
Magicbot-Gen1 SDK Development Documentation

发布 1.2.2-doc2

MagicLab

2025 年 12 月 17 日

关于 MagicBot-Gen1:

1 概述	1
1.1 产品 SKU	1
1.2 电气接口	4
1.3 机载计算机	4
1.4 关节顺序名称与关节限位	5
1.5 激光雷达	6
1.6 深度相机	6
1.7 机器人规格	6
2 遥控器按键说明	7
2.1 功能说明	7
2.2 模式切换	8
2.3 遥控器检查	8
2.4 遥控器开启与关闭	9
2.5 开机时遥控器操作	9
2.6 行走模式切换	10
2.7 停止并返回站立状态	10
2.8 急停	10
2.9 关机时遥控器操作	11
3 SDK 概述	13
3.1 SDK 通信接口介绍	13
3.2 获取 SDK	14
4 软硬件架构	15
4.1 软件系统架构图	16
4.2 硬件系统架构图	17

5 服务介绍	19
6 快速开始	21
6.1 系统环境	21
6.2 网络环境	22
6.3 安装与编译	26
6.4 C++ 例程示例	27
6.5 Python 例程示例	28
6.6 其他	32
7 机器人主控制服务 C++	33
7.1 接口定义	33
8 高层运动控制服务 C++	39
8.1 接口定义	39
8.2 类型定义	42
8.3 枚举类型定义	43
8.4 遥控器示意图	44
8.5 头部运动范围示意图	44
8.6 高层运动控制机器人状态介绍	45
8.7 高层运动控制接口	45
9 底层运动控制服务 C++	47
9.1 接口定义	47
9.2 类型定义	53
9.3 URDF 参考	57
9.4 底层运动控制机器人状态介绍	57
9.5 注意事项	57
10 传感器控制服务 C++	59
10.1 接口定义	59
10.2 类型定义	68
10.3 注意事项	70
11 语音控制服务 C++	71
11.1 接口定义	71
11.2 类型定义	76
11.3 结构体定义	76
11.4 DOA	77
11.5 注意事项	78
12 状态监控服务 C++	79
12.1 接口定义	79
12.2 错误码映射表	82

13 SLAM 导航控制服务 C++	85
13.1 接口定义	85
13.2 类型定义	93
13.3 枚举类型定义	94
13.4 SLAM 导航控制介绍	95
14 机器人主控制服务 Python	97
14.1 接口定义	97
15 高层运动控制服务 Python	103
15.1 接口定义	103
15.2 类型定义	106
15.3 枚举类型定义	107
15.4 遥控器示意图	108
15.5 头部运动范围示意图	108
15.6 高层运动控制机器人状态介绍	109
15.7 高层运动控制接口	109
16 底层运动控制服务 Python	111
16.1 接口定义	111
16.2 类型定义	116
16.3 URDF 参考	120
16.4 底层运动控制机器人状态介绍	120
16.5 注意事项	121
17 传感器控制服务 Python	123
17.1 接口定义	123
17.2 类型定义	132
17.3 注意事项	134
18 语音控制服务 Python	135
18.1 接口定义	135
18.2 类型定义	140
18.3 DOA	141
18.4 注意事项	141
19 状态监控服务 Python	143
19.1 接口定义	143
19.2 错误码映射表	146
20 SLAM 导航控制服务 Python	149
20.1 接口定义	149
20.2 类型定义	157
20.3 枚举类型定义	158

20.4 SLAM 导航控制介绍	159
21 高层运动控制示例	163
21.1 C++	163
21.2 Python	171
21.3 运行说明	184
22 底层运动控制示例	187
22.1 C++	187
22.2 Python	190
22.3 运行说明	199
23 传感器控制示例	201
23.1 C++	201
23.2 Python	214
23.3 运行说明	231
24 语音控制示例	233
24.1 C++	233
24.2 Python	241
24.3 运行说明	253
25 状态监控示例	255
25.1 C++	255
25.2 Python	257
25.3 运行说明	260
26 SLAM 导航示例	263
26.1 C++	263
26.2 Python	290
26.3 运行说明	326
27 FAQ	329
27.1 开发准备环境	329
27.2 支持无线开发吗?	329
27.3 目前底层控制接口通讯频率是多少?	329
27.4 SDK 无法订阅和发布话题数据?	329
27.5 SDK 订阅话题数据频率不稳定?	330
27.6 SDK 接口调用报错: Deadline Exceeded	330
27.7 SDK 内部配置和日志如何查看?	330
27.8 视频推流	331
27.9 SLAM 重定位失败?	331
27.10 SLAM 导航不执行?	331
27.11 运行报错: Invalid IP address	332

27.12 日志报错打印: Call of pthread_setschedparam with insufficient privileges! . .	332
27.13 connect 返回报错: Failed to connect to robot, code: ErrorCode.SERVICE_ERROR, message: failed to connect to all addresses; last error: UNKNOWN: ipv4:192.168.54.119:50051: Failed to connect to remote host: Connection refused	332
27.14 如何查看机器本体软件版本?	332
27.15 执行特技报错: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 优先级过低	335
27.16 执行特技报错: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 危险动作	335
27.17 执行特技报错: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 调用 ROS2 服务失败	335

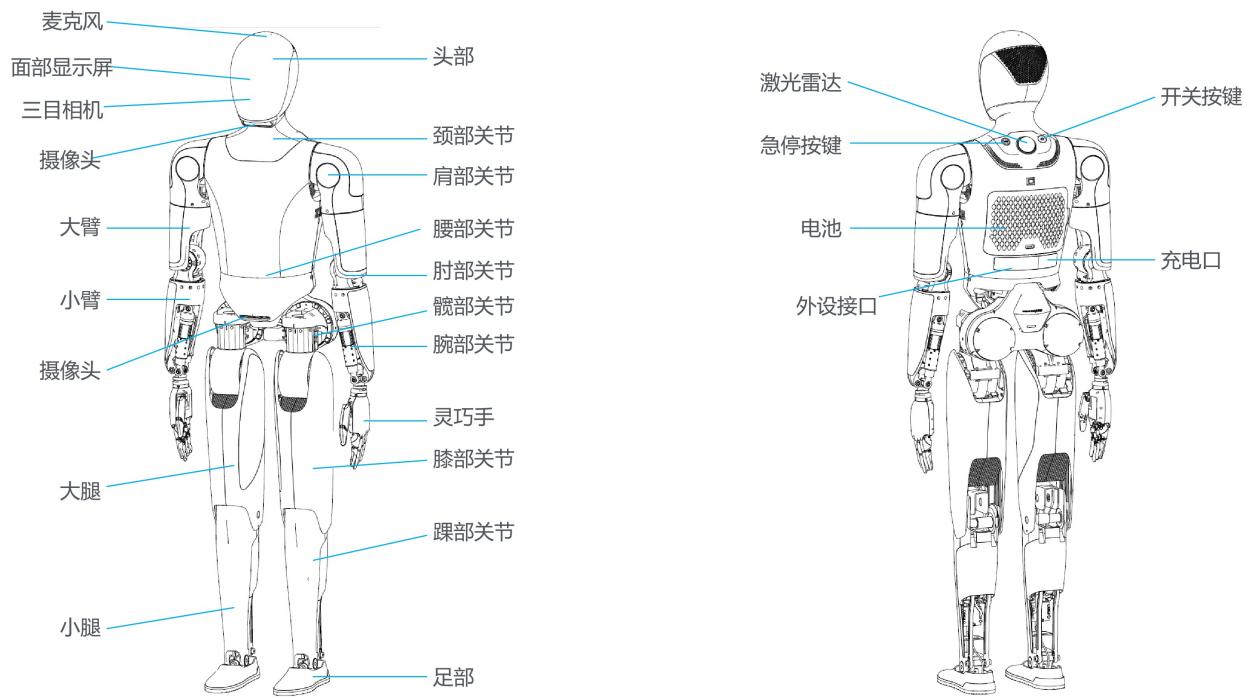
CHAPTER 1

概述

1.1 产品 SKU

MagicBot Gen1 采用优秀的仿生人形外观设计，全身具有多个关节自由度，搭配 360° 激光雷达，高清相机等多传感器，可实现多模态拟人交互。单手臂拥有 7 个自由度，包括肩关节俯仰、肩关节滚动、肩关节偏航、肘关节滚动、肘关节侧展、腕关节俯仰、腕关节侧展。单腿拥有 6 个自由度，包括髋关节滚动、髋关节偏航、髋关节俯仰、膝关节俯仰、踝关节俯仰和踝关节滚动。头部具备 2 个自由度，包括颈关节俯仰和颈关节滚动。腰部具备 2 个自由度，包括腰关节俯仰和腰关节滚动。整机共有 52 个自由度(含被动自由度)，确保机器人能够实现高灵活性的运动和精准的姿态控制。

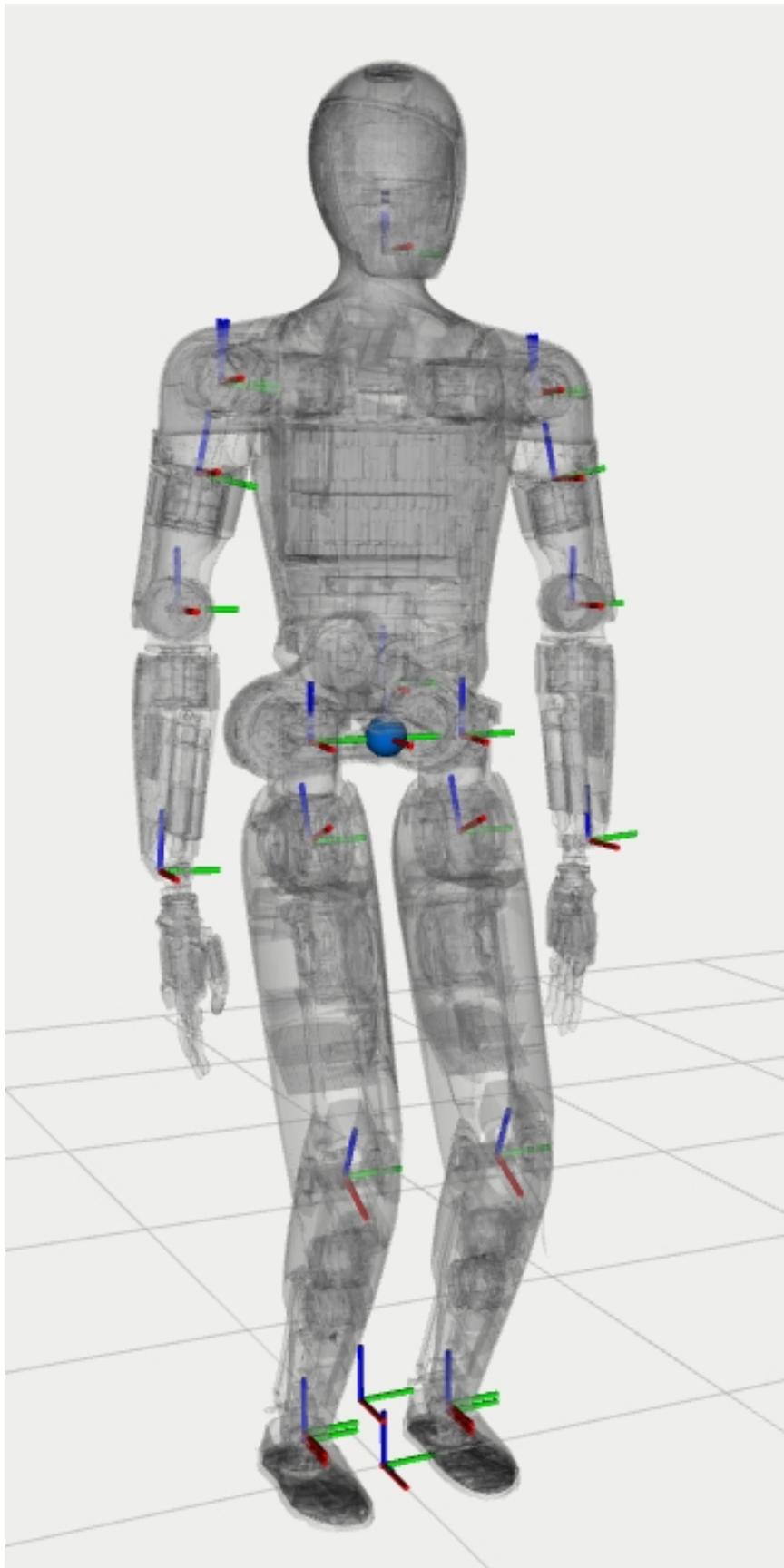
下图为传感器布局示意图，具体请以实物为准。



系统各个部位的自由度如下表所示：

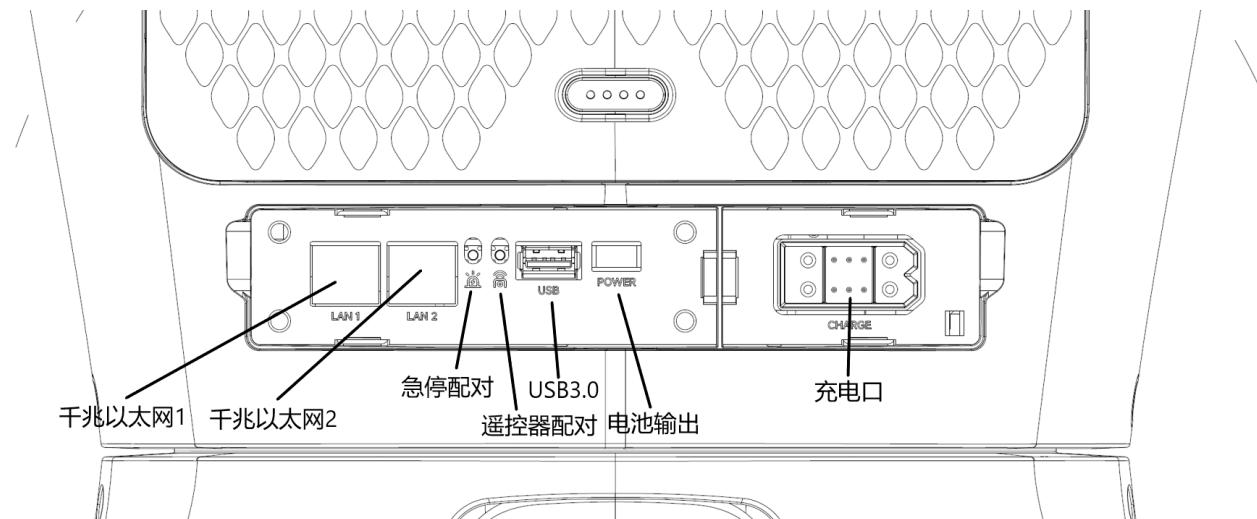
关节名称	自由度
头部	2
手臂	7×2
灵巧手	11×2
腰部	2
腿	6×2
合计	52

当各个关节均为零度时，各坐标系如下图所示。其中红色为 x 轴，绿色为 y 轴，蓝色为 z 轴。



1.2 电气接口

MagicBot Gen1 后腰处(电池盖下方)预留了多种类外设接口, 可进行设备调试、二次开发、挂载外设等多种操作。



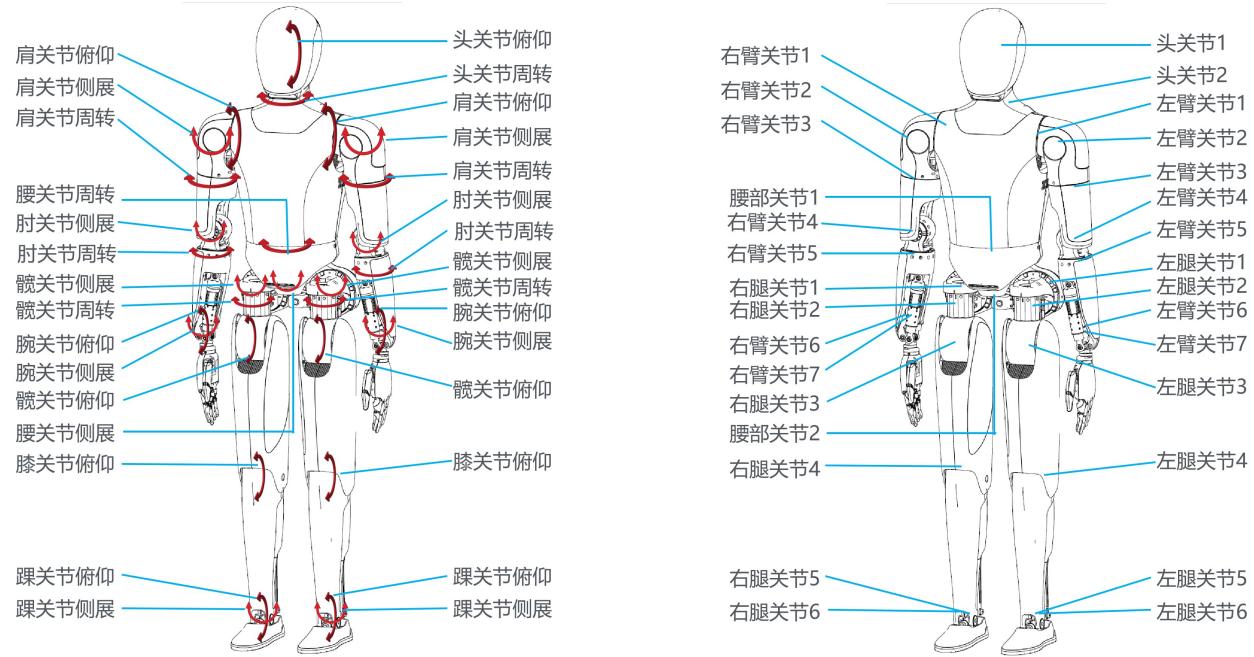
1.3 机载计算机

MagicBot Gen1 机载标配 1 块【运控计算单元】, 并且标配 1 块【开发计算单元】。

参数	开发计算单元
型号	Jetson Orin NX
CPU	Arm® Cortex®-A78AE
内核数	8
线程数	8
最大睿频频率	2GHz
显存	16G
内存	16G
缓存	2MB L2 + 4MB L3
存储	512G
GPU	搭载 32 个 Tensor Core 的 1024 核 NVIDIA Ampere 架构 GPU
显卡最大动态频率	918MHz
高斯和神经加速器	3.0
指令集	64bit
OpenGL	4.6
OpenGL ES	3.2
Vulkan™	1.1
CUDA	11.4

1.4 关节顺序名称与关节限位

系统的具体关节及编号如下：



所属部位	名称	运动范围
头部	头关节周转 Head Yaw	$\pm 60^\circ$
头部	头关节俯仰 Head Pitch	$\pm 20^\circ$
手臂	肩关节俯仰 Shoulder Pitch	$\pm 158^\circ$
手臂	肩关节侧展 Shoulder Roll	-10~165°
手臂	肩关节周转 Shoulder Yaw	$\pm 150^\circ$
手臂	肘关节侧展 Elbow Roll	-10~127°
手臂	肘关节周转 Elbow Yaw	$\pm 167^\circ$
手臂	腕关节俯仰 Wrist Pitch	$\pm 26^\circ$
手臂	腕关节侧展 Wrist Roll	-63~49°
腰部	腰关节转动 Waist Yaw	$\pm 90^\circ$
腰部	腰关节侧摆 Waist Roll	$\pm 20^\circ$
腿部	髋关节侧展 Hip Roll	-20~+30°
腿部	髋关节周转 Hip Yaw	-15~+30°
腿部	髋关节俯仰 Hip Pitch	-120~+40°
腿部	膝关节俯仰 Knee Pitch	0°~+150°
腿部	踝关节俯仰 Ankle Pitch	-75~+30°
腿部	踝关节侧展 Ankle Roll	-20~+20°

1.5 激光雷达

MagicBot Gen1 后颈部搭载了 MID-360 激光雷达，其以小型化、低功耗特性提供 360° 无死角感知，精准获取环境数据，助力建图导航与避障，可与视觉互补适配复杂场景。

1.6 深度相机

MagicBot Gen1 下颌处和腰部搭载 D435 深度相机，该相机视野宽广，能全面捕捉环境信息，全局快门适配动态场景，低光环境表现稳定，且深度感知精准，支持实时环境建模与避障，小巧易集成，使机器人能智能地与周围环境进行交互。

1.7 机器人规格

项目名称	规格参数
高宽厚 (站立)	174cm × 58cm × 28 cm
双臂臂展	160 cm
大腿 + 小腿	95 cm
整机重量	约 70 kg
电池	容量 25Ah (1.35kWh)
运动速度	≥2 m/s
双臂最高搬运负载	15kg
最大关节扭矩	≥350N · m
最大斜坡角度	≥20°
灵巧手	11 自由度触觉灵巧手 (选配)
单手臂自由度	7 (肩关节 ×3 + 肘关节 ×1 + 腕关节 ×3)
单腿自由度	6 (髋关节 ×3 + 膝关节 ×1 + 踝关节 ×2)
算力模块	基础算力：8 核高性能 CPU 感知算力：NVIDIA Jeston Orin
传感器配置	3D 激光雷达、深度相机、三目相机
音频模块	环形麦克风、扬声器
通讯模块	5G 模组 (选配)、Bluetooth 5.2、WIFI6
减速器	行星减速器、谐波减速器

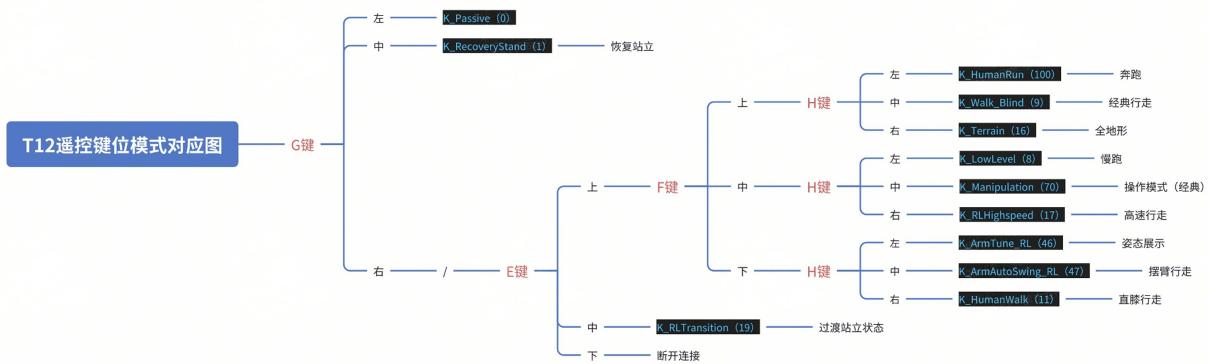
CHAPTER 2

遥控器按键说明

2.1 功能说明

参数	开发计算单元
遥控器开机	长按 3 秒，一键开机，且无需人为干预
遥控器关机	长按 3 秒，一键关机，且无需人为干预
充电	USB-C 直插快充
APP 配对	通过手机 APP 配对
机器人状态	阻尼，recover stand, RL 模式之间切换 切换
机器人步态	ArmAutoSwin 47，行走 baseline k_lowlevel 8，ArmTune 46 之间切换 切换
机器人特技	张手、回手、转身、回身之间切换 切换
手机和遥控器交互	支持将手机夹持到遥控器上 1. 遥控器优先级高于 APP：如果同时用遥控器和 APP 操控，则只能遥控器控制 2. 支持分别控制机器人
拆卸和收纳	遥控器摇杆可拆卸和收纳

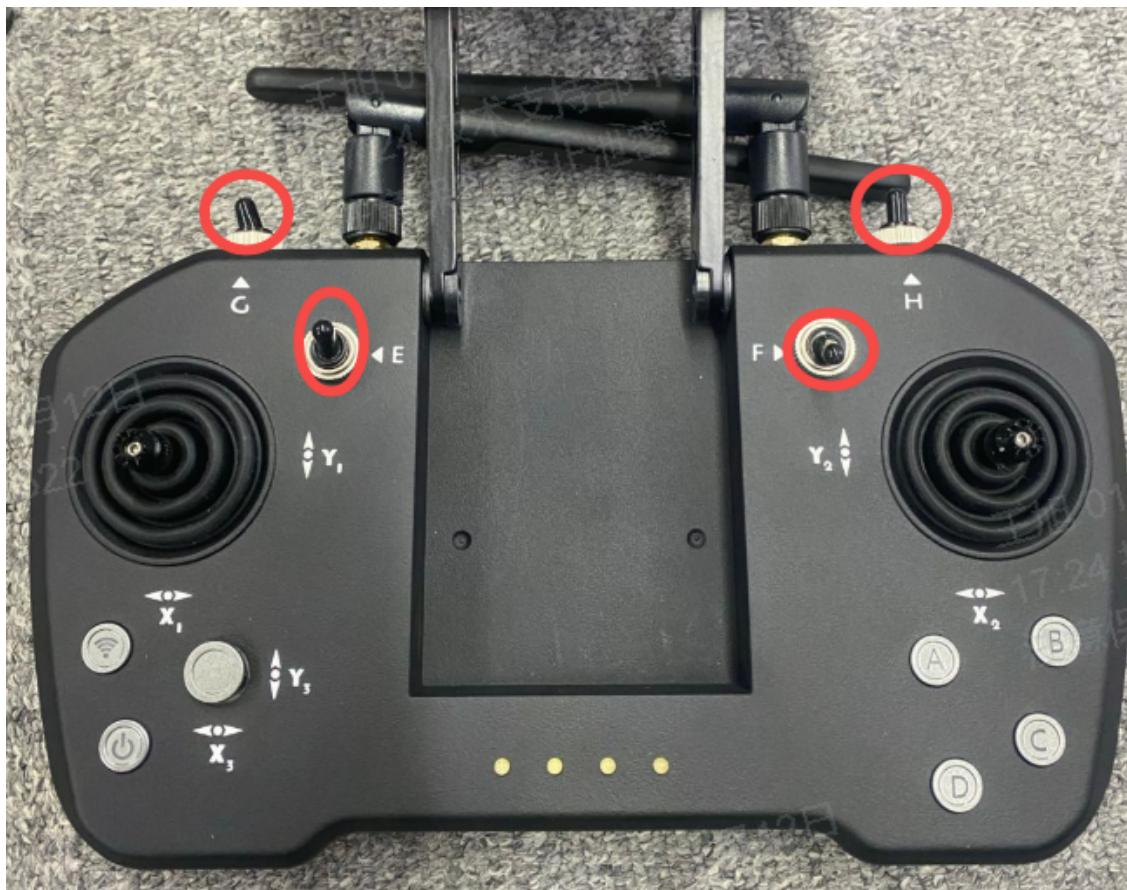
2.2 模式切换



2.3 遥控器检查

- 检查遥控器摇杆是否都在中位，摇杆是否进入沙土等异物；
- 检查遥控器的各个按键是否存在卡顿；

- 将遥控器各拨杆恢复至初始状态，并确认遥控器电池是否电量充足，具体操作步骤如下：
 - 如图所示，将遥控器上的电源键先短按后长按，打开遥控器；
 - 此时遥控器下方中央的指示灯亮起，从左至右分别表示遥控器的电量状态。当亮起第 1、2、3、4 个灯时，对应的电量分别为 25%、50%、75%、100%；
 - 如使用遥控器遥控方式，遥控器开机前需将各控制拨杆置于初始状态，“G” 拨动至最左侧，“E” 拨杆拨动至最上侧，“F” 拨杆拨动至最下侧，“H” 拨杆拨动至中间侧。



2.4 遥控器开启与关闭

- 电源按键先短按一次再长按，待电量指示灯亮起即开机成功；
- 电源按键先短按一次再长按，待电量指示灯熄灭即关机成功。

2.5 开机时遥控器操作

- 短按一下长按开关键，指示灯亮灯后，遥控机开机成功；
- 机器人必须固定在保护支架上，将遥控器上的“G” 拨杆由左侧拨动至中间；
- 观察到机器人腿部和手部均有小幅度关节位移，四肢会缓慢进入站立状态

- 控制保护支架缓慢下降，下放机器人站立在地面，站立后，观察机器人的站立是否平衡，如无抖动、前后左右倾倒等异常则站立平衡；



2.6 行走模式切换

- 确认机器人已处于站立状态并且落置于地面上；
- 将遥控器上的”G”拨杆往最右侧拨动，”H”拨杆拨动至中间，”F”拨杆拨动至最下侧，进入摆臂行走模式，此模式下，机器人步态沉稳；
- 将遥控器上的”G”拨杆往最右侧拨动，”H”拨杆拨动至右侧，”F”拨杆拨动至最下侧，进入拟人行走模式，此模式下，机器人步态轻盈。

2.7 停止并返回站立状态

在摆臂行走模式和拟人行走模式下：

- 遥控机器人保持停止状态；
- 将遥控器上的”G”拨杆往中间拨动，机器人进入站立状态。

注意：切勿在行走过程中直接将遥控器上的”G”拨杆往中间或最左侧拨动；

2.8 急停

遇到以下紧急情况，需要及时将遥控器上的”G”拨杆拨动至最左侧，会让机器人的关节失能。此时关节断电，机器人无法保持任何姿态，会立即瘫软并倒下：

- 机器人冒烟或有焦味；
- 有水或其他异物进入机器人；
- 机器人失控，无法通过遥控器或其他方式停止；
- 机器人损坏，可能导致进一步危险；
- 紧急避险，如高空作业或危险环境中失控；

- 外部环境突变，如地震、火灾等。

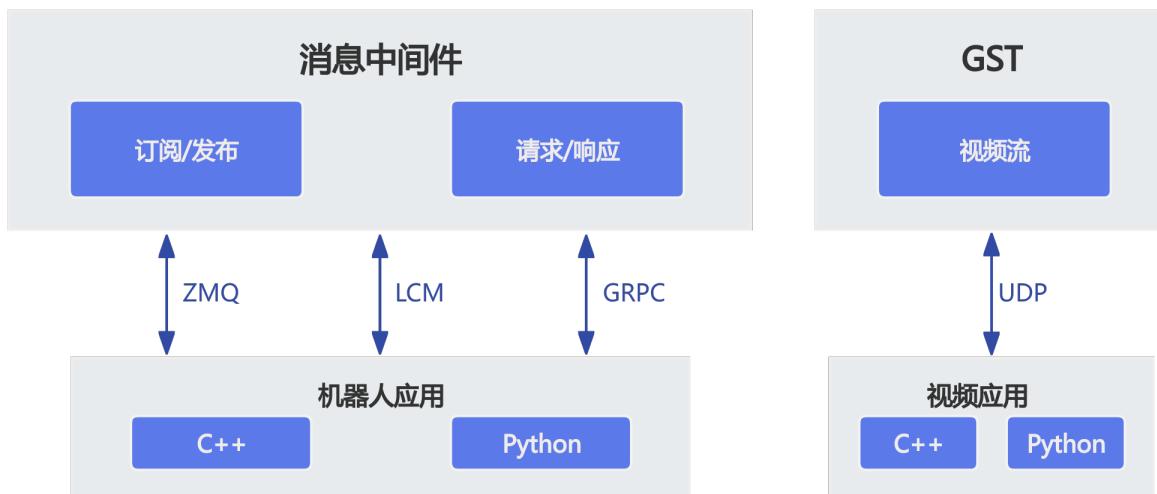
注意：将遥控器上的”G”拨杆拨动至最左侧之前，机器人必须固定在保护支架上。

2.9 关机时遥控器操作

- 确认将遥控器上的”G”拨杆往中间拨动；使机器人进入站立状态；
- 将机器人固定在支架上，并向上吊起；
- 长按机器人背部开关机按键（约 6 秒）电源状态灯熄灭，此时机器人关机断电；

SDK 概述

3.1 SDK 通信接口介绍



- 当前软件版本暂不支持 GST 视频流传输

Gen1 采用 LCM/GRPC/UDP/ZMQ 作为消息中间件，主要数据交互模式：话题（订阅/发布）和 RPC（请求/响应）

- **话题（订阅/发布）：**接收方订阅某个消息，发送方根据订阅列表向接收方发送消息，主要用于中高频或持续的数据交互。

- **RPC (请求/响应)**: 问答模式，通过请求实现数据获取或操作。用于低频或功能切换时的数据交互。
话题和 RPC 接口的主要调用方式：函数式接口
- **函数式接口**: 将 API 调用封装成函数调用，方便用户使用。

3.2 获取 SDK

magicbot-gen1_sdk 是魔法原子新一代机器人开发 SDK。SDK 将高层运动控制、底层电机控制、语音控制等接口进行封装，并提供相关的函数式接口。您可以参考我们提供的 SDK 教程，学习机器人控制，完成 gen1 的二次开发；

3.2.1 SDK 下载地址：

magicbot-gen1_sdk

SDK 版本与机器本体软件版本对齐记录，请参考：[ChangeLog](#)

