智元精灵 G01
GENIE Development Kit
V1.5.0 使用指南

AGIBOT

目录

变更记录

版本	修改内容摘要	备注
V1.0.8	初版	
V1.1.3	• 新增 Python 接口； • 优化相机数据传输方案，优化帧率； • 取消相机压缩图 topic 传输方式； • 模式切换配置文件增加； • 相机参数客制化测试方案调整； • 夹爪支持连续值控制； • ROS 接口增加底盘信息反馈； • ROS 接口头部状态 topic 结构更改； • ROS 接口灵巧手 topic 名称更新；	
V1.2.0	• **ROS2 topic 名称更换（SDK 替换为 HAL）**； • ROS2 新增运控相关接口； • ROS2 新增底盘建图、定位导航接口； • Python 新增底盘建图、定位导航接口；	
V1.2.0 patch	• Python 新增 fault clear 及 pause walk 接口； • Pyhton 新增异步运控接口；	
V1.4.0	• ROS2 末端状态反馈 topic 修改： ◦ 删除末端上报 topic: `/hal/hand_joint_state`, `/hal/gripper_joint_state`, 改为使用`/hal/left_ee_data`, `/hal/right_ee_data` 作为末端状态，分左右末端进行上报。 ◦ `/hal/left_ee_data`, `/hal/right_ee_data` 增加 name 字段 • ROS2 末端控制 topic 修改： ◦ 删除末端接收 topic: `/wbc/hand_command`, `/wbc/gripper_command` ◦ 改为使用`/wbc/left_ee_command`, `/wbc/right_ee_command` • 模式切换配置文件模版更换： ◦ hybrid_deploy_depth53 改为 copilot，recorderpbtxt 改为 vr.pbtxt； • ROS 接口增加末端类型字段： ◦ `/hal/whole_body_status` 中增加字段左右末端类型； • 默认相机配置变更：copilot.pbtxt 默认开启头部鱼眼相机 • Web 前端支持读取标定参数和各模块版本信息读取	
V1.5.0	• robot_service.py 内置为命令行工具： ◦ 例如 robot-service -s -c conf/idle.pbtxt • Python 新增接口： ◦ 运控切换模式、运控全身急停、末端绝对位姿接口 ◦ 整机状态接口、获取底盘状态接口、获取电池状态接口 • ROS 新增接口： ◦ x86 侧新增腕部 Depth 图像 topic：/camera/hand_left_depth, /camera/hand_right_depth	

1. 网络配置

使用前将机器人背面的 Debug 网口与外部 PC 组成局域网。PC 通过网线直接连接 G1 接口面板的 Debug 网口。

G1 控制器网络配置信息：

- IP：10.42.0.101
- 子网掩码：255.255.255.0

PC 网络配置：

- 固定 IP 方式
- IP：10.42.0.xxx（10.42.0.10~10.42.0.99）
- 子网掩码：255.255.255.0
- 网关：（可选）
- 网络带宽：1000Mbps

> 可以使用 iftop 工具监控一下 PC 端网络带宽情况

配置完成后，验证 PC 和 G1 可以使用网络互相访问。

2. 环境配置与基础使用

2.1 PC 环境要求

操作系统要求：Ubuntu 22.04

硬件配置要求：

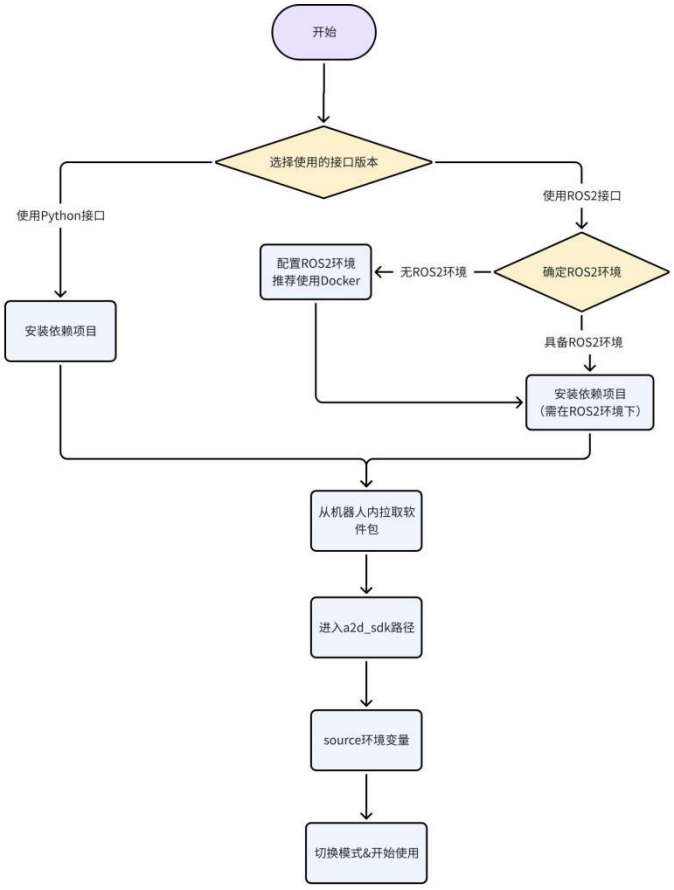
- CPU：x86 架构
- GPU：根据客户需求自行选择

软件环境要求

- Python：3.10
- ROS2 版本要求：ROS2 Humble（若使用 ROS2 接口，则需要具备 ROS2 环境）

2.2 PC 安装部署

PC 侧部署流程



PC 侧安装依赖项目：

```
sudo apt install iproute2
pip install -r requirements_gui.txt
```

requirements_gui.txt

```
requirements_gui.txt
numpy
protobuf==3.12.4
ruckig==0.14.0
opencv-python==4.10.0.84
scipy
zmq==0.0.0
pyzmq==26.2.0
matplotlib
```

GDK 可以运行在 conda、docker 等虚拟环境中，可以在 PC 安装后再安装依赖以防止

与您的环境冲突。

确认 PC 与 G1 的网络连接后。执行以下命令，x86 PC 内部署 GDK 环境。

注意：

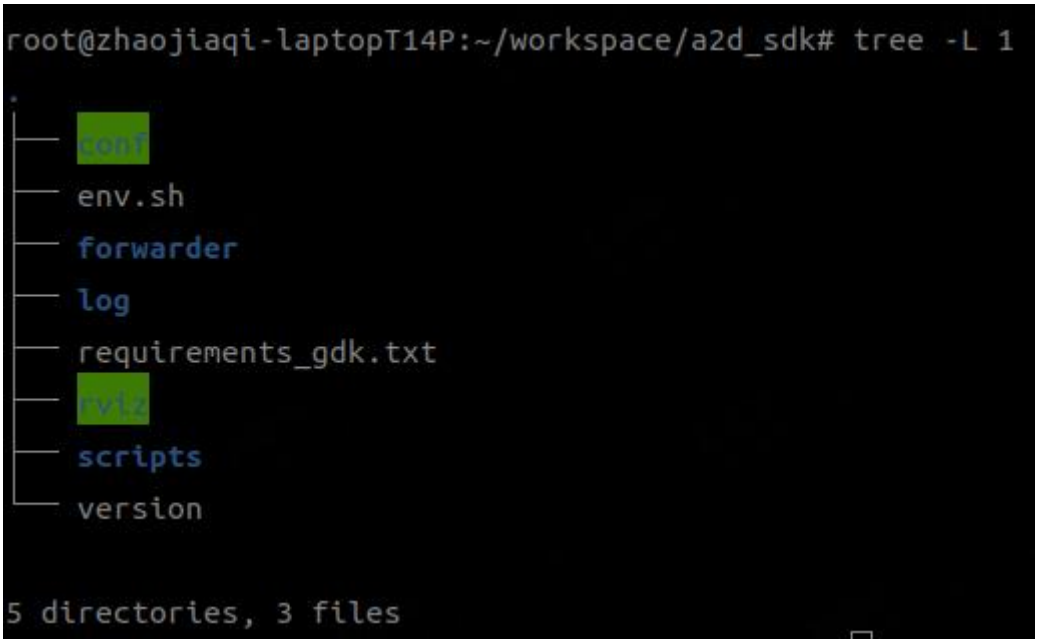
- 需要进入所需的虚拟环境中再从机器人内获取 GDK 软件包
- 更新版本后，需要更新 PC 内的环境信息

```
curl -sSL http://10.42.0.101:8849/install.sh | bash
```

注意，安装完 sdk 后请执行

```
cd a2d_sdk
source env.sh
```

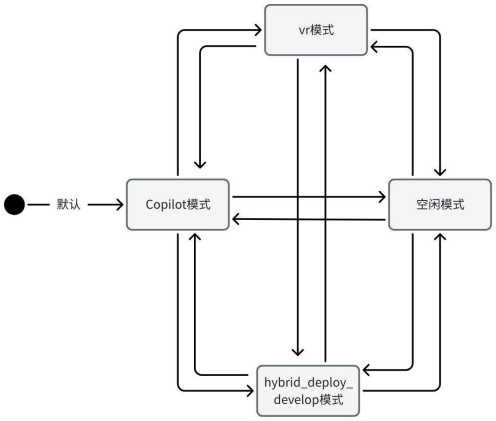
执行后文件夹：

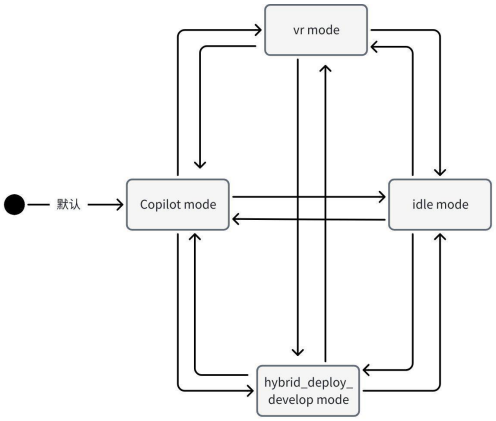


2.3 模式切换

由于部分电机启动初始化时间较长，在开机后请等待半分钟至一分钟再进行模式切换。

G1 内的应用有四种工作模式：





Copilot 模式：使用 copilot.pbtxt，该模式下支持外部调用 GDK 各项接口；

- 该配置文件支持开启头部 D455 或 D457 色彩及深度通道，左右手腕部 D405 色彩通道共 4 路摄像头数据，默认帧率 30Hz。
  - D455 相机分辨率：color 1280×720 depth 1280×720
  - D457 相机分辨率：color 1280×800 depth 1280×800
  - D405 相机分辨率：color 848×480

hybrid_deploy_develop：使用 hybrid_deploy_develop.pbtxt，支持用户配置相机通道以及预处理，详细参考相机客制化详解。

VR 模式：该模式下仅支持通过 VR 或动捕设备遥操作。不支持 GDK 接口调用。

空闲模式：关闭及机器人端侧全部应用，可以用于停止或重启应用；智元精灵 G01 GDK v1.2.0 版使用指南-0512.pdf 智元精灵 G01 GDK v1.2.0 版使用指南-0512.pdf

注意：hybrid_deploy_develop.pbtxt：

注意：更改配置文件后，需保存重启进行模式切换。

2.3.1 模式切换操作

进入 docker 容器或对应的虚拟环境，运行以下命令进入 copilot 模式。

```
robot-service -s -c ./conf/copilot.pbtxt
```

