控制系统的安全评定部分出现错误的情况下方可使用这种保护性停止。要了解更多信息，请参阅 ENISO13850:2008 或 IEC60204-1:2006。

- **1 类停机**：当为机器人供电使其停止时，机器人就停止，当机器人实现停止后切断电源。这是可控性停止，机器人会遵循程序编制的路径。一秒钟之后或一旦机器人停稳后就将电源切断。要了解更多信息，请参阅 ENISO13850:2008 或 IEC60204-1:2006。
- **2 类停机**：机器人通电时的可控性停止。机器人在一秒钟时间内停止所有动作。安全评定控制系统的操控可使机器人停留在停止的位置。要了解更多信息，请参阅 IEC60204-1:2006。

诊断覆盖率（DC）：用于衡量为了达到评定的性能等级而实施的诊断的有效性。要了解更多信息，请参阅 ENISO13849-1:2008。

集成商：集成商即设计机器人最终安装的机构。集成商负责进行最终风险评估，必须确保最终安装遵循当地的法律法规。

平均危险失效时间（MTTFd）：平均危险失效时间（MTTFd）指的是为了达到评定的性能等级而进行计算和检测所得的值。要了解更多信息，请参阅 ENISO13849-1:2008。

风险评估：风险评估即识别所有风险并将风险降低到适当程度的整个过程。风险评估应进行记录存档。详情请参考 ISO12100。

性能等级：性能等级（Performance Level,PL）是一个分离的等级，它用于说明控制系统中各个与安全相关的部分在可预测的条件下执行安全功能的能力。PLd 是第二高的可信度分类，它意味着安全功能相当值得信赖。要了解更多信息，请参阅 EN ISO13849-1:2008。

连接法兰：用于连接外部工具的结构，一般称为法兰盘。

机器人末端：机器人最后一个轴或连接法兰的中心点。

工具中心点（TCP）：工具中心点即机器人工具的特征点，是机器人系统的控点，出厂时默认于最后一个运动轴或连接法兰的中心。每个工具的工具中心点都包含相对于工具输出法兰中心而设定的转换和旋转。位置坐标 X、Y、Z 决定了工具中心点的位置，RX、RY、RZ 决定了工具中心的方向。当其值均为零时，工具中心与连接法兰中心点重合。

工具位姿点（TCF）：在工具中心点 TCP 的基础上，反应工具坐标系相对于末端连杆坐标系的姿态。

基坐标系：基坐标系的原点一般定义在机器人第一轴与安装面的中心点，x 轴向前，在轴向上，y 轴按右手规则确定。

世界坐标系：建立在工作单元或工作站中的固定坐标系。当只有一个机器人时，该坐标系可认为与基坐标系重合；当有多个机器人或外部设备时，世界坐标系可为这些设备提供一个唯一的参考系，在满足方便标定其他设备的坐标系前提下，其具体位置可以任意指定。

关节坐标系：关节坐标系是机器人关节中的坐标系，在关节坐标系下，机器人各轴均可实现在限位范围内的单独的正向或反向运动。适用于需要对机器人进行大范围的运动且不要求机器人 TCP 姿态。机器人手动模式下的单轴点动就是在关节坐标系下进行的。

工具坐标系：用于定义工具中心点的位置和工具姿态的坐标系，未定义时，工具坐标系默认在连接法兰的中心处。安装工具后，TCP 将发生变化，变为工具末端的中心。

外部工具坐标系：用于定义固定在机器人外部的工具位姿的坐标系。

扩展轴：除去机器人本体上的轴，为了工作需要额外增加的轴，扩展轴主要包含滑轨，翻转台和外加伺服设备等类型。

手动模式：在该模式下，机器人的所有运动均由用户手动控制，并且安全光栅、安全门等外部安全设施不起作用，以便近距离调试。

自动模式：该模式一般用于机器人运行示教程序，此时外部安全设施启用。

重复定位精度：机器人在同一条件下，用同一方法操作时，重复 n 次所测得的位置与姿态的一致程度。

示教器：对机器人进行编程或使机器人运动，并与控制系统相连接的手持式单元。

CHAPTER 2

SDK

2.1 C++

本文档为 C++ 版本的二次开发接口文档。

重要： 机器人参数单位说明：机器人位置单位为毫米 (mm)，姿态单位为度 (°)。

重要：

1) 非特别说明的代码示例中都默认机器人已经正常开机使能；
2) 文档中的所有代码示例都默认在机器人的工作空间内没有任何干涉；
3) 实际使用测试时请采用现场机器人的数据使用。

备注： 当前文档适用于 SDK-v2.0.0 版本，向下兼容 v1.x 版本。

2.1.1 数据结构说明

2.1.1.1 接口调用返回值类型

```
typedef  int errno_t;
```

2.1.1.2 关节位置数据类型

```
/**
* @brief 关节位置数据类型
*/
typedef  struct
{
    double jPos[6];    /* 六个关节位置, 单位deg */
}JointPos;
```

2.1.1.3 笛卡尔空间位置数据类型

```
/**
* @brief 笛卡尔空间位置数据类型
*/
typedef struct
{
    double x;     /* x轴坐标, 单位mm  */
    double y;     /* y轴坐标, 单位mm  */
    double z;     /* z轴坐标, 单位mm  */
} DescTran;
```

2.1.1.4 欧拉角姿态数据类型

```
/**
* @brief 欧拉角姿态数据类型
*/
typedef struct
{
    double rx;    /* 绕固定轴X旋转角度, 单位: deg  */
    double ry;    /* 绕固定轴Y旋转角度, 单位: deg  */
    double rz;    /* 绕固定轴Z旋转角度, 单位: deg  */
} Rpy;
```

2.1.1.5 笛卡尔空间位姿数据类型

```
/**
*@brief 笛卡尔空间位姿类型
*/
typedef struct
{
    DescTran tran;      /* 笛卡尔空间位置  */
    Rpy rpy;            /* 笛卡尔空间姿态  */
} DescPose;
```

2.1.1.6 扩展轴位置数据类型

```
/**
* @brief 扩展轴位置数据类型
*/
typedef  struct
{
    double ePos[4];    /* 四个扩展轴位置，单位mm */
}ExaxisPos;
```

2.1.1.7 力矩传感器数据类型

```
/**
* @brief 力传感器的受力分量和力矩分量
*/
typedef struct
{
    double fx;  /* 沿x轴受力分量，单位N  */
    double fy;  /* 沿y轴受力分量，单位N  */
    double fz;  /* 沿z轴受力分量，单位N  */
    double tx;  /* 绕x轴力矩分量，单位Nm */
    double ty;  /* 绕y轴力矩分量，单位Nm */
    double tz;  /* 绕z轴力矩分量，单位Nm */
} ForceTorque;
```

2.1.1.8 螺旋参数数据类型

```
/**
* @brief  螺旋参数数据类型
*/
typedef  struct
{
    int     circle_num;           /* 螺旋圈数  */
    float   circle_angle;         /* 螺旋倾角  */
    float   rad_init;             /* 螺旋初始半径，单位mm  */
    float   rad_add;              /* 半径增量  */
    float   rotaxis_add;          /* 转轴方向增量  */
    unsigned int rot_direction;   /* 旋转方向，0-顺时针，1-逆时针  */
}SpiralParam;
```

2.1.1.9 控制器状态反馈数据包

```
/**
 * @brief  控制器状态反馈数据包
 */
typedef struct _ROBOT_STATE_PKG
{
    uint16_t frame_head;                /* 帧头，约定为0x5A5A */
    uint8_t  frame_cnt;                 /* 帧计数，循环计数0-255 */
    uint16_t data_len;                  /* 数据内容的长度 */
    uint8_t  program_state;             /* 程序运行状态，1-停止；2-运行；3-暂停； */
    uint8_t  robot_state;               /* 机器人运动状态，1-停止；2-运行；3-暂停；4-
↪拖动 */
    int      main_code;                 /* 主故障码 */
    int      sub_code;                  /* 子故障码 */
    uint8_t  robot_mode;                /* 机器人模式，1-手动模式；0-自动模式；  */
    double   jt_cur_pos[6];             /* 6个轴当前关节位置，单位deg */
    double   tl_cur_pos[6];             /* 工具当前位置
                                           tl_cur_pos[0]，沿x轴位置，单位mm，
                                           tl_cur_pos[1]，沿y轴位置，单位mm，
                                           tl_cur_pos[2]，沿z轴位置，单位mm，
                                           tl_cur_pos[3]，绕固定轴X旋转角度，单位deg
                                           tl_cur_pos[4]，绕固定轴y旋转角度，单位deg
                                           tl_cur_pos[5]，绕固定轴z旋转角度，单位deg
↪*/
    double   flange_cur_pos[6];         /* 末端法兰当前位置
                                           flange_cur_pos[0]，沿x轴位置，单位mm，
                                           flange_cur_pos[1]，沿y轴位置，单位mm，
```

(续下页)

(接上页)

```
                                        flange_cur_pos[2], 沿z轴位置, 单位mm,
                                        flange_cur_
↪pos[3], 绕固定轴X旋转角度, 单位deg
                                        flange_cur_
↪pos[4], 绕固定轴y旋转角度, 单位deg
                                        flange_cur_
↪pos[5], 绕固定轴z旋转角度, 单位deg */
    double   actual_qd[6];              /* 当前6个关节速度, 单位deg/s */
    double   actual_qdd[6];             /* 当前6个关节加速度, 单位deg/s^2 */
    double   target_TCP_CmpSpeed[2];    /* target_TCP_
↪CmpSpeed[0], TCP合成指令速度(位置), 单位mm/s
                                        target_TCP_
↪CmpSpeed[1], TCP合成指令速度(姿态), 单位deg/s  */
    double   target_TCP_Speed[6];       /* TCP指令速度
                                        target_TCP_Speed[0], 沿x轴速度, 单位mm/s,
                                        target_TCP_Speed[1], 沿y轴速度, 单位mm/s,
                                        target_TCP_Speed[2], 沿z轴速度, 单位mm/s,
                                        target_TCP_
↪Speed[3], 绕固定轴X旋转角速度, 单位deg/s
                                        target_TCP_
↪Speed[4], 绕固定轴y旋转角速度, 单位deg/s
                                        target_TCP_
↪Speed[5], 绕固定轴z旋转角速度, 单位deg/s */
    double   actual_TCP_CmpSpeed[2];    /* actual_TCP_
↪CmpSpeed[0], TCP合成实际速度(位置), 单位mm/s
                                        actual_TCP_
↪CmpSpeed[1], TCP合成实际速度(姿态), 单位deg/s */
    double   actual_TCP_Speed[6];       /* TCP实际速度
                                        actual_TCP_Speed[0], 沿x轴速度, 单位mm/s,
                                        actual_TCP_Speed[1], 沿y轴速度, 单位mm/s,
                                        actual_TCP_Speed[2], 沿z轴速度, 单位mm/s,
                                        actual_TCP_
↪Speed[3], 绕固定轴X旋转角速度, 单位deg/s
                                        actual_TCP_
↪Speed[4], 绕固定轴y旋转角速度, 单位deg/s
                                        actual_TCP_
↪Speed[5], 绕固定轴z旋转角速度, 单位deg/s */
    double   jt_cur_tor[6];             /* 6个轴当前扭矩, 单位N·m */
    int      tool;                      /* 应用的工具坐标系编号 */
    int      user;                      /* 应用的工件坐标系编号 */
    uint8_t  cl_dgt_output_h;           /* 控制箱数字量IO输出15-8 */
    uint8_t  cl_dgt_output_l;           /* 控制箱数字量IO输出7-0 */
    uint8_t  tl_dgt_output_l;           /* 工具数字量IO输出7-0, 仅bit0-bit1有效 */
```

(续下页)

(接上页)

```
    uint8_t  cl_dgt_input_h;            /* 控制箱数字量IO输入15-8 */
    uint8_t  cl_dgt_input_l;            /* 控制箱数字量IO输入7-0 */
    uint8_t  tl_dgt_input_l;            /* 工具数字量IO输入7-0，仅bit0-bit1有效 */
    uint16_t cl_analog_input[2];        /* cl_analog_input[0]，控制箱模拟量输入0
                                           cl_analog_input[1]，控制箱模拟量输入1 */
    uint16_t tl_anglog_input;           /* 工具模拟量输入 */
    double   ft_sensor_raw_data[6];     /* 力矩传感器原始数据
                                           ft_sensor_raw_data[0]，沿x轴力，单位N
                                           ft_sensor_raw_data[1]，沿y轴力，单位N
                                           ft_sensor_raw_data[2]，沿z轴力，单位N
                                           ft_sensor_raw_data[3]，沿x轴力矩，单位Nm
                                           ft_sensor_raw_data[4]，沿y轴力矩，单位Nm
                                           ft_sensor_raw_data[5]，沿z轴力矩，单位Nm */
    double   ft_sensor_data[6];         /* 力矩传感器数据，
                                           ft_sensor_data[0]，沿x轴力，单位N
                                           ft_sensor_data[1]，沿y轴力，单位N
                                           ft_sensor_data[2]，沿z轴力，单位N
                                           ft_sensor_data[3]，沿x轴力矩，单位Nm
                                           ft_sensor_data[4]，沿y轴力矩，单位Nm
                                           ft_sensor_data[5]，沿z轴力矩，单位Nm */
    uint8_t  ft_sensor_active;          /* 力矩传感器激活状态，0-复位，1-激活 */
    uint8_t  EmergencyStop;             /* 急停标志，0-急停未按下，1-急停按下 */
    int      motion_done;               /* 运动到位信号，1-到位，0-未到位 */
    uint8_t  gripper_motiondone;        /* 夹爪运动完成信号，1-完成，0-未完成 */
    int      mc_queue_len;              /* 运动指令队列长度 */
    uint8_t  collisionState;            /* 碰撞检测，1-碰撞，0-无碰撞 */
    int      trajectory_pnum;           /* 轨迹点编号 */
    uint8_t  safety_stop0_state;        /* 安全停止信号SI0 */
    uint8_t  safety_stop1_state;        /* 安全停止信号SI1 */
    uint8_t  gripper_fault_id;          /* 错误夹爪号 */
    uint16_t gripper_fault;             /* 夹爪故障 */
    uint16_t gripper_active;            /* 夹爪激活状态 */
    uint8_t  gripper_position;          /* 夹爪位置 */
    int8_t   gripper_speed;             /* 夹爪速度 */
    int8_t   gripper_current;           /* 夹爪电流 */
    int      gripper_temp;              /* 夹爪温度 */
    int      gripper_voltage;           /* 夹爪电压 */
}ROBOT_STATE_PKG;
```

2.1.2 机器人基础

2.1.2.1 实例化机器人

```
/**
* @brief  机器人接口类构造函数
*/
FRRobot();
```

2.1.2.2 与控制器建立通信

```
/**
* @brief  与机器人控制器建立通信
* @param  [in] ip  控制器IP地址，出场默认为192.168.58.2
* @return 错误码
*/
errno_t  RPC(const char *ip);
```

2.1.2.3 与控制器关闭通讯

```
/**
 * @brief  与机器人控制器关闭通讯
 * @return 错误码
 */
errno_t  CloseRPC();
```

2.1.2.4 查询 SDK 版本号

```
/**
* @brief  查询SDK版本号
* @param  [out] version   SDK版本号
* @return  错误码
*/
errno_t  GetSDKVersion(char *version);
```

2.1.2.5 获取控制器 IP

```
/**
* @brief  获取控制器IP
* @param  [out] ip  控制器IP
* @return  错误码
*/
errno_t  GetControllerIP(char *ip);
```

2.1.2.6 控制机器人进入或退出拖动示教模式

```
/**
* @brief  控制机器人进入或退出拖动示教模式
* @param  [in] state 0-退出拖动示教模式, 1-进入拖动示教模式
* @return  错误码
*/
errno_t  DragTeachSwitch(uint8_t state);
```

2.1.2.7 查询机器人是否处于拖动模式

```
/**
* @brief  查询机器人是否处于拖动示教模式
* @param  [out] state 0-非拖动示教模式, 1-拖动示教模式
* @return  错误码
*/
errno_t  IsInDragTeach(uint8_t *state);
```

2.1.2.8 控制机器人上使能或下使能

```
/**
* @brief  控制机器人上使能或下使能, 机器人上电后默认自动上使能
* @param  [in] state  0-下使能, 1-上使能
* @return  错误码
*/
errno_t  RobotEnable(uint8_t state);
```

2.1.2.9 控制机器人手自动模式切换

```
/**
* @brief 控制机器人手自动模式切换
* @param [in] mode 0-自动模式, 1-手动模式
* @return 错误码
*/
errno_t  Mode(int mode);
```

2.1.2.10 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    char ip[64]="";
    char version[64] = "";
    uint8_t state;

    robot.GetSDKVersion(version);
    printf("SDK version:%s\n", version);
    robot.GetControllerIP(ip);
    printf("controller ip:%s\n", ip);

    robot.Mode(1);
    sleep(1);
    robot.DragTeachSwitch(1);
    robot.IsInDragTeach(&state);
    printf("drag state :%u\n", state);
    sleep(3);
    robot.DragTeachSwitch(0);
    sleep(1);
```

(续下页)

(接上页)

```
    robot.IsInDragTeach(&state);
    printf("drag state :%u\n", state);
    sleep(3);

    robot.RobotEnable(0);
    sleep(3);
    robot.RobotEnable(1);

    robot.Mode(0);
    sleep(1);
    robot.Mode(1);

    return 0;
}
```

2.1.3 机器人运动

2.1.3.1 jog 点动

```
/**
* @brief  jog点动
* @param  [in]  ref 0-关节点动，2-基坐标系下点动，4-工具坐标系下点动，8-工件坐标系下点动
* @param  [in]  nb 1-关节1(或x轴)，2-关节2(或y轴)，3-关节3(或z轴)，4-关节4(或绕x轴旋转)，5-关节5(或绕y轴旋转)，6-关节6(或绕z轴旋转)
* @param  [in]  dir 0-负方向，1-正方向
* @param  [in]  vel 速度百分比，[0~100]
* @param  [in]  acc 加速度百分比，  [0~100]
* @param  [in]  max_dis 单次点动最大角度，单位[°]或距离，单位[mm]
* @return  错误码
*/
errno_t  StartJOG(uint8_t ref, uint8_t nb, uint8_t dir, float vel, float acc, float max_dis);
```

2.1.3.2 jog 点动减速停止

```
/**
* @brief  jog点动减速停止
* @param  [in]  ref  1-关节点动停止，3-基坐标系下点动停止，5-工具坐标系下点动停止，9-工件坐标系下点动停止
* @return  错误码
*/
errno_t  StopJOG(uint8_t ref);
```

2.1.3.3 jog 点动立即停止

```
/**
* @brief jog点动立即停止
* @return  错误码
*/
errno_t  ImmStopJOG();
```

2.1.3.4 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    robot.StartJOG(0,1,0,20.0,20.0,30.0);    //单关节运动，StartJOG为非阻塞指令，运动状态下接收其他运动指令（包含StartJOG）会被丢弃
    sleep(1);
    //robot.StopJOG(1)  //机器人单轴点动减速停止
    robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(0,2,1,20.0,20.0,30.0);
    sleep(1);
```

(续下页)

(接上页)

```
    robot.ImmStopJOG();
    robot.StartJOG(0,3,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(0,4,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(0,5,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(0,6,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();

    robot.StartJOG(2,1,0,20.0,20.0,30.0);   //基坐标系下点动
    sleep(1);
    //robot.StopJOG(3)  //机器人单轴点动减速停止
    robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(2,2,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(2,3,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(2,4,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(2,5,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(2,6,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();

    robot.StartJOG(4,1,0,20.0,20.0,30.0);   //工具坐标系下点动
    sleep(1);
    //robot.StopJOG(5)  //机器人单轴点动减速停止
    robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(4,2,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(4,3,1,20.0,20.0,30.0);
    sleep(1);
```

(续下页)

(接上页)

```
    robot.ImmStopJOG();
    robot.StartJOG(4,4,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(4,5,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(4,6,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();

    robot.StartJOG(8,1,0,20.0,20.0,30.0);    //工件坐标系下点动
    sleep(1);
    //robot.StopJOG(9)  //机器人单轴点动减速停止
    robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(8,2,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(8,3,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(8,4,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(8,5,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();
    robot.StartJOG(8,6,1,20.0,20.0,30.0);
    sleep(1);
    robot.ImmStopJOG();

    return 0;
}
```

2.1.3.5 关节空间运动

```
/**
* @brief  关节空间运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos   目标笛卡尔位姿
* @param  [in] tool  工具坐标号, 范围[1~15]
* @param  [in] user  工件坐标号, 范围[1~15]
```

(续下页)

(接上页)

```
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] epos  扩展轴位置，单位mm
* @param  [in] blendT [-1.0]-运动到位(阻塞)，[0~500.0]-平滑时间(非阻塞)，单位ms
* @param  [in] offset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
errno_t  MoveJ(JointPos *joint_pos, DescPose *desc_pos, int tool, int user, float vel,
↪ float acc, float ovl, ExaxisPos *epos, float blendT, uint8_t offset_flag, DescPose␣
↪*offset_pos);
```

2.1.3.6 笛卡尔空间直线运动

```
/**
* @brief  笛卡尔空间直线运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos  目标笛卡尔位姿
* @param  [in] tool  工具坐标号，范围[1~15]
* @param  [in] user  工件坐标号，范围[1~15]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @param  [in] epos  扩展轴位置，单位mm
* @param  [in] search  0-不焊丝寻位，1-焊丝寻位
* @param  [in] offset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
errno_t  MoveL(JointPos *joint_pos, DescPose *desc_pos, int tool, int user, float vel,
↪ float acc, float ovl, float blendR, ExaxisPos *epos, uint8_t search, uint8_t␣
↪offset_flag, DescPose *offset_pos);
```

2.1.3.7 笛卡尔空间圆弧运动

```
/**
* @brief  笛卡尔空间圆弧运动
* @param  [in] joint_pos_p  路径点关节位置,单位deg
* @param  [in] desc_pos_p   路径点笛卡尔位姿
* @param  [in] ptool  工具坐标号，范围[1~15]
* @param  [in] puser  工件坐标号，范围[1~15]
* @param  [in] pvel  速度百分比，范围[0~100]
* @param  [in] pacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_p  扩展轴位置，单位mm
* @param  [in] poffset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos_p  位姿偏移量
* @param  [in] joint_pos_t  目标点关节位置,单位deg
* @param  [in] desc_pos_t   目标点笛卡尔位姿
* @param  [in] ttool  工具坐标号，范围[1~15]
* @param  [in] tuser  工件坐标号，范围[1~15]
* @param  [in] tvel  速度百分比，范围[0~100]
* @param  [in] tacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_t  扩展轴位置，单位mm
* @param  [in] toffset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos_t  位姿偏移量
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @return  错误码
*/
errno_t  MoveC(JointPos *joint_pos_p, DescPose *desc_pos_p, int ptool, int puser,␣
↪float pvel, float pacc, ExaxisPos *epos_p, uint8_t poffset_flag, DescPose *offset_
↪pos_p,JointPos *joint_pos_t, DescPose *desc_pos_t, int ttool, int tuser, float tvel,
↪ float tacc, ExaxisPos *epos_t, uint8_t toffset_flag, DescPose *offset_pos_t,float␣
↪ovl, float blendR);
```

2.1.3.8 笛卡尔空间整圆运动

```
/**
* @brief  笛卡尔空间整圆运动
* @param  [in] joint_pos_p  路径点1关节位置,单位deg
* @param  [in] desc_pos_p   路径点1笛卡尔位姿
* @param  [in] ptool  工具坐标号，范围[1~15]
* @param  [in] puser  工件坐标号，范围[1~15]
* @param  [in] pvel  速度百分比，范围[0~100]
* @param  [in] pacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_p  扩展轴位置，单位mm
```

(续下页)

(接上页)

```
* @param  [in] joint_pos_t  路径点2关节位置,单位deg
* @param  [in] desc_pos_t   路径点2笛卡尔位姿
* @param  [in] ttool  工具坐标号, 范围[1~15]
* @param  [in] tuser  工件坐标号, 范围[1~15]
* @param  [in] tvel  速度百分比, 范围[0~100]
* @param  [in] tacc  加速度百分比, 范围[0~100],暂不开放
* @param  [in] epos_t  扩展轴位置, 单位mm
* @param  [in] ovl  速度缩放因子, 范围[0~100]
* @param  [in] offset_flag  0-不偏移, 1-基坐标系/工件坐标系下偏移, 2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
errno_t  Circle(JointPos *joint_pos_p, DescPose *desc_pos_p, int ptool, int puser,␣
↪float pvel, float pacc, ExaxisPos *epos_p, JointPos *joint_pos_t, DescPose *desc_
↪pos_t, int ttool, int tuser, float tvel, float tacc, ExaxisPos *epos_t, float ovl,␣
↪uint8_t offset_flag, DescPose *offset_pos);
```

2.1.3.9 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    JointPos j1,j2,j3,j4;
    DescPose desc_pos1,desc_pos2,desc_pos3,desc_pos4,offset_pos;
    ExaxisPos  epos;

    memset(&j1, 0, sizeof(JointPos));
    memset(&j2, 0, sizeof(JointPos));
    memset(&j3, 0, sizeof(JointPos));
    memset(&j4, 0, sizeof(JointPos));
```

(续下页)

(接上页)

```
    memset(&desc_pos1, 0, sizeof(DescPose));
    memset(&desc_pos2, 0, sizeof(DescPose));
    memset(&desc_pos3, 0, sizeof(DescPose));
    memset(&desc_pos4, 0, sizeof(DescPose));
    memset(&offset_pos, 0, sizeof(DescPose));
    memset(&epos, 0, sizeof(ExaxisPos));

    j1 = {114.578,-117.798,-97.745,-54.436,90.053,-45.216};
    desc_pos1.tran.x = -140.418;
    desc_pos1.tran.y = 619.351;
    desc_pos1.tran.z = 198.369;
    desc_pos1.rpy.rx = -179.948;
    desc_pos1.rpy.ry = 0.023;
    desc_pos1.rpy.rz = 69.793;

    j2 = {121.381,-97.108,-123.768,-45.824,89.877,-47.296};
    desc_pos2.tran.x = -127.772;
    desc_pos2.tran.y = 459.534;
    desc_pos2.tran.z = 221.274;
    desc_pos2.rpy.rx = -177.850;
    desc_pos2.rpy.ry = -2.507;
    desc_pos2.rpy.rz = 78.627;

    j3 = {138.884,-114.522,-103.933,-49.694,90.688,-47.291};
    desc_pos3.tran.x = -360.468;
    desc_pos3.tran.y = 485.600;
    desc_pos3.tran.z = 196.363;
    desc_pos3.rpy.rx = -178.239;
    desc_pos3.rpy.ry = -0.893;
    desc_pos3.rpy.rz = 96.172;

    j4 = {159.164,-96.105,-128.653,-41.170,90.704,-47.290};
    desc_pos4.tran.x = -360.303;
    desc_pos4.tran.y = 274.911;
    desc_pos4.tran.z = 203.968;
    desc_pos4.rpy.rx = -176.720;
    desc_pos4.rpy.ry = -2.514;
    desc_pos4.rpy.rz = 116.407;

    int tool = 0;
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
```

(续下页)

(接上页)

```
    float ovl = 100.0;
    float blendT = 0.0;
    float blendR = 0.0;
    uint8_t flag = 0;
    uint8_t search = 0;

    robot.SetSpeed(20);

    int err1 = robot.MoveJ(&j1, &desc_pos1, tool, user, vel, acc, ovl, &epos, blendT,
↪flag, &offset_pos);
    printf("movej errcode:%d\n", err1);

    int err2 = robot.MoveL(&j2, &desc_pos2, tool, user, vel, acc, ovl, blendR, &epos,
↪search,flag, &offset_pos);
    printf("movel errcode:%d\n", err2);

    int err3 = robot.MoveC(&j3,&desc_pos3,tool,user,vel,acc,&epos,flag,&offset_pos,&
↪j4,&desc_pos4,tool,user,vel,acc,&epos,flag,&offset_pos,ovl,blendR);
    printf("movec errcode:%d\n", err3);

    int err4 = robot.MoveJ(&j2, &desc_pos2, tool, user, vel, acc, ovl, &epos, blendT,
↪flag, &offset_pos);
    printf("movej errcode:%d\n", err4);

    int err5 = robot.Circle(&j3,&desc_pos3,tool,user,vel,acc,&epos,&j4,&desc_pos4,
↪tool,user,vel,acc,&epos,ovl,flag,&offset_pos);
    printf("circle errcode:%d\n", err5);

    return 0;
}
```

2.1.3.10 笛卡尔空间螺旋线运动

```
/**
* @brief  笛卡尔空间螺旋线运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos   目标笛卡尔位姿
* @param  [in] tool  工具坐标号，范围[1~15]
* @param  [in] user  工件坐标号，范围[1~15]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos  扩展轴位置，单位mm
```

(续下页)

(接上页)

```
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] offset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @param  [in] spiral_param  螺旋参数
* @return  错误码
*/
errno_t  NewSpiral(JointPos *joint_pos, DescPose *desc_pos, int tool, int user, float vel, float acc, ExaxisPos *epos, float ovl, uint8_t offset_flag, DescPose *offset_pos, SpiralParam spiral_param);
```

2.1.3.11 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    JointPos j;
    DescPose desc_pos, offset_pos1, offset_pos2;
    ExaxisPos  epos;
    SpiralParam sp;

    memset(&j, 0, sizeof(JointPos));
    memset(&desc_pos, 0, sizeof(DescPose));
    memset(&offset_pos1, 0, sizeof(DescPose));
    memset(&offset_pos2, 0, sizeof(DescPose));
    memset(&epos, 0, sizeof(ExaxisPos));
    memset(&sp, 0, sizeof(SpiralParam));

    j = {127.888,-101.535,-94.860,17.836,96.931,-61.325};
    offset_pos1.tran.x = 50.0;
    offset_pos1.rpy.rx = -30.0;
```

(续下页)

(接上页)

```
    offset_pos2.tran.x = 50.0;
    offset_pos2.rpy.rx = -5.0;

    sp.circle_num = 5;
    sp.circle_angle = 5.0;
    sp.rad_init = 50.0;
    sp.rad_add = 10.0;
    sp.rotaxis_add = 10.0;
    sp.rot_direction = 0;

    int tool = 0;
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
    float ovl = 100.0;
    float blendT = 0.0;
    uint8_t flag = 2;

    robot.SetSpeed(20);

    int ret = robot.GetForwardKin(&j, &desc_pos);  //只有关节位置的情况下，可用正运动学接口求解笛卡尔空间坐标

    if(ret == 0)
    {
        int err1 = robot.MoveJ(&j, &desc_pos, tool, user, vel, acc, ovl, &epos,blendT,flag, &offset_pos1);
        printf("movej errcode:%d\n", err1);

        int err2 = robot.NewSpiral(&j, &desc_pos, tool, user, vel, acc, &epos, ovl,flag, &offset_pos2, sp);
        printf("newspiral errcode:%d\n", err2);
    }
    else
    {
        printf("GetForwardKin errcode:%d\n", ret);
    }

    return 0;
}
```

2.1.3.12 伺服运动开始

```
/**
 * @brief 伺服运动开始，配合ServoJ、ServoCart指令使用
 * @return  错误码
 */
errno_t ServoMoveStart();
```

2.1.3.13 伺服运动结束

```
/**
 * @brief 伺服运动结束，配合ServoJ、ServoCart指令使用
 * @return  错误码
 */
errno_t ServoMoveEnd();
```

2.1.3.14 关节空间伺服模式运动

```
/**
* @brief  关节空间伺服模式运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放，默认为0
* @param  [in] vel  速度百分比，范围[0~100]，暂不开放，默认为0
* @param  [in] cmdT  指令下发周期，单位s，建议范围[0.001~0.0016]
* @param  [in] filterT 滤波时间，单位s，暂不开放，默认为0
* @param  [in] gain  目标位置的比例放大器，暂不开放，默认为0
* @return  错误码
*/
errno_t  ServoJ(JointPos *joint_pos, float acc, float vel, float cmdT, float filterT,␣
↪float gain);
```

2.1.3.15 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"
```

(续下页)

(接上页)

```

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    JointPos j;

    memset(&j, 0, sizeof(JointPos));

    float vel = 0.0;
    float acc = 0.0;
    float cmdT = 0.008;
    float filterT = 0.0;
    float gain = 0.0;
    uint8_t flag = 0;
    int count = 500;
    float dt = 0.1;

    int ret = robot.GetActualJointPosDegree(flag, &j);
    if(ret == 0)
    {
        while (count)
        {
            robot.ServoJ(&j, acc, vel, cmdT, filterT, gain);
            j.jPos[0] += dt;
            count -= 1;
            robot.WaitMs(cmdT*1000);
        }
    }
    else
    {
        printf("GetActualJointPosDegree errcode:%d\n", ret);
    }

    return 0;
}
```

2.1.3.16 笛卡尔空间伺服模式运动

```
/**
* @brief  笛卡尔空间伺服模式运动
* @param  [in]  mode  0-绝对运动(基坐标系), 1-增量运动(基坐标系), 2-增量运动(工具坐标系)
* @param  [in]  desc_pos  目标笛卡尔位姿或位姿增量
* @param  [in]  pos_gain  位姿增量比例系数, 仅在增量运动下生效, 范围[0~1]
* @param  [in] acc  加速度百分比, 范围[0~100],暂不开放, 默认为0
* @param  [in] vel  速度百分比, 范围[0~100], 暂不开放, 默认为0
* @param  [in] cmdT  指令下发周期, 单位s, 建议范围[0.001~0.0016]
* @param  [in] filterT 滤波时间, 单位s, 暂不开放, 默认为0
* @param  [in] gain  目标位置的比例放大器, 暂不开放, 默认为0
* @return  错误码
*/
errno_t  ServoCart(int mode, DescPose *desc_pose, float pos_gain[6], float acc, float vel, float cmdT, float filterT, float gain);
```

2.1.3.17 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    DescPose desc_pos_dt;
    memset(&desc_pos_dt, 0, sizeof(DescPose));

    desc_pos_dt.tran.z = -0.5;
    float pos_gain[6] = {0.0,0.0,1.0,0.0,0.0,0.0};
    int mode = 2;
    float vel = 0.0;
    float acc = 0.0;
```

(续下页)

(接上页)

```
    float cmdT = 0.008;
    float filterT = 0.0;
    float gain = 0.0;
    uint8_t flag = 0;
    int count = 100;

    robot.SetSpeed(20);

    while (count)
    {
        robot.ServoCart(mode, &desc_pos_dt, pos_gain, acc, vel, cmdT, filterT, gain);
        count -= 1;
        robot.WaitMs(cmdT*1000);
    }

    return 0;
}
```

2.1.3.18 笛卡尔空间点到点运动

```
/**
* @brief  笛卡尔空间点到点运动
* @param  [in]  desc_pos  目标笛卡尔位姿或位姿增量
* @param  [in] tool  工具坐标号，范围[1~15]
* @param  [in] user  工件坐标号，范围[1~15]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] blendT [-1.0]-运动到位(阻塞)，[0~500.0]-平滑时间(非阻塞)，单位ms
* @param  [in] config  关节空间配置，[-1]-参考当前关节位置解算，[0~7]-
↪参考特定关节空间配置解算，默认为-1
* @return  错误码
*/
errno_t  MoveCart(DescPose *desc_pos, int tool, int user, float vel, float acc, float␣
↪ovl, float blendT, int config);
```

2.1.3.19 代码示例

```cpp
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    DescPose desc_pos1, desc_pos2, desc_pos3;
    memset(&desc_pos1, 0, sizeof(DescPose));
    memset(&desc_pos2, 0, sizeof(DescPose));
    memset(&desc_pos3, 0, sizeof(DescPose));

    desc_pos1.tran.x = 75.414;
    desc_pos1.tran.y = 568.526;
    desc_pos1.tran.z = 338.135;
    desc_pos1.rpy.rx = -178.348;
    desc_pos1.rpy.ry = -0.930;
    desc_pos1.rpy.rz = 52.611;

    desc_pos2.tran.x = -273.856;
    desc_pos2.tran.y = 643.260;
    desc_pos2.tran.z = 259.235;
    desc_pos2.rpy.rx = -177.972;
    desc_pos2.rpy.ry = -1.494;
    desc_pos2.rpy.rz = 80.866;

    desc_pos3.tran.x = -423.044;
    desc_pos3.tran.y = 229.703;
    desc_pos3.tran.z = 241.080;
    desc_pos3.rpy.rx = -173.990;
    desc_pos3.rpy.ry = -5.772;
    desc_pos3.rpy.rz = 123.971;

    int tool = 0;
```

(续下页)

(接上页)

```
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
    float ovl = 100.0;
    float blendT = -1.0;
    float blendT1 = 0.0;
    int config = -1;

    robot.SetSpeed(20);
    robot.MoveCart(&desc_pos1, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveCart(&desc_pos2, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveCart(&desc_pos3, tool, user, vel, acc, ovl, blendT1, config);

    return 0;
}
```

2.1.3.20 样条运动开始

```
/**
* @brief  样条运动开始
* @return  错误码
*/
errno_t  SplineStart();
```

2.1.3.21 样条运动 PTP

```
/**
* @brief  关节空间样条运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos  目标笛卡尔位姿
* @param  [in] tool  工具坐标号，范围[1~15]
* @param  [in] user  工件坐标号，范围[1~15]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @return  错误码
*/
errno_t  SplinePTP(JointPos *joint_pos, DescPose *desc_pos, int tool, int user, float␣
↪vel, float acc, float ovl);
```

2.1.3.22 样条运动结束

```
/**
* @brief  样条运动结束
* @return  错误码
*/
errno_t  SplineEnd();
```

2.1.3.23 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    JointPos j1,j2,j3,j4;
    DescPose desc_pos1,desc_pos2,desc_pos3,desc_pos4,offset_pos;
    ExaxisPos  epos;

    memset(&j1, 0, sizeof(JointPos));
    memset(&j2, 0, sizeof(JointPos));
    memset(&j3, 0, sizeof(JointPos));
    memset(&j4, 0, sizeof(JointPos));
    memset(&desc_pos1, 0, sizeof(DescPose));
    memset(&desc_pos2, 0, sizeof(DescPose));
    memset(&desc_pos3, 0, sizeof(DescPose));
    memset(&desc_pos4, 0, sizeof(DescPose));
    memset(&offset_pos, 0, sizeof(DescPose));
    memset(&epos, 0, sizeof(ExaxisPos));

    j1 = {114.578,-117.798,-97.745,-54.436,90.053,-45.216};
    desc_pos1.tran.x = -140.418;
    desc_pos1.tran.y = 619.351;
```

(续下页)

(接上页)

```
    desc_pos1.tran.z = 198.369;
    desc_pos1.rpy.rx = -179.948;
    desc_pos1.rpy.ry = 0.023;
    desc_pos1.rpy.rz = 69.793;

    j2 = {115.401,-105.206,-117.959,-49.727,90.054,-45.222};
    desc_pos2.tran.x = -95.586;
    desc_pos2.tran.y = 504.143;
    desc_pos2.tran.z = 186.880;
    desc_pos2.rpy.rx = 178.001;
    desc_pos2.rpy.ry = 2.091;
    desc_pos2.rpy.rz = 70.585;

    j3 = {135.609,-103.249,-120.211,-49.715,90.058,-45.219};
    desc_pos3.tran.x = -252.429;
    desc_pos3.tran.y = 428.903;
    desc_pos3.tran.z = 188.492;
    desc_pos3.rpy.rx = 177.804;
    desc_pos3.rpy.ry = 2.294;
    desc_pos3.rpy.rz = 90.782;

    j4 = {154.766,-87.036,-135.672,-49.045,90.739,-45.223};
    desc_pos4.tran.x = -277.255;
    desc_pos4.tran.y = 272.958;
    desc_pos4.tran.z = 205.452;
    desc_pos4.rpy.rx = 179.289;
    desc_pos4.rpy.ry = 1.765;
    desc_pos4.rpy.rz = 109.966;

    int tool = 0;
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
    float ovl = 100.0;
    float blendT = -1.0;
    uint8_t flag = 0;

    robot.SetSpeed(20);

    int err1 = robot.MoveJ(&j1, &desc_pos1, tool, user, vel, acc, ovl, &epos, blendT,
↪flag, &offset_pos);
    printf("movej errcode:%d\n", err1);
    robot.SplineStart();
```

(续下页)

(接上页)

```
    robot.SplinePTP(&j1, &desc_pos1, tool, user, vel, acc, ovl);
    robot.SplinePTP(&j2, &desc_pos2, tool, user, vel, acc, ovl);
    robot.SplinePTP(&j3, &desc_pos3, tool, user, vel, acc, ovl);
    robot.SplinePTP(&j4, &desc_pos4, tool, user, vel, acc, ovl);
    robot.SplineEnd();

    return 0;
}
```

2.1.3.24 新样条运动开始

```
/**
 * @brief 新样条运动开始
 * @param [in] type  0-圆弧过渡，1-给定点位为路径点
 * @return 错误码
 */
errno_t  NewSplineStart(int type);
```

2.1.3.25 新样条指令点

```
/**
 * @brief 新样条指令点
 * @param [in] joint_pos 目标关节位置,单位deg
 * @param [in] desc_pos  目标笛卡尔位姿
 * @param [in] tool 工具坐标号，范围[1~15]
 * @param [in] user 工件坐标号，范围[1~15]
 * @param [in] vel 速度百分比，范围[0~100]
 * @param [in] acc 加速度百分比，范围[0~100],暂不开放
 * @param [in] ovl 速度缩放因子，范围[0~100]
 * @param [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
 * @return 错误码
 */
errno_t  NewSplinePoint(JointPos *joint_pos, DescPose *desc_pos, int tool, int user,␣
↪float vel, float acc, float ovl, float blendR);
```

2.1.3.26 新样条运动结束

```
/**
 * @brief 新样条运动结束
 * @return  错误码
 */
errno_t  NewSplineEnd();
```

2.1.3.27 终止运动

```
/**
* @brief 终止运动
* @return  错误码
*/
errno_t  StopMotion();
```

2.1.3.28 暂停运动

```
/**
 * @brief 暂停运动
 * @return  错误码
 */
errno_t  PauseMotion();
```

2.1.3.29 恢复运动

```
/**
 * @brief 恢复运动
 * @return  错误码
 */
errno_t  ResumeMotion();
```

2.1.3.30 点位整体偏移开始

```
/**
* @brief  点位整体偏移开始
* @param  [in]  flag  0-基坐标系下/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
```

(续下页)

(接上页)

```
*/
errno_t  PointsOffsetEnable(int flag, DescPose *offset_pos);
```

2.1.3.31 点位整体偏移结束

```
/**
* @brief  点位整体偏移结束
* @return  错误码
*/
errno_t  PointsOffsetDisable();
```

2.1.3.32 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    JointPos j1,j2;
    DescPose desc_pos1,desc_pos2,offset_pos,offset_pos1;
    ExaxisPos  epos;

    memset(&j1, 0, sizeof(JointPos));
    memset(&j2, 0, sizeof(JointPos));
    memset(&desc_pos1, 0, sizeof(DescPose));
    memset(&desc_pos2, 0, sizeof(DescPose));
    memset(&offset_pos, 0, sizeof(DescPose));
    memset(&offset_pos1, 0, sizeof(DescPose));
    memset(&epos, 0, sizeof(ExaxisPos));

    j1 = {114.578,-117.798,-97.745,-54.436,90.053,-45.216};
```

(续下页)

(接上页)

```
    desc_pos1.tran.x = -140.418;
    desc_pos1.tran.y = 619.351;
    desc_pos1.tran.z = 198.369;
    desc_pos1.rpy.rx = -179.948;
    desc_pos1.rpy.ry = 0.023;
    desc_pos1.rpy.rz = 69.793;

    j2 = {115.401,-105.206,-117.959,-49.727,90.054,-45.222};
    desc_pos2.tran.x = -95.586;
    desc_pos2.tran.y = 504.143;
    desc_pos2.tran.z = 186.880;
    desc_pos2.rpy.rx = 178.001;
    desc_pos2.rpy.ry = 2.091;
    desc_pos2.rpy.rz = 70.585;

    offset_pos1.tran.x = 100.0;
    offset_pos1.tran.y = 100.0;
    offset_pos1.tran.z = 100.0;
    offset_pos1.rpy.rx = 5.0;
    offset_pos1.rpy.ry = 5.0;
    offset_pos1.rpy.rz = 5.0;

    int tool = 0;
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
    float ovl = 100.0;
    float blendT = -1.0;
    float blendR = 0.0;
    uint8_t flag = 0;
    int type = 0;

    robot.SetSpeed(20);

    robot.MoveJ(&j1, &desc_pos1, tool, user, vel, acc, ovl, &epos, blendT,flag, &
↪offset_pos);
    robot.MoveJ(&j2, &desc_pos2, tool, user, vel, acc, ovl, &epos, blendT,flag, &
↪offset_pos);
    sleep(2);
    robot.PointsOffsetEnable(type, &offset_pos1);
    robot.MoveJ(&j1, &desc_pos1, tool, user, vel, acc, ovl, &epos, blendT,flag, &
↪offset_pos);
    robot.MoveJ(&j2, &desc_pos2, tool, user, vel, acc, ovl, &epos, blendT,flag, &
```

(续下页)

(接上页)

```
↪offset_pos);
    robot.PointsOffsetDisable();

    return 0;
}
```

2.1.4 机器人 IO

2.1.4.1 设置控制箱数字量输出

```
/**
* @brief  设置控制箱数字量输出
* @param  [in] id  io编号，范围[0~15]
* @param  [in] status 0-关，1-开
* @param  [in] smooth 0-不平滑， 1-平滑
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
errno_t  SetDO(int id, uint8_t status, uint8_t smooth, uint8_t block);
```

2.1.4.2 设置工具数字量输出

```
/**
* @brief  设置工具数字量输出
* @param  [in] id  io编号，范围[0~1]
* @param  [in] status 0-关，1-开
* @param  [in] smooth 0-不平滑， 1-平滑
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
errno_t  SetToolDO(int id, uint8_t status, uint8_t smooth, uint8_t block);
```

2.1.4.3 设置控制箱模拟量输出

```
/**
* @brief  设置控制箱模拟量输出
* @param  [in] id  io编号，范围[0~1]
* @param  [in] value 电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]
* @param  [in] block  0-阻塞，1-非阻塞
```

(续下页)

(接上页)

```
* @return  错误码
*/
errno_t  SetAO(int id, float value, uint8_t block);
```

2.1.4.4 设置工具模拟量输出

```
/**
* @brief  设置工具模拟量输出
* @param  [in] id  io编号，范围[0]
* @param  [in] value 电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
errno_t  SetToolAO(int id, float value, uint8_t block);
```

2.1.4.5 获取控制箱数字量输入

```
/**
* @brief  获取控制箱数字量输入
* @param  [in] id  io编号，范围[0~15]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  0-低电平，1-高电平
* @return  错误码
*/
errno_t  GetDI(int id, uint8_t block, uint8_t *result);
```

2.1.4.6 获取工具数字量输入

```
/**
* @brief  获取工具数字量输入
* @param  [in] id  io编号，范围[0~1]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  0-低电平，1-高电平
* @return  错误码
*/
errno_t  GetToolDI(int id, uint8_t block, uint8_t *result);
```

2.1.4.7 等待控制箱数字量输入

```
/**
* @brief 等待控制箱数字量输入
* @param [in] id io编号，范围[0~15]
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
errno_t WaitDI(int id, uint8_t status, int max_time, int opt);
```

2.1.4.8 等待控制箱多路数字量输入

```
/**
* @brief 等待控制箱多路数字量输入
* @param [in] mode 0-多路与，1-多路或
* @param [in] id io编号，bit0~bit7对应DI0~DI7，bit8~bit15对应CI0~CI7
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
errno_t WaitMultiDI(int mode, int id, uint8_t status, int max_time, int opt);
```

2.1.4.9 等待工具数字量输入

```
/**
* @brief 等待工具数字量输入
* @param [in] id io编号，范围[0~1]
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
errno_t WaitToolDI(int id, uint8_t status, int max_time, int opt);
```

2.1.4.10 获取控制箱模拟量输入

```
/**
* @brief  获取控制箱模拟量输入
* @param  [in] id  io编号，范围[0~1]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @return  错误码
*/
errno_t  GetAI(int id, uint8_t block, float *result);
```

2.1.4.11 获取工具模拟量输入

```
/**
* @brief  获取工具模拟量输入
* @param  [in] id  io编号，范围[0]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @return  错误码
*/
errno_t  GetToolAI(int id, uint8_t block, float *result);
```