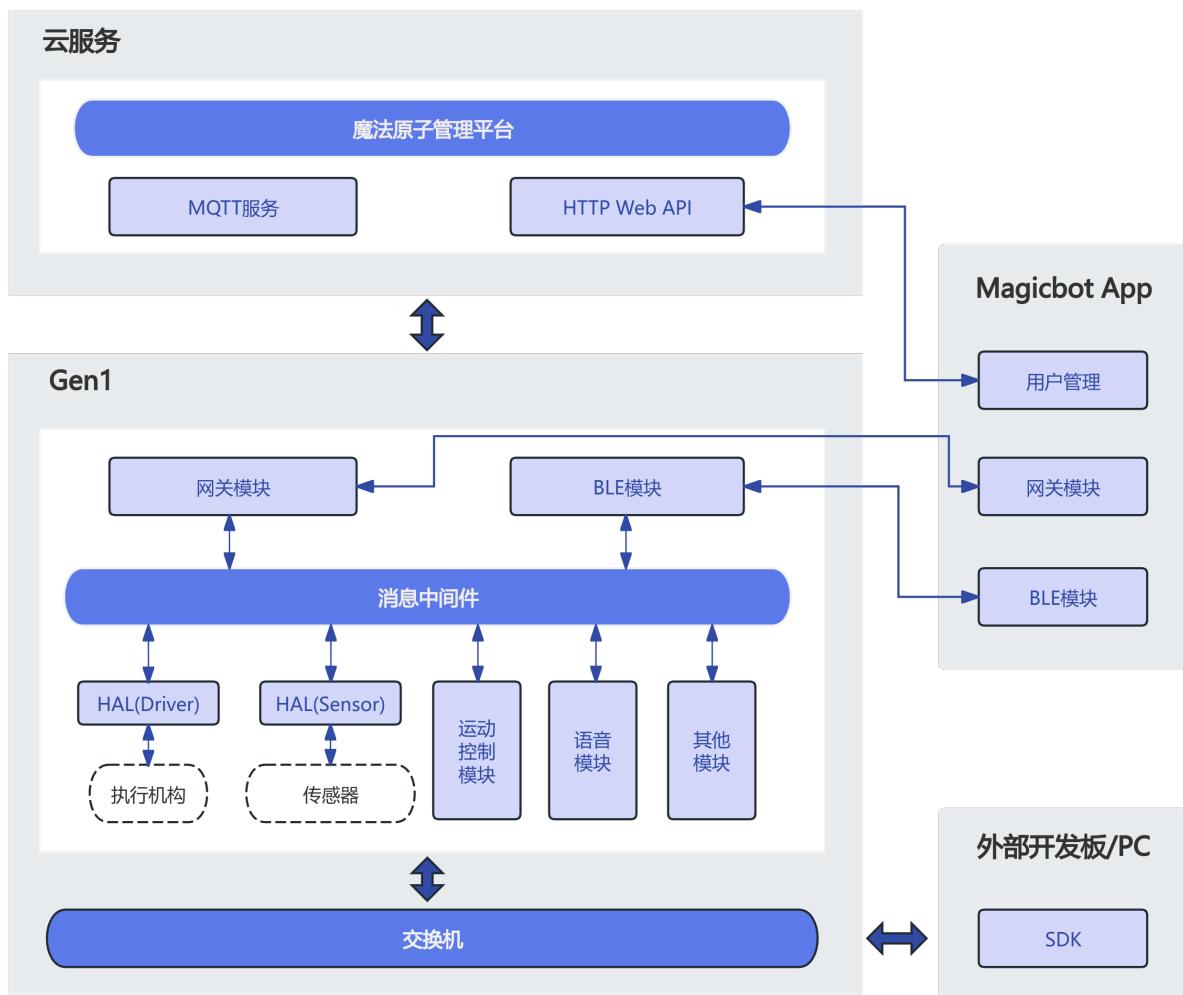
3.2.2 URDF/MJCF 下载地址：

URDF/MJCF

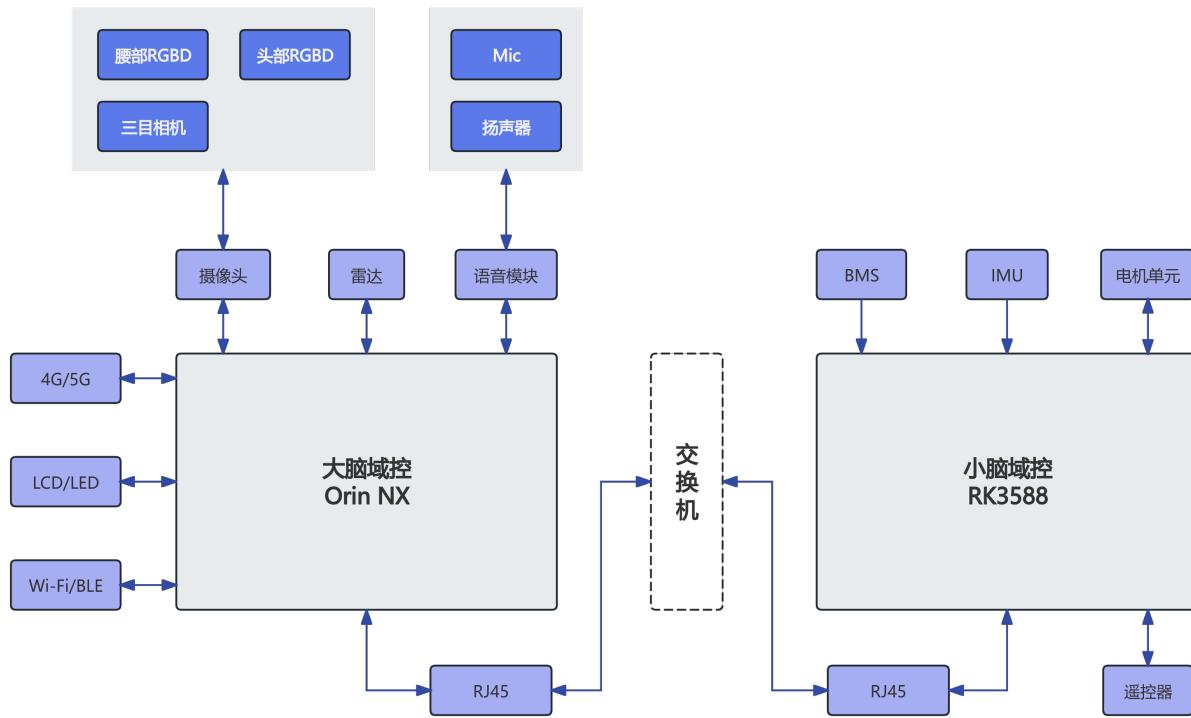
CHAPTER 4

软硬件架构

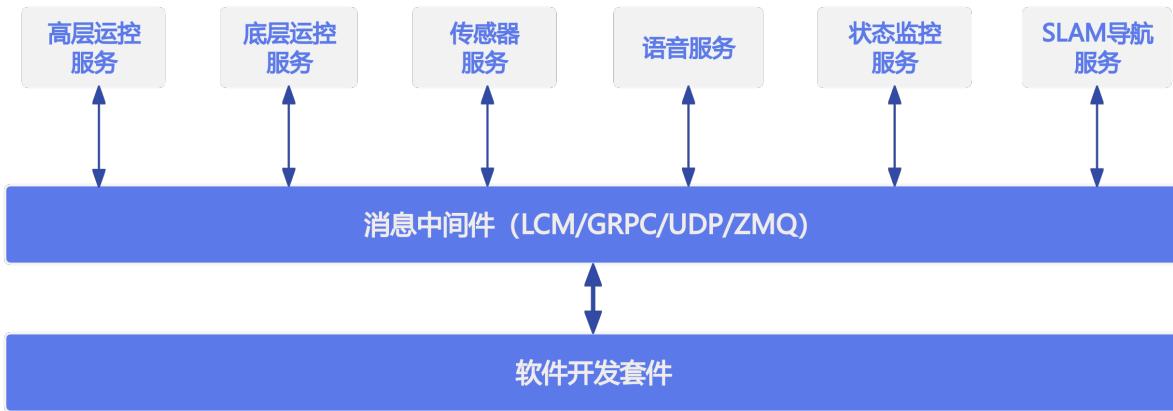
4.1 软件系统架构图



4.2 硬件系统架构图



服务介绍



Magicbot Gen1 通过消息中间件（LCM/GRPC/UDP/ZMQ）提供如下服务：

- **高层运控服务**

基于 Gen1 内置步态控制器，可进行步态切换、特技执行、控制姿态和速度（等价遥控器）等功能。高层运控服务使用 GRPC 进行通信。

- **底层运控服务**

通过服务接口可实时获取关节、IMU 等数据，同时可实时下发关节指令控制电机。底层运控服务使用 LCM 进行通信。

- **语音服务**

通过服务接口可控制音量和语音控制。语音服务使用 GRPC/LCM 进行通信。

- **传感器服务**

支持 lidar、rgbd、三目相机等传感器的数据订阅。传感器服务使用 GRPC/UDP/LCM/ZMQ 进行通信。

Notice: 当前版本仅支持 lidar 数据订阅，rgbd/三目相机数据暂不支持

- **状态监控服务**

通过服务接口可订阅机器人本体软硬件故障和 bms 状态。状态监控服务使用 LCM 进行通信。

- **SLAM 导航服务**

通过服务接口可进行机器人 SLAM 建图和导航控制。SLAM 导航服务使用 GRPC/LCM 进行通信。

CHAPTER 6

快速开始

6.1 系统环境

推荐在 ubuntu22.04 系统下开发，暂不支持 Mac/Windows 系统下开发，机器人本体 PC 运行官方服务，不支持开发；

APP 和 SDK 不支持同时控制机器人

6.1.1 开发环境要求

- GCC ≥ 11.4 (for Linux)
- CMake ≥ 3.16
- Make build system
- C++20 (minimum)
- Eigen3
- python3.10

6.1.2 系统配置

首先，为了实现普通用户下通信实时性，需要在 /etc/security/limits.conf 文件中增加如下配置：

```
* - rtprio 98
```

其次，为了增加每个 socket 链接的接收缓存，需要在 /etc/sysctl.conf 文件中增加如下配置，`sudo sysctl -p` 立即生效或重启生效：

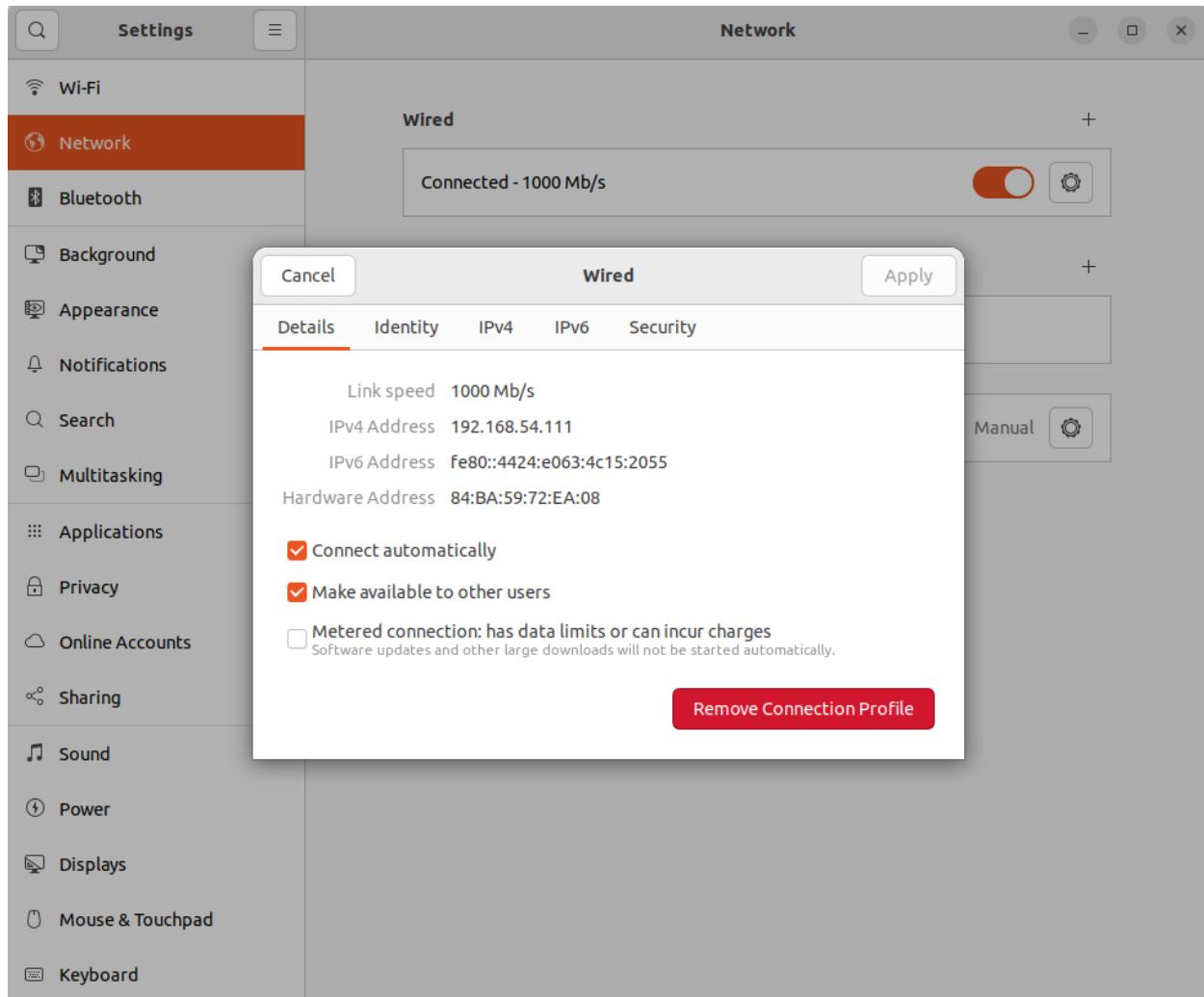
```
net.core.rmem_max=20971520  
net.core.rmem_default=20971520  
net.core.wmem_max=20971520  
net.core.wmem_default=20971520
```

6.2 网络环境

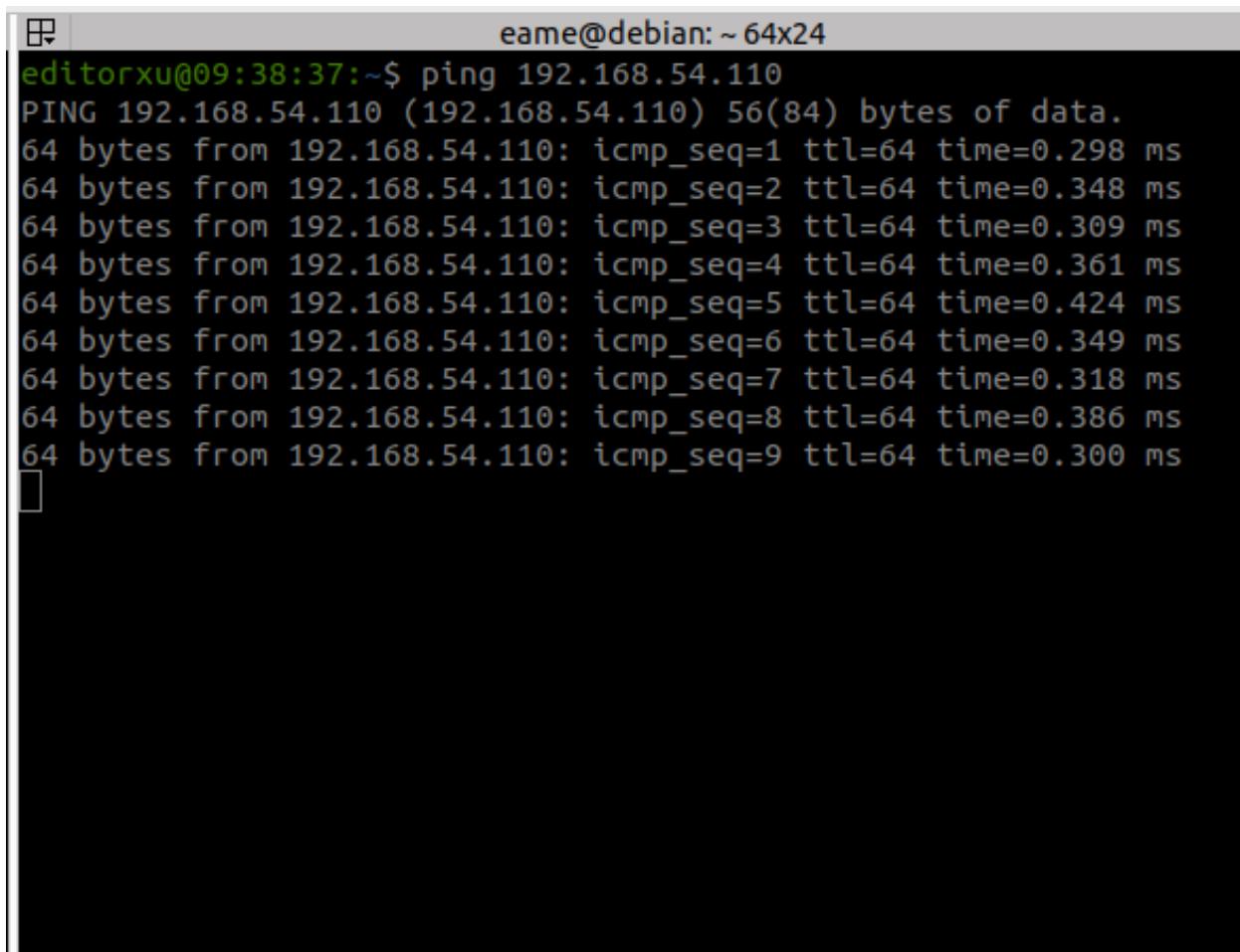
将用户计算机与机器人交换机接入统一网络。建议新用户使用网线将用户计算机接入机器人交换机，并将与机器人通信的网卡设置在 192.168.54.X 网段下，推荐使用 192.168.54.111。有经验的用户可自行配置网络环境。

6.2.1 配置步骤

1. 用网线的一端连接机器人，另一端连接用户电脑。机器人工载电脑的 IP 地址为 192.168.54.110（小脑算力板）和 192.168.54.119（大脑算力板），所以需要将电脑 ip 设置为同一网段，建议 192.168.54.111.



2. 为了测试通信连接是否正常，可以通过 ping 进行测试：



```
eame@debian: ~ 64x24
editorxu@09:38:37:~$ ping 192.168.54.110
PING 192.168.54.110 (192.168.54.110) 56(84) bytes of data.
64 bytes from 192.168.54.110: icmp_seq=1 ttl=64 time=0.298 ms
64 bytes from 192.168.54.110: icmp_seq=2 ttl=64 time=0.348 ms
64 bytes from 192.168.54.110: icmp_seq=3 ttl=64 time=0.309 ms
64 bytes from 192.168.54.110: icmp_seq=4 ttl=64 time=0.361 ms
64 bytes from 192.168.54.110: icmp_seq=5 ttl=64 time=0.424 ms
64 bytes from 192.168.54.110: icmp_seq=6 ttl=64 time=0.349 ms
64 bytes from 192.168.54.110: icmp_seq=7 ttl=64 time=0.318 ms
64 bytes from 192.168.54.110: icmp_seq=8 ttl=64 time=0.386 ms
64 bytes from 192.168.54.110: icmp_seq=9 ttl=64 time=0.300 ms
```

3. 查看 192.168.54.111 网址对应的网卡名称，通过 `ifconfig` 命令查看网卡名字，如图所示：

```
eame@debian: ~/ws/hal_driver/build/install/bin
eame@debian: ~/ws/hal_driver/build/install/bin 92x48
editorxu@11:35:09:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:a5:79:70:7c txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s31f6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.54.111 netmask 255.255.255.0 broadcast 192.168.54.255
        inet6 fe80::4e80:4e8ff:fe00:111 prefixlen 64 scopeid 0x20<link>
            ether 84:ba:59:72:ea:08 txqueuelen 1000 (Ethernet)
            RX packets 1042289 bytes 354005321 (354.0 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1050126 bytes 280690787 (280.6 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
            device interrupt 16 memory 0x82200000-82220000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 87371077 bytes 9065855211 (9.0 GB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 87371077 bytes 9065855211 (9.0 GB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
        ether 52:54:00:9e:77:df txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.29.41.197 netmask 255.255.252.0 broadcast 172.29.43.255
        inet6 fe80::4eb3:a694:2eba:ac34 prefixlen 64 scopeid 0x20<link>
            ether 5c:b4:7e:8f:79:97 txqueuelen 1000 (Ethernet)
            RX packets 26672293 bytes 13099885890 (13.0 GB)
            RX errors 0 dropped 101 overruns 0 frame 0
            TX packets 20349743 bytes 3661275993 (3.6 GB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

假设 SDK 二次开发 PC 与机器人链接的网口为 enp0s31f6，需要进行如下配置以便 SDK 与机器人的底层通信：

```
sudo ifconfig enp0s31f6 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev enp0s31f6
```

为避免每次网卡拔插都需要重新配置网卡多播路由，用户可以选择在其二开 PC 上进行如下操作（要求操作系统：ubuntu22.04，并且本系统网络配置工具是 NetworkManager）：

- 检查服务状态：

```
$ systemctl is-active NetworkManager  
active
```

当显示为: active, 则可以继续下面操作; 否则, 跳过下面操作;

- 编辑配置, sudo vim /etc/NetworkManager/dispatcher.d/90-add-mcast-route:

```
#!/bin/bash

IFACE="$1"
STATUS="$2"

# 目标网卡
TARGET_IF="enp0s31f6"

# 仅在指定网卡 carrier on 或 up 时执行
if [ "$IFACE" = "$TARGET_IF" ] && [[ "$STATUS" = "up" || "$STATUS" = "carrier" ]]; then
    # 删除可能存在的旧路由
    ip route del 224.0.0.0/4 dev "$TARGET_IF" 2>/dev/null

    # 添加你需要的 route (带 scope link)
    ip route add 224.0.0.0/4 dev "$TARGET_IF" scope link
fi
```

- 加上执行权限:

```
sudo chmod +x /etc/NetworkManager/dispatcher.d/90-add-mcast-route
```

- 启动生效:

```
sudo systemctl restart NetworkManager
```

检查网卡路由配置是否正常:

```
$ ip route | grep 224.0.0.0
224.0.0.0/4 dev enp0s31f6 scope link
```

6.3 安装与编译

以下步骤假设工作目录为/home/magicbot/workspace

6.3.1 安装 magicbot-gen1_sdk:

安装步骤：进入 `magicbot-gen1_sdk` 目录，并按步骤执行如下命令，将 `sdk` 安装至`/opt/magic_robots/magicbot_gen1_sdk`目录下，便于二次开发程序 `find_package` 检索编译链接：

```
cd /home/magicbot/workspace/magicbot-gen1_sdk/
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/opt/magic_robots/magicbot_gen1_sdk
sudo make install
```

6.3.2 例程编译

编译步骤：进入 `magicbot-gen1_sdk` 目录，并按步骤执行如下命令，在 `build` 目录下生成例程可执行程序：

```
cd /home/magicbot/workspace/magicbot-gen1_sdk/
mkdir build
cd build
cmake .. -DBUILD_EXAMPLES=ON
make -j8
```

6.4 C++ 例程示例

`magicbot-gen1_sdk/build` 目录中：

- 语音示例：
 - `audio_example`
- 传感器示例：
 - `sensor_example`
- 状态监控示例：
 - `monitor_example`
- 底层遥控示例：
 - `low_level_motion_example`
- 高层遥控示例：
 - `high_level_motion_example`
- 建图示例：
 - `slam_example`
- 导航示例：
 - `navigation_example`

- navigation_example

Notice: 其中，高层运控示例中几种示例有先后依赖顺序（锁定站立-> 平衡站立-> 执行特技/遥控器控制）

6.4.1 进入调试模式：

按照操作流程，确保机器人进入调试模式

6.4.2 运行例程

进入 magicbot-gen1_sdk/build 目录，执行如下命令：

```
cd /home/magicbot/workspace/magicbot-gen1_sdk/build
# 环境配置
sudo ifconfig enp0s31f6 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev enp0s31f6
export LD_LIBRARY_PATH=/opt/magic_robots/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

# 执行语音控制示例
./audio_example
```

6.5 Python 例程示例

magicbot-gen1_sdk/example/python 目录中：

- 语音示例：

- audio_example.py

- 传感器示例：

- sensor_example.py

- 状态监控示例：

- monitor_example.py

- 底层运控示例：

- low_level_motion_example.py

- 高层运控示例：

- high_level_motion_example.py

- 建图示例：

- slam_example.py

- 导航示例：

- navigation_example.py

Notice: 其中，高层运控示例中几种示例有先后依赖顺序（锁定站立-> 平衡站立-> 执行特技/遥控器控制）

6.5.1 进入调试模式：

按照操作流程，确保机器人进入调试模式

6.5.2 运行例程

进入 magicbot-gen1_sdk/example/python 目录，执行如下命令：

```
cd /home/magicbot/workspace/magicbot-gen1_sdk/example/python
# 环境配置
sudo ifconfig enp0s31f6 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev enp0s31f6
export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

# 执行语音控制示例
python3 audio_example.py
```

获取 python api 帮助信息：

```
# 环境配置
export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

$ python3
Python 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
# 导入 Magic Gen1 python SDK 接口
>>> import magicbot_gen1_python
# 查看所有接口信息
>>> help(magicbot_gen1_python)
# 查看 CameraInfo 结构信息
>>> help(magicbot_gen1_python.CameraInfo)
# 查看 HighLevelMotionController 对象信息
>>> help(magicbot_gen1_python.HighLevelMotionController)
# 查看 LowLevelMotionController 对象信息
>>> help(magicbot_gen1_python.LowLevelMotionController)
# 查看 SensorController 对象信息
>>> help(magicbot_gen1_python.SensorController)
# 查看 AudioController 对象信息
>>> help(magicbot_gen1_python.AudioController)
```

(下页继续)

(续上页)

```
# 查看StateMonitor对象信息
>>> help(magicbot_gen1_python.StateMonitor)
```

比如，如果想查看特技动作 TrickAction 的枚举值：

```
$ python3
Python 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import magicbot_gen1_python
>>> help(magicbot_gen1_python.TrickAction)

Help on class TrickAction in module magicbot_gen1_python:

class TrickAction(pybind11_builtins.pybind11_object)
| Method resolution order:
|     TrickAction
|     pybind11_builtins.pybind11_object
|     builtins.object
|
| Methods defined here:
|
|     __eq__(...)
|         __eq__(self: object, other: object, /) -> bool
|
|     __getstate__(...)
|         __getstate__(self: object, /) -> int
|
|     __hash__(...)
|         __hash__(self: object, /) -> int
|
|     __index__(...)
|         __index__(self: magicbot_gen1_python.TrickAction, /) -> int
|
|     __init__(...)
|         __init__(self: magicbot_gen1_python.TrickAction, value: typing.SupportsInt) -> None
|
|     __int__(...)
|         __int__(self: magicbot_gen1_python.TrickAction, /) -> int
|
|     __ne__(...)
|         __ne__(self: object, other: object, /) -> bool
|
|     __repr__(...)
|         __repr__(self: object, /) -> str
```

(下页继续)

(续上页)

```
|  
|     __setstate__(...)  
|         __setstate__(self: magicbot_gen1_python.TrickAction, state: typing.SupportsInt, /) ->  
|     None  
|  
|     __str__(...)  
|         __str__(self: object, /) -> str  
|  
|-----  
| Readonly properties defined here:  
|  
|     __members__  
|  
|     name  
|         name(self: object, /) -> str  
|  
|     value  
|  
|-----  
| Data and other attributes defined here:  
|  
|     ACTION_CELEBRATE = <TrickAction.ACTION_CELEBRATE: 201>  
|  
|     ACTION_CIRCLE_HEAD = <TrickAction.ACTION_CIRCLE_HEAD: 221>  
|  
|     ACTION_GREETING = <TrickAction.ACTION_GREETING: 301>  
|  
|     ACTION_NOD_HEAD = <TrickAction.ACTION_NOD_HEAD: 219>  
|  
|     ACTION_NONE = <TrickAction.ACTION_NONE: 0>  
|  
|     ACTION_POINT_GROUND = <TrickAction.ACTION_POINT_GROUND: 302>  
|  
|     ACTION_POINT_GROUND_WITH_DRAW = <TrickAction.ACTION_POINT_GROUND_WITH_DRAW: ...>  
|  
|     ACTION_RECOVERY_STAND = <TrickAction.ACTION_RECOVERY_STAND: 103>  
|  
|     ACTION_SHAKE_HAND_REACHOUT = <TrickAction.ACTION_SHAKE_HAND_REACHOUT: ...>  
|  
|     ACTION_SHAKE_HAND_WITHDRAW = <TrickAction.ACTION_SHAKE_HAND_WITHDRAW: ...>  
|  
|     ACTION_SHAKE_HEAD = <TrickAction.ACTION_SHAKE_HEAD: 220>  
|
```

(下页继续)

(续上页)

```
| ACTION_SPREAD_HAND = <TrickAction.ACTION_SPREAD_HAND: 304>
|
| ACTION_SPREAD_HAND_WITH_DRAW = <TrickAction.ACTION_SPREAD_HAND_WITH_DR...
|
| ACTION_TRUN_AWAY_LEFT_INTRODUCE = <TrickAction.ACTION_TRUN_AWAY_LEFT_I...
|
| ACTION_TRUN_BACK_LEFT_INTRODUCE = <TrickAction.ACTION_TRUN_BACK_LEFT_I...
|
| -----
| Static methods inherited from pybind11_builtins.pybind11_object:
|
|     __new__(*args, **kwargs) from pybind11_builtins.pybind11_type
|         Create and return a new object. See help(type) for accurate signature.
```

6.6 其他

6.6.1 SDK 配置文件

SDK 运行时默认会在/tmp 目录下生成其默认配置文件，如果配置文件已存在则使用已有配置：

```
$ ls /tmp/magicbot_gen1_mjrrt.yaml
/tmp/magicbot_gen1_mjrrt.yaml
```

6.6.2 SDK 日志

SDK 运行时内部的日志信息默认生成在/tmp/logs 目录下：