使用以下命令进入自主开发模式。

```
robot-service -s -c ./conf/hybrid_deploy_develop.pbtxt
```

使用以下命令进入空闲模式。

```
robot-service -s -c ./conf/idle.pbtxt
```

使用以下命令进入 VR 模式。

```
robot-service -s -c ./conf/vr.pbtxt
```

关闭 ROS2 输出，若在开发中仅使用 Python 相关接口，建议在切换模式的命令中增加参数关闭 ROS2 的相关发布。

robot-service -s -c ./conf/copilot.pbtxt --no-ros

2.3.2 相机客制化详解

可自行修改 hybrid_deploy_develop.pbtxt 来达到自己需要的图像需求，修改 img_preproc 相关的字段即可。

注意：目前 457 相机不支持客制化

头部深度通道当前版本不支持裁剪；

```
支持的摄像头名称：
{
    "head",
    "hand_left",
    "hand_right",
    "back_left_fisheye",
    "hand_left_fisheye",
    "hand_right_fisheye",
    "back_right_fisheye",
    "head_center_fisheye",
    "head_left_fisheye",
    "head_right_fisheye"
}
```

```
mode: HYBRID_DEPLOY #必填项
# two preprocess config examples:
img_preproc [
    {
        pos: head #需要打开的摄像头名称，配置到的才会打开
        FrameRate: 25 #需要的帧率
        depth_enable: true #是否打开深度通道
        crop { #预处理裁剪 该字段不可删除
            enable: true #true 为打开，false 为关闭
            l: 0 #裁剪的左边界
            t: 0 #裁剪的上边界
            r: 424 #裁剪的右边界
            b: 240 #裁剪的下边界
        }
        resize { #预处理改变大小 该字段不可删除
            enable: true #开关    该字段不可删除
            w: 212 #withd 像素
            h: 120 #height 像素
        }
        permute { #提供给需要通道的转换的场景，默认[0,1,2]为 RGB 通道顺序，如
[2,1,0]则为 BGR
            enable: false
            order1: 0
            order2: 1
            order3: 2
        }
    },
    {
        pos: hand_right #需要打开的摄像头名称，配置到的才会打开
        FrameRate: 30 #需要的帧率
        crop { #预处理裁剪 该字段不可删除
            enable: false #true 为打开，false 为关闭
            l: 0 #裁剪的左边界
            t: 0 #裁剪的上边界
            r: 424 #裁剪的右边界
            b: 240 #裁剪的下边界
        }
        resize { #预处理改变大小 该字段不可删除
            enable: false #开关
            w: 212 #withd 像素
            h: 120 #height 像素
        }
        permute { #提供给需要通道的转换的场景，默认[0,1,2]为 RGB 通道顺序，如
```

```
[2,1,0]则为 BGR
            enable: false
            order1: 0
            order2: 1
            order3: 2
        }
    },
    {
        pos: back_left_fisheye #需要打开的鱼眼摄像头名称，若机器人有相关摄像
头
        FrameRate: 30 #需要的帧率
        undistort { #去畸变参数
            enable: true #开关
            bal: 1.0 #调整视野，默认 1.0
        }
        crop { #预处理裁剪
            enable: true #true 为打开，false 为关闭
            l: 200 #裁剪的左边界
            t: 200 #裁剪的上边界
            r: 700 #裁剪的右边界
            b: 700 #裁剪的下边界
        }
        resize { #预处理改变大小
            enable: false #开关
            w: 512 #withd 像素
            h: 512 #height 像素
        }
        permute { #提供给需要通道的转换的场景，默认[0,1,2]为 RGB 通道顺序，如
[2,1,0]则为 BGR
            enable: false
            order1: 0
            order2: 1
            order3: 2
        }
    }
]

hybrid_deploy_config { #必填项，无需更改
  use_cosine_sdk: true
  camera_model: "develop"
}
```

3. 调试工具

3.1 图像查看工具

基于 Rviz2 工具，支持相机实时图像查看。使用以下预设文件可根据 topic 直接预览对应相机的图像。

```
rviz2 -d rviz/hybrid_deploy.rviz
```

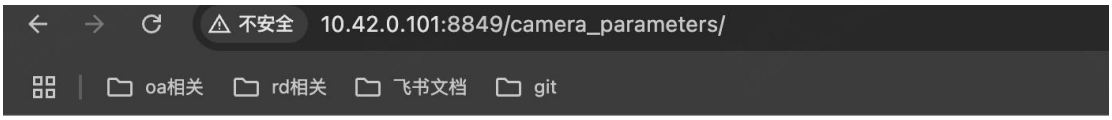
注意：在 docker 中执行时需开权限后再进 docker：

```
xhost +
```

除 RVIZ 外，也可使用 rqt 等工具进行数据的可视化。

3.2 日志记录与日志导出

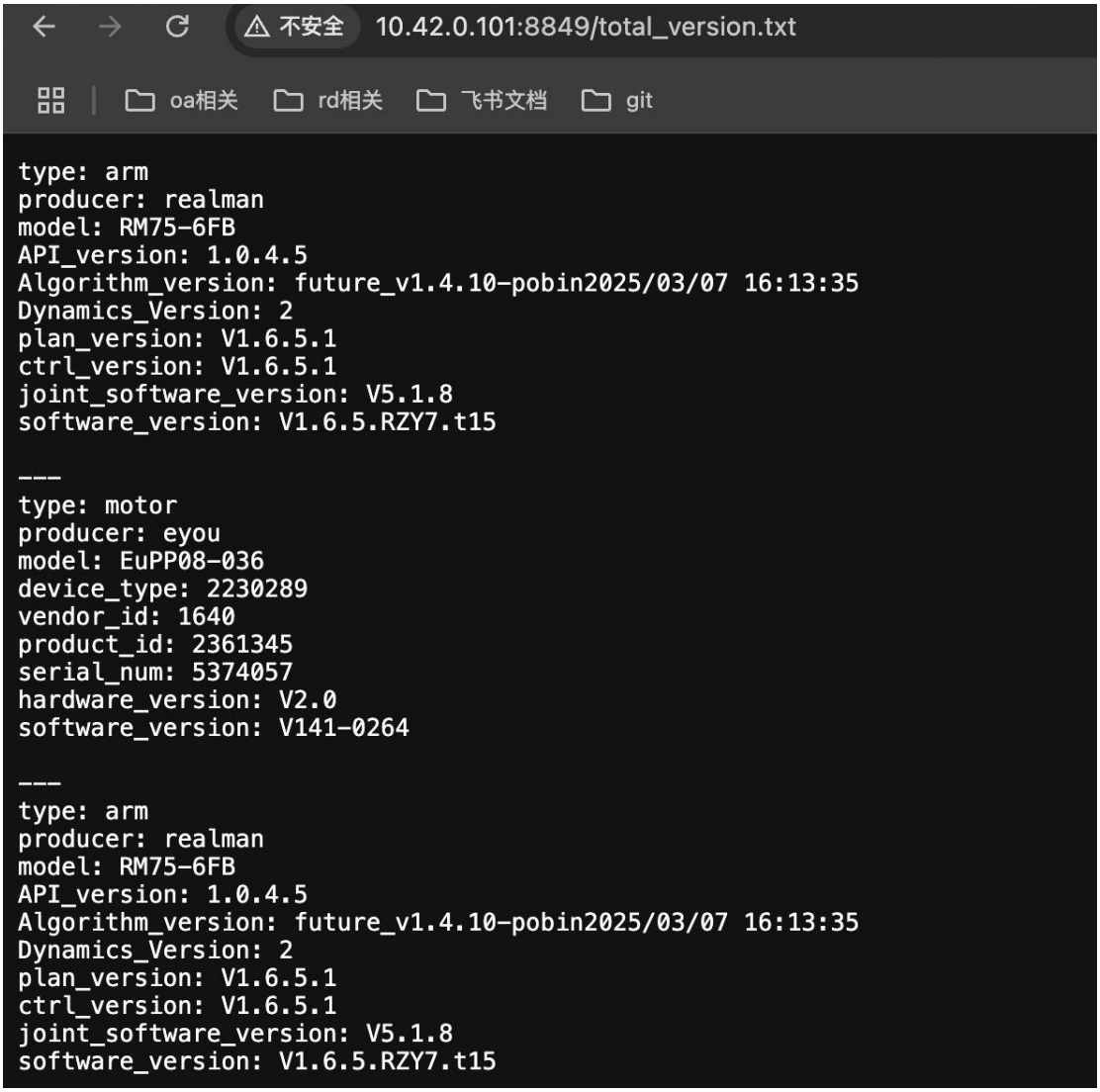
日志获取和管理：支持在 server 侧浏览器中查看 G1 上的日志，访问地址为：http://10.42.0.101:8849/，支持对日志的打包和自动清理



Directory listing for /camera_parameters/

- back_left_fisheye_intrinsic_params.json
- back_right_fisheye_intrinsic_params.json
- dh_parameters.yaml
- fisheye_camera_info.json
- hand_left_extrinsic_params.json
- hand_left_intrinsic_params.json
- hand_right_extrinsic_params.json
- hand_right_intrinsic_params.json
- head_center_fisheye_extrinsic_params.json
- head_center_fisheye_intrinsic_params.json
- head_extrinsic_params.json
- head_intrinsic_params.json
- head_left_fisheye_extrinsic_params.json
- head_left_fisheye_intrinsic_params.json
- head_right_fisheye_extrinsic_params.json
- head_right_fisheye_intrinsic_params.json
- parameters@
- rs_camera_info.json
- threshold_parameters.yaml

全量版本信息



3.3 时间同步工具

PC 端可在容器外通过以下命令，将 PC 与 G1 进行时间同步。

```
sudo ptp4l -i <eth_interface> -s -m
sudo phc2sys -c CLOCK_REALTIME -s <eth_interface> -m
```

若无 PTP4L 工具，则:

```
sudo apt update
sudo apt install linuxptp
```

可通过以下命令检查网卡是否支持:

```
ethtool -T <interface>
```

设置后需要重启底盘。

4. 机器人快速工具

GDK 中预制了 robot-controller，这个工具可以用来手动控制机器人的简单动作。

4.1 头部以及腰部接口示例

查询头部横摆和俯仰状态信息

```
robot-controller
输入 he 或 head
以下为输出示例，单位为角度与角度：
Enter the part and params: he
Current head state: [0.0, 25.000457337210047], usage: he <angle>,<angle>
```

查询腰部俯仰和升降状态信息

```
robot-controller
输入 wa 或 waist
以下为输出示例，单位为角度与厘米：
Enter the part and params: wa
Current waist state: [0.5230003543799336, 0.3000000762939453], usage: wa
<angle>,<height>
```

头部和腰部控制示例

```
robot-controller

控制腰输入为： wa 角度，厘米高度
Enter the part and params: wa 25,30
Current waist state: [0.5230003543799336, 0.3], move to
[0.4363323129985824, 30.0]
Move Waist: joint_flag: 12
joint_states: 0.0
joint_states: 25.000458
joint_states: 25.0
joint_states: 30.0

 控制头输入为：he 角度，角度
Enter the part and params: he 10,10
Current head state: [0.0, 25.00030474932202], move to
```

```
[0.17453292519943295, 0.17453292519943295]
Move Head: joint_flag: 3
joint_states: 10.0
joint_states: 10.0
joint_states: 0.43633285
joint_states: 0.3
```