2.1.4.12 获取机器人末端点记录按钮状态

```
/**
 * @brief 获取机器人末端点记录按钮状态
 * @param [out] state 按钮状态，0-按下，1-松开
 * @return 错误码
 */
errno_t  GetAxlePointRecordBtnState(uint8_t *state);
```

2.1.4.13 获取机器人末端 DO 输出状态

```
/**
 * @brief 获取机器人末端DO输出状态
 * @param [out] do_state DO输出状态，do0~do1对应bit1~bit2,从bit0开始
 * @return 错误码
 */
errno_t  GetToolDO(uint8_t *do_state);
```

2.1.4.14 获取机器人控制器 DO 输出状态

```
/**
 * @brief 获取机器人控制器DO输出状态
 * @param [out] do_state_h DO输出状态，co0~co7对应bit0~bit7
 * @param [out] do_state_l DO输出状态，do0~do7对应bit0~bit7
 * @return 错误码
 */
errno_t  GetDO(uint8_t *do_state_h, uint8_t *do_state_l);
```

2.1.4.15 等待控制箱模拟量输入

```
/**
* @brief 等待控制箱模拟量输入
* @param  [in] id  io编号，范围[0~1]
* @param  [in]  sign 0-大于，1-小于
* @param  [in]  value 输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @param  [in]  max_time  最大等待时间，单位ms
* @param  [in]  opt  超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return  错误码
*/
errno_t  WaitAI(int id, int sign, float value, int max_time, int opt);
```

2.1.4.16 等待工具模拟量输入

```
/**
* @brief 等待工具模拟量输入
* @param  [in] id  io编号，范围[0]
* @param  [in]  sign 0-大于，1-小于
* @param  [in]  value 输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @param  [in]  max_time  最大等待时间，单位ms
* @param  [in]  opt  超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return  错误码
*/
errno_t  WaitToolAI(int id, int sign, float value, int max_time, int opt);
```

2.1.4.17 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    uint8_t status = 1;
    uint8_t smooth = 0;
    uint8_t block  = 0;
    uint8_t di = 0, tool_di = 0;
    float ai = 0.0, tool_ai = 0.0;
    float value = 0.0;
    int i;

    for(i = 0; i < 16; i++)
    {
        robot.SetDO(i, status, smooth, block);
        robot.WaitMs(1000);
    }

    status = 0;

    for(i = 0; i < 16; i++)
    {
        robot.SetDO(i, status, smooth, block);
        robot.WaitMs(1000);
    }

    status = 1;

    for(i = 0; i < 2; i++)
    {
        robot.SetToolDO(i, status, smooth, block);
```

(续下页)

(接上页)

```
        robot.WaitMs(1000);
    }

    status = 0;

    for(i = 0; i < 2; i++)
    {
        robot.SetToolDO(i, status, smooth, block);
        robot.WaitMs(1000);
    }

    value = 50.0;
    robot.SetAO(0, value, block);
    value = 100.0;
    robot.SetAO(1, value, block);
    robot.WaitMs(1000);
    value = 0.0;
    robot.SetAO(0, value, block);
    value = 0.0;
    robot.SetAO(1, value, block);

    value = 100.0;
    robot.SetToolAO(0, value, block);
    robot.WaitMs(1000);
    value = 0.0;
    robot.SetToolAO(0, value, block);

    robot.GetDI(0, block, &di);
    printf("di0:%u\n", di);
    robot.WaitDI(0,1,0,2);                //一直等待
    robot.WaitMultiDI(1,3,3,10000,2);     //一直等待
    tool_di = robot.GetToolDI(1, block, &tool_di);
    printf("tool_di1:%u\n", tool_di);
    robot.WaitToolDI(1,1,0,2);            //一直等待

    robot.GetAI(0,block, &ai);
    printf("ai0:%f\n", ai);
    robot.WaitAI(0,0,50,0,2);             //一直等待
    robot.WaitToolAI(0,0,50,0,2);         //一直等待
    tool_ai = robot.GetToolAI(0,block, &tool_ai);
    printf("tool_ai0:%f\n", tool_ai);

    return 0;
}
```

2.1.5 机器人常用设置

2.1.5.1 设置全局速度

```
/**
* @brief  设置全局速度
* @param  [in]  vel  速度百分比, 范围[0~100]
* @return  错误码
*/
errno_t  SetSpeed(int vel);
```

2.1.5.2 设置系统变量值

```
/**
* @brief  设置系统变量值
* @param  [in]  id  变量编号, 范围[1~20]
* @param  [in]  value 变量值
* @return  错误码
*/
errno_t  SetSysVarValue(int id, float value);
```

2.1.5.3 设置工具参考点-六点法

```
/**
 * @brief 设置工具参考点-六点法
 * @param [in] point_num 点编号,范围[1~6]
 * @return 错误码
 */
errno_t SetToolPoint(int point_num);
```

2.1.5.4 计算工具坐标系

```
/**
 * @brief  计算工具坐标系
 * @param [out] tcp_pose 工具坐标系
 * @return 错误码
 */
errno_t ComputeTool(DescPose *tcp_pose);
```

2.1.5.5 设置工具参考点-四点法

```
/**
 * @brief 设置工具参考点-四点法
 * @param [in] point_num 点编号,范围[1~4]
 * @return 错误码
 */
errno_t SetTcp4RefPoint(int point_num);
```

2.1.5.6 计算工具坐标系

```
/**
 * @brief  计算工具坐标系
 * @param [out] tcp_pose 工具坐标系
 * @return 错误码
 */
errno_t ComputeTcp4(DescPose *tcp_pose);
```

2.1.5.7 设置工具坐标系

```
/**
* @brief  设置工具坐标系
* @param  [in] id 坐标系编号，范围[1~15]
* @param  [in] coord  工具中心点相对于末端法兰中心位姿
* @param  [in] type  0-工具坐标系，1-传感器坐标系
* @param  [in] install 安装位置，0-机器人末端，1-机器人外部
* @return  错误码
*/
errno_t  SetToolCoord(int id, DescPose *coord, int type, int install);
```

2.1.5.8 设置工具坐标系列表

```
/**
* @brief  设置工具坐标系列表
* @param  [in] id 坐标系编号，范围[1~15]
* @param  [in] coord  工具中心点相对于末端法兰中心位姿
* @param  [in] type  0-工具坐标系，1-传感器坐标系
* @param  [in] install 安装位置，0-机器人末端，1-机器人外部
* @return  错误码
*/
errno_t  SetToolList(int id, DescPose *coord, int type, int install);
```

2.1.5.9 设置外部工具参考点-六点法

```
/**
 * @brief 设置外部工具参考点-六点法
 * @param [in] point_num 点编号,范围[1~4]
 * @return 错误码
 */
errno_t SetExTCPPoint(int point_num);
```

2.1.5.10 计算外部工具坐标系

```
/**
 * @brief  计算外部工具坐标系
 * @param [out] tcp_pose 外部工具坐标系
 * @return 错误码
 */
errno_t ComputeExTCF(DescPose *tcp_pose);
```

2.1.5.11 设置外部工具坐标系

```
/**
* @brief  设置外部工具坐标系
* @param  [in] id 坐标系编号, 范围[1~15]
* @param  [in] etcp  工具中心点相对末端法兰中心位姿
* @param  [in] etool  待定
* @return  错误码
*/
errno_t  SetExToolCoord(int id, DescPose *etcp, DescPose *etool);
```

2.1.5.12 设置外部工具坐标系列表

```
/**
* @brief  设置外部工具坐标系列表
* @param  [in] id 坐标系编号, 范围[1~15]
* @param  [in] etcp  工具中心点相对末端法兰中心位姿
* @param  [in] etool  待定
* @return  错误码
*/
errno_t  SetExToolList(int id, DescPose *etcp, DescPose *etool);
```

2.1.5.13 设置工件参考点-三点法

```
/**
 * @brief 设置工件参考点-三点法
 * @param [in] point_num 点编号,范围[1~3]
 * @return 错误码
 */
errno_t SetWObjCoordPoint(int point_num);
```

2.1.5.14 计算工件坐标系

```
/**
 * @brief  计算工件坐标系
 * @param [out] wobj_pose 工件坐标系
 * @return 错误码
 */
errno_t ComputeWObjCoord(DescPose *wobj_pose);
```

2.1.5.15 设置工件坐标系

```
/**
* @brief  设置工件坐标系
* @param  [in] id 坐标系编号, 范围[1~15]
* @param  [in] coord  工件坐标系相对于末端法兰中心位姿
* @return  错误码
*/
errno_t  SetWObjCoord(int id, DescPose *coord);
```

2.1.5.16 设置工件坐标系列表

```
/**
* @brief  设置工件坐标系列表
* @param  [in] id 坐标系编号, 范围[1~15]
* @param  [in] coord  工件坐标系相对于末端法兰中心位姿
* @return  错误码
*/
errno_t  SetWObjList(int id, DescPose *coord);
```

2.1.5.17 设置末端负载重量

```
/**
* @brief  设置末端负载重量
* @param  [in] weight  负载重量，单位kg
* @return  错误码
*/
errno_t  SetLoadWeight(float weight);
```

2.1.5.18 设置末端负载质心坐标

```
/**
* @brief  设置末端负载质心坐标
* @param  [in] coord 质心坐标，单位mm
* @return  错误码
*/
errno_t  SetLoadCoord(DescTran *coord);
```

2.1.5.19 设置机器人安装方式

```
/**
* @brief  设置机器人安装方式
* @param  [in] install  安装方式，0-正装，1-侧装，2-倒装
* @return  错误码
*/
errno_t  SetRobotInstallPos(uint8_t install);
```

2.1.5.20 设置机器人安装角度

```
/**
* @brief  设置机器人安装角度，自由安装
* @param  [in] yangle  倾斜角
* @param  [in] zangle  旋转角
* @return  错误码
*/
errno_t  SetRobotInstallAngle(double yangle, double zangle);
```

2.1.5.21 等待指定时间

```
/**
* @brief  等待指定时间
* @param  [in]  t_ms  单位ms
* @return  错误码
*/
errno_t  WaitMs(int t_ms);
```

2.1.5.22 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int i;
    float value;
    int id;
    int type;
    int install;

    DescTran coord;
    DescPose t_coord, etcp, etool, w_coord;
    memset(&coord, 0, sizeof(DescTran));
    memset(&t_coord, 0, sizeof(DescPose));
    memset(&etcp, 0, sizeof(DescPose));
    memset(&etool, 0, sizeof(DescPose));
    memset(&w_coord, 0, sizeof(DescPose));

    robot.SetSpeed(20);

    for(i = 1; i < 21; i++)
```

(续下页)

(接上页)

```
    {
        robot.SetSysVarValue(i, i+0.5);
        robot.WaitMs(1000);
    }

    for(i = 1; i < 21; i++)
    {
        robot.GetSysVarValue(i, &value);
        printf("sys value:%f\n", value);
    }

    robot.SetLoadWeight(2.5);

    coord.x = 3.0;
    coord.y = 4.0;
    coord.z = 5.0;

    robot.SetLoadCoord(&coord);

    id = 10;
    t_coord.tran.x = 1.0;
    t_coord.tran.y = 2.0;
    t_coord.tran.z = 3.0;
    t_coord.rpy.rx = 4.0;
    t_coord.rpy.ry = 5.0;
    t_coord.rpy.rz = 6.0;
    type = 0;
    install = 0;
    robot.SetToolCoord(id, &t_coord, type, install);
    robot.SetToolList(id, &t_coord, type, install);

    etcp.tran.x = 1.0;
    etcp.tran.y = 2.0;
    etcp.tran.z = 3.0;
    etcp.rpy.rx = 4.0;
    etcp.rpy.ry = 5.0;
    etcp.rpy.rz = 6.0;
    etool.tran.x = 11.0;
    etool.tran.y = 22.0;
    etool.tran.z = 33.0;
    etool.rpy.rx = 44.0;
    etool.rpy.ry = 55.0;
    etool.rpy.rz = 66.0;
```

(续下页)

(接上页)

```
    id = 11;
    robot.SetExToolCoord(id, &etcp, &etool);
    robot.SetExToolList(id, &etcp, &etool);

    w_coord.tran.x = 11.0;
    w_coord.tran.y = 12.0;
    w_coord.tran.z = 13.0;
    w_coord.rpy.rx = 14.0;
    w_coord.rpy.ry = 15.0;
    w_coord.rpy.rz = 16.0;
    id = 12;
    robot.SetWObjCoord(id, &w_coord);
    robot.SetWObjList(id, &w_coord);

    robot.SetRobotInstallPos(0);
    robot.SetRobotInstallAngle(15.0,25.0);

    return 0;
}
```

2.1.6 机器人安全设置

2.1.6.1 设置碰撞等级

```
/**
* @brief 设置碰撞等级
* @param [in] mode 0-等级，1-百分比
* @param [in] level 碰撞阈值，等级对应范围[],百分比对应范围[0~1]
* @param [in] config 0-不更新配置文件，1-更新配置文件
* @return 错误码
*/
errno_t SetAnticollision(int mode, float level[6], int config);
```

2.1.6.2 设置碰撞后策略

```
/**
* @brief 设置碰撞后策略
* @param [in] strategy 0-报错停止，1-继续运行
* @return 错误码
*/
errno_t SetCollisionStrategy(int strategy);
```

2.1.6.3 设置正限位

```
/**
* @brief  设置正限位
* @param  [in] limit 六个关节位置, 单位deg
* @return  错误码
*/
errno_t  SetLimitPositive(float limit[6]);
```

2.1.6.4 设置负限位

```
/**
* @brief  设置负限位
* @param  [in] limit 六个关节位置, 单位deg
* @return  错误码
*/
errno_t  SetLimitNegative(float limit[6]);
```

2.1.6.5 错误状态清除

```
/**
* @brief  错误状态清除
* @return  错误码
*/
errno_t  ResetAllError();
```

2.1.6.6 关节摩擦力补偿开关

```
/**
* @brief  关节摩擦力补偿开关
* @param  [in]  state  0-关, 1-开
* @return  错误码
*/
errno_t  FrictionCompensationOnOff(uint8_t state);
```

2.1.6.7 设置关节摩擦力补偿系数-正装

```
/**
* @brief  设置关节摩擦力补偿系数-正装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
errno_t  SetFrictionValue_level(float coeff[6]);
```

2.1.6.8 设置关节摩擦力补偿系数-侧装

```
/**
* @brief  设置关节摩擦力补偿系数-侧装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
errno_t  SetFrictionValue_wall(float coeff[6]);
```

2.1.6.9 设置关节摩擦力补偿系数-倒装

```
/**
* @brief  设置关节摩擦力补偿系数-倒装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
errno_t  SetFrictionValue_ceiling(float coeff[6]);
```

2.1.6.10 设置关节摩擦力补偿系数-自由安装

```
/**
* @brief  设置关节摩擦力补偿系数-自由安装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
errno_t  SetFrictionValue_freedom(float coeff[6]);
```

2.1.6.11 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int mode = 0;
    int config = 1;
    float level1[6] = {1.0,2.0,3.0,4.0,5.0,6.0};
    float level2[6] = {50.0,20.0,30.0,40.0,50.0,60.0};

    robot.SetAnticollision(mode, level1, config);
    mode = 1;
    robot.SetAnticollision(mode, level2, config);
    robot.SetCollisionStrategy(1);

    float plimit[6] = {170.0,80.0,150.0,80.0,170.0,160.0};
    robot.SetLimitPositive(plimit);
    float nlimit[6] = {-170.0,-260.0,-150.0,-260.0,-170.0,-160.0};
    robot.SetLimitNegative(nlimit);

    robot.ResetAllError();

    float lcoeff[6] = {0.9,0.9,0.9,0.9,0.9,0.9};
    float wcoeff[6] = {0.4,0.4,0.4,0.4,0.4,0.4};
    float ccoeff[6] = {0.6,0.6,0.6,0.6,0.6,0.6};
    float fcoeff[6] = {0.5,0.5,0.5,0.5,0.5,0.5};
    robot.FrictionCompensationOnOff(1);
    robot.SetFrictionValue_level(lcoeff);
    robot.SetFrictionValue_wall(wcoeff);
    robot.SetFrictionValue_ceiling(ccoeff);
    robot.SetFrictionValue_freedom(fcoeff);

```

(续下页)

(接上页)

```
    return 0;
}
```

2.1.7 机器人状态查询

2.1.7.1 获取机器人安装角度

```
/**
* @brief  获取机器人安装角度
* @param  [out] yangle 倾斜角
* @param  [out] zangle 旋转角
* @return  错误码
*/
errno_t  GetRobotInstallAngle(float *yangle, float *zangle);
```

2.1.7.2 获取系统变量值

```
/**
* @brief  获取系统变量值
* @param  [in] id 系统变量编号，范围[1~20]
* @param  [out] value  系统变量值
* @return  错误码
*/
errno_t  GetSysVarValue(int id, float *value);
```

2.1.7.3 获取当前关节位置 (角度)

```
/**
* @brief  获取当前关节位置(角度)
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] jPos 六个关节位置，单位deg
* @return  错误码
*/
errno_t  GetActualJointPosDegree(uint8_t flag, JointPos *jPos);
```

2.1.7.4 获取当前关节位置 (弧度)

```
/**
* @brief  获取当前关节位置(弧度)
* @param  [in] flag 0-阻塞, 1-非阻塞
* @param  [out] jPos 六个关节位置, 单位rad
* @return  错误码
*/
errno_t  GetActualJointPosRadian(uint8_t flag, JointPos *jPos);
```

2.1.7.5 获取关节反馈速度

```
/**
 * @brief  获取关节反馈速度-deg/s
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] speed 六个关节速度
 * @return  错误码
 */
errno_t  GetActualJointSpeedsDegree(uint8_t flag, float speed[6]);
```

2.1.7.6 获取关节反馈加速度

```
/**
 * @brief  获取关节反馈加速度-deg/s^2
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] acc 六个关节加速度
 * @return  错误码
 */
errno_t  GetActualJointAccDegree(uint8_t flag, float acc[6]);
```

2.1.7.7 获取 TCP 指令速度

```
/**
 * @brief  获取TCP指令速度
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] tcp_speed 线性速度
 * @param  [out] ori_speed 姿态速度
 * @return  错误码
 */
errno_t  GetTargetTCPCompositeSpeed(uint8_t flag, float *tcp_speed, float *ori_speed);
```

2.1.7.8 获取 TCP 反馈速度

```
/**
 * @brief  获取TCP反馈速度
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] tcp_speed 线性速度
 * @param  [out] ori_speed 姿态速度
 * @return  错误码
 */
errno_t  GetActualTCPCompositeSpeed(uint8_t flag, float *tcp_speed, float *ori_speed);
```

2.1.7.9 获取 TCP 指令速度

```
/**
 * @brief  获取TCP指令速度
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] speed [x,y,z,rx,ry,rz]速度
 * @return  错误码
 */
errno_t  GetTargetTCPSpeed(uint8_t flag, float speed[6]);
```

2.1.7.10 获取 TCP 反馈速度

```
/**
 * @brief  获取TCP反馈速度
 * @param  [in] flag 0-阻塞, 1-非阻塞
 * @param  [out] speed [x,y,z,rx,ry,rz]速度
 * @return  错误码
 */
errno_t  GetActualTCPSpeed(uint8_t flag, float speed[6]);
```

2.1.7.11 获取当前工具位姿

```
/**
* @brief  获取当前工具位姿
* @param  [in] flag  0-阻塞, 1-非阻塞
* @param  [out] desc_pos  工具位姿
* @return  错误码
*/
errno_t  GetActualTCPPose(uint8_t flag, DescPose *desc_pos);
```

2.1.7.12 获取当前工具坐标系编号

```
/**
* @brief  获取当前工具坐标系编号
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] id  工具坐标系编号
* @return  错误码
*/
errno_t  GetActualTCPNum(uint8_t flag, int *id);
```

2.1.7.13 获取当前工件坐标系编号

```
/**
* @brief  获取当前工件坐标系编号
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] id  工件坐标系编号
* @return  错误码
*/
errno_t  GetActualWObjNum(uint8_t flag, int *id);
```

2.1.7.14 获取当前末端法兰位姿

```
/**
* @brief  获取当前末端法兰位姿
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] desc_pos  法兰位姿
* @return  错误码
*/
errno_t  GetActualToolFlangePose(uint8_t flag, DescPose *desc_pos);
```

2.1.7.15 逆运动学求解

```
/**
* @brief  逆运动学求解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] config 关节空间配置，[-1]-参考当前关节位置解算，[0~7]-
↪依据特定关节空间配置求解
* @param  [out] joint_pos 关节位置
* @return  错误码
```

(续下页)

(接上页)

```
*/
errno_t  GetInverseKin(int type, DescPose *desc_pos, int config, JointPos *joint_pos);
```

2.1.7.16 逆运动学求解

```
/**
* @brief  逆运动学求解，参考指定关节位置求解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] joint_pos_ref 参考关节位置
* @param  [out] joint_pos 关节位置
* @return  错误码
*/
errno_t  GetInverseKinRef(int type, DescPose *desc_pos, JointPos *joint_pos_ref,␣
↪JointPos *joint_pos);
```

2.1.7.17 逆运动学求解

```
/**
* @brief  逆运动学求解，参考指定关节位置判断是否有解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] joint_pos_ref 参考关节位置
* @param  [out] result 0-无解，1-有解
* @return  错误码
*/
errno_t  GetInverseKinHasSolution(int type, DescPose *desc_pos, JointPos *joint_pos_
↪ref, uint8_t *result);
```

2.1.7.18 正运动学求解

```
/**
* @brief  正运动学求解
* @param  [in] joint_pos 关节位置
* @param  [out] desc_pos 笛卡尔位姿
* @return  错误码
*/
errno_t  GetForwardKin(JointPos *joint_pos, DescPose *desc_pos);
```

2.1.7.19 获取当前关节转矩

```
/**
* @brief 获取当前关节转矩
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] torques 关节转矩
* @return 错误码
*/
errno_t GetJointTorques(uint8_t flag, float torques[6]);
```

2.1.7.20 获取当前负载的重量

```
/**
* @brief 获取当前负载的重量
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] weight 负载重量, 单位kg
* @return 错误码
*/
errno_t GetTargetPayload(uint8_t flag, float *weight);
```

2.1.7.21 获取当前负载的质心

```
/**
* @brief 获取当前负载的质心
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] cog 负载质心, 单位mm
* @return 错误码
*/
errno_t GetTargetPayloadCog(uint8_t flag, DescTran *cog);
```

2.1.7.22 获取当前工具坐标系

```
/**
* @brief 获取当前工具坐标系
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] desc_pos 工具坐标系位姿
* @return 错误码
*/
errno_t GetTCPOffset(uint8_t flag, DescPose *desc_pos);
```

2.1.7.23 获取当前工件坐标系

```
/**
* @brief  获取当前工件坐标系
* @param  [in] flag 0-阻塞, 1-非阻塞
* @param  [out] desc_pos 工件坐标系位姿
* @return  错误码
*/
errno_t  GetWObjOffset(uint8_t flag, DescPose *desc_pos);
```

2.1.7.24 获取关节软限位角度

```
/**
* @brief  获取关节软限位角度
* @param  [in] flag 0-阻塞, 1-非阻塞
* @param  [out] negative  负限位角度, 单位deg
* @param  [out] positive  正限位角度, 单位deg
* @return  错误码
*/
errno_t  GetJointSoftLimitDeg(uint8_t flag, float negative[6], float positive[6]);
```

2.1.7.25 获取系统时间

```
/**
* @brief  获取系统时间
* @param  [out] t_ms 单位ms
* @return  错误码
*/
errno_t  GetSystemClock(float *t_ms);
```

2.1.7.26 获取机器人当前关节配置

```
/**
* @brief  获取机器人当前关节位置
* @param  [out]  config  关节空间配置, 范围[0~7]
* @return  错误码
*/
errno_t  GetRobotCurJointsConfig(int *config);
```

2.1.7.27 获取当前速度

```
/**
* @brief  获取机器人当前速度
* @param  [out]  vel  速度，单位mm/s
* @return  错误码
*/
errno_t  GetDefaultTransVel(float *vel);
```

2.1.7.28 查询机器人运动是否完成

```
/**
* @brief  查询机器人运动是否完成
* @param  [out]  state  0-未完成，1-完成
* @return  错误码
*/
errno_t  GetRobotMotionDone(uint8_t *state);
```

2.1.7.29 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    float yangle, zangle;
    int flag = 0;
    JointPos j_deg, j_rad;
    DescPose tcp, flange, tcp_offset, wobj_offset;
    DescTran cog;
    int id;
    float torques[6] = {0.0};
```

(续下页)

(接上页)

```
    float weight;
    float neg_deg[6]={0.0},pos_deg[6]={0.0};
    float t_ms;
    int config;
    float vel;

    memset(&j_deg, 0, sizeof(JointPos));
    memset(&j_rad, 0, sizeof(JointPos));
    memset(&tcp, 0, sizeof(DescPose));
    memset(&flange, 0, sizeof(DescPose));
    memset(&tcp_offset, 0, sizeof(DescPose));
    memset(&wobj_offset, 0, sizeof(DescPose));
    memset(&cog, 0, sizeof(DescTran));

    robot.GetRobotInstallAngle(&yangle, &zangle);
    printf("yangle:%f,zangle:%f\n", yangle, zangle);

    robot.GetActualJointPosDegree(flag, &j_deg);
    printf("joint pos deg:%f,%f,%f,%f,%f,%f\n", j_deg.jPos[0],j_deg.jPos[1],j_deg.
↪jPos[2],j_deg.jPos[3],j_deg.jPos[4],j_deg.jPos[5]);

    robot.GetActualJointPosRadian(flag, &j_rad);
    printf("joint pos rad:%f,%f,%f,%f,%f,%f\n", j_rad.jPos[0],j_rad.jPos[1],j_rad.
↪jPos[2],j_rad.jPos[3],j_rad.jPos[4],j_rad.jPos[5]);

    robot.GetActualTCPPose(flag, &tcp);
    printf("tcp pose:%f,%f,%f,%f,%f,%f\n", tcp.tran.x, tcp.tran.y, tcp.tran.z, tcp.
↪rpy.rx, tcp.rpy.ry, tcp.rpy.rz);

    robot.GetActualToolFlangePose(flag, &flange);
    printf("flange pose:%f,%f,%f,%f,%f,%f\n", flange.tran.x, flange.tran.y, flange.
↪tran.z, flange.rpy.rx, flange.rpy.ry, flange.rpy.rz);

    robot.GetActualTCPNum(flag, &id);
    printf("tcp num:%d\n", id);

    robot.GetActualWObjNum(flag, &id);
    printf("wobj num:%d\n", id);

    robot.GetJointTorques(flag, torques);
    printf("torques:%f,%f,%f,%f,%f,%f\n", torques[0],torques[1],torques[2],torques[3],
↪torques[4],torques[5]);

```

(续下页)

(接上页)

```
    robot.GetTargetPayload(flag, &weight);
    printf("payload weight:%f\n", weight);

    robot.GetTargetPayloadCog(flag, &cog);
    printf("payload cog:%f,%f,%f\n",cog.x, cog.y, cog.z);

    robot.GetTCPOffset(flag, &tcp_offset);
    printf("tcp offset:%f,%f,%f,%f,%f,%f\n", tcp_offset.tran.x,tcp_offset.tran.y,tcp_
↪offset.tran.z,tcp_offset.rpy.rx,tcp_offset.rpy.ry,tcp_offset.rpy.rz);

    robot.GetWObjOffset(flag, &wobj_offset);
    printf("wobj offset:%f,%f,%f,%f,%f,%f\n", wobj_offset.tran.x,wobj_offset.tran.y,
↪wobj_offset.tran.z,wobj_offset.rpy.rx,wobj_offset.rpy.ry,wobj_offset.rpy.rz);

    robot.GetJointSoftLimitDeg(flag, neg_deg, pos_deg);
    printf("neg limit deg:%f,%f,%f,%f,%f,%f\n",neg_deg[0],neg_deg[1],neg_deg[2],neg_
↪deg[3],neg_deg[4],neg_deg[5]);
    printf("pos limit deg:%f,%f,%f,%f,%f,%f\n",pos_deg[0],pos_deg[1],pos_deg[2],pos_
↪deg[3],pos_deg[4],pos_deg[5]);

    robot.GetSystemClock(&t_ms);
    printf("system clock:%f\n", t_ms);

    robot.GetRobotCurJointsConfig(&config);
    printf("joint config:%d\n", config);

    robot.GetDefaultTransVel(&vel);
    printf("trans vel:%f\n", vel);

    return 0;
}
```

2.1.7.30 查询机器人错误码

```
/**
 * @brief  查询机器人错误码
 * @param  [out]  maincode  主错误码
 * @param  [out]  subcode   子错误码
 * @return  错误码
 */
errno_t  GetRobotErrorCode(int *maincode, int *subcode);
```

2.1.7.31 查询机器人示教管理点位数据

```
/**
 * @brief 查询机器人示教管理点位数据
 * @param [in] name 点位名
 * @param [out] data 点位数据
 * @return 错误码
 */
errno_t GetRobotTeachingPoint(char name[64], float data[20]);
```

2.1.7.32 查询机器人运动队列缓存长度

```
/**
 * @brief 查询机器人运动队列缓存长度
 * @param [out] len 缓存长度
 * @return 错误码
 */
errno_t GetMotionQueueLength(int *len);
```

2.1.8 机器人轨迹复现

2.1.8.1 设置轨迹记录参数

```
/**
* @brief 设置轨迹记录参数
* @param [in] type 记录数据类型，1-关节位置
* @param [in] name 轨迹文件名
* @param [in] period_ms 数据采样周期，固定值2ms或4ms或8ms
* @param [in] di_choose DI选择,bit0~bit7对应控制箱DI0~DI7，bit8~bit9对应末端DI0~
↪DI1，0-不选择，1-选择
* @param [in] do_choose DO选择,bit0~bit7对应控制箱DO0~DO7，bit8~bit9对应末端DO0~
↪DO1，0-不选择，1-选择
* @return 错误码
*/
errno_t SetTPDParam(int type, char name[30], int period_ms, uint16_t di_choose,␣
↪uint16_t do_choose);
```

2.1.8.2 开始轨迹记录

```
/**
* @brief  开始轨迹记录
* @param  [in] type  记录数据类型，1-关节位置
* @param  [in] name  轨迹文件名
* @param  [in] period_ms  数据采样周期，固定值2ms或4ms或8ms
* @param  [in] di_choose  DI选择,bit0~bit7对应控制箱DI0~DI7，bit8~bit9对应末端DI0~DI1，0-不选择，1-选择
* @param  [in] do_choose  DO选择,bit0~bit7对应控制箱DO0~DO7，bit8~bit9对应末端DO0~DO1，0-不选择，1-选择
* @return  错误码
*/
errno_t  SetTPDStart(int type, char name[30], int period_ms, uint16_t di_choose, uint16_t do_choose);
```

2.1.8.3 停止轨迹记录

```
/**
* @brief  停止轨迹记录
* @return  错误码
*/
errno_t  SetWebTPDStop();
```

2.1.8.4 删除轨迹记录

```
/**
* @brief  删除轨迹记录
* @param  [in] name  轨迹文件名
* @return  错误码
*/
errno_t  SetTPDDelete(char name[30]);
```

2.1.8.5 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
```

(续下页)

(接上页)

```
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int type = 1;
    char name[30] = "tpd2023";
    int period_ms = 4;
    uint16_t di_choose = 0;
    uint16_t do_choose = 0;

    robot.SetTPDParam(type, name, period_ms, di_choose, do_choose);

    robot.Mode(1);
    sleep(1);
    robot.DragTeachSwitch(1);
    robot.SetTPDStart(type, name, period_ms, di_choose, do_choose);
    sleep(30);
    robot.SetWebTPDStop();
    robot.DragTeachSwitch(0);

    //robot.SetTPDDelete(name);

    return 0;
}
```

2.1.8.6 轨迹预加载

```
/**
* @brief  轨迹预加载
* @param  [in] name  轨迹文件名
* @return  错误码
*/
errno_t  LoadTPD(char name[30]);
```

2.1.8.7 获取轨迹起始位姿

```
/**
 * @brief  获取轨迹起始位姿
 * @param  [in] name 轨迹文件名,不需要文件后缀
 * @return  错误码
 */
errno_t  GetTPDStartPose(char name[30], DescPose *desc_pose);
```

2.1.8.8 轨迹复现

```
/**
* @brief  轨迹复现
* @param  [in] name  轨迹文件名
* @param  [in] blend 0-不平滑, 1-平滑
* @param  [in] ovl  速度缩放百分比, 范围[0~100]
* @return  错误码
*/
errno_t  MoveTPD(char name[30], uint8_t blend, float ovl);
```

2.1.8.9 轨迹预处理

```
/**
 * @brief  轨迹预处理
 * @param  [in] name  轨迹文件名
 * @param  [in] ovl 速度缩放百分比, 范围[0~100]
 * @param  [in] opt 1-控制点, 默认为1
 * @return  错误码
 */
errno_t  LoadTrajectoryJ(char name[30], float ovl, int opt);
```

2.1.8.10 轨迹复现

```
/**
 * @brief  轨迹复现
 * @return  错误码
 */
errno_t  MoveTrajectoryJ();
```

2.1.8.11 获取轨迹起始位姿

```
/**
 * @brief  获取轨迹起始位姿
 * @param  [in] name 轨迹文件名
 * @return  错误码
 */
errno_t  GetTrajectoryStartPose(char name[30], DescPose *desc_pose);
```

2.1.8.12 获取轨迹点编号

```
/**
 * @brief  获取轨迹点编号
 * @return  错误码
 */
errno_t  GetTrajectoryPointNum(int *pnum);
```

2.1.8.13 设置轨迹运行中的速度

```
/**
 * @brief  设置轨迹运行中的速度
 * @param  [in] ovl 速度百分比
 * @return  错误码
 */
errno_t  SetTrajectoryJSpeed(float ovl);
```

2.1.8.14 设置轨迹运行中的力和扭矩

```
/**
 * @brief  设置轨迹运行中的力和扭矩
 * @param  [in] ft 三个方向的力和扭矩，单位N和Nm
 * @return  错误码
 */
errno_t  SetTrajectoryJForceTorque(ForceTorque *ft);
```

2.1.8.15 设置轨迹运行中的沿 x 方向的力

```
/**
 * @brief  设置轨迹运行中的沿x方向的力
 * @param  [in] fx 沿x方向的力，单位N
 * @return  错误码
 */
errno_t  SetTrajectoryJForceFx(double fx);
```

2.1.8.16 设置轨迹运行中的沿 y 方向的力

```
/**
 * @brief  设置轨迹运行中的沿y方向的力
 * @param  [in] fy 沿y方向的力，单位N
 * @return  错误码
 */
errno_t  SetTrajectoryJForceFy(double fy);
```

2.1.8.17 设置轨迹运行中的沿 z 方向的力

```
/**
 * @brief  设置轨迹运行中的沿z方向的力
 * @param  [in] fz 沿x方向的力，单位N
 * @return  错误码
 */
errno_t  SetTrajectoryJForceFz(double fz);
```

2.1.8.18 设置轨迹运行中的绕 x 轴的扭矩

```
/**
 * @brief  设置轨迹运行中的绕x轴的扭矩
 * @param  [in] tx 绕x轴的扭矩，单位Nm
 * @return  错误码
 */
errno_t  SetTrajectoryJTorqueTx(double tx);
```

2.1.8.19 设置轨迹运行中的绕 y 轴的扭矩

```
/**
 * @brief  设置轨迹运行中的绕y轴的扭矩
 * @param  [in] ty 绕y轴的扭矩，单位Nm
 * @return  错误码
 */
errno_t  SetTrajectoryJTorqueTy(double ty);
```

2.1.8.20 设置轨迹运行中的绕 z 轴的扭矩

```
/**
 * @brief  设置轨迹运行中的绕z轴的扭矩
 * @param  [in] tz 绕z轴的扭矩，单位Nm
 * @return  错误码
 */
errno_t  SetTrajectoryJTorqueTz(double tz);
```

2.1.8.21 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    char name[30] = "tpd2023";
    int tool = 1;
    int user = 0;
    float vel = 100.0;
    float acc = 100.0;
    float ovl = 100.0;
    float blendT = -1.0;
```

(续下页)

(接上页)

```
    int config = -1;
    uint8_t blend = 1;

    DescPose desc_pose;
    memset(&desc_pose, 0, sizeof(DescPose));

    desc_pose.tran.x = -378.9;
    desc_pose.tran.y = -340.3;
    desc_pose.tran.z = 107.2;
    desc_pose.rpy.rx = 179.4;
    desc_pose.rpy.ry = -1.3;
    desc_pose.rpy.rz = 125.0;

    robot.LoadTPD(name);
    robot.MoveCart(&desc_pose, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveTPD(name, blend, ovl);

    return 0;
}
```

2.1.9 机器人 WebAPP 程序使用

2.1.9.1 设置开机自动加载默认的作业程序

```
/**
* @brief  设置开机自动加载默认的作业程序
* @param  [in] flag  0-开机不自动加载默认程序，1-开机自动加载默认程序
* @param  [in] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
↪"为固定路径
* @return  错误码
*/
errno_t  LoadDefaultProgConfig(uint8_t flag, char program_name[64]);
```

2.1.9.2 加载指定的作业程序

```
/**
* @brief  加载指定的作业程序
* @param  [in] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
↪"为固定路径
* @return  错误码
```

(续下页)

(接上页)

```
*/
errno_t  ProgramLoad(char program_name[64]);
```

2.1.9.3 获取已加载的作业程序名

```
/**
* @brief  获取已加载的作业程序名
* @param  [out] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
↪"为固定路径
* @return  错误码
*/
errno_t  GetLoadedProgram(char program_name[64]);
```

2.1.9.4 获取当前机器人作业程序的执行行号

```
/**
* @brief  获取当前机器人作业程序执行的行号
* @param  [out] line  行号
* @return  错误码
*/
errno_t  GetCurrentLine(int *line);
```

2.1.9.5 运行当前加载的作业程序

```
/**
* @brief  运行当前加载的作业程序
* @return  错误码
*/
errno_t  ProgramRun();
```

2.1.9.6 暂停当前运行的作业程序

```
/**
* @brief  暂停当前运行的作业程序
* @return  错误码
*/
errno_t  ProgramPause();
```

2.1.9.7 恢复当前暂停的作业程序

```
/**
* @brief  恢复当前暂停的作业程序
* @return  错误码
*/
errno_t  ProgramResume();
```

2.1.9.8 终止当前运行的作业程序

```
/**
* @brief  终止当前运行的作业程序
* @return  错误码
*/
errno_t  ProgramStop();
```

2.1.9.9 获取机器人作业程序执行状态

```
/**
* @brief  获取机器人作业程序执行状态
* @param  [out]  state 1-程序停止或无程序运行，2-程序运行中，3-程序暂停
* @return  错误码
*/
errno_t  GetProgramState(uint8_t *state);
```

2.1.9.10 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
```

(续下页)

(接上页)

```
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    char program_name[64] = "/fruser/ptps.lua";
    char loaded_name[64] = "";
    uint8_t state;
    int line;

    robot.Mode(0);
    robot.ProgramLoad(program_name);
    robot.ProgramRun();
    sleep(5);
    robot.ProgramPause();
    robot.GetProgramState(&state);
    printf("program state:%u\n", state);
    robot.GetCurrentLine(&line);
    printf("current line:%d\n", line);
    robot.GetLoadedProgram(loaded_name);
    printf("program name:%s\n", loaded_name);
    sleep(5);
    robot.ProgramResume();
    sleep(5);
    robot.ProgramStop();
    sleep(2);

    return 0;
}
```

2.1.10 机器人外设

2.1.10.1 配置夹爪

```
/**
* @brief  配置夹爪
* @param  [in] company  夹爪厂商，待定
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
errno_t  SetGripperConfig(int company, int device, int softvesion, int bus);
```

2.1.10.2 获取夹爪配置

```
/**
* @brief  获取夹爪配置
* @param  [in] company  夹爪厂商，待定
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
errno_t  GetGripperConfig(int *company, int *device, int *softvesion, int *bus);
```

2.1.10.3 激活夹爪

```
/**
* @brief  激活夹爪
* @param  [in] index  夹爪编号
* @param  [in] act  0-复位，1-激活
* @return  错误码
*/
errno_t  ActGripper(int index, uint8_t act);
```

2.1.10.4 控制夹爪

```
/**
* @brief  控制夹爪
* @param  [in] index  夹爪编号
* @param  [in] pos  位置百分比，范围[0~100]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] force  力矩百分比，范围[0~100]
* @param  [in] max_time  最大等待时间，范围[0~30000]，单位ms
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
errno_t  MoveGripper(int index, int pos, int vel, int force, int max_time, uint8_t␣
↪block);
```

2.1.10.5 获取夹爪运动状态

```
/**
 * @brief  获取夹爪运动状态
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] staus  0-运动未完成, 1-运动完成
 * @return  错误码
 */
errno_t  GetGripperMotionDone(uint16_t *fault, uint8_t *status);
```

2.1.10.6 获取夹爪激活状态

```
/**
 * @brief  获取夹爪激活状态
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] status  bit0~bit15对应夹爪编号0~15, bit=0为未激活, bit=1为激活
 * @return  错误码
 */
errno_t  GetGripperActivateStatus(uint16_t *fault, uint16_t *status);
```

2.1.10.7 获取夹爪位置

```
/**
 * @brief  获取夹爪位置
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] position  位置百分比, 范围0~100%
 * @return  错误码
 */
errno_t  GetGripperCurPosition(uint16_t *fault, uint8_t *position);
```

2.1.10.8 获取夹爪速度

```
/**
 * @brief  获取夹爪速度
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] speed  速度百分比, 范围0~100%
 * @return  错误码
 */
errno_t  GetGripperCurSpeed(uint16_t *fault, int8_t *speed);
```

2.1.10.9 获取夹爪电流

```
/**
 * @brief  获取夹爪电流
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] current  电流百分比, 范围0~100%
 * @return  错误码
 */
errno_t  GetGripperCurCurrent(uint16_t *fault, int8_t *current);
```

2.1.10.10 获取夹爪电压

```
/**
 * @brief  获取夹爪电压
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] voltage  电压,单位0.1V
 * @return  错误码
 */
errno_t  GetGripperVoltage(uint16_t *fault, int *voltage);
```

2.1.10.11 获取夹爪温度

```
/**
 * @brief  获取夹爪温度
 * @param  [out] fault  0-无错误, 1-有错误
 * @param  [out] temp  温度, 单位℃
 * @return  错误码
 */
errno_t  GetGripperTemp(uint16_t *fault, int *temp);
```

2.1.10.12 计算预抓取点-视觉

```
/**
 * @brief  计算预抓取点-视觉
 * @param  [in] desc_pos  抓取点笛卡尔位姿
 * @param  [in] zlength  z轴偏移量
 * @param  [in] zangle  绕z轴旋转偏移量
 * @return  错误码
 */
errno_t  ComputePrePick(DescPose *desc_pos, double zlength, double zangle, DescPose␣
↪*pre_pos);
```

2.1.10.13 计算撤退点-视觉

```
/**
 * @brief  计算撤退点-视觉
 * @param  [in] desc_pos  抓取点笛卡尔位姿
 * @param  [in] zlength   z轴偏移量
 * @param  [in] zangle    绕z轴旋转偏移量
 * @return  错误码
 */
errno_t  ComputePostPick(DescPose *desc_pos, double zlength, double zangle, DescPose
↪*post_pos);
```

2.1.10.14 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int company = 4;
    int device = 0;
    int softversion = 0;
    int bus = 1;
    int index = 1;
    int act = 0;
    int max_time = 30000;
    uint8_t block = 0;
    uint8_t status, fault;

    robot.SetGripperConfig(company, device, softversion, bus);
    sleep(1);
    robot.GetGripperConfig(&company, &device, &softversion, &bus);
    printf("gripper config:%d,%d,%d,%d\n", company, device, softversion, bus);
```

(续下页)

(接上页)

```
30
31      robot.ActGripper(index, act);
32      sleep(1);
33      act = 1;
34      robot.ActGripper(index, act);
35      sleep(2);
36
37      robot.MoveGripper(index, 100, 50, 50, max_time, block);
38      sleep(3);
39      robot.MoveGripper(index, 0, 50, 0, max_time, block);
40
41      robot.GetGripperMotionDone(&fault, &status);
42      printf("motion status:%u,%u\n", fault, status);
43
44      return 0;
45  }
```

2.1.11 机器人力控

2.1.11.1 力传感器配置

```
/**
* @brief  配置力传感器
* @param  [in] company  力传感器厂商，17-坤维科技
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
errno_t  FT_SetConfig(int company, int device, int softvesion, int bus);
```

2.1.11.2 获取力传感器配置

```
/**
* @brief  获取力传感器配置
* @param  [in] company  力传感器厂商，待定
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
```

(续下页)

(接上页)

```
*/
errno_t  FT_GetConfig(int *company, int *device, int *softvesion, int *bus);
```

2.1.11.3 力传感器激活

```
/**
* @brief  力传感器激活
* @param  [in] act  0-复位，1-激活
* @return  错误码
*/
errno_t  FT_Activate(uint8_t act);
```

2.1.11.4 力传感器校零

```
/**
* @brief  力传感器校零
* @param  [in] act  0-去除零点，1-零点矫正
* @return  错误码
*/
errno_t  FT_SetZero(uint8_t act);
```

2.1.11.5 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>

#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int company = 17;
```

(续下页)

(接上页)

```
    int device = 0;
    int softversion = 0;
    int bus = 1;
    int index = 1;
    int act = 0;

    robot.FT_SetConfig(company, device, softversion, bus);
    sleep(1);
    robot.FT_GetConfig(&company, &device, &softversion, &bus);
    printf("FT config:%d,%d,%d,%d\n", company, device, softversion, bus);
    sleep(1);

    robot.FT_Activate(act);
    sleep(1);
    act = 1;
    robot.FT_Activate(act);
    sleep(1);

    robot.SetLoadWeight(0.0);
    sleep(1);
    DescTran coord;
    memset(&coord, 0, sizeof(DescTran));
    robot.SetLoadCoord(&coord);
    sleep(1);
    robot.FT_SetZero(0);
    sleep(1);

    ForceTorque ft;
    memset(&ft, 0, sizeof(ForceTorque));
    robot.FT_GetForceTorqueOrigin(&ft);
    printf("ft origin:%f,%f,%f,%f,%f,%f\n", ft.fx,ft.fy,ft.fz,ft.tx,ft.ty,ft.tz);
    robot.FT_SetZero(1);
    sleep(1);
    memset(&ft, 0, sizeof(ForceTorque));
    printf("ft rcs:%f,%f,%f,%f,%f,%f\n",ft.fx,ft.fy,ft.fz,ft.tx,ft.ty,ft.tz);

    return 0;
}
```

2.1.11.6 设置力传感器参考坐标系

```
/**
* @brief  设置力传感器参考坐标系
* @param  [in] ref  0-工具坐标系，1-基坐标系
* @return  错误码
*/
errno_t  FT_SetRCS(uint8_t ref);
```

2.1.11.7 负载重量辨识记录

```
/**
* @brief  负载重量辨识记录
* @param  [in] id  传感器坐标系编号，范围[1~14]
* @return  错误码
*/
errno_t  FT_PdIdenRecord(int id);
```

2.1.11.8 负载重量辨识计算

```
/**
* @brief  负载重量辨识计算
* @param  [out] weight  负载重量，单位kg
* @return  错误码
*/
errno_t  FT_PdIdenCompute(float *weight);
```

2.1.11.9 负载质心辨识记录

```
/**
* @brief  负载质心辨识记录
* @param  [in] id  传感器坐标系编号，范围[1~14]
* @param  [in] index 点编号，范围[1~3]
* @return  错误码
*/
errno_t  FT_PdCogIdenRecord(int id, int index);
```

2.1.11.10 负载质心辨识计算

```
/**
* @brief  负载质心辨识计算
* @param  [out] cog  负载质心，单位mm
* @return  错误码
*/
errno_t  FT_PdCogIdenCompute(DescTran *cog);
```

2.1.11.11 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    float weight;

    DescPose tcoord, desc_p1, desc_p2, desc_p3;
    memset(&tcoord, 0, sizeof(DescPose));
    memset(&desc_p1, 0, sizeof(DescPose));
    memset(&desc_p2, 0, sizeof(DescPose));
    memset(&desc_p3, 0, sizeof(DescPose));

    robot.FT_SetRCS(0);
    sleep(1);

    tcoord.tran.z = 35.0;
    robot.SetToolCoord(10, &tcoord, 1, 0);
    sleep(1);
    robot.FT_PdIdenRecord(10);
    sleep(1);
    robot.FT_PdIdenCompute(&weight);
```

(续下页)