```
$ ls /tmp/logs/magicbot_gen1_sdk.log
/tmp/logs/magicbot_gen1_sdk.log
```

机器人主控制服务 C++

提供机器人系统主控制器，通过 MagicRobot 可以进行资源初始化、管理通信连接、访问各子控制器如运动控制器、语音、状态监控以及传感器控制器等。

7.1 接口定义

MagicRobot 是机器人 SDK 的统一入口类。

7.1.1 MagicRobot

项目	内容
函数名	MagicRobot
函数声明	MagicRobot();
功能概述	构造函数，创建 MagicRobot 实例。
备注	构造内部状态。

7.1.2 ~MagicRobot

项目	内容
函数名	<code>~MagicRobot</code>
函数声明	<code>~MagicRobot();</code>
功能概述	析构函数，释放 MagicRobot 实例资源。
备注	确保资源安全释放。

7.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>bool Initialize(const std::string& local_ip);</code>
功能概述	初始化机器人系统，包括控制器与网络模块。
参数说明	<code>local_ip</code> : 本地通信 IP 地址。
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败。
备注	首次使用前必须调用。

7.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>void Shutdown();</code>
功能概述	关闭机器人系统，释放资源。
备注	配合 Initialize 使用。

7.1.5 Release

项目	内容
函数名	<code>Release</code>
函数声明	<code>void Release();</code>
功能概述	释放资源。
备注	释放 SDK 占用话题资源。

7.1.6 Connect

项目	内容
函数名	Connect
函数声明	Status Connect();
功能概述	与机器人服务建立通信连接。
返回值	Status::OK 表示成功。
备注	阻塞接口，需初始化后调用。

7.1.7 Disconnect

项目	内容
函数名	Disconnect
函数声明	Status Disconnect();
功能概述	断开与机器人服务的连接。
返回值	gRPC 调用状态。
备注	阻塞接口，与 Connect 配对使用。

7.1.8 GetSDKVersion

项目	内容
函数名	GetSDKVersion
函数声明	std::string GetSDKVersion() const;
功能概述	获取当前 SDK 的版本号。
返回值	SDK 版本字符串（如 0.0.1）。
备注	非阻塞接口，便于调试或日志标记。

7.1.9 GetMotionControlLevel

项目	内容
函数名	GetMotionControlLevel
函数声明	ControllerLevel GetMotionControlLevel();
功能概述	获取当前运动控制级别（高层/低层）。
返回值	ControllerLevel 枚举值。
备注	非阻塞接口，用于判断当前控制权限。

7.1.10 SetMotionControlLevel

项目	内容
函数名	SetMotionControlLevel
函数声明	Status SetMotionControlLevel(ControllerLevel level);
功能概述	设置当前运动控制级别。
参数说明	level: 控制权限枚举值。
返回值	Status::OK 表示设置成功，其他代表设置失败。
备注	阻塞接口，控制模式切换时使用。

7.1.11 GetHighLevelMotionController

项目	内容
函数名	GetHighLevelMotionController
函数声明	HighLevelMotionController& GetHighLevelMotionController();
功能概述	获取高层运动控制器对象。
返回值	引用类型，用于调用高层控制接口。
备注	非阻塞接口，封装了步态、特技、遥控等控制。

7.1.12 GetLowLevelMotionController

项目	内容
函数名	GetLowLevelMotionController
函数声明	LowLevelMotionController& GetLowLevelMotionController();
功能概述	获取低层运动控制器对象。
返回值	引用类型，用于控制关节或电机。
备注	非阻塞接口，直接控制关节电机、获取 imu 等。

7.1.13 GetAudioController

项目	内容
函数名	GetAudioController
函数声明	AudioController& GetAudioController();
功能概述	获取音频控制器对象。
返回值	引用类型，用于语音控制。
备注	非阻塞接口，播放语音和控制音量。

7.1.14 GetSensorController

项目	内容
函数名	GetSensorController
函数声明	SensorController& GetSensorController();
功能概述	获取传感器控制器对象。
返回值	引用类型，用于访问传感器数据。
备注	非阻塞接口，封装了 Lidar、RGBD 等读取。

7.1.15 GetStateMonitor

项目	内容
函数名	GetStateMonitor
函数声明	StateMonitor& GetStateMonitor();
功能概述	获取状态监控器对象。
返回值	引用类型，用于获取机器人当前状态信息。
备注	非阻塞接口，封装了 BMS、故障等状态信息的读取。

7.1.16 GetSlamNavController

项目	内容
函数名	GetSlamNavController
函数声明	SlamNavController& GetSlamNavController();
功能概述	获取 SLAM 与导航控制器对象。
返回值	引用类型，用于建图与定位。
备注	非阻塞接口，用于机器人 SLAM 建图与自主导航功能。

CHAPTER 8

高层运动控制服务 C++

提供机器人系统高层运动控制服务，通过 HighLevelMotionController 可以通过 RPC 通信方式实现对机器人的步态、特技、遥控的控制。

8.1 接口定义

HighLevelMotionController 是面向语义控制的高层运动控制器，支持如行走、特技、头部运动等控制操作，封装底层细节以供上层系统调用。

8.1.1 HighLevelMotionController

项目	内容
函数名	HighLevelMotionController
函数声明	HighLevelMotionController();
功能概述	构造函数，初始化高层控制器状态
备注	构造内部控制资源

8.1.2 ~HighLevelMotionController

项目	内容
函数名	<code>~HighLevelMotionController</code>
函数声明	<code>virtual ~HighLevelMotionController();</code>
功能概述	析构函数，释放控制器资源
备注	配合构造函数使用

8.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>virtual bool Initialize() override;</code>
功能概述	初始化控制器，准备高层控制功能
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败
备注	首次使用前必须调用

8.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>virtual void Shutdown() override;</code>
功能概述	关闭控制器并释放资源
备注	配合 <code>Initialize</code> 使用，安全断开连接

8.1.5 SetGait

项目	内容
函数名	<code>SetGait</code>
函数声明	<code>Status SetGait(const GaitMode gait_mode, int timeout_ms);</code>
功能概述	设置机器人步态模式（如锁定站立、平衡站立、拟人行走等）
参数说明	<code>gait_mode</code> : 步态控制枚举
返回值	<code>Status::OK</code> 表示成功，其他为失败
备注	阻塞接口，可切换多种步态模式

8.1.6 GetGait

项目	内容
函数名	GetGait
函数声明	Status GetGait(GaitMode& gait_mode);
功能概述	获取机器人步态模式（如锁定站立、平衡站立、拟人行走等）
参数说明	gait_mode: 步态控制枚举
返回值	Status::OK 表示成功，其他为失败
备注	阻塞接口，获取当前步态模式

8.1.7 ExecuteTrick

项目	内容
函数名	ExecuteTrick
函数声明	Status ExecuteTrick(const TrickAction trick_action, int timeout_ms);
功能概述	执行特技动作（如庆祝、挥手等）
参数说明	trick_action: 特技动作标识
返回值	Status::OK 表示成功，其他为失败
备注	阻塞接口，需确保机器人当前可执行特技注意事项：特技动作必须要在 GaitMode::GAIT_BALANCE_STAND(46) 步态下才能进行

- 注意事项：特技动作必须要在 GaitMode::GAIT_BALANCE_STAND(46) 步态下才能进行特技展示

8.1.8 SendJoyStickCommand

项目	内容
函数名	SendJoyStickCommand
函数声明	Status SendJoyStickCommand(const JoyStickCommand& joy_command);
功能概述	发送实时摇杆控制指令
参数说明	joy_command: 包含摇杆坐标的控制数据
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，建议发送频率为 20Hz

8.1.9 HeadMove

项 目	内容
函数名	HeadMove
函数声明	Status HeadMove(float shake_angle, float nod_angle, int timeout_ms);
功能概述	控制机器人头部摇摆和点头运动
参数说明	shake_angle: 头部左右摇摆角度, 方向: 左负右正, 单位: 弧度, 范围: [-0.5236, 0.5236] nod_angle: 头部点头角度, 方向: 上正下负, 单位: 弧度, 范围: [-0.3491, 0.3491] timeout_ms: 超时时间, 默认 5000 毫秒
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 控制机器人头部左右摇摆和上下点头运动

8.2 类型定义

8.2.1 JoystickCommand — 高层运动控制摇杆指令结构体

字段名	类型	描述
left_x_axis	float	左侧摇杆的 X 轴方向值 (-1.0: 左, 1.0: 右)
left_y_axis	float	左侧摇杆的 Y 轴方向值 (-1.0: 下, 1.0: 上)
right_x_axis	float	右侧摇杆的 X 轴方向值 (旋转 -1.0: 左, 1.0: 右)
right_y_axis	float	右侧摇杆的 Y 轴方向值 (暂未定义用途)

8.3 枚举类型定义

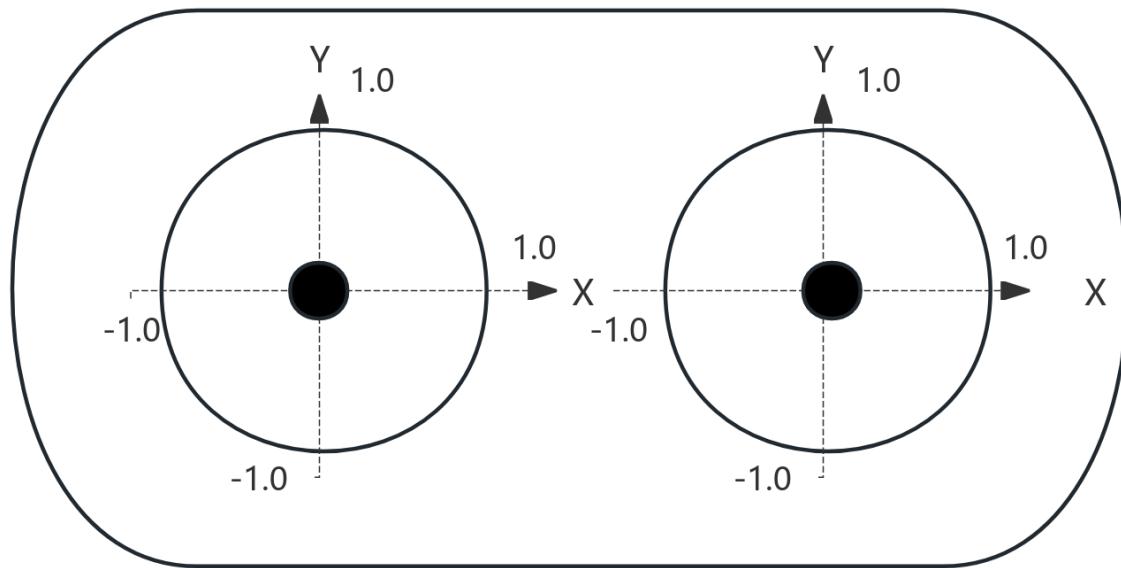
8.3.1 GaitMode — 机器人步态模式枚举

枚举值	数值	描述
GAIT_PASSIVE	0	Passive
GAIT_RECOVERY_STAND	1	站立锁定
GAIT_BALANCE_STAND	46	平衡站立（支持移动）
GAIT_ARM_SWING_WALK	47	摆臂行走
GAIT_LOWLEVEL_SDK	200	底层控制 SDK 模式

8.3.2 TrickAction — 特技动作指令枚举

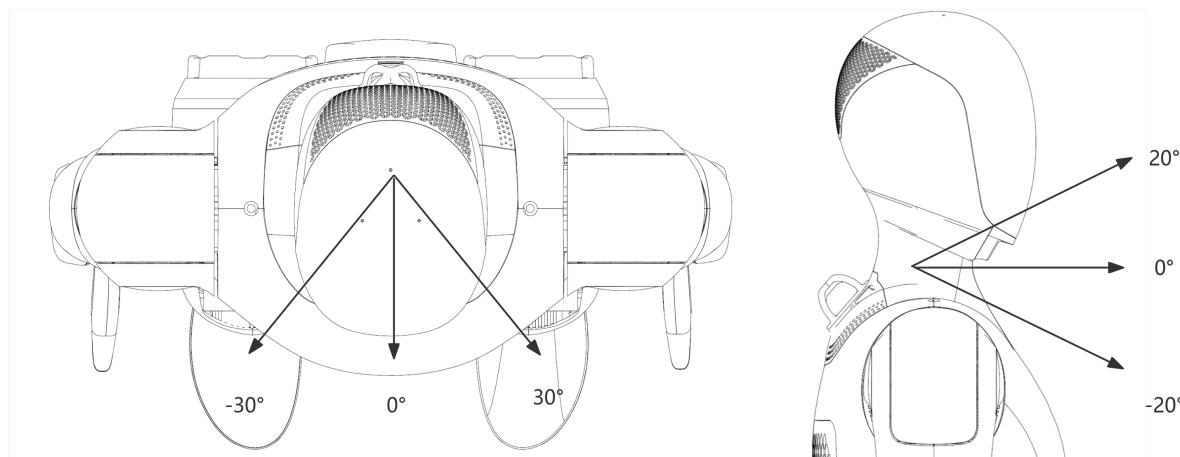
枚举值	数值	描述
ACTION_RECOVERY_STAND	103	平衡站立
ACTION_CELEBRATE	201	庆祝
ACTION_SHAKE_HAND_REACHOUT	217	握手 - 伸出
ACTION_SHAKE_HAND_WITHDRAW	218	握手 - 撤回
ACTION_NOD_HEAD	219	点头
ACTION_SHAKE_HEAD	220	摇头
ACTION_CIRCLE_HEAD	221	摇头晃脑
ACTION_GREETING	301	打招呼
ACTION_POINT_GROUND	302	指地 - 伸出
ACTION_POINT_GROUND_WITH_DRAW	303	指地 - 撤回
ACTION_SPREAD_HAND	304	张手
ACTION_SPREAD_HAND_WITH_DRAW	305	张手 - 撤回
ACTION_TRUN_AWAY_LEFT_INTRODUCE	306	扭身朝后
ACTION_TRUN_BACK_LEFT_INTRODUCE	307	扭身朝前

8.4 遥控器示意图



1. 左右摇杆 x 轴和 y 轴的取值范围为 [-1.0, 1.0];
2. 左右摇杆 x 轴和 y 轴的方向上/右为正，如示意图所示；

8.5 头部运动范围示意图



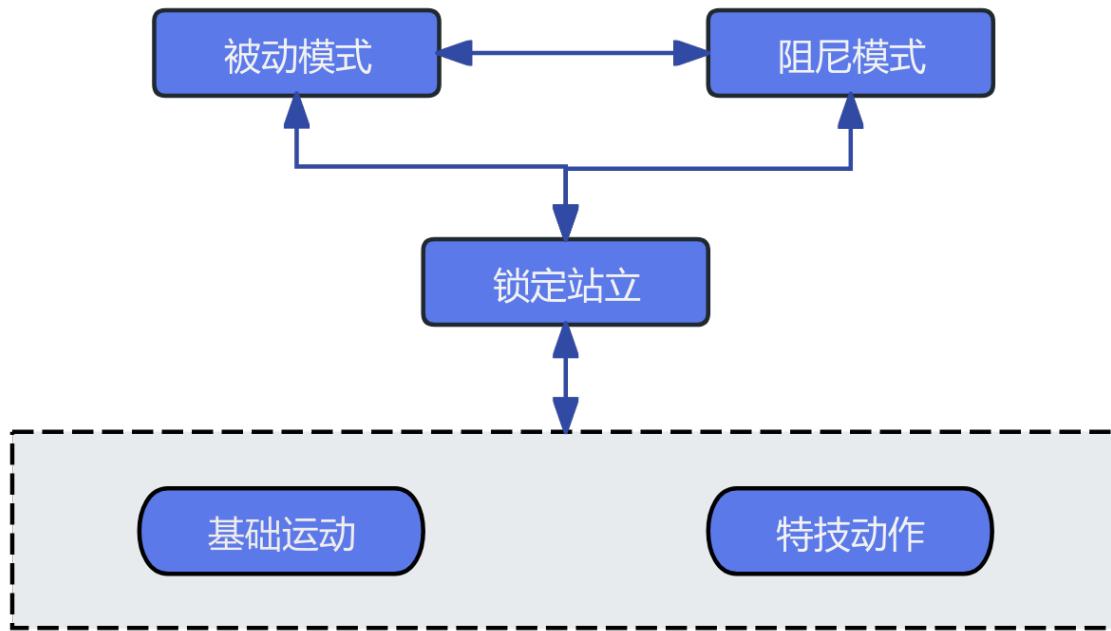
- 摆头运动范围 [-30°, 30°] ([-0.5236rad, 0.5236rad])
- 点头运动范围 [-20°, 20°] ([-0.3491rad, 0.3491rad])

8.6 高层运动控制机器人状态介绍

机器人的运动包含站立锁定、平衡站立、基础运动、特技动作状态，机器人在运行过程中，通过状态机在不同状态之间进行切换，以实现不同的控制任务。各个状态的解释说明如下：

- **站立锁定**：站立锁定是连接机器人挂起落地和平衡站立的状态，机器狗需要进入平衡站立状态后，才能调用相应的运动服务实现机器人的控制。
- **平衡站立**：在平衡站立状态下，可调用 SDK 的各部分接口实现机器人的特技动作和基础运动控制。
- **基础运动**：在运动执行过程中，可调用 SDK 接口，让机器人进入不同的步态。
- **特技动作**：当进入特殊动作执行状态后，其他运动控制服务会先被挂起，等待当前动作执行完毕并进入平衡站立状态后再生效。

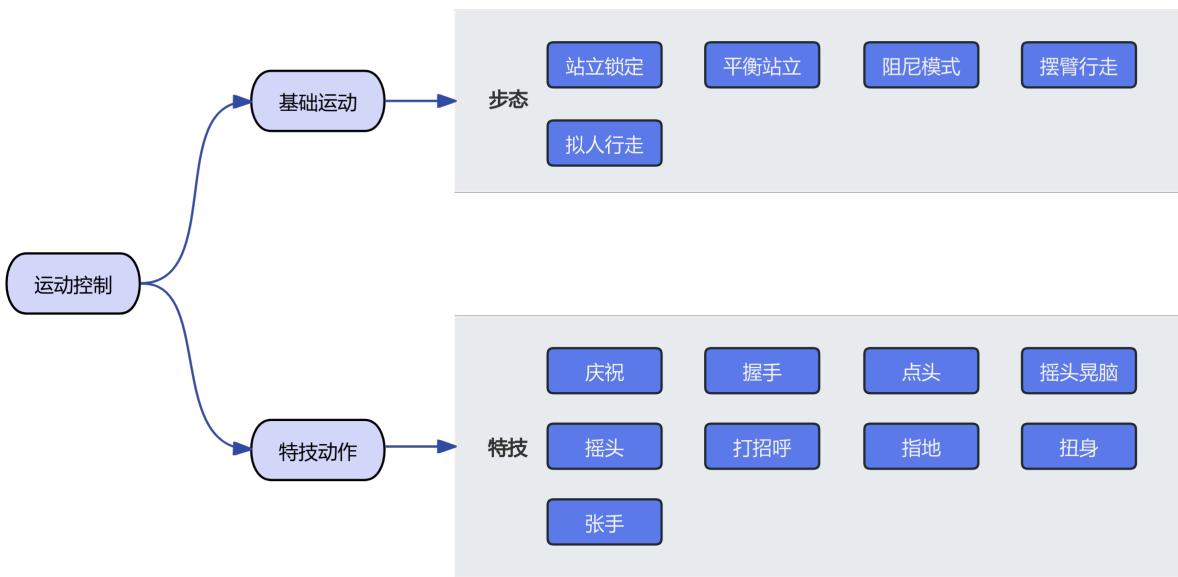
机器人状态切换机制：



8.7 高层运动控制接口

机器人的高层运动控制服务可分为基础运动控制和特技动作控制。

- 基础运动控制服务中，可调用相应的接口，根据不同的地形场景和任务需求切换机器人的行走步态。
- 特技动作控制服务中，可调用相应的接口，实现机器人内置的特殊特技，比如庆祝、握手、打招呼等。



底层运动控制服务 C++

提供机器人系统底层运动控制服务，通过 LowLevelMotionController 可以通过话题通信方式实现对机器人的关节进行指令控制和状态获取。

9.1 接口定义

LowLevelMotionController 是面向底层开发的运动控制器，支持对手臂、腿、头、腰、手等运动部件的直接控制与状态订阅，以及身体 IMU 数据的获取。

9.1.1 LowLevelMotionController

项目	内容
函数名	LowLevelMotionController
函数声明	LowLevelMotionController();
功能概述	构造函数，初始化低层控制器对象。
备注	构造内部资源。

9.1.2 ~LowLevelMotionController

项目	内容
函数名	<code>~LowLevelMotionController</code>
函数声明	<code>virtual ~LowLevelMotionController();</code>
功能概述	析构函数，释放资源。
备注	清理底层资源。

9.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>virtual bool Initialize() override;</code>
功能概述	初始化控制器，建立底层连接。
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败。
备注	首次调用必须初始化。

9.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>virtual void Shutdown() override;</code>
功能概述	关闭控制器，释放底层资源。
备注	配合 <code>Initialize</code> 使用。

9.1.5 SubscribeArmState

项目	内容
函数名	<code>SubscribeArmState</code>
函数声明	<code>void SubscribeArmState(ArmJointStateCallback callback);</code>
功能概述	订阅手臂关节状态数据
参数说明	callback：接收数据处理函数
备注	非阻塞接口。

9.1.6 UnsubscribeArmState

项目	内容
函数名	UnsubscribeArmState
函数声明	void UnsubscribeArmState();
功能概述	取消订阅手臂关节状态数据
备注	非阻塞接口。与 SubscribeArmState 配合使用。

9.1.7 PublishArmCommand

项目	内容
函数名	PublishArmCommand
函数声明	Status PublishArmCommand(const JointCommand& command);
功能概述	发布手臂控制指令
参数说明	command: 目标位置/速度等
返回值	true 表示成功, false 表示失败。
备注	非阻塞接口。

9.1.8 SubscribeLegState

项目	内容
函数名	SubscribeLegState
函数声明	void SubscribeLegState(LegJointStateCallback callback);
功能概述	订阅腿部关节状态数据
参数说明	callback: 接收数据处理函数
备注	非阻塞接口。

9.1.9 UnsubscribeLegState

项目	内容
函数名	UnsubscribeLegState
函数声明	void UnsubscribeLegState();
功能概述	取消订阅腿部关节状态数据
备注	非阻塞接口。与 SubscribeLegState 配合使用。

9.1.10 PublishLegCommand

项目	内容
函数名	PublishLegCommand
函数声明	Status PublishLegCommand(const JointCommand& command);
功能概述	发布腿部控制指令
参数说明	command: 目标位置/速度等
返回值	true 表示成功, false 表示失败。
备注	非阻塞接口, 发布或订阅时调用。

9.1.11 SubscribeHeadState

项目	内容
函数名	SubscribeHeadState
函数声明	void SubscribeHeadState(HeadJointStateCallback callback);
功能概述	订阅头部关节状态数据
参数说明	callback: 接收数据处理函数
备注	非阻塞接口。

9.1.12 UnsubscribeHeadState

项目	内容
函数名	UnsubscribeHeadState
函数声明	void UnsubscribeHeadState();
功能概述	取消订阅头部关节状态数据
备注	非阻塞接口。与 SubscribeHeadState 配合使用。

9.1.13 PublishHeadCommand

项目	内容
函数名	PublishHeadCommand
函数声明	Status PublishHeadCommand(const JointCommand& command);
功能概述	发布头部控制指令
参数说明	command: 目标位置/速度等
返回值	true 表示成功, false 表示失败。
备注	非阻塞接口。

9.1.14 SubscribeWaistState

项目	内容
函数名	SubscribeWaistState
函数声明	<code>void SubscribeWaistState(WaistJointStateCallback callback);</code>
功能概述	订阅腰部关节状态数据
参数说明	callback: 接收数据处理函数
备注	非阻塞接口。

9.1.15 UnsubscribeWaistState

项目	内容
函数名	UnsubscribeWaistState
函数声明	<code>void UnsubscribeWaistState();</code>
功能概述	取消订阅腰部关节状态数据
备注	非阻塞接口。与 SubscribeWaistState 配合使用。

9.1.16 PublishWaistCommand

项目	内容
函数名	PublishWaistCommand
函数声明	<code>Status PublishWaistCommand(const JointCommand& command);</code>
功能概述	发布腰部控制指令
参数说明	command: 目标位置/速度等
返回值	true 表示成功, false 表示失败。
备注	非阻塞接口。

9.1.17 SubscribeHandState

项目	内容
函数名	SubscribeHandState
函数声明	<code>void SubscribeHandState(HandStateCallback callback);</code>
功能概述	订阅手部状态数据
参数说明	callback: 接收数据处理函数
备注	非阻塞接口。

9.1.18 UnsubscribeHandState

项目	内容
函数名	UnsubscribeHandState
函数声明	void UnsubscribeHandState();
功能概述	取消订阅手部状态数据
备注	非阻塞接口。与 SubscribeHandState 配合使用。

9.1.19 PublishHandCommand

项目	内容
函数名	PublishHandCommand
函数声明	Status PublishHandCommand(const HandCommand& command);
功能概述	发布手部控制指令
参数说明	command: 手部关节目标位置等
返回值	true 表示成功, false 表示失败。
备注	非阻塞接口。

9.1.20 SubscribeBodyImu

项目	内容
函数名	SubscribeBodyImu
函数声明	void SubscribeBodyImu(const BodyImuCallback& callback);
功能概述	订阅机体 IMU 数据
参数说明	callback: IMU 数据处理函数
备注	非阻塞接口。

9.1.21 UnsubscribeBodyImu

项目	内容
函数名	UnsubscribeBodyImu
函数声明	void UnsubscribeBodyImu();
功能概述	取消订阅机体 IMU 数据
备注	非阻塞接口。与 SubscribeBodyImu 配合使用。

9.2 类型定义

9.2.1 SingleHandJointCommand —单个手部关节的控制命令

字段名	类型	描述
operation_mode	int16_t	控制字
pos	vector<float>;	期望位置数组 (7 个自由度, 单位: rad)

- 灵巧手 operation_mode 需要从模式: 200 切换到模式: 4 启动，并进行指令下发；
- 灵巧手关节指令范围值: [0-1000]

9.2.2 HandCommand —整个手部控制命令

字段名	类型	描述
timestamp	int64_t	时间戳 (单位: 纳秒)
cmd	vector<SingleHandJointCommand>;	控制命令数组, 依次为左手和右手

9.2.3 SingleHandJointState —单个手部关节的状态

字段名	类型	描述
status_word	int16_t	状态字
pos	vector<float>;	当前位置 (单位: rad)
toq	vector<float>;	当前力矩 (单位: Nm)
cur	vector<float>;	当前电流 (单位: A)
error_code	int16_t	错误码

9.2.4 HandState —整个手部状态信息

字段名	类型	描述
timestamp	int64_t	时间戳 (单位: 纳秒)
state	vector<SingleHandJointState>;	所有手部关节状态 (左手、右手顺序)

9.2.5 SingleJointCommand —单个关节的控制命令

字段名	类型	描述
operation_mode int16_t		控制模式标识: • 200 = 准备状态 • 3 = 并联三环控制 (MIT 模式) • 4 = 串联三环控制
pos float		目标位置 (单位: rad)
vel float		速度限制 (单位: rad/s)
torque float		力矩限制 (单位: Nm)
kp float		位置环比例增益
kd float		速度环微分增益

- 上臂 1-5 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 上臂 6-7 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 头部 1-2 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 腰部 1-2 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 腿部 1-6 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 3 (并联三环控制 (MIT 模式)) 进行指令下发;

9.2.6 JointCommand —关节控制命令

下肢包含 12 个关节项，上肢 14 个，头部 2 个，腰部 2 个:

字段名	类型	描述
timestamp int64_t		时间戳 (单位: 纳秒)
joints vector<SingleJointCommand>;		所有关节的控制命令

9.2.7 SingleJointState —单个关节的状态信息

字段名	类型	描述
status_word	int16_t	当前关节状态（自定义状态机编码）
posH	float	实际位置（高精度编码器读数）
posL	float	实际位置（低精度编码器读数，用于冗余校验）
vel	float	当前速度（单位：rad/s 或 m/s，根据关节类型自动切换）
toq	float	当前力矩（单位：Nm）
current	float	当前电流（单位：A）
err_code	int16_t	错误码（如编码器异常、电机过流等，详见错误码对照表）

9.2.8 JointState —关节状态数据

下肢包含 12 个关节项，上肢 14 个，头部 2 个，腰部 2 个：

字段名	类型	描述
timestamp	int64_t	时间戳（单位：纳秒）
joints	vector<SingleJointState>	所有关节的状态数据

9.2.9 关节电机顺序

头部关节

索引	关节名
0	head_yaw_joint
1	head_pitch_joint

上臂关节

索引	关节名
0	left_shoulder_pitch_joint
1	left_shoulder_roll_joint
2	left_shoulder_yaw_joint
3	left_elbow_roll_joint
4	left_wrist_yaw_joint
5	left_wrist_roll_joint
6	left_wrist_pitch_joint
7	right_shoulder_pitch_joint
8	right_shoulder_roll_joint
9	right_shoulder_yaw_joint
10	right_elbow_roll_joint
11	right_wrist_yaw_joint
12	right_wrist_roll_joint
13	right_wrist_pitch_joint

腰部关节

索引	关节名
0	waist_roll_joint
1	waist_yaw_joint

腿部关节

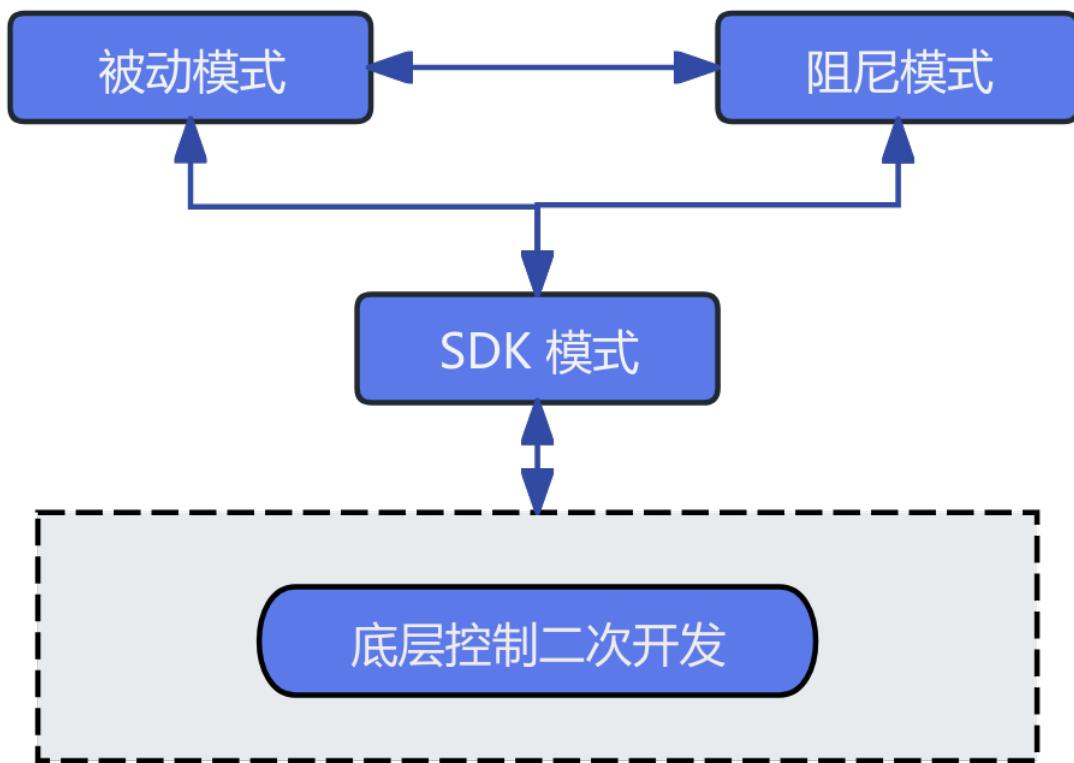
索引	关节名
0	left_hip_roll_joint
1	left_hip_yaw_joint
2	left_hip_pitch_joint
3	left_knee_pitch_joint
4	left_ankle_pitch_joint
5	left_ankle_roll_joint
6	right_hip_roll_joint
7	right_hip_yaw_joint
8	right_hip_pitch_joint
9	right_knee_pitch_joint
10	right_ankle_pitch_joint
11	right_ankle_roll_joint

9.3 URDF 参考

机器人 URDF

9.4 底层运动控制机器人状态介绍

机器人底层运动主要是开发关节的三环控制给开发人员进行机器人运动能力的二次开发，基本的控制状态切换机制：



9.5 注意事项

在使用 `Subscribe` 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

CHAPTER 10

传感器控制服务 C++

提供机器人系统传感器（雷达/rgbd 相机/三目相机）服务，通过 SensorController 可以通过 RPC 和话题方式实现对机器人的传感器进行指令控制和状态获取。

10.1 接口定义

SensorController 是封装机器人各类传感器的管理类，支持 Lidar、RGBD 相机与三目相机的初始化、控制与数据订阅。

△ **Notice:** 当前三目、RGBD 传感器接口暂未完全支持，请勿使用。

10.1.1 SensorController

项目	内容
函数名	SensorController
函数声明	SensorController();
功能概述	构造函数，初始化传感器控制器对象。
备注	构造内部状态。

10.1.2 ~SensorController

项目	内容
函数名	<code>~SensorController</code>
函数声明	<code>virtual ~SensorController();</code>
功能概述	析构函数，释放所有传感器资源。
备注	调用前应关闭传感器。

10.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>bool Initialize();</code>
功能概述	初始化控制器，包括资源申请和驱动加载。
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败。
备注	调用前需先构造对象。

10.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>void Shutdown();</code>
功能概述	关闭所有传感器连接并释放资源。
备注	配合 <code>Initialize</code> 使用。

10.1.5 OpenLidar

项目	内容
函数名	<code>OpenLidar</code>
函数声明	<code>Status OpenLidar();</code>
功能概述	打开雷达。
返回值	<code>Status::OK</code> 表示成功，其他为失败。
备注	阻塞接口。

10.1.6 CloseLidar

项目	内容
函数名	CloseLidar
函数声明	Status CloseLidar();
功能概述	关闭雷达。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口，配合打开函数使用。

10.1.7 OpenHeadRgbdCamera

项目	内容
函数名	OpenHeadRgbdCamera
函数声明	Status OpenHeadRgbdCamera();
功能概述	打开头部 RGBD 相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口。

10.1.8 CloseHeadRgbdCamera

项目	内容
函数名	CloseHeadRgbdCamera
函数声明	Status CloseHeadRgbdCamera();
功能概述	关闭头部 RGBD 相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口，配合打开函数使用。

10.1.9 OpenWaistRgbdCamera

项目	内容
函数名	OpenWaistRgbdCamera
函数声明	Status OpenWaistRgbdCamera();
功能概述	打开腰部 RGBD 相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口。

10.1.10 CloseWaistRgbdCamera

项目	内容
函数名	CloseWaistRgbdCamera
函数声明	Status CloseWaistRgbdCamera();
功能概述	关闭腰部 RGBD 相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口，配合打开函数使用。

10.1.11 OpenTrinocularCamera

项目	内容
函数名	OpenTrinocularCamera
函数声明	Status OpenTrinocularCamera();
功能概述	打开三目相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口。

10.1.12 CloseTrinocularCamera

项目	内容
函数名	CloseTrinocularCamera
函数声明	Status CloseTrinocularCamera();
功能概述	关闭三目相机。
返回值	Status::OK 表示成功，其他为失败。
备注	阻塞接口。

10.1.13 SubscribeLidarImu

项目	内容
函数名	SubscribeLidarImu
函数声明	void SubscribeLidarImu(const LidarImuCallback& callback);
功能概述	订阅雷达 IMU 数据
参数说明	callback: 接收到数据后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.14 UnsubscribeLidarImu

项目	内容
函数名	UnsubscribeLidarImu
函数声明	void UnsubscribeLidarImu();
功能概述	取消订阅雷达 IMU 数据
备注	非阻塞接口。与 SubscribeLidarImu 配合使用。

10.1.15 SubscribeLidarPointCloud

项目	内容
函数名	SubscribeLidarPointCloud
函数声明	void SubscribeLidarPointCloud(const LidarPointCloudCallback& callback);
功能概述	订阅雷达点云数据
参数说明	callback: 接收到点云后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.16 UnsubscribeLidarPointCloud

项目	内容
函数名	UnsubscribeLidarPointCloud
函数声明	void UnsubscribeLidarPointCloud();
功能概述	取消订阅雷达点云数据
备注	非阻塞接口。与 SubscribeLidarPointCloud 配合使用。

10.1.17 SubscribeHeadRgbdColorImage

项目	内容
函数名	SubscribeHeadRgbdColorImage
函数声明	void SubscribeHeadRgbdColorImage(const RgbdImageCallback& callback);
功能概述	订阅头部 RGBD 彩色图像数据
参数说明	callback: 接收到图像后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.18 UnsubscribeHeadRgbIColorImage

项目	内容
函数名	UnsubscribeHeadRgbIColorImage
函数声明	void UnsubscribeHeadRgbIColorImage();
功能概述	取消订阅头部 RGBD 彩色图像数据
备注	非阻塞接口。与 SubscribeHeadRgbIColorImage 配合使用。

10.1.19 SubscribeHeadRgbIColorCameraInfo

项目	内容
函数名	SubscribeHeadRgbIColorCameraInfo
函数声明	void SubscribeHeadRgbIColorCameraInfo(const RgbdCameraInfoCallback& callback);
功能概述	订阅头部 RGBD 彩色相机参数
参数说明	callback：接收到参数后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.20 UnsubscribeHeadRgbIColorCameraInfo

项目	内容
函数名	UnsubscribeHeadRgbIColorCameraInfo
函数声明	void UnsubscribeHeadRgbIColorCameraInfo();
功能概述	取消订阅头部 RGBD 彩色相机参数
备注	非阻塞接口。与 SubscribeHeadRgbIColorCameraInfo 配合使用。

10.1.21 SubscribeHeadRgbDepthImage

项目	内容
函数名	SubscribeHeadRgbDepthImage
函数声明	void SubscribeHeadRgbDepthImage(const RgbdImageCallback& callback);
功能概述	订阅头部 RGBD 深度图像数据
参数说明	callback：接收到图像后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.22 UnsubscribeHeadRgbDepthImage

项目	内容
函数名	UnsubscribeHeadRgbDepthImage
函数声明	void UnsubscribeHeadRgbDepthImage();
功能概述	取消订阅头部 RGBD 深度图像数据
备注	非阻塞接口。与 SubscribeHeadRgbDepthImage 配合使用。

10.1.23 SubscribeHeadRgbDepthCameraInfo

项目	内容
函数名	SubscribeHeadRgbDepthCameraInfo
函数声明	void SubscribeHeadRgbDepthCameraInfo(const RgbdCameraInfoCallback& callback);
功能概述	订阅头部 RGBD 深度相机参数
参数说明	callback: 接收到参数后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.24 UnsubscribeHeadRgbDepthCameraInfo

项目	内容
函数名	UnsubscribeHeadRgbDepthCameraInfo
函数声明	void UnsubscribeHeadRgbDepthCameraInfo();
功能概述	取消订阅头部 RGBD 深度相机参数
备注	非阻塞接口。与 SubscribeHeadRgbDepthCameraInfo 配合使用。

10.1.25 SubscribeWaistRgbColorImage

项目	内容
函数名	SubscribeWaistRgbColorImage
函数声明	void SubscribeWaistRgbColorImage(const RgbdImageCallback& callback);
功能概述	订阅腰部 RGBD 彩色图像数据
参数说明	callback: 接收到图像后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.26 UnsubscribeWaistRgbIColorImage

项目	内容