4.2 末端执行器接口示例

4.2.1 夹爪使用示例

若机器配置的是二指夹爪，则使用以下命令获取夹爪的位置信息。

```
robot-controller

输入 gr 或 gripper
以下为输出事例，单位为左手和右手的张合程度：
Enter the part and params: gr
Current gripper state: [35.52, 35.36], usage: gr num,num
```

4.2.2 夹爪控制示例

```
robot-controller

控制腰输入为： gr 0-1，0-1 其中 0 为松开，1 为夹紧 g

实例：
Enter the part and params: gr 0.8,0
Current gripper state: [35.480000000000004, 35.36], move to [0.8, 0.0]
```

4.2.3 灵巧手使用实例

```
robot-controller

两种模式：
    hand 或 ha，此时需要输入灵巧手所有关节的角度值
    hand_as_gripper 或 ha_as_gr，此时只需要输入 0-1,0-1 的模式，即可按照控制夹爪的方式来控制灵巧手
```

4.3 恢复初始位姿

```
robot-controller
输入 re 或 reset，此时会恢复到设定的位姿，可以观察到 arm 的移动
```

5. ROS2 版接口说明

5.1 手臂接口

接口名称	类型（topic）	说明	周期
/hal/arm_joint_state	sensor_msgs/msg/JointState	手臂关节位置反馈	10ms
/hal/left_arm_data	genie_msgs/msg/ArmState	左臂关节详细状态反馈	10ms
/hal/right_arm_data	genie_msgs/msg/ArmState	右臂关节详细状态反馈	10ms
/hal/arm_command	sensor_msgs/msg/JointState	手臂关节绝对位置控制接口	10ms

接口名称	字段		说明
/hal/arm_joint_state	position		[left_arm_joint1_position, left_arm_joint2_position, left_arm_joint3_position, left_arm_joint4_position, left_arm_joint5_position, left_arm_joint6_position, left_arm_joint7_position, right_arm_joint1_position, right_arm_joint2_position, right_arm_joint3_position, right_arm_joint4_position, right_arm_joint5_position, right_arm_joint6_position, right_arm_joint7_position] 单位：弧度
接口名称	**字段**		**说明**
/hal/left_arm_data	motor_states[	id	关节序号

/hal/right_arm_data	]	enable	关节使能状态
		position	关节绝对位置，单位：弧度
		velocity	关节速度，单位: RPM
		current	关节电流, 单位：mA
		voltage	关节电压, 单位：V
		temperature	关节温度，单位：℃
		status	关节状态
		err_code	关节错误码
	force_data[]		六维力数据
	force_coordinate		六维力坐标系
	arm_state		机械臂状态
	system_error		系统错误码 0x0000 系统正常 0x1001 关节通信异常 0x1002 目标角度超过限位 0x1003 该处不可达，为奇异点 0x1004 实时内核通信错误 0x1005 关节通信总线错误 0x1006 规划层内核错误 0x1007 关节超速 0x1008 末端接口板无法连接 0x1009 超速度限制 0x100A 超加速度限制
/wbc/arm_command	position		[left_arm_joint1_position, left_arm_joint2_position, left_arm_joint3_position, left_arm_joint4_position, left_arm_joint5_position, left_arm_joint6_position, left_arm_joint7_position, right_arm_joint1_position, right_arm_joint2_position,

		right_arm_joint3_position, right_arm_joint4_position, right_arm_joint5_position, right_arm_joint6_position, right_arm_joint7_position] 单位：弧度 限制：目标关节角度与实时关节角之间差值小于 0.2618 rad (15°); 角速度限制：3rad/s; 加速度限制：6rad/s*2

5.2 头部接口

接口名称	类型（topic）	说明	周期
/hal/neck_state	genie_msgs/msg/HeadState	头部状态反馈	50ms
接口名称	**类型（service）**	**说明**	
/wbc/body_pose	genie_msgs/srv/BodyPose	头部、腰部控制接口	

接口详细说明

接口名称	字段	说明
/wbc/body_pose	joint_flag	15：默认 15
	joint_states[]	[head_yaw, head_pitch, body_pitch, lift_body] 头部横摆：head_yaw 1. 角度单位：角度 2. 原点位置：正前方 3. 正方向：与机器人同向，向左为正 4. 有效值范围：-90~90° 头部俯仰：head_pitch 1. 角度单位：角度 2. 原点位置：水平 3. 正方向：低头为正方向 4. 有效值范围：-20~25° 腰部俯仰：body_pitch

		1. 角度单位：角度 2. 原点位置：竖直位置与底盘垂直 3. 有效值范围：0~90° 腰部升降：lift_body 1. 单位：厘米 2. 原点位置：竖直方向最低点 3. 有效值范围：0~50cm
	block	默认为 true，暂未使用

接口名称	字段		说明
/sdk/neck_state	motor_states[]	id	关节序号
		enable	关节使能状态
		position	关节绝对位置，单位：弧度
		velocity	关节速度，单位: RPM
		effort	关节力矩，单位：
		current	关节电流，单位：mA
		voltage	关节电压，单位：V
		temperature	关节温度，单位：℃
		status	关节状态
		err_code	关节错误码

5.3 腰部接口

接口详细说明如下：

接口名称	类型（topic）	说明	周期
/hal/waist_state	genie_msgs/msg/WaistState	腰部俯仰和升降状态反馈接口	50ms
接口名称	**类型（service）**	**说明**	
/wbc/body_pose	genie_msgs/srv/BodyPose	头部、腰部控制接口	

接口名称	字段	元素	说明
/sdk/waist_state	motor_states[]	id	电机 ID
		enable	电机使能状态
		position	腰部俯仰： • 单位：弧度 腰部升降： • 单位：米
		velocity	电机速度（暂时未使用）
		effort	电机力矩（暂时未使用）

接口名称	类型（topic）	说明	周期
/hal/left_ee_data	genie_msgs/msg/EndState	Left endeffector position feedback (if use)	10 ms
/hal/right_ee_data	genie_msgs/msg/EndState	Right endeffector position feedback (if use)	10 ms
/wbc/left_ee_command	sensor_msgs/msg/JointState	Left endeffector control interface(if use)	10 ms
/wbc/right_ee_command	sensor_msgs/msg/JointState	right endeffector control interface(if use)	10 ms

		current	电机电流（暂时未使用）
		voltage	电机电压（暂时未使用）
		temperature	电机温度（暂时未使用）
		status	电机状态（暂时未使用用）
		err_code	故障码

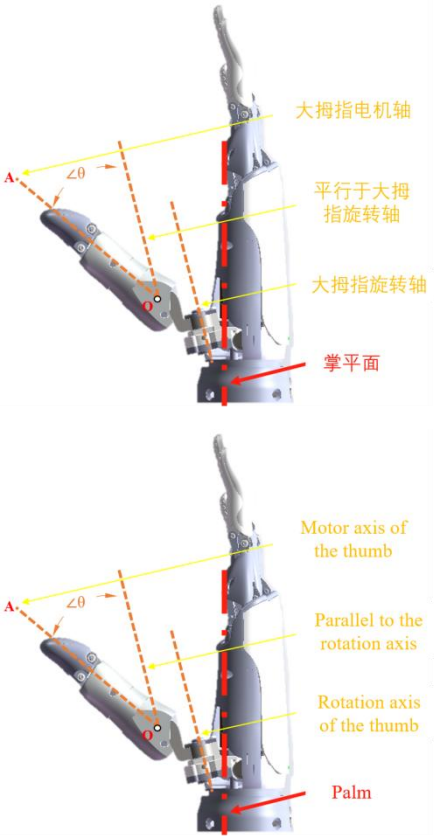
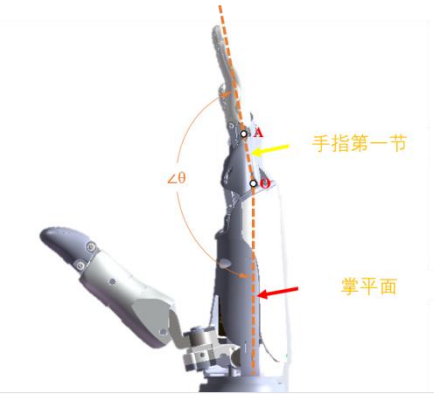
5.4 末端执行器接口

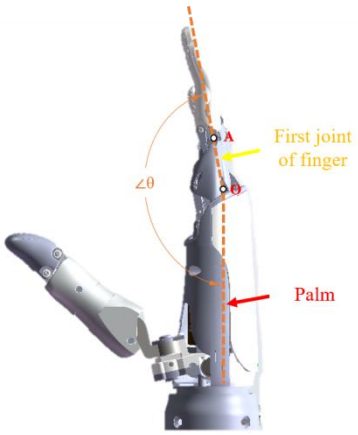
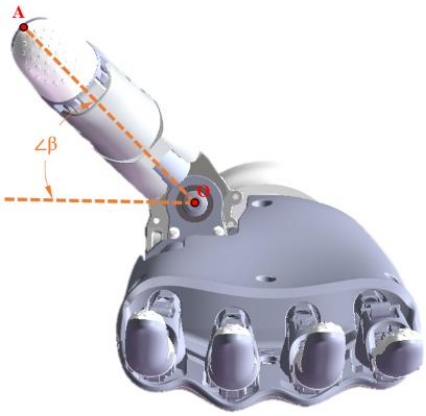
已适配多末端自动识别

接口名称	字段	说明
/wbc/left_ee_command	name[]	执行器种类
	position[]	夹爪：左夹爪闭合的距离：0 代表全行程张开，120 代表全关闭，单位 mm； 灵巧手：各关节角度，单位 rad； [拇指弯曲、食指弯曲、中指弯曲、无名指弯曲、小指弯曲、拇指旋转]
	velocity[]	预留
	effort[]	预留
/wbc/right_ee_command	name[]	执行器种类
	position[]	夹爪：右夹爪闭合的距离：0 代表全行程张开，120 代表全关闭，单位 mm； 灵巧手：各关节角度，单位 rad； [拇指弯曲、食指弯曲、中指弯曲、无名指弯曲、小指弯曲、拇指旋转]
	velocity[]	预留

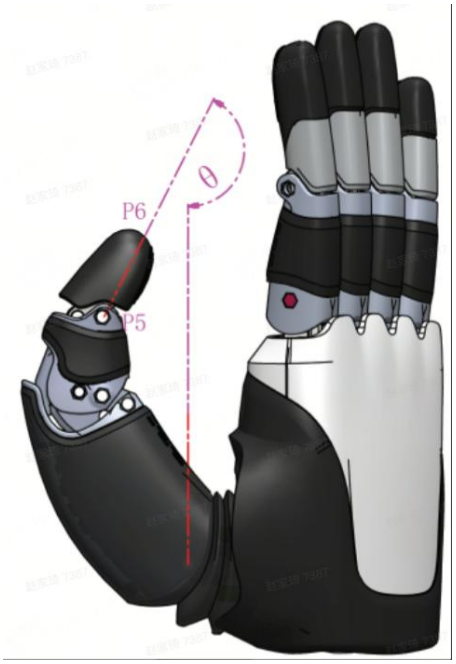
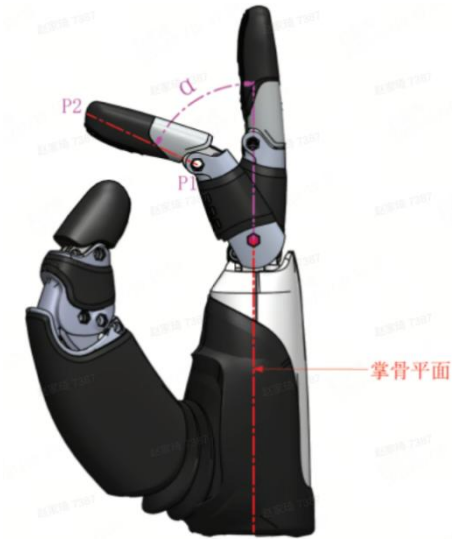
	effort[]	预留