(接上页)

```
    printf("payload weight:%f\n", weight);

    desc_p1.tran.x = -160.619;
    desc_p1.tran.y = -586.138;
    desc_p1.tran.z = 384.988;
    desc_p1.rpy.rx = -170.166;
    desc_p1.rpy.ry = -44.782;
    desc_p1.rpy.rz = 169.295;

    desc_p2.tran.x = -87.615;
    desc_p2.tran.y = -606.209;
    desc_p2.tran.z = 556.119;
    desc_p2.rpy.rx = -102.495;
    desc_p2.rpy.ry = 10.118;
    desc_p2.rpy.rz = 178.985;

    desc_p3.tran.x = 41.479;
    desc_p3.tran.y = -557.243;
    desc_p3.tran.z = 484.407;
    desc_p3.rpy.rx = -125.174;
    desc_p3.rpy.ry = 46.995;
    desc_p3.rpy.rz = -132.165;

    robot.MoveCart(&desc_p1, 9, 0, 100.0, 100.0, 100.0, -1.0, -1);
    sleep(1);
    robot.FT_PdCogIdenRecord(10, 1);
    robot.MoveCart(&desc_p2, 9, 0, 100.0, 100.0, 100.0, -1.0, -1);
    sleep(1);
    robot.FT_PdCogIdenRecord(10, 2);
    robot.MoveCart(&desc_p3, 9, 0, 100.0, 100.0, 100.0, -1.0, -1);
    sleep(1);
    robot.FT_PdCogIdenRecord(10, 3);
    sleep(1);
    DescTran cog;
    memset(&cog, 0, sizeof(DescTran));
    robot.FT_PdCogIdenCompute(&cog);
    printf("cog:%f,%f,%f\n",cog.x, cog.y, cog.z);

    return 0;
}
```

2.1.11.12 获取参考坐标系下力/扭矩数据

```
/**
* @brief  获取参考坐标系下力/扭矩数据
* @param  [out] ft  力/扭矩，fx,fy,fz,tx,ty,tz
* @return  错误码
*/
errno_t  FT_GetForceTorqueRCS(ForceTorque *ft);
```

2.1.11.13 获取力传感器原始力/扭矩数据

```
/**
* @brief  获取力传感器原始力/扭矩数据
* @param  [out] ft  力/扭矩，fx,fy,fz,tx,ty,tz
* @return  错误码
*/
errno_t  FT_GetForceTorqueOrigin(ForceTorque *ft);
```

2.1.11.14 碰撞守护

```
/**
* @brief  碰撞守护
* @param  [in] flag 0-关闭碰撞守护，1-开启碰撞守护
* @param  [in] sensor_id 力传感器编号
* @param  [in] select  选择六个自由度是否检测碰撞，0-不检测，1-检测
* @param  [in] ft  碰撞力/扭矩，fx,fy,fz,tx,ty,tz
* @param  [in] max_threshold 最大阈值
* @param  [in] min_threshold 最小阈值
* @note  力/扭矩检测范围：(ft-min_threshold, ft+max_threshold)
* @return  错误码
*/
errno_t  FT_Guard(uint8_t flag, int sensor_id, uint8_t select[6], ForceTorque *ft,␣
↪float max_threshold[6], float min_threshold[6]);
```

2.1.11.15 代码示例

```cpp
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    uint8_t flag = 1;
    uint8_t sensor_id = 1;
    uint8_t select[6] = {1,1,1,1,1,1};
    float max_threshold[6] = {10.0,10.0,10.0,10.0,10.0,10.0};
    float min_threshold[6] = {5.0,5.0,5.0,5.0,5.0,5.0};

    ForceTorque ft;
    DescPose desc_p1, desc_p2, desc_p3;
    memset(&ft, 0, sizeof(ForceTorque));
    memset(&desc_p1, 0, sizeof(DescPose));
    memset(&desc_p2, 0, sizeof(DescPose));
    memset(&desc_p3, 0, sizeof(DescPose));

    desc_p1.tran.x = -160.619;
    desc_p1.tran.y = -586.138;
    desc_p1.tran.z = 384.988;
    desc_p1.rpy.rx = -170.166;
    desc_p1.rpy.ry = -44.782;
    desc_p1.rpy.rz = 169.295;

    desc_p2.tran.x = -87.615;
    desc_p2.tran.y = -606.209;
    desc_p2.tran.z = 556.119;
    desc_p2.rpy.rx = -102.495;
    desc_p2.rpy.ry = 10.118;
    desc_p2.rpy.rz = 178.985;

```

(续下页)

(接上页)

```
    desc_p3.tran.x = 41.479;
    desc_p3.tran.y = -557.243;
    desc_p3.tran.z = 484.407;
    desc_p3.rpy.rx = -125.174;
    desc_p3.rpy.ry = 46.995;
    desc_p3.rpy.rz = -132.165;

    robot.FT_Guard(flag, sensor_id, select, &ft, max_threshold, min_threshold);
    robot.MoveCart(&desc_p1,9,0,100.0,100.0,100.0,-1.0,-1);
    robot.MoveCart(&desc_p2,9,0,100.0,100.0,100.0,-1.0,-1);
    robot.MoveCart(&desc_p3,9,0,100.0,100.0,100.0,-1.0,-1);
    flag = 0;
    robot.FT_Guard(flag, sensor_id, select, &ft, max_threshold, min_threshold);

    return 0;
}
```

2.1.11.16 恒力控制

```
/**
* @brief  恒力控制
* @param  [in] flag 0-关闭恒力控制，1-开启恒力控制
* @param  [in] sensor_id 力传感器编号
* @param  [in] select  选择六个自由度是否检测碰撞，0-不检测，1-检测
* @param  [in] ft  碰撞力/扭矩，fx,fy,fz,tx,ty,tz
* @param  [in] ft_pid 力pid参数，力矩pid参数
* @param  [in] adj_sign 自适应启停控制，0-关闭，1-开启
* @param  [in] ILC_sign ILC启停控制，  0-停止，1-训练，2-实操
* @param  [in] 最大调整距离，单位mm
* @param  [in] 最大调整角度，单位deg
* @return  错误码
*/
errno_t  FT_Control(uint8_t flag, int sensor_id, uint8_t select[6], ForceTorque *ft,␣
↪float ft_pid[6], uint8_t adj_sign, uint8_t ILC_sign, float max_dis, float max_ang);
```

2.1.11.17 代码示例

```cpp
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    uint8_t flag = 1;
    uint8_t sensor_id = 1;
    uint8_t select[6] = {0,0,1,0,0,0};
    float ft_pid[6] = {0.0005,0.0,0.0,0.0,0.0,0.0};
    uint8_t adj_sign = 0;
    uint8_t ILC_sign = 0;
    float max_dis = 100.0;
    float max_ang = 0.0;

    ForceTorque ft;
    DescPose desc_p1, desc_p2, offset_pos;
    JointPos j1,j2;
    ExaxisPos epos;
    memset(&ft, 0, sizeof(ForceTorque));
    memset(&desc_p1, 0, sizeof(DescPose));
    memset(&desc_p2, 0, sizeof(DescPose));
    memset(&offset_pos, 0, sizeof(DescPose));
    memset(&epos, 0, sizeof(ExaxisPos));
    memset(&j1, 0, sizeof(JointPos));
    memset(&j2, 0, sizeof(JointPos));

    j1 = {-68.987,-96.414,-111.45,-61.105,92.884,11.089};
    j2 = {-107.596,-109.154,-104.735,-56.176,90.739,11.091};

    desc_p1.tran.x = 62.795;
    desc_p1.tran.y = -511.979;
    desc_p1.tran.z = 291.697;
```

(续下页)

(接上页)

```
    desc_p1.rpy.rx = -179.545;
    desc_p1.rpy.ry = 3.027;
    desc_p1.rpy.rz = -170.039;

    desc_p2.tran.x = -294.768;
    desc_p2.tran.y = -503.708;
    desc_p2.tran.z = 233.158;
    desc_p2.rpy.rx = 179.799;
    desc_p2.rpy.ry = 0.713;
    desc_p2.rpy.rz = 151.309;

    ft.fz = -10.0;

    robot.MoveJ(&j1,&desc_p1,9,0,100.0,180.0,100.0,&epos,-1.0,0,&offset_pos);
    robot.FT_Control(flag, sensor_id, select, &ft, ft_pid, adj_sign, ILC_sign, max_
↪dis, max_ang);
    robot.MoveL(&j2,&desc_p2,9,0,100.0,180.0,20.0,-1.0,&epos,0,0,&offset_pos);
    flag = 0;
    robot.FT_Control(flag, sensor_id, select, &ft, ft_pid, adj_sign, ILC_sign, max_
↪dis, max_ang);

    return 0;
}
```

2.1.11.18 螺旋线探索

```
/**
* @brief  螺旋线探索
* @param  [in] rcs 参考坐标系，0-工具坐标系，1-基坐标系
* @param  [in] dr 每圈半径进给量
* @param  [in] ft 力/扭矩阈值，fx,fy,fz,tx,ty,tz，范围[0~100]
* @param  [in] max_t_ms 最大探索时间，单位ms
* @param  [in] max_vel 最大线速度，单位mm/s
* @return  错误码
*/
errno_t  FT_SpiralSearch(int rcs, float dr, float ft, float max_t_ms, float max_vel);
```

2.1.11.19 旋转插入

```
/**
* @brief  旋转插入
* @param  [in] rcs 参考坐标系，0-工具坐标系，1-基坐标系
* @param  [in] angVelRot 旋转角速度，单位deg/s
* @param  [in] ft  力/扭矩阈值，fx,fy,fz,tx,ty,tz，范围[0~100]
* @param  [in] max_angle 最大旋转角度，单位deg
* @param  [in] orn 力/扭矩方向，1-沿z轴方向，2-绕z轴方向
* @param  [in] max_angAcc 最大旋转加速度，单位deg/s^2，暂不使用，默认为0
* @param  [in] rotorn  旋转方向，1-顺时针，2-逆时针
* @return  错误码
*/
errno_t  FT_RotInsertion(int rcs, float angVelRot, float ft, float max_angle, uint8_t orn, float max_angAcc, uint8_t rotorn);
```

2.1.11.20 直线插入

```
/**
* @brief  直线插入
* @param  [in] rcs 参考坐标系，0-工具坐标系，1-基坐标系
* @param  [in] ft  力/扭矩阈值，fx,fy,fz,tx,ty,tz，范围[0~100]
* @param  [in] lin_v 直线速度，单位mm/s
* @param  [in] lin_a 直线加速度，单位mm/s^2，暂不使用
* @param  [in] max_dis 最大插入距离，单位mm
* @param  [in] linorn  插入方向，0-负方向，1-正方向
* @return  错误码
*/
errno_t  FT_LinInsertion(int rcs, float ft, float lin_v, float lin_a, float max_dis, uint8_t linorn);
```

2.1.11.21 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;
```

(续下页)

(接上页)

```

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    //恒力参数
    uint8_t status = 1;  //恒力控制开启标志，0-关，1-开
    int sensor_num = 1; //力传感器编号
    float gain[6] = {0.0001,0.0,0.0,0.0,0.0,0.0};  //最大阈值
    uint8_t adj_sign = 0;  //自适应启停状态，0-关闭，1-开启
    uint8_t ILC_sign = 0;  //ILC控制启停状态，0-停止，1-训练，2-实操
    float max_dis = 100.0;  //最大调整距离
    float max_ang = 5.0;  //最大调整角度

    ForceTorque ft;
    memset(&ft, 0, sizeof(ForceTorque));

    //螺旋线探索参数
    int rcs = 0;  //参考坐标系，0-工具坐标系，1-基坐标系
    float dr = 0.7;  //每圈半径进给量，单位mm
    float fFinish = 1.0; //力或力矩阈值（0~100），单位N或Nm
    float t = 60000.0; //最大探索时间，单位ms
    float vmax = 3.0; //线速度最大值，单位mm/s

    //直线插入参数
    float force_goal = 20.0;  //力或力矩阈值（0~100），单位N或Nm
    float lin_v = 0.0; //直线速度，单位mm/s
    float lin_a = 0.0; //直线加速度，单位mm/s^2,暂不使用
    float disMax = 100.0; //最大插入距离，单位mm
    uint8_t linorn = 1; //插入方向，1-正方向，2-负方向

    //旋转插入参数
    float angVelRot = 2.0;  //旋转角速度，单位°/s
    float forceInsertion = 1.0; //力或力矩阈值（0~100），单位N或Nm
    int angleMax= 45; //最大旋转角度，单位°
    uint8_t orn = 1; //力的方向，1-fz,2-mz
    float angAccmax = 0.0; //最大旋转角加速度，单位°/s^2,暂不使用
    uint8_t rotorn = 1; //旋转方向，1-顺时针，2-逆时针

    uint8_t select1[6] = {0,0,1,1,1,0}; //六个自由度选择[fx,fy,fz,mx,my,mz]，0-
↪不生效，1-生效
    ft.fz = -10.0;

```

(续下页)

(接上页)

```
    robot.FT_Control(status,sensor_num,select1,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);
    robot.FT_SpiralSearch(rcs,dr,fFinish,t,vmax);
    status = 0;
    robot.FT_Control(status,sensor_num,select1,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);

    uint8_t select2[6] = {1,1,1,0,0,0};  //六个自由度选择[fx,fy,fz,mx,my,mz], 0-不生效, 1-生效
    gain[0] = 0.00005;
    ft.fz = -30.0;
    status = 1;
    robot.FT_Control(status,sensor_num,select2,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);
    robot.FT_LinInsertion(rcs,force_goal,lin_v,lin_a,disMax,linorn);
    status = 0;
    robot.FT_Control(status,sensor_num,select2,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);

    uint8_t select3[6] = {0,0,1,1,1,0};  //六个自由度选择[fx,fy,fz,mx,my,mz], 0-不生效, 1-生效
    ft.fz = -10.0;
    gain[0] = 0.0001;
    status = 1;
    robot.FT_Control(status,sensor_num,select3,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);
    robot.FT_RotInsertion(rcs,angVelRot,forceInsertion,angleMax,orn,angAccmax,rotorn);
    status = 0;
    robot.FT_Control(status,sensor_num,select3,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);

    uint8_t select4[6] = {1,1,1,0,0,0};  //六个自由度选择[fx,fy,fz,mx,my,mz], 0-不生效, 1-生效
    ft.fz = -30.0;
    status = 1;
    robot.FT_Control(status,sensor_num,select4,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);
    robot.FT_LinInsertion(rcs,force_goal,lin_v,lin_a,disMax,linorn);
    status = 0;
    robot.FT_Control(status,sensor_num,select4,&ft,gain,adj_sign,ILC_sign,max_dis,max_ang);

    return 0;
}
```

2.1.11.22 表面定位

```
/**
* @brief  表面定位
* @param  [in] rcs 参考坐标系，0-工具坐标系，1-基坐标系
* @param  [in] dir  移动方向，1-正方向，2-负方向
* @param  [in] axis 移动轴，1-x轴，2-y轴，3-z轴
* @param  [in] lin_v 探索直线速度，单位mm/s
* @param  [in] lin_a 探索直线加速度，单位mm/s^2，暂不使用，默认为0
* @param  [in] max_dis 最大探索距离，单位mm
* @param  [in] ft  动作终止力/扭矩阈值，fx,fy,fz,tx,ty,tz
* @return  错误码
*/
errno_t  FT_FindSurface(int rcs, uint8_t dir, uint8_t axis, float lin_v, float lin_a,␣
↪float max_dis, float ft);
```

2.1.11.23 计算中间平面位置开始

```
/**
* @brief  计算中间平面位置开始
* @return  错误码
*/
errno_t  FT_CalCenterStart();
```

2.1.11.24 计算中间平面位置结束

```
/**
* @brief  计算中间平面位置结束
* @param  [out] pos 中间平面位姿
* @return  错误码
*/
errno_t  FT_CalCenterEnd(DescPose *pos);
```

2.1.11.25 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
```

(续下页)

(接上页)

```cpp
#include "FRRobot.h"
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    int rcs = 0;
    uint8_t dir = 1;
    uint8_t axis = 1;
    float lin_v = 3.0;
    float lin_a = 0.0;
    float maxdis = 50.0;
    float ft_goal = 2.0;

    DescPose desc_pos, xcenter, ycenter;
    ForceTorque ft;
    memset(&desc_pos, 0, sizeof(DescPose));
    memset(&xcenter, 0, sizeof(DescPose));
    memset(&ycenter, 0, sizeof(DescPose));
    memset(&ft, 0, sizeof(ForceTorque));

    desc_pos.tran.x = -230.959;
    desc_pos.tran.y = -364.017;
    desc_pos.tran.z = 217.5;
    desc_pos.rpy.rx = -179.004;
    desc_pos.rpy.ry = 0.002;
    desc_pos.rpy.rz = 89.999;

    ft.fx = -2.0;

    robot.MoveCart(&desc_pos, 9,0,100.0,100.0,100.0,-1.0,-1);

    robot.FT_CalCenterStart();
    robot.FT_FindSurface(rcs, dir, axis, lin_v, lin_a, maxdis, ft_goal);
    robot.MoveCart(&desc_pos, 9,0,100.0,100.0,100.0,-1.0,-1);
    robot.WaitMs(1000);

    dir = 2;
    robot.FT_FindSurface(rcs, dir, axis, lin_v, lin_a, maxdis, ft_goal);
```

(续下页)

(接上页)

```
    robot.FT_CalCenterEnd(&xcenter);
    printf("xcenter:%f,%f,%f,%f,%f,%f\n",xcenter.tran.x,xcenter.tran.y,xcenter.tran.z,
↪xcenter.rpy.rx,xcenter.rpy.ry,xcenter.rpy.rz);
    robot.MoveCart(&xcenter, 9,0,60.0,50.0,50.0,-1.0,-1);

    robot.FT_CalCenterStart();
    dir = 1;
    axis = 2;
    lin_v = 6.0;
    maxdis = 150.0;
    robot.FT_FindSurface(rcs, dir, axis, lin_v, lin_a, maxdis, ft_goal);
    robot.MoveCart(&desc_pos, 9,0,100.0,100.0,100.0,-1.0,-1);
    robot.WaitMs(1000);

    dir = 2;
    robot.FT_FindSurface(rcs, dir, axis, lin_v, lin_a, maxdis, ft_goal);
    robot.FT_CalCenterEnd(&ycenter);
    printf("ycenter:%f,%f,%f,%f,%f,%f\n",ycenter.tran.x,ycenter.tran.y,ycenter.tran.z,
↪ycenter.rpy.rx,ycenter.rpy.ry,ycenter.rpy.rz);
    robot.MoveCart(&ycenter, 9,0,60.0,50.0,50.0,0.0,-1);

    return 0;
}
```

2.1.11.26 柔顺控制开启

```
/**
* @brief  柔顺控制开启
* @param  [in] p 位置调节系数或柔顺系数
* @param  [in] force 柔顺开启力阈值，单位N
* @return  错误码
*/
errno_t  FT_ComplianceStart(float p, float force);
```

2.1.11.27 柔顺控制关闭

```
/**
* @brief  柔顺控制关闭
* @return  错误码
*/
errno_t  FT_ComplianceStop();
```

2.1.11.28 负载辨识初始化

```
/**
 * @brief 负载辨识初始化
 * @return 错误码
 */
errno_t LoadIdentifyDynFilterInit();
```

2.1.11.29 负载辨识初始化

```
/**
 * @brief 负载辨识初始化
 * @return 错误码
 */
errno_t LoadIdentifyDynVarInit();
```

2.1.11.30 负载辨识主程序

```
/**
 * @brief 负载辨识主程序
 * @param [in] joint_torque 关节扭矩
 * @param [in] joint_pos 关节位置
 * @param [in] t 采样周期
 * @return 错误码
 */
errno_t LoadIdentifyMain(double joint_torque[6], double joint_pos[6], double t);
```

2.1.11.31 获取负载辨识结果

```
/**
 * @brief 获取负载辨识结果
 * @param [in] gain
 * @param [out] weight 负载重量
 * @param [out] cog 负载质心
 * @return 错误码
 */
errno_t LoadIdentifyGetResult(double gain[12], double *weight, DescTran *cog);
```

2.1.11.32 传动带启动、停止

```
/**
 * @brief 传动带启动、停止
 * @param [in] status 状态，1-启动，0-停止
 * @return 错误码
 */
errno_t ConveyorStartEnd(uint8_t status);
```

2.1.11.33 记录 IO 检测点

```
/**
 * @brief 记录IO检测点
 * @return 错误码
 */
errno_t ConveyorPointIORecord();
```

2.1.11.34 记录 A 点

```
/**
 * @brief 记录A点
 * @return 错误码
 */
errno_t ConveyorPointARecord();
```

2.1.11.35 记录参考点

```
/**
 * @brief 记录参考点
 * @return 错误码
 */
errno_t ConveyorRefPointRecord();
```

2.1.11.36 记录 B 点

```
/**
 * @brief 记录B点
 * @return 错误码
 */
errno_t ConveyorPointBRecord();
```

2.1.11.37 传送带工件 IO 检测

```
/**
 * @brief 传送带工件IO检测
 * @param [in] max_t 最大检测时间，单位ms
 * @return 错误码
 */
errno_t ConveyorIODetect(int max_t);
```

2.1.11.38 获取物体当前位置

```
/**
 * @brief 获取物体当前位置
 * @param [in] mode
 * @return 错误码
 */
errno_t ConveyorGetTrackData(int mode);
```

2.1.11.39 传动带跟踪开始

```
/**
 * @brief 传动带跟踪开始
 * @param [in] status 状态，1-启动，0-停止
 * @return 错误码
 */
errno_t ConveyorTrackStart(uint8_t status);
```

2.1.11.40 传动带跟踪停止

```
/**
 * @brief 传动带跟踪停止
 * @return 错误码
 */
errno_t ConveyorTrackEnd();
```

2.1.11.41 传动带参数配置

```
/**
 * @brief 传动带参数配置
 * @param [in]
 * @return 错误码
 */
errno_t ConveyorSetParam(float param[5]);
```

2.1.11.42 传动带抓取点补偿

```
/**
 * @brief 传动带抓取点补偿
 * @param [in] cmp 补偿位置
 * @return 错误码
 */
errno_t ConveyorCatchPointComp(double cmp[3]);
```

2.1.11.43 直线运动

```
/**
 * @brief 直线运动
 * @param [in] status 状态，1-启动，0-停止
 * @return 错误码
 */
errno_t TrackMoveL(char name[32], int tool, int wobj, float vel, float acc, float ovl,
↪ float blendR, uint8_t flag, uint8_t type);
```

2.1.11.44 获取 SSH 公钥

```
/**
 * @brief 获取SSH公钥
 * @param [out] keygen 公钥
 * @return 错误码
 */
errno_t GetSSHKeygen(char keygen[1024]);
```

2.1.11.45 下发 SCP 指令

```
/**
 * @brief 下发SCP指令
 * @param [in] mode 0-上传（上位机->控制器），1-下载（控制器->上位机）
 * @param [in] sshname 上位机用户名
 * @param [in] sship 上位机ip地址
 * @param [in] usr_file_url 上位机文件路径
 * @param [in] robot_file_url 机器人控制器文件路径
 * @return 错误码
 */
errno_t SetSSHScpCmd(int mode, char sshname[32], char sship[32], char usr_file_
↪url[128], char robot_file_url[128]);
```

2.1.11.46 计算指定路径下文件的 MD5 值

```
/**
 * @brief 计算指定路径下文件的MD5值
 * @param [in] file_path 文件路径包含文件名，默认Traj文件夹路径为:"/fruser/traj/",如"/
↪fruser/traj/trajHelix_aima_1.txt"
 * @param [out] md5 文件MD5值
```

(续下页)

(接上页)

```
 * @return 错误码
 */
errno_t ComputeFileMD5(char file_path[256], char md5[256]);
```

2.1.11.47 获取机器人急停状态

```
/**
 * @brief 获取机器人急停状态
 * @param [out] state 急停状态, 0-非急停, 1-急停
 * @return 错误码
 */
errno_t GetRobotEmergencyStopState(uint8_t *state);
```

2.1.11.48 获取 SDK 与机器人的通讯状态

```
/**
 * @brief 获取SDK与机器人的通讯状态
 * @param [out]  state 通讯状态, 0-通讯正常, 1-通讯异常
 */
errno_t GetSDKComState(int *state);
```

2.1.11.49 获取安全停止信号

```
/**
 * @brief 获取安全停止信号
 * @param [out]  si0_state 安全停止信号SI0, 0-无效, 1-有效
 * @param [out]  si1_state 安全停止信号SI1, 0-无效, 1-有效
 */
errno_t GetSafetyStopState(uint8_t *si0_state, uint8_t *si1_state);
```

2.1.11.50 代码示例

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <unistd.h>
#include "FRRobot.h"
```

(续下页)

(接上页)

```
#include "RobotTypes.h"

using namespace std;

int main(void)
{
    FRRobot robot;                  //实例化机器人对象
    robot.RPC("192.168.58.2");      //与机器人控制器建立通信连接

    uint8_t flag = 1;
    int sensor_id = 1;
    uint8_t select[6] = {1,1,1,0,0,0};
    float ft_pid[6] = {0.0005,0.0,0.0,0.0,0.0,0.0};
    uint8_t adj_sign = 0;
    uint8_t ILC_sign = 0;
    float max_dis = 100.0;
    float max_ang = 0.0;

    ForceTorque ft;
    DescPose desc_p1, desc_p2, offset_pos;
    ExaxisPos epos;
    JointPos j1, j2;
    memset(&ft, 0, sizeof(ForceTorque));
    memset(&desc_p1, 0, sizeof(DescPose));
    memset(&desc_p2, 0, sizeof(DescPose));
    memset(&offset_pos, 0, sizeof(DescPose));
    memset(&j1, 0, sizeof(JointPos));
    memset(&j2, 0, sizeof(JointPos));
    memset(&epos, 0, sizeof(ExaxisPos));

    j1 = {-105.3,-68.0,-127.9,-75.5,90.8,77.8};
    j2 = {-105.3,-97.9,-101.5,-70.3,90.8,77.8};

    desc_p1.tran.x = -208.9;
    desc_p1.tran.y = -274.5;
    desc_p1.tran.z = 334.6;
    desc_p1.rpy.rx = 178.8;
    desc_p1.rpy.ry = -1.3;
    desc_p1.rpy.rz = 86.7;

    desc_p2.tran.x = -264.8;
    desc_p2.tran.y = -480.5;
    desc_p2.tran.z = 341.8;
```

(续下页)

(接上页)

```
    desc_p2.rpy.rx = 179.2;
    desc_p2.rpy.ry = 0.3;
    desc_p2.rpy.rz = 86.7;

    ft.fx = -10.0;
    ft.fy = -10.0;
    ft.fz = -10.0;
    robot.FT_Control(flag, sensor_id, select, &ft, ft_pid, adj_sign, ILC_sign, max_
↪dis, max_ang);
    float p = 0.00005;
    float force = 30.0;
    robot.FT_ComplianceStart(p, force);
    int count = 15;
    while (count)
    {
        robot.MoveL(&j1,&desc_p1,9,0,100.0,180.0,100.0,-1.0,&epos,0,1,&offset_pos);
        robot.MoveL(&j2,&desc_p2,9,0,100.0,180.0,100.0,-1.0,&epos,0,0,&offset_pos);
        count -= 1;
    }
    robot.FT_ComplianceStop();
    flag = 0;
    robot.FT_Control(flag, sensor_id, select, &ft, ft_pid, adj_sign, ILC_sign, max_
↪dis, max_ang);

    return 0;
}
```

2.2 C#

本文档为 C# 版本的二次开发接口文档。

重要: 机器人参数单位说明：机器人位置单位为毫米 (mm)，姿态单位为度 (°)。

重要:

1) 非特别说明的代码示例中都默认机器人已经正常开机使能；

2) 文档中的所有代码示例都默认在机器人的工作空间内没有任何干涉；

3) 实际使用测试时请采用现场机器人的数据使用。

4) 使用本 SDK 前，需通过 NuGet 查找“xmlrpcnet”包，并添加至项目引用；

2.2.1 数据结构说明

2.2.1.1 关节位置数据类型

```
/**
* @brief 关节位置数据类型
*/
struct JointPos
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] jPos;   /* 六个关节位置，单位deg */
}
```

2.2.1.2 笛卡尔空间位置数据类型

```
/**
* @brief 笛卡尔空间位置数据类型
*/
struct DescTran
{
    public double x;    /* x轴坐标，单位mm  */
    public double y;    /* y轴坐标，单位mm  */
    public double z;    /* z轴坐标，单位mm  */
}
```

2.2.1.3 欧拉角姿态数据类型

```
/**
* @brief 欧拉角姿态数据类型
*/
struct Rpy
{
public double rx;   /* 绕固定轴X旋转角度，单位：deg  */
public double ry;   /* 绕固定轴Y旋转角度，单位：deg  */
public double rz;   /* 绕固定轴Z旋转角度，单位：deg  */
}
```

2.2.1.4 笛卡尔空间位姿数据类型

```
/**
*@brief 笛卡尔空间位姿类型
*/
struct DescPose
{
    public DescTran tran;      /* 笛卡尔空间位置  */
    public Rpy rpy;                  /* 笛卡尔空间姿态  */
}
```

2.2.1.5 扩展轴位置数据类型

```
/**
* @brief 扩展轴位置数据类型
*/
struct ExaxisPos
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 4)]
    public double[] ePos;    /* 四个扩展轴位置，单位mm */
}
```

2.2.1.6 力矩传感器数据类型

```
/**
* @brief 力传感器的受力分量和力矩分量
*/
struct ForceTorque
{
    public double fx;  /* 沿x轴受力分量，单位N  */
    public double fy;  /* 沿y轴受力分量，单位N  */
    public double fz;  /* 沿z轴受力分量，单位N  */
    public double tx;  /* 绕x轴力矩分量，单位Nm */
    public double ty;  /* 绕y轴力矩分量，单位Nm */
    public double tz;  /* 绕z轴力矩分量，单位Nm */
}
```

2.2.1.7 螺旋参数数据类型

```
/**
* @brief  螺旋参数数据类型
*/
struct SpiralParam
{
    public int circle_num;             /* 螺旋圈数  */
    public float circle_angle;          /* 螺旋倾角  */
    public float rad_init;              /* 螺旋初始半径，单位mm  */
    public float rad_add;               /* 半径增量  */
    public float rotaxis_add;           /* 转轴方向增量  */
    public uint rot_direction;          /* 旋转方向，0-顺时针，1-逆时针  */
}
```

2.2.2 机器人基础

2.2.2.1 实例化机器人

```
/**
* @brief  机器人接口类构造函数
*/
Robot();
```

2.2.2.2 与控制器建立通信

```
/**
* @brief  与机器人控制器建立通信
* @param  [in] ip  控制器IP地址，出场默认为192.168.58.2
* @return 错误码
*/
int RPC(string ip);
```

2.2.2.3 与机器人断开通信

```
/**
* @brief 与机器人控制器断开通信
* @return 错误码
*/
int CloseRPC();
```

2.2.2.4 查询 SDK 版本号

```
/**
* @brief 查询 SDK 版本号
* @param [out] version SDK 版本号
* @return 错误码
*/
int GetSDKVersion(ref string version);
```

2.2.2.5 获取控制器 IP

```
/**
* @brief  获取控制器IP
* @param  [out] ip  控制器IP
* @return  错误码
*/
int GetControllerIP(ref string ip);
```

2.2.2.6 控制机器人进入或退出拖动示教模式

```
/**
* @brief  控制机器人进入或退出拖动示教模式
* @param  [in] state 0-退出拖动示教模式, 1-进入拖动示教模式
* @return  错误码
*/
int DragTeachSwitch(byte state);
```

2.2.2.7 查询机器人是否处于拖动模式

```
/**
* @brief  查询机器人是否处于拖动示教模式
* @param  [out] state 0-非拖动示教模式, 1-拖动示教模式
* @return  错误码
*/
int IsInDragTeach(ref byte state);
```

2.2.2.8 控制机器人上使能或下使能

```
/**
* @brief  控制机器人上使能或下使能，机器人上电后默认自动上使能
* @param  [in] state  0-下使能，1-上使能
* @return  错误码
*/
int RobotEnable(byte state);
```

2.2.2.9 控制机器人手自动模式切换

```
/**
* @brief 控制机器人手自动模式切换
* @param [in] mode 0-自动模式，1-手动模式
* @return 错误码
*/
int Mode(int mode);
```

2.2.2.10 代码示例

```
private void btnStandard_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    string ip = "";
    string version = "";
    byte state = 0;

    robot.GetSDKVersion(ref version);
    Console.WriteLine($"SDK version : {version}");
    robot.GetControllerIP(ref ip);
    Console.WriteLine($"controller ip : {ip}");

    robot.Mode(1);
    Thread.Sleep(1000);
    robot.DragTeachSwitch(1);
    int rtn = robot.IsInDragTeach(ref state);
    Console.WriteLine($"drag state : {state}");
    Thread.Sleep(3000);
    robot.DragTeachSwitch(0);
    Thread.Sleep(1000);
```

(续下页)

(接上页)

```
    robot.IsInDragTeach(ref state);
    Console.WriteLine($"drag state : {state}");
    Thread.Sleep(3000);
    robot.RobotEnable(0);
    Thread.Sleep(3000);
    robot.RobotEnable(1);

    robot.Mode(0);
    Thread.Sleep(1000);
    robot.Mode(1);
}
```

2.2.3 机器人运动

2.2.3.1 jog 点动

```
/**
* @brief jog 点动
* @param [in] refType 点动类型：0-关节点动，2-基坐标系下点动，4-工具坐标系下点动，8-工件坐标系下点动
* @param [in] nb 1-关节 1(或 x 轴)，2-关节 2(或 y 轴)，3-关节 3(或 z 轴)，4-关节4(或绕x轴旋转)，5-关节5(或绕y轴旋转)，6-关节6(或绕z轴旋转)
* @param [in] dir 0-负方向，1-正方向
* @param [in] vel 速度百分比，[0~100]
* @param [in] acc 加速度百分比， [0~100]
* @param [in] max_dis 单次点动最大角度，单位[°]或距离，单位[mm]
* @return 错误码
*/
int StartJOG(byte refType, byte nb, byte dir, float vel, float acc, float max_dis);
```

2.2.3.2 jog 点动减速停止

```
/**
* @brief  jog点动减速停止
* @param  [in]  ref  1-关节点动停止，3-基坐标系下点动停止，5-工具坐标系下点动停止，9-工件坐标系下点动停止
* @return  错误码
*/
int StopJOG(byte stopType);
```

2.2.3.3 jog 点动立即停止

```
/**
* @brief jog点动立即停止
* @return  错误码
*/
int ImmStopJOG();
```

2.2.3.4 代码示例

```
private void btnJOG_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    robot.SetSpeed(35);
    robot.StartJOG(0, 1, 0, 15, 20.0f, 30.0f);    //
↪单关节运动，StartJOG为非阻塞指令，运动状态下接收其他运动指令（包含StartJOG）会被丢弃
    Thread.Sleep(1000);
    robot.StopJOG(1);  //机器人单轴点动减速停止
    //robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(0, 2, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(0, 3, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(0, 4, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(0, 5, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(0, 6, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();

    robot.StartJOG(2, 1, 0, 15, 20.0f, 30.0f);    //基坐标系下点动
    Thread.Sleep(1000);
    robot.StopJOG(3);  //机器人单轴点动减速停止
    //robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(2, 2, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
```

(续下页)

(接上页)

```
    robot.ImmStopJOG();
    robot.StartJOG(2, 3, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(2, 4, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(2, 5, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(2, 6, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();

    robot.StartJOG(4, 1, 0, 15, 20.0f, 30.0f);   //工具坐标系下点动
    Thread.Sleep(1000);
    robot.StopJOG(5);  //机器人单轴点动减速停止
    //robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(4, 2, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(4, 3, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(4, 4, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(4, 5, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(4, 6, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();

    robot.StartJOG(8, 1, 0, 15, 20.0f, 30.0f);   //工件坐标系下点动
    Thread.Sleep(1000);
    robot.StopJOG(9);  //机器人单轴点动减速停止
    //robot.ImmStopJOG();  //机器人单轴点动立即停止
    robot.StartJOG(8, 2, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(8, 3, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
```

(续下页)

(接上页)

```
    robot.ImmStopJOG();
    robot.StartJOG(8, 4, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(8, 5, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
    robot.StartJOG(8, 6, 1, 15, 20.0f, 30.0f);
    Thread.Sleep(1000);
    robot.ImmStopJOG();
}
```

2.2.3.5 关节空间运动

```
/**
* @brief  关节空间运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos   目标笛卡尔位姿
* @param  [in] tool  工具坐标号, 范围[1~15]
* @param  [in] user  工件坐标号, 范围[1~15]
* @param  [in] vel  速度百分比, 范围[0~100]
* @param  [in] acc  加速度百分比, 范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子, 范围[0~100]
* @param  [in] epos  扩展轴位置, 单位mm
* @param  [in] blendT [-1.0]-运动到位(阻塞), [0~500.0]-平滑时间(非阻塞), 单位ms
* @param  [in] offset_flag  0-不偏移, 1-基坐标系/工件坐标系下偏移, 2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
int MoveJ(JointPos joint_pos, DescPose desc_pos, int tool, int user, float vel, float␣
↪acc, float ovl, ExaxisPos epos, float blendT, byte offset_flag, DescPose offset_
↪pos);
```

2.2.3.6 笛卡尔空间直线运动

```
/**
* @brief  笛卡尔空间直线运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos   目标笛卡尔位姿
* @param  [in] tool  工具坐标号, 范围[1~15]
* @param  [in] user  工件坐标号, 范围[1~15]
```

(续下页)

(接上页)

```
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @param  [in] epos  扩展轴位置，单位mm
* @param  [in] search  0-不焊丝寻位，1-焊丝寻位
* @param  [in] offset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
int MoveL(JointPos joint_pos, DescPose desc_pos, int tool, int user, float vel, float␣
↪acc, float ovl, float blendR, ExaxisPos epos, byte search, byte offset_flag,␣
↪DescPose offset_pos);
```

2.2.3.7 笛卡尔空间圆弧运动

```
/**
* @brief  笛卡尔空间圆弧运动
* @param  [in] joint_pos_p  路径点关节位置,单位deg
* @param  [in] desc_pos_p  路径点笛卡尔位姿
* @param  [in] ptool  工具坐标号，范围[1~15]
* @param  [in] puser  工件坐标号，范围[1~15]
* @param  [in] pvel  速度百分比，范围[0~100]
* @param  [in] pacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_p  扩展轴位置，单位mm
* @param  [in] poffset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos_p  位姿偏移量
* @param  [in] joint_pos_t  目标点关节位置,单位deg
* @param  [in] desc_pos_t  目标点笛卡尔位姿
* @param  [in] ttool  工具坐标号，范围[1~15]
* @param  [in] tuser  工件坐标号，范围[1~15]
* @param  [in] tvel  速度百分比，范围[0~100]
* @param  [in] tacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_t  扩展轴位置，单位mm
* @param  [in] toffset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos_t  位姿偏移量
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @return  错误码
*/
int MoveC(JointPos joint_pos_p, DescPose desc_pos_p, int ptool, int puser, float pvel,
↪ float pacc, ExaxisPos epos_p, byte poffset_flag, DescPose offset_pos_p, JointPos␣
```

(续下页)

(接上页)

```
↪joint_pos_t, DescPose desc_pos_t, int ttool, int tuser, float tvel, float tacc,␣
↪ExaxisPos epos_t, byte toffset_flag, DescPose offset_pos_t, float ovl, float␣
↪blendR);
```

2.2.3.8 笛卡尔空间整圆运动

```
/**
* @brief  笛卡尔空间整圆运动
* @param  [in] joint_pos_p  路径点1关节位置,单位deg
* @param  [in] desc_pos_p  路径点1笛卡尔位姿
* @param  [in] ptool  工具坐标号，范围[1~15]
* @param  [in] puser  工件坐标号，范围[1~15]
* @param  [in] pvel  速度百分比，范围[0~100]
* @param  [in] pacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_p  扩展轴位置，单位mm
* @param  [in] joint_pos_t  路径点2关节位置,单位deg
* @param  [in] desc_pos_t  路径点2笛卡尔位姿
* @param  [in] ttool  工具坐标号，范围[1~15]
* @param  [in] tuser  工件坐标号，范围[1~15]
* @param  [in] tvel  速度百分比，范围[0~100]
* @param  [in] tacc  加速度百分比，范围[0~100],暂不开放
* @param  [in] epos_t  扩展轴位置，单位mm
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @param  [in] offset_flag  0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
int Circle(JointPos joint_pos_p, DescPose desc_pos_p, int ptool, int puser, float␣
↪pvel, float pacc, ExaxisPos epos_p, JointPos joint_pos_t, DescPose desc_pos_t, int␣
↪ttool, int tuser, float tvel, float tacc, ExaxisPos epos_t, float ovl, byte offset_
↪flag, DescPose offset_pos);
```

2.2.3.9 代码示例

```
private void btnMovetest_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    JointPos j1, j2, j3, j4;
    DescPose desc_pos1, desc_pos2, desc_pos3, desc_pos4, offset_pos;
```

(续下页)

(接上页)

```
    ExaxisPos epos;

    j1 = new JointPos(-58.982, -90.717, 127.647, -129.041, -87.989, -0.062);
    desc_pos1 = new DescPose(-437.039, 411.064, 426.189, -177.886, 2.007, 31.155);

    j2 = new JointPos(-58.978, -76.817, 112.494, -127.348, -89.145, -0.063);
    desc_pos2 = new DescPose(-525.55, 562.3, 417.199, -178.325, 0.847, 31.109);

    j3 = new JointPos(-71.746, -87.177, 123.953, -126.25, -89.429, -0.089);
    desc_pos3 = new DescPose(-345.155, 535.733, 421.269, 179.475, 0.571, 18.332);

    ExaxisPos ePos = new ExaxisPos(0, 0, 0, 0);
    DescPose offset = new DescPose();

    int tool = 0;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = 0.0f;
    float blendR = 0.0f;
    byte flag = 0;
    byte search = 0;

    robot.SetSpeed(20);
    int err = -1;
    err = robot.MoveJ(j1, desc_pos1, tool, user, vel, acc, ovl, ePos, blendT, flag,␣
↪offset);
    Console.WriteLine($"movej errcode:  {err}");

    Thread.Sleep(1000);
    err = robot.MoveL(j2, desc_pos2, tool, user, vel, acc, ovl, blendR, ePos, search,␣
↪flag, offset);
    Console.WriteLine($"moveL errcode:  {err}");

    Thread.Sleep(1000);
    err = robot.MoveL(j1, desc_pos1, tool, user, vel, acc, ovl, blendR, ePos, search,␣
↪flag, offset);
    Console.WriteLine($"moveL errcode:  {err}");

    Thread.Sleep(1000);
    err = robot.MoveC(j2, desc_pos2, tool, user, vel, acc, ePos, flag, offset, j3,␣
↪desc_pos3, tool, user, vel, acc, ePos, flag, offset, ovl, blendR);
```

(续下页)

(接上页)

```
    Console.WriteLine($"circle errcode:  {err}");

    Thread.Sleep(1000);
    err = robot.MoveJ(j1, desc_pos1, tool, user, vel, acc, ovl, ePos, blendT, flag,␣
↪offset);
    Console.WriteLine($"movej errcode:  {err}");

    Thread.Sleep(1000);
    err = robot.Circle(j2, desc_pos2, tool, user, vel, acc, ePos, j3, desc_pos3, tool,
↪ user, vel, acc, ePos, ovl, flag, offset);
    Console.WriteLine($"circle errcode:  {err}");
}
```

2.2.3.10 笛卡尔空间螺旋线运动

```
/**
* @brief 笛卡尔空间螺旋线运动
* @param [in] joint_pos 目标关节位置,单位 deg
* @param [in] desc_pos 目标笛卡尔位姿
* @param [in] tool 工具坐标号，范围[0~14]
* @param [in] user 工件坐标号，范围[0~14]
* @param [in] vel 速度百分比，范围[0~100]
* @param [in] acc 加速度百分比，范围[0~100],暂不开放
* @param [in] epos 扩展轴位置，单位 mm
* @param [in] ovl 速度缩放因子，范围[0~100]
* @param [in] offset_flag 0-不偏移，1-基坐标系/工件坐标系下偏移，2-工具坐标系下偏移
* @param [in] offset_pos 位姿偏移量
* @param [in] spiral_param 螺旋参数
* @return 错误码
*/
int NewSpiral(JointPos joint_pos, DescPose desc_pos, int tool, int user, float vel,␣
↪float acc, ExaxisPos epos, float ovl, byte offset_flag, DescPose offset_pos,␣
↪SpiralParam spiral_param);
```

2.2.3.11 代码示例

```
private void btnDescSpiral_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");
    JointPos j;
    DescPose desc_pos;
    DescPose offset_pos1 = new DescPose(0, 0, 0, 0, 0, 0);
    DescPose offset_pos2 = new DescPose(0, 0, 0, 0, 0, 0);
    ExaxisPos epos = new ExaxisPos(0, 0, 0, 0);
    SpiralParam sp;

    j = new JointPos(-58.982, -90.717, 127.647, -129.041, -87.989, -0.062);
    desc_pos = new DescPose(-437.039, 411.064, 426.189, -177.886, 2.007, 31.155);

    offset_pos1.tran.x = 50.0;
    offset_pos1.rpy.rx = -30.0;
    offset_pos2.tran.x = 50.0;
    offset_pos2.rpy.rx = -5.0;

    sp.circle_num = 5;
    sp.circle_angle = 1.0f;
    sp.rad_init = 10.0f;
    sp.rad_add = 40.0f;
    sp.rotaxis_add = 10.0f;
    sp.rot_direction = 0;

    int tool = 0;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = 0.0f;
    byte flag = 2;

    robot.SetSpeed(20);
    int ret = robot.GetForwardKin(j, ref desc_pos);  //
↪只有关节位置的情况下，可用正运动学接口求解笛卡尔空间坐标
    if (ret == 0)
    {
        int err = -1;
        err = robot.MoveJ(j, desc_pos, tool, user, vel, acc, ovl, epos, blendT, flag,␣
↪offset_pos1);
```

(续下页)

(接上页)

```
        Console.WriteLine($"movej errcode:  {err}");

        err = robot.NewSpiral(j, desc_pos, tool, user, vel, acc, epos, ovl, flag,␣
↪offset_pos2, sp);
        Console.WriteLine($"newspiral errcode:  {err}");
    }
    else
    {
        Console.WriteLine($"GetForwardKin errcode: {ret}");
    }
}
```

2.2.3.12 伺服运动开始

```
/**
* @brief 伺服运动开始，配合ServoJ、ServoCart指令使用
* @return 错误码
*/
int ServoMoveStart();
```

2.2.3.13 伺服运动结束

```
/**
* @brief 伺服运动结束，配合ServoJ、ServoCart指令使用
* @return 错误码
*/
int ServoMoveEnd();
```

2.2.3.14 关节空间伺服模式运动

```
/**
* @brief  关节空间伺服模式运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放，默认为0
* @param  [in] vel  速度百分比，范围[0~100]，暂不开放，默认为0
* @param  [in] cmdT  指令下发周期，单位s，建议范围[0.001~0.0016]
* @param  [in] filterT 滤波时间，单位s，暂不开放，默认为0
* @param  [in] gain  目标位置的比例放大器，暂不开放，默认为0
* @return  错误码
*/
```

(续下页)

(接上页)

```
int ServoJ(JointPos joint_pos, float acc, float vel, float cmdT, float filterT, float␣
↪gain);
```

2.2.3.15 代码示例

```
private void btnJointServoMove_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    JointPos j = new JointPos(0, 0, 0, 0, 0, 0);

    float vel = 0.0f;
    float acc = 0.0f;
    float cmdT = 0.008f;
    float filterT = 0.0f;
    float gain = 0.0f;
    byte flag = 0;
    int count = 200;
    float dt = 0.1f;
    int ret = robot.GetActualJointPosDegree(flag, ref j);
    if (ret == 0)
    {
        while (count > 0)
        {
            robot.ServoJ(j, acc, vel, cmdT, filterT, gain);
            j.jPos[0] += dt;
            count -= 1;
            robot.WaitMs((int)(cmdT * 1000));
        }
    }
    else
    {
        Console.WriteLine($"GetActualJointPosDegree errcode:  {ret}");
    }
}
```

2.2.3.16 笛卡尔空间伺服模式运动