函数名	UnsubscribeWaistRgbIColorImage
函数声明	void UnsubscribeWaistRgbIColorImage();
功能概述	取消订阅腰部 RGBD 彩色图像数据
备注	非阻塞接口。与 SubscribeWaistRgbIColorImage 配合使用。

10.1.27 SubscribeWaistRgbIColorCameraInfo

项目	内容
函数名	SubscribeWaistRgbIColorCameraInfo
函数声明	void SubscribeWaistRgbIColorCameraInfo(const RgbdCameraInfoCallback& callback);
功能概述	订阅腰部 RGBD 彩色相机参数
参数说明	callback: 接收到参数后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.28 UnsubscribeWaistRgbIColorCameraInfo

项目	内容
函数名	UnsubscribeWaistRgbIColorCameraInfo
函数声明	void UnsubscribeWaistRgbIColorCameraInfo();
功能概述	取消订阅腰部 RGBD 彩色相机参数
备注	非阻塞接口。与 SubscribeWaistRgbIColorCameraInfo 配合使用。

10.1.29 SubscribeWaistRgbDepthImage

项目	内容
函数名	SubscribeWaistRgbDepthImage
函数声明	void SubscribeWaistRgbDepthImage(const RgbdImageCallback& callback);
功能概述	订阅腰部 RGBD 深度图像数据
参数说明	callback: 接收到图像后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.30 UnsubscribeWaistRgbDepthImage

项目	内容
函数名	UnsubscribeWaistRgbDepthImage
函数声明	void UnsubscribeWaistRgbDepthImage();
功能概述	取消订阅腰部 RGBD 深度图像数据
备注	非阻塞接口。与 SubscribeWaistRgbDepthImage 配合使用。

10.1.31 SubscribeWaistRgbDepthCameraInfo

项目	内容
函数名	SubscribeWaistRgbDepthCameraInfo
函数声明	void SubscribeWaistRgbDepthCameraInfo(const RgbdCameraInfoCallback& callback);
功能概述	订阅腰部 RGBD 深度相机参数
参数说明	callback: 接收到参数后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.32 UnsubscribeWaistRgbDepthCameraInfo

项目	内容
函数名	UnsubscribeWaistRgbDepthCameraInfo
函数声明	void UnsubscribeWaistRgbDepthCameraInfo();
功能概述	取消订阅腰部 RGBD 深度相机参数
备注	非阻塞接口。与 SubscribeWaistRgbDepthCameraInfo 配合使用。

10.1.33 SubscribeTrinocularImage

项目	内容
函数名	SubscribeTrinocularImage
函数声明	void SubscribeTrinocularImage(const TrinocularImageCallback& callback);
功能概述	订阅三目相机图像帧
参数说明	callback: 接收到图像帧后的处理函数
备注	非阻塞接口，回调函数会在数据更新时被调用。

10.1.34 UnsubscribeTrinocularImage

项目	内容
函数名	UnsubscribeTrinocularImage
函数声明	void UnsubscribeTrinocularImage();
功能概述	取消订阅三目相机图像帧
备注	非阻塞接口。与 SubscribeTrinocularImage 配合使用。

10.2 类型定义

10.2.1 Imu —IMU 数据结构体

字段名	类型	描述
timestamp	int64_t	时间戳 (单位: 纳秒)
orientation[4]	std::array<double, 4>;	姿态四元数 (w, x, y, z)
angular_velocity[3]	std::array<double, 3>;	角速度 (单位: rad/s)
linear_acceleration[3]	std::array<double, 3>;	线加速度 (单位: m/s ²)
temperature	float	温度 (单位: 摄氏度或其他, 应明确单位)

10.2.2 Header —通用消息头结构体

字段名	类型	描述
stamp	int64_t	时间戳 (单位: 纳秒)
frame_id	std::string	坐标系名称

10.2.3 PointField —点字段描述

字段名	类型	描述
name	std::string	字段名 (如 x、y、z、intensity)
offset	int32_t	起始字节偏移
datatype	int8_t	数据类型 (对应常量)
count	int32_t	每点此字段包含元素数量

10.2.4 PointCloud2 —点云数据结构体

字段名	类型	描述
header	Header	消息头
height	int32_t	行数
width	int32_t	列数
fields	vector<PointField>;	点字段数组
is_bigendian	bool	字节序
point_step	int32_t	每个点的字节数
row_step	int32_t	每行的字节数
data	vector<uint8_t>;	原始点云字节数据
is_dense	bool	是否稠密点云

10.2.5 Image —图像数据结构体

字段名	类型	描述
header	Header	消息头
height	int32_t	图像高度（像素）
width	int32_t	图像宽度（像素）
encoding	std::string	编码类型（如 rgb8, mono8）
is_bigendian	bool	是否为大端模式
step	int32_t	每行图像占用字节数
data	vector<uint8_t>;	原始图像字节数据

10.2.6 CameraInfo — 相机内参与畸变信息

字段名	类型	描述
header	Header	消息头
height	int32_t	图像高度
width	int32_t	图像宽度
distortion_model	std::string	畸变模型 (如 plumb_bob)
D	vector<double>;	畸变参数数组
K	std::array<double, 9>;	相机内参矩阵
R	std::array<double, 9>;	矫正矩阵
P	std::array<double, 12>;	投影矩阵
binning_x	int32_t	水平 binning 系数
binning_y	int32_t	垂直 binning 系数
roi_x_offset	int32_t	ROI 起始 x 坐标
roi_y_offset	int32_t	ROI 起始 y 坐标
roi_height	int32_t	ROI 高度
roi_width	int32_t	ROI 宽度
roi_do_rectify	bool	是否进行矫正

10.2.7 TrinocularCameraFrame — 三目相机帧数据结构

字段名	类型	描述
header	Header	通用消息头
vin_time	int64_t	图像采集时间 (纳秒)
decode_time	int64_t	图像解码时间 (纳秒)
imgfl_array	vector<uint8_t>;	左目图像数据
imgf_array	vector<uint8_t>;	中目图像数据
imgfr_array	vector<uint8_t>;	右目图像数据

10.3 注意事项

在使用 Subscribe 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

语音控制服务 C++

提供机器人系统语音服务控制器，通过 `AudioController` 可以通过 RPC 方式实现对机器人的音频进行指令控制和状态获取。

11.1 接口定义

`AudioController` 是一个封装音频控制功能的 C++ 类，主要用于音频播放控制、TTS 播放、音量设置与查询等场景。

11.1.1 `AudioController`

项目	内容
函数名	<code>AudioController</code>
函数声明	<code>AudioController();</code>
功能概述	初始化音频控制器对象，构造内部状态，分配资源等。
备注	构造内部状态。

11.1.2 ~AudioController

项目	内容
函数名	~AudioController
函数声明	~AudioController();
功能概述	释放音频控制器资源，确保停止播放并清理底层资源。
备注	确保资源安全释放。

11.1.3 Initialize

项目	内容
函数名	Initialize
函数声明	bool Initialize();
功能概述	初始化音频控制模块，准备播放资源与设备。
返回值	true 表示成功，false 表示失败。
备注	与 Shutdown() 配对使用。

11.1.4 Shutdown

项目	内容
函数名	Shutdown
函数声明	void Shutdown();
功能概述	关闭音频控制器并释放资源。
备注	确保在销毁前调用。

11.1.5 Play

项目	内容
函数名	Play
函数声明	Status Play(const TtsCommand& cmd, int timeout_ms);
功能概述	播放 TTS (文本转语音) 语音命令。
参数说明	cmd: TTS 命令，包含文本、语速、语调等。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，调用前需确保已初始化模块。

11.1.6 Stop

项目	内容
函数名	Stop
函数声明	Status Stop();
功能概述	停止当前音频播放。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，通常用于中断当前语音。

11.1.7 SetVolume

项目	内容
函数名	SetVolume
函数声明	Status SetVolume(int volume);
功能概述	设置音频输出的音量。
参数说明	volume: 音量值，通常范围为 0~100。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，设置后立即生效。

11.1.8 GetVolume

项目	内容
函数名	GetVolume
函数声明	Status GetVolume(int& volume);
功能概述	获取当前音频输出音量。
参数说明	volume: 通过引用返回当前音量值。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，返回值需检查后再使用 volume。

11.1.9 OpenAudioStream

项目	内容
函数名	OpenAudioStream
函数声明	Status OpenAudioStream();
功能概述	打开音频流，准备进行音频播放。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，返回值需检查是否执行成功

11.1.10 CloseAudioStream

项目	内容
函数名	CloseAudioStream
函数声明	Status CloseAudioStream();
功能概述	关闭音频流。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，返回值需检查是否执行成功

11.1.11 OpenWakeupStatusStream

项目	内容
函数名	OpenWakeupStatusStream
函数声明	Status OpenWakeupStatusStream();
功能概述	打开语音唤醒状态流。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，返回值需检查是否执行成功。

11.1.12 CloseWakeupStatusStream

项目	内容
函数名	CloseWakeupStatusStream
函数声明	Status CloseWakeupStatusStream();
功能概述	关闭语音唤醒状态流。
返回值	Status::OK 表示成功，其他为失败状态。
备注	阻塞接口，配合 OpenWakeupStatusStream 使用。

11.1.13 SubscribeWakeupStatus

项目	内容
函数名	SubscribeWakeupStatus
函数声明	void SubscribeWakeupStatus(const WakeupStatusCallback callback);
功能概述	订阅语音唤醒状态数据。
参数说明	callback: 接收到唤醒状态后的处理回调函数。
备注	非阻塞接口，回调函数会在唤醒状态更新时被调用。

11.1.14 UnsubscribeWakeupStatus

项目	内容
函数名	UnsubscribeWakeupStatus
函数声明	void UnsubscribeWakeupStatus();
功能概述	取消订阅语音唤醒状态数据。
备注	非阻塞接口。与 SubscribeWakeupStatus 配合使用。

11.1.15 SubscribeOriginAudioStream

项目	内容
函数名	SubscribeOriginAudioStream
函数声明	void SubscribeOriginAudioStream(const OriginAudioStreamCallback callback);
功能概述	订阅原始音频流数据。
参数说明	callback: 音频流数据回调函数。
备注	非阻塞接口。

11.1.16 UnsubscribeOriginAudioStream

项目	内容
函数名	UnsubscribeOriginAudioStream
函数声明	void UnsubscribeOriginAudioStream();
功能概述	取消订阅原始音频流数据。
备注	非阻塞接口。与 SubscribeOriginAudioStream 配合使用。

11.1.17 SubscribeBfAudioStream

项目	内容
函数名	SubscribeBfAudioStream
函数声明	void SubscribeBfAudioStream(const BfAudioStreamCallback callback);
功能概述	订阅 BF 音频流数据。
参数说明	callback: 音频流数据回调函数。
备注	非阻塞接口。

11.1.18 UnsubscribeBfAudioStream

项目	内容
函数名	UnsubscribeBfAudioStream
函数声明	void UnsubscribeBfAudioStream();
功能概述	取消订阅 BF 音频流数据。
备注	非阻塞接口。与 SubscribeBfAudioStream 配合使用。

11.2 类型定义

11.2.1 TtsPriority —TTS 播报优先级等级

用于控制不同 TTS 任务之间的中断行为。优先级越高的任务将中断当前低优先级任务的播放。

枚举值	描述
TtsPriority::HIGH	最高优先级，例如：低电告警、紧急提醒
TtsPriority::MIDDLE	中优先级，例如：系统提示、状态播报
TtsPriority::LOW	最低优先级，例如：日常语音对话、背景播报

11.2.2 TtsMode —同一优先级下的任务调度策略

用于细化控制在相同优先级条件下多个 TTS 任务的播放顺序和清除逻辑。

枚举值	描述
TtsMode::CLEARTOP	清空当前优先级所有任务（包括正在播放和等待队列），立即播放本次请求
TtsMode::ADD	将本次请求追加到当前优先级队列尾部，顺序播放（不打断当前播放）
TtsMode::CLEARBUFFER	清空队列中未播放的请求，保留当前播放，之后播放本次请求

11.3 结构体定义

11.3.1 TtsCommand —TTS 播放命令结构体

描述一次 TTS 播放请求的完整信息，支持设置唯一标识、文本内容、优先级控制以及同优先级下的调度模式。

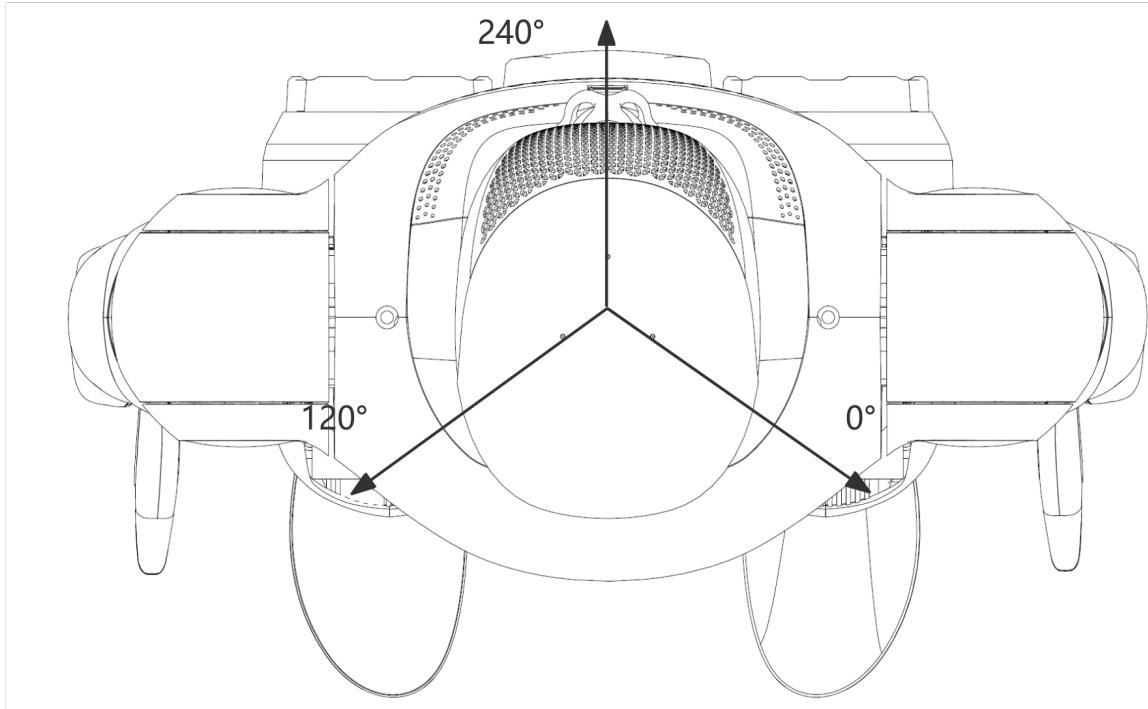
字段名	类型	描述
id	std::string	TTS 任务唯一 ID, 例如 "id_01", 用于追踪播放状态
content	std::string	要播放的文本内容, 例如 "你好, 欢迎使用智能语音系统。"
priority	TtsPriority	播报优先级, 控制是否中断正在播放的低优先级语音
mode	TtsMode	同优先级下的调度策略, 控制任务是否追加、覆盖等

11.3.2 AudioStream - 音频流结构体

描述原始音频流或 BF 音频流数据。

字段名	类型	描述
data_length	int32_t	数据长度
raw_data	std::vector<uint8_t>;	音频流数据

11.4 DOA



11.5 注意事项

在使用 Subscribe 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

CHAPTER 12

状态监控服务 C++

提供机器人状态监控接口类，通过 StateMonitor 可以提供状态查询等接口。

12.1 接口定义

StateMonitor 是机器人系统状态监控管理类，该类通常用于控制机器人的状态管理，支持状态查询。

12.1.1 StateMonitor

项目	内容
函数名	StateMonitor
函数声明	StateMonitor();
功能概述	构造函数，初始化状态监控器对象。
备注	构造内部状态。

12.1.2 ~StateMonitor

项目	内容
函数名	<code>~StateMonitor</code>
函数声明	<code>virtual ~StateMonitor();</code>
功能概述	析构函数，释放资源。
备注	调用前应关闭传感器。

12.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>bool Initialize();</code>
功能概述	初始化控制器，包括资源申请和驱动加载。
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败。
备注	调用前需先构造对象。

12.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>void Shutdown();</code>
功能概述	关闭并释放资源。
备注	配合 <code>Initialize</code> 使用。

12.1.5 GetcurrentState

项目	内容
函数名	<code>GetCurrentState</code>
函数声明	<code>Status GetcurrentState(RobotState& robot_state);</code>
功能概述	获取当前机器人聚合状态
返回值	<code>Status::OK</code> 表示成功，其他为失败状态。
备注	非阻塞接口，获取最近监控到的机器人状态数据，包括 BMS/故障状态监控等，后续会迭代拓展

12.1.6 RobotState — 机器人状态数据结构体

用于表示机器人总体状态信息：

字段名	类型	描述
faults	vector<Fault>;	故障信息列表
bms_data	BmsData	电池管理系统数据

12.1.7 BmsData — 电池管理系统数据结构体

表示电池的状态信息：

字段名	类型	描述
battery_percentage	float	当前电池剩余电量 (百分比, 0~100)
battery_health	float	电池健康状态 (越高越好)
battery_state	BatteryState	电池状态 (见下方枚举)
power_supply_status	PowerSupplyStatus	电池充放电状态 (见下方枚举)

12.1.8 枚举类型定义

BatteryState — 电池状态枚举类型

用于表示电池当前的状态，用于系统中电池状态的判断和处理：

枚举值	数值	描述
BatteryState::UNKNOWN	0	未知状态
BatteryState::GOOD	1	电池状态良好
BatteryState::OVERHEAT	2	电池过热
BatteryState::DEAD	3	电池损坏
BatteryState::OVERVOLTAGE	4	电池过电压
BatteryState::UNSPEC_FAILURE	5	未知故障
BatteryState::COLD	6	电池过冷
BatteryState::WATCHDOG_TIMER_EXPIRE	7	看门狗定时器超时
BatteryState::SAFETY_TIMER_EXPIRE	8	安全定时器超时

PowerSupplyStatus — 电池充放电状态

用于表示当前电池的充放电状态：

枚举值	数值	描述
PowerSupplyStatus::UNKNOWN	0	未知状态
PowerSupplyStatus::CHARGING	1	电池充电中
PowerSupplyStatus::DISCHARGING	2	电池放电中
PowerSupplyStatus::NOTCHARGING	3	电池未充放电
PowerSupplyStatus::FULL	4	电池充满

12.2 错误码映射表

错误码（十六进制）	错误描述
0x0000	No fault
0x1101	Service invocation failed
0x1301	Central control node lost
0x1302	App node lost
0x1303	Audio node lost
0x1304	Stereo camera node lost
0x1305	LIDAR node lost
0x1306	SLAM node lost
0x1307	Navigation node lost
0x1308	AI node lost
0x1309	Head node lost
0x130A	Point cloud node lost
0x2201	No LIDAR data received
0x2202	No stereo camera data received
0x2203	Stereo camera data error
0x2204	Stereo camera initialization failed
0x220B	No odometry data received
0x220C	No IMU data received
0x2215	Depth camera not detected
0x3101	Failed to connect robot to app
0x3102	Heartbeat lost - assertion failed
0x4201	Failed to open head serial port
0x4202	No head data received
0x5201	No navigation TF data
0x5202	No navigation map data
0x5203	No navigation localization data
0x5204	No navigation LIDAR data

下页继续

表 1 - 续上页

错误码（十六进制）	错误描述
0x5205	No navigation depth camera data
0x5206	No navigation multi-line LIDAR data
0x5207	No navigation odometry data
0x6201	SLAM localization error
0x6102	No SLAM LIDAR data
0x6103	No SLAM odometry data
0x6104	SLAM map data error
0x7201	LCM connection timeout
0x8201	Left leg hardware error
0x8202	Right leg hardware error
0x8203	Left arm hardware error
0x8204	Right arm hardware error
0x8205	Waist hardware error
0x8206	Head hardware error
0x8207	Hand hardware error
0x8208	Gripper hardware error
0x8209	IMU hardware error
0x820A	Power system hardware error
0x820B	Leg force sensor hardware error
0x820C	Arm force sensor hardware error
0x9201	ECAT (EtherCAT) hardware error
0xA201	Motion posture error
0xA202	Foot position deviation during movement
0xA203	Joint velocity error during motion

CHAPTER 13

SLAM 导航控制服务 C++

提供机器人系统 SLAM（同时定位与地图构建）和导航控制服务，通过 SlamNavController 可以通过 RPC 通信方式实现对机器人的建图、定位、导航等功能的控制。

13.1 接口定义

SlamNavController 是面向 SLAM 和导航控制的控制器，支持如建图、定位、导航、地图管理等控制操作，封装底层细节以供上层系统调用。

13.1.1 SlamNavController

项目	内容
函数名	SlamNavController
函数声明	SlamNavController();
功能概述	构造函数，初始化 SLAM 导航控制器状态
备注	构造内部控制资源

13.1.2 ~SlamNavController

项目	内容
函数名	<code>~SlamNavController</code>
函数声明	<code>~SlamNavController();</code>
功能概述	析构函数，释放控制器资源
备注	配合构造函数使用

13.1.3 Initialize

项目	内容
函数名	<code>Initialize</code>
函数声明	<code>bool Initialize();</code>
功能概述	初始化 SLAM 导航控制器，准备 SLAM 和导航功能
返回值	<code>true</code> 表示成功， <code>false</code> 表示失败
备注	首次使用前必须调用

13.1.4 Shutdown

项目	内容
函数名	<code>Shutdown</code>
函数声明	<code>void Shutdown();</code>
功能概述	关闭控制器并释放资源
备注	配合 <code>Initialize</code> 使用，安全断开连接

13.1.5 ActivateSlamMode

项目	内容
函数名	<code>ActivateSlamMode</code>
函数声明	<code>Status ActivateSlamMode(SlamMode mode, std::string map_path = "", int timeout_ms);</code>
功能概述	激活 SLAM 模式并开始建图或定位模式
参数说明	<code>mode</code> : SLAM 模式枚举 <code>map_path</code> : 地图绝对路径 (定位模式时使用), 可通过 <code>GetMapPath</code> 获取 <code>timeout_ms</code> : 超时时间 (毫秒)
返回值	<code>Status::OK</code> 表示成功, 其他为失败
备注	阻塞接口, 支持建图和定位模式切换

13.1.6 StartMapping

项目	内容
函数名	StartMapping
函数声明	Status StartMapping(int timeout_ms);
功能概述	开始建图
参数说明	timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 需先激活建图模式

13.1.7 CancelMapping

项目	内容
函数名	CancelMapping
函数声明	Status CancelMapping(int timeout_ms);
功能概述	取消建图
参数说明	timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 取消当前建图任务

13.1.8 SaveMap

项目	内容
函数名	SaveMap
函数声明	Status SaveMap(const std::string& map_name, int timeout_ms);
功能概述	结束建图并保存地图
参数说明	map_name: 地图名称 timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 需在建图模式下调用

13.1.9 LoadMap

项目	内容
函数名	LoadMap
函数声明	Status LoadMap(const std::string& map_name, int timeout_ms);
功能概述	加载地图并设置为当前地图
参数说明	map_name: 地图名称 timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 加载指定名称的地图

13.1.10 DeleteMap

项目	内容
函数名	DeleteMap
函数声明	Status DeleteMap(const std::string& map_name, int timeout_ms);
功能概述	删除地图
参数说明	map_name: 要删除的地图名称 timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 永久删除指定地图

13.1.11 GetMapPath

项目	内容
函数名	GetMapPath
函数声明	Status GetMapPath(const std::string& map_name, std::vector<std::vector<float>>& map_path, int timeout_ms);
功能概述	获取地图路径
参数说明	map_name: 地图名称 map_path: 地图路径 (输出参数) timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 获取指定地图的存储路径

13.1.12 GetAllMapInfo

项目	内容
函数名	GetAllMapInfo
函数声明	Status GetAllMapInfo(AllMapInfo& all_map_info, int timeout_ms);
功能概述	获取所有地图信息
参数说明	all_map_info: 所有地图信息 (输出参数) timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 获取系统中所有地图的详细信息

13.1.13 InitPose

项目	内容
函数名	InitPose
函数声明	Status InitPose(const Pose3DEuler& pose, int timeout_ms);
功能概述	初始化位姿
参数说明	pose: 要发布的位姿信息 timeout_ms: 超时时间 (毫秒), 默认 15000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 设置机器人初始位姿; 发布位姿时要对 Yaw 方向施加一个固定的-1.57rad 偏置, 因为激光雷达的机械安装与机器人本体存在一个-1.57rad 的偏置

13.1.14 GetCurrentLocalizationInfo

项目	内容
函数名	GetCurrentLocalizationInfo
函数声明	Status GetCurrentLocalizationInfo(LocalizationInfo& pose_info, int timeout_ms);
功能概述	获取当前位姿信息
参数说明	pose_info: 当前位置和姿态信息 (输出参数) timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 获取机器人当前位姿状态

13.1.15 ActivateNavMode

项目	内容
函数名	ActivateNavMode
函数声明	Status ActivateNavMode(NavMode mode, std::string map_path = "", int timeout_ms);
功能	激活导航模式
概述	
参数	mode: 目标导航模式 map_path: 地图绝对路径 (GRID_MAP 模式时使用), 可通过 GetMapPath 获得
说明	timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 激活指定导航模式

13.1.16 SetNavTarget

项目	内容
函数名	SetNavTarget
函数声明	Status SetNavTarget(const NavTarget& goal, int timeout_ms);
功能概述	设置全局导航目标点并开始导航任务
参数说明	goal: 目标点的全局坐标 timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 设置导航目标并开始导航

13.1.17 PauseNavTask

项目	内容
函数名	PauseNavTask
函数声明	Status PauseNavTask();
功能概述	暂停当前导航任务
返回值	Status::OK 表示成功, 其他为失败
备注	非阻塞接口, 暂停导航任务

13.1.18 ResumeNavTask

项目	内容
函数名	ResumeNavTask
函数声明	Status ResumeNavTask();
功能概述	恢复暂停的导航任务
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，恢复暂停的导航任务

13.1.19 CancelNavTask

项目	内容
函数名	CancelNavTask
函数声明	Status CancelNavTask();
功能概述	取消当前导航任务
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，取消当前导航任务

13.1.20 GetNavStatus

项目	内容
函数名	GetNavStatus
函数声明	Status GetNavStatus(NavStatus& nav_status);
功能概述	获取导航状态
参数说明	nav_status: 导航状态 (输出参数)
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，获取当前导航状态信息

13.1.21 OpenOdometryStream

项目	内容
函数名	OpenOdometryStream
函数声明	Status OpenOdometryStream();
功能概述	打开里程计数据流
返回值	Status::OK 表示成功，其他为失败
备注	阻塞接口，返回值需检查是否执行成功

13.1.22 CloseOdometryStream

项目	内容
函数名	CloseOdometryStream
函数声明	Status CloseOdometryStream();
功能概述	关闭里程计数据流
返回值	Status::OK 表示成功，其他为失败
备注	阻塞接口，配合 OpenOdometryStream 使用

13.1.23 SubscribeOdometry

项目	内容
函数名	SubscribeOdometry
函数声明	void SubscribeOdometry(const OdometryCallback callback);
功能概述	订阅里程计数据
参数说明	callback: 接收到里程计数据后的处理回调函数
备注	非阻塞接口，回调函数会在里程计数据更新时被调用

13.1.24 UnsubscribeOdometry

项目	内容
函数名	UnsubscribeOdometry
函数声明	void UnsubscribeOdometry();
功能概述	取消订阅里程计数据
备注	非阻塞接口。与 SubscribeOdometry 配合使用

13.1.25 GetPointCloudMap

项目	内容
函数名	GetPointCloudMap
函数声明	Status GetPointCloudMap(PointCloud2& point_cloud_map, int timeout_ms);
功能概述	获取点云地图数据
参数说明	point_cloud_map: 点云地图数据 (输出参数) timeout_ms: 超时时间 (毫秒)
返回值	Status::OK 表示成功，其他为失败
备注	阻塞接口，获取当前点云地图数据

13.2 类型定义

13.2.1 Pose3DEuler —3D 位姿结构体

字段名	类型	描述
position	std::array<double, 3>;	位置坐标(x, y, z)
orientation	std::array<double, 3>;	欧拉角(roll, pitch, yaw)

13.2.2 NavTarget —导航目标点结构体

字段名	类型	描述
id	int	目标点 ID
frame_id	std::string	目标点坐标系 ID
goal	Pose3DEuler	目标点位姿

13.2.3 LocalizationInfo —位姿信息结构体

字段名	类型	描述
is_localization	bool	是否已定位
pose	Pose3DEuler	当前位姿

13.2.4 NavStatus —导航状态结构体

字段名	类型	描述
id	int	目标点 ID, -1 表示无目标点
status	NavStatusType	导航状态类型
error_code	std::string	导航状态码
error_desc	std::string	导航状态消息

13.2.5 PolyRegion —多边形区域结构体

字段名	类型	描述
points	std::vector<Point2D>;	多边形顶点列表

13.2.6 Point2D —2D 点结构体

字段名	类型	描述
x	double	X 坐标
y	double	Y 坐标

13.3 枚举类型定义

13.3.1 SlamMode —SLAM 模式枚举

枚举值	数值	描述
IDLE	0	空闲模式
LOCALIZATION	1	定位模式
MAPPING	2	建图模式

13.3.2 NavMode —导航模式枚举

枚举值	数值	描述
IDLE	0	空闲模式
GRID_MAP	1	网格地图导航模式

13.3.3 NavStatusType —导航状态类型枚举

枚举值	数值	描述
NONE	0	无状态
RUNNING	1	运行中
END_SUCCESS	2	成功结束
END_FAILED	3	失败结束
PAUSE	4	暂停
CONTINUE	5	继续
CANCEL	6	取消

13.4 SLAM 导航控制介绍

机器人的 SLAM 导航控制服务可分为 SLAM 功能和导航功能。

- SLAM 功能中，可调用相应的接口，实现建图、定位、地图管理等功能。
- 导航功能中，可调用相应的接口，实现目标导航、导航控制等功能。

13.4.1 适用场景与范围

- 小于 100m * 100m 且特征丰富的静态室内平地场景，请勿超过建议范围
- 该部分功能适用于教育科研行业，不建议用于行业应用，行业应用请联系销售

13.4.2 SLAM 功能流程

功能	操作流程
建图	激活建图模式 → 开始建图 → 保存地图
地图管理	加载地图、获取地图信息、删除地图、获取地图绝对路径

13.4.3 导航功能流程

功能	操作流程
定位	加载地图 → 激活定位模式 → 初始化位姿 → 获取定位状态
导航	定位成功 → 激活导航模式 → 设置目标位姿 (开始导航任务) → 获取导航状态
导航控制	设置目标姿态 (开始导航任务) → 暂停导航 → 恢复导航 → 取消导航

13.4.4 定位模式初始化

SLAM 处于定位模式时，可以指定初始位姿以快速完成重定位与初始化，在使用大规模地图时必须指定初始位姿：

小地图模式

满足如下条件：

- 条件一：地图不能太大例如在一个房间内，地图面积最好控制在 10m * 10m 以内：
- 条件二，修改配置适配该模式：