灵巧手 1 活动空间范围

	关节序号	含义	示意图	角度范围
左手	0	大拇指电机轴与旋转轴夹角的目标值	 Thumb motor shaft Parallel to the thumb rotation shaft Thumb rotation shaft Palm plane	2.26° ~ 36.76° 0.0394~0.6416rad
	1	食指第一节与掌平面夹角的目标值		100.22°~178.37° 1.7492 ~3.1131 rad
	2	中指第一节与掌平面夹角的目标值		97.81° ~ 176.06° 1.7071 ~ 3.0728 rad

	3	无名指第一节与掌平面夹角的目标值	 First joint of the finger Palm plane	101.38° ~ 176.54° 1.7694 ~ 3.0812 rad
	4	小指第一节与掌平面夹角的目标值		98.84° ~ 174.86° 1.7251 ~ 3.0519 rad
	5	大拇旋转目标角度		0° ~ 90° 0 ~ 1.5708 rad
右手 （角度关系同左手）	0			
	1			
	2			
	3			
	4			
	5			

自研灵巧手运动范围

	关节序号	含义	示意图	角度范围
	0	拇指尖端弯曲与掌平面夹角		93.1°~217.1° 1.6249~3.7891rad
	1~4	食指 中指 无名指 小指		6.6°~201.5° 0.1152~3.5168rad

	5	拇指旋转		49°~180° 0.8552~3.1416rad

5.5 传感器接口

5.5.1 相机接口

接口名称	类型（topic）	说明
/camera/head_color	sensor_msgs/msg/Image	头部深度相机 RGB 图像
/camera/head_depth	sensor_msgs/msg/Image	头部深度相机深度图像
/camera/hand_left_color	sensor_msgs/msg/Image	左侧腕部深度相机 RGB 图像
/camera/hand_left_depth	sensor_msgs/msg/Image	左侧腕部深度相机深度图像
/camera/hand_right_color	sensor_msgs/msg/Image	右侧腕部深度相机 RGB 图像
/camera/hand_right_depth	sensor_msgs/msg/Image	右侧腕部深度相机深度图像
/camera/head_center_fisheye	sensor_msgs/msg/Image	头部中央鱼眼相机图像
/camera/head_left_fisheye	sensor_msgs/msg/Image	头部左侧鱼眼相机图像
/camera/head_right_fisheye	sensor_msgs/msg/Image	头部右侧鱼眼相机图像
/camera/hand_left_fisheye	sensor_msgs/msg/Image	左臂腕部鱼眼相机图像
/camera/hand_right_fisheye	sensor_msgs/msg/Image	右臂腕部鱼眼相机图像
/camera/back_right_fisheye	sensor_msgs/msg/Image	后向右后鱼眼相机图像
/camera/back_left_fisheye	sensor_msgs/msg/Image	后向右后鱼眼相机图像

5.5.2 六维力传感器

接口名称	类型（topic）	说明	周期
/hal/right_arm_data	genie_msgs/msg/ArmState	右臂关节与六维力传感器反馈接口	10ms
/hal/left_arm_data	genie_msgs/msg/ArmState	左臂关节与六维力传感器反馈接口	10ms

接口说明

接口名称	字段	说明

/hal/right_arm_data /hal/left_arm_data	force_data:	[x, y, z, rx, ry, rz] x：x 轴力，单位 N, 精度 0.001N, 力测量量程：0~200N; y：y 轴力，单位 N, 精度 0.001N z：z 轴力，单位 N, 精度 0.001N rx：x 轴力矩，单位 Nm, 精度 0.001Nm，力矩量程 0~7Nm; ry：y 轴力矩，单位 Nm, 精度 0.001Nm，力矩量程 0~7Nm; rz：z 轴力矩，单位 Nm, 精度 0.001Nm，力矩量程 0~7Nm;
	force_coordinate	坐标系：0 传感器坐标系； 1 工作坐标系； 2 工具坐标系；

5.6 底盘接口

接口名称	类型（topic）	说明	周期
/hal/batt_state	genie_msgs/msg/BatteryStatus	电池电量反馈	50ms
/mbc/wheel_command	geometry_msgs/msg/TwistStamped	底盘直接控制接口	50ms
/hal/position	genie_msgs/msg/Position	底盘信号反馈	50ms

接口说明

接口名称	字段	说明
/hal/batt_state	energy	电量：百分比
	voltage	电池电压，单位 V
	current	电流，单位 A
	status	状态：0: 保留, 1: 充电中, 2: 放电, 3: 未充电, 4: 充满, 5: 电量低
/hal/position	agv_status	底盘状态
	position_conf	定位置信度 0-100
	agv_pos_x	地图坐标系 x，单位（m）
	agv_pos_y	地图坐标系 y，单位（m）
	agv_pos_z	地图坐标系 z，单位（m）
	agv_angle	地图坐标系朝向角度，单位（弧度）
	odom_x	odom 坐标系 x， 单位（m）
	odom_y	odom 坐标系 y， 单位（m）
	odom_z	odom 坐标系 z， 单位（m）
	odom_angle	odom 坐标系朝向角度，单位（弧度）
	linear_speed	线速度，单位 m/s
	angular_speed	角速度，单位 弧度/s
	acc_x	IMU 坐标系加速度 x
	acc_y	IMU 坐标系加速度 y
	acc_z	IMU 坐标系加速度 z
	gyro_x	IMU 坐标系角速度 x

	gyro_y		IMU 坐标系角速度 y
	gyro_z		IMU 坐标系角速度 z
	roll		IMU 翻滚角
	pitch		IMU 俯仰角
	yaw		IMU 航向角
	map_id		当前地图 ID
	motor_states[]	id	电机 ID
		enable	使能状态
		position	暂未使用
		velocity	速度
		effort	预留
		current	预留
		voltage	预留
		temperature	预留
		status	预留
		err_code	预留
/mbc/wheel_command	twist.linear.x		沿 x 轴的线速度。向前为正，单位 m/s
	twist.angular.z		绕 z 轴的角速度

5.7 运动控制接口

5.7.1 接口说明

接口名称	类型（topic）	说明	周期
/hal/whole_body_status	genie_msgs/msg/WholeBodyStatus	整机执行器工作状态	50ms
/wbc/motion_control_status	genie_msgs/msg/MotionControlStatus	查看当前末端位置状态/碰撞对状态/MC 模块状态	10ms
/wbc/motion_stop	genie_msgs/msg/MotionStop	全身急停控制	
/wbc/set_control_mode	genie_msgs/msg/SetControlMode	设置运控的控制模式	
/wbc/end_effector_pose_control	genie_msgs/msg/EndEffectorPoseControl	末端绝对位置实时控制	
/wbc/joint_position_control	genie_msgs/msg/JointPositionControl	关节实时位置控制模式	

注意：调用/wbc/end_effector_pose_control 建议不要同时控制腰部俯仰和高度，可能会无法达到预期效果，因为腰部电机的响应时间不一致。

接口名称	字段	说明	周期

/hal/whole_body_status	right_arm_error: 0			
	left_arm_error: 0			
	right_arm_control: true			
	left_arm_control: true			
	right_arm_estop: false			
	left_arm_estop: false			
	right_end_error: 0			
	left_end_error: 0			
	right_end_model:		末端种类	
	left_end_model:		末端种类	
	waist_error: 0			
	lift_error: 0			
	neck_error: 0			
	chassis_error: 0			
接口名称	**字段**	**数据类型**	**说明**	**周期**
/wbc/motion_control_status	frame_names		运控参数配置定义	10 ms
	frame_poses	geometry_msgs/Pose[]	末端位姿	
	collision_pairs_1	string[]	[collision_pair_1, collision_pair_2]组成一组碰撞对的 frame 名称	
	collision_pairs_2	string[]		
	mode		运控模式: MODE_STOP=0 MODE_SERVO=1 MODE_PLANNING=2	
/wbc/motion_stop	input_type		急停触发源 GDK：54；	
/wbc/set_control_mode	input_type		触发源 GDK：54；	
	control_mode		MODE_STOP=0 停止 MODE_SERVO=1 伺服模式	
/wbc/end_effector_pose_control	lifetime	float64	指令有效时间，超过 lifetime 不发送下一帧指令则停止运动，单位 s	
	control_group	uint32	控制组: 4：左臂 8：右臂 12：双臂 20：左臂+腰部升降 36：左臂+腰部俯仰 52： 左臂+腰部 24：右臂+腰部升降 40：右臂+腰部俯仰 56：右臂+腰部	

			28：双臂+腰部升降 44：双臂+腰部俯仰 60：双臂+腰部	
	left_end_effector_pose	geometry_msgs/Pose	左臂末端绝对位姿	
	right_end_effector_pose	geometry_msgs/Pose	右臂末端绝对位姿	
/wbc/joint_position_control	lifetime	float64	指令有效时间，超过 lifetime 不发送下一帧指令则停止运动，单位 s	
	control_group	uint32	1：GROUP_HEAD_YAW 2:GROUP_HEAD_PITCH 4: GROUP_LEFT_ARM 8：GROUP_RIGHT_ARM 16：GROUP_WAIST_LIFT 32：GROUP_WAIST_PITCH 128：手或夹爪，mc 根据配置 256： 手或夹爪，mc 根据配置	
	head_yaw_joint_position	float64		
	head_pitch_joint_position	float64		
	left_arm_joint_positions	float64[]		
	right_arm_joint_positions	float64[]		
	waist_lift_joint_position	float64		
	waist_pitch_joint_position	float64		
	left_tool_joint_positions	float64[]		
	right_tool_joint_positions	float64[]	右末端执行器	

5.7.2 接口使用示例

获取运控状态

```
ros2 topic echo --once /wbc/motion_control_status
```

控制头部 yaw 角

```
ros2 topic pub /wbc/joint_position_control genie_msgs/msg/JointPositionControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
lifetime: 5
```

```
control_group: 1
head_yaw_joint_position: 0.5"
```

控制腰部升降

```
ros2 topic pub /wbc/joint_position_control genie_msgs/msg/JointPositionControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
lifetime: 5
control_group: 16
waist_lift_joint_position: 0.3"
```

控制腰部俯仰

```
ros2 topic pub /wbc/joint_position_control genie_msgs/msg/JointPositionControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
lifetime: 5
control_group: 32
waist_pitch_joint_position: 0.2"
```

控制右夹爪

```
ros2 topic pub /wbc/joint_position_control genie_msgs/msg/JointPositionControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
lifetime: 5
control_group: 256
right_tool_joint_positions:
- 0.1"
```