```
/**
* @brief  笛卡尔空间伺服模式运动
* @param  [in]  mode  0-绝对运动(基坐标系), 1-增量运动(基坐标系), 2-增量运动(工具坐标系)
* @param  [in]  desc_pos  目标笛卡尔位姿或位姿增量
* @param  [in]  pos_gain  位姿增量比例系数, 仅在增量运动下生效, 范围[0~1]
* @param  [in] acc  加速度百分比, 范围[0~100],暂不开放, 默认为0
* @param  [in] vel  速度百分比, 范围[0~100], 暂不开放, 默认为0
* @param  [in] cmdT  指令下发周期, 单位s, 建议范围[0.001~0.0016]
* @param  [in] filterT 滤波时间, 单位s, 暂不开放, 默认为0
* @param  [in] gain  目标位置的比例放大器, 暂不开放, 默认为0
* @return  错误码
*/
int ServoCart(int mode, DescPose desc_pos, double[] pos_gain, float acc, float vel, float cmdT, float filterT, float gain);
```

2.2.3.17 代码示例

```
private void btnDescServoMove_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    DescPose desc_pos_dt = new DescPose(0, 0, 0, 0, 0, 0);
    desc_pos_dt.tran.z = -0.5;
    double[] pos_gain = new double[6]{ 0.0, 0.0, 1.0, 0.0, 0.0, 0.0 };
    int mode = 2;
    float vel = 0.0f;
    float acc = 0.0f;
    float cmdT = 0.008f;
    float filterT = 0.0f;
    float gain = 0.0f;
    int flag = 0;
    int count = 500;

    robot.SetSpeed(20);
    while (count > 0)
    {
        robot.ServoCart(mode, desc_pos_dt, pos_gain, acc, vel, cmdT, filterT, gain);
        count -= 1;
        robot.WaitMs((int)(cmdT * 1000));
```

(续下页)

(接上页)

```
    }
}
```

2.2.3.18 笛卡尔空间点到点运动

```
/**
* @brief 笛卡尔空间点到点运动
* @param [in] desc_pos 基坐标系下目标笛卡尔位姿
* @param [in] tool 工具坐标号，范围[0~14]
* @param [in] user 工件坐标号，范围[0~14]
* @param [in] vel 速度百分比，范围[0~100]
* @param [in] acc 加速度百分比，范围[0~100],暂不开放
* @param [in] ovl 速度缩放因子，范围[0~100]
* @param [in] blendT [-1.0]-运动到位(阻塞)，[0~500.0]-平滑时间(非阻塞)，单位 ms
* @param [in] config 关节空间配置，[-1]-参考当前关节位置解算，[0~7]-
↪参考特定关节空间配置解算，默认为-1
* @return 错误码
*/
int MoveCart(DescPose desc_pos, int tool, int user, float vel, float acc, float ovl,␣
↪float blendT, int config);
```

2.2.3.19 代码示例

```
private void btnDescPTPMove_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    DescPose desc_pos1, desc_pos2, desc_pos3;
    desc_pos1 = new DescPose(-437.039, 411.064, 426.189, -177.886, 2.007, 31.155);
    desc_pos2 = new DescPose(-525.55, 562.3, 417.199, -178.325, 0.847, 31.109);
    desc_pos3 = new DescPose(-345.155, 535.733, 421.269, 179.475, 0.571, 18.332);

    int tool = 0;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = -1.0f;
    float blendT1 = 0.0f;
    int config = -1;
```

(续下页)

(接上页)

```
    robot.SetSpeed(20);
    robot.MoveCart(desc_pos1, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveCart(desc_pos2, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveCart(desc_pos3, tool, user, vel, acc, ovl, blendT1, config);
}
```

2.2.3.20 样条运动开始

```
/**
* @brief  样条运动开始
* @return  错误码
*/
int SplineStart();
```

2.2.3.21 样条运动 PTP

```
/**
* @brief  关节空间样条运动
* @param  [in] joint_pos  目标关节位置,单位deg
* @param  [in] desc_pos  目标笛卡尔位姿
* @param  [in] tool  工具坐标号，范围[1~15]
* @param  [in] user  工件坐标号，范围[1~15]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] acc  加速度百分比，范围[0~100],暂不开放
* @param  [in] ovl  速度缩放因子，范围[0~100]
* @return  错误码
*/
int SplinePTP(JointPos joint_pos, DescPose desc_pos, int tool, int user, float vel,␣
↪float acc, float ovl);
```

2.2.3.22 样条运动结束

```
/**
* @brief  样条运动结束
* @return  错误码
*/
int SplineEnd();
```

2.2.3.23 代码示例

```
private void btnSplineMove_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    JointPos j1, j2, j3, j4;
    DescPose desc_pos1, desc_pos2, desc_pos3, desc_pos4, offset_pos;
    ExaxisPos epos = new ExaxisPos(0, 0, 0, 0);

    j1 = new JointPos(-58.982, -90.717, 127.647, -129.041, -87.989, -0.062);
    desc_pos1 = new DescPose(-437.039, 411.064, 426.189, -177.886, 2.007, 31.155);

    j2 = new JointPos(-58.978, -76.817, 112.494, -127.348, -89.145, -0.063);
    desc_pos2 = new DescPose(-525.55, 562.3, 417.199, -178.325, 0.847, 31.109);

    j3 = new JointPos(-49.129, -68.49, 103.297, -128.898, -91.478, -0.062);
    desc_pos3 = new DescPose(-680.308, 547.378, 399.189, -175.909, -1.479, 40.827);

    j4 = new JointPos(-56.126, -54.093, 80.686, -121.655, -91.428, -0.064);
    desc_pos4 = new DescPose(-719.201, 790.816, 389.118, -174.939, -1.428, 33.809);

    offset_pos = new DescPose(0, 0, 0, 0, 0, 0);

    int tool = 0;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = -1.0f;
    byte flag = 0;
    robot.SetSpeed(20);
    int err = -1;
    err = robot.MoveJ(j1, desc_pos1, tool, user, vel, acc, ovl, epos, blendT, flag,␣
↪offset_pos);
    Console.WriteLine($"movej errcode:  {err}");

    robot.SplineStart();
    robot.SplinePTP(j1, desc_pos1, tool, user, vel, acc, ovl);
    robot.SplinePTP(j2, desc_pos2, tool, user, vel, acc, ovl);
    robot.SplinePTP(j3, desc_pos3, tool, user, vel, acc, ovl);
    robot.SplinePTP(j4, desc_pos4, tool, user, vel, acc, ovl);
    robot.SplineEnd();
```

(续下页)

(接上页)

```
}
```

2.2.3.24 新样条运动开始

```
/**
* @brief 新样条运动开始
* @param [in] type  0-圆弧过渡，1-给定点位为路径点
* @return 错误码
*/
int NewSplineStart(int type);
```

2.2.3.25 新样条指令点

```
/**
* @brief 增加样条运动指令点
* @param [in] joint_pos 目标关节位置,单位 deg
* @param [in] desc_pos 目标笛卡尔位姿
* @param [in] tool 工具坐标号，范围[0~14]
* @param [in] user 工件坐标号，范围[0~14]
* @param [in] vel 速度百分比，范围[0~100]
* @param [in] acc 加速度百分比，范围[0~100],暂不开放
* @param [in] ovl 速度缩放因子，范围[0~100]
* @param [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @param [in] lastFlag  是否为最后一个点，0-否，1-是
* @return 错误码
*/
int NewSplinePoint(JointPos joint_pos, DescPose desc_pos, int tool, int user, float␣
↪vel, float acc, float ovl, float blendR, int lastFlag);
```

2.2.3.26 新样条运动结束

```
/**
* @brief 新样条运动开始
* @return 错误码
*/
int NewSplineEnd();
```

2.2.3.27 终止运动

```
/**
* @brief 终止运动
* @return  错误码
*/
int StopMotion();
```

2.2.3.28 暂停运动

```
/**
  * @brief 暂停运动
  * @return 错误码
*/
int PauseMotion();
```

2.2.3.29 恢复运动

```
/**
* @brief 恢复运动
* @return 错误码
*/
int ResumeMotion();
```

2.2.3.30 点位整体偏移开始

```
/**
* @brief  点位整体偏移开始
* @param  [in]  flag  0-基坐标系下/工件坐标系下偏移，2-工具坐标系下偏移
* @param  [in] offset_pos  位姿偏移量
* @return  错误码
*/
int PointsOffsetEnable(int flag, DescPose offset_pos);
```

2.2.3.31 点位整体偏移结束

```
/**
* @brief  点位整体偏移结束
* @return  错误码
*/
int PointsOffsetDisable();
```

2.2.3.32 代码示例

```
private void btnPointOffect_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    JointPos j1, j2;
    DescPose desc_pos1, desc_pos2, offset_pos, offset_pos1;
    ExaxisPos epos = new ExaxisPos(0, 0, 0, 0);

    j1 = new JointPos(-58.982, -90.717, 127.647, -129.041, -87.989, -0.062);
    desc_pos1 = new DescPose(-437.039, 411.064, 426.189, -177.886, 2.007, 31.155);

    j2 = new JointPos(-58.978, -76.817, 112.494, -127.348, -89.145, -0.063);
    desc_pos2 = new DescPose(-525.55, 562.3, 417.199, -178.325, 0.847, 31.109);

    offset_pos = new DescPose(0, 0, 0, 0, 0, 0);
    offset_pos1 = new DescPose(50.0, 50.0, 50.0, 5.0, 5.0, 5.0);

    int tool = 0;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = -1.0f;
    float blendR = 0.0f;
    byte flag = 0;
    int type = 0;

    robot.SetSpeed(20);

    robot.MoveJ(j1, desc_pos1, tool, user, vel, acc, ovl, epos, blendT, flag, offset_
↪pos);
    robot.MoveJ(j2, desc_pos2, tool, user, vel, acc, ovl, epos, blendT, flag, offset_
```

(续下页)

(接上页)

```
↪pos);
    Thread.Sleep(1000);
    robot.PointsOffsetEnable(type, offset_pos1);
    robot.MoveJ(j1, desc_pos1, tool, user, vel, acc, ovl, epos, blendT, flag, offset_
↪pos);
    robot.MoveJ(j2, desc_pos2, tool, user, vel, acc, ovl, epos, blendT, flag, offset_
↪pos);
    robot.PointsOffsetDisable();
}
```

2.2.4 机器人 IO

2.2.4.1 设置控制箱数字量输出

```
/**
* @brief  设置控制箱数字量输出
* @param  [in] id  io编号，范围[0~15]
* @param  [in] status 0-关，1-开
* @param  [in] smooth 0-不平滑，  1-平滑
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
int SetDO(int id, byte status, byte smooth, byte block);
```

2.2.4.2 设置工具数字量输出

```
/**
* @brief  设置工具数字量输出
* @param  [in] id  io编号，范围[0~1]
* @param  [in] status 0-关，1-开
* @param  [in] smooth 0-不平滑，  1-平滑
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
int SetToolDO(int id, byte status, byte smooth, byte block);
```

2.2.4.3 设置控制箱模拟量输出

```
/**
* @brief  设置控制箱模拟量输出
* @param  [in] id  io编号，范围[0~1]
* @param  [in] value 电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
int SetAO(int id, float value, byte block);
```

2.2.4.4 设置工具模拟量输出

```
/**
* @brief  设置工具模拟量输出
* @param  [in] id  io编号，范围[0]
* @param  [in] value 电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
int SetToolAO(int id, float value, byte block);
```

2.2.4.5 获取控制箱数字量输入

```
/**
* @brief  获取控制箱数字量输入
* @param  [in] id  io编号，范围[0~15]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  0-低电平，1-高电平
* @return  错误码
*/
int GetDI(int id, byte block, ref byte level);
```

2.2.4.6 获取工具数字量输入

```
/**
* @brief  获取工具数字量输入
* @param  [in] id  io编号，范围[0~1]
* @param  [in] block  0-阻塞，1-非阻塞
* @param  [out] result  0-低电平，1-高电平
```

(续下页)

(接上页)

```
* @return  错误码
*/
int GetToolDI(int id, byte block, ref byte level);
```

2.2.4.7 等待控制箱数字量输入

```
/**
* @brief 等待控制箱数字量输入
* @param [in] id io编号，范围[0~15]
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
int WaitDI(int id, byte status, int max_time, int opt);
```

2.2.4.8 等待控制箱多路数字量输入

```
/**
* @brief 等待控制箱多路数字量输入
* @param [in] mode 0-多路与，1-多路或
* @param [in] id io编号，bit0~bit7对应DI0~DI7，bit8~bit15对应CI0~CI7
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
int WaitMultiDI(int mode, int id, byte status, int max_time, int opt);
```

2.2.4.9 等待工具数字量输入

```
/**
* @brief 等待工具数字量输入
* @param [in] id io编号，范围[0~1]
* @param [in] status 0-关，1-开
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
```

(续下页)

(接上页)

```
 * @return  错误码
 */
int WaitToolDI(int id, byte status, int max_time, int opt);
```

2.2.4.10 获取控制箱模拟量输入

```
/**
 * @brief  获取控制箱模拟量输入
 * @param  [in] id  io编号，范围[0~1]
 * @param  [in] block  0-阻塞，1-非阻塞
 * @param  [out] result  输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
 * @return  错误码
 */
int GetAI(int id, byte block, ref float persent);
```

2.2.4.11 获取工具模拟量输入

```
/**
 * @brief  获取工具模拟量输入
 * @param  [in] id  io编号，范围[0]
 * @param  [in] block  0-阻塞，1-非阻塞
 * @param  [out] result  输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
 * @return  错误码
 */
int GetToolAI(int id, byte block, ref float persent);
```

2.2.4.12 获取机器人末端记录按钮状态

```
/**
 * @brief 获取机器人末端记录按钮状态
 * @param [out] state 按钮状态，0-按下，1-松开
 * @return 错误码
 */
int GetAxlePointRecordBtnState(ref byte state);
```

2.2.4.13 等待控制箱模拟量输入

```
/**
* @brief 等待控制箱模拟量输入
* @param [in] id io编号，范围[0~1]
* @param [in] sign 0-大于，1-小于
* @param [in] value 输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
int WaitAI(int id, int sign, float value, int max_time, int opt);
```

2.2.4.14 等待工具模拟量输入

```
/**
* @brief 等待工具模拟量输入
* @param [in] id io编号，范围[0]
* @param [in] sign 0-大于，1-小于
* @param [in] value 输入电流或电压值百分比，范围[0~100]对应电流值[0~20mS]或电压[0~10V]
* @param [in] max_time 最大等待时间，单位ms
* @param [in] opt 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
* @return 错误码
*/
int WaitToolAI(int id, int sign, float value, int max_time, int opt);
```

2.2.4.15 获取机器人末端 DO 输出状态

```
/**
* @brief 获取机器人末端DO输出状态
* @param [out] do_state DO输出状态，do0~do1对应bit1~bit2,从bit0开始
* @return 错误码
*/
int GetToolDO(ref byte do_state);
```

2.2.4.16 获取机器控制器 DO 输出状态

```
/**
* @brief 获取机器人控制器DO输出状态
* @param [out] do_state_h DO输出状态，co0~co7对应bit0~bit7
* @param [out] do_state_l DO输出状态，do0~do7对应bit0~bit7
* @return 错误码
*/
int GetDO(ref int do_state_h, ref int do_state_l);
```

2.2.4.17 代码示例

```
private void btnIOTest_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    byte status = 1;
    byte smooth = 0;
    byte block = 0;
    byte di = 0, tool_di = 0;
    float ai = 0.0f, tool_ai = 0.0f;
    float value = 0.0f;
    int i;

    for (i = 0; i < 16; i++)//所有控制器IO输出置 1
    {
        robot.SetDO(i, status, smooth, block);
        robot.WaitMs(500);
    }

    status = 0;

    for (i = 0; i < 16; i++)//所有控制器IO输出置 0
    {
        robot.SetDO(i, status, smooth, block);
        robot.WaitMs(500);
    }

    status = 1;

    for (i = 0; i < 2; i++)//所有工具端IO输出置 1
    {
```

(续下页)

(接上页)

```
        robot.SetToolDO(i, status, smooth, block);
        robot.WaitMs(500);
    }
    status = 0;
    for (i = 0; i < 2; i++)//所有工具端IO输出置 0
    {
        robot.SetToolDO(i, status, smooth, block);
        robot.WaitMs(500);
    }

    value = 50.0f;
    robot.SetAO(0, value, block);//设置控制器0号模拟量输出50%
    value = 100.0f;
    robot.SetAO(1, value, block);//设置控制器1号模拟量输出100%
    robot.WaitMs(300);
    value = 0.0f;
    robot.SetAO(0, value, block);//设置控制器0号模拟量输出0%
    value = 0.0f;
    robot.SetAO(1, value, block);//设置控制器1号模拟量输出0%

    value = 100.0f;
    robot.SetToolAO(0, value, block);//设置工具端0号模拟量输出100%
    robot.WaitMs(1000);
    value = 0.0f;
    robot.SetToolAO(0, value, block);//设置工具端0号模拟量输出0%

    robot.GetDI(0, block, ref di);//获取数字输入
    Console.WriteLine($"di0 : {di}");
    robot.WaitDI(0, 1, 0, 2);       //等待0号端口数字量输入1, 一直等待
    Console.WriteLine("wait di success");
    robot.WaitMultiDI(0, 3, 0, 10000, 2);   //等待多路与, 
↪0和1端口, 输入置1, 等待时间10000ms,  一直等待
    Console.WriteLine("wait multi di success");
    robot.GetToolDI(1, block, ref tool_di);//获取工具端数字量输入
    Console.WriteLine($"tool_di1 : {tool_di}");
    robot.WaitToolDI(1, 0, 0, 2);          //一直等待
    Console.WriteLine("wait tool di success");
    robot.GetAI(0, block, ref ai);
    Console.WriteLine($"ai0 : {ai}");
    robot.GetAI(1, block, ref ai);
    Console.WriteLine($"ai1 : {ai}");
    robot.WaitAI(0, 1, 50.0f, 0, 2);    //等待0号口,  小于 ,  %50,  一直等待
    Console.WriteLine("wait ai success");
```

(续下页)

(接上页)

```
    robot.WaitToolAI(0, 1, 50, 0, 2);    //一直等待
    Console.WriteLine("wait tool ai success");
    robot.GetToolAI(0, block, ref tool_ai);
    Console.WriteLine($"tool_ai0 : {tool_ai}");
}
```

2.2.5 机器人常用设置

2.2.5.1 设置全局速度

```
/**
* @brief  设置全局速度
* @param  [in]  vel  速度百分比, 范围[0~100]
* @return  错误码
*/
int SetSpeed(int vel);
```

2.2.5.2 设置系统变量值

```
/**
* @brief  设置系统变量值
* @param  [in]  id  变量编号, 范围[1~20]
* @param  [in]  value 变量值
* @return  错误码
*/
int SetSysVarValue(int id, double value);
```

2.2.5.3 设置工具参考点-六点法

```
/**
* @brief 设置工具参考点-六点法
* @param [in] point_num 点编号,范围[1~6]
* @return 错误码
*/
int SetToolPoint(int point_num);
```

2.2.5.4 计算工具坐标系-六点法

```
/**
* @brief 计算工具坐标系
* @param [out] tcp_pose 工具坐标系
* @return 错误码
*/
int ComputeTool(ref DescPose tcp_pose);
```

2.2.5.5 设置工具参考点-四点法

```
/**
* @brief 设置工具参考点-四点法
* @param [in] point_num 点编号,范围[1~4]
* @return 错误码
*/
```

2.2.5.6 计算工具坐标系-四点法

```
/**
* @brief 计算工具坐标系
* @param [out] tcp_pose 工具坐标系
* @return 错误码
*/
int ComputeTcp4(ref DescPose tcp_pose);
```

2.2.5.7 设置工具坐标系

```
/**
* @brief 设置工具坐标系
* @param [in] id 坐标系编号，范围[0~14]
* @param [in] coord 工具中心点相对于末端法兰中心位姿
* @param [in] type 0-工具坐标系，1-传感器坐标系
* @param [in] install 安装位置，0-机器人末端，1-机器人外部
* @return 错误码
*/
int SetToolCoord(int id, DescPose coord, int type, int install);
```

2.2.5.8 设置工具坐标系列表

```
/**
* @brief  设置工具坐标系列表
* @param  [in] id 坐标系编号，范围[1~15]
* @param  [in] coord  工具中心点相对于末端法兰中心位姿
* @param  [in] type  0-工具坐标系，1-传感器坐标系
* @param  [in] install 安装位置，0-机器人末端，1-机器人外部
* @return  错误码
*/
int SetToolList(int id, DescPose coord, int type, int install);
```

2.2.5.9 设置外部工具坐标参考点-三点法

```
/**
* @brief 设置外部工具参考点-三点法
* @param [in] point_num 点编号,范围[1~3]
* @return 错误码
*/
int SetExTCPPoint(int point_num);
```

2.2.5.10 计算外部工具坐标系-三点法

```
/**
* @brief 计算外部工具坐标系-三点法
* @param [out] tcp_pose 外部工具坐标系
* @return 错误码
*/
int ComputeExTCF(ref DescPose tcp_pose);
```

2.2.5.11 设置外部工具坐标系

```
/**
* @brief 设置外部工具坐标系
* @param [in] id 坐标系编号，范围[0~14]
* @param [in] etcp 工具中心点相对末端法兰中心位姿
* @param [in] etool 待定
* @return 错误码
*/
int SetExToolCoord(int id, DescPose etcp, DescPose etool);
```

2.2.5.12 设置外部工具坐标系列表

```
/**
* @brief  设置外部工具坐标系列表
* @param  [in] id 坐标系编号, 范围[0~14]
* @param  [in] etcp  工具中心点相对末端法兰中心位姿
* @param  [in] etool  待定
* @return  错误码
*/
int SetExToolList(int id, DescPose etcp, DescPose etool);
```

2.2.5.13 设置工件坐标系参考点-三点法

```
/**
* @brief 设置工件参考点-三点法
* @param [in] point_num 点编号,范围[1~3]
* @return 错误码
*/
int SetWObjCoordPoint(int point_num);
```

2.2.5.14 计算工件坐标系

```
/**
* @brief 计算工件坐标系
* @param [in] method 计算方式 0: 原点-x轴-z轴  1: 原点-x轴-xy平面
* @param [out] wobj_pose 工件坐标系
* @return 错误码
*/
int ComputeWObjCoord(int method, ref DescPose wobj_pose);
```

2.2.5.15 设置工件坐标系

```
/**
* @brief  设置工件坐标系
* @param  [in] id 坐标系编号, 范围[0~14]
* @param  [in] coord  工件坐标系相对于末端法兰中心位姿
* @return  错误码
*/
int SetWObjCoord(int id, DescPose coord);
```

2.2.5.16 设置工件坐标系列表

```
/**
* @brief  设置工件坐标系列表
* @param  [in] id 坐标系编号，范围[0~14]
* @param  [in] coord  工件坐标系相对于末端法兰中心位姿
* @return  错误码
*/
int SetWObjList(int id, DescPose coord);
```

2.2.5.17 设置末端负载重量

```
/**
* @brief  设置末端负载重量
* @param  [in] weight  负载重量，单位kg
* @return  错误码
*/
int SetLoadWeight(float weight);
```

2.2.5.18 设置末端负载质心坐标

```
/**
* @brief  设置末端负载质心坐标
* @param  [in] coord 质心坐标，单位mm
* @return  错误码
*/
int SetLoadCoord(DescTran coord);
```

2.2.5.19 设置机器人安装方式

```
/**
* @brief  设置机器人安装方式
* @param  [in] install  安装方式，0-正装，1-侧装，2-倒装
* @return  错误码
*/
int SetRobotInstallPos(byte install);
```

2.2.5.20 设置机器人安装角度

```
/**
* @brief  设置机器人安装角度，自由安装
* @param  [in] yangle  倾斜角
* @param  [in] zangle  旋转角
* @return  错误码
*/
int SetRobotInstallAngle(double yangle, double zangle);
```

2.2.5.21 代码示例

```
private void btnCommonSets_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    int i;
    double value = 0;
    int id;
    int type;
    int install;

    DescTran coord = new DescTran();
    DescPose t_coord, etcp, etool, w_coord;
    t_coord = new DescPose();
    etcp = new DescPose();
    w_coord = new DescPose();

    robot.SetSpeed(20);

    for (i = 1; i < 21; i++)
    {
        robot.SetSysVarValue(i, (float)(i + 0.5));
        robot.WaitMs(100);
    }

    for (i = 1; i < 21; i++)
    {
        robot.GetSysVarValue(i, ref value);
        Console.WriteLine($"sys value : {value}");
    }

```

(续下页)

(接上页)

```
    robot.SetLoadWeight((float)2.5);
    coord.x = 3.0;
    coord.y = 4.0;
    coord.z = 5.0;
    robot.SetLoadCoord(coord);

    id = 3;
    t_coord.tran.x = 1.0;
    t_coord.tran.y = 2.0;
    t_coord.tran.z = 300.0;
    t_coord.rpy.rx = 4.0;
    t_coord.rpy.ry = 5.0;
    t_coord.rpy.rz = 6.0;
    type = 0;
    install = 0;

    int rtn1 = -1;
    int rtn2 = -1;
    rtn1 = robot.SetToolCoord(id, t_coord, type, install);
    rtn2 = robot.SetToolList(id, t_coord, type, install);
    Console.WriteLine($"set tool coord result {rtn1}, set tool list rtn{rtn2}");

    etcp.tran.x = 1.0;
    etcp.tran.y = 2.0;
    etcp.tran.z = 3.0;
    etcp.rpy.rx = 4.0;
    etcp.rpy.ry = 5.0;
    etcp.rpy.rz = 6.0;
    etool.tran.x = 11.0;
    etool.tran.y = 22.0;
    etool.tran.z = 330.0;
    etool.rpy.rx = 44.0;
    etool.rpy.ry = 55.0;
    etool.rpy.rz = 66.0;
    id = 5;
    robot.SetExToolCoord(id, etcp, etool);
    robot.SetExToolList(id, etcp, etool);

    w_coord.tran.x = 110.0;
    w_coord.tran.y = 12.0;
    w_coord.tran.z = 13.0;
    w_coord.rpy.rx = 14.0;
    w_coord.rpy.ry = 15.0;
```

(续下页)

(接上页)

```
    w_coord.rpy.rz = 16.0;
    id = 12;
    robot.SetWObjCoord(id, w_coord);
    //robot.SetWObjList(id, w_coord);

    double yangle = 0, zangle = 0;
    robot.SetRobotInstallPos(1);//侧装
    robot.SetRobotInstallAngle(15.0, 25.0);
    Thread.Sleep(1000);
    robot.GetRobotInstallAngle(ref yangle, ref zangle);
    Console.WriteLine($"yangle  {yangle}   zangle  {zangle}");
    robot.SetRobotInstallAngle(10.0, 10.0);
    Thread.Sleep(1000);
    robot.GetRobotInstallAngle(ref yangle, ref zangle);
    Console.WriteLine($"yangle  {yangle}   zangle  {zangle}");
}
```

2.2.5.22 等待指定时间

```
/**
* @brief  等待指定时间
* @param  [in]  t_ms  单位ms
* @return  错误码
*/
int WaitMs(int t_ms);
```

2.2.6 机器人安全设置

2.2.6.1 设置碰撞等级

```
/**
* @brief 设置碰撞等级
* @param  [in]  mode  0-等级，1-百分比
* @param  [in]  level 碰撞阈值，等级对应范围[],百分比对应范围[0~1]
* @param  [in]  config 0-不更新配置文件，1-更新配置文件
* @return  错误码
*/
int SetAnticollision(int mode, double[] level, int config);
```

2.2.6.2 设置碰撞后策略

```
/**
* @brief  设置碰撞后策略
* @param  [in] strategy  0-报错停止，1-继续运行
* @return  错误码
*/
int SetCollisionStrategy(int strategy);
```

2.2.6.3 设置正限位

```
/**
* @brief  设置正限位
* @param  [in] limit 六个关节位置，单位deg
* @return  错误码
*/
int SetLimitPositive(double[] limit);
```

2.2.6.4 设置负限位

```
/**
* @brief  设置负限位
* @param  [in] limit 六个关节位置，单位deg
* @return  错误码
*/
int SetLimitNegative(double[] limit);
```

2.2.6.5 错误状态清除

```
/**
* @brief  错误状态清除
* @return  错误码
*/
int ResetAllError();
```

2.2.6.6 关节摩擦力补偿开关

```
/**
* @brief 关节摩擦力补偿开关
* @param [in] state 0-关，1-开
* @return 错误码
*/
int FrictionCompensationOnOff(byte state);
```

2.2.6.7 设置关节摩擦力补偿系数-正装

```
/**
* @brief  设置关节摩擦力补偿系数-正装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
int SetFrictionValue_level(double[] coeff);
```

2.2.6.8 设置关节摩擦力补偿系数-侧装

```
/**
* @brief  设置关节摩擦力补偿系数-侧装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
int SetFrictionValue_wall(double[] coeff);
```

2.2.6.9 设置关节摩擦力补偿系数-倒装

```
/**
* @brief  设置关节摩擦力补偿系数-倒装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
int SetFrictionValue_ceiling(double[] coeff);
```

2.2.6.10 设置关节摩擦力补偿系数-自由安装

```
/**
* @brief  设置关节摩擦力补偿系数-自由安装
* @param  [in]  coeff 六个关节补偿系数，范围[0~1]
* @return  错误码
*/
int SetFrictionValue_freedom(double[] coeff);
```

2.2.6.11 代码示例

```
private void btnRobotSafetySet_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    int mode = 0;
    int config = 1;
    double[] level1 = new double[6] { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 };
    double[] level2 = new double[6] { 0.5, 0.2, 0.3, 0.4, 0.5, 0.12 };

    robot.SetAnticollision(mode, level1, config);
    mode = 1;
    robot.SetAnticollision(mode, level2, config);
    robot.SetCollisionStrategy(2);

    double[] plimit = new double[6] { 170.0, 80.0, 150.0, 80.0, 170.0, 160.0 };
    int rtn = robot.SetLimitPositive(plimit);
    Console.WriteLine($"SetLimitPositive  rtn  {rtn}");
    double[] nlimit = new double[6] { -170.0, -260.0, -150.0, -260.0, -170.0, -160.0 }
↪;
    rtn = robot.SetLimitNegative(nlimit);
    Console.WriteLine($"SetLimitNegative  rtn  {rtn}");

    robot.ResetAllError();

    double[] lcoeff = new double[6] { 0.9, 0.9, 0.9, 0.9, 0.9, 0.9 };
    double[] wcoeff = new double[6] { 0.4, 0.4, 0.4, 0.4, 0.4, 0.4 };
    double[] ccoeff = new double[6] { 0.6, 0.6, 0.6, 0.6, 0.6, 0.6 };
    double[] fcoeff = new double[6] { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 };
    robot.FrictionCompensationOnOff(1);
    rtn = robot.SetFrictionValue_level(lcoeff);
    Console.WriteLine($"SetFrictionValue_level  rtn  {rtn}");
```

(续下页)

(接上页)

```
    rtn = robot.SetFrictionValue_wall(wcoeff);
    Console.WriteLine($"SetFrictionValue_wall  rtn  {rtn}");
    rtn = robot.SetFrictionValue_ceiling(ccoeff);
    Console.WriteLine($"SetFrictionValue_ceiling  rtn  {rtn}");
    rtn = robot.SetFrictionValue_freedom(fcoeff);
    Console.WriteLine($"SetFrictionValue_freedom  rtn  {rtn}");
}
```

2.2.7 机器人状态查询

2.2.7.1 获取机器人安装角度

```
/**
* @brief  获取机器人安装角度
* @param  [out] yangle 倾斜角
* @param  [out] zangle 旋转角
* @return  错误码
*/
int GetRobotInstallAngle(ref double yangle, ref double zangle);
```

2.2.7.2 获取系统变量值

```
/**
* @brief  获取系统变量值
* @param  [in] id 系统变量编号，范围[1~20]
* @param  [out] value  系统变量值
* @return  错误码
*/
int GetSysVarValue(int id, ref double value);
```

2.2.7.3 获取当前关节位置 (角度)

```
/**
* @brief  获取当前关节位置(角度)
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] jPos 六个关节位置，单位deg
* @return  错误码
*/
int GetActualJointPosDegree(byte flag, ref JointPos jPos);
```

2.2.7.4 获取当前关节位置 (弧度)

```
/**
* @brief  获取当前关节位置(弧度)
* @param  [in] flag 0-阻塞, 1-非阻塞
* @param  [out] jPos 六个关节位置, 单位rad
* @return  错误码
*/
int GetActualJointPosRadian(byte flag, ref JointPos jPos);
```

2.2.7.5 获取关节反馈速度

```
/**
* @brief 获取关节反馈速度-deg/s
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] speed 六个关节速度
* @return 错误码
*/
int GetActualJointSpeedsDegree(byte flag, ref double[] speed);
```

2.2.7.6 获取关节反馈加速度

```
/**
* @brief 获取关节反馈加速度-deg/s^2
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] acc 六个关节加速度
* @return 错误码
*/
int GetActualJointAccDegree(byte flag, ref double[] acc);
```

2.2.7.7 获取 TCP 指令速度-合速度

```
/**
* @brief 获取TCP指令速度-合速度
* @param [in] flag 0-阻塞, 1-非阻塞
* @param [out] tcp_speed 线性速度
* @param [out] ori_speed 姿态速度
* @return 错误码
*/
int GetTargetTCPCompositeSpeed(byte flag, ref double tcp_speed, ref double ori_speed);
```

2.2.7.8 获取 TCP 反馈速度-合速度

```
/**
* @brief 获取TCP反馈速度-合速度
* @param [in] flag 0-阻塞，1-非阻塞
* @param [out] tcp_speed 线性速度
* @param [out] ori_speed 姿态速度
* @return 错误码
*/
int GetActualTCPCompositeSpeed(byte flag, ref double tcp_speed, ref double ori_speed);
```

2.2.7.9 获取 TCP 指令速度-分速度

```
/**
* @brief 获取TCP指令速度-分速度
* @param [in] flag 0-阻塞，1-非阻塞
* @param [out] speed [x,y,z,rx,ry,rz]速度
* @return 错误码
*/
int GetTargetTCPSpeed(byte flag, ref double[] speed);
```

2.2.7.10 获取 TCP 反馈速度-分速度

```
/**
* @brief 获取TCP反馈速度-分速度
* @param [in] flag 0-阻塞，1-非阻塞
* @param [out] speed [x,y,z,rx,ry,rz]速度
* @return 错误码
*/
int GetActualTCPSpeed(byte flag, ref double[] speed);
```

2.2.7.11 获取当前工具位姿

```
/**
* @brief  获取当前工具位姿
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] desc_pos  工具位姿
* @return  错误码
*/
int GetActualTCPPose(byte flag, ref DescPose desc_pos);
```

2.2.7.12 获取当前工具坐标系编号

```
/**
* @brief  获取当前工具坐标系编号
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] id  工具坐标系编号
* @return  错误码
*/
int GetActualTCPNum(byte flag, ref int id);
```

2.2.7.13 获取当前工件坐标系编号

```
/**
* @brief  获取当前工件坐标系编号
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] id  工件坐标系编号
* @return  错误码
*/
int GetActualWObjNum(byte flag, ref int id);
```

2.2.7.14 获取当前末端法兰位姿

```
/**
* @brief  获取当前末端法兰位姿
* @param  [in] flag  0-阻塞，1-非阻塞
* @param  [out] desc_pos  法兰位姿
* @return  错误码
*/
int GetActualToolFlangePose(byte flag, ref DescPose desc_pos);
```

2.2.7.15 逆运动学求解

```
/**
* @brief  逆运动学求解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] config 关节空间配置，[-1]-参考当前关节位置解算，[0~7]-
↪依据特定关节空间配置求解
* @param  [out] joint_pos 关节位置
* @return  错误码
```

(续下页)

(接上页)

```
*/
int GetInverseKin(int type, DescPose desc_pos, int config, ref JointPos joint_pos);
```

2.2.7.16 逆运动学求解

```
/**
* @brief  逆运动学求解，参考指定关节位置求解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] joint_pos_ref 参考关节位置
* @param  [out] joint_pos 关节位置
* @return  错误码
*/
int GetInverseKin(int type, DescPose desc_pos, int config, ref JointPos joint_pos);
```

2.2.7.17 逆运动学求解（参考指定关节位置）

```
/**
* @brief  逆运动学求解，参考指定关节位置判断是否有解
* @param  [in] type 0-绝对位姿(基坐标系)，1-增量位姿(基坐标系)，2-增量位姿(工具坐标系)
* @param  [in] desc_pos 笛卡尔位姿
* @param  [in] joint_pos_ref 参考关节位置
* @param  [out] result 0-无解，1-有解
* @return  错误码
*/
int GetInverseKinRef(int posMode, DescPose desc_pos, JointPos joint_pos_ref, ref␣
↪JointPos joint_pos);
```

2.2.7.18 判断逆运动学是否有解

```
/**
* @brief 逆运动学求解，判断指定参考关节位置是否有解
* @param [in] posMode 0绝对位姿，1相对位姿-基坐标系，2相对位姿-工具坐标系
* @param [in] desc_pos 笛卡尔位姿
* @param [in] joint_pos_ref 参考关节位置
* @param [out] hasResult 0-无解，1-有解
* @return 错误码
*/
int GetInverseKinHasSolution(int posMode, DescPose desc_pos, JointPos joint_pos_ref,␣
↪ref bool hasResult);
```

2.2.7.19 正运动学求解

```
/**
* @brief  正运动学求解
* @param  [in] joint_pos 关节位置
* @param  [out] desc_pos 笛卡尔位姿
* @return  错误码
*/
int GetForwardKin(JointPos joint_pos, ref DescPose desc_pos);
```

2.2.7.20 获取当前关节转矩

```
/**
* @brief 获取当前关节转矩
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] torques 关节转矩
* @return  错误码
*/
int GetJointTorques(byte flag, float[] torques);
```

2.2.7.21 获取当前负载的重量

```
/**
* @brief  获取当前负载的重量
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] weight 负载重量，单位kg
* @return  错误码
*/
int GetTargetPayload(byte flag, ref double weight);
```

2.2.7.22 获取当前负载的质心

```
/**
* @brief  获取当前负载的质心
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] cog 负载质心，单位mm
* @return  错误码
*/
int GetTargetPayloadCog(byte flag, ref DescTran cog);
```

2.2.7.23 获取当前工具坐标系

```
/**
* @brief  获取当前工具坐标系
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] desc_pos 工具坐标系位姿
* @return  错误码
*/
int GetTCPOffset(byte flag, ref DescPose desc_pos);
```

2.2.7.24 获取当前工件坐标系

```
/**
* @brief  获取当前工件坐标系
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] desc_pos 工件坐标系位姿
* @return  错误码
*/
int GetWObjOffset(byte flag, ref DescPose desc_pos);
```

2.2.7.25 获取关节软限位角度

```
/**
* @brief  获取关节软限位角度
* @param  [in] flag 0-阻塞，1-非阻塞
* @param  [out] negative  负限位角度，单位deg
* @param  [out] positive  正限位角度，单位deg
* @return  错误码
*/
int GetJointSoftLimitDeg(byte flag, ref double[] negative, ref double[] positive);
```

2.2.7.26 获取系统时间

```
/**
* @brief  获取系统时间
* @param  [out] t_ms 单位ms
* @return  错误码
*/
int GetSystemClock(ref double t_ms);
```

2.2.7.27 获取机器人当前关节配置

```
/**
* @brief  获取机器人当前关节位置
* @param  [out]  config  关节空间配置，范围[0~7]
* @return  错误码
*/
int GetRobotCurJointsConfig(ref int config);
```

2.2.7.28 获取当前速度

```
/**
* @brief  获取机器人当前速度
* @param  [out]  vel  速度，单位mm/s
* @return  错误码
*/
int GetDefaultTransVel(ref double vel);
```

2.2.7.29 查询机器人运动是否完成

```
/**
* @brief  查询机器人运动是否完成
* @param  [out]  state  0-未完成，1-完成
* @return  错误码
*/
int GetRobotMotionDone(ref byte state);
```

2.2.7.30 代码示例

```
private void btnRobotState_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    double yangle = 0, zangle = 0;
    byte flag = 0;
    JointPos j_deg = new JointPos(0, 0, 0, 0, 0, 0);
    JointPos j_rad = new JointPos(0, 0, 0, 0, 0, 0);
    DescPose tcp, flange, tcp_offset, wobj_offset;
    DescTran cog;
```

(续下页)

(接上页)

```
    tcp = new DescPose();
    flange = new DescPose();
    tcp_offset = new DescPose();
    wobj_offset = new DescPose();
    cog = new DescTran();

    int id = 0;
    float[] torques = new float[6] { 0, 0, 0, 0, 0, 0};
    double weight = 0.0;
    double[] neg_deg = new double[6] { 0, 0, 0, 0, 0, 0 };
    double[] pos_deg = new double[6] { 0, 0, 0, 0, 0, 0 };
    double t_ms = 0;
    int config = 0;
    double vel = 0;

    robot.GetRobotInstallAngle(ref yangle, ref zangle);
    Console.WriteLine($"yangle:{yangle},zangle:{zangle}");

    robot.GetActualJointPosDegree(flag, ref j_deg);
    Console.WriteLine($"joint pos deg:{j_deg.jPos[0]}, {j_deg.jPos[1]}, {j_deg.
↪jPos[2]}, {j_deg.jPos[3]},{j_deg.jPos[4]},{j_deg.jPos[5]}");
    robot.GetActualJointPosRadian(flag, ref j_rad);
    Console.WriteLine($"joint pos rad:{j_rad.jPos[0]}, {j_rad.jPos[1]}, {j_rad.
↪jPos[2]},{j_rad.jPos[3]},{j_rad.jPos[4]},{j_rad.jPos[5]}");

    robot.GetActualTCPPose(flag, ref tcp);
    Console.WriteLine($"tcp pose:{tcp.tran.x}, {tcp.tran.y}, {tcp.tran.z}, {tcp.rpy.
↪rx}, {tcp.rpy.ry},{tcp.rpy.rz}");

    robot.GetActualToolFlangePose(flag, ref flange);
    Console.WriteLine($"flange pose:{flange.tran.x}, {flange.tran.y}, {flange.tran.z},
↪ {flange.rpy.rx},{flange.rpy.ry},{flange.rpy.rz}");

    robot.GetActualTCPNum(flag, ref id);
    Console.WriteLine($"tcp num : {id}");

    robot.GetActualWObjNum(flag, ref id);
    Console.WriteLine($"wobj num : {id}");

    robot.GetJointTorques(flag, torques);
    Console.WriteLine($"torques:{torques[0]},{torques[1]},{torques[2]},{torques[3]},
↪{torques[4]},{torques[5]}");

```

(续下页)

(接上页)

```
    robot.GetTargetPayload(flag, ref weight);
    Console.WriteLine($"payload weight : {weight}");

    robot.GetTargetPayloadCog(flag, ref cog);
    Console.WriteLine($"payload cog:{cog.x},{cog.y},{cog.z}");

    robot.GetTCPOffset(flag, ref tcp_offset);
    Console.WriteLine($"tcp offset:{tcp_offset.tran.x}, {tcp_offset.tran.y}, {tcp_
↪offset.tran.z},{tcp_offset.rpy.rx},{tcp_offset.rpy.ry},{tcp_offset.rpy.rz}");

    robot.GetWObjOffset(flag, ref wobj_offset);
    Console.WriteLine($"wobj offset:{wobj_offset.tran.x}, {wobj_offset.tran.y},{wobj_
↪offset.tran.z},{wobj_offset.rpy.rx},{wobj_offset.rpy.ry},{wobj_offset.rpy.rz}");

    robot.GetJointSoftLimitDeg(flag, ref neg_deg, ref pos_deg);
    Console.WriteLine($"neg limit deg:{neg_deg[0]}, {neg_deg[1]}, {neg_deg[2]}, {neg_
↪deg[3]},{neg_deg[4]},{neg_deg[5]}");
    Console.WriteLine($"pos limit deg:{pos_deg[0]}, {pos_deg[1]}, {pos_deg[2]}, {pos_
↪deg[3]},{pos_deg[4]},{pos_deg[5]}");

    robot.GetSystemClock(ref t_ms);
    Console.WriteLine($"system clock : {t_ms}");

    robot.GetRobotCurJointsConfig(ref config);
    Console.WriteLine($"joint config : {config}");

    robot.GetDefaultTransVel(ref vel);
    Console.WriteLine($"trans vel : {vel}");
}
```

2.2.7.31 查询机器人错误码

```
/**
* @brief 查询机器人错误码
* @param [out] maincode   主错误码
* @param [out] subcode    子错误码
* @return 错误码
*/
int GetRobotErrorCode(ref int maincode, ref int subcode);
```

2.2.7.32 查询机器人示教管理点数据

```
/**
* @brief 查询机器人示教管理点位数据
* @param [in] name    点位名
* @param [out] data   点位数据double[20]{x,y,z,rx,ry,rz,j1,j2,j3,j4,j5,j6,tool, wobj,speed,acc,e1,e2,e3,e4}
* @return 错误码
*/
int GetRobotTeachingPoint(string name, ref double[] data);
```

2.2.7.33 查询机器人运动队列缓存长度

```
/**
* @brief 查询机器人运动队列缓存长度
* @param [out] len   缓存长度
* @return 错误码
*/
int GetMotionQueueLength(ref int len);
```

2.2.7.34 代码示例

```
private void btnRobotState2_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    byte robotMotionState = 255;
    robot.GetRobotMotionDone(ref robotMotionState);
    Console.WriteLine($"robotMotionState  {robotMotionState}");

    int mainErrCode = -1;
    int subErrCode = -1;
    robot.GetRobotErrorCode(ref mainErrCode, ref subErrCode);
    Console.WriteLine($"mainErrCode  {mainErrCode}  subErrCode  {subErrCode} ");

    string name = "a1";
    double[] point = new double[6] {0, 0, 0, 0, 0, 0};
    robot.GetRobotTeachingPoint(name, ref point);
    Console.WriteLine($"GetRobotTeachingPoint:{point[0]},{point[1]},{point[2]},{point[3]},{point[4]},{point[5]}");

```

(续下页)

(接上页)

```
    int length = -1;
    robot.GetMotionQueueLength(ref length);
    Console.WriteLine($"GetMotionQueueLength  {length}");
}
```

2.2.8 机器人轨迹复现

2.2.8.1 设置轨迹记录参数

```
/**
* @brief  设置轨迹记录参数
* @param  [in] type  记录数据类型，1-关节位置
* @param  [in] name  轨迹文件名
* @param  [in] period_ms  数据采样周期，固定值2ms或4ms或8ms
* @param  [in] di_choose  DI选择,bit0~bit7对应控制箱DI0~DI7，bit8~bit9对应末端DI0~DI1，0-不选择，1-选择
* @param  [in] do_choose  DO选择,bit0~bit7对应控制箱DO0~DO7，bit8~bit9对应末端DO0~DO1，0-不选择，1-选择
* @return  错误码
*/
int SetTPDParam(int type, string name, int period_ms, UInt16 di_choose, UInt16 do_choose);
```

2.2.8.2 开始轨迹记录

```
/**
* @brief  开始轨迹记录
* @param  [in] type  记录数据类型，1-关节位置
* @param  [in] name  轨迹文件名
* @param  [in] period_ms  数据采样周期，固定值2ms或4ms或8ms
* @param  [in] di_choose  DI选择,bit0~bit7对应控制箱DI0~DI7，bit8~bit9对应末端DI0~DI1，0-不选择，1-选择
* @param  [in] do_choose  DO选择,bit0~bit7对应控制箱DO0~DO7，bit8~bit9对应末端DO0~DO1，0-不选择，1-选择
* @return  错误码
*/
int SetTPDStart(int type, string name, int period_ms, UInt16 di_choose, UInt16 do_choose);
```

2.2.8.3 停止轨迹记录

```
/**
* @brief  停止轨迹记录
* @return  错误码
*/
int SetWebTPDStop();
```

2.2.8.4 删除轨迹记录

```
/**
* @brief  删除轨迹记录
* @param  [in] name  轨迹文件名
* @return  错误码
*/
int SetTPDDelete(string name);
```

2.2.8.5 代码示例

```
private void btnTCPRecord_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    int type = 1;
    string name = "tpd2023";
    int period_ms = 2;
    UInt16 di_choose = 0;
    UInt16 do_choose = 0;

    robot.SetTPDParam(type, name, period_ms, di_choose, do_choose);

    robot.Mode(1);
    Thread.Sleep(1000);
    robot.DragTeachSwitch(1);
    robot.SetTPDStart(type, name, period_ms, di_choose, do_choose);
    Thread.Sleep(10000);
    robot.SetWebTPDStop();
    robot.DragTeachSwitch(0);

    //robot.SetTPDDelete(name);
}
```

2.2.8.6 轨迹预加载

```
/**
* @brief  轨迹预加载
* @param  [in] name  轨迹文件名
* @return  错误码
*/
int LoadTPD(string name);
```

2.2.8.7 获取轨迹起始位姿