```
# 登录机器人本体
ssh eame@192.168.54.119
# 修改如下配置
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
```

(下页继续)

(续上页)

```
reloc_options: true

# 首次更改yaml文件后需要重启SLAM模块生效，重启指令为：
sudo systemctl restart slam.service
```

大地图模式

满足如下条件：

- 条件一：机器人必须处于建图原点（建图开始时机器人的出发点），机器人朝向也要保持一致
- 条件二：修改配置适配该模式：

SDK 方式 SLAM 导航不同于 App 端方便可视化进行定位，需要修改 SLAM 配置，固定地图方向，方便 SLAM 定位：

```
# 登录机器人本体
ssh eame@192.168.54.119
# 修改如下配置
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# 首次更改yaml文件后需要重启SLAM模块生效，重启指令为：
sudo systemctl restart slam.service
```

- 条件三：初始化位姿时，需要对 yaw 角施加一个-90°(-1.57rad) 偏置：

因为激光雷达的机械安装与机器人本体相差-90°

13.4.5 地图噪点擦除以及添加障碍

- 软件名称：GIMP
- 下载地址：<https://www.gimp.org/downloads/>
- 操作系统：Ubuntu
- 编辑对象：/home/eame/cust_para/maps/\${map_name}/map/map.pgm 登录机器人本体（ssh eame@192.168.54.119），将.pgm 地图文件通过 GIMP 进行编辑，编辑完成后进行替换；
- 编辑视频：

机器人主控制服务 Python

提供机器人系统主控制器，通过 `MagicRobot` 类可以进行资源初始化、管理通信连接、访问各子控制器如运动控制器、语音、状态监控以及传感器控制器等。

14.1 接口定义

`MagicRobot` 是机器人 SDK 的统一入口类。

14.1.1 MagicRobot

项目	内容
类名	<code>MagicRobot</code>
构造函数	<code>robot = MagicRobot()</code>
功能概述	构造函数，创建 <code>MagicRobot</code> 实例。
备注	构造内部状态。

14.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize(str local_ip)
功能概述	初始化机器人系统，包括控制器与网络模块。
参数说明	local_ip: 本地通信 IP 地址。
返回值	True 表示成功, False 表示失败。
备注	首次使用前必须调用。

14.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭机器人系统，释放资源。
备注	配合 initialize 使用。

14.1.4 connect

项目	内容
函数名	connect
函数声明	Status connect()
功能概述	与机器人服务建立通信连接。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口，需初始化后调用。

14.1.5 disconnect

项目	内容
函数名	disconnect
函数声明	Status disconnect()
功能概述	断开与机器人服务的连接。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口，与 connect 配对使用。

14.1.6 get_sdk_version

项目	内容
函数名	get_sdk_version
函数声明	str get_sdk_version()
功能概述	获取当前 SDK 的版本号。
返回值	SDK 版本字符串 (如 0.0.1)。
备注	非阻塞接口，便于调试或日志标记。

14.1.7 get_motion_control_level

项目	内容
函数名	get_motion_control_level
函数声明	ControllerLevel get_motion_control_level()
功能概述	获取当前运动控制级别 (高层/低层)。
返回值	ControllerLevel 枚举值。
备注	非阻塞接口，用于判断当前控制权限。

14.1.8 set_motion_control_level

项目	内容
函数名	set_motion_control_level
函数声明	Status set_motion_control_level(ControllerLevel level)
功能概述	设置当前运动控制级别。
参数说明	level：控制权限枚举值。
返回值	Status 对象，Status.code == ErrorCode.OK 表示设置成功。
备注	阻塞接口，控制模式切换时使用。

14.1.9 get_high_level_motion_controller

项目	内容
函数名	get_high_level_motion_controller
函数声明	HighLevelMotionController get_high_level_motion_controller()
功能概述	获取高层运动控制器对象。
返回值	HighLevelMotionController 对象，用于调用高层控制接口。
备注	非阻塞接口，封装了步态、特技、遥控等控制。

14.1.10 get_low_level_motion_controller

项目	内容
函数名	get_low_level_motion_controller
函数声明	LowLevelMotionController get_low_level_motion_controller()
功能概述	获取低层运动控制器对象。
返回值	LowLevelMotionController 对象，用于控制关节或电机。
备注	非阻塞接口，直接控制关节电机、获取 imu 等。

14.1.11 get_audio_controller

项目	内容
函数名	get_audio_controller
函数声明	AudioController get_audio_controller()
功能概述	获取音频控制器对象。
返回值	AudioController 对象，用于语音控制。
备注	非阻塞接口，播放语音和控制音量。

14.1.12 get_sensor_controller

项目	内容
函数名	get_sensor_controller
函数声明	SensorController get_sensor_controller()
功能概述	获取传感器控制器对象。
返回值	SensorController 对象，用于访问传感器数据。
备注	非阻塞接口，封装了 IMU、RGBD 等读取。

14.1.13 get_state_monitor

项目	内容
函数名	get_state_monitor
函数声明	StateMonitor get_state_monitor()
功能概述	获取状态监控器对象。
返回值	StateMonitor 对象，用于获取机器人当前状态信息。
备注	非阻塞接口，封装了 BMS、故障等状态信息的读取。

14.1.14 get_slam_nav_controller

项目	内容
函数名	get_slam_nav_controller
函数声明	SlamNavController get_slam_nav_controller();
功能概述	获取 SLAM 与导航控制器对象。
返回值	引用类型，用于建图与定位。
备注	非阻塞接口，用于机器人 SLAM 建图与自主导航功能。

CHAPTER 15

高层运动控制服务 Python

提供机器人系统高层运动控制服务，通过 HighLevelMotionController 可以通过 RPC 通信方式实现对机器人的步态、特技、遥控的控制。

15.1 接口定义

HighLevelMotionController 是面向语义控制的高层运动控制器，支持如行走、特技、头部运动等控制操作，封装底层细节以供上层系统调用。

15.1.1 HighLevelMotionController

项目	内容
类名	HighLevelMotionController
构造函数	controller = HighLevelMotionController()
功能概述	构造函数，初始化高层控制器状态
备注	构造内部控制资源

15.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize()
功能概述	初始化控制器，准备高层控制功能
返回值	True 表示成功，False 表示失败
备注	首次使用前必须调用

15.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭控制器并释放资源
备注	配合 initialize 使用，安全断开连接

15.1.4 set_gait

项目	内容
函数名	set_gait
函数声明	Status set_gait(GaitMode gait_mode, int timeout_ms)
功能概述	设置机器人步态模式（如锁定站立、平衡站立、拟人行走等）
参数说明	gait_mode: 步态控制枚举
返回值	Status 对象，Status.code == ErrorCode.OK 表示成功
备注	阻塞接口，可切换多种步态模式

15.1.5 get_gait

项目	内容
函数名	get_gait
函数声明	tuple[Status, GaitMode] get_gait()
功能概述	获取机器人步态模式（如锁定站立、平衡站立、拟人行走等）
返回值	Status 对象，包含当前步态模式信息
备注	阻塞接口，获取当前步态模式

15.1.6 execute_trick

项目	内容
函数名	execute_trick
声明	函数 Status execute_trick(TrickAction trick_action, int timeout_ms)
功能	执行特技动作（如庆祝、挥手等）
概述	
参数	trick_action: 特技动作标识
说明	
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功
备注	阻塞接口, 需确保机器人当前可执行特技注意事项: 特技动作必须要在 GaitMode.GAIT_BALANCE_STAND(46) 步态下才能进行

15.1.7 send_joystick_command

项目	内容
函数名	send_joystick_command
函数声明	Status send_joystick_command(JoystickCommand joy_command)
功能概述	发送实时摇杆控制指令
参数说明	joy_command: 包含摇杆坐标的控制数据
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功
备注	非阻塞接口, 建议发送频率为 20Hz

15.1.8 head_move

项 目	内容
函数名	head_move
函数声明	Status head_move(float shake_angle, float nod_angle, int timeout_ms);
功能概述	控制机器人头部摇摆和点头运动
参数说明	shake_angle: 头部左右摇摆角度, 方向: 左负右正, 单位: 弧度, 范围: [-0.5236, 0.5236] nod_angle: 头部点头角度, 方向: 上正下负, 单位: 弧度, 范围: [-0.3491, 0.3491] timeout_ms: 超时时间, 默认 5000 毫秒
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 控制机器人头部左右摇摆和上下点头运动

15.2 类型定义

15.2.1 JoystickCommand — 高层运动控制摇杆指令结构体

字段名	类型	描述
x	float	左侧摇杆的 X 轴方向值 (-1.0: 左, 1.0: 右)
y	float	左侧摇杆的 Y 轴方向值 (-1.0: 下, 1.0: 上)
yaw	float	右侧摇杆的 X 轴方向值 (旋转 -1.0: 左, 1.0: 右)
z	float	右侧摇杆的 Y 轴方向值 (暂未定义用途)

15.3 枚举类型定义

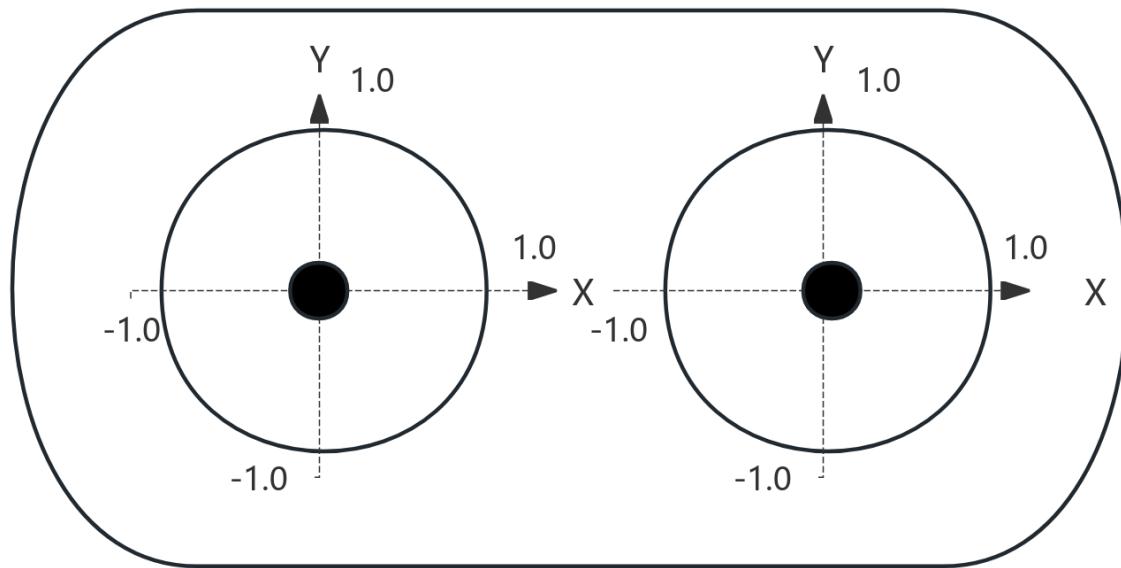
15.3.1 GaitMode — 机器人步态模式枚举

枚举值	数值	描述
GAIT_PASSIVE	0	Passive
GAIT_RECOVERY_STAND	1	站立锁定
GAIT_BALANCE_STAND	46	平衡站立（支持移动）
GAIT_ARM_SWING_WALK	47	摆臂行走
GAIT_LOWLEV_L_SDK	200	底层控制 SDK 模式

15.3.2 TrickAction — 特技动作指令枚举

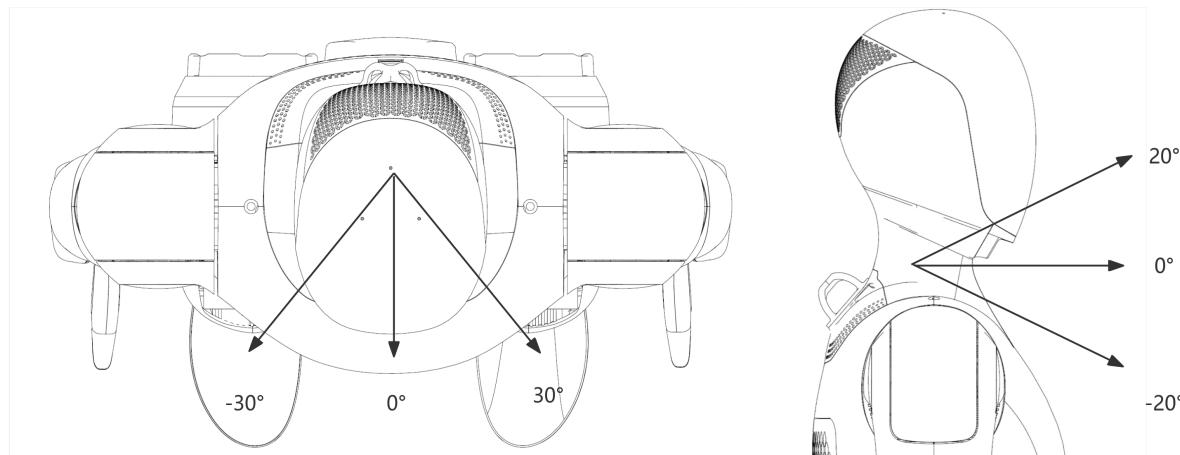
枚举值	数值	描述
ACTION_RECOVERY_STAND	103	平衡站立
ACTION_CELEBRATE	201	庆祝
ACTION_SHAKE_HAND_REACHOUT	217	握手 - 伸出
ACTION_SHAKE_HAND_WITHDRAW	218	握手 - 撤回
ACTION_NOD_HEAD	219	点头
ACTION_SHAKE_HEAD	220	摇头
ACTION_CIRCLE_HEAD	221	摇头晃脑
ACTION_GREETING	301	打招呼
ACTION_POINT_GROUND	302	指地 - 伸出
ACTION_POINT_GROUND_WITH_DRAW	303	指地 - 撤回
ACTION_SPREAD_HAND	304	张手
ACTION_SPREAD_HAND_WITH_DRAW	305	张手 - 撤回
ACTION_TRUN_AWAY_LEFT_INTRODUCE	306	扭身朝后
ACTION_TRUN_BACK_LEFT_INTRODUCE	307	扭身朝前

15.4 遥控器示意图



1. 左右摇杆 x 轴和 y 轴的取值范围为 [-1.0, 1.0];
2. 左右摇杆 x 轴和 y 轴的方向上/右为正，如示意图所示；

15.5 头部运动范围示意图



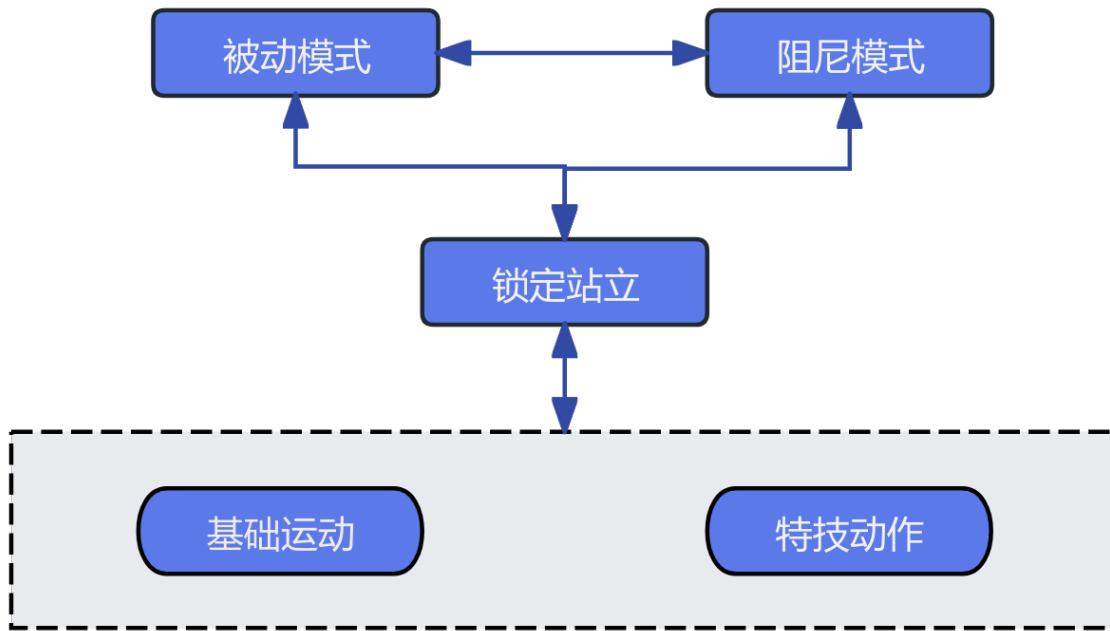
- 摆头运动范围 [-30°, 30°] ([-0.5236rad , 0.5236rad])
- 点头运动范围 [-20°, 20°] ([-0.3491rad , 0.3491rad])

15.6 高层运动控制机器人状态介绍

机器人的运动包含站立锁定、平衡站立、基础运动、特技动作状态，机器人在运行过程中，通过状态机在不同状态之间进行切换，以实现不同的控制任务。各个状态的解释说明如下：

- **站立锁定**：站立锁定是连接机器人挂起落地和平衡站立的状态，机器狗需要进入平衡站立状态后，才能调用相应的运动服务实现机器人的控制。
- **平衡站立**：在平衡站立状态下，可调用 SDK 的各部分接口实现机器人的特技动作和基础运动控制。
- **基础运动**：在运动执行过程中，可调用 SDK 接口，让机器人进入不同的步态。
- **特技动作**：当进入特殊动作执行状态后，其他运动控制服务会先被挂起，等待当前动作执行完毕并进入平衡站立状态后再生效。

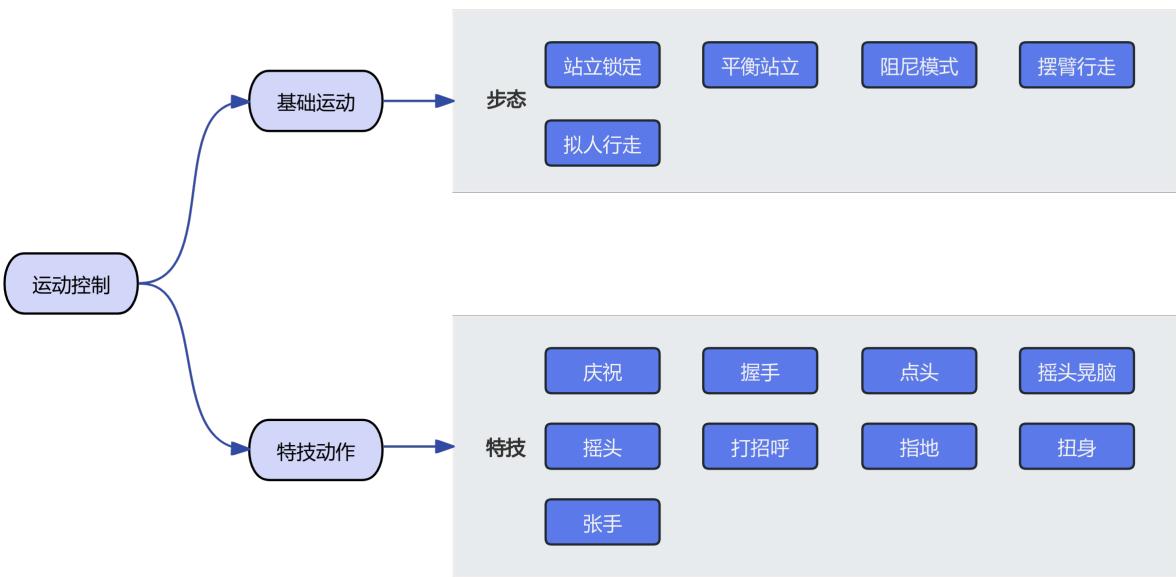
机器人状态切换机制：



15.7 高层运动控制接口

机器人的高层运动控制服务可分为基础运动控制和特技动作控制。

- 基础运动控制服务中，可调用相应的接口，根据不同的地形场景和任务需求切换机器人的行走步态。
- 特技动作控制服务中，可调用相应的接口，实现机器人内置的特殊特技，比如庆祝、握手、打招呼等。



底层运动控制服务 Python

提供机器人系统底层运动控制服务，通过 LowLevelMotionController 可以通过话题通信方式实现对机器人的关节进行指令控制和状态获取。

16.1 接口定义

LowLevelMotionController 是面向底层开发的运动控制器，支持对手臂、腿、头、腰、手等运动部件的直接控制与状态订阅，以及身体 IMU 数据的获取。

16.1.1 LowLevelMotionController

项目	内容
类名	LowLevelMotionController
构造函数	controller = LowLevelMotionController()
功能概述	构造函数，初始化低层控制器对象。
备注	构造内部资源。

16.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize()
功能概述	初始化控制器，建立底层连接。
返回值	True 表示成功，False 表示失败。
备注	首次调用必须初始化。

16.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭控制器，释放底层资源。
备注	配合 initialize 使用。

16.1.4 subscribe_arm_state

项目	内容
函数名	subscribe_arm_state
函数声明	void subscribe_arm_state(callback)
功能概述	订阅手臂关节状态数据
参数说明	callback：接收数据处理函数，签名为 callback(data: JointState) -> None
备注	非阻塞接口。

16.1.5 unsubscribe_arm_state

项目	内容
函数名	unsubscribe_arm_state
函数声明	void unsubscribe_arm_state()
功能概述	取消订阅手臂关节状态数据
备注	非阻塞接口。与 subscribe_arm_state 配合使用。

16.1.6 publish_arm_command

项目	内容
函数名	publish_arm_command
函数声明	Status publish_arm_command(JointCommand command)
功能概述	发布手臂控制指令
参数说明	command: 目标位置/速度等
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	非阻塞接口。

16.1.7 subscribe_leg_state

项目	内容
函数名	subscribe_leg_state
函数声明	void subscribe_leg_state(callback)
功能概述	订阅腿部关节状态数据
参数说明	callback: 接收数据处理函数, 签名为 callback(data: JointState) -> None
备注	非阻塞接口。

16.1.8 unsubscribe_leg_state

项目	内容
函数名	unsubscribe_leg_state
函数声明	void unsubscribe_leg_state()
功能概述	取消订阅腿部关节状态数据
备注	非阻塞接口。与 subscribe_leg_state 配合使用。

16.1.9 publish_leg_command

项目	内容
函数名	publish_leg_command
函数声明	Status publish_leg_command(JointCommand command)
功能概述	发布腿部控制指令
参数说明	command: 目标位置/速度等
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	非阻塞接口, 发布或订阅时调用。

16.1.10 subscribe_head_state

项目	内容
函数名	subscribe_head_state
函数声明	void subscribe_head_state(callback)
功能概述	订阅头部关节状态数据
参数说明	callback: 接收数据处理函数, 签名为 callback(data: JointState) -> None
备注	非阻塞接口。

16.1.11 unsubscribe_head_state

项目	内容
函数名	unsubscribe_head_state
函数声明	void unsubscribe_head_state()
功能概述	取消订阅头部关节状态数据
备注	非阻塞接口。与 subscribe_head_state 配合使用。

16.1.12 publish_head_command

项目	内容
函数名	publish_head_command
函数声明	Status publish_head_command(JointCommand command)
功能概述	发布头部控制指令
参数说明	command: 目标位置/速度等
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	非阻塞接口。

16.1.13 subscribe_waist_state

项目	内容
函数名	subscribe_waist_state
函数声明	void subscribe_waist_state(callback)
功能概述	订阅腰部关节状态数据
参数说明	callback: 接收数据处理函数, 签名为 callback(data: JointState) -> None
备注	非阻塞接口。

16.1.14 unsubscribe_waist_state

项目	内容
函数名	unsubscribe_waist_state
函数声明	void unsubscribe_waist_state()
功能概述	取消订阅腰部关节状态数据
备注	非阻塞接口。与 subscribe_waist_state 配合使用。

16.1.15 publish_waist_command

项目	内容
函数名	publish_waist_command
函数声明	Status publish_waist_command(JointCommand command)
功能概述	发布腰部控制指令
参数说明	command: 目标位置/速度等
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	非阻塞接口。

16.1.16 subscribe_hand_state

项目	内容
函数名	subscribe_hand_state
函数声明	void subscribe_hand_state(callback)
功能概述	订阅手部状态数据
参数说明	callback: 接收数据处理函数, 签名为 callback(data: HandState) -> None
备注	非阻塞接口。

16.1.17 unsubscribe_hand_state

项目	内容
函数名	unsubscribe_hand_state
函数声明	void unsubscribe_hand_state()
功能概述	取消订阅手部状态数据
备注	非阻塞接口。与 subscribe_hand_state 配合使用。

16.1.18 publish_hand_command

项目	内容
函数名	publish_hand_command
函数声明	Status publish_hand_command(HandCommand command)
功能概述	发布手部控制指令
参数说明	command: 手部关节目标位置等
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	非阻塞接口。

16.1.19 subscribe_body_imu

项目	内容
函数名	subscribe_body_imu
函数声明	void subscribe_body_imu(callback)
功能概述	订阅机体 IMU 数据
参数说明	callback: IMU 数据处理函数, 签名为 callback(data: Imu) -> None
备注	非阻塞接口。

16.1.20 unsubscribe_body_imu

项目	内容
函数名	unsubscribe_body_imu
函数声明	void unsubscribe_body_imu()
功能概述	取消订阅机体 IMU 数据
备注	非阻塞接口。与 subscribe_body_imu 配合使用。

16.2 类型定义

16.2.1 SingleHandJointCommand —单个手部关节的控制命令

字段名	类型	描述
operation_mode	int	控制字
pos	list[float]	期望位置数组 (7 个自由度, 单位: rad)

- 灵巧手 operation_mode 需要从模式: 200 切换到模式: 4 启动, 并进行指令下发;

- 灵巧手关节指令范围值: [0-1000]

16.2.2 HandCommand — 整个手部控制命令

字段名	类型	描述
timestamp	int	时间戳 (单位: 纳秒)
cmd	list[SingleHandJointCommand]	控制命令数组, 依次为左手和右手

16.2.3 SingleHandJointState — 单个手部关节的状态

字段名	类型	描述
status_word	int	状态字
pos	list[float]	当前位置 (单位: rad)
toq	list[float]	当前力矩 (单位: Nm)
cur	list[float]	当前电流 (单位: A)
error_code	int	错误码

16.2.4 HandState — 整个手部状态信息

字段名	类型	描述
timestamp	int	时间戳 (单位: 纳秒)
state	list[SingleHandJointState]	所有手部关节状态 (左手、右手顺序)

16.2.5 SingleJointCommand — 单个关节的控制命令

字段名	类型	描述
operation_mode	int	控制模式标识: • 200 = 准备状态 • 3 = 并联三环控制 (MIT 模式) • 4 = 串联三环控制
pos	float	期望位置 (单位: rad)
vel	float	期望速度 (单位: rad/s)
toq	float	期望力矩 (单位: Nm)
kp	float	位置增益
kd	float	速度增益

- 上臂 1-5 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;

- 上臂 6-7 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 头部 1-2 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 腰部 1-2 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 4 (串联三环控制模式) 进行指令下发;
- 腿部 1-6 关节 operation_mode 需要从模式: 200 (准备状态) 切换到模式: 3 (并联三环控制 (MIT 模式)) 进行指令下发;

16.2.6 JointCommand —关节控制命令

字段名	类型	描述
timestamp	int	时间戳 (单位: 纳秒)
joints	list[SingleJointCommand]	关节控制命令数组

16.2.7 SingleJointState —单个关节的状态

字段名	类型	描述
operation_mode	int	当前控制模式
pos	float	当前位置 (单位: rad)
vel	float	当前速度 (单位: rad/s)
toq	float	当前力矩 (单位: Nm)
cur	float	当前电流 (单位: A)
error_code	int	错误码

16.2.8 JointState —关节状态

字段名	类型	描述
timestamp	int	时间戳 (单位: 纳秒)
joints	list[SingleJointState]	关节状态数组

16.2.9 关节电机顺序

头部关节

索引	关节名
0	head_yaw_joint
1	head_pitch_joint

上臂关节

索引	关节名
0	left_shoulder_pitch_joint
1	left_shoulder_roll_joint
2	left_shoulder_yaw_joint
3	left_elbow_roll_joint
4	left_wrist_yaw_joint
5	left_wrist_roll_joint
6	left_wrist_pitch_joint
7	right_shoulder_pitch_joint
8	right_shoulder_roll_joint
9	right_shoulder_yaw_joint
10	right_elbow_roll_joint
11	right_wrist_yaw_joint
12	right_wrist_roll_joint
13	right_wrist_pitch_joint

腰部关节

索引	关节名
0	waist_roll_joint
1	waist_yaw_joint

腿部关节

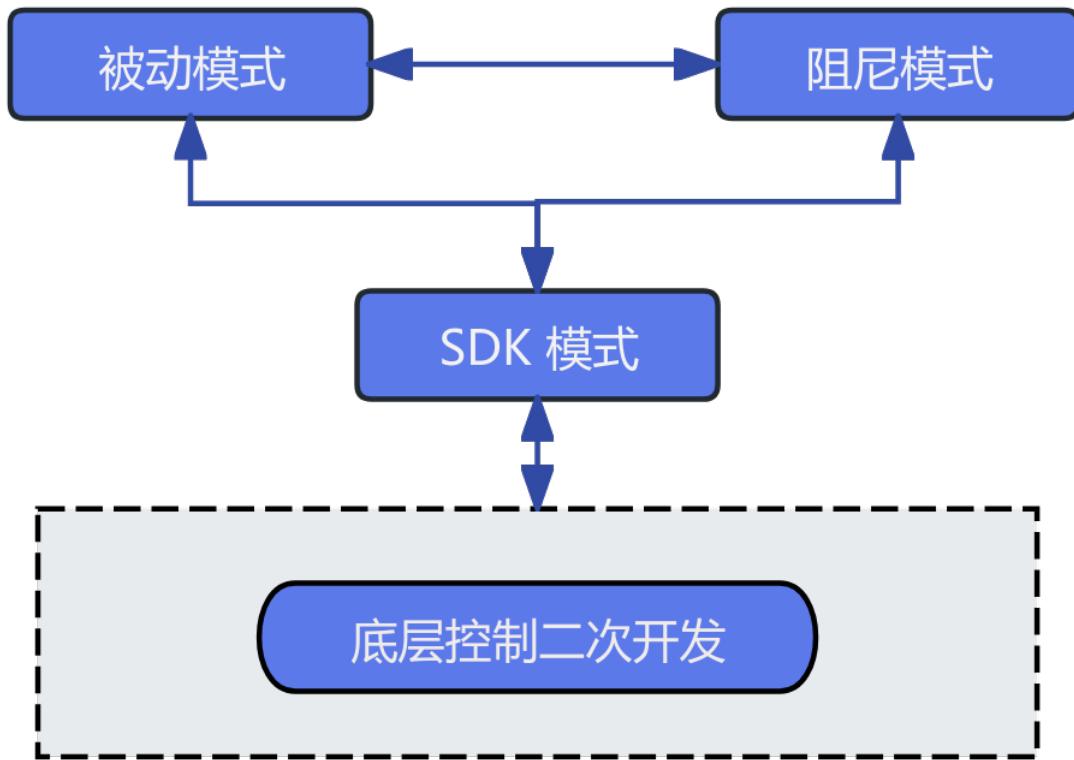
索引	关节名
0	left_hip_roll_joint
1	left_hip_yaw_joint
2	left_hip_pitch_joint
3	left_knee_pitch_joint
4	left_ankle_pitch_joint
5	left_ankle_roll_joint
6	right_hip_roll_joint
7	right_hip_yaw_joint
8	right_hip_pitch_joint
9	right_knee_pitch_joint
10	right_ankle_pitch_joint
11	right_ankle_roll_joint

16.3 URDF 参考

机器人 URDF

16.4 底层运动控制机器人状态介绍

机器人底层运动主要是开发关节的三环控制给开发人员进行机器人运动能力的二次开发，基本的控制状态切换机制：



16.5 注意事项

在使用 `Subscribe` 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

传感器控制服务 Python

提供机器人系统传感器（雷达/rgbd 相机/三目相机）服务，通过 SensorController 可以通过 RPC 和话题方式实现对机器人的传感器进行指令控制和状态获取。

17.1 接口定义

SensorController 是封装机器人各类传感器的管理类，支持 Lidar、RGBD 相机与三目相机的初始化、控制与数据订阅。

△ **Notice:** 当前三目、RGBD 传感器接口暂未完全支持，请勿使用。

17.1.1 SensorController

项目	内容
类名	SensorController
构造函数	controller = SensorController()
功能概述	构造函数，初始化传感器控制器对象。
备注	构造内部状态。

17.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize()
功能概述	初始化控制器，包括资源申请和驱动加载。
返回值	True 表示成功，False 表示失败。
备注	调用前需先构造对象。

17.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭所有传感器连接并释放资源。
备注	配合 initialize 使用。

17.1.4 open_lidar

项目	内容
函数名	open_lidar
函数声明	Status open_lidar()
功能概述	打开雷达。
返回值	Status 对象，Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口。

17.1.5 close_lidar

项目	内容
函数名	close_lidar
函数声明	Status close_lidar()
功能概述	关闭雷达。
返回值	Status 对象，Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口，配合打开函数使用。

17.1.6 open_head_rgbd_camera

项目	内容
函数名	open_head_rgbd_camera
函数声明	Status open_head_rgbd_camera()
功能概述	打开头部 RGBD 相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口。

17.1.7 close_head_rgbd_camera

项目	内容
函数名	close_head_rgbd_camera
函数声明	Status close_head_rgbd_camera()
功能概述	关闭头部 RGBD 相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 配合打开函数使用。

17.1.8 open_waist_rgbd_camera

项目	内容
函数名	open_waist_rgbd_camera
函数声明	Status open_waist_rgbd_camera()
功能概述	打开腰部 RGBD 相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口。

17.1.9 close_waist_rgbd_camera

项目	内容
函数名	close_waist_rgbd_camera
函数声明	Status close_waist_rgbd_camera()
功能概述	关闭腰部 RGBD 相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 配合打开函数使用。

17.1.10 open_trinocular_camera

项目	内容
函数名	open_trinocular_camera
函数声明	Status open_trinocular_camera()
功能概述	打开三目相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口。

17.1.11 close_trinocular_camera

项目	内容
函数名	close_trinocular_camera
函数声明	Status close_trinocular_camera()
功能概述	关闭三目相机。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口。

17.1.12 subscribe_lidar_imu

项目	内容
函数名	subscribe_lidar_imu
函数声明	void subscribe_lidar_imu(callback)
功能概述	订阅雷达 IMU 数据
参数说明	callback: 接收到数据后的处理函数, 签名为 callback(data: Imu) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.13 unsubscribe_lidar_imu

项目	内容
函数名	unsubscribe_lidar_imu
函数声明	void unsubscribe_lidar_imu()
功能概述	取消订阅雷达 IMU 数据
备注	非阻塞接口。与 subscribe_lidar_imu 配合使用。

17.1.14 subscribe_lidar_point_cloud

项目	内容
函数名	subscribe_lidar_point_cloud
函数声明	void subscribe_lidar_point_cloud(callback)
功能概述	订阅雷达点云数据
参数说明	callback: 接收到点云后的处理函数, 签名为 callback(data: PointCloud2) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.15 unsubscribe_lidar_point_cloud

项目	内容
函数名	unsubscribe_lidar_point_cloud
函数声明	void unsubscribe_lidar_point_cloud()
功能概述	取消订阅雷达点云数据
备注	非阻塞接口。与 subscribe_lidar_point_cloud 配合使用。

17.1.16 subscribe_head_rgbd_color_image

项目	内容
函数名	subscribe_head_rgbd_color_image
函数声明	void subscribe_head_rgbd_color_image(callback)
功能概述	订阅头部 RGBD 彩色图像数据
参数说明	callback: 接收到图像后的处理函数, 签名为 callback(data: Image) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.17 unsubscribe_head_rgbd_color_image

项目	内容
函数名	unsubscribe_head_rgbd_color_image
函数声明	void unsubscribe_head_rgbd_color_image()
功能概述	取消订阅头部 RGBD 彩色图像数据
备注	非阻塞接口。与 subscribe_head_rgbd_color_image 配合使用。

17.1.18 subscribe_head_rgbd_color_camera_info

项目	内容
函数名	subscribe_head_rgbd_color_camera_info
函数声明	void subscribe_head_rgbd_color_camera_info(callback)
功能概述	订阅头部 RGBD 彩色相机参数
参数说明	callback: 接收到参数后的处理函数, 签名为 callback(data: CameraInfo) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.19 unsubscribe_head_rgbd_color_camera_info

项目	内容
函数名	unsubscribe_head_rgbd_color_camera_info
函数声明	void unsubscribe_head_rgbd_color_camera_info()
功能概述	取消订阅头部 RGBD 彩色相机参数
备注	非阻塞接口。与 subscribe_head_rgbd_color_camera_info 配合使用。

17.1.20 subscribe_head_rgbd_depth_image

项目	内容
函数名	subscribe_head_rgbd_depth_image
函数声明	void subscribe_head_rgbd_depth_image(callback)
功能概述	订阅头部 RGBD 深度图像数据
参数说明	callback: 接收到图像后的处理函数, 签名为 callback(data: Image) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.21 unsubscribe_head_rgbd_depth_image

项目	内容
函数名	unsubscribe_head_rgbd_depth_image
函数声明	void unsubscribe_head_rgbd_depth_image()
功能概述	取消订阅头部 RGBD 深度图像数据
备注	非阻塞接口。与 subscribe_head_rgbd_depth_image 配合使用。

17.1.22 subscribe_head_rgbd_depth_camera_info

项目	内容
函数名	subscribe_head_rgbd_depth_camera_info
函数声明	void subscribe_head_rgbd_depth_camera_info(callback)
功能概述	订阅头部 RGBD 深度相机参数
参数说明	callback: 接收到参数后的处理函数, 签名为 callback(data: CameraInfo) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.23 unsubscribe_head_rgbd_depth_camera_info

项目	内容
函数名	unsubscribe_head_rgbd_depth_camera_info
函数声明	void unsubscribe_head_rgbd_depth_camera_info()
功能概述	取消订阅头部 RGBD 深度相机参数
备注	非阻塞接口。与 subscribe_head_rgbd_depth_camera_info 配合使用。

17.1.24 subscribe_waist_rgbd_color_image

项目	内容
函数名	subscribe_waist_rgbd_color_image
函数声明	void subscribe_waist_rgbd_color_image(callback)
功能概述	订阅腰部 RGBD 彩色图像数据
参数说明	callback: 接收到图像后的处理函数, 签名为 callback(data: Image) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.25 unsubscribe_waist_rgbd_color_image

项目	内容
函数名	unsubscribe_waist_rgbd_color_image
函数声明	void unsubscribe_waist_rgbd_color_image()
功能概述	取消订阅腰部 RGBD 彩色图像数据
备注	非阻塞接口。与 subscribe_waist_rgbd_color_image 配合使用。

17.1.26 subscribe_waist_rgbd_color_camera_info

项目	内容
函数名	subscribe_waist_rgbd_color_camera_info
函数声明	void subscribe_waist_rgbd_color_camera_info(callback)
功能概述	订阅腰部 RGBD 彩色相机参数
参数说明	callback: 接收到参数后的处理函数, 签名为 callback(data: CameraInfo) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.27 unsubscribe_waist_rgbd_color_camera_info

项目	内容
函数名	unsubscribe_waist_rgbd_color_camera_info
函数声明	void unsubscribe_waist_rgbd_color_camera_info()
功能概述	取消订阅腰部 RGBD 彩色相机参数
备注	非阻塞接口。与 subscribe_waist_rgbd_color_camera_info 配合使用。

17.1.28 subscribe_waist_rgbd_depth_image

项目	内容
函数名	subscribe_waist_rgbd_depth_image
函数声明	void subscribe_waist_rgbd_depth_image(callback)
功能概述	订阅腰部 RGBD 深度图像数据
参数说明	callback: 接收到图像后的处理函数, 签名为 callback(data: Image) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.29 unsubscribe_waist_rgbd_depth_image

项目	内容
函数名	unsubscribe_waist_rgbd_depth_image
函数声明	void unsubscribe_waist_rgbd_depth_image()
功能概述	取消订阅腰部 RGBD 深度图像数据
备注	非阻塞接口。与 subscribe_waist_rgbd_depth_image 配合使用。

17.1.30 subscribe_waist_rgbd_depth_camera_info

项目	内容
函数名	subscribe_waist_rgbd_depth_camera_info
函数声明	void subscribe_waist_rgbd_depth_camera_info(callback)
功能概述	订阅腰部 RGBD 深度相机参数
参数说明	callback: 接收到参数后的处理函数, 签名为 callback(data: CameraInfo) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.31 unsubscribe_waist_rgbd_depth_camera_info

项目	内容
函数名	unsubscribe_waist_rgbd_depth_camera_info
函数声明	void unsubscribe_waist_rgbd_depth_camera_info()
功能概述	取消订阅腰部 RGBD 深度相机参数
备注	非阻塞接口。与 subscribe_waist_rgbd_depth_camera_info 配合使用。

17.1.32 subscribe_trinocular_image

项目	内容
函数名	subscribe_trinocular_image
函数声明	void subscribe_trinocular_image(callback)
功能概述	订阅三目相机图像帧
参数说明	callback: 接收到图像帧后的处理函数, 签名为 callback(data: TrinocularCameraFrame) -> None
备注	非阻塞接口, 回调函数会在数据更新时被调用。

17.1.33 unsubscribe_trinocular_image

项目	内容
函数名	unsubscribe_trinocular_image
函数声明	void unsubscribe_trinocular_image()
功能概述	取消订阅三目相机图像帧
备注	非阻塞接口。与 subscribe_trinocular_image 配合使用。

17.2 类型定义

17.2.1 Imu —IMU 数据结构体

字段名	类型	描述
timestamp	int64	时间戳 (单位: 纳秒)
orientation	list[float]	姿态四元数 (w, x, y, z)
angular_velocity	list[float]	角速度 (单位: rad/s)
linear_acceleration	list[float]	线加速度 (单位: m/s ²)
temperature	float	温度 (单位: 摄氏度或其他, 应明确单位)

17.2.2 Header —通用消息头结构体

字段名	类型	描述
stamp	int64	时间戳 (单位: 纳秒)
frame_id	str	坐标系名称

17.2.3 PointField —点字段描述

字段名	类型	描述
name	str	字段名 (如 x、y、z、intensity)
offset	int	起始字节偏移
datatype	int8	数据类型 (对应常量)
count	int	每点此字段包含元素数量

17.2.4 PointCloud2 —点云数据结构体

字段名	类型	描述
header	Header	消息头
height	int	行数
width	int	列数
fields	list[PointField]	点字段数组
is_bigendian	bool	字节序
point_step	int	每个点的字节数
row_step	int	每行的字节数
data	bytes	原始点云字节数据
is_dense	bool	是否稠密点云

17.2.5 Image — 图像数据结构体

字段名	类型	描述
header	Header	消息头
height	int	图像高度 (像素)
width	int	图像宽度 (像素)
encoding	str	编码类型 (如 rgb8, mono8)
is_little_endian	bool	是否为小端模式
step	int	每行图像占用字节数
data	bytes	原始图像字节数据

17.2.6 CameraInfo — 相机内参与畸变信息

字段名	类型	描述
header	Header	消息头
height	int	图像高度
width	int	图像宽度
distortion_model	str	畸变模型 (如 plumb_bob)
D	list[float]	畸变参数数组
K	list[float]	相机内参矩阵 (3x3)
R	list[float]	矫正矩阵 (3x3)
P	list[float]	投影矩阵 (3x4)
binning_x	int	水平 binning 系数
binning_y	int	垂直 binning 系数
roi_x_offset	int	ROI 起始 x 坐标
roi_y_offset	int	ROI 起始 y 坐标
roi_height	int	ROI 高度
roi_width	int	ROI 宽度
roi_do_rectify	bool	是否进行矫正

17.2.7 TrinocularCameraFrame —三目相机帧数据结构

字段名	类型	描述
header	Header	通用消息头
vin_time	int64	图像采集时间（纳秒）
decode_time	int64	图像解码时间（纳秒）
imgfl_array	bytes	左目图像数据
imgf_array	bytes	中目图像数据
imgfr_array	bytes	右目图像数据

17.3 注意事项

在使用 Subscribe 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

CHAPTER 18

语音控制服务 Python

提供机器人系统语音服务控制器，通过 `AudioController` 可以通过 RPC 方式实现对机器人的音频进行指令控制和状态获取。

18.1 接口定义

`AudioController` 是一个封装音频控制功能的 Python 类，主要用于音频播放控制、TTS 播放、音量设置与查询等场景。

18.1.1 `AudioController`

项目	内容
类名	<code>AudioController</code>
构造函数	<code>controller = AudioController()</code>
功能概述	初始化音频控制器对象，构造内部状态，分配资源等。
备注	构造内部状态。

18.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize()
功能概述	初始化音频控制模块，准备播放资源与设备。
返回值	True 表示成功，False 表示失败。
备注	与 shutdown() 配对使用。

18.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭音频控制器并释放资源。
备注	确保在销毁前调用。

18.1.4 play

项目	内容
函数名	play
函数声明	Status play(TtsCommand cmd, int timeout_ms)
功能概述	播放 TTS（文本转语音）语音命令。
参数说明	cmd: TTS 命令，包含文本、语速、语调等。
返回值	Status 对象，Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口，调用前需确保已初始化模块。

18.1.5 stop

项目	内容
函数名	stop
函数声明	Status stop()
功能概述	停止当前音频播放。
返回值	Status 对象，Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口，通常用于中断当前语音。

18.1.6 set_volume

项目	内容
函数名	set_volume
函数声明	Status set_volume(int volume)
功能概述	设置音频输出的音量。
参数说明	volume: 音量值, 通常范围为 0~100。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 设置后立即生效。

18.1.7 get_volume

项目	内容
函数名	get_volume
函数声明	tuple[Status, int] get_volume()
功能概述	获取当前音频输出音量。
返回值	返回元组 (Status, volume), Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 返回值需检查后再使用 volume。

18.1.8 open_audio_stream

项目	内容
函数名	open_audio_stream
函数声明	Status open_audio_stream()
功能概述	打开音频流, 准备进行音频播放。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 返回值需检查是否执行成功

18.1.9 close_audio_stream

项目	内容
函数名	close_audio_stream
函数声明	Status close_audio_stream()
功能概述	关闭音频流。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 返回值需检查是否执行成功

18.1.10 subscribe_origin_audio_stream

项目	内容
函数名	subscribe_origin_audio_stream
函数声明	void subscribe_origin_audio_stream(callback)
功能概述	订阅原始音频流数据。
参数说明	callback: 接收到数据后的处理函数, 签名为 callback(data: AudioStream) -> None
备注	非阻塞接口。

18.1.11 unsubscribe_origin_audio_stream

项目	内容
函数名	unsubscribe_origin_audio_stream
函数声明	void unsubscribe_origin_audio_stream()
功能概述	取消订阅原始音频流数据。
备注	非阻塞接口。与 subscribe_origin_audio_stream 配合使用。

18.1.12 subscribe_bf_audio_stream

项目	内容
函数名	subscribe_bf_audio_stream
函数声明	void subscribe_bf_audio_stream(callback)
功能概述	订阅 BF 音频流数据。
参数说明	callback: 接收到数据后的处理函数, 签名为 callback(data: AudioStream) -> None
备注	非阻塞接口。

18.1.13 unsubscribe_bf_audio_stream

项目	内容
函数名	unsubscribe_bf_audio_stream
函数声明	void unsubscribe_bf_audio_stream()
功能概述	取消订阅 BF 音频流数据。
备注	非阻塞接口。与 subscribe_bf_audio_stream 配合使用。

18.1.14 open_wakeup_status_stream

项目	内容
函数名	open_wakeup_status_stream
函数声明	Status open_wakeup_status_stream()
功能概述	打开语音唤醒状态流。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 返回值需检查是否执行成功。

18.1.15 close_wakeup_status_stream

项目	内容
函数名	close_wakeup_status_stream
函数声明	Status close_wakeup_status_stream()
功能概述	关闭语音唤醒状态流。
返回值	Status 对象, Status.code == ErrorCode.OK 表示成功。
备注	阻塞接口, 配合 open_wakeup_status_stream 使用。

18.1.16 subscribe_wakeup_status

项目	内容
函数名	subscribe_wakeup_status
函数声明	void subscribe_wakeup_status(callback)
功能概述	订阅语音唤醒状态数据。
参数说明	callback: 接收到唤醒状态后的处理函数, 签名为 callback(data: WakeupStatus) -> None
备注	非阻塞接口, 回调函数会在唤醒状态更新时被调用。

18.1.17 unsubscribe_wakeup_status

项目	内容
函数名	unsubscribe_wakeup_status
函数声明	void unsubscribe_wakeup_status()
功能概述	取消订阅语音唤醒状态数据。
备注	非阻塞接口。与 subscribe_wakeup_status 配合使用。

18.2 类型定义

18.2.1 TtsPriority —TTS 播报优先级等级

用于控制不同 TTS 任务之间的中断行为。优先级越高的任务将中断当前低优先级任务的播放。

枚举值	描述
TtsPriority.HIGH	最高优先级，例如：低电告警、紧急提醒
TtsPriority.MIDDLE	中优先级，例如：系统提示、状态播报
TtsPriority.LOW	最低优先级，例如：日常语音对话、背景播报

18.2.2 TtsMode —同一优先级下的任务调度策略

用于细化控制在相同优先级条件下多个 TTS 任务的播放顺序和清除逻辑。

枚举值	描述
TtsMode.CLEARTOP	清空当前优先级所有任务（包括正在播放和等待队列），立即播放本次请求
TtsMode.ADD	将本次请求追加到当前优先级队列尾部，顺序播放（不打断当前播放）
TtsMode.CLEARBUFFER	清空队列中未播放的请求，保留当前播放，之后播放本次请求

s## 结构体定义

18.2.3 TtsCommand —TTS 播放命令结构体

描述一次 TTS 播放请求的完整信息，支持设置唯一标识、文本内容、优先级控制以及同优先级下的调度模式。

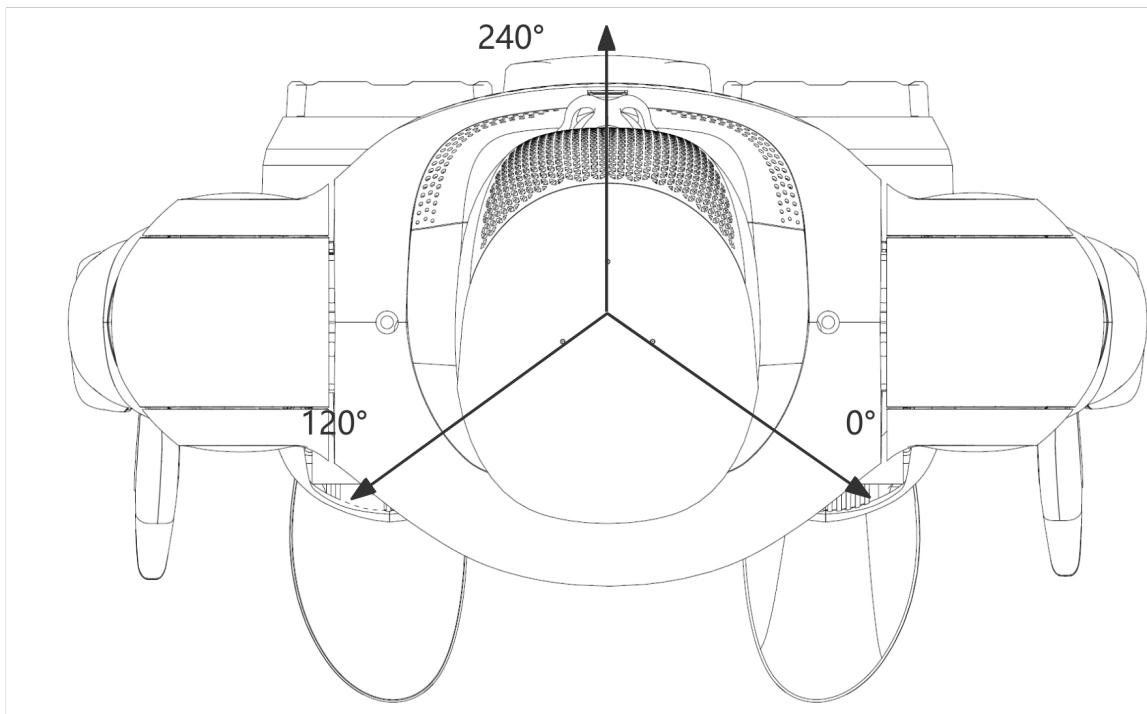
字段名	类型	描述
id	str	TTS 任务唯一 ID，例如 "id_01"，用于追踪播放状态
content	str	要播放的文本内容，例如 "你好，欢迎使用智能语音系统。"
priority	TtsPriority	播报优先级，控制是否中断正在播放的低优先级语音
mode	TtsMode	同优先级下的调度策略，控制任务是否追加、覆盖等

18.2.4 AudioStream - 音频流结构体

描述原始音频流或 BF 音频流数据。

字段名	类型	描述
data_length	int	数据长度
raw_data	bytes	音频流数据

18.3 DOA



18.4 注意事项

在使用 `Subscribe` 等订阅类 SDK 接口时，请避免在回调函数中执行阻塞式的数据处理操作。否则，可能导致消息堆积、处理延迟，甚至引发不可预期的错误。

CHAPTER 19

状态监控服务 Python

提供机器人状态监控接口类，通过 StateMonitor 可以提供状态查询等接口。

19.1 接口定义

StateMonitor 是机器人系统状态监控管理类，该类通常用于控制机器人的状态管理，支持状态查询。

19.1.1 StateMonitor

项目	内容
类名	StateMonitor
构造函数	controller = StateMonitor()
功能概述	构造函数，初始化控制器对象。
备注	构造内部状态。

19.1.2 initialize

项目	内容
函数名	initialize
函数声明	bool initialize()
功能概述	初始化控制器，包括资源申请和驱动加载。
返回值	True 表示成功，False 表示失败。
备注	调用前需先构造对象。

19.1.3 shutdown

项目	内容
函数名	shutdown
函数声明	void shutdown()
功能概述	关闭所有连接并释放资源。
备注	配合 initialize 使用。

19.1.4 get_current_state

项目	内容
函数名	get_current_state
函数声明	RobotState get_current_state()
功能概述	获取当前机器人聚合状态
返回值	tuple[Status, RobotState] 对象
备注	非阻塞接口，获取最近监控到的机器人状态数据，包括 BMS/故障状态监控等，后续会迭代拓展

19.1.5 RobotState —— 机器人状态数据结构体

用于表示机器人总体状态信息：

字段名	类型	描述
faults	list[Fault]	故障信息列表
bms_data	BmsData	电池管理系统数据

19.1.6 BmsData — 电池管理系统数据结构体

表示电池的状态信息：

字段名	类型	描述
battery_percentage	float	当前电池剩余电量（百分比，0~100）
battery_health	float	电池健康状态（越高越好）
battery_state	BatteryState	电池状态（见下方枚举）
power_supply_status	PowerSupplyStatus	电池充放电状态（见下方枚举）

19.1.7 枚举类型定义

BatteryState — 电池状态枚举类型

用于表示电池当前的状态，用于系统中电池状态的判断和处理：

枚举值	数值	描述
BatteryState.UNKNOWN	0	未知状态
BatteryState.GOOD	1	电池状态良好
BatteryState.OVERHEAT	2	电池过热
BatteryState.DEAD	3	电池损坏
BatteryState.OVERVOLTAGE	4	电池过电压
BatteryState.UNSPEC_FAILURE	5	未知故障
BatteryState.COLD	6	电池过冷
BatteryState.WATCHDOG_TIMER_EXPIRE	7	看门狗定时器超时
BatteryState.SAFETY_TIMER_EXPIRE	8	安全定时器超时

PowerSupplyStatus — 电池充放电状态

用于表示当前电池的充放电状态：

枚举值	数值	描述
PowerSupplyStatus.UNKNOWN	0	未知状态
PowerSupplyStatus.CHARGING	1	电池充电中
PowerSupplyStatus.DISCHARGING	2	电池放电中
PowerSupplyStatus.NOTCHARGING	3	电池未充放电
PowerSupplyStatus.FULL	4	电池充满

19.2 错误码映射表

错误码（十六进制）	错误描述
0x0000	No fault
0x1101	Service invocation failed
0x1301	Central control node lost
0x1302	App node lost
0x1303	Audio node lost
0x1304	Stereo camera node lost
0x1305	LIDAR node lost
0x1306	SLAM node lost
0x1307	Navigation node lost
0x1308	AI node lost
0x1309	Head node lost
0x130A	Point cloud node lost
0x2201	No LIDAR data received
0x2202	No stereo camera data received
0x2203	Stereo camera data error
0x2204	Stereo camera initialization failed
0x220B	No odometry data received
0x220C	No IMU data received
0x2215	Depth camera not detected
0x3101	Failed to connect robot to app
0x3102	Heartbeat lost - assertion failed
0x4201	Failed to open head serial port
0x4202	No head data received
0x5201	No navigation TF data
0x5202	No navigation map data
0x5203	No navigation localization data
0x5204	No navigation LIDAR data
0x5205	No navigation depth camera data
0x5206	No navigation multi-line LIDAR data
0x5207	No navigation odometry data
0x6201	SLAM localization error
0x6102	No SLAM LIDAR data
0x6103	No SLAM odometry data
0x6104	SLAM map data error
0x7201	LCM connection timeout
0x8201	Left leg hardware error

[下页继续](#)

表 1 - 续上页

错误码（十六进制）	错误描述
0x8202	Right leg hardware error
0x8203	Left arm hardware error
0x8204	Right arm hardware error
0x8205	Waist hardware error
0x8206	Head hardware error
0x8207	Hand hardware error
0x8208	Gripper hardware error
0x8209	IMU hardware error
0x820A	Power system hardware error
0x820B	Leg force sensor hardware error
0x820C	Arm force sensor hardware error
0x9201	ECAT (EtherCAT) hardware error
0xA201	Motion posture error
0xA202	Foot position deviation during movement
0xA203	Joint velocity error during motion

CHAPTER 20

SLAM 导航控制服务 Python

提供机器人系统 SLAM（同时定位与地图构建）和导航控制服务，通过 SlamNavController 可以通过 RPC 通信方式实现对机器人的建图、定位、导航等功能的控制。

20.1 接口定义

SlamNavController 是面向 SLAM 和导航控制的控制器，支持如建图、定位、导航、地图管理等控制操作，封装底层细节以供上层系统调用。

20.1.1 SlamNavController

项目	内容
类名	SlamNavController
类声明	SlamNavController()
功能概述	构造函数，初始化 SLAM 导航控制器状态
备注	构造内部控制资源

20.1.2 initialize

项目	内容
方法名	initialize
方法声明	bool initialize()
功能概述	初始化 SLAM 导航控制器，准备 SLAM 和导航功能
返回值	True 表示成功，False 表示失败
备注	首次使用前必须调用

20.1.3 shutdown

项目	内容
方法名	shutdown
方法声明	void shutdown()
功能概述	关闭控制器并释放资源
备注	配合 initialize 使用，安全断开连接

20.1.4 activate_slam_mode

项目	内容
方 法 名	activate_slam_mode
方 法 声 明	Status activate_slam_mode(SlamMode mode, str map_path = "", int timeout_ms)
功 能 概 述	激活 SLAM 模式并开始建图或定位模式
参 数 说 明	mode: SLAM 模式枚举 map_path: 地图绝对路径 (定位模式时使用), 可通过 get_map_path 获取 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返 回 值	Status::OK 表示成功, 其他为失败
备 注	阻塞接口, 支持建图和定位模式切换

20.1.5 start_mapping

项目	内容
方法名	start_mapping
方法声明	Status start_mapping(int timeout_ms)
功能概述	开始建图
参数说明	timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 需先激活建图模式

20.1.6 cancel_mapping

项目	内容
方法名	cancel_mapping
方法声明	Status cancel_mapping(int timeout_ms)
功能概述	取消建图
参数说明	timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 取消当前建图任务

20.1.7 save_map

项目	内容
方法名	save_map
方法声明	Status save_map(str map_name, int timeout_ms)
功能概述	结束建图并保存地图
参数说明	map_name: 地图名称 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 需在建图模式下调用

20.1.8 load_map

项目	内容
方法名	load_map
方法声明	Status load_map(str map_name, int timeout_ms)
功能概述	加载地图并设置为当前地图
参数说明	map_name: 地图名称 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 加载指定名称的地图

20.1.9 delete_map

项目	内容
方法名	delete_map
方法声明	Status delete_map(str map_name, int timeout_ms)
功能概述	删除地图
参数说明	map_name: 要删除的地图名称 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 永久删除指定地图

20.1.10 get_map_path

项目	内容
方法名	get_map_path
方法声明	tuple[Status, List[str]] get_map_path(str map_name, int timeout_ms)
功能概述	获取地图路径
参数说明	map_name: 地图名称 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败 List[str]: 地图路径 (输出参数)
备注	阻塞接口, 获取指定地图的存储路径

20.1.11 get_all_map_info

项目	内容
方法名	get_all_map_info
方法声明	tuple[Status, all_map_info] get_all_map_info(int timeout_ms)
功能概述	获取所有地图信息
参数说明	timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK: 表示成功, 其他为失败 all_map_info: 所有地图信息 (输出参数)
备注	阻塞接口, 获取系统中所有地图的详细信息

20.1.12 init_pose

项目	内容
方法名	init_pose
方法声明	Status init_pose(Pose3DEuler pose, int timeout_ms)
功能概述	初始化位姿
参数说明	pose: 要发布的位姿信息 timeout_ms: 超时时间 (毫秒), 默认值为 15000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 设置机器人初始位姿; 发布位姿时要对 Yaw 方向施加一个固定的-1.57rad 偏置, 因为激光雷达的机械安装与机器人本体存在一个-1.57rad 的偏置

20.1.13 get_current_localization_info

项目	内容
方法名	get_current_localization_info
方法声明	Status get_current_localization_info(LocalizationInfo pose_info, int timeout_ms)
功能概述	获取当前位姿信息
参数说明	pose_info: 当前位置和姿态信息 (输出参数) timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 获取机器人当前位姿状态

20.1.14 activate_nav_mode

项目	内容
方法名	activate_nav_mode
方法声明	Status activate_nav_mode(NavMode mode, str map_path = "", int timeout_ms)
功能概述	激活导航模式
参数说明	mode: 目标导航模式 map_path: 地图绝对路径 (GRID_MAP 模式时使用), 可通过 get_map_path
返回值	timeout_ms: 超时时间 (毫秒), 默认值为 10000
备注	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 激活指定导航模式

20.1.15 set_nav_target

项目	内容
方法名	set_nav_target
方法声明	Status set_nav_target(NetTarget goal, int timeout_ms)
功能概述	设置全局导航目标点并开始导航任务
参数说明	goal: 目标点的全局坐标 timeout_ms: 超时时间 (毫秒), 默认值为 10000
返回值	Status::OK 表示成功, 其他为失败
备注	阻塞接口, 设置导航目标并开始导航

20.1.16 pause_nav_task

项目	内容
方法名	pause_nav_task
方法声明	Status pause_nav_task()
功能概述	暂停当前导航任务
返回值	Status::OK 表示成功, 其他为失败
备注	非阻塞接口, 暂停导航任务

20.1.17 resume

项目	内容
方法名	resume
方法声明	Status resume()
功能概述	恢复暂停的导航任务
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，恢复暂停的导航任务

20.1.18 cancel_nav_task

项目	内容
方法名	cancel_nav_task
方法声明	Status cancel_nav_task()
功能概述	取消当前导航任务
返回值	Status::OK 表示成功，其他为失败
备注	非阻塞接口，取消当前导航任务

20.1.19 get_nav_task_status

项目	内容
方法名	get_nav_task_status
方法声明	tuple[Status, NavStatus] get_nav_task_status()
功能概述	获取当前导航任务的状态信息。
返回值	status::OK 表示成功，其他为失败，同时返回 NavStatus 对象（失败时为空对象）。
备注	非阻塞接口，用于查询导航任务状态。调用后返回 status 状态码与 NavStatus 对象，可用于检测导航进度及状态。

20.1.20 open_odometry_stream

项目	内容
函数名	open_odometry_stream
函数声明	status open_odometry_stream()
功能概述	打开里程计数据流
返回值	status 对象, Status.code == ErrorCode.OK 表示成功
备注	阻塞接口, 返回值需检查是否执行成功

20.1.21 close_odometry_stream

项目	内容
函数名	close_odometry_stream
函数声明	status close_odometry_stream()
功能概述	关闭里程计数据流
返回值	status 对象, Status.code == ErrorCode.OK 表示成功
备注	阻塞接口, 配合 open_odometry_stream 使用

20.1.22 subscribe_odometry

项目	内容
函数名	subscribe_odometry
函数声明	void subscribe_odometry(callback)
功能概述	订阅里程计数据
参数说明	callback: 接收到里程计数据后的处理函数, 签名为 callback(data: Odometry) -> None
备注	非阻塞接口, 回调函数会在里程计数据更新时被调用

20.1.23 unsubscribe_odometry

项目	内容
函数名	unsubscribe_odometry
函数声明	void unsubscribe_odometry()
功能概述	取消订阅里程计数据
备注	非阻塞接口。与 subscribe_odometry 配合使用

20.1.24 get_point_cloud_map

项目	内容
函数名	get_point_cloud_map
函数声明	tuple[Status, PointCloud2] get_point_cloud_map(int timeout_ms)
功能概述	获取点云地图数据
参数说明	timeout_ms: 超时时间 (毫秒)
返回值	返回元组, 第一个元素为 Status 对象, Status.code == ErrorCode.OK 表示成功; 第二个元素为点云地图数据
备注	阻塞接口, 获取当前点云地图数据

20.2 类型定义

20.2.1 Pose3DEuler —3D 位姿结构体

字段名	类型	描述
position	List[float]	位置坐标 (x, y, z)
orientation	List[float]	欧拉角 (roll, pitch, yaw)

20.2.2 NavTarget —导航目标点结构体

字段名	类型	描述
id	int	目标点 ID
frame_id	str	目标点坐标系 ID
goal	Pose3DEuler	目标点位姿

20.2.3 LocalizationInfo —位姿信息结构体

字段名	类型	描述
is_localization	bool	是否已定位
pose	Pose3DEuler	当前位姿

20.2.4 NavStatus —导航状态结构体

字段名	类型	描述
id	int	目标点 ID, -1 表示无目标点
status	NavStatusType	导航状态类型
message	str	导航状态消息

20.2.5 PolyRegion —多边形区域结构体

字段名	类型	描述
points	List[Point2D]	多边形顶点列表

20.2.6 Point2D —2D 点结构体

字段名	类型	描述
x	float	X 坐标
y	float	Y 坐标

20.3 枚举类型定义

20.3.1 SlamMode —SLAM 模式枚举

枚举值	数值	描述
IDLE	0	空闲模式
LOCALIZATION	1	定位模式
MAPPING	2	建图模式

20.3.2 NavMode —导航模式枚举

枚举值	数值	描述
IDLE	0	空闲模式
GRID_MAP	1	网格地图导航模式

20.3.3 NavStatusType — 导航状态类型枚举

枚举值	数值	描述
NONE	0	无状态
RUNNING	1	运行中
END_SUCCESS	2	成功结束
END_FAILED	3	失败结束
PAUSE	4	暂停
CONTINUE	5	继续
CANCEL	6	取消

20.4 SLAM 导航控制介绍

机器人的 SLAM 导航控制服务可分为 SLAM 功能和导航功能。

- SLAM 功能中，可调用相应的接口，实现建图、定位、地图管理等功能。
- 导航功能中，可调用相应的接口，实现目标导航、导航控制等功能。

20.4.1 适用场景与范围

- 小于 100m * 100m 且特征丰富的静态室内平地场景，请勿超过建议范围
- 该部分功能适用于教育科研行业，不建议用于行业应用，行业应用请联系销售

20.4.2 SLAM 功能流程

功能	操作流程
建图	激活建图模式 → 开始建图 → 保存地图
地图管理	加载地图、获取地图信息、删除地图、获取地图绝对路径

20.4.3 导航功能流程

功能	操作流程
定位	加载地图 → 激活定位模式 → 初始化位姿 → 获取定位状态
导航	定位成功 → 激活导航模式 → 设置目标姿态(开始导航任务) → 获取导航状态
导航控制	设置目标姿态(开始导航任务) → 暂停导航 → 恢复导航 → 取消导航

20.4.4 定位模式初始化

SLAM 处于定位模式时，可以指定初始位姿以快速完成重定位与初始化，在使用大规模地图时必须指定初始位姿：

小地图模式

满足如下条件：

- 条件一：地图不能太大例如在一个房间内，地图面积最好控制在 10m * 10m 以内；
- 条件二，修改配置适配该模式：