使用关节角度接口控制右臂

```
ros2 topic pub /wbc/joint_position_control genie_msgs/msg/JointPositionControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
lifetime: 5
control_group: 8
right_arm_joint_positions:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.5"
```

使用末端位姿控制左右臂

```
ros2 topic pub -r 50 /wbc/end_effector_pose_control genie_msgs/msg/EndEffectorPoseControl
"header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: base_link
lifetime: 10
control_group: 12
left_end_effector_pose:
  position:
    x: 0.6706307451432025
    y: 0.29775424009454377
    z: 0.6109336265657352
  orientation:
    x: -0.20724560103774245
    y: 0.7973315022771579
    z: -0.5668400344607115
    w: -0.002027722746358685
right_end_effector_pose:
  position:
    x: 0.6298673206706626
    y: -0.34436789386677213
    z: 0.5942527678756625
  orientation:
    x: -0.7895529235635838
    y: 0.15635523877588278
    z: -0.1802825440499219
    w: 0.5653825470514844"
```

全身急停

```
ros2 topic pub /wbc/motion_stop genie_msgs/msg/MotionStop "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
input_type: 54"
```

清除急停

```
ros2 topic pub /wbc/set_control_mode genie_msgs/msg/SetControlMode "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
input_type: 54
control_mode: 1"
```

5.8 SLAM、地图管理与导航接口

5.8.1 接口说明

接口名称	类型（service）	说明	
/hal/switch_nav_mode	genie_msgs/srv/AGVModeControl	底盘工作模式切换，需要切换至自动模式后才能自主导航	
/hal/slam/mapping_control	genie_msgs/srv/AGVMappingControl	底盘建图设置	
/hal/map/get_map	genie_msgs/srv/AGVGetMap	获取当前所有可用地图信	
/hal/map/switch_map	genie_msgs/srv/AGVSwitchMap	切换地图	
/hal/map/remove_map	genie_msgs/srv/AGVMapDelete	删除地图	
/hal/nav/navigate_to_pose	genie_msgs/srv/NavigatePose	建图后移动到指定地点	
/hal/nav/move_relative	genie_msgs/srv/NavigatePose	移动到相对位置	
/hal/loc/relocalize	genie_msgs/srv/ReLocalization	重定位	
/hal/nav/collision	genie_msgs/srv/AGVCollisionLevel	底盘避障等级设置	
接口名称	**类型（topic）**	**说明**	**周期**
/hal/position	genie_msgs/msg/Position	获取底盘工作状态	50ms
/hal/map	nav_msgs/msg/OccupancyGrid	建图时广播地图数据	

/hal/switch_nav_mode

```
std_msgs/Header     header
int32               mode        # 1: 手动, 2: 自动
---
int32               error_code   #  0: success, 1:切换失败
```

/hal/slam/mapping_control

```
std_msgs/Header     header
int32 control         # 1 开始建图 2 停止建图
bool save_map       # 当 contol 为 2(停止建图时使用)时 true 保存地图
false 丢弃地图
string map_name     # 地图名称
---
# 0:成功， 1:已经在建图中，3：地图名已经存在，4：切换地图失败，5：开始建
图失败，6：当前没有在建图中，停止建图失败
# 7:停止建图失败， 8:没有找到保存的地图， 9：不支持当前的操作类型
int32 error_code
```

/hal/map/switch_map

```
std_msgs/Header header
int32 map_id
---
int32 error_code
```

/hal/map/remove_map

```
#根据 map 列表获取的地图，选择一个可以进行删除
std_msgs/Header header
int32 map_id          # map id
---
int32 error_code
```

/hal/map/get_map

```
#根据指定 map id 获取地图详细信息
std_msgs/Header header
int32 map_id          # map id
---
int32 x             #地图原点 x
int32 y             #地图原点 y
int32 width         #地图 宽
int32 height        #地图 高
string name    #地图 ming
string data         #地图图片，以 base64 编码的 png 图片，png 背景为透明
int32 error_code      #错误码 0:成功
```

/hal/nav/navigate_to_pose

```
# /hal/nav/navigate_to_pose  #全局路径规划
std_msgs/Header header
float32 x
float32 y
float32 theta
---
int32 error_code
```

/hal/nav/move_relative

```
# /sdk/nav/move_relative     #相对位置移动
std_msgs/Header header
float32 x
float32 y
float32 theta
---
int32 error_code
```

/hal/loc/relocalize

```
std_msgs/Header header
int32 relocalization_mode        # 重定位类型，暂时只支持手动重定位
1:手动重定位
float32 x
float32 y
float32 theta
---
int32 error_code
```

/hal/nav/collision

```
# /sdk/nav/collision
std_msgs/Header header
int32 collision_level          # 避障等级 1，2，3 ，  等级越高，越灵敏
---
int32 error_code
```

5.8.2 使用示例

切换底盘控制模式

```
ros2 service call /hal/switch_nav_mode
```

```
genie_msgs/srv/AGVModeControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
mode: 1"

------
响应：
返回 errcode = 0
```

常规建图控制

```
ros2 service call /hal/slam/mapping_control
genie_msgs/srv/AGVMappingControl "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
control: 1
save_map: false
map_name: 'test3'"
#1 开始建图，2 停止建图，True/False 是否保存地图，map_name 不能重复
```

```
ros2 service call /hal/waist_errclean std_srvs/srv/Trigger
```

6. ROS2 使用示例

使用前请执行

```
source env.sh
```

6.1 手臂接口示例

读取手臂关节位置

```
ros2 topic echo /hal/arm_joint_state
```

读取左臂关节详细状态示例

```
ros2 topic echo /hal/left_arm_data
```

控制双臂控制示例

使用手臂关节绝对角度接口控制手臂运动时，需要注意结合手臂当前位置。具体注意事项在接口描述表格中

```
ros2 topic pub -r 100 /wbc/arm_command sensor_msgs/msg/JointState "header:
  stamp:
    sec: 1735468396
    nanosec: 47343790
  frame_id: ''
position:
- 2.423124313354492
- -1.5161956548690796
- -1.1007285118103027
- -0.1759658008813858
- 0.05287349969148636
- -0.10593894869089127
- 0.21552494168281555
- 0.4446958005428314
- -1.509669303894043
- -0.1099175438284874
- 0.20767244696617126
- 0.04812709987163544
- 0.38030532002449036
- 0.124674300104379654
velocity: []
effort: []"
```

6.2 头部以及腰部接口示例

查询头部横摆和俯仰状态信息

```
ros2 topic echo /hal/neck_state
```

查询腰部俯仰和升降状态信息

```
ros2 topic echo /hal/waist_state
```

头部和腰部控制示例

```
ros2 service call /wbc/body_pose genie_msgs/srv/BodyPose "header:
  stamp:
    sec: 0
```

```
    nanosec: 0
  frame_id: ''
joint_flag: 15
joint_states: [0.0, 0.0, 0.5, 20.0]
block: true"
```

6.3 末端执行器接口示例

6.3.1 已适配多种末端执行器

获取执行器数据:

```
ros2 topic echo /hal/left_ee_data
ros2 topic echo /hal/right_ee_data
```

夹爪控制示例

```
ros2 topic pub /wbc/left_ee_command sensor_msgs/msg/JointState "header:
  stamp:
    sec: 1735467567
    nanosec: 603753378
  frame_id: ''
name:
- left
position:
- 1.0
velocity: []
effort: []"
```

6.3.2 灵巧手使用示例

控制灵巧手示例

```
ros2 topic pub /wbc/left_ee_command sensor_msgs/msg/JointState "header:
  stamp:
    sec: 1732845808
    nanosec: 921740724
  frame_id:
name: []
position:
- 0.6414620000000001
- 3.1116840000000003
- 3.072247
- 3.080623
- 3.044327
- 0.0
velocity: []
effort: []"
```

6.4 预览相机数据

注意：在 docker 中执行时需要给容器加上权限

```
rviz2 -d rviz/hybrid_deploy.rviz
```

6.5 读取六维力数据

读取左手末端六维力传感器数据。

```
ros2 topic echo /hal/left_arm_data
```

话题中包含了六维力字段：

```
force_data:
- -0.21400000154972076
- 8.555999755859375
- 8.677000045776367
- -0.6140000224113464
- -0.2709999978542328
- 0.05400000140070915
force_coordinate: 0
force_state: 0
```

6.6 底盘接口示例

查看电池电量。

```
ros2 topic echo /hal/batt_state
```

控制底盘前进

```
ros2 topic pub -r 20 /wbc/wheel_command geometry_msgs/msg/TwistStamped "header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
twist:
  linear:
    x: 0.1
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
"
```

7. python 函数 RobotDds 接口使用

```
from a2d_sdk.robot import RobotDds as Robot
```

RobotDds 类提供了控制机器人各个部位的接口，包括手臂、手部、头部、腰部和轮子的控制。

7.1 状态查询函数

7.1.1 实时状态查询

```
# 所有状态查询函数返回格式：(数据, 时间戳)，时间戳单位为纳秒
body_pose_joint_states() → list          # 获取整体姿态关节状态
head_joint_states() → (list, int)        # 获取头部关节状态 [yaw, pitch]
waist_joint_states() → (list, int)       # 获取腰部关节状态 [pitch, height]
arm_joint_states() → (list, int)         # 获取双臂 14 个关节状态
gripper_states() → (list, int)           # 获取夹持器状态 [left, right]
hand_joint_states() → (list, int)        # 获取手部 12 个关节状态
hand_force_states() → list               # 获取手部力传感器状态
whole_body_status() → （）                # 获取整机状态
```

7.1.2 指定时间戳查询

```
# 输入时间戳(纳秒)，返回最接近该时间戳的状态数据
head_joint_states_nearest(timestamp_ns) → list
arm_joint_states_nearest(timestamp_ns) → list
waist_joint_states_nearest(timestamp_ns) → list
gripper_joint_states_nearest(timestamp_ns) → list
hand_joint_states_nearest(timestamp_ns) → list
```

7.2 运动控制接口

7.2.1 基础运动控制

```
# 所有角度输入均使用弧度制
move_arm(positions)                      # 控制双臂运动，输入 14 个关节角度
move_gripper(positions)                  # 控制夹持器，输入范围[0,1]或[35,120]
move_hand(positions)                     # 控制手部 12 个关节运动
move_hand_as_gripper(positions)          # 将手部作为夹持器控制，输入范围[0,1]
move_head(positions)                     # 控制头部运动[yaw, pitch]
move_waist(positions)                    # 控制腰部运动[pitch(rad), height(cm)]
move_head_and_waist(head_pos, waist_pos) # 同时控制头部和腰部
```

```
move_wheel(linear, angular)            # 控制轮子运动，设置线速度和角速度
```

7.2.2 高级控制

```
# 重置机器人到初始或指定位置
reset(
    arm_positions=None,      # 手臂目标位置
    gripper_positions=None,  # 夹持器目标位置
    hand_positions=None,     # 手部目标位置
    waist_positions=None,    # 腰部目标位置
    head_positions=None      # 头部目标位置
)
```

7.3 其他功能

- `shutdown()` - 关闭 DDS 连接

7.4 使用示例

```
import time
import sys
from a2d_sdk.robot import RobotDds as Robot
# 初始化机器人
robot = Robot()
time.sleep(0.5) #等待资源初始化，收到消息

# 获取当前状态
arm_pos, timestamp = robot.arm_joint_states()
head_pos, timestamp = robot.head_joint_states()

# 基础运动控制
robot.move_head([0.0, 0.0])  # 控制头部运动
robot.move_gripper([0.5, 0.5])  # 控制夹持器

# 重置到初始位置
robot.reset()

# 关闭连接
robot.shutdown()

sys.exit(0)
```