```
/**
* @brief 获取轨迹起始位姿
* @param [in] name  轨迹文件名
* @param [out] desc_pose 轨迹起始位姿
* @return 错误码
*/
int GetTPDStartPose(string name, ref DescPose desc_pose);
```

2.2.8.8 轨迹复现

```
/**
* @brief  轨迹复现
* @param  [in] name  轨迹文件名
* @param  [in] blend 0-不平滑, 1-平滑
* @param  [in] ovl  速度缩放百分比, 范围[0~100]
* @return  错误码
*/
int MoveTPD(string name, byte blend, float ovl);
```

2.2.8.9 代码示例

```
private void btnTPDMove_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    string name = "tpd2023";
    int tool = 0;
    int user = 0;
```

(续下页)

(接上页)

```
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = -1.0f;
    int config = -1;
    byte blend = 1;

    DescPose desc_pose = new DescPose();
    robot.GetTPDStartPose(name, ref desc_pose);
    Console.WriteLine($"GetTPDStartPose:{desc_pose.tran.x},{desc_pose.tran.y},{desc_
↪pose.tran.z},{desc_pose.rpy.rx},{desc_pose.rpy.ry},{desc_pose.rpy.rz}");
    robot.SetTrajectoryJSpeed(100.0f);

    robot.LoadTPD(name);
    robot.MoveCart(desc_pose, tool, user, vel, acc, ovl, blendT, config);
    robot.MoveTPD(name, blend, 100.0f);
}
```

2.2.8.10 外部轨迹文件预处理

```
/**
* @brief 外部轨迹文件预处理
* @param [in] name 轨迹文件名
* @param [in] ovl 速度缩放百分比，范围[0~100]
* @param [in] opt 1-控制点，默认为1
* @return 错误码
*/
int LoadTrajectoryJ(string name, float ovl, int opt);
```

2.2.8.11 外部轨迹文件轨迹复现

```
/**
* @brief 外部轨迹文件轨迹复现
* @return 错误码
*/
int MoveTrajectoryJ();
```

2.2.8.12 获取轨迹文件轨迹起始位置

```
/**
* @brief 获取轨迹文件轨迹起始位置
* @param [in] name 轨迹文件名
* @param [out] desc_pose 轨迹起始位姿
* @return 错误码
*/
int GetTrajectoryStartPose(string name, ref DescPose desc_pose);
```

2.2.8.13 获取轨迹文件轨迹点编号

```
/**
* @brief 获取轨迹点编号
* @param [out] pnum 轨迹点编号
* @return 错误码
*/
int GetTrajectoryPointNum(ref int pnum);
```

2.2.8.14 设置轨迹文件轨迹运行速度

```
/**
* @brief 设置轨迹文件轨迹运行速度
* @param [in] ovl 速度百分比
* @return 错误码
*/
int SetTrajectoryJSpeed(double ovl);
```

2.2.8.15 设置轨迹文件轨迹运行中的力和力矩

```
/**
* @brief 设置轨迹文件轨迹运行中的力和力矩
* @param [in] ft 三个方向的力和扭矩，单位N和Nm
* @return 错误码
*/
int SetTrajectoryJForceTorque(ForceTorque ft);
```

2.2.8.16 设置轨迹运行中的沿 x 方向的力

```
/**
* @brief 设置轨迹运行中的沿x方向的力
* @param [in] fx  沿x方向的力，单位N
* @return 错误码
*/
int SetTrajectoryJForceFx(double fx);
```

2.2.8.17 设置轨迹运行中的沿 y 方向的力

```
/**
* @brief 设置轨迹运行中的沿y方向的力
* @param [in] fy  沿y方向的力，单位N
* @return 错误码
*/
int SetTrajectoryJForceFy(double fy);
```

2.2.8.18 设置轨迹运行中的沿 z 方向的力

```
/**
* @brief 设置轨迹运行中的沿z方向的力
* @param [in] fz  沿z方向的力，单位N
* @return 错误码
*/
int SetTrajectoryJForceFz(double fz);
```

2.2.8.19 设置轨迹运行中的绕 x 轴的扭矩

```
/**
* @brief 设置轨迹运行中的绕x轴的扭矩
* @param [in] tx  绕x轴的扭矩，单位Nm
* @return 错误码
*/
int SetTrajectoryJTorqueTx(double tx);
```

2.2.8.20 设置轨迹运行中的绕 y 轴的扭矩

```
/**
* @brief 设置轨迹运行中的绕y轴的扭矩
* @param [in] ty  绕y轴的扭矩，单位Nm
* @return 错误码
*/
int SetTrajectoryJTorqueTy(double ty);
```

2.2.8.21 设置轨迹运行中的绕 z 轴的扭矩

```
/**
* @brief 设置轨迹运行中的绕z轴的扭矩
* @param [in] tz  绕z轴的扭矩，单位Nm
* @return 错误码
*/
int SetTrajectoryJTorqueTz(double tz);
```

2.2.8.22 代码示例

```
private void btnTrajectory_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");
    string name = "/fruser/traj/trajHelix_aima_1.txt";
    int rtn = -1;

    rtn = robot.LoadTrajectoryJ(name, 100, 1);
    Console.WriteLine($"LoadTrajectoryJ:{rtn}");

    DescPose desc_pos2 = new DescPose(0, 0, 0, 0, 0, 0);
    rtn = robot.GetTrajectoryStartPose(name, ref desc_pos2);
    Console.WriteLine($"GetTrajectoryStartPose:{desc_pos2.tran.x},{desc_pos2.tran.y},
↪{desc_pos2.tran.z},{desc_pos2.rpy.rx},{desc_pos2.rpy.ry},{desc_pos2.rpy.rz}");

    int tool = 1;
    int user = 0;
    float vel = 100.0f;
    float acc = 100.0f;
    float ovl = 100.0f;
    float blendT = -1.0f;
    float blendT1 = 0.0f;
```

(续下页)

(接上页)

```
    int config = -1;
    robot.MoveCart(desc_pos2, tool, user, vel, acc, ovl, blendT, config);

    rtn = robot.SetTrajectoryJSpeed(20);
    Console.WriteLine($"SetTrajectoryJSpeed: rtn  {rtn}");

    rtn = robot.MoveTrajectoryJ();
    Console.WriteLine($"MoveTrajectoryJ:{rtn}");

    int pnum = -1;
    rtn = robot.GetTrajectoryPointNum(ref pnum);
    Console.WriteLine($"GetTrajectoryPointNum: rtn  {rtn}    num {pnum}");

    rtn = robot.SetTrajectoryJSpeed(100);
    Console.WriteLine($"SetTrajectoryJSpeed: rtn  {rtn}");

    ForceTorque ft = new ForceTorque(1, 1, 1, 1, 1, 1);
    rtn = robot.SetTrajectoryJForceTorque(ft);
    Console.WriteLine($"SetTrajectoryJForceTorque: rtn  {rtn}");

    rtn = robot.SetTrajectoryJForceFx(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
    rtn = robot.SetTrajectoryJForceFy(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
    rtn = robot.SetTrajectoryJForceFz(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
    rtn = robot.SetTrajectoryJTorqueTx(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
    rtn = robot.SetTrajectoryJTorqueTy(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
    rtn = robot.SetTrajectoryJTorqueTz(1.0);
    Console.WriteLine($"SetTrajectoryJForceFx: rtn  {rtn}");
}
```

2.2.9 机器人 WebAPP 程序使用

2.2.9.1 设置开机自动加载默认的作业程序

```
/**
* @brief  设置开机自动加载默认的作业程序
* @param  [in] flag  0-开机不自动加载默认程序，1-开机自动加载默认程序
* @param  [in] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
```

(续下页)

(接上页)

```
    ↪"为固定路径
 * @return  错误码
 */
int LoadDefaultProgConfig(byte flag, string program_name);
```

2.2.9.2 加载指定的作业程序

```
/**
 * @brief  加载指定的作业程序
 * @param  [in] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
    ↪"为固定路径
 * @return  错误码
 */
int ProgramLoad(string program_name);
```

2.2.9.3 获取已加载的作业程序名

```
/**
 * @brief  获取已加载的作业程序名
 * @param  [out] program_name 作业程序名及路径，如"/fruser/movej.lua"，其中"/fruser/
    ↪"为固定路径
 * @return  错误码
 */
int GetLoadedProgram(ref string program_name);
```

2.2.9.4 获取当前机器人作业程序的执行行号

```
/**
 * @brief  获取当前机器人作业程序执行的行号
 * @param  [out] line  行号
 * @return  错误码
 */
int GetCurrentLine(ref int line);
```

2.2.9.5 运行当前加载的作业程序

```
/**
* @brief  运行当前加载的作业程序
* @return  错误码
*/
int ProgramRun();
```

2.2.9.6 暂停当前运行的作业程序

```
/**
* @brief  暂停当前运行的作业程序
* @return  错误码
*/
int ProgramPause();
```

2.2.9.7 恢复当前暂停的作业程序

```
/**
* @brief  恢复当前暂停的作业程序
* @return  错误码
*/
int ProgramResume();
```

2.2.9.8 终止当前运行的作业程序

```
/**
* @brief  终止当前运行的作业程序
* @return  错误码
*/
int ProgramStop();
```

2.2.9.9 获取机器人作业程序执行状态

```
/**
* @brief  获取机器人作业程序执行状态
* @param  [out]  state 1-程序停止或无程序运行，2-程序运行中，3-程序暂停
* @return  错误码
```

(续下页)

(接上页)

```
*/
int GetProgramState(ref byte state);
```

2.2.9.10 代码示例

```
private void btnWebApp_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    string program_name = "/fruser/testWebApp.lua";
    string loaded_name = "";
    byte state = 0;
    int line = 0;

    robot.Mode(0);
    robot.ProgramLoad(program_name);
    robot.ProgramRun();
    Thread.Sleep(2000);
    robot.ProgramPause();
    robot.GetProgramState(ref state);
    Console.WriteLine($"program state : {state}");
    robot.GetCurrentLine(ref line);
    Console.WriteLine($"current line : {line}");
    robot.GetLoadedProgram(ref loaded_name);
    Console.WriteLine($"program name : {loaded_name}");
    Thread.Sleep(1000);
    robot.ProgramResume();
    Thread.Sleep(1000);
    robot.ProgramStop();
}
```

2.2.10 机器人外设

2.2.10.1 配置夹爪

```
/**
* @brief  配置夹爪
* @param  [in] company  夹爪厂商，待定
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
```

(续下页)

(接上页)

```
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
errno_t  SetGripperConfig(int company, int device, int softvesion, int bus);
```

2.2.10.2 获取夹爪配置

```
/**
* @brief  获取夹爪配置
* @param  [in] company  夹爪厂商，待定
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
errno_t  GetGripperConfig(int *company, int *device, int *softvesion, int *bus);
```

2.2.10.3 激活夹爪

```
/**
* @brief  激活夹爪
* @param  [in] index  夹爪编号
* @param  [in] act  0-复位，1-激活
* @return  错误码
*/
int ActGripper(int index, byte act);
```

2.2.10.4 控制夹爪

```
/**
* @brief  控制夹爪
* @param  [in] index  夹爪编号
* @param  [in] pos  位置百分比，范围[0~100]
* @param  [in] vel  速度百分比，范围[0~100]
* @param  [in] force  力矩百分比，范围[0~100]
* @param  [in] max_time  最大等待时间，范围[0~30000]，单位ms
* @param  [in] block  0-阻塞，1-非阻塞
* @return  错误码
*/
int MoveGripper(int index, int pos, int vel, int force, int max_time, byte block);
```

2.2.10.5 获取夹爪运动状态

```
/**
* @brief  获取夹爪运动状态
* @param  [out] fault  0-无错误，1-有错误
* @param  [out] staus  0-运动未完成，1-运动完成
* @return  错误码
*/
int GetGripperMotionDone(ref int fault, ref int status);
```

2.2.10.6 代码示例

```
private void btnOperateGripper_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    int company = 4;
    int device = 0;
    int softversion = 0;
    int bus = 1;
    int index = 1;
    byte act = 0;
    int max_time = 30000;
    byte block = 0;
    int status = 0, fault = 0;
    int rtn = -1;

    robot.SetGripperConfig(company, device, softversion, bus);
    Thread.Sleep(1000);
    robot.GetGripperConfig(ref company, ref device, ref softversion, ref bus);
    Console.WriteLine($"gripper config : {company}, {device}, {softversion}, {bus}");

    rtn = robot.ActGripper(index, act);
    Console.WriteLine($"ActGripper  {rtn}");
    Thread.Sleep(1000);
    act = 1;
    rtn = robot.ActGripper(index, act);
    Console.WriteLine($"ActGripper  {rtn}");
    Thread.Sleep(4000);

    rtn = robot.MoveGripper(index, 20, 50, 50, max_time, block);
    Console.WriteLine($"MoveGripper  {rtn}");
```

(续下页)

(接上页)

```
    Thread.Sleep(2000);
    robot.MoveGripper(index, 10, 50, 0, max_time, block);

    Thread.Sleep(4000);
    robot.GetGripperMotionDone(ref fault, ref status);
    Console.WriteLine($"motion status : {fault}, {status}");
}
```

2.2.10.7 计算预抓取点-视觉

```
/**
* @brief 计算预抓取点-视觉
* @param [in] desc_pos 抓取点笛卡尔位姿
* @param [in] zlength z轴偏移量
* @param [in] zangle 绕z轴旋转偏移量
* @param [out] pre_pos 预抓取点
* @return 错误码
*/
int ComputePrePick(DescPose desc_pos, double zlength, double zangle, ref DescPose pre_
↪pos);
```

2.2.10.8 计算撤退点-视觉

```
/**
* @brief 计算撤退点-视觉
* @param [in] desc_pos 撤退点笛卡尔位姿
* @param [in] zlength z轴偏移量
* @param [in] zangle 绕z轴旋转偏移量
* @param [out] post_pos 撤退点
* @return 错误码
*/
int ComputePostPick(DescPose desc_pos, double zlength, double zangle, ref DescPose␣
↪post_pos);
```

2.2.11 机器人力控

2.2.11.1 力传感器配置

```
/**
* @brief  配置力传感器
* @param  [in] company  力传感器厂商，17-坤维科技
* @param  [in] device  设备号，暂不使用，默认为0
* @param  [in] softvesion  软件版本号，暂不使用，默认为0
* @param  [in] bus 设备挂在末端总线位置，暂不使用，默认为0
* @return  错误码
*/
int FT_SetConfig(int company, int device, int softvesion, int bus);
```

2.2.11.2 获取力传感器配置

```
/**
* @brief 获取力传感器配置
* @param [out] deviceID 力传感器编号
* @param [out] company 力传感器厂商，，力传感器厂商，17-坤维科技，19-航天十一院，20-ATI传感器，21-中科米点，22-伟航敏芯
* @param [out] device  设备号，坤维(0-KWR75B)，航天十一院(0-MCS6A-200-4)，ATI (0-AXIA80 -M8)，中科米点(0-MST2010)，伟航敏芯(0-WHC6L-YB-10A)
* @param [out] softvesion 软件版本号，暂不使用，默认为 0
* @return 错误码
*/
int FT_GetConfig(ref int deviceID, ref int company, ref int device, ref int softvesion);
```

2.2.11.3 力传感器激活

```
/**
* @brief  力传感器激活
* @param  [in] act  0-复位，1-激活
* @return  错误码
*/
int FT_Activate(byte act);
```

2.2.11.4 力传感器校零

```
/**
* @brief  力传感器校零
* @param  [in] act  0-去除零点，1-零点矫正
* @return  错误码
*/
int FT_SetZero(byte act);
```

2.2.11.5 代码示例

```
private void btnFT_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    int company = 17;
    int device = 0;
    int softversion = 0;
    int bus = 1;
    int index = 1;
    byte act = 0;

    robot.FT_SetConfig(company, device, softversion, bus);
    Thread.Sleep(1000);
    company = 0;
    robot.FT_GetConfig(ref company, ref device, ref softversion, ref bus);
    Console.WriteLine($"FT config : {company}, {device}, {softversion}, {bus}");
    Thread.Sleep(1000);

    robot.FT_Activate(act);
    Thread.Sleep(1000);
    act = 1;
    robot.FT_Activate(act);
    Thread.Sleep(1000);

    robot.SetLoadWeight(0.0f);
    Thread.Sleep(1000);
    DescTran coord = new DescTran(0, 0, 0);

    robot.SetLoadCoord(coord);
    Thread.Sleep(1000);
    robot.FT_SetZero(0);//0去除零点  1零点矫正
```

(续下页)

(接上页)

```
    Thread.Sleep(1000);

    ForceTorque ft = new ForceTorque(0, 0, 0, 0, 0, 0);
    int rtn = robot.FT_GetForceTorqueOrigin(1, ref ft);
    Console.WriteLine($"ft origin : {ft.fx}, {ft.fy}, { ft.fz}, { ft.tx}, { ft.ty}, {ft.tz}    rtn   {rtn}");
    rtn = robot.FT_SetZero(1);//零点矫正
    //Console.WriteLine($"set zero rtn {rtn}");

    Thread.Sleep(2000);
    rtn = robot.FT_GetForceTorqueOrigin(1, ref ft);
    Console.WriteLine($"ft rcs : {ft.fx}, {ft.fy}, {ft.fz}, {ft.tx}, {ft.ty}, {ft.tz} rtn  {rtn}");

    robot.FT_GetForceTorqueRCS(1, ref ft);
    Console.WriteLine($"FT_GetForceTorqueRCS rcs : {ft.fx}, {ft.fy}, {ft.fz}, {ft.tx}, {ft.ty}, {ft.tz}");
}
```

2.2.11.6 设置力传感器参考坐标系

```
/**
* @brief  设置力传感器参考坐标系
* @param  [in] ref  0-工具坐标系，1-基坐标系
* @return  错误码
*/
int FT_SetRCS(byte type);
```

2.2.11.7 负载重量辨识记录

```
/**
* @brief  负载重量辨识记录
* @param  [in] id  传感器坐标系编号，范围[1~14]
* @return  错误码
*/
int FT_PdIdenRecord(int id);
```

2.2.11.8 负载重量辨识计算

```
/**
* @brief  负载重量辨识计算
* @param  [out] weight  负载重量，单位kg
* @return  错误码
*/
int FT_PdIdenCompute(ref double weight);
```

2.2.11.9 负载质心辨识记录

```
/**
* @brief  负载质心辨识记录
* @param  [in] id  传感器坐标系编号，范围[1~14]
* @param  [in] index 点编号，范围[1~3]
* @return  错误码
*/
int FT_PdCogIdenRecord(int id, int index);
```

2.2.11.10 负载质心辨识计算

```
/**
* @brief  负载质心辨识计算
* @param  [out] cog  负载质心，单位mm
* @return  错误码
*/
int FT_PdCogIdenCompute(ref DescTran cog);
```

2.2.11.11 代码示例

```
private void btnFTPdCog_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    double weight = 0.1;
    int rtn = -1;
    DescPose tcoord, desc_p1, desc_p2, desc_p3;
    tcoord = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p1 = new DescPose(0, 0, 0, 0, 0, 0);
```

(续下页)

(接上页)

```
    desc_p2 = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p3 = new DescPose(0, 0, 0, 0, 0, 0);

    robot.FT_SetRCS(0);
    Thread.Sleep(1000);

    tcoord.tran.z = 35.0;
    robot.SetToolCoord(10, tcoord, 1, 0);
    Thread.Sleep(1000);
    robot.FT_PdIdenRecord(10);
    Thread.Sleep(1000);
    robot.FT_PdIdenCompute(ref weight);
    Console.WriteLine($"payload weight : {weight}");

    desc_p1.tran.x = -47.805;
    desc_p1.tran.y = -362.266;
    desc_p1.tran.z = 317.754;
    desc_p1.rpy.rx = -179.496;
    desc_p1.rpy.ry = -0.255;
    desc_p1.rpy.rz = 34.948;

    desc_p2.tran.x = -77.805;
    desc_p2.tran.y = -312.266;
    desc_p2.tran.z = 317.754;
    desc_p2.rpy.rx = -179.496;
    desc_p2.rpy.ry = -0.255;
    desc_p2.rpy.rz = 34.948;

    desc_p3.tran.x = -167.805;
    desc_p3.tran.y = -312.266;
    desc_p3.tran.z = 387.754;
    desc_p3.rpy.rx = -179.496;
    desc_p3.rpy.ry = -0.255;
    desc_p3.rpy.rz = 34.948;

    rtn = robot.MoveCart(desc_p1, 0, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
    Console.WriteLine($"MoveCart rtn  {rtn}");
    Thread.Sleep(1000);
    robot.FT_PdCogIdenRecord(10, 1);
    robot.MoveCart(desc_p2, 0, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
    Thread.Sleep(1000);
    robot.FT_PdCogIdenRecord(10, 2);
    robot.MoveCart(desc_p3, 0, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
```

(续下页)

(接上页)

```
    Thread.Sleep(1000);
    robot.FT_PdCogIdenRecord(10, 3);
    Thread.Sleep(1000);
    DescTran cog = new DescTran(0, 0, 0);

    robot.FT_PdCogIdenCompute(ref cog);
    Console.WriteLine($"cog : {cog.x}, {cog.y}, {cog.z}");
}
```

2.2.11.12 获取参考坐标系下力/扭矩数据

```
/**
* @brief  获取参考坐标系下力/扭矩数据
* @param  [out] ft  力/扭矩，fx,fy,fz,tx,ty,tz
* @return  错误码
*/
int FT_GetForceTorqueRCS(byte flag, ref ForceTorque ft);
```

2.2.11.13 获取力传感器原始力/扭矩数据

```
/**
* @brief  获取力传感器原始力/扭矩数据
* @param  [out] ft  力/扭矩，fx,fy,fz,tx,ty,tz
* @return  错误码
*/
int FT_GetForceTorqueOrigin(byte flag, ref ForceTorque ft);
```

2.2.11.14 碰撞守护

```
/**
* @brief  碰撞守护
* @param  [in] flag 0-关闭碰撞守护，1-开启碰撞守护
* @param  [in] sensor_id 力传感器编号
* @param  [in] select  选择六个自由度是否检测碰撞，0-不检测，1-检测
* @param  [in] ft  碰撞力/扭矩，fx,fy,fz,tx,ty,tz
* @param  [in] max_threshold 最大阈值
* @param  [in] min_threshold 最小阈值
* @note   力/扭矩检测范围：(ft-min_threshold, ft+max_threshold)
* @return  错误码
*/
```

(续下页)

(接上页)

```
int FT_Guard(int flag, int sensor_id, int[] select, ForceTorque ft, double[] max_threshold, double[] min_threshold);
```

2.2.11.15 代码示例

```
private void btnFTGuard_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    byte flag = 1;
    byte sensor_id = 1;
    int[] select = new int[6]{ 1, 0, 0, 0, 0, 0 };//只启用x轴碰撞守护
    double[] max_threshold = new double[6]{ 5.0f, 0.01f, 0.01f, 0.01f, 0.01f, 0.01f };
    double[] min_threshold = new double[6]{ 3.0f, 0.01f, 0.01f, 0.01f, 0.01f, 0.01f };

    ForceTorque ft = new ForceTorque(0, 0, 0, 0, 0, 0);
    DescPose desc_p1, desc_p2, desc_p3;
    desc_p1 = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p2 = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p3 = new DescPose(0, 0, 0, 0, 0, 0);

    desc_p1.tran.x = 1.299;
    desc_p1.tran.y = -719.159;
    desc_p1.tran.z = 141.314;
    desc_p1.rpy.rx = 177.999;
    desc_p1.rpy.ry = -0.715;
    desc_p1.rpy.rz = -161.937;

    desc_p2.tran.x = 245.047;
    desc_p2.tran.y = -675.509;
    desc_p2.tran.z = 139.538;
    desc_p2.rpy.rx = 177.987;
    desc_p2.rpy.ry = -0.129;
    desc_p2.rpy.rz = -142.238;

    desc_p3.tran.x = 157.233;
    desc_p3.tran.y = -550.088;
    desc_p3.tran.z = 112.485;
    desc_p3.rpy.rx = -176.579;
    desc_p3.rpy.ry = -2.819;
    desc_p3.rpy.rz = -148.415;
```

(续下页)

(接上页)

```
    robot.SetSpeed(5);

    int rtn =  robot.FT_Guard(flag, sensor_id, select, ft, max_threshold, min_
↪threshold);
    Console.WriteLine($"FT_Guard start rtn {rtn}");
    robot.MoveCart(desc_p1, 1, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
    robot.MoveCart(desc_p2, 1, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
    robot.MoveCart(desc_p3, 1, 0, 100.0f, 100.0f, 100.0f, -1.0f, -1);
    flag = 0;
    rtn = robot.FT_Guard(flag, sensor_id, select, ft, max_threshold, min_threshold);
    Console.WriteLine($"FT_Guard end rtn {rtn}");
}
```

2.2.11.16 恒力控制

```
/**
* @brief  恒力控制
* @param  [in] flag 0-关闭恒力控制，1-开启恒力控制
* @param  [in] sensor_id 力传感器编号
* @param  [in] select  选择六个自由度是否检测碰撞，0-不检测，1-检测
* @param  [in] ft  碰撞力/扭矩，fx,fy,fz,tx,ty,tz
* @param  [in] ft_pid 力pid参数，力矩pid参数
* @param  [in] adj_sign 自适应启停控制，0-关闭，1-开启
* @param  [in] ILC_sign ILC启停控制，  0-停止，1-训练，2-实操
* @param  [in] 最大调整距离，单位mm
* @param  [in] 最大调整角度，单位deg
* @return  错误码
*/
int FT_Control(int flag, int sensor_id, int[] select, ForceTorque ft, double[] ft_pid,
↪ int adj_sign, int ILC_sign, double max_dis, double max_ang);
```

2.2.11.17 代码示例

```
private void btnFTConttol_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    byte flag = 1;
    byte sensor_id = 1;
    int[] select = new int[6]{ 0, 0, 1, 0, 0, 0 };
```

(续下页)

(接上页)

```
    double[] ft_pid = new double[6]{ 0.0005f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
    byte adj_sign = 0;
    byte ILC_sign = 0;
    float max_dis = 100.0f;
    float max_ang = 0.0f;

    ForceTorque ft = new ForceTorque(0, 0, 0, 0 ,0 ,0);
    DescPose desc_p1, desc_p2, offset_pos;
    JointPos j1, j2;
    ExaxisPos epos = new ExaxisPos(0, 0, 0, 0);
    desc_p1 = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p2 = new DescPose(0, 0, 0, 0, 0, 0);
    offset_pos = new DescPose(0, 0, 0, 0, 0, 0);

    j2 = new JointPos(0, 0, 0, 0, 0, 0);
    j1 = new JointPos(0, 0, 0, 0, 0, 0);

    desc_p1.tran.x = 1.299;
    desc_p1.tran.y = -719.159;
    desc_p1.tran.z = 141.314;
    desc_p1.rpy.rx = 177.999;
    desc_p1.rpy.ry = -0.715;
    desc_p1.rpy.rz = -161.937;

    desc_p2.tran.x = 245.047;
    desc_p2.tran.y = -675.509;
    desc_p2.tran.z = 139.538;
    desc_p2.rpy.rx = 177.987;
    desc_p2.rpy.ry = -0.129;
    desc_p2.rpy.rz = -142.238;
    ft.fz = -10.0;

    robot.GetInverseKin(0, desc_p1, -1, ref j1);
    robot.GetInverseKin(0, desc_p2, -1, ref j2);

    robot.MoveJ(j1, desc_p1, 1, 0, 100.0f, 180.0f, 100.0f, epos, -1.0f, 0, offset_
↪pos);
    int rtn = robot.FT_Control(flag, sensor_id, select, ft, ft_pid, adj_sign, ILC_
↪sign, max_dis, max_ang);
    Console.WriteLine($"FT_Control start rtn {rtn}");

    robot.MoveL(j2, desc_p2, 1, 0, 100.0f, 180.0f, 20.0f, -1.0f, epos, 0, 0, offset_
↪pos);
```

(续下页)

(接上页)

```
    flag = 0;
    rtn = robot.FT_Control(flag, sensor_id, select, ft, ft_pid, adj_sign, ILC_sign,␣
↪max_dis, max_ang);
    Console.WriteLine($"FT_Control end rtn {rtn}");
}
```

2.2.11.18 柔顺控制开启

```
/**
* @brief  柔顺控制开启
* @param  [in] p 位置调节系数或柔顺系数
* @param  [in] force 柔顺开启力阈值，单位N
* @return  错误码
*/
int FT_ComplianceStart(float p, float force);
```

2.2.11.19 柔顺控制关闭

```
/**
* @brief  柔顺控制关闭
* @return  错误码
*/
int FT_ComplianceStop();
```

2.2.11.20 代码示例

```
private void btnComplience_Click(object sender, EventArgs e)
{
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");

    byte flag = 1;
    int sensor_id = 1;
    int[] select = new int[6]{ 1, 1, 1, 0, 0, 0 };
    double[] ft_pid = new double[6] { 0.0005f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f };
    byte adj_sign = 0;
    byte ILC_sign = 0;
    float max_dis = 100.0f;
    float max_ang = 0.0f;

```

(续下页)

(接上页)

```
    ForceTorque ft = new ForceTorque(0, 0, 0, 0, 0, 0);
    DescPose desc_p1, desc_p2, offset_pos;
    JointPos j1, j2;

    ExaxisPos epos = new ExaxisPos(0, 0, 0, 0);
    desc_p1 = new DescPose(0, 0, 0, 0, 0, 0);
    desc_p2 = new DescPose(0, 0, 0, 0, 0, 0);
    offset_pos = new DescPose(0, 0, 0, 0, 0, 0);

    j2 = new JointPos(0, 0, 0, 0, 0, 0);
    j1 = new JointPos(0, 0, 0, 0, 0, 0);

    desc_p1.tran.x = 1.299;
    desc_p1.tran.y = -719.159;
    desc_p1.tran.z = 141.314;
    desc_p1.rpy.rx = 177.999;
    desc_p1.rpy.ry = -0.715;
    desc_p1.rpy.rz = -161.937;

    desc_p2.tran.x = 245.047;
    desc_p2.tran.y = -675.509;
    desc_p2.tran.z = 139.538;
    desc_p2.rpy.rx = 177.987;
    desc_p2.rpy.ry = -0.129;
    desc_p2.rpy.rz = -142.238;
    ft.fz = -10.0;

    robot.GetInverseKin(0, desc_p1, -1, ref j1);
    robot.GetInverseKin(0, desc_p2, -1, ref j2);

    ft.fx = -10.0;
    ft.fy = -10.0;
    ft.fz = -10.0;
    robot.FT_Control(flag, sensor_id, select, ft, ft_pid, adj_sign, ILC_sign, max_dis,
↪ max_ang);
    float p = 0.00005f;
    float force = 30.0f;
    int rtn = robot.FT_ComplianceStart(p, force);
    Console.WriteLine($"FT_ComplianceStart rtn {rtn}");
    int count = 15;
    while (count > 0)
    {
        robot.MoveL(j1, desc_p1, 1, 0, 100.0f, 180.0f, 100.0f, -1.0f, epos, 0, 1,␣
```

(续下页)

(接上页)

```
↪offset_pos);
57          robot.MoveL(j2, desc_p2, 1, 0, 100.0f, 180.0f, 100.0f, -1.0f, epos, 0, 0,␣
↪offset_pos);
58          count -= 1;
59      }
60      rtn = robot.FT_ComplianceStop();
61      Console.WriteLine($"FT_ComplianceStop rtn {rtn}");
62      flag = 0;
63      robot.FT_Control(flag, sensor_id, select, ft, ft_pid, adj_sign, ILC_sign, max_dis,
↪ max_ang);
64  }
```

2.2.12 其他接口

2.2.12.1 传动带启动、停止

```
1  /**
2  * @brief 传动带启动、停止
3  * @param [in] status 状态，1-启动，0-停止
4  * @return 错误码
5  */
6  int ConveyorStartEnd(byte status);
```

2.2.12.2 记录 IO 检测点

```
1  /**
2  * @brief 记录IO检测点
3  * @return 错误码
4  */
5  int ConveyorPointIORecord();
```

2.2.12.3 记录 A 点

```
1  /**
2  * @brief 记录A点
3  * @return 错误码
4  */
5  int ConveyorPointARecord();
```

2.2.12.4 记录参考点

```
/**
* @brief 记录参考点
* @return 错误码
*/
int ConveyorRefPointRecord();
```

2.2.12.5 记录 B 点

```
/**
* @brief 记录B点
* @return 错误码
*/
int ConveyorPointBRecord();
```

2.2.12.6 传送带工件 IO 检测

```
/**
* @brief 传送带工件IO检测
* @param [in] max_t 最大检测时间，单位ms
* @return 错误码
*/
int ConveyorIODetect(int max_t);
```

2.2.12.7 获取物体当前位置

```
/**
* @brief 获取物体当前位置
* @param [in] mode 1-跟踪抓取，2-跟踪运动，3-TPD跟踪
* @return 错误码
*/
int ConveyorGetTrackData(int mode);
```

2.2.12.8 传动带跟踪开始

```
/**
* @brief 传动带跟踪开始
* @param [in] status 状态，1-启动，0-停止
* @return 错误码
*/
int ConveyorTrackStart(byte status);
```

2.2.12.9 传动带跟踪停止

```
/**
* @brief 传动带跟踪停止
* @return 错误码
*/
int ConveyorTrackEnd();
```

2.2.12.10 传动带参数配置

```
/**
* @brief 传动带参数配置
* @param [in] encChannel 编码器通道 1~2
* @param [in] resolution 编码器转一圈的脉冲数
* @param [in] lead 编码器转一圈传送带行走距离
* @param [in] wpAxis 工件坐标系编号
↪针对跟踪运动功能选择工件坐标系编号，跟踪抓取、TPD跟踪设为0
* @param [in] vision 是否配视觉  0-不配，1-配
* @param [in] speedRadio 速度比:针对传送带跟踪抓取选项（1-100）其他选项默认为1
* @return 错误码
*/
int ConveyorSetParam(int encChannel, int resolution, double lead, int wpAxis, int
↪vision, double speedRadio);
```

2.2.12.11 设置传动带抓取点补偿

```
/**
* @brief 设置传动带抓取点补偿
* @param [in] cmp 补偿位置 double[3]{x, y, z}
* @return 错误码
*/
int ConveyorCatchPointComp(double[] cmp);
```

2.2.12.12 传送带跟踪直线运动

```
/**
* @brief 传送带跟踪直线运动
* @param [in] name 运动点名称
* @param [in] tool 工具坐标号，范围[0~14]
* @param [in] wobj 工件坐标号，范围[0~14]
* @param [in] vel 速度百分比，范围[0~100]
* @param [in] acc 加速度百分比，范围[0~100],暂不开放
* @param [in] ovl 速度缩放因子，范围[0~100]
* @param [in] blendR [-1.0]-运动到位(阻塞)，[0~1000.0]-平滑半径(非阻塞)，单位mm
* @return 错误码
*/
int ConveyorTrackMoveL(string name, int tool, int wobj, float vel, float acc, float ovl, float blendR);
```

2.2.12.13 代码示例

```
private void btnConvert_Click(object sender, EventArgs e)
    {
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");
    DescPose pos1 = new DescPose(0, 0, 0, 0 ,0 ,0);
    DescPose pos2 = new DescPose(0, 0, 0, 0, 0, 0);

    pos1.tran.x = -351.175;
    pos1.tran.y = 3.389;
    pos1.tran.z = 431.172;
    pos1.rpy.rx = -179.111;
    pos1.rpy.ry = -0.241;
    pos1.rpy.rz = 90.388;

    pos2.tran.x = -333.654;
    pos2.tran.y = -229.003;
    pos2.tran.z = 404.335;
    pos2.rpy.rx = -179.139;
    pos2.rpy.ry = -0.779;
    pos2.rpy.rz = 91.269;
    int rtn = -1;

    double[] cmp = new double[3] { 0, 9.99, 0};
    rtn = robot.ConveyorCatchPointComp(cmp);
    if(rtn != 0)
```

(续下页)

(接上页)

```
    {
        return;
    }
    Console.WriteLine($"ConveyorCatchPointComp: rtn  {rtn}");

    rtn = robot.MoveCart(pos1, 0, 0, 100.0f, 180.0f, 100.0f, -1.0f, -1);
    Console.WriteLine($"MoveCart: rtn  {rtn}");

    rtn = robot.ConveyorIODetect(10000);
    Console.WriteLine($"ConveyorIODetect: rtn  {rtn}");

    robot.ConveyorGetTrackData(1);
    rtn = robot.ConveyorTrackStart(1);
    Console.WriteLine($"ConveyorTrackStart: rtn  {rtn}");

    rtn = robot.ConveyorTrackMoveL("cvrCatchPoint", 0, 0, 100.0f, 0.0f, 100.0f, -1.0f,
↪ 0, 0);
    Console.WriteLine($"ConveyorTrackMoveL: rtn  {rtn}");

    rtn = robot.MoveGripper(1, 59, 43, 21, 30000, 0);
    Console.WriteLine($"MoveGripper: rtn  {rtn}");

    rtn = robot.ConveyorTrackMoveL("cvrRaisePoint", 0, 0, 100.0f, 0.0f, 100.0f, -1.0f,
↪ 0, 0);
    Console.WriteLine($"ConveyorTrackMoveL: rtn  {rtn}");

    rtn = robot.ConveyorTrackEnd();
    Console.WriteLine($"ConveyorTrackEnd: rtn  {rtn}");

    rtn = robot.MoveCart(pos2, 0, 0, 100.0f, 180.0f, 100.0f, -1.0f, -1);
    Console.WriteLine($"MoveCart: rtn  {rtn}");

    rtn = robot.MoveGripper(1, 100, 43, 21, 30000, 0);
    Console.WriteLine($"MoveGripper: rtn  {rtn}");
}
```

2.2.12.14 获取 SSH 公钥

```
/**
* @brief 获取SSH公钥
* @param [out] keygen 公钥
* @return 错误码
*/
int GetSSHKeygen(ref string keygen);
```

2.2.12.15 计算指定路径下文件的 MD5 值

```
/**
* @brief 计算指定路径下文件的MD5值
* @param [in] file_path 文件路径包含文件名，默认Traj文件夹路径为:"/fruser/traj/",如"/fruser/traj/trajHelix_aima_1.txt"
* @param [out] md5 文件MD5值
* @return 错误码
*/
int ComputeFileMD5(string file_path, ref string md5);
```

2.2.12.16 获取机器人急停状态

```
/**
* @brief 获取机器人急停状态
* @param [out] state 急停状态，0-非急停，1-急停
* @return 错误码
*/
int GetRobotEmergencyStopState(ref byte state);
```

2.2.12.17 获取 SDK 与机器人的通讯状态

```
/**
* @brief 获取SDK与机器人的通讯状态
* @param [out] state 通讯状态，0-通讯正常，1-通讯异常
* @return 错误码
*/
int GetSDKComState(ref int state)
```

2.2.12.18 获取安全停止信号

```
/**
* @brief 获取安全停止信号
* @param [out] si0_state 安全停止信号SI0, 0-无效, 1-有效
* @param [out] si1_state 安全停止信号SI1, 0-无效, 1-有效
* @return 错误码
*/
int GetSafetyStopState(ref byte si0_state, ref byte si1_state)
```

2.2.12.19 代码示例

```
private void btnTestOthers_Click(object sender, EventArgs e)
    {
    Robot robot = new Robot();
    robot.RPC("192.168.58.2");
    int rtn = -1;
    string ssh = "";
    rtn = robot.GetSSHKeygen(ref ssh);
    Console.WriteLine($"GetSSHKeygen: ssh {ssh} rtn {rtn}");
    string file_path = "/fruser/test.txt";
    string md5 = "";
    robot.ComputeFileMD5(file_path, ref md5);

    byte state = 255;
    rtn = robot.GetRobotEmergencyStopState(ref state);
    Console.WriteLine($"GetRobotEmergencyStopState: rtn {rtn} state {state}");

    int comState = -1;
    rtn = robot.GetSDKComState(ref comState);
    Console.WriteLine($"GetSDKComState: rtn {rtn} state {comState}");

    byte si0_state = 255;
    byte si1_state = 255;

    rtn = robot.GetSafetyStopState(ref si0_state, ref si1_state);
    Console.WriteLine($"GetSafetyStopState: rtn {rtn} si0_state {si0_state} ␣
↪si1_state {si1_state}");
}
```

2.3 Python

本文档为 Python 版本的二次开发接口文档。

重要： 机器人参数单位说明：机器人位置单位为毫米 (mm)，姿态单位为度 (°)。

重要：

1) 非特别说明的代码示例中都默认机器人已经正常开机使能；

2) 文档中的所有代码示例都默认在机器人的工作空间内没有任何干涉；

3) 实际使用测试时请采用现场机器人的数据使用。

备注： 当前文档适用于 SDK-v2.0.0 版本，向下兼容 v1.x 版本。

2.3.1 机器人基础

2.3.1.1 实例化机器人

原型	`RPC(ip)`
描述	实例化一个机器人对象
参数	• 必选参数 `ip`：机器人的 IP 地址，默认出厂 IP 为“192.168.58.2”
返回值	• 成功：返回一个机器人对象 • 失败：创建的对象会被销毁

2.3.1.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
```

2.3.1.2 查询 SDK 版本号

原型	`GetSDKVersion()`
描述	查询 SDK 版本号
参数	无
返回值	• 错误码成功-0 失败-errcode • 返回值（调用成功返回）[SDK_version, Controller_version]

2.3.1.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetSDKVersion()     # 查询SDK版本号
if ret[0] == 0:
    # 0-无故障，返回格式：[errcode,data],errcode-故障码，data-数据
    print("SDK version is:",ret[1])
else:
    print("the errcode is: ", ret[0])
```

2.3.1.3 获取控制器 IP

原型	`GetControllerIP()`
描述	查询控制器 IP
参数	无
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）ip 控制器 IP

2.3.1.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetControllerIP()     #查询控制器IP
if ret[0] == 0:
    print("controller ip is:",ret[1])
else:
    print("the errcode is: ", ret[0])
```

2.3.1.4 控制机器人手自动模式切换

原型	`Mode(state)`
描述	控制机器人手自动模式切换
参数	• 必选参数 `state`：0-自动模式，1-手动模式
返回值	错误码成功-0 失败- errcode

2.3.1.4.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.Mode(0)    #机器人切入自动运行模式
time.sleep(1)
robot.Mode(1)    #机器人切入手动模式
```

2.3.1.5 机器人拖动模式

2.3.1.5.1 控制机器人进入或退出拖动示教模式

原型	`DragTeachSwitch(state)`
描述	控制机器人进入或退出拖动示教模式
参数	• 必选参数 `state`：1-进入拖动示教模式，0-退出拖动示教模式
返回值	错误码成功-0 失败- errcode

2.3.1.5.2 查询机器人是否处于拖动模式

原型	`IsInDragTeach()`
描述	查询机器人是否处于拖动示教模式
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）state 0-非拖动示教模式，1-拖动示教模式

2.3.1.5.2.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.Mode(1) #机器人切入手动模式
time.sleep(1)
robot.DragTeachSwitch(1)
↪#机器人切入拖动示教模式，必须在手动模式下才能切入拖动示教模式
time.sleep(1)
ret = robot.IsInDragTeach()     #查询是否处于拖动示教模式，1-拖动示教模式，0-
↪非拖动示教模式
if ret[0] == 0:
    print("drag state is:",ret[1])
else:
    print("the errcode is: ", ret[0])
time.sleep(3)
robot.DragTeachSwitch(0)
↪#机器人切入非拖动示教模式，必须在手动模式下才能切入非拖动示教模式
```

(续下页)

(接上页)

```
time.sleep(1)
ret = robot.IsInDragTeach()     #查询是否处于拖动示教模式，1-拖动示教模式，0-非拖动示教模式
if ret[0] == 0:
    print("drag state is:",ret[1])
else:
    print("the errcode is: ", ret[0])
```

2.3.1.6 控制机器人上使能或下使能

原型	`RobotEnable(state)`
描述	控制机器人上使能或下使能
参数	• 必选参数 `state`：1-上使能，0-下使能
返回值	错误码成功-0 失败- errcode

2.3.1.6.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.RobotEnable(0)    #机器人下使能
time.sleep(3)
robot.RobotEnable(1)    #机器人上使能，机器人上电后默认自动上使能
```

2.3.2 机器人运动

2.3.2.1 机器人点动

2.3.2.1.1 jog 点动

原型	`StartJOG(ref,nb,dir,max_dis,vel=20.0,acc=100.0)`
描述	jog 点动
参数	• 必选参数 `ref`：0-关节点动,2-基坐标系点动,4-工具坐标系点动,8-工件坐标系点动； • 必选参数 `nb`：1-1 关节 (x 轴),2-2 关节 (y 轴),3-3 关节 (z 轴),4-4 关节 (rx),5-5 关节 (ry),6-6 关节 (rz)； • 必选参数 `dir`：0-负方向，1-正方向； • 必选参数 `max_dis`：单次点动最大角度/距离，单位 ° 或 mm； • 默认参数 `vel`：速度百分比，[0~100] 默认 20； • 默认参数 `acc`：加速度百分比，[0~100] 默认 100；
返回值	错误码成功-0 失败- errcode

2.3.2.1.2 jog 点动减速停止

原型	`StopJOG(ref)`
描述	jog 点动减速停止
参数	• 必选参数 `ref`：1-关节点动停止,3-基坐标系点动停止,5-工具坐标系点动停止,9-工件坐标系点动停止
返回值	错误码成功-0 失败- errcode

2.3.2.1.3 jog 点动立即停止

原型	`ImmStopJOG()`
描述	jog 点动立即停止
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.1.3.1 代码示例

```python
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
# 机器人单轴点动
robot.StartJOG(0,1,0,20.0,20.0,30.0)    # 单关节运动，
↪StartJOG为非阻塞指令，运动状态下接收其他运动指令（包含StartJOG）会被丢弃
time.sleep(1)
#机器人单轴点动减速停止
# robot.StopJOG(1)
#机器人单轴点动立即停止
robot.ImmStopJOG()
robot.StartJOG(0,2,1,20.0,20.0,30.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(0,3,1,20.0,20.0,30.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(0,4,1,20.0,20.0,30.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(0,5,1,20.0,20.0,30.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(0,6,1,20.0,20.0,30.0)
time.sleep(1)
robot.ImmStopJOG()
# 基坐标
robot.StartJOG(2,1,0,20.0,20.0,100.0)  #基坐标系下点动
time.sleep(1)
#机器人单轴点动减速停止
# robot.StopJOG(2)
# #机器人单轴点动立即停止
robot.ImmStopJOG()
robot.StartJOG(2,1,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(2,2,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(2,3,1,20.0,20.0,100.0)
time.sleep(1)
```

(续下页)

(接上页)

```
robot.ImmStopJOG()
robot.StartJOG(2,4,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(2,5,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(2,6,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
# 工具坐标
robot.StartJOG(4,1,0,20.0,20.0,100.0)  #工具坐标系下点动
time.sleep(1)
#机器人单轴点动减速停止
# robot.StopJOG(5)
# #机器人单轴点动立即停止
robot.ImmStopJOG()
robot.StartJOG(4,1,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(4,2,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(4,3,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(4,4,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(4,5,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(4,6,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
# 工件坐标
robot.StartJOG(8,1,0,20.0,20.0,100.0)  #工件坐标系下点动
time.sleep(1)
#机器人单轴点动减速停止
# robot.StopJOG(9)
# #机器人单轴点动立即停止
robot.ImmStopJOG()
robot.StartJOG(8,1,1,20.0,20.0,100.0)
```

(续下页)

(接上页)