```
# 登录机器人本体
ssh eame@192.168.54.119
# 修改如下配置
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
reloc_options: true

# 首次更改yaml文件后需要重启SLAM模块生效，重启指令为：
sudo systemctl restart slam.service
```

大地图模式

满足如下条件：

- 条件一：机器人必须处于建图原点（建图开始时机器人的出发点），机器人朝向也要保持一致
- 条件二：修改配置适配该模式：

SDK 方式 SLAM 导航不同于 App 端方便可视化进行定位，需要修改 SLAM 配置，固定地图方向，方便 SLAM 定位：

```
# 登录机器人本体
ssh eame@192.168.54.119
# 修改如下配置
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# 首次更改yaml文件后需要重启SLAM模块生效，重启指令为：
sudo systemctl restart slam.service
```

- 条件三：初始化位姿时，需要对 yaw 角施加一个-90°(-1.57rad) 偏置：

因为激光雷达的机械安装与机器人本体相差-90°

20.4.5 地图噪点擦除以及添加障碍

- 软件名称: GIMP
- 下载地址: <https://www.gimp.org/downloads/>
- 操作系统: Ubuntu
- 编辑对象: /home/eame/cust_para/maps/\${map_name}/map/map.pgm 登录机器人本体 (ssh eame@192.168.54.119), 将.pgm 地图文件通过 GIMP 进行编辑, 编辑完成后进行替换;
- 编辑视频:

CHAPTER 21

高层运动控制示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行初始化、连接机器人、高层运动控制（步态、特技、遥控）等基本操作。

21.1 C++

示例文件: high_level_motion_example.cpp

```
#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <termios.h>
#include <unistd.h>
#include <csignal>

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
```

(下页继续)

(续上页)