7.5 注意事项注意事项

1. 手臂控制时需要按照前面所述要求进行控制

2. 所有角度输入均使用弧度
3. 腰部高度控制使用厘米为单位
4. 夹持器控制可使用归一化值 [0,1] 或原始值 [35,120]
5. reset 函数会执行平滑的轨迹规划

8. python 函数 RobotController 接口使用

RobotController 提供了异步运控 python 模块接口

8.1 接口列表

接口名称	接口作用	参数说明
set_motion_control_mode(mode)	设置运动控制模式	mode=0, stop 模式; mode=1, servo 模式; mode=2, planning 模式;
set_motion_stop()	全身急停/设置 stop 控制模式	

set_end_effector_pose_control(lifetime, control_group, left_pose=None, right_pose=None)	末端绝对位姿控制接口	1. lifetime 为指令有效时间，单位为秒，超过 lifetime 不发送下一帧指令则停止运动;
		2. control_group 类型为 list，是 ['left_arm','right_arm','dual_arm','left_arm_waist_lift',left_arm_waist_pitch',left_arm_waist','right_arm_waist_lift','right_arm_waist_pitch','right_arm_waist','dual_arm_waist_lift','dual_arm_waist_pitch','dual_arm_waist']的任意子集;
		3. left_pose 和 right_pose 均为 dict，分别为左右末端的位姿控制 dict
		dict 格式
		{
		'x':value_x,
		'y':value_y,
		'z':value_z,
		'qx':value_qx,
		'qy':value_qy,
		'qz':value_qz,
		'qw':value_qw,
		}
		left_pose 或者 right_pose 均可以为空

set_joint_position_control(lifetime, joint_group)	关节绝对位置控制接口	1. lifetime 为指令有效时间，单位为秒，超过 lifetime 不发送下一帧指令则停止运动；
		{
		'head_yaw': [0.0],
		'head_pitch': [0.0],
		'waist_pitch': [0.0],
		'waist_lift': [0.0],
		'left_tool': [0.0],
		'right_tool': [0.0],
		'left_arm': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
		'right_arm': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
		}
		根据控制关节的情况，dict 的 key 可以为空

get_motion_status()	获取运动控制状态	
trajectory_tracking_control((*self*, *infer_timestamp*, *robot_states*, *robot_actions*, *robot_link*="base_link", *trajectory_reference_time*=1.0))	轨迹跟踪控制命令	

8.2 接口说明

8.2.1 配置说明

```
CONTROL_TYPES = { #控制模式，需要从中挑选
    'ABS_POSE'    #笛卡尔坐标系下，末端绝对位姿控制，（当前版本接口暂不支持）
    'DELTA_POSE'  #笛卡尔坐标系下，末端相对位姿控制
    'END_SPEED'   #笛卡尔坐标系下，末端速度控制（当前版本暂不支持）
    'ABS_JOINT'   #关节空间下，绝对关节角控制
    'DELTA_JOINT' #关节空间下，相对关节角控制（暂未支持）
    'JOINT_SPEED' #关节空间下，关节速度控制（暂未支持）
    'PASSIVE'
}

GROUP_IDS = { # action 中的 key 值，即可以控制的部件
    'head'
    'left_arm'
    'right_arm'
    'gripper'
    'hand'
    'waist'
    'chassis'
```

```
}

robot_link = {
    'base_link' #底盘坐标系
    'arm_base_link' #胸部坐标系
}
```

8.2.2 初始化

```
from a2d_sdk.robot import RobotController
robot_controller = RobotController()
```

8.2.3 获取运动控制状态 get_motion_status()

```
def get_motion_status(self, timestamp=None):
        """获取运动控制状态

        Args:
            timestamp: 可选的时间戳(纳秒)，如果提供则返回最接近该时间戳的状态（最大范围为历史 1s）
                       如果不提供则返回最新状态

        Returns:
            包含运控状态信息的字典，如果没有状态则返回 None
            status = {
                'timestamp': status_timestamp, 时间戳
                'mode': {
                    'value': status_msg.mode, 控制模式的值
                    'name': mode, 控制模式的名字
                },
                'error': { 错误信息
                    'code': status_msg.error_code,
                    'message': status_msg.error_msg,
                    'has_error': status_msg.error_code != 0
                },
                'frames': {},包含了位姿等信息
                'collisions': []碰撞对
            }
        """
        return self._motion_controller.get_motion_control_status(timestamp)
```

运行下面代码可获取运动控制状态:

```
import time
from a2d_sdk.robot import RobotController
robor_controller = RobotController()
while True:
    motion_status = robor_controller.get_motion_status()
    print(motion_status)
    time.sleep(1)
```

可得

```
{
        'timestamp': 0,
        'mode': {
                'value': 1,
                'name': 'SERVO'
        },
        'error': {
                'code': 0,
                'message': '',
                'has_error': False
        },
        'frames': {
                'arm_left_link7': {
                        'position': {
                                'x': 0.13099616517341356,
                                'y': 0.8939011203073466,
                                'z': 0.853494291980361
                        },
                        'orientation': {
                                'quaternion': {
                                        'x': -0.7071067811710451,
                                        'y': -6.493375363146978e-06,
                                        'z': 1.2986693485271928e-06,
                                        'w': 0.7071067811710441
                                },
                                'euler': {
                                        'roll': -1.5707963268353753,
                                        'pitch': -7.346423698262328e-06,
                                        'yaw': 1.1019615310145943e-05
                                }
                        },
                        'xyzrpy': [0.13099616517341356, 0.8939011203073466,
0.853494291980361, -1.5707963268353753, -7.346423698262328e-06, 1.1019615310145943e-05]
                },
                'arm_right_link7': {
                        'position': {
                                'x': 0.13099881724075527,
                                'y': -0.8959988796513015,
                                'z': 0.8535006829930586
                        },
                        'orientation': {
                                'quaternion': {
                                        'x': 4.834221011001293e-06,
                                        'y': -0.7071041838347732,
                                        'z': 0.7071093785087194,
                                        'w': 2.236884963327504e-06
                                },
                                'euler': {
                                        'roll': -1.5707889804349158,
                                        'pitch': -1.000006746210147e-05,
                                        'yaw': -3.1415889804386605
                                }
                        },
                        'xyzrpy': [0.13099881724075527, -0.8959988796513015,
0.8535006829930586, -1.5707889804349158, -1.000006746210147e-05, -3.1415889804386605]
                }
        },
        'collisions': []
}
```

8.2.4 轨迹跟踪控制命令 trajectory_tracking_control( )

```
def trajectory_tracking_control(self, infer_timestamp, robot_states, robot_actions,
                                robot_link="base_link", trajectory_reference_time=1.0):
    """创建轨迹跟踪控制命令

    Args:
        infer_timestamp: Timestamp in nanoseconds
        robot_states: robot joint states by group when infer_timestamp, optional
type['waist', 'head', 'arm', 'chassis', 'gripper', 'hand']
        robot_actions: List of action dictionaries for trajectory
            - robot_action['part']['action_data']
            - robot_action['part']['control_type']
        robot_link: Link name of the robot
        trajectory_reference_time: Reference time for trajectory

    Returns:
        TrajectoryTrackingControl 消息
    """
    self._motion_controller.create_trajectory_tracking(
        infer_timestamp, robot_states, robot_actions,
        robot_link, trajectory_reference_time
    )
```

参考下面代码可实现绝对关节角控制条件下的轨迹跟踪:

```
from a2d_sdk.robot import RobotController,RobotDds

# 初始化机器人控制器
robot_controller = RobotController()
robot = RobotDds()
def execute(robot, robot_controller,action):
        robot_states = {}
        robot_actions = []
        robot_link = "base_link"
        trajectory_ref_time = 1.0
        wait_control_time = 0.5
        infer_timestamp = action["observation_timestamp"]
        robot_states["head"] = action["head_joint_states"]
        robot_states["waist"] = action["waist_joint_states"]
        robot_states["arm"] = action["arm_joint_states"]

        arm_joint_action = action["arm_cmd"]
        for i in range(len(arm_joint_action)):
            robot_action = {
                "left_arm": {
                    "action_data": arm_joint_action[i][:7],
                    "control_type": "ABS_JOINT"
                },
                "right_arm": {
                    "action_data": arm_joint_action[i][7:14],
                    "control_type": "ABS_JOINT"
                }
            }
            robot_actions.append(robot_action)
        time.sleep(1)
        robot_controller.trajectory_tracking_control(infer_timestamp,
```

```
                                                    robot_states,
                                                    robot_actions,
                                                    robot_link,
                                                    trajectory_ref_time)
        time.sleep(wait_control_time)
        print(f"Infer timestamp: {infer_timestamp}")
        print(f"Robot states: {robot_states}")
        print(f"Robot actions: {robot_actions}")
        while True:
            state, _= robot.arm_joint_states()
            print(f"Arm joint states: {state}")

if __name__ == "__main__":
    import time
    action = {
        "observation_timestamp": int(time.time() * 1e9),  # 当前时间戳（纳秒）
        "head_joint_states": [0.0, 0.0],
        "waist_joint_states": [0.5, 0.3],
        "arm_joint_states": [0] * 14,
        "arm_cmd":[[0.1]*14],

    }

    execute(robot,robot_controller, action)
    print("Done")
```

参考下面代码可实现末端相对位姿控制条件下的轨迹跟踪:

```
#!/usr/bin/env python3
import time
from a2d_sdk.robot import RobotController, RobotDds

def poll_state(getter, length, name, timeout=2.0, interval=0.1):
    deadline = time.time() + timeout
    last_vals = None

    while time.time() < deadline:
           vals, _ = getter()
        last_vals = vals
        # 获取长度
        try:
            actual_len = len(vals)
        except Exception:
            actual_len = None

        # 判断每个元素是否可转 float
        all_numeric = (actual_len == length) and all(
            _is_number(v) for v in vals
        )
        if all_numeric:
            print(f"✓ {name} 就绪")
            return list(vals)

        time.sleep(interval)

    raise RuntimeError(f"{name} 在 {timeout}s 内未就绪，最后 vals={last_vals!r}")

def _is_number(x):
```

```
    try:
        float(x)
        return True
    except Exception:
        return False

def execute(rc: RobotController, action: dict):
    """
    调用 SDK 下发相对位姿轨迹控制命令。
    """
    # 定义机器人状态字典
    robot_states = {
        "head":  action["head_joint_states"],
        "waist": action["waist_joint_states"],
        "arm":   action["arm_joint_states"],
    }
    # 定义机器人动作列表
    robot_actions = [
        {
            "left_arm":  {"action_data": delta[:6],  "control_type": "DELTA_POSE"},
            "right_arm": {"action_data": delta[6:12], "control_type": "DELTA_POSE"},
        }
        for delta in action["arm_cmd"]
    ]
    # 调用 SDK 下发相对位姿轨迹控制命令
    rc.trajectory_tracking_control(
        infer_timestamp      = action["observation_timestamp"],#必须有的时间
        robot_states         = robot_states,# 必须的参考状态，有了更好
        robot_actions        = robot_actions,
        robot_link           = "base_link",#基础坐标系
        trajectory_reference_time = 1.0,#运行的参考时间，越小运行的越快
    )

def main():
    rc = RobotController()
    rd = RobotDds()
    time.sleep(2.0)
    head_states  = poll_state(rd.head_joint_states,  length=2,  name="head_joint_states")
    waist_states = poll_state(rd.waist_joint_states, length=2,  name="waist_joint_states")
    arm_states   = poll_state(rd.arm_joint_states,   length=14, name="arm_joint_states")

    print(">>> 所有传感器数据就绪，开始下发相对位姿控制")
    try:
        while True:
            ts_ns = int(time.time() * 1e9)
            action = {
                "observation_timestamp": ts_ns,
                "head_joint_states":     head_states,
                "waist_joint_states":    waist_states,
                "arm_joint_states":      arm_states,
                "arm_cmd": [
                    [0.02, 0, 0, 0, 0, 0,   0.02, 0, 0, 0, 0, 0]
                ],
            }
            execute(rc, action)
            print(f">>> [{time.strftime('%H:%M:%S')}] 相对位姿控制命令已下发")
            time.sleep(1.0)  # 到下一次推理开始的等待时间
    except KeyboardInterrupt:
        print("\n>>> 收到中断，退出。")
```

```
    finally:
        rd.shutdown()

if __name__ == "__main__":
    main()
```