```
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(8,2,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(8,3,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(8,4,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(8,5,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
robot.StartJOG(8,6,1,20.0,20.0,100.0)
time.sleep(1)
robot.ImmStopJOG()
```

2.3.2.2 关节空间运动

原型	`MoveJ(joint_pos, tool, user, desc_pos = [0.0,0.0,0.0,0.0,0.0,0.0], vel = 20.0, acc = 0.0, ovl = 100.0, exaxis_pos = [0.0,0.0,0.0,0.0], blendT = -1.0, offset_flag = 0, offset_pos = [0.0,0.0,0.0,0.0,0.0,0.0])`
描述	关节空间运动
参数	• 必选参数 `joint_pos`: 目标关节位置，单位 [°]; • 必选参数 `tool`: 工具号，[0~14]; • 必选参数 `user`: 工件号，[0~14]; • 默认参数 `desc_pos`: 目标笛卡尔位姿，单位 [mm][°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用正运动学求解返回值; • 默认参数 `vel`: 速度百分比，[0~100] 默认 20.0; • 默认参数 `acc`: 加速度百分比，[0~100]，暂不开放; • 默认参数 `ovl`: 速度缩放因子，[0~100] 默认 100.0; • 默认参数 `exaxis_pos`: 外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]; • 默认参数 `blendT`:[-1.0]-运动到位 (阻塞)，[0~500.0]-平滑时间 (非阻塞)，单位 [ms] 默认-1.0; • 默认参数 `offset_flag`:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • 默认参数 `offset_pos`: 位姿偏移量，单位 [mm][°] 默认 [0.0,0.0,0.0,0.0,0.0,0.0];
返回值	错误码成功-0 失败- errcode

2.3.2.2.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[-168.847,-93.977,-93.118,-80.262,88.985,11.831]
P1=[-558.082,27.343,208.135,-177.205,-0.450,89.288]
eP1=[0.000,0.000,0.000,0.000]
dP1=[1.000,1.000,1.000,1.000,1.000,1.000]
J2=[168.968,-93.977,-93.118,-80.262,88.986,11.831]
P2=[-506.436,236.053,208.133,-177.206,-0.450,67.102]
```

(续下页)

(接上页)

```
eP2=[0.000,0.000,0.000,0.000]
dP2=[1.000,1.000,1.000,1.000,1.000,1.000]
robot.MoveJ(J1,P1,1,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)     #关节空间运动PTP,
↪工具号1，实际测试根据现场数据及工具号使用
robot.MoveJ(J2,P2,1,0,100.0,180.0,100.0,eP2,-1.0,0,dP2)
time.sleep(2)
j1 = robot.GetInverseKin(0,P1,-1)
↪#只有笛卡尔空间坐标的情况下，可用逆运动学接口求解关节位置
print(j1)
j1 = [j1[1],j1[2],j1[3],j1[4],j1[5],j1[6]]
robot.MoveJ(j1,P1,1,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
j2 = robot.GetInverseKin(0,P2,-1)
print(j2)
j2 = [j2[1],j2[2],j2[3],j2[4],j2[5],j2[6]]
robot.MoveJ(j2,P2,1,0,100.0,180.0,100.0,eP2,-1.0,0,dP2)
time.sleep(2)
p1 = robot.GetForwardKin(J1)
↪#只有关节位置的情况下，可用正运动学接口求解笛卡尔空间坐标
print(p1)
p1 = [p1[1],p1[2],p1[3],p1[4],p1[5],p1[6]]
robot.MoveJ(J1,p1,1,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
p2 = robot.GetForwardKin(J2)
print(p2)
p2 = [p2[1],p2[2],p2[3],p2[4],p2[5],p2[6]]
robot.MoveJ(J2,p2,1,0,100.0,180.0,100.0,eP2,-1.0,0,dP2)
```

2.3.2.3 笛卡尔空间直线运动

原型	`MoveL(desc_pos, tool, user, joint_pos = [0.0,0.0,0.0,0.0,0.0,0.0], vel = 20.0, acc = 0.0 , ovl = 100.0, blendR = -1.0, exaxis_pos = [0.0,0.0,0.0,0.0], search = 0, offset_flag = 0, offset_pos = [0.0,0.0,0.0,0.0,0.0,0.0] )`
描述	笛卡尔空间直线运动
参数	• 必选参数 desc_pos: 目标笛卡尔位姿，单位 [mm][°]； • 必选参数 tool: 工具号，[0~14]； • 必选参数 user: 工件号，[0~14]； • 默认参数 joint_pos: 目标关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 vel: 速度百分比，[0~100] 默认 20.0； • 默认参数 acc: 加速度百分比，[0~100]，暂不开放默认 0.0； • 默认参数 ovl: 速度缩放因子，[0~100] 默认 100.0； • 默认参数 blendR:blendR:[-1.0]-运动到位 (阻塞)，[0~1000]-平滑半径 (非阻塞)，单位 [mm] 默认-1.0; • 默认参数 exaxis_pos: 外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]; • 默认参数 search:[0]-不焊丝寻位，[1]-焊丝寻位； • 默认参数 offset_flag:offset_flag:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • 默认参数 offset_pos: 位姿偏移量，单位 [mm][°] 默认 [0.0,0.0,0.0,0.0,0.0,0.0]
返回值	错误码成功-0 失败- errcode

2.3.2.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[95.442,-101.149,-98.699,-68.347,90.580,-47.174]
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
eP1=[0.000,0.000,0.000,0.000]
dP1=[10.000,10.000,10.000,0.000,0.000,0.000]
J2=[123.709,-121.190,-82.838,-63.499,90.471,-47.174]
P2=[-273.856,643.260,259.235,-177.972,-1.494,80.866]
```

(续下页)

(接上页)

```
eP2=[0.000,0.000,0.000,0.000]
dP2=[0.000,0.000,0.000,0.000,0.000,0.000]
J3=[167.066,-95.700,-123.494,-42.493,90.466,-47.174]
P3=[-423.044,229.703,241.080,-173.990,-5.772,123.971]
eP3=[0.000,0.000,0.000,0.000]
dP3=[0.000,0.000,0.000,0.000,0.000,0.000]
robot.MoveL(J1,P1,0,0,100.0,180.0,100.0,-1.0,eP1,0,1 ,dP1)    #笛卡尔空间直线运动
robot.MoveL(J2,P2,0,0,100.0,180.0,100.0,-1.0,eP2,0,0,dP2)
robot.MoveL(J3,P3,0,0,100.0,180.0,100.0,-1.0,eP3,0,0,dP3)
```

2.3.2.4 笛卡尔空间圆弧运动

原型	`MoveC(desc_pos_p, tool_p, user_p, desc_pos_t, tool_t, user_t, joint_pos_p =[0.0,0.0,0.0, 0.0,0.0,0.0], joint_pos_t=[0.0,0.0,0.0,0.0,0.0,0.0], vel_p = 20.0,acc_p=100.0, exaxis_pos_p =[0.0,0.0,0.0,0.0], offset_flag_p = 0, offset_pos_p = [0.0,0.0,0.0,0.0,0.0,0.0], vel_t= 20.0, acc_t=100.0,exaxis_pos_t=[0.0,0.0,0.0,0.0], offset_flag_t = 0, offset_pos_t = [0.0,0.0,0.0,0.0,0.0,0.0], ovl = 100.0, blendR = -1.0)`
描述	笛卡尔空间圆弧运动
参数	• 必选参数 `desc_pos_p`: 路径点笛卡尔位姿，单位 [mm][°]; • 必选参数 `tool_p`: 路径点工具号，[0~14]; • 必选参数 `user_p`: 路径点工件号，[0~14]; • 必选参数 `desc_pos_t`: 目标点笛卡尔位姿，单位 [mm][°]; • 必选参数 `tool_t`: 工具号，[0~14]; • 必选参数 `user_t`: 工件号，[0~14]; • 默认参数 `joint_pos_p`: 路径点关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 `joint_pos_t`: 目标点关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 `vel_p`: 路径点速度百分比，[0~100] 默认 20.0; • 默认参数 `acc_p`: 路径点加速度百分比，[0~100] 暂不开放, 默认 0.0; • 默认参数 `exaxis_pos_p`: 路径点外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]; • 默认参数 `offset_flag_p`: 路径点是否偏移 [0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • 默认参数 `vel_t`: 目标点速度百分比，[0~100] 默认 20.0; • 默认参数 `acc_t`: 目标点加速度百分比，[0~100] 暂不开放默认 0.0; • 默认参数 `exaxis_pos_t`: 目标点外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]; • 默认参数 `offset_flag_t`: 目标点是否偏移 [0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • 默认参数 `offset_pos_t`: 目标点位姿偏移量，单位 [mm][°] 默认 [0.0,0.0,0.0,0.0,0.0,0.0]; • 默认参数 `ovl`:: 速度缩放因子，[0~100] 默认 100.0; • 默认参数 `blendR`:[-1.0]-运动到位 (阻塞)，[0~1000]-平滑半径 (非阻塞)，单位 [mm] 默认-1.0;
返回值	错误码成功-0 失败- errcode

2.3.2.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[121.381,-97.108,-123.768,-45.824,89.877,-47.296]
P1=[-127.772,459.534,221.274,-177.850,-2.507,78.627]
eP1=[0.000,0.000,0.000,0.000]
dP1=[10.000,10.000,10.000,10.000,10.000,10.000]
J2=[138.884,-114.522,-103.933,-49.694,90.688,-47.291]
P2=[-360.468,485.600,196.363,-178.239,-0.893,96.172]
eP2=[0.000,0.000,0.000,0.000]
dP2=[10.000,10.000,10.000,10.000,10.000,10.000]
pa2=[0.0,0.0,100.0,180.0]
J3=[159.164,-96.105,-128.653,-41.170,90.704,-47.290]
P3=[-360.303,274.911,203.968,-176.720,-2.514,116.407]
eP3=[0.000,0.000,0.000,0.000]
dP3=[10.000,10.000,10.000,10.000,10.000,10.000]
pa3=[0.0,0.0,100.0,180.0]
dP=[10.000,10.000,10.000,10.000,10.000,10.000]
robot.MoveJ(J1,P1,0,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)        #关节空间运动PTP
robot.MoveC(J2,P2,pa2,eP2,0,dP2,J3,P3,pa3,eP3,0,dP3,100.0,-1.0)     #笛卡尔空间圆弧运动
```

2.3.2.5 笛卡尔空间整圆运动

原型	`Circle(desc_pos_p,tool_p,user_p,desc_pos_t,tool_t, user_t,joint_pos_p=[0.0,0.0,0.0,0.0,0.0,0.0], joint_pos_t = [0.0,0.0,0.0,0.0,0.0,0.0], vel_p = 20.0, acc_p=0.0, exaxis_pos_p= [0.0,0.0, 0.0,0.0], vel_t=20.0, acc_t = 0.0, exaxis_pos_t =[0.0,0.0,0.0,0.0], ovl=100.0, offset_flag=0, offset_pos= [0.0,0.0,0.0,0.0,0.0,0.0])`
描述	笛卡尔空间整圆运动
参数	• 必选参数 `desc_pos_p`: 路径点笛卡尔位姿，单位 [mm][°]； • 必选参数 `tool_p`: 工具号，[0~14]； • 必选参数 `user_p`: 工件号，[0~14]； • 必选参数 `desc_pos_t`: 目标点笛卡尔位姿，单位 [mm][°]； • 必选参数 `tool_t`: 工具号，[0~14]； • 必选参数 `user_t`: 工件号，[0~14]； • 默认参数 `joint_pos_p`: 路径点关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 `joint_pos_t`: 目标点关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 `vel_p`: 速度百分比，[0~100] 默认 20.0; • 默认参数 `acc_p`: 路径点加速度百分比，[0~100] 暂不开放默认 0.0; • 默认参数 `exaxis_pos_p`: 路径点外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]; • 默认参数 `vel_t`: 目标点速度百分比，[0~100] 默认 20.0; • 默认参数 `acc_t`: 目标点加速度百分比，[0~100] 暂不开放默认 0.0; • 默认参数 `exaxis_pos_t`: 标点外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0] • 默认参数 `ovl`: 速度缩放因子，[0~100] 默认 100.0; • 默认参数 `offset_flag`: 是否偏移 [0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • 默认参数 `offset_pos`: 位姿偏移量，单位 [mm][°] 默认 [0.0,0.0,0.0,0.0,0.0,0.0]
返回值	错误码成功-0 失败- errcode

2.3.2.5.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[121.381,-97.108,-123.768,-45.824,89.877,-47.296]
P1=[-127.772,459.534,221.274,-177.850,-2.507,78.627]
eP1=[0.000,0.000,0.000,0.000]
dP1=[10.000,10.000,10.000,10.000,10.000,10.000]
J2=[138.884,-114.522,-103.933,-49.694,90.688,-47.291]
P2=[-360.468,485.600,196.363,-178.239,-0.893,96.172]
eP2=[0.000,0.000,0.000,0.000]
dP2=[10.000,10.000,10.000,10.000,10.000,10.000]
pa2=[0.0,0.0,100.0,180.0]
J3=[159.164,-96.105,-128.653,-41.170,90.704,-47.290]
P3=[-360.303,274.911,203.968,-176.720,-2.514,116.407]
eP3=[0.000,0.000,0.000,0.000]
dP3=[10.000,10.000,10.000,10.000,10.000,10.000]
pa3=[0.0,0.0,100.0,180.0]
dP=[10.000,10.000,10.000,10.000,10.000,10.000]
robot.MoveJ(J1,P1,0,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)     #关节空间运动PTP
robot.Circle(J2,P2,pa2,eP2,J3,P3,pa3,eP3,100.0,0,dP)     #笛卡尔空间整圆运动
```

2.3.2.6 笛卡尔空间螺旋线运动

原型	`NewSpiral(desc_pos, tool, user, param, joint_pos = [0.0, 0.0,0.0,0.0,0.0,0.0], vel = 20.0, acc = 0.0, exaxis_pos = [0.0,0.0,0.0,0.0], ovl = 100.0, offset_flag = 0, offset_pos = [0.0,0.0,0.0,0.0,0.0,0.0])`
描述	笛卡尔空间螺旋线运动
参数	• 必选参数 `desc_pos`: 目标笛卡尔位姿，单位 [mm][°]； • 必选参数 `tool`: 工具号，[0~14]； • 必选参数 `user`: 工件号，[0~14]； • 必选参数 **`param`:[circle_num, circle_angle, rad_init, rad_add, rotaxis_add, rot_direction]：** circle_num: 螺旋圈数; circle_angle: 螺旋倾角; rad_init: 螺旋初始半径; rad_add: 半径增量; rotaxis_add: 转轴方向增量; rot_direction: 旋转方向，0-顺时针，1-逆时针； • 默认参数 `joint_pos`: 目标关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值； • 默认参数 `vel`: 速度百分比，[0~100] 默认 20.0； • 默认参数 `acc`: 加速度百分比，[0~100] 默认 100.0； • 默认参数 `exaxis_pos`: 外部轴 1 位置 ~ 外部轴 4 位置默认 [0.0,0.0,0.0,0.0]； • 默认参数 `ovl`: 速度缩放因子，[0~100] 默认 100.0； • 默认参数 `offset_flag`:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0； • 默认参数 `offset_pos`: 位姿偏移量，单位 [mm][°] 默认 [0.0,0.0,0.0,0.0,0.0,0.0]
返回值	错误码成功-0 失败- errcode

2.3.2.6.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[127.888,-101.535,-94.860,17.836,96.931,-61.325]
eP1=[0.000,0.000,0.000,0.000]
dP1=[50.0,0.0,0.0,-30.0,0.0,0.0]
J2=[127.888,-101.535,-94.860,17.836,96.931,-61.325]
eP2=[0.000,0.000,0.000,0.000]
dP2=[50.0,0.0,0.0,-5.0,0.0,0.0]
```

(续下页)

(接上页)

```
Pa = [5.0,5.0,50.0,10.0,10.0,0.0]
P1 = robot.GetForwardKin(J1)
↪#只有关节位置的情况下，可用正运动学接口求解笛卡尔空间坐标
print(P1)
P1 = [P1[1],P1[2],P1[3],P1[4],P1[5],P1[6]]
robot.MoveJ(J1,P1,0,0,100.0,180.0,100.0,eP1,0.0,2,dP1)
P2 = robot.GetForwardKin(J2)
↪#只有关节位置的情况下，可用正运动学接口求解笛卡尔空间坐标
print(P2)
P2 = [P2[1],P2[2],P2[3],P2[4],P2[5],P2[6]]
robot.NewSpiral(J2,P2,0,0,100.0,180.0,eP2,100.0,2,dP2,Pa)    #螺旋线运动
```

2.3.2.7 伺服运动开始

原型	ServoMoveStart()
描述	伺服运动开始，配合 ServoJ、ServoCart 指令使用
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.7.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
error,joint_pos = robot.GetActualJointPosDegree()
print("机器人当前关节位置",joint_pos)
joint_pos = [joint_pos[0],joint_pos[1],joint_pos[2],joint_pos[3],joint_pos[4],joint_
↪pos[5]]
error_joint = 0
count =100
error = robot.ServoMoveStart()  #伺服运动开始
print("伺服运动开始错误码",error)
while(count):
    error = robot.ServoJ(joint_pos)    #关节空间伺服模式运动
    if error!=0:
        error_joint =error
    joint_pos[0] = joint_pos[0] + 0.1   #每次1轴运动0.1度，运动100次
    count = count - 1
    time.sleep(0.008)
```

(续下页)

(接上页)

```
print("关节空间伺服模式运动错误码",error_joint)
error = robot.ServoMoveEnd()  #伺服运动结束
print("伺服运动结束错误码",error)
mode = 2  #[0]-绝对运动(基坐标系), [1]-增量运动(基坐标系), [2]-增量运动(工具坐标系)
n_pos = [0.0,0.0,0.5,0.0,0.0,0.0]   #笛卡尔空间位姿增量
error,desc_pos = robot.GetActualTCPPose()
print("机器人当前笛卡尔位置",desc_pos)
count = 100
error_cart =0
error = robot.ServoMoveStart()  #伺服运动开始
print("伺服运动开始错误码",error)
while(count):
    error = robot.ServoCart(mode, n_pos, vel=40)    #笛卡尔空间伺服模式运动
    if error!=0:
        error_cart =error
    count = count - 1
    time.sleep(0.008)
print("笛卡尔空间伺服模式运动错误码", error_cart)
error = robot.ServoMoveEnd()  #伺服运动开始
print("伺服运动结束错误码",error)
```

2.3.2.8 伺服运动结束

原型	`ServoMoveEnd()`
描述	伺服运动结束，配合 ServoJ、ServoCart 指令使用
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.9 关节空间伺服模式运动

原型	`ServoJ(joint_pos, acc = 0.0, vel = 0.0, cmdT = 0.008, filterT = 0.0, gain = 0.0)`
描述	关节空间伺服模式运动
参数	• 必选参数 `joint_pos`: 目标关节位置，单位 [°]; • 默认参数 `acc`: 加速度，范围 [0~100]，暂不开放，默认为 0.0; • 默认参数 `vel`: 速度，范围 [0~100]，暂不开放，默认为 0.0; • 默认参数 `cmdT`: 指令下发周期，单位 s，建议范围 [0.001~0.0016], 默认为 0.008; • 默认参数 `filterT`: 滤波时间，单位 [s]，暂不开放，默认为 0.0; • 默认参数 `gain`: 目标位置的比例放大器，暂不开放，默认为 0.0;
返回值	错误码成功-0 失败- errcode

2.3.2.9.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
joint_pos = robot.GetActualJointPosDegree(0)
print(joint_pos)
joint_pos = [joint_pos[1],joint_pos[2],joint_pos[3],joint_pos[4],joint_pos[5],joint_
↪pos[6]]
acc = 0.0
vel = 0.0
t = 0.008
lookahead_time = 0.0
P = 0.0
count = 100
while(count):
    robot.ServoJ(joint_pos, acc, vel, t, lookahead_time, P)
    joint_pos[0] = joint_pos[0] + 0.1
    count = count - 1
    time.sleep(0.008)
```

2.3.2.10 笛卡尔空间伺服模式运动

原型	`ServoCart(mode, desc_pos, pos_gain = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0] , acc = 0.0, vel = 0.0, cmdT = 0.008, filterT = 0.0, gain = 0.0)`
描述	笛卡尔空间伺服模式运动
参数	• 必选参数 `mode`:[0]-绝对运动 (基坐标系)，[1]-增量运动 (基坐标系)，[2]-增量运动 (工具坐标系)； • 必选参数 `desc_pos`: 目标笛卡尔位置/目标笛卡尔位置增量； • 默认参数 `pos_gain`: 位姿增量比例系数，仅在增量运动下生效，范围 [0~1], 默认为 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]; • 默认参数 `acc`: 加速度，范围 [0~100]，暂不开放，默认为 0.0; • 默认参数 `vel`: 速度，范围 [0~100]，暂不开放，默认为 0.0; • 默认参数 `cmdT`: 指令下发周期，单位 s，建议范围 [0.001~0.0016], 默认为 0.008; • 默认参数 `filterT`: 滤波时间，单位 [s]，暂不开放，默认为 0.0; • 默认参数 `gain`: 目标位置的比例放大器，暂不开放，默认为 0.0;
返回值	错误码成功-0 失败- errcode

2.3.2.10.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
mode = 2  #工具坐标系增量运动
n_pos = [0.0,0.0,0.5,0.0,0.0,0.0]   #笛卡尔空间位姿增量
gain = [0.0,0.0,1.0,0.0,0.0,0.0]
acc = 0.0
vel = 0.0
t = 0.008
lookahead_time = 0.0
P = 0.0
count = 100
while(count):
    robot.ServoCart(mode, n_pos, gain, acc, vel, t, lookahead_time, P)
    count = count - 1
    time.sleep(0.008)
```

2.3.2.11 笛卡尔空间点到点运动

原型	`MoveCart(desc_pos, tool, user, vel = 20.0, acc = 0.0, ovl = 100.0, blendT = -1.0, config = -1)`
描述	笛卡尔空间点到点运动
参数	• 必选参数 `desc_pos`: 目标笛卡尔位置; • 必选参数 `tool`: 工具号，[0~14]; • 必选参数 `user`: 工件号，[0~14]; • 默认参数 `vel`: 速度，范围 [0~100]，默认为 20.0; • 默认参数 `acc`: 加速度，范围 [0~100]，暂不开放, 默认为 0.0; • 默认参数 `ovl`: 速度缩放因子，[0~100]，默认为 100.0; • 默认参数 `blendT`:[-1.0]-运动到位 (阻塞), [0~500]-平滑时间 (非阻塞), 单位 [ms] 默认为 -1.0; • 默认参数 `config`: 关节配置, [-1]-参考当前关节位置求解, [0~7]-依据关节配置求解默认为 -1
返回值	错误码成功-0 失败- errcode

2.3.2.11.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
P2=[-273.856,643.260,259.235,-177.972,-1.494,80.866]
P3=[-423.044,229.703,241.080,-173.990,-5.772,123.971]
robot.MoveCart(P1,0,0,100.0,100.0,100.0,-1.0,-1)        #笛卡尔空间点到点运动
robot.MoveCart(P2,0,0,100.0,100.0,100.0,-1.0,-1)
robot.MoveCart(P3,0,0,100.0,100.0,100.0,0.0,-1)
time.sleep(1)
robot.StopMotion()     #停止运动
```

2.3.2.12 机器人样条运动

2.3.2.12.1 样条运动开始

原型	`SplineStart()`
描述	样条运动开始
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.12.2 样条运动 PTP

原型	`SplinePTP(joint_pos, tool, user, desc_pos = [0.0,0.0,0.0,0.0,0.0,0.0], vel = 20.0, acc = 100.0, ovl = 100.0)`
描述	样条运动 PTP
参数	• 必选参数 `joint_pos`: 目标关节位置，单位 [°]; • 必选参数 `tool`: 工具号，[0~14]; • 必选参数 `user`: 工件号，[0~14]; • 默认参数 `desc_pos`: 目标笛卡尔位姿，单位 [mm][°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用正运动学求解返回值; • 默认参数 `vel`: 速度，范围 [0~100]，默认为 20.0; • 默认参数 `acc`: 加速度，范围 [0~100]，默认为 100.0; • 默认参数 `ovl`: 速度缩放因子，[0~100]，默认为 100.0
返回值	• 成功：[0] • 失败：[errcode]

2.3.2.12.3 样条运动结束

原型	`SplineEnd()`
描述	样条运动结束
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.12.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1 = [114.578,-117.798,-97.745,-54.436,90.053,-45.216]
P1 = [-140.418,619.351,198.369,-179.948,0.023,69.793]
eP1 = [0.000,0.000,0.000,0.000]
dP1 = [0.000,0.000,0.000,0.000,0.000,0.000]
J2 = [115.401,-105.206,-117.959,-49.727,90.054,-45.222]
P2 = [-95.586,504.143,186.880,178.001,2.091,70.585]
J3 = [135.609,-103.249,-120.211,-49.715,90.058,-45.219]
P3 = [-252.429,428.903,188.492,177.804,2.294,90.782]
J4 = [154.766,-87.036,-135.672,-49.045,90.739,-45.223]
P4 = [-277.255,272.958,205.452,179.289,1.765,109.966]
robot.MoveJ(J1,P1,0,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
robot.SplineStart()     #样条运动开始
robot.SplinePTP(J1,P1,0,0,100.0,180.0,100.0)     #样条PTP运动
robot.SplinePTP(J2,P2,0,0,100.0,180.0,100.0)
robot.SplinePTP(J3,P3,0,0,100.0,180.0,100.0)
robot.SplinePTP(J4,P4,0,0,100.0,180.0,100.0)
robot.SplineEnd()      #样条运动结束
```

2.3.2.13 机器人新样条运动

2.3.2.13.1 新样条运动开始

原型	`NewSplineStart(type)`
描述	新样条运动开始
参数	• 必选参数 `type`:0-圆弧过渡，1-给定点位路径点
返回值	错误码成功-0 失败- errcode

2.3.2.13.2 新样条运动结束

原型	`NewSplineEnd()`
描述	新样条运动结束
参数	无
返回值	• 成功：[0] • 失败：[errcode]

2.3.2.13.3 新样条指令点

原型	`NewSplinePoint(desc_pos,tool,user,lastFlag,joint_pos=[0.0,0.0,0.0,0.0,0.0,0.0], vel = 0.0, acc = 0.0, ovl = 100.0 ,blendR = 0.0 )`
描述	新样条指令点
参数	• 必选参数 `desc_pos`: 目标笛卡尔位姿，单位 [mm][°]; • 必选参数 `tool`: 工具号，[0~14]; • 必选参数 `user`: 工件号，[0~14]; • 必选参数 `lastFlag`: 是否为最后一个点，0-否，1-是; • 默认参数 `joint_pos`: 目标关节位置，单位 [°] 默认初值为 [0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用逆运动学求解返回值; • 默认参数 `vel`: 速度，范围 [0~100]，暂不开放，默认为 0.0;; • 默认参数 `acc`: 加速度，范围 [0~100]，暂不开放，默认为 0.0; • 默认参数 `ovl`: 速度缩放因子，[0~100] 默认为 100.0; • 默认参数 `blendR`: [0~1000]-平滑半径，单位 [mm] 默认 0.0;
返回值	错误码成功-0 失败- errcode

2.3.2.13.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1 = [114.578,-117.798,-97.745,-54.436,90.053,-45.216]
P1 = [-140.418,619.351,198.369,-179.948,0.023,69.793]
eP1 = [0.000,0.000,0.000,0.000]
dP1 = [0.000,0.000,0.000,0.000,0.000,0.000]
```

(续下页)

(接上页)

```
J2 = [115.401,-105.206,-117.959,-49.727,90.054,-45.222]
P2 = [-95.586,504.143,186.880,178.001,2.091,70.585]
J3 = [135.609,-103.249,-120.211,-49.715,90.058,-45.219]
P3 = [-252.429,428.903,188.492,177.804,2.294,90.782]
J4 = [154.766,-87.036,-135.672,-49.045,90.739,-45.223]
P4 = [-277.255,272.958,205.452,179.289,1.765,109.966]
robot.MoveJ(J1,P1,0,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
robot.NewSplineStart(1)     #样条运动开始
robot.NewSplinePoint(J1,P1,0,0,50.0,50.0,50.0,0.0,0)     #样条控制点
robot.NewSplinePoint(J2,P2,0,0,50.0,50.0,50.0,0.0,0)
robot.NewSplinePoint(J3,P3,0,0,50.0,50.0,50.0,0.0,0)
robot.NewSplinePoint(J4,P4,0,0,50.0,50.0,50.0,0.0,1)
robot.NewSplineEnd()
```

2.3.2.14 机器人终止运动

原型	StopMotion()
描述	终止运动，使用终止运动需运动指令为非阻塞状态
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.14.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
P2=[-273.856,643.260,259.235,-177.972,-1.494,80.866]
P3=[-423.044,229.703,241.080,-173.990,-5.772,123.971]
robot.MoveCart(P1,0,0,100.0,100.0,100.0,-1.0,-1)        #关节空间点到点运动
robot.MoveCart(P2,0,0,100.0,100.0,100.0,-1.0,-1)
robot.MoveCart(P3,0,0,100.0,100.0,100.0,0.0,-1)    #此条运动指令为非阻塞状态
time.sleep(1)
robot.StopMotion()     #停止运动
```

2.3.2.15 机器人点位整体偏移

2.3.2.15.1 点位整体偏移开始

原型	`PointsOffsetEnable(flag,offset_pos)`
描述	点位整体偏移开始
参数	• 必选参数 `flag`:0-基坐标或工件坐标系下偏移，2-工具坐标系下偏移； • 必选参数 `offset_pos`: 偏移量，单位 [mm][°]。
返回值	错误码成功-0 失败- errcode

2.3.2.15.2 点位整体偏移结束

原型	`PointsOffsetDisable()`
描述	点位整体偏移结束
参数	无
返回值	错误码成功-0 失败- errcode

2.3.2.15.2.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#机器人点位整体偏移
J1=[-168.847,-93.977,-93.118,-80.262,88.985,11.831]
P1=[-558.082,27.343,208.135,-177.205,-0.450,89.288]
eP1=[0.000,0.000,0.000,0.000]
dP1=[10.000,10.000,10.000,0.000,0.000,0.000]
J2=[168.968,-93.977,-93.118,-80.262,88.986,11.831]
P2=[-506.436,236.053,208.133,-177.206,-0.450,67.102]
eP2=[0.000,0.000,0.000,0.000]
dP2=[0.000,0.000,0.000,0.000,0.000,0.000]
robot.MoveJ(J1,P1,1,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
robot.MoveJ(J2,P2,1,0,100.0,180.0,100.0,eP2,-1.0,0,dP2)
time.sleep(2)
flag = 0
offset = [100.0,5.0,6.0,0.0,0.0,0.0]    #位姿偏移量
```

(续下页)

(接上页)

```
robot.PointsOffsetEnable(flag, offset)    #整体偏移开始
robot.MoveJ(J1,P1,1,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)
robot.MoveJ(J2,P2,1,0,100.0,180.0,100.0,eP2,-1.0,0,dP2)
robot.PointsOffsetDisable()   #整体偏移结束
```

2.3.3 机器人 IO

2.3.3.1 设置控制箱数字量输出

原型	`SetDO(id, status, smooth = 0, block = 0)`
描述	设置控制箱数字量输出
参数	• 必选参数 `id`:io 编号，范围 [0~15]； • 必选参数 `status`:0-关，1-开； • 默认参数 `smooth`:0-不平滑，1-平滑默认 0; • 默认参数 `block`:0-阻塞，1-非阻塞默认 0
返回值	错误码成功-0 失败- errcode

2.3.3.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
for i in range(0,16):
    robot.SetDO(i,1,0,0)    #打开控制箱DO
robot.WaitMs(1000)
for i in range(0,16):
    robot.SetDO(i,0,0,0)    #关闭控制箱DO
robot.WaitMs(1000)
```

2.3.3.2 设置工具数字量输出

原型	`SetToolDO (id, status, smooth = 0, block = 0)`
描述	设置工具数字量输出
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 必选参数 `status`:0-关，1-开； • 默认参数 `smooth`:0-不平滑，1-平滑； • 默认参数 `block`:0-阻塞，1-非阻塞。
返回值	错误码成功-0 失败- errcode

2.3.3.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
for i in range(0,2):
    robot.SetToolDO(i,1,0,0)     #打开工具DO
robot.WaitMs(1000)
for i in range(0,2):
    robot.SetToolDO(i,0,0,0)     #关闭工具DO
```

2.3.3.3 设置控制箱模拟量输出

原型	`SetAO(id,value,block = 0)`
描述	设置控制箱模拟量输出
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 必选参数 `value`: 电流或电压值百分比，范围 [0~100%] 对应电流值 [0~20mA] 或电压 [0~10V]； • 默认参数 `block`:[0]-阻塞，[1]-非阻塞默认 0
返回值	错误码成功-0 失败- errcode

2.3.3.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetAO(0,0.0,0)     # 设置控制箱模拟量输出
robot.WaitMs(1000)
robot.SetAO(1,100.0,0)
```

2.3.3.4 设置工具模拟量输出

原型	`SetToolAO(id,value,block = 0)`
描述	设置工具模拟量输出
参数	• 必选参数 `id`:io 编号，范围 [0]； • 必选参数 `value`: 电流或电压值百分比，范围 [0~100%] 对应电流值 [0~20mA] 或电压 [0~10V]； • 默认参数 `block`:[0]-阻塞，[1]-非阻塞默认 0
返回值	错误码成功-0 失败- errcode

2.3.3.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetToolAO(0,100.0,0)    # 设置工具模拟量输出
robot.WaitMs(1000)
robot.SetToolAO(0,0.0,0)
```

2.3.3.5 获取控制箱数字量输入

原型	`GetDI(id, block = 0)`
描述	获取控制箱数字量输入
参数	• 必选参数 `id`:io 编号，范围 [0~15]； • 默认参数 `block`:0-阻塞，1-非阻塞默认 0
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）di: 0-低电平，1-高电平

2.3.3.5.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
di = robot.GetDI(0,0)    # 获取控制箱数字量输入
print(di)
```

2.3.3.6 获取工具数字量输入

原型	`GetToolDI(id, block = 0)`
描述	获取工具数字量输入
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 默认参数 `block`:0-阻塞，1-非阻塞默认 0
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）di: 0-低电平，1-高电平

2.3.3.6.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
tool_di = robot.GetToolDI(1,0)    # 获取工具数字量输入
print(tool_di)
```

2.3.3.7 等待控制箱数字量输入

原型	`WaitDI(id,status,maxtime,opt)`
描述	等待控制箱数字量输入
参数	• 必选参数 `id`:io 编号，范围 [0~15]； • 必选参数 `status`:0-关，1-开； • 必选参数 `maxtime`: 最大等待时间，单位 [ms]； • 必选参数 `opt`: 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
返回值	错误码成功-0 失败- errcode

2.3.3.7.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitDI(0,1,0,2)     # 一直等待控制箱数字量输入
```

2.3.3.8 等待控制箱多路数字量输入

原型	`WaitMultiDI(mode,id,status,maxtime,opt)`
描述	等待控制箱多路数字量输入
参数	• 必选参数 `mode`:[0]-多路与，[1]-多路或； • 必选参数 `id`:io 编号，bit0~bit7 对应 DI0~DI7，bit8~bit15 对应 CI0~CI7； • 必选参数 `status`:bit0~bit7 对应 DI0~DI7 状态，bit8~bit15 对应 CI0~CI7 状态位的状态 [0]-关，[1]-开； • 必选参数 `maxtime`: 最大等待时间，单位 [ms]； • 必选参数 `opt`: 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	错误码成功-0 失败- errcode

2.3.3.8.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitMultiDI(1,3,3,10000,2)    #  一直等待控制箱多路数字量输入
```

2.3.3.9 等待工具数字量输入

原型	`WaitToolDI(id,status,maxtime,opt)`
描述	等待末端数字量输入
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 必选参数 `status`:0-关，1-开； • 必选参数 `maxtime`: 最大等待时间，单位 [ms]； • 必选参数 `opt`: 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
返回值	错误码成功-0 失败- errcode

2.3.3.9.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitToolDI(1,1,0,2)     #  一直等待工具数字量输入
```

2.3.3.10 获取控制箱模拟量输入

原型	`GetAI(id, block = 0)`
描述	获取控制箱模拟量输入
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 默认参数 `block`:0-阻塞，1-非阻塞默认 0
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

2.3.3.10.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ai = robot.GetAI(0,1)    #  获取控制箱模拟量输入
print(ai)
```

2.3.3.11 获取工具模拟量输入

原型	`GetToolAI (id, block = 0)`
描述	获取末端模拟量输入
参数	• 必选参数 `id`:io 编号，范围 [0]； • 默认参数 `block`:0-阻塞，1-非阻塞默认 0
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

2.3.3.11.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
tool_ai = robot.GetToolAI(0,1)     #    获取工具模拟量输入
print(tool_ai)
```

2.3.3.12 等待控制箱模拟量输入

原型	`WaitAI(id,sign,value,maxtime,opt)`
描述	等待控制箱模拟量输入
参数	• 必选参数 `id`:io 编号，范围 [0~1]； • 必选参数 `sign`:0-大于，1-小于 • 必选参数 `value`: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]； • 必选参数 `maxtime`: 最大等待时间，单位 [ms]； • 必选参数 `opt`: 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
返回值	错误码成功-0 失败- errcode

2.3.3.12.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitAI(0,0,50,0,2)    #  一直等待控制箱模拟量输入
```

2.3.3.13 等待工具模拟量输入

原型	`WaitToolAI(id,sign,value,maxtime,opt)`
描述	等待末端模拟量输入
参数	• 必选参数 `id`:io 编号，范围 [0]； • 必选参数 `sign`:0-大于，1-小于 • 必选参数 `value`: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]； • 必选参数 `maxtime`: 最大等待时间，单位 [ms]； • 必选参数 *'opt'*: 超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待
返回值	错误码成功-0 失败- errcode

2.3.3.13.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitToolAI(0,0,50,0,2)    #  一直等待工具模拟量输入
```

2.3.4 机器人常用设置

2.3.4.1 设置全局速度

原型	`SetSpeed(vel)`
描述	设置全局速度
参数	• 必选参数 `vel`: 速度百分比，范围 [0~100]
返回值	错误码成功-0 失败- errcode

2.3.4.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetSpeed(20)    # 设置全局速度，手动模式与自动模式独立设置
```

2.3.4.2 设置系统变量值

原型	`SetSysVarValue(id,value)`
描述	设置系统变量
参数	• 必选参数 `id`：变量编号，范围 [1~20]; • 必选参数 `value`：变量值
返回值	错误码成功-0 失败- errcode

2.3.4.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
for i in range(1,21):
    robot.SetSysVarValue(i,i+0.5)     #  设置系统变量值
robot.WaitMs(1000)
for i in range(1,21):
    sys_var = robot.GetSysVarValue(i)   #  查询系统变量值
    print(sys_var)
```

2.3.4.3 设置工具参考点-六点法

原型	`SetToolPoint(point_num)`
描述	设置工具参考点-六点法
参数	必选参数 `point_num`：点编号, 范围 [1~6]
返回值	错误码成功-0 失败- errcode

2.3.4.3.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
t_coord = [1.0,2.0,3.0,4.0,5.0,6.0]
for i in range(1,7):
    robot.DragTeachSwitch(1)#切入拖动示教模式
    time.sleep(5)
    error = robot.SetToolPoint(i)
↪#实际应当控制机器人按照要求移动到合适位置后再发送指令
    print("六点法设置工具坐标系，记录点",i,"错误码",error)
    robot.DragTeachSwitch(0)
    time.sleep(1)
error = robot.ComputeTool()
print("六点法设置工具坐标系错误码",error)
```

2.3.4.4 计算工具坐标系-六点法

原型	`ComputeTool()`
描述	计算工具坐标系-六点法（设置完六个工具参考点后再进行计算）
参数	无
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）tcp_pose [x,y,z,rx,ry,rz] 工具坐标系

2.3.4.5 设置工具参考点-四点法

原型	`SetTcp4RefPoint(point_num)`
描述	设置工具参考点-四点法
参数	必选参数 `point_num`：点编号, 范围 [1~4]
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）tcp_pose [x,y,z,rx,ry,rz] 工具坐标系

2.3.4.5.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
t_coord = [1.0,2.0,3.0,4.0,5.0,6.0]
for i in range(1,5):
    robot.DragTeachSwitch(1)#切入拖动示教模式
    time.sleep(5)
    error = robot.SetTcp4RefPoint(i) #应当控制机器人按照要求移动到合适位置后再发送指令
    print("四点法设置工具坐标系，记录点",i,"错误码",error)
    robot.DragTeachSwitch(0)
    time.sleep(1)
error,t_coord= robot.ComputeTcp4()
print("四点法设置工具坐标系错误码",error,"工具TCP",t_coord)
```

2.3.4.6 计算工具坐标系-四点法

原型	`ComputeTcp4()`
描述	计算工具坐标系-四点法（设置完四个工具参考点后再进行计算）
参数	无
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）tcp_pose [x,y,z,rx,ry,rz] 工具坐标系

2.3.4.7 设置工具坐标系

原型	`SetToolCoord(id,t_coord,type,install)`
描述	设置工具坐标系
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `t_coord`: 工具中心点相对末端法兰中心位姿，单位 [mm][°]； • 必选参数 `type`:0-工具坐标系，1-传感器坐标系； • 必选参数 `install`: 安装位置，0-机器人末端，1-机器人外部
返回值	错误码成功-0 失败- errcode

2.3.4.7.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
t_coord = [1.0,2.0,3.0,4.0,5.0,6.0]
robot.SetToolCoord(10,t_coord,0,0)  #  设置工具坐标系
```

2.3.4.8 设置工具坐标系列表

原型	`SetToolList(id,t_coord ,type,install)`
描述	设置工具坐标系列表
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `t_coord`:[x,y,z,rx,ry,rz] 工具中心点相对末端法兰中心位姿，单位 [mm][°]； • 必选参数 `type`:0-工具坐标系，1-传感器坐标系； • 必选参数 `install`: 安装位置，0-机器人末端，1-机器人外部
返回值	错误码成功-0 失败- errcode

2.3.4.9 设置外部工具参考点-三点法

原型	SetExTCPPoint(point_num)
描述	设置外部工具参考点-三点法
参数	必选参数 point_num：点编号, 范围 [1~3]
返回值	错误码成功-0 失败- errcode

2.3.4.9.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
etcp = [1.0,2.0,3.0,4.0,5.0,6.0]
etool = [21.0,22.0,23.0,24.0,25.0,26.0]
for i in range(1,4):
    error = robot.SetExTCPPoint(i) #应当控制机器人按照要求移动到合适位置后再发送指令
    print("三点法设置外部工具坐标系，记录点",i,"错误码",error)
    time.sleep(1)
error,etcp = robot.ComputeExTCF()
print("三点法设置外部工具坐标系错误码",error,"外部工具TCP",etcp)
error = robot.SetExToolCoord(10,etcp,etool)
print("设置外部工具坐标系错误码",error)
error = robot.SetExToolList(10,etcp,etool)
print("设置外部工具坐标系列表错误码",error)
```

2.3.4.10 计算外部工具坐标系-三点法

原型	ComputeExTCF (point_num)
描述	计算外部工具坐标系-三点法（设置完三个参考点后再进行计算）
参数	必选参数 point_num：点编号, 范围 [1~3]
返回值	• 错误码成功-0 失败- errcode • etcp [x,y,z,rx,ry,rz] 外部工具坐标系

2.3.4.11 设置外部工具坐标系

原型	`SetExToolCoord(id,etcp ,etool)`
描述	设置外部工具坐标系
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `etcp`: 外部工具坐标系，单位 [mm][°]； • 必选参数 `etool`: 末端工具坐标系，单位 [mm][°]；
返回值	错误码成功-0 失败- errcode

2.3.4.11.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
etcp = [1.0,2.0,3.0,4.0,5.0,6.0]
etool = [21.0,22.0,23.0,24.0,25.0,26.0]
robot.SetExToolCoord(10,etcp,etool)
```

2.3.4.12 设置外部工具坐标系列表

原型	`SetExToolList(id,etcp ,etool)`
描述	设置外部工具坐标系列表
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `etcp`: 外部工具坐标系，单位 [mm][°]； • 必选参数 `etool`: 末端工具坐标系，单位 [mm][°]；
返回值	错误码成功-0 失败- errcode

2.3.4.12.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
etcp = [1.0,2.0,3.0,4.0,5.0,6.0]
etool = [21.0,22.0,23.0,24.0,25.0,26.0]
robot.SetExToolList(10,etcp,etool)
```

2.3.4.13 设置工件参考点-三点法

原型	`SetWObjCoordPoint(point_num)`
描述	设置工件参考点-三点法
参数	必选参数 `point_num`: 点编号, 范围 [1~3]
返回值	错误码成功-0 失败- errcode

2.3.4.13.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
w_coord = [11.0,12.0,13.0,14.0,15.0,16.0]
robot.SetToolList(0,[0,0,0,0,0,0],0,0)#设置参考点前应当将工具和工件号坐标系切换至0
robot.SetWObjList(0,[0,0,0,0,0,0])
for i in range(1,4):
    error = robot.SetWObjCoordPoint(i)
↪#实际应当控制机器人按照要求移动到合适位置后再发送指令
    print("三点法设置工件坐标系，记录点",i,"错误码",error)
    time.sleep(1)
error, w_coord = robot.ComputeWObjCoord(0)
print("三点法计算工件坐标系错误码",error,"工件坐标系", w_coord)
```

2.3.4.14 计算工件坐标系-三点法

原型	`ComputeWObjCoord()`
描述	计算工件坐标系-三点法（三个参考点设置完后再进行计算;
参数	`method` 计算方式:0：原点-x 轴-z 轴 1：原点-x 轴-xy 平面
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）wobj_pose [x,y,z,rx,ry,rz] 工件坐标系

2.3.4.15 设置工件坐标系

原型	`SetWObjCoord(id,w_coord)`
描述	设置工件坐标系
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `w_coord`: 坐标系相对位姿，单位 [mm][°]；
返回值	错误码成功-0 失败- errcode

2.3.4.15.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
w_coord = [11.0,12.0,13.0,14.0,15.0,16.0]
robot.SetWObjCoord(11,w_coord)
```

2.3.4.16 设置工件坐标系列表

原型	`SetWObjList(id,w_coord)`
描述	设置工件坐标系列表
参数	• 必选参数 `id`: 坐标系编号，范围 [0~14]； • 必选参数 `w_coord`: 坐标系相对位姿，单位 [mm][°]；
返回值	错误码成功-0 失败- errcode

2.3.4.16.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
w_coord = [11.0,12.0,13.0,14.0,15.0,16.0]
robot.SetWObjList(11,w_coord)
```

2.3.4.17 设置末端负载重量

原型	`SetLoadWeight(weight)`
描述	设置末端负载重量, 错误负载重量设置可能会导致拖动模式下机器人失控
参数	• 必选参数 `weight`: 单位 [kg]
返回值	错误码成功-0 失败- errcode

2.3.4.17.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetLoadWeight(3.0)    # 设置负载重量
```

2.3.4.18 设置机器人安装方式-固定安装

原型	`SetRobotInstallPos(method)`
描述	设置机器人安装方式-固定安装, 错误安装方式设置会导致拖动模式下机器人失控
参数	• 必选参数 `method`:0-平装，1-侧装，2-挂装
返回值	错误码成功-0 失败- errcode

2.3.4.18.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetRobotInstallPos(0)    #    设置机器人安装方式
```

2.3.4.19 设置机器人安装角度-自由安装

原型	`SetRobotInstallAngle(yangle,zangle)`
描述	设置机器人安装角度-自由安装, 错误安装角度设置会导致拖动模式下机器人失控
参数	• 必选参数 `yangle`: 倾斜角 • 必选参数 `zangle`: 旋转角
返回值	错误码成功-0 失败- errcode

2.3.4.19.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetRobotInstallAngle(0.0,0.0)     #    设置机器人安装角度
```

2.3.4.20 设置末端负载质心坐标

原型	`SetLoadCoord(x,y,z)`
描述	设置末端负载质心坐标, 错误负载质心设置可能会导致拖动模式下机器人失控
参数	• 必选参数 `x:`,“必选参数 y:,``必选参数 z`: 质心坐标，单位 [mm]
返回值	错误码成功-0 失败- errcode

2.3.4.20.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetLoadCoord(3.0,4.0,5.0)     #    设置负载质心坐标
```

2.3.4.21 等待指定时间

原型	`WaitMs(t_ms)`
描述	等待指定时间
参数	• 必选参数 `t_ms`: 单位 [ms]
返回值	错误码成功-0 失败- errcode

2.3.4.21.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.WaitMs(1000)     #  等待1000ms
```

2.3.5 机器人安全设置

2.3.5.1 设置碰撞等级

原型	`SetAnticollision (mode,level,config)`
描述	设置碰撞等级
参数	• 必选参数 `mode`:0-等级，1-百分比； • 必选参数 `level=[j1,j2,j3,j4,j5,j6]`: 碰撞阈值； • 必选参数 `config`:0-不更新配置文件，1-更新配置文件
返回值	错误码成功-0 失败- errcode

2.3.5.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
level = [1.0,2.0,3.0,4.0,5.0,6.0]
robot.SetAnticollision(0,level,1)      #  设置碰撞等级
level = [50.0,20.0,30.0,40.0,50.0,60.0]
robot.SetAnticollision(1,level,1)      #  设置碰撞百分比
```

2.3.5.2 设置碰撞后策略

原型	`SetCollisionStrategy (strategy)`
描述	设置碰撞后策略
参数	• 必选参数 `strategy`：0-报错暂停，1-继续运行
返回值	错误码成功-0 失败- errcode