```

robot.Shutdown();
// Exit process
exit(signum);
}

void print_help() {
    std::cout << "Key Function Description:\n";
    std::cout << "Gait and Trick Functions:\n";
    std::cout << "  1      Function 1: Recovery stand\n";
    std::cout << "  2      Function 2: Balance stand\n";
    std::cout << "  3      Function 3: Execute trick - celebrate action\n";
    std::cout << "\n";
    std::cout << "Joystick Functions:\n";
    std::cout << "  w      Function w: Move forward\n";
    std::cout << "  a      Function a: Move left\n";
    std::cout << "  s      Function s: Move backward\n";
    std::cout << "  d      Function d: Move right\n";
    std::cout << "  x      Function x: Stop\n";
    std::cout << "  t      Function t: Turn left\n";
    std::cout << "  g      Function g: Turn right\n";
    std::cout << "\n";
    std::cout << "Head Functions:\n";
    std::cout << "  b      Function b: Head look up\n";
    std::cout << "  j      Function j: Head look down\n";
    std::cout << "  k      Function k: Head turn left\n";
    std::cout << "  l      Function l: Head turn right\n";
    std::cout << "  u      Function u: Head reset\n";
    std::cout << "\n";
    std::cout << "  ESC    Exit program\n";
    std::cout << "  ?      Function ?: Print help\n";
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt); // Get current terminal settings
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO); // Disable buffering and echo
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar(); // Read key press
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt); // Restore settings
    return ch;
}

```

(下页继续)

(续上页)

```

void RecoveryStand() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Set gait
    auto status = controller.SetGait(GaitMode::GAIT_RECOVERY_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "set robot gait failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
}

void BalanceStand() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Set posture display gait
    auto status = controller.SetGait(GaitMode::GAIT_BALANCE_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "set robot gait failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "robot gait set to GAIT_BALANCE_STAND successfully." << std::endl;
}

void ExecuteTrick() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Execute trick
    auto status = controller.ExecuteTrick(TrickAction::ACTION_CELEBRATE, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "execute robot trick failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "robot trick executed successfully." << std::endl;
}

```

(下页继续)

(续上页)

```

}

void JoyStickCommand(float left_x_axis,
                      float left_y_axis,
                      float right_x_axis,
                      float right_y_axis) {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    JoystickCommand joy_command;
    joy_command.left_x_axis = left_x_axis;
    joy_command.left_y_axis = left_y_axis;
    joy_command.right_x_axis = right_x_axis;
    joy_command.right_y_axis = right_y_axis;
    auto status = controller.SendJoyStickCommand(joy_command);
    if (status.code != ErrorCode::OK) {
        std::cerr << "execute robot trick failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        usleep(50000);
        return;
    }
    // Wait 50ms
    usleep(50000);
}

void HeadLookUp() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Head look up
    auto status = controller.HeadMove(0.0, 0.1, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "head look up failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "head look up successfully." << std::endl;
}

void HeadLookDown() {
    // Get high-level motion controller
}

```

(下页继续)

(续上页)

```
auto& controller = robot.GetHighLevelMotionController();

// Head look down
auto status = controller.HeadMove(0.0, -0.1, 10000);
if (status.code != ErrorCode::OK) {
    std::cerr << "head look down failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}
std::cout << "head look down successfully." << std::endl;
}

void HeadTurnLeft() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Head turn left
    auto status = controller.HeadMove(-0.2, 0.0, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "head turn left failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "head turn left successfully." << std::endl;
}

void HeadTurnRight() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Head turn right
    auto status = controller.HeadMove(0.2, 0.0, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "head turn right failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "head turn right successfully." << std::endl;
}
```

(下页继续)

(续上页)

```

void HeadReset() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Head reset
    auto status = controller.HeadMove(0.0, 0.0, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "head reset failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "head reset successfully." << std::endl;
}

int main(int argc, char* argv[]) {
    // Bind SIGINT (Ctrl+C)
    signal(SIGINT, signalHandler);

    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;

    print_help();

    std::string local_ip = "192.168.54.111";
    // Configure local IP address for direct network connection and initialize SDK
    if (!robot.Initialize(local_ip)) {
        std::cerr << "robot sdk initialize failed." << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Connect to robot
    auto status = robot.Connect();
    if (status.code != ErrorCode::OK) {
        std::cerr << "connect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Switch motion control controller to high-level controller, default is high-level controller
    status = robot.SetMotionControlLevel(ControllerLevel::HighLevel);
}

```

(下页继续)

(续上页)

```

if (status.code != ErrorCode::OK) {
    std::cerr << "switch robot motion control level failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}

std::cout << "Press any key to continue (ESC to exit)..." 
        << std::endl;

// Wait for user input
while (1) {
    int key = getch();
    if (key == 27)
        break; // ESC key ASCII code is 27

    std::cout << "Key ASCII: " << key << ", Character: " << static_cast<char>(key) << std::endl;
    switch (key) {
        // 1. Gait and Trick Functions
        case '1': {
            RecoveryStand();
            break;
        }
        case '2': {
            BalanceStand();
            break;
        }
        case '3': {
            ExecuteTrick();
            break;
        }
        // 2. Joystick Functions
        case 'w':
        case 'W': {
            JoyStickCommand(0.0, 1.0, 0.0, 0.0); // Forward
            break;
        }
        case 'a':
        case 'A': {
            JoyStickCommand(-1.0, 0.0, 0.0, 0.0); // Left
            break;
        }
    }
}

```

(下页继续)

(续上页)

```

case 's':
case 'S': {
    JoyStickCommand(0.0, -1.0, 0.0, 0.0); // Backward
    break;
}
case 'd':
case 'D': {
    JoyStickCommand(1.0, 0.0, 0.0, 0.0); // Right
    break;
}
case 'x':
case 'X': {
    JoyStickCommand(0.0, 0.0, 0.0, 0.0); // Stop
    break;
}
case 't':
case 'T': {
    JoyStickCommand(0.0, 0.0, -1.0, 0.0); // Turn left
    break;
}
case 'g':
case 'G': {
    JoyStickCommand(0.0, 0.0, 1.0, 0.0); // Turn right
    break;
}

// 3. Head Functions
case 'b': {
    HeadLookUp();
    break;
}
case 'j': {
    HeadLookDown();
    break;
}
case 'k': {
    HeadTurnLeft();
    break;
}
case 'l': {
    HeadTurnRight();
    break;
}
case 'u': {

```

(下页继续)

(续上页)

```

        HeadReset();
        break;
    }
    case '?': {
        print_help();
        break;
    }
    default:
        std::cout << "Unknown key: " << key << std::endl;
        break;
    }
    usleep(10000);
}

// Disconnect from robot
status = robot.Disconnect();
if (status.code != ErrorCode::OK) {
    std::cerr << "disconnect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}

robot.Shutdown();

return 0;
}

```

21.2 Python

示例文件: high_level_motion_example.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import time
import signal
import logging
import termios
import tty
from typing import Optional

```

(下页继续)

(续上页)

```

import magicbot_gen1_python as magicbot

# Configure logging format and level
logging.basicConfig(
    level=logging.INFO,    # Minimum log level
    format="%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True

def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global running, robot
    logging.info("Received interrupt signal (%s), exiting...", signum)
    running = False
    if robot:
        robot.shutdown()
    logging.info("Robot shutdown")
    exit(-1)

def print_help():
    """Print help information"""
    logging.info("High-Level Motion Control Function Demo Program")
    logging.info("")
    logging.info("Gait and Trick Functions:")
    logging.info("  0      Function 0: Pure damper")
    logging.info("  1      Function 1: Recovery stand")
    logging.info("  2      Function 2: Balance stand")
    logging.info("  3      Function 3: Execute trick - celebrate action")
    logging.info("")
    logging.info("Joystick Functions:")
    logging.info("  w      Function w: Move forward")
    logging.info("  a      Function a: Move left")
    logging.info("  s      Function s: Move backward")
    logging.info("  d      Function d: Move right")
    logging.info("  x      Function x: Stop move")
    logging.info("  t      Function t: Turn left")

```

(下页继续)

(续上页)

```

logging.info(" g           Function g: Turn right")
logging.info(" ")
logging.info("Head Functions:")
logging.info(" b           Function b: Head look up")
logging.info(" j           Function j: Head look down")
logging.info(" k           Function k: Head turn left")
logging.info(" l           Function l: Head turn right")
logging.info(" u           Function u: Head reset")
logging.info(" ")
logging.info(" ?           Function ?: Print help")
logging.info(" ESC         Exit program")

def get_user_input():
    """Get user input - Read a single line of data"""
    try:
        # Method 1: Read a line using input() (recommended)
        return input("Enter command: ").strip()
    except (EOFError, KeyboardInterrupt):
        return ""

def recovery_stand():
    """Recovery stand"""
    global robot
    try:
        logging.info("== Executing Recovery Stand ==")

        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Set gait to recovery stand
        status = controller.set_gait(magicbot.GaitMode.GAIT_RECOVERY_STAND)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set robot gait, code: %s, message: %s",
                status.code,
                status.message,
            )
            return False

        logging.info("Robot gait set to recovery stand")
        return True
    
```

(下页继续)

(续上页)

```

except Exception as e:
    logging.error("Exception occurred while executing recovery stand: %s", e)
    return False


def balance_stand():
    """Balance stand"""
    global robot
    try:
        logging.info("== Executing Balance Stand ===")

        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Set gait to balance stand
        status = controller.set_gait(magicbot.GaitMode.GAIT_BALANCE_STAND, 10000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set robot gait, code: %s, message: %s",
                status.code,
                status.message,
            )
        return False

        logging.info("Robot gait set to balance stand (supports movement)")
        return True

    except Exception as e:
        logging.error("Exception occurred while executing balance stand: %s", e)
        return False


def pure_damper():
    """Pure damper"""
    global robot
    try:
        logging.info("== Executing Pure Damper ===")

        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Set gait to balance stand

```

(下页继续)

(续上页)

```

status = controller.set_gait(magicbot.GaitMode.GAIT_PURE_DAMPER, 10000)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to set robot gait, code: %s, message: %s",
        status.code,
        status.message,
    )
    return False

logging.info("Robot gait set to pure damper")
return True

except Exception as e:
    logging.error("Exception occurred while executing pure damper: %s", e)
    return False


def get_action(cmd):
    if cmd == "201":
        return magicbot.TrickAction.ACTION_CELEBRATE
    elif cmd == "217":
        return magicbot.TrickAction.ACTION_SHAKE_HAND_REACHOUT
    elif cmd == "218":
        return magicbot.TrickAction.ACTION_SHAKE_HAND_WITHDRAW
    elif cmd == "219":
        return magicbot.TrickAction.ACTION_NOD_HEAD
    elif cmd == "220":
        return magicbot.TrickAction.ACTION_SHAKE_HEAD
    elif cmd == "221":
        return magicbot.TrickAction.ACTION_CIRCLE_HEAD
    elif cmd == "301":
        return magicbot.TrickAction.ACTION_GREETING
    elif cmd == "302":
        return magicbot.TrickAction.ACTION_POINT_GROUND
    elif cmd == "303":
        return magicbot.TrickAction.ACTION_POINT_GROUND_WITH_DRAW
    elif cmd == "304":
        return magicbot.TrickAction.ACTION_SPREAD_HAND
    elif cmd == "305":
        return magicbot.TrickAction.ACTION_SPREAD_HAND_WITH_DRAW
    elif cmd == "306":
        return magicbot.TrickAction.ACTION_TRUN_AWAY_LEFT_INTRODUCE
    elif cmd == "307":

```

(下页继续)

(续上页)

```

    return magicbot.TrickAction.ACTION_TRUN_BACK_LEFT_INTRODUCE
else:
    return magicbot.TrickAction.ACTION_NONE


def execute_trick_action(cmd):
    """Execute trick action"""
    global robot
    try:
        logging.info("== Executing Trick - %s Action ==", cmd)

        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Execute celebrate trick
        status = controller.execute_trick(get_action(cmd), 10000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to execute robot trick, code: %s, message: %s",
                status.code,
                status.message,
            )
        return False

        logging.info("Robot trick executed successfully")
        return True

    except Exception as e:
        logging.error("Exception occurred while executing trick: %s", e)
        return False


def joystick_command(left_x_axis, left_y_axis, right_x_axis, right_y_axis):
    """Send joystick control command"""
    global robot
    try:
        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Create joystick command
        joy_command = magicbot.JoystickCommand()
        joy_command.left_x_axis = left_x_axis
        joy_command.left_y_axis = left_y_axis

```

(下页继续)

(续上页)

```

joy_command.right_x_axis = right_x_axis
joy_command.right_y_axis = right_y_axis

# Send joystick command
status = controller.send_joystick_command(joy_command)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to send joystick command, code: %s, message: %s",
        status.code,
        status.message,
    )
    time.sleep(0.05) # Wait 50ms
    return False

# Wait 50ms
time.sleep(0.05)
return True

except Exception as e:
    logging.error("Exception occurred while sending joystick command: %s", e)
    return False

def move_forward():
    """Move forward"""
    logging.info("==== Moving Forward ====")
    return joystick_command(0.0, 1.0, 0.0, 0.0)

def move_backward():
    """Move backward"""
    logging.info("==== Moving Backward ====")
    return joystick_command(0.0, -1.0, 0.0, 0.0)

def move_left():
    """Move left"""
    logging.info("==== Moving Left ====")
    return joystick_command(-1.0, 0.0, 0.0, 0.0)

def move_right():
    """Move right"""

```

(下页继续)

(续上页)

```

logging.info("== Moving Right ==")
return joystick_command(1.0, 0.0, 0.0, 0.0)

def turn_left():
    """Turn left"""
    logging.info("== Turning Left ==")
    return joystick_command(0.0, 0.0, -1.0, 0.0)

def turn_right():
    """Turn right"""
    logging.info("== Turning Right ==")
    return joystick_command(0.0, 0.0, 1.0, 0.0)

def stop_move():
    """Stop Move"""
    logging.info("== Stop Move ==")
    return joystick_command(0.0, 0.0, 0.0, 0.0)

def head_move(shake_angle, nod_angle):
    """Move head to specified shake and nod angles"""
    global robot
    try:
        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Move head
        status = controller.head_move(shake_angle, nod_angle, 5000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to move head, code: %s, message: %s",
                status.code,
                status.message,
            )
        return False
    except Exception as e:
        logging.error("Error occurred during head move: %s", str(e))

    logging.info(
        "Head moved successfully: shake=%.2f rad, nod=%.2f rad",
        shake_angle,
        nod_angle,
    )

```

(下页继续)

(续上页)

```

        )
    return True

except Exception as e:
    logging.error("Exception occurred while moving head: %s", e)
    return False

def head_look_up():
    """Head look up"""
    logging.info("==== Head Looking Up ====")
    # Nod angle: up is positive
    return head_move(0.0, 0.349)

def head_look_down():
    """Head look down"""
    logging.info("==== Head Looking Down ====")
    # Nod angle: down is negative
    return head_move(0.0, -0.349)

def head_turn_left():
    """Head turn left"""
    logging.info("==== Head Turning Left ====")
    # Shake angle: left is negative
    return head_move(-0.523, 0.0)

def head_turn_right():
    """Head turn right"""
    logging.info("==== Head Turning Right ====")
    # Shake angle: right is positive
    return head_move(0.523, 0.0)

def head_reset():
    """Head reset to center position"""
    logging.info("==== Head Reset ====")
    return head_move(0.0, 0.0)

# Get single character input (no echo)

```

(下页继续)

(续上页)

```
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
        logging.info(f"Received character: {ch}")

        sys.stdout.write("\r")
        sys.stdout.flush()

    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def main():
    """Main function"""
    global robot, running

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
            return -1

        # Connect to robot
        status = robot.connect()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to connect to robot, code: %s, message: %s",
                status.code,
                status.message,
            )
    
```

(下页继续)

(续上页)

```

    robot.shutdown()
    return -1

logging.info("Successfully connected to robot")

# get gait status
status, gait_mode = robot.get_high_level_motion_controller().get_gait()
logging.info("Gait mode: %s", gait_mode)

# Switch motion control controller to high-level controller
status = robot.set_motion_control_level(magicbot.ControllerLevel.HighLevel)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to switch robot motion control level, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Switched to high-level motion controller")

# Initialize high-level motion controller
controller = robot.get_high_level_motion_controller()
if not controller.initialize():
    logging.error("Failed to initialize high-level motion controller")
    robot.disconnect()
    robot.shutdown()
    return -1

logging.info("Successfully initialized high-level motion controller")

print_help()
logging.info("Press any key to continue (ESC to exit)...")


# Main loop
while running:
    try:
        key = getch()
        if key == "\x1b": # ESC key
            break

# 1. Gait and Trick Functions

```

(下页继续)

(续上页)

```

# 1.1 Pure damper mode
if key == "0":
    pure_damper()

# 1.2 Recovery stand
elif key == "1":
    recovery_stand()

# 1.3 Balance stand
elif key == "2":
    balance_stand()

# 1.4 Execute trick action
elif key == "3":
    str_input = get_user_input()
    # Split input parameters by space
    parts = str_input.strip().split()
    # Parse parameters
    cmd = parts[0] if parts else "201"
    print("cmd: ", cmd)
    execute_trick_action(cmd)

# 2. Joystick Functions

# 2.1 Move forward
elif key.upper() == "W":
    move_forward()

# 2.2 Move left
elif key.upper() == "A":
    move_left()

# 2.3 Move backward
elif key.upper() == "S":
    move_backward()

# 2.4 Move right
elif key.upper() == "D":
    move_right()

# 2.5 Turn left
elif key.upper() == "T":
    turn_left()

# 2.6 Turn right
elif key.upper() == "G":
    turn_right()

# 2.7 Stop movement
elif key.upper() == "X":
    stop_move()

# 3. Head Functions

# 3.1 Look up
elif key.upper() == "B":

```

(下页继续)

(续上页)

```

        head_look_up()

    # 3.2 Look down
    elif key.upper() == "J":
        head_look_down()

    # 3.3 Turn left
    elif key.upper() == "K":
        head_turn_left()

    # 3.4 Turn right
    elif key.upper() == "L":
        head_turn_right()

    # 3.5 Reset position
    elif key.upper() == "U":
        head_reset()

    # 4. Print help information
    elif key.upper() == "?":
        print_help()

    else:
        logging.info("Unknown key: %s", key)

        time.sleep(0.01) # Brief delay

except KeyboardInterrupt:
    break
except Exception as e:
    logging.error("Exception occurred while processing user input: %s", e)

return 0

except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close high-level motion controller
        controller = robot.get_high_level_motion_controller()
        controller.shutdown()
        logging.info("High-level motion controller closed")

        # Disconnect
        robot.disconnect()

```

(下页继续)

(续上页)

```

    logging.info("Robot connection disconnected")

    # Shutdown robot
    robot.shutdown()
    logging.info("Robot shutdown")

except Exception as e:
    logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())

```

21.3 运行说明

21.3.1 环境准备

```

export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

```

21.3.2 运行示例

```

# C++
./high_level_motion_example
# Python
python3 high_level_motion_example.py

```

21.3.3 控制说明

- 0: 纯阻尼模式
- 1: 恢复站立
- 2: 平衡站立
- 3: 庆祝动作
- W/A/S/D/X: 前进/左移/后退/右移/停止移动
- T/G: 左转/右转
- B/J/K/L/U: 头部向上/向下/向左/向右/复位
- ?: 打印帮助信息

注意：人形机器人要恢复站立状态下落地，落地之后才能切平衡站立，之后才能做特技等其他动作

21.3.4 停止程序

- 按 ESC 可以安全停止程序
- 程序会自动清理所有资源

CHAPTER 22

底层运动控制示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行初始化、连接机器人、底层运动控制（手、臂、腿、腰、头）、Imu 传感器数据读取等基本操作。

22.1 C++

示例文件：low_level_motion_example.cpp

```
#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <unistd.h>
#include <atomic>
#include <csignal>
#include <iostream>
#include <chrono>
#include <thread>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;

std::atomic<bool> running(true);
```

(下页继续)

(续上页)

```

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
    running = false;

    robot.Shutdown();
    // Exit process
    exit(signum);
}

int main() {
    // Bind SIGINT (Ctrl+C)
    signal(SIGINT, signalHandler);

    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;

    std::string local_ip = "192.168.54.111";
    // Configure local IP address for direct network connection and initialize SDK
    if (!robot.Initialize(local_ip)) {
        std::cerr << "robot sdk initialize failed." << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Connect to robot
    auto status = robot.Connect();
    if (status.code != ErrorCode::OK) {
        std::cerr << "connect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Switch motion control controller to low-level controller, default is high-level controller
    status = robot.SetMotionControlLevel(ControllerLevel::LowLevel);
    if (status.code != ErrorCode::OK) {
        std::cerr << "switch robot motion control level failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }
}

```

(下页继续)

(续上页)

```

// Get low-level controller
auto& controller = robot.GetLowLevelMotionController();

// Subscribe to IMU data
controller.SubscribeBodyImu([](const std::shared_ptr<Imu> msg) {
    static int32_t count = 0;
    if (count++ % 1000 == 1) {
        std::cout << "++++++ receive imu data." << std::endl;
        std::cout << "timestamp: " << msg->timestamp << std::endl;
        std::cout << "temperature: " << msg->temperature << std::endl;
        std::cout << "orientation: " << msg->orientation[0] << ", " << msg->orientation[1] << ",
        << msg->orientation[2] << ", " << msg->orientation[3] << std::endl;
        std::cout << "angular_velocity: " << msg->angular_velocity[0] << ", " << msg->angular_
        << velocity[1] << ", " << msg->angular_velocity[2] << std::endl;
        std::cout << "linear_acceleration: " << msg->linear_acceleration[0] << ", " << msg->
        << linear_acceleration[1] << ", " << msg->linear_acceleration[2] << std::endl;
    }
    // TODO: handle imu data
});

// Subscribe to arm data
controller.SubscribeArmState([](const std::shared_ptr<JointState> msg) {
    static int32_t count = 0;
    if (count++ % 1000 == 1) {
        std::cout << "++++++ receive arm joint data." << std::endl;
        std::cout << "timestamp: " << msg->timestamp << std::endl;
        std::cout << "pos: " << msg->joints[0].posH << ", " << msg->joints[0].posL << std::endl;
        std::cout << "vel: " << msg->joints[0].vel << std::endl;
        std::cout << "toq: " << msg->joints[0].toq << std::endl;
        std::cout << "current: " << msg->joints[0].current << std::endl;
        std::cout << "error_code: " << msg->joints[0].err_code << std::endl;
    }
    // TODO: handle arm joint data
});

// Using arm joint control as an example:
// Subsequent joint control commands, joint operation mode is 1, indicating the joint is in_
// position control mode
auto now = std::chrono::steady_clock::now();
while (running.load()) {
    // Left arm joints, refer to documentation:
    // Left arm or right arm joints 1-5 operation_mode needs to switch from mode: 200 to mode:_
```

(下页继续)

(续上页)

```

    ↵4 (series PID mode) for command execution;

    JointCommand arm_command;
    arm_command.joints.resize(kArmJointNum);
    for (int ii = 0; ii < kArmJointNum; ii++) {
        // Set joint to ready state
        arm_command.joints[ii].operation_mode = 200;
        // TODO: Set target position, velocity, torque and gains
        arm_command.joints[ii].pos = 0.0;
        arm_command.joints[ii].vel = 0.0;
        arm_command.joints[ii].toq = 0.0;
        arm_command.joints[ii].kp = 0.0;
        arm_command.joints[ii].kd = 0.0;
    }
    // Publish control command
    controller.PublishArmCommand(arm_command);

    // Send control command at 500Hz frequency (2ms)
    now += std::chrono::milliseconds(2);
    std::this_thread::sleep_until(now);
}

// Disconnect from robot
status = robot.Disconnect();
if (status.code != ErrorCode::OK) {
    std::cerr << "disconnect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}

robot.Shutdown();

return 0;
}

```

22.2 Python

示例文件: low_level_motion_example.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

```

(下页继续)

(续上页)

```

import sys
import time
import signal
import logging
from typing import Optional

import magicbot_gen1_python as magicbot

# Configure logging format and level
logging.basicConfig(
    level=logging.INFO, # Minimum log level
    format"%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True

body_imu_counter = 0
arm_state_counter = 0
leg_state_counter = 0
head_state_counter = 0
waist_state_counter = 0


def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global robot, running
    running = False
    logging.info("Received interrupt signal (%s), exiting...", signum)
    if robot:
        robot.shutdown()
        logging.info("Robot shutdown")
    exit(-1)


def body_imu_callback(imu_data):
    """Body IMU data callback function"""
    global body_imu_counter
    if body_imu_counter % 1000 == 0:
        logging.info("++++++ Received body IMU data")
        # Print IMU data

```

(下页继续)

(续上页)

```

logging.info("Received body IMU data, timestamp: %d", imu_data.timestamp)
logging.info(
    "Received body IMU data, orientation: [%f, %f, %f, %f]",
    imu_data.orientation[0],
    imu_data.orientation[1],
    imu_data.orientation[2],
    imu_data.orientation[3],
)
logging.info(
    "Received body IMU data, angular_velocity: [%f, %f, %f]",
    imu_data.angular_velocity[0],
    imu_data.angular_velocity[1],
    imu_data.angular_velocity[2],
)
logging.info(
    "Received body IMU data, linear_acceleration: [%f, %f, %f]",
    imu_data.linear_acceleration[0],
    imu_data.linear_acceleration[1],
    imu_data.linear_acceleration[2],
)
logging.info("Received body IMU data, temperature: %s", imu_data.temperature)
body_imu_counter += 1

```

```

def arm_state_callback(joint_state):
    """Arm joint state callback function"""
    global arm_state_counter
    if arm_state_counter % 1000 == 0:
        logging.info("++++++ Received arm joint state data")
        # Print joint state data
        logging.info(
            "Received arm joint state data, status_word: %d",
            joint_state.joints[0].status_word,
        )
        logging.info(
            "Received arm joint state data, posH: %s", joint_state.joints[0].posH
        )
        logging.info(
            "Received arm joint state data, posL: %s", joint_state.joints[0].posL
        )
        logging.info(
            "Received arm joint state data, vel: %s", joint_state.joints[0].vel
        )

```

(下页继续)

(续上页)

```
logging.info(
    "Received arm joint state data, toq: %s", joint_state.joints[0].toq
)
logging.info(
    "Received arm joint state data, current: %s", joint_state.joints[0].current
)
logging.info(
    "Received arm joint state data, error_code: %s",
    joint_state.joints[0].err_code,
)
arm_state_counter += 1

def leg_state_callback(joint_state):
    """Leg joint state callback function"""
    global leg_state_counter
    if leg_state_counter % 1000 == 0:
        logging.info("++++++ Received leg joint state data")
        # Print joint state data
        logging.info(
            "Received leg joint state data, status_word: %d",
            joint_state.joints[0].status_word,
        )
        logging.info(
            "Received leg joint state data, posH: %s", joint_state.joints[0].posH
        )
        logging.info(
            "Received leg joint state data, posL: %s", joint_state.joints[0].posL
        )
        logging.info(
            "Received leg joint state data, vel: %s", joint_state.joints[0].vel
        )
        logging.info(
            "Received leg joint state data, toq: %s", joint_state.joints[0].toq
        )
        logging.info(
            "Received leg joint state data, current: %s", joint_state.joints[0].current
        )
        logging.info(
            "Received leg joint state data, error_code: %s",
            joint_state.joints[0].err_code,
        )
    leg_state_counter += 1
```

(下页继续)

(续上页)

```

def head_state_callback(joint_state):
    """Head joint state callback function"""
    global head_state_counter
    if head_state_counter % 1000 == 0:
        logging.info("++++++ Received head joint state data")
        # Print joint state data
        logging.info(
            "Received head joint state data, status_word: %d",
            joint_state.joints[0].status_word,
        )
        logging.info(
            "Received head joint state data, posH: %s", joint_state.joints[0].posH
        )
        logging.info(
            "Received head joint state data, posL: %s", joint_state.joints[0].posL
        )
        logging.info(
            "Received head joint state data, vel: %s", joint_state.joints[0].vel
        )
        logging.info(
            "Received head joint state data, toq: %s", joint_state.joints[0].toq
        )
        logging.info(
            "Received head joint state data, current: %s", joint_state.joints[0].current
        )
        logging.info(
            "Received head joint state data, error_code: %s",
            joint_state.joints[0].err_code,
        )
    head_state_counter += 1

def waist_state_callback(joint_state):
    """Waist joint state callback function"""
    global waist_state_counter
    if waist_state_counter % 1000 == 0:
        logging.info("++++++ Received waist joint state data")
        # Print joint state data
        logging.info(
            "Received waist joint state data, status_word: %d",
            joint_state.joints[0].status_word,
        )

```

(下页继续)

(续上页)

```

)
logging.info(
    "Received waist joint state data, posH: %s", joint_state.joints[0].posH
)
logging.info(
    "Received waist joint state data, posL: %s", joint_state.joints[0].posL
)
logging.info(
    "Received waist joint state data, vel: %s", joint_state.joints[0].vel
)
logging.info(
    "Received waist joint state data, toq: %s", joint_state.joints[0].toq
)
logging.info(
    "Received waist joint state data, current: %s",
    joint_state.joints[0].current,
)
logging.info(
    "Received waist joint state data, error_code: %s",
    joint_state.joints[0].err_code,
)
waist_state_counter += 1

def main():
    """Main function"""
    global robot

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
            return -1

```

(下页继续)

(续上页)

```
# Connect to robot
status = robot.connect()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to connect to robot, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Successfully connected to robot")

# Switch motion control controller to low-level controller, default is high-level
controller
status = robot.set_motion_control_level(magicbot.ControllerLevel.LowLevel)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to switch robot motion control level, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Switched to low-level motion controller")

# Get low-level motion controller
controller = robot.get_low_level_motion_controller()

# Subscribe to body IMU data
controller.subscribe_body_imu(body_imu_callback)
logging.info("Subscribed to body IMU data")

# Subscribe to arm joint state
controller.subscribe_arm_state(arm_state_callback)
logging.info("Subscribed to arm joint state")

# Subscribe to leg joint state
controller.subscribe_leg_state(leg_state_callback)
logging.info("Subscribed to leg joint state")
```

(下页继续)

(续上页)

```
# Subscribe to head joint state
controller.subscribe_head_state(head_state_callback)
logging.info("Subscribed to head joint state")

# Subscribe to waist joint state
controller.subscribe_waist_state(waist_state_callback)
logging.info("Subscribed to waist joint state")

# Main loop
interval = 0.002 # 2ms
next_t = time.perf_counter() + interval

global running
while running:
    # Create arm joint control command
    arm_command = magicbot.JointCommand()
    leg_command = magicbot.JointCommand()
    waist_command = magicbot.JointCommand()
    head_command = magicbot.JointCommand()

    # Set all joints to preparation state (operation mode 200)
    for i in range(magicbot.ARM_JOINT_NUM):
        joint = magicbot.SingleJointCommand()
        joint.operation_mode = 200 # Preparation state
        joint.pos = 0.0
        joint.vel = 0.0
        joint.toq = 0.0
        joint.kp = 0.0
        joint.kd = 0.0
        arm_command.joints.append(joint)

    for i in range(magicbot.LEG_JOINT_NUM):
        joint = magicbot.SingleJointCommand()
        joint.operation_mode = 200 # Preparation state
        joint.pos = 0.0
        joint.vel = 0.0
        joint.toq = 0.0
        joint.kp = 0.0
        joint.kd = 0.0
        leg_command.joints.append(joint)

    for i in range(magicbot.HEAD_JOINT_NUM):
        joint = magicbot.SingleJointCommand()
```

(下页继续)

(续上页)

```
joint.operation_mode = 200 # Preparation state
joint.pos = 0.0
joint.vel = 0.0
joint.toq = 0.0
joint.kp = 0.0
joint.kd = 0.0
head_command.joints.append(joint)

for i in range(magicbot.WAIST_JOINT_NUM):
    joint = magicbot.SingleJointCommand()
    joint.operation_mode = 200 # Preparation state
    joint.pos = 0.0
    joint.vel = 0.0
    joint.toq = 0.0
    joint.kp = 0.0
    joint.kd = 0.0
    waist_command.joints.append(joint)

# Publish arm joint control command
controller.publish_arm_command(arm_command)

# Publish leg joint control command
controller.publish_leg_command(leg_command)

# Publish waist joint control command
controller.publish_waist_command(waist_command)

# Publish head joint control command
controller.publish_head_command(head_command)

next_t += interval
sleep_time = next_t - time.perf_counter()
if sleep_time > 0:
    time.sleep(sleep_time)

except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
```

(下页继续)

(续上页)

```

# Close low-level motion controller
controller = robot.get_low_level_motion_controller()
controller.shutdown()
logging.info("Low-level motion controller closed")

# Disconnect
robot.disconnect()
logging.info("Robot connection disconnected")

# Shutdown robot
robot.shutdown()
logging.info("Robot shutdown")

except Exception as e:
    logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())

```

22.3 运行说明

22.3.1 环境准备

```

export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

```

22.3.2 运行示例

```

# C++
./low_level_motion_example
# Python
python3 low_level_motion_example.py

```

22.3.3 停止程序

- 按 Ctrl+C 可以安全停止程序
- 程序会自动清理所有资源

CHAPTER 23

传感器控制示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行初始化、连接机器人、机器人传感器控制（雷达、RGBD 相机、三目相机）等基本操作。

23.1 C++

示例文件: sensor_example.cpp

```
#include "magic_robot.h"
#include "magic_sdk_version.h"
#include "magic_type.h"

#include <termios.h>
#include <unistd.h>
#include <csignal>

#include <atomic>
#include <iostream>
#include <map>
#include <memory>
#include <string>

using namespace magic::gen1;
```

(下页继续)

(续上页)

```

// Global counters for periodic logging
std::atomic<int> lidar_imu_counter{0};
std::atomic<int> lidar_pointcloud_counter{0};
std::atomic<int> head_rgbd_color_counter{0};
std::atomic<int> head_rgbd_depth_counter{0};
std::atomic<int> waist_rgbd_color_counter{0};
std::atomic<int> waist_rgbd_depth_counter{0};
std::atomic<int> trinocular_image_counter{0};

class SensorManager {
public:
    explicit SensorManager(SensorController& controller)
        : sensor_controller_(controller) {
        sensors_state_[ "lidar" ] = false;
        sensors_state_[ "head_rgbd_camera" ] = false;
        sensors_state_[ "waist_rgbd_camera" ] = false;
        sensors_state_[ "trinocular_camera" ] = false;

        subscriptions_[ "lidar_imu" ] = false;
        subscriptions_[ "lidar_point_cloud" ] = false;
        subscriptions_[ "head_rgbd_color_image" ] = false;
        subscriptions_[ "head_rgbd_depth_image" ] = false;
        subscriptions_[ "waist_rgbd_color_image" ] = false;
        subscriptions_[ "waist_rgbd_depth_image" ] = false;
        subscriptions_[ "trinocular_image" ] = false;
    }