9. python 函数 CosineCamera 接口使用

9.1 初始化

```
from a2d_sdk.robot import CosineCamera as Camera
```

```
# 创建相机实例，指定要使用的相机组
camera = CosineCamera(camera_group: list[str])
[
    "head",
    "hand_left",
    "hand_right",
    "head_depth",
    "hand_left_fisheye",
    "hand_right_fisheye",
    "back_left_fisheye",
    "back_right_fisheye",
    "head_center_fisheye",
    "head_left_fisheye",
    "head_right_fisheye"
]
```

9.2 图像获取接口

9.2.1 实时图像获取

```
# 获取最新的图像和数据包
get_latest_image(camera_name: str) → np.ndarray     # 返回最新图像数据及时间戳
get_latest_packet(camera_name: str)                 # 返回最新原始数据包
```

9.2.2 指定时间戳获取

```
# 获取指定时间戳(纳秒)最近的图像和数据包
get_image_nearest(camera_name: str, timestamp_ns: int) → np.ndarray
get_packet_nearest(camera_name: str, timestamp_ns: int)
```

9.3 状态监控接口

```
# 获取相机性能统计信息
get_fps(camera_name: str)                                        # 获取当前帧率
get_latency_stats(camera_name: str, window_seconds: float=5.0) # 获取延迟统计
```

9.4 资源管理

```
close()  # 关闭相机连接并释放资源
```

9.5 使用示例

```
from a2d_sdk.robot import CosineCamera as Camera
# 初始化相机
cameras = ["head", "hand_left", "hand_right"]
camera = Camera(cameras)

# 获取实时图像
image, time_stamp = camera.get_latest_image("head")

# 获取指定时间戳的图像
timestamp_ns = int(time.time() * 1e9)  # 当前时间戳(纳秒)
image = camera.get_image_nearest("head", timestamp_ns)

# 监控相机状态
fps = camera.get_fps("head")
latency = camera.get_latency_stats("head", window_seconds=5.0)

# 使用完毕后关闭相机
camera.close()
```

9.6 注意事项

- get_latest_image 接口获得的第一帧数据为 None，需要从第二帧开始读取图像数据
- 时间戳统一使用纳秒为单位
- 图像数据以 numpy 数组格式返回
- 使用完毕后必须调用 close()释放资源
- 确保 camera_name 在初始化时指定的 camera_group 中存在
- get_latency_stats 的统计窗口默认为 5 秒

10. Python 函数 SLAM 导航定位接口

```
新增 Slam 类
from a2d_sdk.robot import Slam
slam = Slam()
```

10.1 建图模块

```
#开始建图时切换，1 为手动模式，2 为自动模式，使用建图工具建图需切换手动模式
slam.switch_nav_mode(1)

#1 开始建图，2 停止建图，True/False 是否保存地图，map_name 不能重复
slam.mapping_control(1, True/False, "map_name")

#停止建图
slam.mapping_control(2, True/False, "map_name")

#获取已经保存的地图列表及对应的 map_id
slam.get_map_list()

#根据 map_id 获取地图
slam.get_map(map_id)

#根据 map_id 删除地图
slam.delete_map(map_id)

#根据 map_id 切换地图
slam.switch_map(map_id)

#获取自动更新的最新的一张地图
slam.get_latest_map()
```

10.2 定位巡航

```
#建图后移动到指定地点，设置任务后需要切换至自动模式
slam.navigate_to_pose(x, y, theta)

#建图后移动到相对位置，设置任务后需要切换至自动模式
slam.move_to_relative(x, y, theta)

#建图后机器人重定位
slam.relocation(x, y, theta)

#设置底盘限速
slam.set_chassis_info(max_linear_speed, max_angular_speed, min_collision_distance)

#获取底盘速度信息
slam.get_chassis_info()

#获取底盘电池状态
```

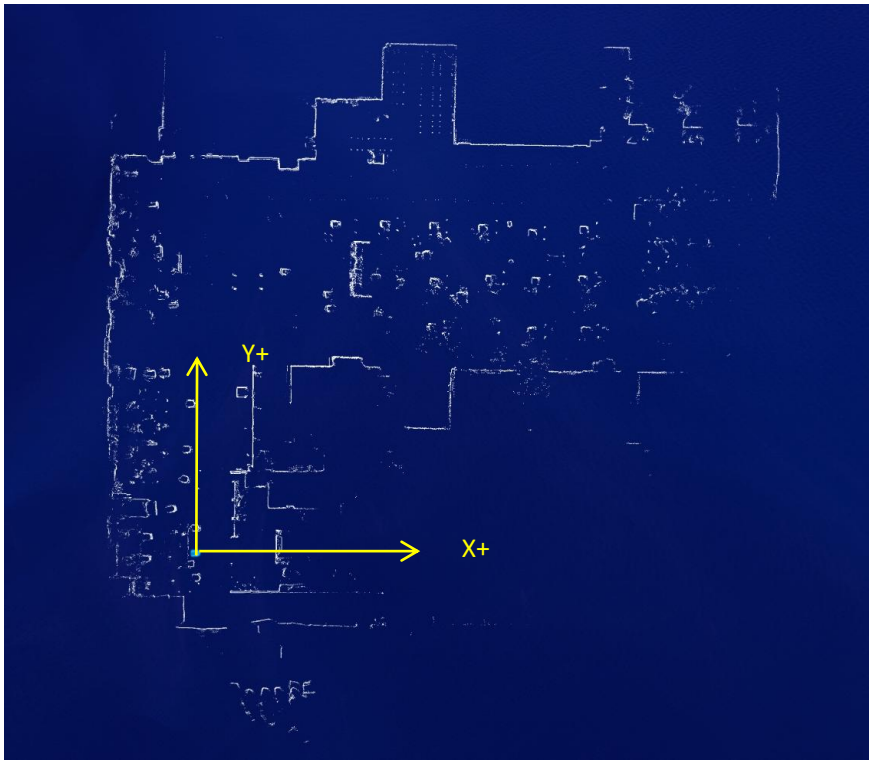
```
slam.get_chassis_battery_status()


#获取底盘状态，包括定位以及 IMU 等信息
slam.get_chassis_position()
```

10.3 避障设置

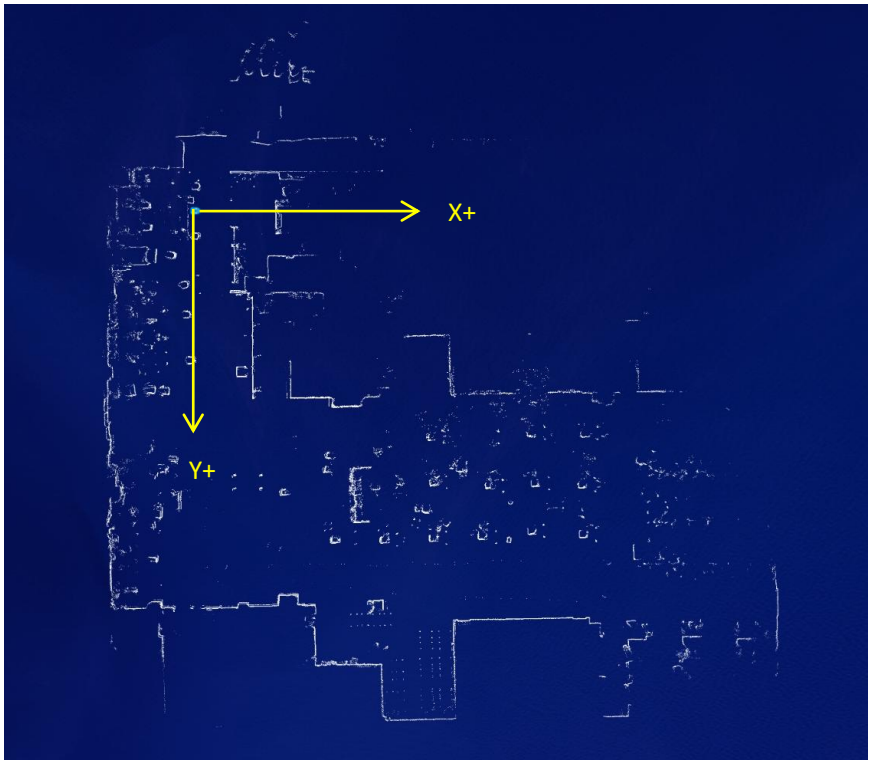
```
#设置底盘避障等级
slam.set_agv_col_level(level)

#设置是否开启避障
slam.set_agv_col_ctrl(True/False)

#设置机械臂避障等级
slam.set_arm_col_level(level)

#获取机械臂避障等级
slam.get_arm_col_level()
```