2.3.5.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetCollisionStrategy(1)     # 设置碰撞后策略，1-继续运行
```

2.3.5.3 设置正限位

原型	`SetLimitPositive(p_limit)`
描述	设置正限位
参数	• 必选参数 `p_limit=[j1,j2,j3,j4,j5,j6]`：六个关节位置
返回值	错误码成功-0 失败- errcode

2.3.5.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
p_limit = [170.0,80.0,150.0,80.0,170.0,160.0]
robot.SetLimitPositive(p_limit)    #  设置正限位
```

2.3.5.4 设置负限位

原型	`SetLimitNegative(n_limit)`
描述	设置负限位
参数	• 必选参数 `n_limit=[j1,j2,j3,j4,j5,j6]`：六个关节位置
返回值	错误码成功-0 失败- errcode

2.3.5.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
n_limit = [-170.0,-260.0,-150.0,-260.0,-170.0,-160.0]
robot.SetLimitNegative(n_limit)    # 设置负限位
```

2.3.5.5 错误状态清除

原型	`ResetAllError()`
描述	错误状态清除，只能清除可复位的错误
参数	无
返回值	错误码成功-0 失败- errcode

2.3.5.5.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.ResetAllError()     #  错误状态清除
```

2.3.5.6 关节摩擦力补偿开关

原型	`FrictionCompensationOnOff(state)`
描述	关节摩擦力补偿开关
参数	• 必选参数 `state`：0-关，1-开
返回值	错误码成功-0 失败- errcode

2.3.5.6.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FrictionCompensationOnOff(1)    #  关节摩擦力补偿开
```

2.3.5.7 设置关节摩擦力补偿系数-正装

原型	`SetFrictionValue_level(coeff)`
描述	设置关节摩擦力补偿系数-固定安装-正装
参数	• 必选参数 `coeff=[j1,j2,j3,j4,j5,j6]`：六个关节补偿系数
返回值	错误码成功-0 失败- errcode

2.3.5.7.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FrictionCompensationOnOff(1)    #  关节摩擦力补偿开
lcoeff = [0.9,0.9,0.9,0.9,0.9,0.9]
robot.SetFrictionValue_level(lcoeff)    #  设置关节摩擦力补偿系数
```

2.3.5.8 设置关节摩擦力补偿系数-侧装

原型	`SetFrictionValue_wall(coeff)`
描述	设置关节摩擦力补偿系数-固定安装-侧装
参数	• 必选参数 `coeff=[j1,j2,j3,j4,j5,j6]`：六个关节补偿系数
返回值	错误码成功-0 失败- errcode

2.3.5.8.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FrictionCompensationOnOff(1)    #  关节摩擦力补偿开
wcoeff = [0.4,0.4,0.4,0.4,0.4,0.4]
robot.SetFrictionValue_wall(wcoeff)   #  设置关节摩擦力补偿系数
```

2.3.5.9 设置关节摩擦力补偿系数-倒装

原型	`SetFrictionValue_ceiling(coeff)`
描述	设置关节摩擦力补偿系数-固定安装-倒装
参数	• 必选参数 `coeff=[j1,j2,j3,j4,j5,j6]`：六个关节补偿系数
返回值	错误码成功-0 失败- errcode

2.3.5.9.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FrictionCompensationOnOff(1)    #  关节摩擦力补偿开
ccoeff = [0.6,0.6,0.6,0.6,0.6,0.6]
robot.SetFrictionValue_ceiling(ccoeff)  #  设置关节摩擦力补偿系数
```

2.3.5.10 设置关节摩擦力补偿系数-自由安装

原型	`SetFrictionValue_freedom(coeff)`
描述	设置关节摩擦力补偿系数-自由安装
参数	• 必选参数 `coeff=[j1,j2,j3,j4,j5,j6]`：六个关节补偿系数
返回值	错误码成功-0 失败- errcode

2.3.5.10.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FrictionCompensationOnOff(1)    #  关节摩擦力补偿开
fcoeff = [0.5,0.5,0.5,0.5,0.5,0.5]
robot.SetFrictionValue_freedom(fcoeff)    #  设置关节摩擦力补偿系数
```

2.3.6 机器人状态查询

2.3.6.1 获取机器人安装角度

原型	`GetRobotInstallAngle()`
描述	获取机器人安装角度
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[yangle,zangle] yangle-倾斜角,zangle-旋转角

2.3.6.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetRobotInstallAngle()  # 获取机器人安装角度
print(ret)
```

2.3.6.2 获取系统变量值

原型	`GetSysVarValue(id)`
描述	获取系统变量值
参数	• 必选参数 `id`：系统变量编号，范围 [1~20]
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）var_value：系统变量值

2.3.6.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
for i in range(1,21):
    robot.SetSysVarValue(i,i+0.5)    #  设置系统变量值
robot.WaitMs(1000)
for i in range(1,21):
    sys_var = robot.GetSysVarValue(i)  #  查询系统变量值
    print(sys_var)
```

2.3.6.3 获取当前关节位置 (角度)

原型	`GetActualJointPosDegree(flag = 1)`
描述	获取关节当前位置 (角度)
参数	• `flag`：0-阻塞，1-非阻塞
返回值	错误码成功-0 失败- errcode - 返回值 (调用成功返回) joint_pos=[j1,j2,j3,j4,j5,j6]

2.3.6.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualJointPosDegree(0)  # 获取机器人当前关节位置
print(ret)
```

2.3.6.4 获取当前关节位置 (弧度)

原型	`GetActualJointPosRadian(flag = 1)`
描述	获取关节当前位置 (弧度)
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）joint_pos=[j1,j2,j3,j4,j5,j6]

2.3.6.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualJointPosRadian(0)  # 获取机器人当前关节位置
print(ret)
```

2.3.6.5 获取关节反馈速度-deg/s

原型	`GetActualJointSpeedsDegree (flag = 1 )`
描述	获取关节反馈速度-deg/s
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）speed=[j1,j2,j3,j4,j5,j6]

2.3.6.5.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetActualJointSpeedsDegree()
print("获取关节反馈速度-deg/s", ret)
```

2.3.6.6 获取 TCP 指令合速度

原型	`GetTargetTCPCompositeSpeed (flag = 1)`
描述	获取 TCP 指令合速度
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[tcp_speed,ori_speed] tcp_speed 线性合速度 ori_speed 姿态合速度

2.3.6.6.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetTargetTCPCompositeSpeed()
print("获取TCP指令合速度", ret)
```

2.3.6.7 获取 TCP 反馈合速度

原型	`GetActualTCPCompositeSpeed (flag = 1)`
描述	获取 TCP 反馈合速度
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[tcp_speed,ori_speed] tcp_speed 线性合速度 ori_speed 姿态合速度

2.3.6.7.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetActualTCPCompositeSpeed()
print("获取TCP反馈合速度", ret)
```

2.3.6.8 获取 TCP 指令速度

原型	`GetTargetTCPSpeed (flag = 1)`
描述	获取 TCP 指令速度
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）speed [x,y,z,rx,ry,rz]

2.3.6.8.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetTargetTCPSpeed()
print("获取TCP指令速度", ret)
```

2.3.6.9 获取 TCP 反馈速度

原型	`GetActualTCPSpeed (flag = 1)`
描述	获取 TCP 反馈速度
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）speed [x,y,z,rx,ry,rz]

2.3.6.9.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetActualTCPSpeed()
print("获取TCP反馈速度", ret)
```

2.3.6.10 获取当前工具位姿

原型	`GetActualTCPPose (flag = 1)`
描述	获取当前工具位姿
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）tcp_pose=[x,y,z,rx,ry,rz]

2.3.6.10.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualTCPPose(0)  # 获取机器人当前工具位姿
print(ret)
```

2.3.6.11 获取当前工具坐标系编号

原型	`GetActualTCPNum (flag = 1)`
描述	获取当前工具坐标系编号
参数	• 默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）tool_id: 工具坐标系编号

2.3.6.11.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualTCPNum(0)  # 获取机器人当前工具坐标系编号
print(ret)
```

2.3.6.12 获取当前工件坐标系编号

原型	`GetActualWObjNum (flag = 1)`
描述	获取当前工件坐标系编号
参数	默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）wobj_id: 工件坐标系编号

2.3.6.12.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualWObjNum(0)  # 获取机器人当前工件坐标系编号
print(ret)
```

2.3.6.13 获取当前末端法兰位姿

原型	`GetActualToolFlangePose (flag = 1)`
描述	获取当前末端法兰位姿
参数	• 默认参数 `flag`：默认参数 flag：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）flange_pose=[x,y,z,rx,ry,rz]

2.3.6.13.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetActualToolFlangePose(0)  # 获取机器人当前末端法兰位姿
print(ret)
```

2.3.6.14 逆运动学求解

原型	`GetInverseKin(type,desc_pos,config)`
描述	逆运动学，笛卡尔位姿求解关节位置
参数	• 必选参数 `type`:0-绝对位姿 (基坐标系)，1-相对位姿（基坐标系），2-相对位姿（工具坐标系） • 必选参数 `desc_pose`:[x,y,z,rx,ry,rz], 工具位姿，单位 [mm][°] • 默认参数 `config`: 关节配置，[-1]-参考当前关节位置求解，[0~7]-依据关节配置求解默认-1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）joint_pos=[j1,j2,j3,j4,j5,j6]

2.3.6.14.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
ret = robot.GetInverseKin(0,P1,-1)
print(ret)
```

2.3.6.15 逆运动学求解-指定参考位置

原型	`GetInverseKinRef(type,desc_pos,joint_pos_ref)`
描述	逆运动学，工具位姿求解关节位置，参考指定关节位置求解
参数	• 必选参数 `type`:0-绝对位姿 (基坐标系)，1-相对位姿（基坐标系），2-相对位姿（工具坐标系） • 必选参数 `desc_pos`：[x,y,z,rx,ry,rz] 工具位姿，单位 [mm][°] • 必选参数 `joint_pos_ref`：[j1,j2,j3,j4,j5,j6]，关节参考位置，单位 [°]
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）joint_pos=[j1,j2,j3,j4,j5,j6]

2.3.6.15.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
J1=[95.442,-101.149,-98.699,-68.347,90.580,-47.174]
ret = robot.GetInverseKinRef(0,P1,J1)
print(ret)
```

2.3.6.16 逆运动学求解-是否有解

原型	`GetInverseKinHasSolution(type,desc_pos,joint_pos_ref)`
描述	逆运动学，工具位姿求解关节位置是否有解
参数	• 必选参数 `type`:0-绝对位姿 (基坐标系)，1-相对位姿（基坐标系），2-相对位姿（工具坐标系） • 必选参数 `desc_pos`：[x,y,z,rx,ry,rz] 工具位姿，单位 [mm][°] • 必选参数 `joint_pos_ref`：[j1,j2,j3,j4,j5,j6]，关节参考位置，单位 [°]
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）result: “True” -有解, “False” -无解

2.3.6.16.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[75.414,568.526,338.135,-178.348,-0.930,52.611]
J1=[95.442,-101.149,-98.699,-68.347,90.580,-47.174]
ret = robot.GetInverseKinHasSolution(0,P1,J1)
print(ret)
```

2.3.6.17 正运动学求解

原型	`GetForwardKin(joint_pos)`
描述	正运动学，关节位置求解工具位姿
参数	• 必选参数 `joint_pos`:[j1,j2,j3,j4,j5,j6]: 关节位置，单位 [°]
返回值	• 错误码成功-0 失败- errcode • 返回值（调用成功返回）desc_pos=[x,y,z,rx,ry,rz]

2.3.6.17.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[95.442,-101.149,-98.699,-68.347,90.580,-47.174]
ret = robot.GetForwardKin(J1)
print(ret)
```

2.3.6.18 获取当前关节转矩

原型	`GetJointTorques (flag = 1)`
描述	获取当前关节转矩
参数	默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）torques=[j1,j2,j3,j4,j5,j6]

2.3.6.18.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetJointTorques(0)  # 获取机器人当前关节转矩
print(ret)
```

2.3.6.19 获取当前负载的重量

原型	`GetTargetPayload (flag = 1)`
描述	获取当前负载的质量
参数	默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）weight 单位 [kg]

2.3.6.19.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetTargetPayload(0)  # 获取机器人当前负载重量
print(ret)
```

2.3.6.20 获取当前负载的质心

原型	`GetTargetPayloadCog (flag = 1)`
描述	获取当前负载的质心
参数	默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）cog=[x,y,z]: 质心坐标，单位 [mm]

2.3.6.20.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetTargetPayloadCog(0)  # 获取机器人当前负载质心
print(ret)
```

2.3.6.21 获取当前工具坐标系

原型	GetTCPOffset (flag = 1)
描述	获取当前工具坐标系
参数	默认参数 flag：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）tcp_offset=[x,y,z,rx,ry,rz]: 相对位姿，单位 [mm][°]

2.3.6.21.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetTCPOffset(0)  # 获取机器人当前工具坐标系
print(ret)
```

2.3.6.22 获取当前工件坐标系

原型	GetWObjOffset (flag = 1)
描述	获取当前工件坐标系
参数	默认参数 flag：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）wobj_offset=[x,y,z,rx,ry,rz]: 相对位姿，单位 [mm][°]

2.3.6.22.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetWObjOffset(0)  # 获取机器人当前工件坐标系
print(ret)
```

2.3.6.23 获取关节软限位角度

原型	`GetJointSoftLimitDeg (flag = 1)`
描述	获取关节软限位角度
参数	默认参数 `flag`：0-阻塞，1-非阻塞默认 1
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[j1min,j1max,j2min,j2max,j3min,j3max, j4min,j4max,j5min, j5max, j6min,j6max]：轴 1~ 轴 6 关节负限位与正限位，单位 [mm]

2.3.6.23.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetJointSoftLimitDeg(0)  # 获取机器人关节软限位角度
print(ret)
```

2.3.6.24 获取系统时间

原型	`GetSystemClock()`
描述	获取系统时间
参数	无
返回值	-错误码成功-0 失败- errcode - 返回值（调用成功返回）t_ms: 单位 [ms]

2.3.6.24.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetSystemClock()  # 获取控制器系统时间
print(ret)
```

2.3.6.25 获取机器人当前关节配置

原型	`GetRobotCurJointsConfig()`
描述	获取机器人当前关节配置
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）config: 范围 [0~7]

2.3.6.25.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetRobotCurJointsConfig()  # 获取机器人当前关节配置
print(ret)
```

2.3.6.26 获取默认速度

原型	`GetDefaultTransVel()`
描述	获取默认速度
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）vel: 单位 [mm/s]

2.3.6.26.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetDefaultTransVel()  # 获取机器人当前速度
print(ret)
```

2.3.6.27 查询机器人运动是否完成

原型	GetRobotMotionDone()
描述	查询机器人运动是否完成
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）state:0-未完成，1-完成

2.3.6.27.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
ret = robot.GetRobotMotionDone()    #查询机器人运动完成状态
if ret[0] == 0:
    print(ret[1])
else:
    print("the errcode is: ", ret[0])
```

2.3.6.28 查询机器人错误码

原型	GetRobotErrorCode()
描述	查询机器人错误码
参数	无
返回值	错误码成功-0 失败- errcode - 返回值(调用成功返回)[maincode subcode] maincode 主错误码 subcode 子错误码

2.3.6.28.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetRobotErrorCode()
print("查询机器人错误码", ret)
```

2.3.6.29 查询机器人示教管理点位数据

原型	GetRobotTeachingPoint(name)
描述	查询机器人示教管理点位数据
参数	必选参数 name：点位名
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）点位数据[x,y,z,rx,ry,rz,j1,j2,j3,j4,j5,j6,tool, wobj,speed,acc,e1,e2,e3,e4]

2.3.6.29.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetRobotTeachingPoint("11")
print("查询机器人示教管理点位数据错误码", ret)
```

2.3.6.30 获取 SSH 公钥

原型	GetSSHKeygen()
描述	获取 SSH 公钥
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）keygen 公钥

2.3.6.30.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.GetSSHKeygen() #获取SSH
print("获取SSH", ret)
```

2.3.6.31 计算指定路径下文件的 MD5 值

原型	`ComputeFileMD5(file_path)`
描述	计算指定路径下文件的 MD5 值
参数	• 必选参数 `file_path`：文件路径包含文件名，默认 Traj 文件夹路径为:/fruser/traj/, 如/fruser/traj/trajHelix_aima_1.txt
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）keygen 公钥

2.3.6.31.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.ComputeFileMD5("/fruser/201.lua")    #计算指定路径下文件的MD5值
print("计算指定路径下文件的MD5值", ret)
```

2.3.7 机器人轨迹复现

2.3.7.1 设置轨迹记录参数

原型	`SetTPDParam(name, period_ms, type = 1,di_choose = 0, do_choose = 0)`
描述	设置轨迹记录参数
参数	• 必选参数 `name`：轨迹名； • 必选参数 `period_ms`：采样周期，固定值，2ms 或 4ms 或 8ms; • 默认参数 `type`：数据类型，1-关节位置； • 默认参数 `di_choose`：DI 选择,bit0~bit7 对应控制箱 DI0~DI7，bit8~bit9 对应末端 DI0~DI1，0-不选择，1-选择默认 0; • 默认参数 `do_choose`：DO 选择,bit0~bit7 对应控制箱 DO0~DO7，bit8~bit9 对应末端 DO0~DO1，0-不选择，1-选择默认 0
返回值	错误码成功-0 失败- errcode

2.3.7.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
type = 1  # 数据类型，1-关节位置
name = 'tpd2023'  # 轨迹名
period = 4  #采样周期，2ms或4ms或8ms
di_choose = 0 # di输入配置
do_choose = 0 # do输出配置
robot.SetTPDParam(type, name, period, di_choose, do_choose)     #配置TPD参数
```

2.3.7.2 开始轨迹记录

原型	`SetTPDStart(name, period_ms, type = 1,di_choose = 0, do_choose = 0)`
描述	开始轨迹记录
参数	• 必选参数 `name`：轨迹名； • 必选参数 `period_ms`：采样周期，固定值，2ms 或 4ms 或 8ms； • 默认参数 `type`：数数据类型，1-关节位置默认 1； • 默认参数 `di_choose`：DI 选择,bit0~bit7 对应控制箱 DI0~DI7，bit8~bit9 对应末端 DI0~DI1，0-不选择，1-选择默认 0； • 默认参数 `do_choose`：DO 选择,bit0~bit7 对应控制箱 DO0~DO7，bit8~bit9 对应末端 DO0~DO1，0-不选择，1-选择默认 0
返回值	错误码成功-0 失败- errcode

2.3.7.3 停止轨迹记录

原型	`SetWebTPDStop()`
描述	停止轨迹记录
参数	无
返回值	错误码成功-0 失败- errcode

2.3.7.3.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
type = 1  # 数据类型，1-关节位置
name = 'tpd2023'  # 轨迹名
period = 4  #采样周期，2ms或4ms或8ms
di_choose = 0 # di输入配置
do_choose = 0 # do输出配置
robot.SetTPDParam(type, name, period, di_choose, do_choose)     #配置TPD参数
robot.Mode(1)  # 机器人切入手动模式
time.sleep(1)
robot.DragTeachSwitch(1)  #机器人切入拖动示教模式
robot.SetTPDStart(type, name, period, di_choose, do_choose)    # 开始记录示教轨迹
```

(续下页)

(接上页)

```
time.sleep(30)
robot.SetWebTPDStop()  # 停止记录示教轨迹
robot.DragTeachSwitch(0)  #机器人切入非拖动示教模式
```

2.3.7.4 删除轨迹记录

原型	`SetTPDDelete(name)`
描述	删除轨迹记录
参数	• 必选参数 `name`: 轨迹名
返回值	错误码成功-0 失败- errcode

2.3.7.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetTPDDelete('tpd2023')    # 删除TPD轨迹
```

2.3.7.5 轨迹预加载

原型	`LoadTPD(name)`
描述	轨迹预加载
参数	• 必选参数 `name`: 轨迹名
返回值	错误码成功-0 失败- errcode

2.3.7.6 获取轨迹起始位姿

原型	`GetTPDStartPose(name)`
描述	获取轨迹起始位姿
参数	• 必选参数 `name`: 轨迹名
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）desc_pose [x,y,z,rx,ry,rz]

2.3.7.7 轨迹复现

原型	`MoveTPD(name,blend,ovl)`
描述	轨迹复现
参数	• 必选参数 `name`: 轨迹名 • 必选参数 `blend`：是否平滑，0-不平滑，1-平滑 • 必选参数 `ovl`：速度缩放因子，范围 [0~100]
返回值	错误码成功-0 失败- errcode

2.3.7.7.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
P1=[-378.9,-340.3,107.2,179.4,-1.3,125.0]
name = 'tpd2023'    #轨迹名
blend = 1    #是否平滑，1-平滑，0-不平滑
ovl = 100.0    #速度缩放
robot.LoadTPD(name)   #轨迹预加载
robot.MoveCart(P1,1,0,100.0,100.0,100.0,-1.0,-1)        #运动到起始点
robot.MoveTPD(name, blend, ovl)   #轨迹复现
```

2.3.7.8 轨迹预处理

原型	`LoadTrajectoryJ(name,ovl,opt = 1)`
描述	轨迹预处理
参数	• 必选参数 `name`: 轨迹名, 如：/fruser/traj/trajHelix_aima_1.txt; • 必选参数 `ovl`：速度缩放百分比，范围 [0~100]; • 默认参数 `opt`：1-控制点，默认为 1
返回值	错误码成功-0 失败- errcode

2.3.7.9 轨迹复现

原型	`MoveTrajectoryJ()`
描述	轨迹复现
参数	无
返回值	错误码成功-0 失败- errcode

2.3.7.10 获取轨迹起始位姿

原型	`GetTrajectoryStartPose(name)`
描述	获取轨迹起始位姿
参数	必选参数 `name`: 轨迹名
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）desc_pose [x,y,z,rx,ry,rz]

2.3.7.11 获取轨迹点编号

原型	`GetTrajectoryPointNum()`
描述	获取轨迹点编号
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）pnum

2.3.7.12 设置轨迹运行中的速度

原型	`SetTrajectoryJSpeed(ovl)`
描述	设置轨迹运行中的速度
参数	必选参数 `ovl`: 速度缩放百分比，范围 [0~100]
返回值	错误码成功-0 失败- errcode

2.3.7.13 设置轨迹运行中的力和扭矩

原型	`SetTrajectoryJForceTorque(ft)`
描述	设置轨迹运行中的力和扭矩
参数	必选参数 `ft [fx,fy,fz,tx,ty,tz]`: 单位 N 和 Nm
返回值	错误码成功-0 失败- errcode

2.3.7.14 设置轨迹运行中的沿 x 方向的力

原型	`SetTrajectoryJForceFx(fx)`
描述	设置轨迹运行中的沿 x 方向的力
参数	必选参数 `ft`: 沿 x 方向的力，单位 N
返回值	错误码成功-0 失败- errcode

2.3.7.15 设置轨迹运行中的沿 y 方向的力

原型	`SetTrajectoryJForceFx(fy)`
描述	设置轨迹运行中的沿 y 方向的力
参数	必选参数 `fy`: 沿 y 方向的力，单位 N
返回值	错误码成功-0 失败- errcode

2.3.7.16 设置轨迹运行中的沿 z 方向的力

原型	`SetTrajectoryJForceFx(fz)`
描述	设置轨迹运行中的沿 z 方向的力
参数	必选参数 `fz`: 沿 z 方向的力，单位 N
返回值	错误码成功-0 失败- errcode

2.3.7.17 设置轨迹运行中的绕 x 轴的扭矩

原型	`SetTrajectoryJTorqueTx(tx)`
描述	设置轨迹运行中的绕 x 轴的扭矩
参数	必选参数 `tx`: 绕 x 轴的扭矩，单位 Nm
返回值	错误码成功-0 失败- errcode

2.3.7.18 设置轨迹运行中的绕 y 轴的扭矩

原型	`SetTrajectoryJTorqueTx(ty)`
描述	设置轨迹运行中的绕 y 轴的扭矩
参数	必选参数 `ty`: 绕 y 轴的扭矩，单位 Nm
返回值	错误码成功-0 失败- errcode

2.3.7.19 设置轨迹运行中的绕 z 轴的扭矩

原型	`SetTrajectoryJTorqueTx(tz)`
描述	设置轨迹运行中的绕 z 轴的扭矩
参数	必选参数 `tz`: 绕 z 轴的扭矩，单位 Nm
返回值	错误码成功-0 失败- errcode

2.3.8 机器人 WebAPP 程序使用

2.3.8.1 设置开机自动加载默认的作业程序

原型	`LoadDefaultProgConfig(flag,program_name)`
描述	设置开机自动加载默认的作业程序
参数	• 必选参数 `flag`：1-开机自动加载默认程序，0-不自动加载默认程序 • 必选参数 `program_name`：作业程序名及路径，如“/fruser/movej.lua”，其中“/fruser/”为固定路径
返回值	错误码成功-0 失败- errcode

2.3.8.1.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
flag = 1
ret = robot.LoadDefaultProgConfig(flag,'/fruser/testPTP.lua')     #␣
↪设置开机自动加载默认的作业程序
print("Set up and automatically load the default operating program ", ret)
```

2.3.8.2 加载指定的作业程序

原型	`ProgramLoad(program_name)`
描述	加载指定的作业程序
参数	• 必选参数 `program_name`：作业程序名及路径，如“/fruser/movej.lua”，其中“/fruser/”为固定路径
返回值	错误码成功-0 失败- errcode

2.3.8.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#机器人webapp程序使用接口
robot.Mode(0)    #机器人切入自动运行模式
robot.ProgramLoad('/fruser/testPTP.lua')    #加载要执行的机器人程序，testPTP.
↪lua程序需要先在webapp上编写好
```

2.3.8.3 获取当前机器人作业程序的执行行号

原型	`GetCurrentLine()`
描述	获取当前机器人作业程序的执行行号
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）line_num

2.3.8.4 运行当前加载的作业程序

原型	`ProgramRun()`
描述	运行当前加载的作业程序
参数	无
返回值	错误码成功-0 失败- errcode

2.3.8.5 暂停当前运行的作业程序

原型	`ProgramPause()`
描述	暂停当前运行的作业程序
参数	无
返回值	错误码成功-0 失败- errcode

2.3.8.6 恢复当前暂停的作业程序

原型	`ProgramResume()`
描述	恢复当前暂停的作业程序
参数	无
返回值	错误码成功-0 失败- errcode

2.3.8.7 终止当前运行的作业程序

原型	`ProgramStop()`
描述	终止当前运行的作业程序
参数	无
返回值	错误码成功-0 失败- errcode

2.3.8.8 获取机器人作业程序执行状态

原型	`GetProgramState()`
描述	获取机器人作业程序执行状态
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）state:1-程序停止或无程序运行，2-程序运行中，3-程序暂停

2.3.8.9 获取已加载的作业程序名

原型	GetLoadedProgram()
描述	获取已加载的作业程序名
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）program_name

2.3.8.9.1 代码示例

```
import frrpc
import time
import _thread
def print_program_state(name,rb):
    while(1):
        pstate = robot.GetProgramState()     #查询程序运行状态,1-程序停止或无程序运行，2-程序运行中，3-程序暂停
        linenum = robot.GetCurrentLine()     #查询当前作业程序执行的行号
        name = robot.GetLoadedProgram()      #查询已加载的作业程序名
        print("the robot program state is:",pstate[1])
        print("the robot program line number is:",linenum[1])
        print("the robot program name is:",name[1])
        time.sleep(1)
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#机器人webapp程序使用接口
robot.Mode(0)    #机器人切入自动运行模式
robot.ProgramLoad('/fruser/testPTP.lua')    #加载要执行的机器人程序，testPTP.lua程序需要先在webapp上编写好
robot.ProgramRun()      #执行机器人程序
_thread.start_new_thread(print_program_state,("print_state",robot))
time.sleep(5)          #休息10s
robot.ProgramPause()    #暂停正在执行的机器人程序
time.sleep(5)
robot.ProgramResume()   #恢复暂停执行的机器人程序
time.sleep(5)
robot.ProgramStop()     #停止正在执行的机器人程序
time.sleep(2)
```

2.3.9 机器人外设

2.3.9.1 获取夹爪配置

原型	`GetGripperConfig()`
描述	获取夹爪配置
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[number,company,device,softversion] number 夹爪编号范围 [1]; company 夹爪厂商，1-Robotiq，2-慧灵，3-天机，4-大寰，5-知行; device 设备号，Robotiq(0-2F-85 系列)，慧灵 (0-NK 系列,1-Z-EFG-100)，天机 (0-TEG-110)，大寰 (0-PGI-140)，知行 (0-CTPM2F20); softvesion 软件版本号，暂不使用，默认为 0 ;

2.3.9.2 激活夹爪

原型	`ActGripper(index,action)`
描述	激活夹爪
参数	• 必选参数 `index`: 夹爪编号; • 必选参数 `action`:0-复位，1-激活
返回值	错误码成功-0 失败- errcode

2.3.9.3 控制夹爪

原型	`MoveGripper(index,pos,speed,force,maxtime,block)`
描述	控制夹爪
参数	• 必选参数 `index`: 夹爪编号; • 必选参数 `pos`: 位置百分比，范围 [0~100]; • 必选参数 `speed`: 速度百分比，范围 [0~100]; • 必选参数 `force`: 力矩百分比，范围 [0~100]; • 必选参数 `maxtime`: 最大等待时间，范围 [0~30000]，单位 [ms]; • 必选参数 `block`:0-阻塞，1-非阻塞。
返回值	错误码成功-0 失败- errcode

2.3.9.4 获取夹爪运动状态

原型	`GetGripperMotionDone()`
描述	获取夹爪运动状态
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[fault,status] fault:0-无错误，1-有错误 status:0-运动未完成，1-运动完成

2.3.9.5 配置夹爪

原型	`SetGripperConfig(company,device,softversion,bus)`
描述	配置夹爪
参数	• 必选参数 `company`：夹爪厂商，1-Robotiq，2-慧灵，3-天机，4-大寰，5-知行； • 必选参数 `device`：设备号，Robotiq(0-2F-85 系列)，慧灵 (0-NK 系列,1-Z-EFG-100)，天机 (0-TEG-110)，大寰 (0-PGI-140)，知行 (0-CTPM2F20) • 默认参数 `softversion`：软件版本号，暂不使用，默认为 0； • 默认参数 `bus`：设备挂载末端总线位置，暂不使用，默认为 0；
返回值	错误码成功-0 失败- errcode

2.3.9.5.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.SetGripperConfig(4,0,0,1)  # 配置夹爪
time.sleep(1)
config = robot.GetGripperConfig()  # 获取夹爪配置
print(config)
robot.ActGripper(1,0)   # 夹爪复位
time.sleep(1)
robot.ActGripper(1,1)   # 夹爪激活
time.sleep(2)
robot.MoveGripper(1,100,48,46,30000,0)   # 夹爪运动
time.sleep(3)
robot.MoveGripper(1,0,50,0,30000,0)
```

(续下页)

(接上页)

```
ret = robot.GetGripperMotionDone()     # 查询夹爪运动完成状态
print(ret)
```

2.3.9.6 计算预抓取点-视觉

原型	`ComputePrePick(desc_pos, zlength, zangle)`
描述	计算预抓取点-视觉
参数	• 必选参数 `desc_pos`：夹抓取点笛卡尔位姿; • 必选参数 `zlength`：z 轴偏移量; • 必选参数 `zangle`：绕 z 轴旋转偏移量
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）pre_pos 预抓取点笛卡尔位姿

2.3.9.7 计算撤退点-视觉

原型	`ComputePostPick(desc_pos, zlength, zangle)`
描述	计算撤退点-视觉
参数	• 必选参数 `desc_pos`：抓取点笛卡尔位姿; • 必选参数 `zlength`：z 轴偏移量; • 必选参数 `zangle`：绕 z 轴旋转偏移量
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）post_pos 撤退点笛卡尔位姿

2.3.10 机器人力控

2.3.10.1 获取力传感器配置

原型	FT_GetConfig()
描述	获取力传感器配置
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）[number,company,device,softversion,bus] - number 传感器编号; - company 力传感器厂商，17-坤维科技，19-航天十一院，20-ATI 传感器，21-中科米点，22-伟航敏芯; - device 设备号，坤维 (0-KWR75B)，航天十一院 (0-MCS6A-200-4)，ATI(0-AXIA80-M8)，中科米点 (0-MST2010)，伟航敏芯 (0-WHC6L-YB10A); - softvesion 软件版本号，暂不使用，默认为 0

2.3.10.2 力传感器配置

原型	FT_SetConfig(company,device,softversion=0,bus=0)
描述	力传感器配置
参数	• 必选参数 company：传感器厂商，17-坤维科技，19-航天十一院，20-ATI 传感器，21-中科米点，22-伟航敏芯； • 必选参数 device：设备号，坤维 (0-KWR75B)，航天十一院 (0-MCS6A-200-4)，ATI(0-AXIA80-M8)，中科米点 (0-MST2010)，伟航敏芯 (0-WHC6L-YB-10A)； • 默认参数 softversion：软件版本号，暂不使用，默认为 0； • 默认参数 bus：设备挂载末端总线位置，暂不使用，默认为 0；
返回值	• 成功：[0] • 失败：[errcode]

2.3.10.2.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
company = 17     #传感器厂商，17-坤维科技
device = 0       #传感器设备号
```

(续下页)

(接上页)

```
softversion = 0 #软件版本号
bus = 1         #末端总线位置
robot.FT_SetConfig(company, device, softversion, bus)    #配置力传感器
config = robot.FT_GetConfig() #获取力传感器配置信息，厂商编号下发比反馈大1
print(config)
```

2.3.10.3 力传感器激活

原型	FT_Activate(state)
描述	力传感器激活
参数	• 必选参数 state：0-复位，1-激活
返回值	错误码成功-0 失败- errcode

2.3.10.3.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FT_Activate(0)  #传感器复位
time.sleep(1)
robot.FT_Activate(1)  #传感器激活
time.sleep(1)
```

2.3.10.4 力传感器校零

原型	FT_SetZero(state)
描述	力传感器校零
参数	• 必选参数 state：0-去除零点，1-零点矫正
返回值	错误码成功-0 失败- errcode

2.3.10.4.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FT_SetZero(0)    #传感器去除零点
time.sleep(1)
robot.FT_SetZero(1)    #传感器零点矫正,注意此时末端不能安装工具，只有力传感器
time.sleep(1)
```

2.3.10.5 设置力传感器参考坐标系

原型	FT_SetRCS(ref)
描述	设置力传感器参考坐标系
参数	• 必选参数 ref：0-工具坐标系，1-基坐标系
返回值	错误码成功-0 失败- errcode

2.3.10.5.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
robot.FT_SetRCS(0)     #设置参考坐标系为工具坐标系，0-工具坐标系，1-基坐标系
time.sleep(1)
```

2.3.10.6 负载重量辨识计算

原型	FT_PdIdenCompute()
描述	负载重量辨识计算
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）weight-负载重量，单位 kg

2.3.10.7 负载重量辨识记录

原型	FT_PdIdenRecord(tool_id)
描述	负载重量辨识记录
参数	• 必选参数 tool_id：传感器坐标系编号，范围 [0~14]
返回值	错误码成功-0 失败- errcode

2.3.10.7.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#负载辨识，此时末端安装要辨识的工具，工具安装在力传感器下方,末端竖直向下
robot.FT_SetRCS(0)    #设置参考坐标系为工具坐标系，0-工具坐标系，1-基坐标系
time.sleep(1)
tool_id = 10  #传感器坐标系编号
tool_coord = [0.0,0.0,35.0,0.0,0.0,0.0]    # 传感器相对末端法兰位姿
tool_type = 1  # 0-工具，1-传感器
tool_install = 0 # 0-安装末端，1-机器人外部
robot.SetToolCoord(tool_id,tool_coord,tool_type,tool_install)
 ↪#设置传感器坐标系，传感器相对末端法兰位姿
time.sleep(1)
robot.FT_PdIdenRecord(tool_id)    #记录辨识数据
time.sleep(1)
weight = robot.FT_PdIdenCompute()  #计算负载重量，单位kg
print(weight)
```

2.3.10.8 负载质心辨识计算

原型	FT_PdCogIdenCompute ()
描述	负载质心辨识计算
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）cog=[cogx,cogy,cogz]，负载质心，单位 mm

2.3.10.9 负载质心辨识记录

原型	FT_PdCogIdenRecord(tool_id,index)
描述	负载质心辨识记录
参数	• 必选参数 tool_id：传感器坐标系编号，范围 [0~14]; • 必选参数 index：点编号 [1~3]
返回值	错误码成功-0 失败- errcode

2.3.10.9.1 代码示例

```
import frrpc
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#负载质心辨识，机器人需要示教三个不同的姿态，然后记录辨识数据，最后计算负载质心
P1=[-160.619,-586.138,384.988,-170.166,-44.782,169.295]
robot.MoveCart(P1,9,0,100.0,100.0,100.0,-1.0,-1)          #关节空间点到点运动
time.sleep(1)
robot.FT_PdCogIdenRecord(tool_id,1)                                  #记录辨识数据
time.sleep(1)
P2=[-87.615,-606.209,556.119,-102.495,10.118,178.985]
robot.MoveCart(P2,9,0,100.0,100.0,100.0,-1.0,-1)
time.sleep(1)
robot.FT_PdCogIdenRecord(tool_id,2)
time.sleep(1)
P3=[41.479,-557.243,484.407,-125.174,46.995,-132.165]
robot.MoveCart(P3,9,0,100.0,100.0,100.0,-1.0,-1)
time.sleep(1)
robot.FT_PdCogIdenRecord(tool_id,3)
time.sleep(1)
cog = robot.FT_PdCogIdenCompute()    # 计算辨识的负载质心
print(cog)
```

2.3.10.10 获取参考坐标系下力/扭矩数据

原型	FT_GetForceTorqueRCS()
描述	获取参考坐标系下力/扭矩数据
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）data=[fx,fy,fz,tx,ty,tz]

2.3.10.10.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
rcs = robot.FT_GetForceTorqueRCS()  #查询传感器坐标系下数据
print(rcs)
```

2.3.10.11 获取力传感器原始力/扭矩数据

原型	FT_GetForceTorqueOrigin()
描述	获取力传感器原始力/扭矩数据
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）data=[fx,fy,fz,tx,ty,tz]

2.3.10.11.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
origin = robot.FT_GetForceTorqueOrigin()    #查询传感器原始数据
print(origin)
```

2.3.10.12 碰撞守护

原型	FT_Guard(flag,sensor_num,select,force_torque,max_threshold,min_threshold)
描述	碰撞守护
参数	• 必选参数 flag：0-关闭碰撞守护，1-开启碰撞守护； • 必选参数 sensor_num：力传感器编号； • s 必选参数 elect：六个自由度是否检测碰撞 [fx,fy,fz,mx,my,mz]，0-不生效，1-生效； • 必选参数 force_torque：碰撞检测力/力矩，单位 N 或 Nm； • 必选参数 max_threshold：最大阈值； • 必选参数 min_threshold：最小阈值； • 力/力矩检测范围:(force_torque-min_threshold,force_torque+max_threshold)
返回值	错误码成功-0 失败- errcode

2.3.10.12.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
actFlag = 1   #开启标志，0-关闭碰撞守护，1-开启碰撞守护
sensor_num = 1  #力传感器编号
is_select = [1,1,1,1,1,1]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,0.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
max_threshold = [10.0,10.0,10.0,10.0,10.0,10.0]  #最大阈值
min_threshold = [5.0,5.0,5.0,5.0,5.0,5.0]   #最小阈值
P1=[-160.619,-586.138,384.988,-170.166,-44.782,169.295]
P2=[-87.615,-606.209,556.119,-102.495,10.118,178.985]
P3=[41.479,-557.243,484.407,-125.174,46.995,-132.165]
robot.FT_Guard(actFlag, sensor_num, is_select, force_torque, max_threshold, min_
↪threshold)    #开启碰撞守护
robot.MoveCart(P1,9,0,100.0,100.0,100.0,-1.0,-1)          #关节空间点到点运动
robot.MoveCart(P2,9,0,100.0,100.0,100.0,-1.0,-1)
robot.MoveCart(P3,9,0,100.0,100.0,100.0,-1.0,-1)
actFlag = 0
robot.FT_Guard(actFlag, sensor_num, is_select, force_torque, max_threshold, min_
↪threshold)    #关闭碰撞守护
```

2.3.10.13 恒力控制

原型	FT_Control(flag,sensor_num,select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
描述	恒力控制
参数	• 必选参数 flag：恒力控制开启标志，0-关，1-开； • 必选参数 sensor_num：力传感器编号； • 必选参数 select：六个自由度是否检测 [fx,fy,fz,mx,my,mz]，0-不生效，1-生效； • 必选参数 force_torque：检测力/力矩，单位 N 或 Nm； • 必选参数 gain：[f_p,f_i,f_d,m_p,m_i,m_d], 力 PID 参数，力矩 PID 参数； • 必选参数 adj_sign：自适应启停状态，0-关闭，1-开启； • 必选参数 ILC_sign: ILC 控制启停状态，0-停止，1-训练，2-实操； • 必选参数 max_dis：最大调整距离； • 必选参数 max_ang：最大调整角度；
返回值	• 成功：[0] • 失败：[errcode]

2.3.10.13.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [0,0,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
gain = [0.0005,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 100.0  #最大调整距离
max_ang = 0.0  #最大调整角度
J1=[-68.987,-96.414,-111.45,-61.105,92.884,11.089]
P1=[62.795,-511.979,291.697,-179.545,3.027,-170.039]
eP1=[0.000,0.000,0.000,0.000]
dP1=[0.000,0.000,0.000,0.000,0.000,0.000]
J2=[-107.596,-109.154,-104.735,-56.176,90.739,11.091]
P2=[-294.768,-503.708,233.158,179.799,0.713,151.309]
```

(续下页)

(接上页)

```
eP2=[0.000,0.000,0.000,0.000]
dP2=[0.000,0.000,0.000,0.000,0.000,0.000]
robot.MoveJ(J1,P1,9,0,100.0,180.0,100.0,eP1,-1.0,0,dP1)     #关节空间运动PTP,
↪工具号9，实际测试根据现场数据及工具号使用
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)    #恒力控制
robot.MoveL(J2,P2,9,0,100.0,180.0,20.0,-1.0,eP2,0,0,dP2)    #笛卡尔空间直线运动
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
```

2.3.10.14 螺旋线探索

原型	FT_SpiralSearch(rcs,ft, dr = 0.7,max_t_ms = 60000, max_vel = 5)
描述	螺旋线探索
参数	• 必选参数 rcs：参考坐标系，0-工具坐标系，1-基坐标系 • 必选参数 ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • 默认参数 dr：每圈半径进给量，单位 mm 默认 0.7; • 默认参数 max_t_ms：最大探索时间，单位 ms 默认 60000; • 默认参数 max_vel：线速度最大值，单位 mm/s 默认 5
返回值	• 成功：[0] • 失败：[errcode]

2.3.10.14.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#恒力参数
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [0,0,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
gain = [0.0001,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
```

(续下页)

(接上页)

```
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 100.0  #最大调整距离
max_ang = 5.0  #最大调整角度
#螺旋线探索参数
rcs = 0  #参考坐标系，0-工具坐标系，1-基坐标系
dr = 0.7  #每圈半径进给量，单位mm
fFinish = 1.0 #力或力矩阈值（0~100），单位N或Nm
t = 60000.0 #最大探索时间，单位ms
vmax = 3.0 #线速度最大值，单位mm/s
is_select = [0,0,1,1,1,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
robot.FT_SpiralSearch(rcs,dr,fFinish,t,vmax)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
```

2.3.10.15 旋转插入

原型	FT_RotInsertion(rcs,ft, orn, angVelRot = 3, angleMax = 45, angAccmax = 0, rotorn =1)
描述	旋转插入
参数	• 必选参数 rcs：参考坐标系，0-工具坐标系，1-基坐标系； • 必选参数 ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • 必选参数 orn：力/扭矩方向，1-沿 z 轴方向，2-绕 z 轴方向; • 默认参数 angVelRot：旋转角速度，单位 °/s, 默认 3; • 默认参数 angleMax：最大旋转角度，单位 ° 默认 5; • 默认参数 angAccmax：最大旋转加速度，单位 °/s^2，暂不使用默认 0; • 默认参数 rotorn：旋转方向，1-顺时针，2-逆时针默认 1
返回值	• 成功：[0] • 失败：[errcode]

2.3.10.15.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#恒力参数
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [0,0,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
gain = [0.0001,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 100.0  #最大调整距离
max_ang = 5.0  #最大调整角度
#旋转插入参数
rcs = 0  #参考坐标系，0-工具坐标系，1-基坐标系
angVelRot = 2.0  #旋转角速度，单位°/s
forceInsertion = 1.0 #力或力矩阈值（0~100），单位N或Nm
angleMax= 45 #最大旋转角度，单位°
orn = 1 #力的方向，1-fz,2-mz
angAccmax = 0.0 #最大旋转角加速度，单位°/s^2,暂不使用
rotorn = 1 #旋转方向，1-顺时针，2-逆时针
s_select = [0,0,1,1,1,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
gain = [0.0001,0.0,0.0,0.0,0.0,0.0]  #最大阈值
status = 1
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
robot.FT_RotInsertion(rcs,angVelRot,forceInsertion,angleMax,orn,angAccmax,rotorn)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
```

2.3.10.16 直线插入

原型	FT_LinInsertion(rcs, ft, disMax, linorn, lin_v = 1.0, lin_a = 1.0)
描述	直线插入
参数	• 必选参数 rcs：参考坐标系，0-工具坐标系，1-基坐标系； • 必选参数 ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • 必选参数 disMax：最大插入距离，单位 mm; • 必选参数 linorn：插入方向:0-负方向，1-正方向 • 默认参数 lin_v：直线速度，单位 mm/s 默认 1; • 默认参数 lin_a：直线加速度，单位 mm/s^2，暂不使用默认 0
返回值	• 成功：[0] • 失败：[errcode]

2.3.10.16.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#恒力参数
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [0,0,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [0.0,0.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-
↪min_threshold,force_torque+max_threshold)
gain = [0.0001,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 100.0  #最大调整距离
max_ang = 5.0  #最大调整角度
#直线插入参数
rcs = 0  #参考坐标系，0-工具坐标系，1-基坐标系
force_goal = 20.0  #力或力矩阈值（0~100），单位N或Nm
lin_v = 0.0 #直线速度，单位mm/s
lin_a = 0.0 #直线加速度，单位mm/s^2,暂不使用
disMax = 100.0 #最大插入距离，单位mm
linorn = 1 #插入方向，1-正方向，2-负方向
is_select = [1,1,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
gain = [0.00005,0.0,0.0,0.0,0.0,0.0]  #最大阈值
```

(续下页)

(接上页)

```
force_torque = [0.0,0.0,-30.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-min_threshold,force_torque+max_threshold）
status = 1
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
robot.FT_LinInsertion(rcs,force_goal,lin_v,lin_a,disMax,linorn)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
```

2.3.10.17 计算中间平面位置开始

原型	FT_CalCenterStart()
描述	计算中间平面位置开始
参数	无
返回值	错误码成功-0 失败- errcode

2.3.10.18 计算中间平面位置结束

原型	FT_CalCenterEnd()
描述	计算中间平面位置结束
参数	无
返回值	错误码成功-0 失败- errcode - 返回值（调用成功返回）pos=[x,y,z,rx,ry,rz]

2.3.10.19 表面定位

原型	FT_FindSurface (rcs, dir, axis, disMax, ft, lin_v = 3.0, lin_a = 0.0)
描述	表面定位
参数	• 必选参数 rcs：参考坐标系，0-工具坐标系，1-基坐标系； • 必选参数 dir：移动方向，1-正方向，2-负方向； • 必选参数 axis：移动轴，1-x，2-y，3-z； • 必选参数 disMax：大探索距离，单位 mm; • 必选参数 ft：动作终止力阈值，单位 N； • 默认参数 lin_v：探索直线速度，单位 mm/s 默认 3; • 默认参数 lin_a：探索直线加速度，单位 mm/s^2 默认 0;
返回值	错误码成功-0 失败- errcode

2.3.10.19.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
#恒力控制
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [1,0,0,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [-2.0,0.0,0.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_torque-min_threshold,force_torque+max_threshold)
gain = [0.0002,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 15.0  #最大调整距离
max_ang = 0.0  #最大调整角度
#表面定位参数
rcs = 0 #参考坐标系，0-工具坐标系，1-基坐标系
direction = 1 #移动方向，1-正方向，2-负方向
axis = 1 #移动轴，1-X,2-Y,3-Z
lin_v = 3.0  #探索直线速度，单位mm/s
lin_a = 0.0  #探索直线加速度，单位mm/s^2
disMax = 50.0 #最大探索距离，单位mm
force_goal = 2.0 #动作终止力阈值，单位N
P1=[-230.959,-364.017,226.179,-179.004,0.002,89.999]
robot.MoveCart(P1,9,0,100.0,100.0,100.0,-1.0,-1)        #关节空间点到点运动
#x方向寻找中心
```

(续下页)

(接上页)

```
#第1个表面
robot.FT_CalCenterStart()
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
robot.FT_FindSurface(rcs,direction,axis,lin_v,lin_a,disMax,force_goal)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
robot.MoveCart(P1,9,0,100.0,100.0,100.0,-1.0,-1)        #关节空间点到点运动
robot.WaitMs(1000)
#第2个表面
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
direction = 2 #移动方向，1-正方向，2-负方向
robot.FT_FindSurface(rcs,direction,axis,lin_v,lin_a,disMax,force_goal)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
#计算x方向中心位置
xcenter= robot.FT_CalCenterEnd()
print(xcenter)
xcenter = [xcenter[1],xcenter[2],xcenter[3],xcenter[4],xcenter[5],xcenter[6]]
robot.MoveCart(xcenter,9,0,60.0,50.0,50.0,0.0,-1)
#y方向寻找中心
#第1个表面
robot.FT_CalCenterStart()
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
direction = 1 #移动方向，1-正方向，2-负方向
axis = 2 #移动轴，1-X,2-Y,3-Z
disMax = 150.0 #最大探索距离，单位mm
lin_v = 6.0  #探索直线速度，单位mm/s
robot.FT_FindSurface(rcs,direction,axis,lin_v,lin_a,disMax,force_goal)
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
robot.MoveCart(P1,9,0,100.0,100.0,100.0,-1.0,-1)        #关节空间点到点运动
robot.WaitMs(1000)
#第2个表面
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
direction = 2 #移动方向，1-正方向，2-负方向
robot.FT_FindSurface(rcs,direction,axis,lin_v,lin_a,disMax,force_goal)
```

(续下页)

(接上页)

```
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_dis,max_ang)
#计算y方向中心位置
ycenter=robot.FT_CalCenterEnd()
print(ycenter)
ycenter = [ycenter[1],ycenter[2],ycenter[3],ycenter[4],ycenter[5],ycenter[6]]
robot.MoveCart(ycenter,9,0,60.0,50.0,50.0,-1.0,-1)
```

2.3.10.20 柔顺控制关闭

原型	FT_ComplianceStop()
描述	柔顺控制关闭
参数	无
返回值	错误码成功-0 失败- errcode

2.3.10.21 柔顺控制开启

原型	FT_ComplianceStart(p,force)
描述	柔顺控制开启
参数	• 必选参数 p: 位置调节系数或柔顺系数 • 必选参数 force：柔顺开启力阈值，单位 N
返回值	错误码成功-0 失败- errcode

2.3.10.21.1 代码示例

```
import frrpc
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = frrpc.RPC('192.168.58.2')
J1=[-105.3,-68.0,-127.9,-75.5,90.8,77.8]
P1=[-208.9,-274.5,334.6,178.8,-1.3,86.7]
eP1=[0.000,0.000,0.000,0.000]
dP1=[0.000,0.000,0.000,0.000,0.000,0.000]
J2=[-105.3,-97.9,-101.5,-70.3,90.8,77.8]
P2=[-264.8,-480.5,341.8,179.2,0.3,86.7]
eP2=[0.000,0.000,0.000,0.000]
```

(续下页)

(接上页)

```
dP2=[0.000,0.000,0.000,0.000,0.000,0.000]
#恒力控制参数
status = 1  #恒力控制开启标志，0-关，1-开
sensor_num = 1 #力传感器编号
is_select = [1,1,1,0,0,0]  #六个自由度选择[fx,fy,fz,mx,my,mz]，0-不生效，1-生效
force_torque = [-10.0,-10.0,-10.0,0.0,0.0,0.0]  #碰撞检测力和力矩，检测范围（force_
↪torque-min_threshold,force_torque+max_threshold)
gain = [0.0005,0.0,0.0,0.0,0.0,0.0]  #最大阈值
adj_sign = 0  #自适应启停状态，0-关闭，1-开启
ILC_sign = 0  #ILC控制启停状态，0-停止，1-训练，2-实操
max_dis = 1000.0  #最大调整距离
max_ang = 0.0  #最大调整角度
#柔顺控制
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
p = 0.00005  #位置调节系数或柔顺系数
force = 30.0 #柔顺开启力阈值，单位N
robot.FT_ComplianceStart(p,force)
count = 15  #循环次数
while(count):
    robot.MoveL(J1,P1,9,0,100.0,180.0,100.0,-1.0,eP1,0,1,dP1)    #笛卡尔空间直线运动
    robot.MoveL(J2,P2,9,0,100.0,180.0,100.0,-1.0,eP2,0,0,dP2)
    count = count - 1
robot.FT_ComplianceStop()
status = 0
robot.FT_Control(status,sensor_num,is_select,force_torque,gain,adj_sign,ILC_sign,max_
↪dis,max_ang)
```

2.3.11 传动带

2.3.11.1 传动带启动、停止

原型	ConveyorStartEnd(status)
描述	传动带启动、停止
参数	必选参数 status：状态，1-启动，0-停止
返回值	错误码成功-0 失败- errcode

2.3.11.1.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
#传送带运动，1-运动，0-停止
status = 1
robot.ConveyorStartEnd(status)
#点记录
ret = robot.ConveyorPointIORecord()
print("记录IO检测点",ret)
ret = robot.ConveyorPointARecord()
print("记录A点",ret)
ret = robot.ConveyorRefPointRecord()
print("记录参考点",ret)
ret = robot.ConveyorPointBRecord()
print("记录B点",ret)
```

2.3.11.2 记录 IO 检测点

原型	ConveyorPointIORecord()
描述	记录 IO 检测点
参数	无
返回值	错误码成功-0 失败- errcode

2.3.11.3 记录 A 点

原型	ConveyorPointARecord()
描述	记录 A 点
参数	无
返回值	错误码成功-0 失败- errcode

2.3.11.4 记录参考点

原型	ConveyorRefPointRecord()
描述	记录参考点
参数	无
返回值	错误码成功-0 失败- errcode

2.3.11.4.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接。 成功连接返回机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot. ConveyorRefPointRecord()
print("Convey record reference point ",ret)
```

2.3.11.5 记录 B 点

原型	ConveyorPointBRecord()
描述	记录 B 点
参数	无
返回值	错误码成功-0 失败- errcode

2.3.11.5.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接。 成功连接返回机器人对象
robot = Robot.RPC('192.168.58.2')
ret = robot.ConveyorPointBRecord()
print("Convey record B point ",ret)
```

2.3.11.6 传动带参数配置

原型	ConveyorSetParam(param)
描述	传动带参数配置
参数	必选参数 **param: =[encChannel,resolution,lead,wpAxis,vision,speedRadio]** • encChannel: 编码器通道 1-2 • resolution: 编码器分辨率编码器旋转一圈脉冲个数 • lead: 机械传动比编码器旋转一圈传送带移动距离 • wpAxis: 工件坐标系编号针对跟踪运动功能选择工件坐标系编号，跟踪抓取、TPD 跟踪设为 0 • vision: 是否配视觉 0-不配 1-配, • speedRadio: 速度比针对传送带跟踪抓取速度范围为（1-100）跟踪运动、TPD 跟踪设置为 1
返回值	错误码成功-0 失败- errcode

2.3.11.6.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接。 成功连接返回机器人对象
robot = Robot.RPC('192.168.58.2')
param=[1,10000,200,0,0,20]
ret = robot.ConveyorSetParam(param)
print("Set Conveyor Param",ret)
```

2.3.11.7 传动带抓取点补偿

原型	ConveyorCatchPointComp(cmp)
描述	传动带抓取点补偿
参数	必选参数 cmp：补偿位置 [x,y,z]
返回值	错误码成功-0 失败- errcode

2.3.11.8 传送带工件 IO 检测

原型	ConveyorIODetect(max_t)
描述	传送带工件 IO 检测
参数	必选参数 max_t：最大检测时间，单位 ms
返回值	错误码成功-0 失败- errcode

2.3.11.8.1 代码示例

```
from fairino import Robot
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
#传送带跟踪抓取
while(1):
    robot.MoveL([-333.597, 60.354, 404.341, -179.143, -0.778, 91.275],0,0)
    error =robot.ConveyorIODetect(1000)
    print("传送带工件IO检测错误码",error)
    error =robot.ConveyorGetTrackData(1)
    print("获取物体当前位置错误码",error)
    error =robot.ConveyorTrackStart(1)
    print("传动带跟踪开始错误码",error)
    error =robot.ConveyorTrackMoveL("cvrCatchPoint",0,0,vel = 60.0)
    print("直线运动错误码",error)
    error =robot.MoveGripper(1,55,20,20,30000,0)
    print("夹爪控制错误码",error)
    error =robot.ConveyorTrackMoveL("cvrRaisePoint",0,0,vel = 60.0)
    print("直线运动错误码",error)
    error = robot.ConveyorTrackEnd()
    print("传动带跟踪结束错误码错误码",error)
    error = robot.MoveL([-333.625, -229.039, 404.340, -179.141, -0.778, 91.276], 0, 0,
↪vel =30)
    error = robot.MoveL([-333.564, 332.204, 342.217, -179.145, -0.780, 91.268], 0, 0,
↪vel =30)
    error = robot.MoveGripper(1,100,10,21,30000,0)
    print("夹爪控制错误码",error)
```

2.3.11.9 获取物体当前位置

原型	ConveyorGetTrackData(mode)
描述	获取物体当前位置
参数	必选参数 `mode`：1-跟踪抓取 2-跟踪运动 3-TPD 跟踪
返回值	错误码成功-0 失败- errcode

2.3.11.10 传动带跟踪开始

原型	ConveyorTrackStart(status)
描述	传动带跟踪开始
参数	必选参数 `status`：状态，1-启动，0-停止
返回值	错误码成功-0 失败- errcode

2.3.11.11 传动带跟踪停止

原型	ConveyorTrackEnd()
描述	传动带跟踪停止
参数	无
返回值	错误码成功-0 失败- errcode

2.3.11.12 直线运动

原型	ConveyorTrackMoveL(name,tool,wobj,vel=20,acc=100,ovl=100,blendR=-1.0)
描述	直线运动
参数	• 必选参数 `name cvrCatchPoint 或 cvrRaisePoint` • 必选参数 `tool`: 工具号 • 必选参数 `wobj`: 工件号 • 默认参数 `vel`: 速度默认 20 • 默认参数 `acc`: 加速度默认 100 • 默认参数 `ovl`: 速度缩放因子默认 100 • 默认参数 `blendR`: [-1.0]-运动到位 (阻塞)，[0~1000]-平滑半径 (非阻塞)，单位 [mm] 默认-1.0
返回值	错误码成功-0 失败- errcode

2.3.11.12.1 代码示例

```
from fairino import Robot
import time
# 与机器人控制器建立连接，连接成功返回一个机器人对象
robot = Robot.RPC('192.168.58.2')
#参数配置
param=[1,10000,200,0,0,20]
ret = robot.ConveyorSetParam(param)
print("传送带参数配置错误码",ret)
time.sleep(1)
#抓取点补偿
comp = [0.00, 0.00, 0.00]
ret1 = robot.ConveyorCatchPointComp(comp)
print("传动带抓取点补偿错误码",ret1)
```

2.4 SDK 错误码对照表

errcode	描述	处理方式
-4	xmlrpc 接口执行失败	请联系售后工程
-3	xmlrpc 通讯失败	请检查网络连接以及服务器 IP 地址是否正确
-2	与控制器通讯异常	检查与控制器软硬件连接
-1	其他错误	联系售后工程师查看控制器日志
0	调用成功	
3	接口参数个数不一致	检查接口参数个数
4	接口参数值异常	检查参数值类型或范围
8	轨迹文件打开失败	检查 TPD 轨迹文件是否存在或轨迹名是否正确
9	TPD 文件名发送失败	检查 TPD 轨迹名是否正确
10	TPD 文件内容发送失败	检查 TPD 文件内容是否正确
14	接口执行失败	检查 web 界面是否报故障或状态反馈是否报故障
15	TDP 记录点数超限	重新记录
18	机器人程序正在运行，请先停止	先停止程序，再进行其他操作
25	数据异常，计算失败	重新标定或辨识
28	逆运动学计算结果异常	检查位姿是否合理
29	ServoJ 关节超限	检查关节数据是否在合理范围
30	不可复位故障，请断电重启控制箱	请断电重启控制箱
32	关节超限	切换至拖动模式，将关节移动至软限位范围
34	工件号错误	请检查工件号是否合理

续下页

表 1 – 接上页

errcode	描述	处理方式
36	文件名过长	请缩减文件名长度
37	工具号错误	请检查工具号是否合理
38	奇异位姿，计算失败	请更换位姿
40	速度百分比超限	检查速度百分比是否合理
42	姿态变化过大	插入中间姿态进行过度
59	力/扭矩传感器未激活	激活力传感器
60	力/扭矩传感器参考坐标系未切换至工具	将力传感器坐标系切换至工具
64	未加入指令队列	联系售后工程师查看控制器日志
66	整圆/螺旋线指令中间点 1 错误	检查中间点 1 数据是否正确
67	整圆/螺旋线指令中间点 2 错误	检查中间点 2 数据是否正确
68	整圆/螺旋线指令中间点 3 错误	检查中间点 3 数据是否正确
69	圆弧指令中间点错误	检查中间点数据是否正确
70	圆弧指令目标点错误	检查目标点数据是否正确
73	夹爪运动报错	检查夹爪通信状态是否正常
74	直线指令点错误	检查点位数据是否正确
75	通道错误	检查 IO 编号是否在范围内
76	等待超时	检查 IO 信号是否输入或接线是否正确
82	TPD 指令点错误	重新记录示教轨迹
83	TPD 指令工具与当前工具不符	更改为 TPD 示教时所用的工具坐标系
94	样条指令点错误	检查点位数据是否正确
95	样条参数错误	检查样条参数是否合理
108	螺旋线指令起点错误	检查起点数据是否正确
112	给定位姿无法到达	检查目标位姿是否合理

CHAPTER 3

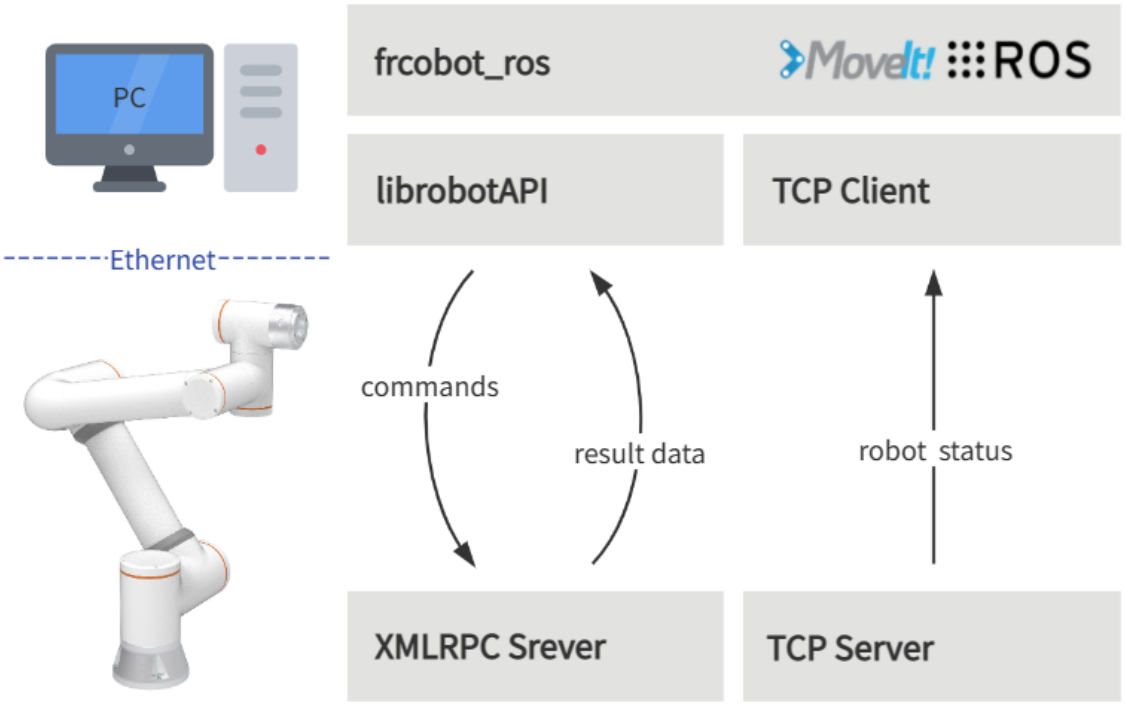
frcobot_ros

3.1 概述

frcobot_ros 简要架构如下图所示，协作机器人端提供了 XMLRPC 服务器和 TCP 服务器。

- XMLRPC 服务器主要提供机器人指令 API 完成机器人运动和状态值获取功能
- 状态反馈的 TCP 服务器提供了机器人状态的实时反馈，反馈周期 8ms。

用户 PC 端中已安装了 ROS 和 Moveit!，编译完成 frcobot_ros。在 frcobot_ros 中每个功能包都包含了机器人 API 的 lib 库，以及在 frcobot_hw 建立与机器人状态反馈服务器通讯的 TCP 客户端，获取机器人状态反馈数据。



3.2 安装

本章介绍如何构建 frcobot_ros 以及所需的安装环境。

3.2.1 环境要求

frcobot_ros 推荐环境如下：

备注:

- Ubuntu 18.04 LTS Bionic Beaver 和 ROS Melodic Morenia
- Ubuntu 20.04 LTS Focal Fossa 和 ROS Noetic Ninjemys

以下说明适用于 Ubuntu 20.04 LTS 系统和 ROS Noetic Ninjemys。如果使用的是 Melodic，则将下发命令行中的 `noetic` 替换成 `melodic`.

3.2.2 ROS 安装要求

在安装好 Ubuntu 系统后，安装和配置好 ROS Noetic 环境。

在配置好 ROS Noetic 后，安装如下所需环境：

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt-get install -y \
    ros-noetic-rosparam-shortcuts \
    ros-noetic-ros-control \
    ros-noetic-ros-controllers \
```

```
cd ~/catkin_ws/src
git clone https://gitee.com/fair-innovation/frcobot_ros.git
```

构建 frcobot_ros 包

```
cd ~/catkin_ws
catkin_make
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

如果出现报错请检查 ROS 安装要求中的包是否都已安装成功，编译完成后，将 lib 库拷贝到 ROS 的 lib 环境下 (路径为：/opt/ros/noetic/lib)，以便程序可以正常运行。

```
# 此处catkin_ws默认路径为“~”，如有不同，将“~”改为实际路径即可
sudo cp ~/catkin_ws/src/frcobot_ros/frcobot_hw/lib/* /opt/ros/noetic/lib
```

3.3 快速开始

3.3.1 frcobot_hw

frcobot_hw 主要提供了和协作机器人通讯的基本功能。

备注:

- 包含协作机器人状态反馈 msg
- 提供控制协作机器人的指令 demo
- 提供协作机器人状态反馈节点和 Topic
- 可通过 launch 文件快速启动状态节点和指令 demo

---

frcobot_hw.launch 内容如下：

```
<launch>

    <!-- params -->
    <param name="robot_ip" type="string" value="192.168.58.2"/>
    <param name="robot_port" type="int" value="8083"/>

    <!-- frcobot status node -->
    <node pkg="frcobot_hw" type="frcobot_status_node" name="frcobot_status_node"␣
↪output="screen" />
```

(续下页)

(接上页)

```
9
10      <!-- frcobot control demo -->
11      <node pkg="frcobot_hw" type="frcobot_cmd_demo" name="frcobot_cmd_demo" output=
    ↪"screen" />
12
13  </launch>
```

重要:

- robot_ip 和 robot_port 需要注意与被控制的协作机器人 IP 和端口一致
- 出厂机器人默认 IP 为 192.168.58.2，用户状态反馈端口为 8083

通过以下指令可快速启动机器人状态反馈节点和指令 demo 功能。

```
1 roslaunch frcobot_hw frcobot_hw.launch
```

新开一个 terminal，通过以下指令可打印并查看实时的状态反馈数据。

```
1 rostopic ehco /frcobot_status
```

CHAPTER 4

frcobot_ros2

4.1 前言

frcobot_ros2 为法奥协作机器人基于 ROS2 开发的 API 接口，旨在针对入门级用户更便捷的使用法奥 SDK。通过参数配置文件对默认参数的配置，即可适应不同的客户要求。

4.2 fr_ros2

本章节说明 APP 运行环境如何配置。

4.2.1 基本环境安装

推荐在 Ubuntu22.04LTS(Jammy) 上使用，系统安装完毕后，就可以安装 ROS2，推荐用 ros2-humble，ROS2 的安装可以参考教程：https://docs.ros.org/en/humble/index.html。

4.2.2 编译及构建 fr_ros2

1. 创建 colcon 工作区 fr_ros2 有两个功能包组成，一个是自定义数据结构的功能包 frhal_msgs，另外一个是程序主体 fr_ros2 功能包。在安装好基本环境后，先创建一个 colcon 工作区，比如:

```
cd ~/
mkdir -p ros2_ws/src
```

2. 编译功能包将安装包的代码拷贝至 ros2_ws/src 目录下，在 ros2_ws 目录下运行如下命令：

```
colcon build --packages-select frhal_msgs
```

等待上一条命令完成编译后

```
colcon build --packages-select fr_ros2
```

4.3 快速开始

4.3.1 启动流程

在 Ubuntu 下打开命令行，输入：

```
cd ros2_ws
source install/setup.bash
ros2 run fr_ros2 ros2_cmd_server
```

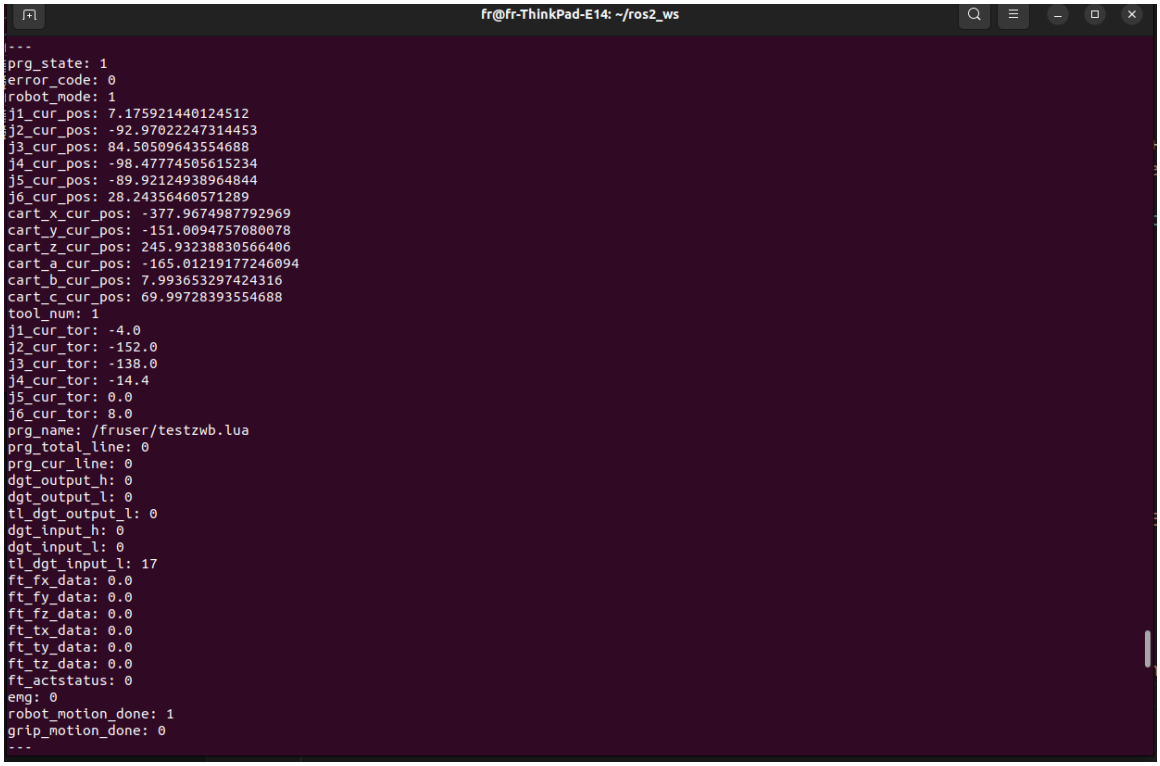
4.3.2 查看机械臂状态反馈流程

机械臂的状态反馈是通过 topic 发布的，用户可以通过 ros2 自带的命令观察到状态数据刷新，也可以编写程序获取该数据，下面展示如何通过 ros2 命令观察机械臂状态数据。

在 ubuntu 下打开命令行，输入：

```
cd ros2_ws
source install/setup.bash
ros2 topic echo /nonrt_state_data
```

可以看到命令行窗口中不断刷新的状态数据，如下图所示。

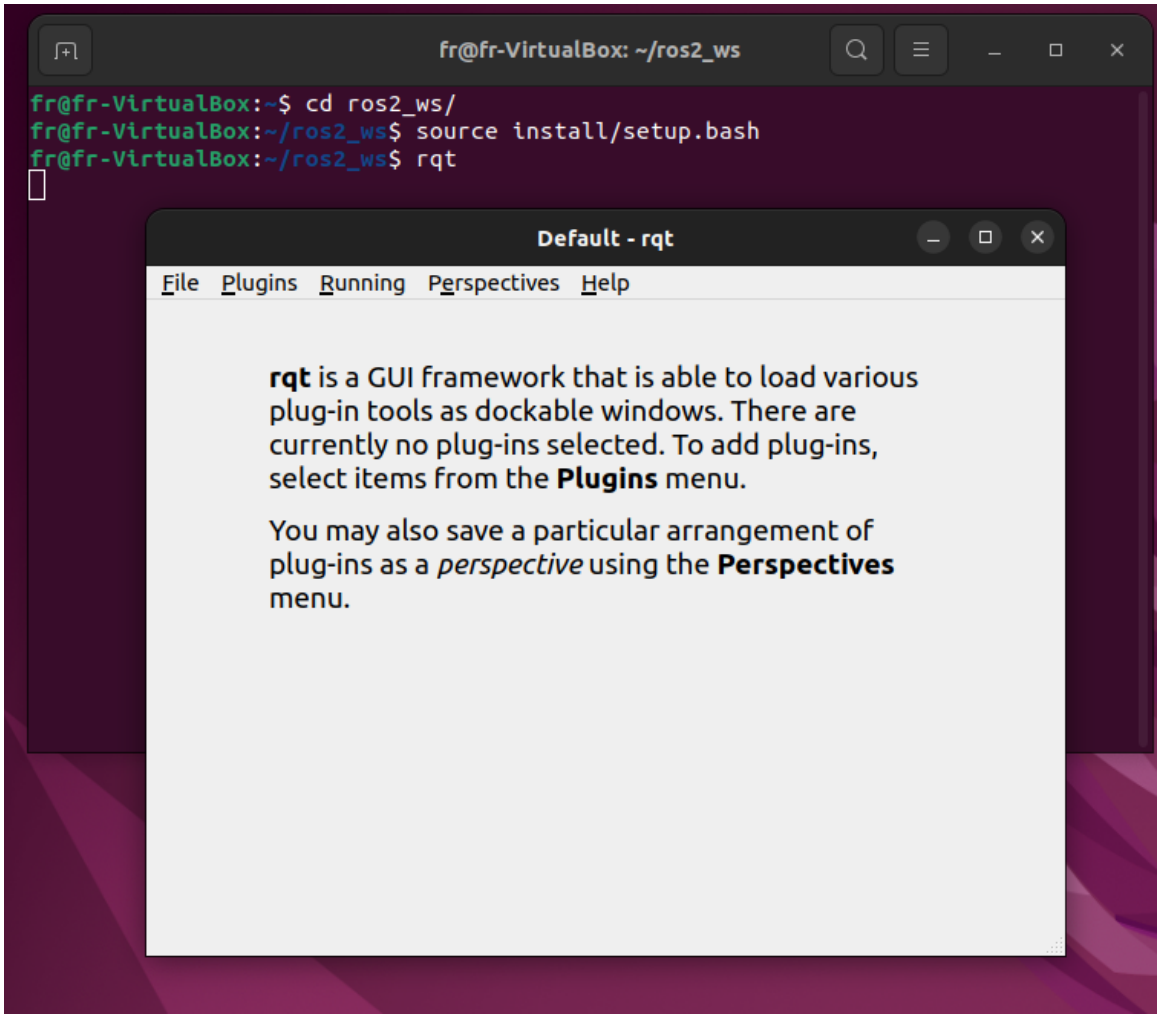


4.3.3 下发指令流程

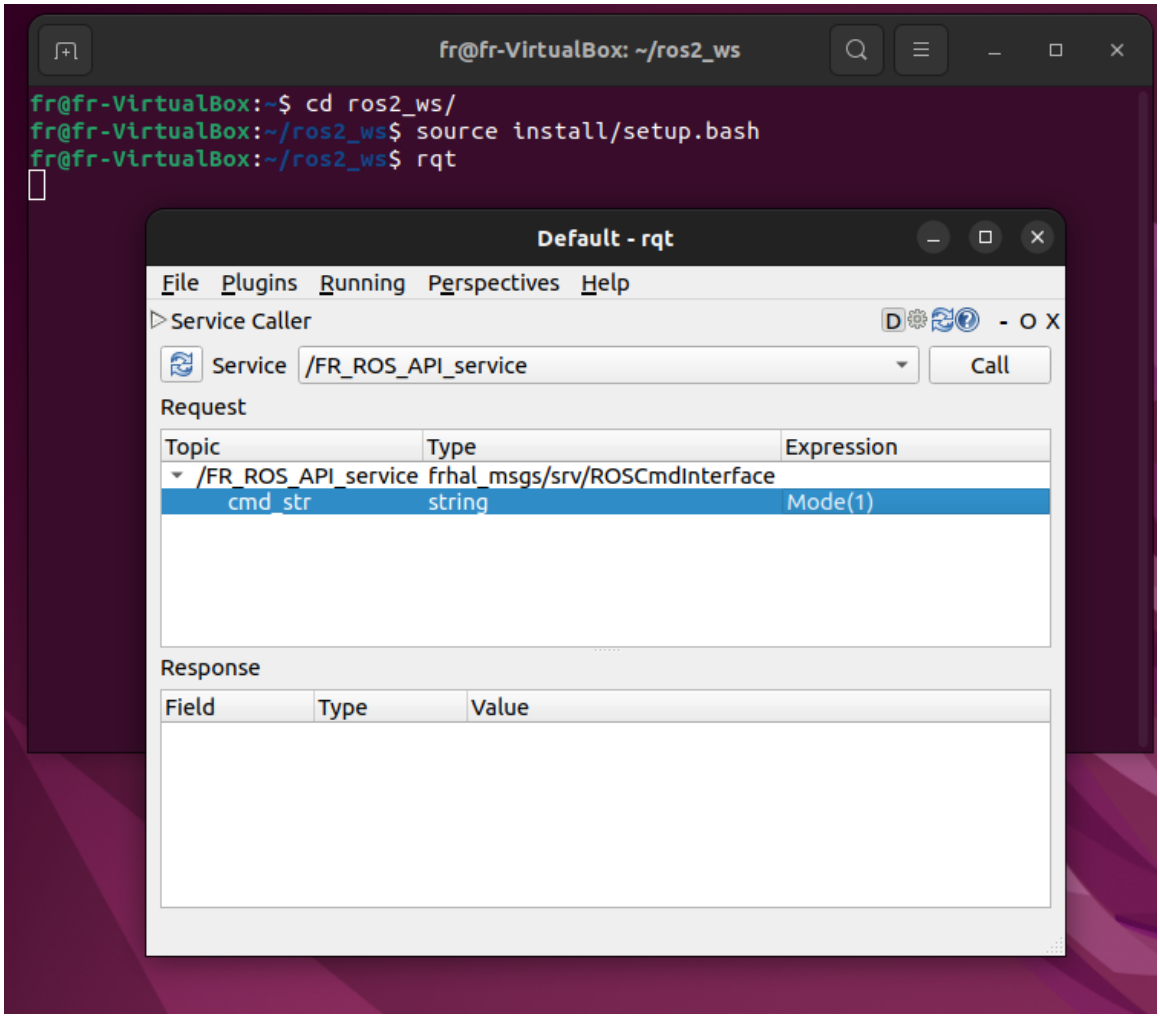
在 ubuntu 下打开命令行，输入：

```
cd ros2_ws
source install/setup.bash
rqt
```

以上命令执行完毕后，会调出一个 rqt GUI 界面，如下图所示。



在 GUI 界面选择 plugins->serivce->serivce caller，调出如下界面，选择/FR_ROS_API_service 这项，在界面 expression 中输入指令字符串点击 call 即可看到下方对话框中跳出回复信息。



重要:

- 输入字符串规则说明:

程序内部对输入的字符串形式进行了筛选，函数输入的格式必须是 [函数名]() 这样的形式，且圆括号的参数字符串必须是由字母，数字，逗号还有负号组成，出现其他字符或者空格均会报错。

- 指令反馈值说明:

除了 GET 指令会反馈一串字符串，其余的函数反馈值都是 int 型，一般 0 为出现错误，1 为正确执行，如果出现其他的值那么参考 xmlrpc SDK 中定义的错误代码对应的错误。

4.3.4 修改参数流程

由于简化 SDK 是改进自原生的 SDK 接口，能够简化是因为赋予了一些参数默认值，而在实际使用过程中也会遇到默认参数无法满足要求的情况，这个时候可以通过修改对应默认参数的数值，然后加载到节点中。

源代码文件中存在一个 fr_ros2_para.yaml 参数文件，文件中的参数为预先设置的默认参数，用于简化指令输入参数，可以根据自己的具体需要修改其中的参数，然后使用命令动态修改参数: ros2 param load FR_ROS_API_nod ~/ros2_ws/src/fr_ros2/fr_ros2_para.yaml。

4.4 API 说明

```
/*
函数功能描述:存储一个关节点位信息
id - 存储点位id号,从1开始,注意该id与CARTPoint的点位id号各自独立
double j1-j6 - 6个关节位置,单位是度
*/
int JNTPoint(int id, double j1, double j2, double j3, double j4, double j5, double j6)
// 例子
JNTPoint(1,10,11,12,13,14,15)

/*
函数功能描述:存储一个笛卡尔点位信息
id - 存储点位id号,从1开始,注意该id与JNTPoint的点位id号各自独立
double x,y,z,rx,ry,yz - 笛卡尔点位信息,位置单位是mm,角度单位是度
*/
int CARTPoint(int id, double x,y,z,rx,ry,rz)//存储一个笛卡尔空间点位
// 例子
CARTPoint(1,100,110,200,0,0,0)

/*
函数功能描述:获取指定序号点的关节或者笛卡尔位置信息
string name - 'JNT'或者'CART',JNT代表获取关节点位信息,'CART'代表获取笛卡尔点位信息
int id - 点位id,从1开始
*/
string GET(string name, int id)//获取对应id序号点位的内容,name可以输入JNT或者CART
// 例子
GET(JNT,1)

/*
函数功能描述:拖动模式开关
uint8_t state - 1-打开拖动模式,0-关闭拖动模式
*/
int DragTeachSwitch(uint8_t state)
```

(续下页)

(接上页)

```
// 例子
DragTeachSwitch(0)

/*
函数功能描述:机械臂使能开关
uint8_t state - 1-机械臂使能,0-机械臂去使能
*/
int RobotEnable(uint8_t state)
// 例子
RobotEnable(1)

/*
函数功能描述:模式切换
uint8_t state - 1-手动模式,0-自动模式
*/
int Mode(uint8_t state)
// 例子
Mode(1)

/*
函数功能描述:设置当前模式下机械臂速度
float vel - 速度百分比,范围为1-100
*/
int SetSpeed(float vel)
// 例子
SetSpeed(10)

/*
函数功能描述:设置并加载指定序号的工具坐标系
int id - 工具坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 工具坐标系的偏移量信息
*/
int SetToolCoord(int id, float x,float y, float z,float rx,float ry,float rz)
// 例子
SetToolCoord(1,0,0,0,0,0,0)

/*
函数功能描述:设置工具坐标系列表
int id - 工具坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 工具坐标系的偏移量信息
*/
int SetToolList(int id, float x,float y, float z,float rx,float ry,float rz );
// 例子
```

(续下页)

(接上页)

```
SetToolList(1,0,0,0,0,0,0)

/*
函数功能描述:设置外部工具坐标系
int id - 工具坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 外部工具坐标系的偏移量信息
*/
int SetExToolCoord(int id, float x,float y, float z,float rx,float ry,float rz);
// 例子
SetExToolCoord(1,0,0,0,0,0,0)

/*
函数功能描述:设置外部工具坐标系列表
int id - 工具坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 外部工具坐标系的偏移量信息
*/
int SetExToolList(int id, float x,float y, float z,float rx,float ry,float rz);
// 例子
SetExToolList(1,0,0,0,0,0,0)

/*
函数功能描述:设置工件坐标系
int id - 工件坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 工件坐标系的偏移量信息
*/
int SetWObjCoord(int id, float x,float y, float z,float rx,float ry,float rz);
// 例子
SetWObjCoord(1,0,0,0,0,0,0)

/*
函数功能描述:设置工件坐标系列表
int id - 工件坐标系编号,范围1-15
float x,y,z,rx,ry,rz - 工件坐标系的偏移量信息
*/
int SetWObjList(int id, float x,float y, float z,float rx,float ry,float rz);
// 例子
SetWObjList(1,0,0,0,0,0,0)

/*
函数功能描述:设置末端负载重量
float weight - 负载重量,单位kg
*/
int SetLoadWeight(float weight);
```

(续下页)

(接上页)

```
// 例子
SetLoadWeight(3.5)

/*
函数功能描述:设置末端负载质心坐标
float x,y,z - 质心坐标,单位为mm
*/
int SetLoadCoord(float x,float y,float z);
// 例子
SetLoadCoord(10,20,30)

/*
函数功能描述:设置机器人安装方式
uint8_t install - 安装方式,0-正装,1-侧装,2-倒装
*/
int SetRobotInstallPos(uint8_t install);
// 例子
SetRobotInstallPos(0)

/*
函数功能描述:设置机器人安装角度,自由安装
double yangle - 倾斜角
double zangle - 旋转角
*/
int SetRobotInstallAngle(double yangle,double zangle);
// 例子
SetRobotInstallAngle(90,0)


//安全配置
/*
函数功能描述:设置机器人碰撞等级
float level1-level6 - 1-6轴的碰撞等级,范围是1-10
*/
int SetAnticollision(float level1, float level2, float level3, float level4, float␣
↪level5, folat level6);
// 例子
SetAnticollision(1,1,1,1,1,1)

/*
函数功能描述:设置碰撞后策略
int strategy - 0-报错停止,1-继续运行
*/
```

(续下页)

(接上页)

```
int SetCollisionStrategy(int strategy);
// 例子
SetCollisionStrategy(1)

/*
函数功能描述:设置正限位,注意设置值必须在硬限位范围内
float limit1-limit6 - 6个关节限位值
*/
int SetLimitPositive(float limit1, float limit2, float limit3, float limit4, float limit5, float limit6);
// 例子
SetLimitPositve(100,90,90,90,90,90)

/*
函数功能描述:设置负限位,注意设置值必须在硬限位范围内
float limit1-limit6 - 6个关节限位值
*/
int SetLimitNegative(float limit1, float limit2, float limit3, float limit4, float limit5, float limit6);
// 例子
SetLimitNegative(-100,-90,-90,-90,-90,-90)

/*
函数功能描述:错误状态清除
*/
int ResetAllError();

/*
函数功能描述:关节摩擦力补偿开关
uint8_t state - 0-关, 1-开
*/
int FrictionCompensationOnOff(uint8_t state);
// 例子
FrictionCompensationOnOff(1)

/*
函数功能描述:设置关节摩擦力补偿系数-正装
float coeff1-coeff6 - 6个关节补偿系数,范围是0-1
*/
int SetFrictionValue_level(float coeff1,float coeff1,float coeff3,float coeff4,float coeff5,float coeff6);
// 例子
SetFrictionValue_level(1,1,1,1,1,1)
```

(续下页)

(接上页)

```

/*
函数功能描述:设置关节摩擦力补偿系数-侧装
float coeff1-coeff6 - 6个关节补偿系数,范围是0-1
*/
int SetFrictionValue_wall(float coeff1,float coeff1,float coeff3,float coeff4,float
↪coeff5,float coeff6);
// 例子
SetFrictionValue_wall(0.5,0.5,0.5,0.5,0.5,0.5)

/*
函数功能描述:设置关节摩擦力补偿系数-倒装
float coeff1-coeff6 - 6个关节补偿系数,范围是0-1
*/
int SetFrictionValue_ceiling(float coeff1,float coeff1,float coeff3,float coeff4,
↪float coeff5,float coeff6);
// 例子
SetFrictionValue_ceiling(0.5,0.5,0.5,0.5,0.5,0.5)


//外设控制
/*
函数功能描述:激活夹爪
int index - 夹爪编号
uint8_t act - 0-复位, 1-激活
*/
int ActGripper(int index,uint8_t act);
// 例子
ActGripper(1,1)

/*
函数功能描述:控制夹爪
int index - 夹爪编号
int pos - 位置百分比,范围0-100
*/
int MoveGripper(int index,int pos);
// 例子
MoveGripper(1,10)


//IO控制
/*
函数功能描述:设置控制箱数字量输出

```

(续下页)

(接上页)

```
int id - io编号,范围0-15
uint_t status - 0-关, 1-开
*/
int SetDO(int id,uint8_t status);
// 例子
SetDO(1,1)

/*
函数功能描述:设置工具数字量输出
int id - io编号,范围0-1
uint_t status - 0-关, 1-开
*/
int SetToolDO(int id,uint8_t status);
// 例子
SetToolDO(0,1)

/*
函数功能描述:设置控制箱模拟量输出
int id - io编号,范围0-1
float vlaue - 电流或者电压值百分比,范围0-100
*/
int SetAO(int id,float value);
// 例子
SetAO(1,100)

/*
函数功能描述:设置工具模拟量输出
int id - io编号,范围0
float vlaue - 电流或者电压值百分比,范围0-100
*/
int SetToolAO(int id,float value);
// 例子
SetToolAO(0,100)


//运动指令
/*
函数功能描述:机器人点动
uint8_t ref - 0-关节点动, 2-基坐标系下点动, 4-工具坐标系下点动, 8-工件坐标系下点动
uint8_t nb - 1-关节1(或x轴),2-关节2(或y轴),3-关节3(或z轴),4-关节4(或绕x轴旋转),5-
↪关节5(或绕y轴旋转),6-关节6(或绕z轴旋转)
uint8_t dir - 0-负方向, 1-正方向
float vel - 速度百分比, 范围为0-100
```

(续下页)

(接上页)

```
*/
int StartJOG(uint8_t ref, uin8_t nb, uint8_t dir, float vel);
// 例子
StartJOG(1,1,1,10)

/*
函数功能描述:机器人点动停止
uint8_t ref - 0-关节点动停止, 2-基坐标系下点动停止, 4-工具坐标系下点动停止, 8-
↪工件坐标系下点动停止
*/
int StopJOG(uint8_t ref);
// 例子
StopJOG(1)

/*
函数功能描述:机器人点动立即停止
*/
int ImmStopJOG();

/*
函数功能描述:关节空间运动
string point_name - 预存点位名称,比如JNT1就是关节点位信息序号为1的点位,
↪CART1就是笛卡尔点位信息序号为1的点位,
↪MoveJ指令支持输入关节点位或者笛卡尔点位。需要注意的,
↪MoveJ指令由于默认参数中有指定工具坐标系和工件坐标系,
↪当这两个坐标系序号与当前加载的不一致时,该指令会导致报错,
↪需要在默认参数中修改坐标系参数并load参数后再运行该运动指令。
float vel - 指令速度百分比,范围0-100
*/
int MoveJ(string point_name, float vel);//point_name是输入预存点位信息,
// 例子
MoveJ(JNT1,10)

/*
函数功能描述:笛卡尔空间直线运动
string point_name - 预存点位名称,比如JNT1就是关节点位信息序号为1的点位,
↪CART1就是笛卡尔点位信息序号为1的点位,
↪MoveL指令支持输入关节点位或者笛卡尔点位。需要注意的,
↪MoveL指令由于默认参数中有指定工具坐标系和工件坐标系,
↪当这两个坐标系序号与当前加载的不一致时,该指令会导致报错,
↪需要在默认参数中修改坐标系参数并load参数后再运行该运动指令。
float vel - 指令速度百分比,范围0-100
*/
```

(续下页)

(接上页)

```
int MoveL(string point_name,float vel);
// 例子
MoveL(CART1,10)

/*
函数功能描述:笛卡尔空间圆弧运动
string point1_name point2_name - 预存点位名称,比如JNT1就是关节点位信息序号为1的点位,
↪CART1就是笛卡尔点位信息序号为1的点位,MoveC指令支持输入关节点位或者笛卡尔点位,
↪但是两个点位必须同类型的,即不支持第一个点位输入关节空间点位,
↪第二个点位输入笛卡尔点位。需要注意的,
↪MoveC指令由于默认参数中有指定工具坐标系和工件坐标系,
↪当这两个坐标系序号与当前加载的不一致时,该指令会导致报错,
↪需要在默认参数中修改坐标系参数并load参数后再运行该运动指令。
float vel - 指令速度百分比,范围0-100
*/
int MoveC(string point1_name,string point2_name, float vel);
// 例子
MoveC(JNT1,JNT2,10)

/*
函数功能描述:样条运动开始
*/
int SplineStart();

/*
函数功能描述:关节空间样条运动,该指令只支持输入JNT1这样的关节数据,输入笛卡尔点位会报错
string point_name - 预存点位名称,比如JNT1就是关节点位信息序号为1的点位。
float vel - 速度百分比,范围0-100
*/
int SplinePTP(string point_name, float vel);
// 例子
SplinePTP(JNT2,10)

/*
函数功能描述:样条运动结束
*/
int SplineEnd();

/*
函数功能描述:笛卡尔空间样条运动开始
uint8_t ctlpoint - 0-轨迹经过路径点, 1-轨迹不经过控制点,至少4个点
*/
int NewSplineStart(uint8_t ctlpoint);
```

(续下页)

(接上页)

```
// 例子
NewSplineStrart(1)

/*
函数功能描述:笛卡尔空间样条运动,只能输入CART1这样的笛卡尔空间点位,
↪输入关节空间点位会报错
string point_name - 预存点位名称,比如CART1就是笛卡尔空间点位信息序号为1的点位。
float vel - 速度百分比,范围0-100
int lastflag - 0-不是最后一个点, 1-是最后一个点
*/
int NewSplinePoint(string point_name, float vel, int lastflag);
// 例子
NewSplinePoint(JNT2,20,0)

/*
函数功能描述:笛卡尔空间样条运动结束
*/
int NewSplineEnd();

/*
函数功能描述:停止运动
*/
int StopMotion();

/*
函数功能描述:点位整体偏移开始
int flag - 0-基坐标系下/工件坐标系下偏移, 2-工具坐标系下偏移
double x,y,z,rx,ry,rz - 偏移位姿量
*/
int PointsOffsetEnable(int flag,double x,double y,double z,double rx,double ry,double
↪rz);
// 例子
PointsOffsetEnable(1,10,10,10,0,0,0)

/*
函数功能描述:点位整体偏移结束
*/
int PointsOffsetDisable();
```

CHAPTER 5

宣传册

法奥意威公司简介

法奥意威产品样册

法奥意威医疗行业案例

法奥意威焊接宣传册

法奥意威码垛宣传册

CHAPTER 6

资质认证

法奥意威 CE 证书

法奥意威 CR 证书

CHAPTER 7

二次开发

FRCobot CPP SDK

FRCobot CPP SDK-v2.0.0

FRCobot C#-v1.0.0 SDK

FRCobot Python SDK

FRCobot Python-v2.0.0 SDK

FRCobot ROS

FRCobot ROS2

CHAPTER 8

本体 & 尺寸图纸

FR3 图纸

FR5 图纸

FR10 图纸

FR16 图纸

FR20 图纸

CHAPTER 9

3D 模型

FRCobots-V5.0 STEP 模型

FRCobots-V6.0 STEP 模型

CHAPTER 10

软件下载

QNX 3.6.6