10.4 地图坐标系说明

HMI 中可以显示当前的地图，地图原点位置为建图起始位置的坐标，车头指向位置为 x 轴正方向，车体左侧为 Y 轴正方向。



通过接口获得的地图数据经过可视化后得到的图像，在 Y 轴方向为镜像。以上图为例。同时会将地图顶点相对于原点坐标发布。

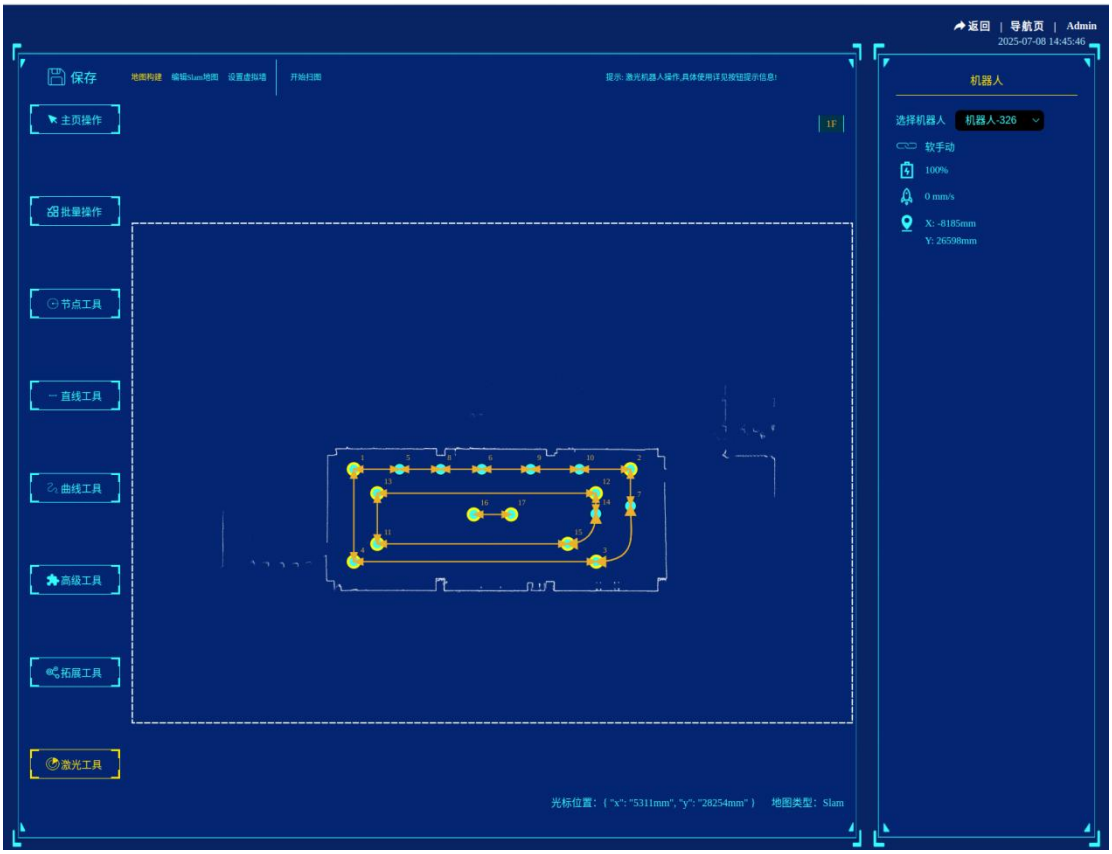


11. SLAM 与自主导航功能使用

11.1 常规建图

可以通过 web 页面进行可视化的建图工作 访问 10.42.0.153:8080，选择登录管理员，密码 Aa123789

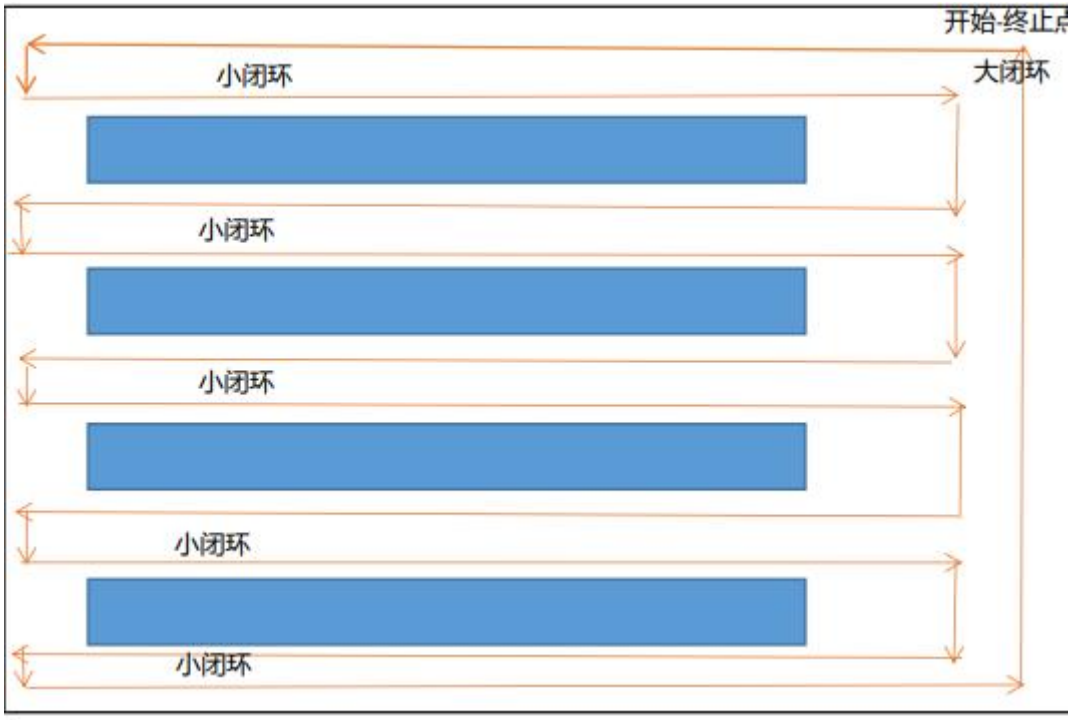
系统设计-地图管理-编辑地图



- 选择激光工具-地图构建，在右侧机器人属性面内，选择对应编号的机器人，点击开始扫图；
- 新建一个标签页，登录底盘后，进图 AMR 详情中，使用手动进行建图
- 扫描完成后点击停止扫图，待数据下载解析完成后，点击返回编辑按钮，保存并上传地图；

11.2 扫图中的注意事项

- 注意轮子是否存在打滑现象，因为轮子打滑会影响小车走的里程计
- 建图路线要形成闭环，先小闭环后大闭环（小闭环不得超过 40m）,如下图:



- 过低洼地带或者高低不平路面，小车需要很缓慢平滑过去，否则扫图的地图会失真
- 平整路线允许手动最大速度 0.5m/s
- 扫图过程中需要确认一下周围动态障碍物或者半静态障碍物
- 扫图过程中需要确认是否有长通道(25m)情况，若有看下是否增加一些固定的静态障碍物增　　加识别可靠性

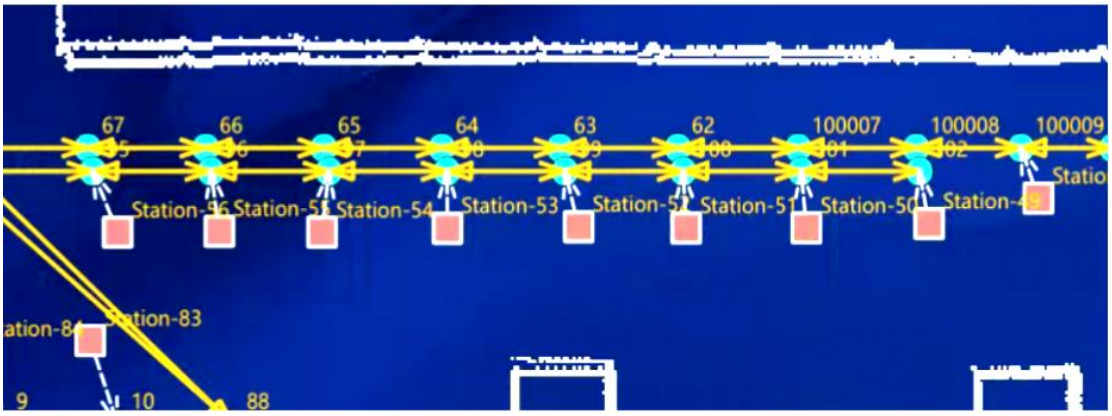
1. 扫图后注意事项

- 扫描完成后进行编辑 slam 地图
- 擦拭掉半静态障碍物：一天或者一周内会发生移动或者改变，如：物料|箱子等
- 擦拭掉动态障碍物：实时移动的物体如：人|车
- 固定障碍物无需擦除：长时间存在，不会发生变化，如：墙壁，大型设备机台
- 最后点击保存并且地图上传。

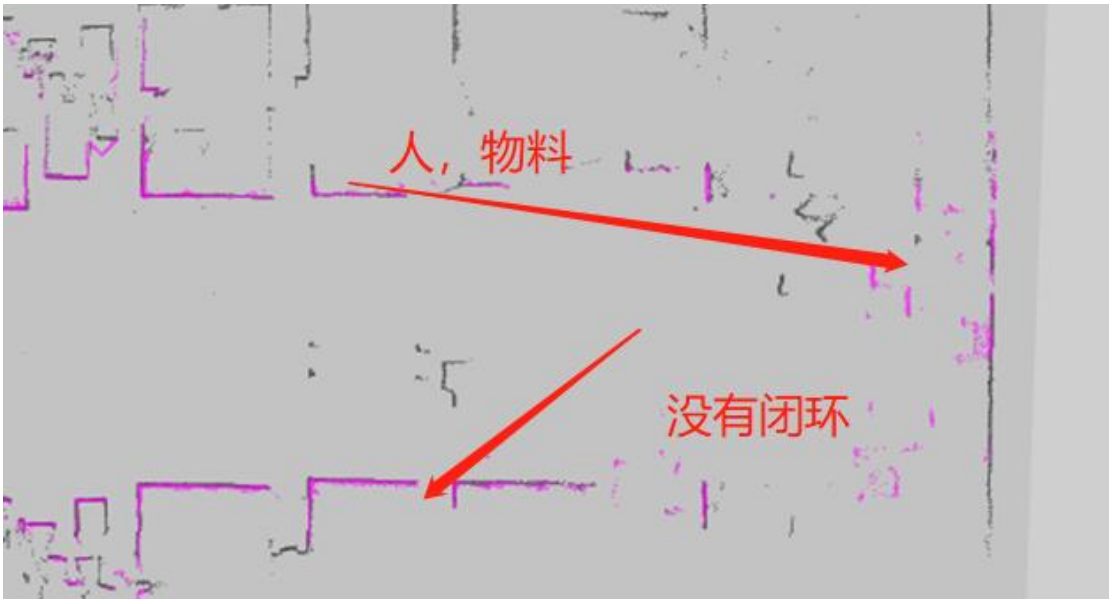
2. 地图确认工作

结合构建地图的错误案例，判断构建出来的地图是否可以投入使用

- 地图存在重影，地图不合格



- 地图没有闭环，只扫了一个方向，路线上动态障碍物没有去除。



- 地图和实际设备不匹配

11.3 自主导航功能使用

11.3.1 通过 HMI

在系统监控页面，单击任务信息列表右侧的加号，新增任务。可以根据实际需要，选择任务创建或者地图选点；

将底盘切换至自动模式，即可开始导航。

11.3.2 通过 ROS2 接口

在已经建图的基础上，通过对应 service，传入目的坐标和车头朝向。切换底盘工作模式后开启任务；

11.3.3 通过 Python 接口

在已经建图的基础上，参考 SLAM 接口用法，选择对应控制方式建立自主导航任务。

11.4 常见问题

11.4.1 切换为自动模式后，HMI 显示指令未接收

需要检查系统设置菜单中机器人组设置，机器人导航模式需要为自主导航模式才可执行导航点或者地图选点任务。更改设置后需要重启底盘。

12. 常见问题处理

12.1 执行模式切换无反应等跟 sdk 相关的报错

请先执行

```
source env.sh
```

12.2 执行相关控制命令异常

请检查机器人的头腰消息是否缺失，头腰的消息均为两个电机。

如果缺失则为机器人电机控制器状态机异常，注意，此问题为供应商驱动问题，需要在启动机器人后不要立刻切换模式。

解决此问题操作为重启机器人，启动一分钟后再切换模式。