// === LiDAR Control ===
    bool OpenLidar() {
        if (sensors_state_[ "lidar" ]) {
            std::cout << "[WARNING] LiDAR already opened" << std::endl;
            return true;
        }

        auto status = sensor_controller_.OpenLidar();
        if (status.code != ErrorCode::OK) {
            std::cerr << "[ERROR] Failed to open LiDAR: " << status.message << std::endl;
            return false;
        }

        sensors_state_[ "lidar" ] = true;
        std::cout << "[INFO] ✓ LiDAR opened successfully" << std::endl;
        return true;
    }
}

```

(下页继续)

(续上页)

```

}

bool CloseLidar() {
    if (!sensors_state_["lidar"]) {
        std::cout << "[WARNING] LiDAR already closed" << std::endl;
        return true;
    }

    // Unsubscribe if subscribed
    if (subscriptions_["lidar_imu"]) ToggleLidarImuSubscription();
    if (subscriptions_["lidar_point_cloud"]) ToggleLidarPointCloudSubscription();

    auto status = sensor_controller_.CloseLidar();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Failed to close LiDAR: " << status.message << std::endl;
        return false;
    }

    sensors_state_["lidar"] = false;
    std::cout << "[INFO] ✓ LiDAR closed" << std::endl;
    return true;
}

// === Head RGBD Camera Control ===
bool OpenHeadRgbdCamera() {
    if (sensors_state_["head_rgbd_camera"]) {
        std::cout << "[WARNING] Head RGBD camera already opened" << std::endl;
        return true;
    }

    auto status = sensor_controller_.OpenHeadRgbdCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Failed to open head RGBD camera: " << status.message << std::endl;
        return false;
    }

    sensors_state_["head_rgbd_camera"] = true;
    std::cout << "[INFO] ✓ Head RGBD camera opened" << std::endl;
    return true;
}

bool CloseHeadRgbdCamera() {
    if (!sensors_state_["head_rgbd_camera"]) {
}

```

(下页继续)

(续上页)

```

    std::cout << "[WARNING] Head RGBD camera already closed" << std::endl;
    return true;
}

if (subscriptions_["head_rgbd_color_image"]) ToggleHeadRgbdColorImageSubscription();
if (subscriptions_["head_rgbd_depth_image"]) ToggleHeadRgbdDepthImageSubscription();

auto status = sensor_controller_.CloseHeadRgbdCamera();
if (status.code != ErrorCode::OK) {
    std::cerr << "[ERROR] Failed to close head RGBD camera: " << status.message << std::endl;
    return false;
}

sensors_state_["head_rgbd_camera"] = false;
std::cout << "[INFO] ✓ Head RGBD camera closed" << std::endl;
return true;
}

// === Waist RGBD Camera Control ===

bool OpenWaistRgbdCamera() {
    if (sensors_state_["waist_rgbd_camera"]) {
        std::cout << "[WARNING] Waist RGBD camera already opened" << std::endl;
        return true;
    }

    auto status = sensor_controller_.OpenWaistRgbdCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Failed to open waist RGBD camera: " << status.message << std::endl;
        return false;
    }

    sensors_state_["waist_rgbd_camera"] = true;
    std::cout << "[INFO] ✓ Waist RGBD camera opened" << std::endl;
    return true;
}

bool CloseWaistRgbdCamera() {
    if (!sensors_state_["waist_rgbd_camera"]) {
        std::cout << "[WARNING] Waist RGBD camera already closed" << std::endl;
        return true;
    }

    if (subscriptions_["waist_rgbd_color_image"]) ToggleWaistRgbdColorImageSubscription();
}

```

(下页继续)

(续上页)

```

if (subscriptions_["waist_rgbd_depth_image"]) ToggleWaistRgbdDepthImageSubscription();

auto status = sensor_controller_.CloseWaistRgbdCamera();
if (status.code != ErrorCode::OK) {
    std::cerr << "[ERROR] Failed to close waist RGBD camera: " << status.message << std::endl;
    return false;
}

sensors_state_["waist_rgbd_camera"] = false;
std::cout << "[INFO] ✓ Waist RGBD camera closed" << std::endl;
return true;
}

// === Trinocular Camera Control ===
bool OpenTrinocularCamera() {
    if (sensors_state_["trinocular_camera"]) {
        std::cout << "[WARNING] Trinocular camera already opened" << std::endl;
        return true;
    }

    auto status = sensor_controller_.OpenTrinocularCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Failed to open trinocular camera: " << status.message << std::endl;
        return false;
    }

    sensors_state_["trinocular_camera"] = true;
    std::cout << "[INFO] ✓ Trinocular camera opened" << std::endl;
    return true;
}

bool CloseTrinocularCamera() {
    if (!sensors_state_["trinocular_camera"]) {
        std::cout << "[WARNING] Trinocular camera already closed" << std::endl;
        return true;
    }

    if (subscriptions_["trinocular_image"]) ToggleTrinocularImageSubscription();

    auto status = sensor_controller_.CloseTrinocularCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Failed to close trinocular camera: " << status.message << std::endl;
        return false;
    }
}

```

(下页继续)

(续上页)

```

    }

    sensors_state_[ "trinocular_camera" ] = false;
    std::cout << "[INFO] ✓ Trinocular camera closed" << std::endl;
    return true;
}

// === LiDAR Subscriptions ===

void ToggleLidarImuSubscription() {
    if (subscriptions_[ "lidar_imu" ]) {
        sensor_controller_.UnsubscribeLidarImu();
        subscriptions_[ "lidar_imu" ] = false;
        std::cout << "[INFO] ✓ LiDAR IMU unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeLidarImu([](const std::shared_ptr<Imu> msg) {
            lidar_imu_counter++;
            if (lidar_imu_counter % 100 == 0) {
                std::cout << "[DATA] LiDAR IMU received (count: " << lidar_imu_counter << ")" <<
                    std::endl;
            }
        });
        subscriptions_[ "lidar_imu" ] = true;
        std::cout << "[INFO] ✓ LiDAR IMU subscribed" << std::endl;
    }
}

void ToggleLidarPointCloudSubscription() {
    if (subscriptions_[ "lidar_point_cloud" ]) {
        sensor_controller_.UnsubscribeLidarPointCloud();
        subscriptions_[ "lidar_point_cloud" ] = false;
        std::cout << "[INFO] ✓ LiDAR point cloud unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeLidarPointCloud([](const std::shared_ptr<PointCloud2> msg) {
            lidar_pointcloud_counter++;
            if (lidar_pointcloud_counter % 10 == 0) {
                std::cout << "[DATA] LiDAR Point Cloud: " << msg->data.size()
                    << " bytes (count: " << lidar_pointcloud_counter << ")" << std::endl;
            }
        });
        subscriptions_[ "lidar_point_cloud" ] = true;
        std::cout << "[INFO] ✓ LiDAR point cloud subscribed" << std::endl;
    }
}

```

(下页继续)

(续上页)

```

// === Head RGBD Subscriptions ===

void ToggleHeadRgbdColorImageSubscription() {
    if (subscriptions_["head_rgbd_color_image"]) {
        sensor_controller_.UnsubscribeHeadRgbdColorImage();
        subscriptions_["head_rgbd_color_image"] = false;
        std::cout << "[INFO] ✓ Head RGBD color image unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeHeadRgbdColorImage([](const std::shared_ptr<Image> msg) {
            head_rgbd_color_counter++;
            if (head_rgbd_color_counter % 15 == 0) {
                std::cout << "[DATA] Head RGBD Color Image: " << msg->data.size()
                    << " bytes, " << msg->width << "x" << msg->height
                    << " (count: " << head_rgbd_color_counter << ")" << std::endl;
            }
        });
        subscriptions_["head_rgbd_color_image"] = true;
        std::cout << "[INFO] ✓ Head RGBD color image subscribed" << std::endl;
    }
}

void ToggleHeadRgbdDepthImageSubscription() {
    if (subscriptions_["head_rgbd_depth_image"]) {
        sensor_controller_.UnsubscribeHeadRgbdDepthImage();
        subscriptions_["head_rgbd_depth_image"] = false;
        std::cout << "[INFO] ✓ Head RGBD depth image unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeHeadRgbdDepthImage([](const std::shared_ptr<Image> msg) {
            head_rgbd_depth_counter++;
            if (head_rgbd_depth_counter % 15 == 0) {
                std::cout << "[DATA] Head RGBD Depth Image: " << msg->data.size()
                    << " bytes, " << msg->width << "x" << msg->height
                    << " (count: " << head_rgbd_depth_counter << ")" << std::endl;
            }
        });
        subscriptions_["head_rgbd_depth_image"] = true;
        std::cout << "[INFO] ✓ Head RGBD depth image subscribed" << std::endl;
    }
}

// === Waist RGBD Subscriptions ===

void ToggleWaistRgbdColorImageSubscription() {
    if (subscriptions_["waist_rgbd_color_image"]) {

```

(下页继续)

(续上页)

```

sensor_controller_.UnsubscribeWaistRgbColorImage();
subscriptions_["waist_rgbd_color_image"] = false;
std::cout << "[INFO] ✓ Waist RGBD color image unsubscribed" << std::endl;
} else {
    sensor_controller_.SubscribeWaistRgbColorImage([](const std::shared_ptr<Image> msg) {
        waist_rgbd_color_counter++;
        if (waist_rgbd_color_counter % 15 == 0) {
            std::cout << "[DATA] Waist RGBD Color Image: " << msg->data.size()
                << " bytes, " << msg->width << "x" << msg->height
                << " (count: " << waist_rgbd_color_counter << ")" << std::endl;
        }
    });
    subscriptions_["waist_rgbd_color_image"] = true;
    std::cout << "[INFO] ✓ Waist RGBD color image subscribed" << std::endl;
}
}

void ToggleWaistRgbDepthImageSubscription() {
if (subscriptions_["waist_rgbd_depth_image"]) {
    sensor_controller_.UnsubscribeWaistRgbDepthImage();
    subscriptions_["waist_rgbd_depth_image"] = false;
    std::cout << "[INFO] ✓ Waist RGBD depth image unsubscribed" << std::endl;
} else {
    sensor_controller_.SubscribeWaistRgbDepthImage([](const std::shared_ptr<Image> msg) {
        waist_rgbd_depth_counter++;
        if (waist_rgbd_depth_counter % 15 == 0) {
            std::cout << "[DATA] Waist RGBD Depth Image: " << msg->data.size()
                << " bytes, " << msg->width << "x" << msg->height
                << " (count: " << waist_rgbd_depth_counter << ")" << std::endl;
        }
    });
    subscriptions_["waist_rgbd_depth_image"] = true;
    std::cout << "[INFO] ✓ Waist RGBD depth image subscribed" << std::endl;
}
}

// === Trinocular Camera Subscriptions ===
void ToggleTrinocularImageSubscription() {
if (subscriptions_["trinocular_image"]) {
    sensor_controller_.UnsubscribeTrinocularImage();
    subscriptions_["trinocular_image"] = false;
    std::cout << "[INFO] ✓ Trinocular camera image unsubscribed" << std::endl;
} else {
}
}

```

(下页继续)

(续上页)

```

    sensor_controller_.SubscribeTrinocularImage([] (const std::shared_ptr
    <TrinocularCameraFrame> msg) {
        trinocular_image_counter++;
        if (trinocular_image_counter % 15 == 0) {
            std::cout << "[DATA] Trinocular Camera Frame received (count: "
            << trinocular_image_counter << ")" << std::endl;
        }
    });
    subscriptions_["trinocular_image"] = true;
    std::cout << "[INFO] ✓ Trinocular camera image subscribed" << std::endl;
}

void ShowStatus() {
    std::cout << "\n"
        << std::string(80, '=') << std::endl;
    std::cout << "MAGICBOT GEN1 SENSOR STATUS" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
    std::cout << "LiDAR: " << (sensors_state_["lidar"] ? "OPEN" :
    "CLOSED") << std::endl;
    std::cout << "Head RGBD Camera: " << (sensors_state_["head_rgbd_camera"] ?
    "OPEN" : "CLOSED") << std::endl;
    std::cout << "Waist RGBD Camera: " << (sensors_state_["waist_rgbd_camera"] ?
    "OPEN" : "CLOSED") << std::endl;
    std::cout << "Trinocular Camera: " << (sensors_state_["trinocular_camera"] ?
    "OPEN" : "CLOSED") << std::endl;
    std::cout << "\nLiDAR SUBSCRIPTIONS:" << std::endl;
    std::cout << " LiDAR IMU: " << (subscriptions_["lidar_imu"] ? "✓" :
    "SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " LiDAR Point Cloud: " << (subscriptions_["lidar_point_cloud"] ? "✓" :
    "SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nHEAD RGBD SUBSCRIPTIONS:" << std::endl;
    std::cout << " Color Image: " << (subscriptions_["head_rgbd_color_image"] ?
    "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " Depth Image: " << (subscriptions_["head_rgbd_depth_image"] ?
    "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nWAIST RGBD SUBSCRIPTIONS:" << std::endl;
    std::cout << " Color Image: " << (subscriptions_["waist_rgbd_color_image"] ?
    "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " Depth Image: " << (subscriptions_["waist_rgbd_depth_image"] ?
    "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nTRINOCULAR CAMERA SUBSCRIPTIONS:" << std::endl;
    std::cout << " Trinocular Image: " << (subscriptions_["trinocular_image"] ? "✓" :
    
```

(下页继续)

(续上页)

```

    ↵SUBSCRIBED" : "UNSUBSCRIBED") << std::endl;
    std::cout << std::string(80, '=') << "\n"
        << std::endl;
}

void CleanupAll() {
    if (sensors_state_["lidar"]) CloseLidar();
    if (sensors_state_["head_rgbd_camera"]) CloseHeadRgbdCamera();
    if (sensors_state_["waist_rgbd_camera"]) CloseWaistRgbdCamera();
    if (sensors_state_["trinocular_camera"]) CloseTrinocularCamera();
}

private:
SensorController& sensor_controller_;
std::map<std::string, bool> sensors_state_;
std::map<std::string, bool> subscriptions_;
};

void PrintMenu() {
    std::cout << "\n"
        << std::string(80, '=') << std::endl;
    std::cout << "MAGICBOT GEN1 SENSOR CONTROL MENU" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
    std::cout << "Sensor Open/Close:" << std::endl;
    std::cout << " 1 - Open LiDAR           2 - Close LiDAR" << std::endl;
    std::cout << " 3 - Open Head RGBD Camera   4 - Close Head RGBD Camera" << std::endl;
    std::cout << " 5 - Open Waist RGBD Camera  6 - Close Waist RGBD Camera" << std::endl;
    std::cout << " 7 - Open Trinocular Camera  8 - Close Trinocular Camera" << std::endl;
    std::cout << "\nLiDAR Subscriptions:" << std::endl;
    std::cout << "  i - Toggle LiDAR IMU       p - Toggle LiDAR Point Cloud" << std::endl;
    std::cout << "\nHead RGBD Camera Subscriptions:" << std::endl;
    std::cout << "  c - Toggle Head Color Image d - Toggle Head Depth Image" << std::endl;
    std::cout << "\nWaist RGBD Camera Subscriptions:" << std::endl;
    std::cout << "  w - Toggle Waist Color Image e - Toggle Waist Depth Image" << std::endl;
    std::cout << "\nTrinocular Camera Subscriptions:" << std::endl;
    std::cout << "  t - Toggle Trinocular Image" << std::endl;
    std::cout << "\nCommands:" << std::endl;
    std::cout << "  s - Show Status           ESC - Exit program      ? - Help" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
}

magic::gen1::MagicRobot robot;

```

(下页继续)

(续上页)

```

SensorManager* g_sensor_manager = nullptr;

void signalHandler(int signum) {
    std::cout << "\n[INFO] Interrupt signal (" << signum << ") received.\n";

    if (g_sensor_manager) {
        std::cout << "[INFO] Cleaning up..." << std::endl;
        g_sensor_manager->CleanupAll();
    }

    robot.Shutdown();
    exit(signum);
}

int main() {
    // Bind SIGINT (Ctrl+C)
    signal(SIGINT, signalHandler);

    std::cout << "\n"
        << std::string(80, '=') << std::endl;
    std::cout << "MagicBot Gen1 SDK Sensor Interactive Example" << std::endl;
    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;
    std::cout << std::string(80, '=') << "\n"
        << std::endl;

    std::string local_ip = "192.168.54.111";

    // Initialize SDK
    if (!robot.Initialize(local_ip)) {
        std::cerr << "[ERROR] Robot SDK initialize failed." << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Connect to robot
    auto status = robot.Connect();
    if (status.code != ErrorCode::OK) {
        std::cerr << "[ERROR] Connect robot failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }
}

```

(下页继续)

(续上页)

```
std::cout << "[INFO] ✓ Robot connected successfully\n"
    << std::endl;

auto& controller = robot.GetSensorController();
SensorManager sensor_manager(controller);
g_sensor_manager = &sensor_manager;

PrintMenu();

try {
    std::string choice;
    while (true) {
        std::cout << "\nEnter your choice: ";
        std::getline(std::cin, choice);

        if (choice.empty()) continue;

        char cmd = choice[0];

        // ESC key command
        if (cmd == 27) {
            std::cout << "[INFO] ESC key pressed, exiting program..." << std::endl;
            break;
        }

        // Sensor open/close control
        switch (cmd) {
            case '1':
                sensor_manager.OpenLidar();
                break;
            case '2':
                sensor_manager.CloseLidar();
                break;
            case '3':
                sensor_manager.OpenHeadRgbCamera();
                break;
            case '4':
                sensor_manager.CloseHeadRgbCamera();
                break;
            case '5':
                sensor_manager.OpenWaistRgbCamera();
                break;
        }
    }
}
```

(下页继续)

(续上页)

```
case '6':
    sensor_manager.CloseWaistRgbCamera();
    break;
case '7':
    sensor_manager.OpenTrinocularCamera();
    break;
case '8':
    sensor_manager.CloseTrinocularCamera();
    break;

// LiDAR subscriptions
case 'i':
    sensor_manager.ToggleLidarImuSubscription();
    break;
case 'p':
    sensor_manager.ToggleLidarPointCloudSubscription();
    break;

// Head RGBD subscriptions
case 'c':
    sensor_manager.ToggleHeadRgbColorImageSubscription();
    break;
case 'd':
    sensor_manager.ToggleHeadRgbDepthImageSubscription();
    break;

// Waist RGBD subscriptions
case 'w':
    sensor_manager.ToggleWaistRgbColorImageSubscription();
    break;
case 'e':
    sensor_manager.ToggleWaistRgbDepthImageSubscription();
    break;

// Trinocular camera subscriptions
case 't':
    sensor_manager.ToggleTrinocularImageSubscription();
    break;

// Commands
case 's':
case 'S':
    sensor_manager.ShowStatus();
```

(下页继续)

(续上页)

```

        break;

    case '?':
        PrintMenu();
        break;

    default:
        std::cout << "[WARNING] Invalid choice: '" << choice << "'. Press '?' for help." <<
        std::endl;
        break;
    }
}

} catch (const std::exception& e) {
    std::cerr << "[ERROR] Exception occurred: " << e.what() << std::endl;
}

// Cleanup
std::cout << "[INFO] Cleaning up..." << std::endl;
sensor_manager.CleanupAll();

sleep(1); // Allow time for cleanup

// Disconnect from robot
status = robot.Disconnect();
if (status.code != ErrorCode::OK) {
    std::cerr << "[ERROR] Disconnect robot failed"
    << ", code: " << status.code
    << ", message: " << status.message << std::endl;
} else {
    std::cout << "[INFO] Robot disconnected" << std::endl;
}

robotShutdown();
std::cout << "[INFO] Robot shutdown" << std::endl;

return 0;
}

```

23.2 Python

示例文件: sensor_example.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

```

(下页继续)

(续上页)

```

import sys
import time
import logging
from typing import Optional, Dict
import magicbot_gen1_python as magicbot
from magicbot_gen1_python import ErrorCode

logging.basicConfig(
    level=logging.INFO,
    format"%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

lidar_imu_counter = 0
lidar_pointcloud_counter = 0
head_rgbd_color_counter = 0
head_rgbd_color_camera_info_counter = 0
head_rgbd_depth_image_counter = 0
head_rgbd_depth_camera_info_counter = 0
waist_rgbd_color_counter = 0
waist_rgbd_color_camera_info_counter = 0
waist_rgbd_depth_image_counter = 0
waist_rgbd_depth_camera_info_counter = 0
trinocular_image_counter = 0

class SensorManager:
    """Manages sensor subscriptions for MagicBot Gen1"""

    def __init__(self, sensor_controller):
        self.sensor_controller = sensor_controller
        self.sensors_state = {
            "lidar": False,
            "head_rgbd_camera": False,
            "waist_rgbd_camera": False,
            "trinocular_camera": False,
        }
        self.subscriptions = {
            # LiDAR subscriptions
            "lidar_imu": False,
            "lidar_point_cloud": False,
        }

```

(下页继续)

(续上页)

```

# Head RGBD subscriptions
"head_rgbd_color_image": False,
"head_rgbd_color_camera_info": False,
"head_rgbd_depth_image": False,
"head_rgbd_depth_camera_info": False,
# Waist RGBD subscriptions
"waist_rgbd_color_image": False,
"waist_rgbd_color_camera_info": False,
"waist_rgbd_depth_image": False,
"waist_rgbd_depth_camera_info": False,
# Trinocular camera subscriptions
"trinocular_image": False,
}

# === LiDAR Control ===
def open_lidar(self) -> bool:
    """Open LiDAR"""
    if self.sensors_state["lidar"]:
        logging.warning("LiDAR already opened")
        return True

    status = self.sensor_controller.open_lidar()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to open LiDAR: {status.message}")
        return False

    self.sensors_state["lidar"] = True
    logging.info("✓ LiDAR opened successfully")
    return True

def close_lidar(self) -> bool:
    """Close LiDAR"""
    if not self.sensors_state["lidar"]:
        logging.warning("LiDAR already closed")
        return True

    # Unsubscribe if subscribed
    if self.subscriptions["lidar_imu"]:
        self.toggle_lidar_imu_subscription()
    if self.subscriptions["lidar_point_cloud"]:
        self.toggle_lidar_point_cloud_subscription()

    status = self.sensor_controller.close_lidar()

```

(下页继续)

(续上页)

```

if status.code != ErrorCode.OK:
    logging.error(f"Failed to close LiDAR: {status.message}")
    return False

self.sensors_state["lidar"] = False
logging.info("✓ LiDAR closed")
return True

# === Head RGBD Camera Control ===
def open_head_rgbd_camera(self) -> bool:
    """Open head RGBD camera"""
    if self.sensors_state["head_rgbd_camera"]:
        logging.warning("Head RGBD camera already opened")
        return True

    status = self.sensor_controller.open_head_rgbd_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to open head RGBD camera: {status.message}")
        return False

    self.sensors_state["head_rgbd_camera"] = True
    logging.info("✓ Head RGBD camera opened")
    return True

def close_head_rgbd_camera(self) -> bool:
    """Close head RGBD camera"""
    if not self.sensors_state["head_rgbd_camera"]:
        logging.warning("Head RGBD camera already closed")
        return True

    # Unsubscribe all head RGBD subscriptions if subscribed
    if self.subscriptions["head_rgbd_color_image"]:
        self.toggle_head_rgbd_color_image_subscription()
    if self.subscriptions["head_rgbd_color_camera_info"]:
        self.toggle_head_rgbd_color_camera_info_subscription()
    if self.subscriptions["head_rgbd_depth_image"]:
        self.toggle_head_rgbd_depth_image_subscription()
    if self.subscriptions["head_rgbd_depth_camera_info"]:
        self.toggle_head_rgbd_depth_camera_info_subscription()

    status = self.sensor_controller.close_head_rgbd_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to close head RGBD camera: {status.message}")

```

(下页继续)

(续上页)

```
        return False

    self.sensors_state["head_rgbd_camera"] = False
    logging.info("✓ Head RGBD camera closed")
    return True

# === Waist RGBD Camera Control ===

def open_waist_rgbd_camera(self) -> bool:
    """Open waist RGBD camera"""
    if self.sensors_state["waist_rgbd_camera"]:
        logging.warning("Waist RGBD camera already opened")
        return True

    status = self.sensor_controller.open_waist_rgbd_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to open waist RGBD camera: {status.message}")
        return False

    self.sensors_state["waist_rgbd_camera"] = True
    logging.info("✓ Waist RGBD camera opened")
    return True

def close_waist_rgbd_camera(self) -> bool:
    """Close waist RGBD camera"""
    if not self.sensors_state["waist_rgbd_camera"]:
        logging.warning("Waist RGBD camera already closed")
        return True

    # Unsubscribe all waist RGBD subscriptions if subscribed
    if self.subscriptions["waist_rgbd_color_image"]:
        self.toggle_waist_rgbd_color_image_subscription()
    if self.subscriptions["waist_rgbd_color_camera_info"]:
        self.toggle_waist_rgbd_color_camera_info_subscription()
    if self.subscriptions["waist_rgbd_depth_image"]:
        self.toggle_waist_rgbd_depth_image_subscription()
    if self.subscriptions["waist_rgbd_depth_camera_info"]:
        self.toggle_waist_rgbd_depth_camera_info_subscription()

    status = self.sensor_controller.close_waist_rgbd_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to close waist RGBD camera: {status.message}")
        return False
```

(下页继续)

(续上页)

```

        self.sensors_state["waist_rgbd_camera"] = False
        logging.info("✓ Waist RGBD camera closed")
        return True

# === Trinocular Camera Control ===
def open_trinocular_camera(self) -> bool:
    """Open trinocular camera"""
    if self.sensors_state["trinocular_camera"]:
        logging.warning("Trinocular camera already opened")
        return True

    status = self.sensor_controller.open_trinocular_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to open trinocular camera: {status.message}")
        return False

    self.sensors_state["trinocular_camera"] = True
    logging.info("✓ Trinocular camera opened")
    return True

def close_trinocular_camera(self) -> bool:
    """Close trinocular camera"""
    if not self.sensors_state["trinocular_camera"]:
        logging.warning("Trinocular camera already closed")
        return True

    # Unsubscribe if subscribed
    if self.subscriptions["trinocular_image"]:
        self.toggle_trinocular_image_subscription()

    status = self.sensor_controller.close_trinocular_camera()
    if status.code != ErrorCode.OK:
        logging.error(f"Failed to close trinocular camera: {status.message}")
        return False

    self.sensors_state["trinocular_camera"] = False
    logging.info("✓ Trinocular camera closed")
    return True

# === LiDAR Subscribe/Unsubscribe Methods ===
def toggle_lidar_imu_subscription(self):
    """Toggle LiDAR IMU subscription"""
    if self.subscriptions["lidar_imu"]:

```

(下页继续)

(续上页)

```

        self.sensor_controller.unsubscribe_lidar_imu()
        self.subscriptions["lidar_imu"] = False
        logging.info("✓ LiDAR IMU unsubscribed")

    else:

        def lidar_imu_callback(imu):
            global lidar_imu_counter
            lidar_imu_counter += 1
            if lidar_imu_counter % 100 == 0:
                logging.info(f"LiDAR IMU received (count: {lidar_imu_counter})")

        self.sensor_controller.subscribe_lidar_imu(lidar_imu_callback)
        self.subscriptions["lidar_imu"] = True
        logging.info("✓ LiDAR IMU subscribed")

    def toggle_lidar_point_cloud_subscription(self):
        """Toggle LiDAR point cloud subscription"""
        if self.subscriptions["lidar_point_cloud"]:
            self.sensor_controller.unsubscribe_lidar_point_cloud()
            self.subscriptions["lidar_point_cloud"] = False
            logging.info("✓ LiDAR point cloud unsubscribed")
        else:

            def lidar_pointcloud_callback(pointcloud):
                global lidar_pointcloud_counter
                lidar_pointcloud_counter += 1
                if lidar_pointcloud_counter % 10 == 0:
                    logging.info(
                        f"LiDAR Point Cloud: {len(pointcloud.data)} bytes (count: {lidar_
                        -pointcloud_counter})"
                    )

            self.sensor_controller.subscribe_lidar_point_cloud(
                lidar_pointcloud_callback
            )
            self.subscriptions["lidar_point_cloud"] = True
            logging.info("✓ LiDAR point cloud subscribed")

    # === Head RGBD Subscribe/Unsubscribe Methods ===
    def toggle_head_rgbd_color_image_subscription(self):
        """Toggle head RGBD color image subscription"""
        if self.subscriptions["head_rgbd_color_image"]:
            self.sensor_controller.unsubscribe_head_rgbd_color_image()

```

(下页继续)

(续上页)

```

    self.subscriptions["head_rgbd_color_image"] = False
    logging.info("✓ Head RGBD color image unsubscribed")

else:

    def head_rgbd_color_image_callback(img):
        global head_rgbd_color_counter
        head_rgbd_color_counter += 1
        if head_rgbd_color_counter % 15 == 0:
            logging.info(
                f"Head RGBD Color Image: {len(img.data)} bytes, {img.width}x{img.height}"
                ↪ (count: {head_rgbd_color_counter})"
            )

        self.sensor_controller.subscribe_head_rgbd_color_image(
            head_rgbd_color_image_callback
        )
        self.subscriptions["head_rgbd_color_image"] = True
        logging.info("✓ Head RGBD color image subscribed")

def toggle_head_rgbd_color_camera_info_subscription(self):
    """Toggle head RGBD color camera info subscription"""
    if self.subscriptions["head_rgbd_color_camera_info"]:
        self.sensor_controller.unsubscribe_head_rgbd_color_camera_info()
        self.subscriptions["head_rgbd_color_camera_info"] = False
        logging.info("✓ Head RGBD color camera info unsubscribed")
    else:
        self.sensor_controller.subscribe_head_rgbd_color_camera_info(
            lambda info: logging.info(
                f"Head RGBD Color Camera Info: {info.width}x{info.height}"
            )
        )
        self.subscriptions["head_rgbd_color_camera_info"] = True
        logging.info("✓ Head RGBD color camera info subscribed")

def toggle_head_rgbd_depth_image_subscription(self):
    """Toggle head RGBD depth image subscription"""
    if self.subscriptions["head_rgbd_depth_image"]:
        self.sensor_controller.unsubscribe_head_rgbd_depth_image()
        self.subscriptions["head_rgbd_depth_image"] = False
        logging.info("✓ Head RGBD depth image unsubscribed")
    else:

        def head_rgbd_depth_image_callback(img):

```

(下页继续)

(续上页)

```

    global head_rgbd_depth_image_counter
    head_rgbd_depth_image_counter += 1
    if head_rgbd_depth_image_counter % 15 == 0:
        logging.info(
            f"Head RGBD Depth Image: {len(img.data)} bytes, {img.width}x{img.height}"
            ↵ (count: {head_rgbd_depth_image_counter})"
        )

    self.sensor_controller.subscribe_head_rgbd_depth_image(
        head_rgbd_depth_image_callback
    )
    self.subscriptions["head_rgbd_depth_image"] = True
    logging.info("✓ Head RGBD depth image subscribed")

def toggle_head_rgbd_depth_camera_info_subscription(self):
    """Toggle head RGBD depth camera info subscription"""
    if self.subscriptions["head_rgbd_depth_camera_info"]:
        self.sensor_controller.unsubscribe_head_rgbd_depth_camera_info()
        self.subscriptions["head_rgbd_depth_camera_info"] = False
        logging.info("✓ Head RGBD depth camera info unsubscribed")
    else:

        def head_rgbd_depth_camera_info_callback(info):
            global head_rgbd_depth_camera_info_counter
            head_rgbd_depth_camera_info_counter += 1
            if head_rgbd_depth_camera_info_counter % 15 == 0:
                logging.info(
                    f"Head RGBD Depth Camera Info: {info.width}x{info.height} (count: {head_
                    ↵rgbd_depth_camera_info_counter})"
                )

        self.sensor_controller.subscribe_head_rgbd_depth_camera_info(
            head_rgbd_depth_camera_info_callback
        )
        self.subscriptions["head_rgbd_depth_camera_info"] = True
        logging.info("✓ Head RGBD depth camera info subscribed")

    # === Waist RGBD Subscribe/Unsubscribe Methods ===
    def toggle_waist_rgbd_color_image_subscription(self):
        """Toggle waist RGBD color image subscription"""
        if self.subscriptions["waist_rgbd_color_image"]:
            self.sensor_controller.unsubscribe_waist_rgbd_color_image()
            self.subscriptions["waist_rgbd_color_image"] = False

```

(下页继续)

(续上页)

```

        logging.info("✓ Waist RGBD color image unsubscribed")
    else:

        def waist_rgbd_color_image_callback(img):
            global waist_rgbd_color_counter
            waist_rgbd_color_counter += 1
            if waist_rgbd_color_counter % 15 == 0:
                logging.info(
                    f"Waist RGBD Color Image: {len(img.data)} bytes, ({img.width}x{img.
                    height}) (count: {waist_rgbd_color_counter})"
                )

            self.sensor_controller.subscribe_waist_rgbd_color_image(
                waist_rgbd_color_image_callback
            )
            self.subscriptions["waist_rgbd_color_image"] = True
            logging.info("✓ Waist RGBD color image subscribed")

    def toggle_waist_rgbd_color_camera_info_subscription(self):
        """Toggle waist RGBD color camera info subscription"""
        if self.subscriptions["waist_rgbd_color_camera_info"]:
            self.sensor_controller.unsubscribe_waist_rgbd_color_camera_info()
            self.subscriptions["waist_rgbd_color_camera_info"] = False
            logging.info("✓ Waist RGBD color camera info unsubscribed")
        else:

            def waist_rgbd_color_camera_info_callback(info):
                global waist_rgbd_color_camera_info_counter
                waist_rgbd_color_camera_info_counter += 1
                if waist_rgbd_color_camera_info_counter % 15 == 0:
                    logging.info(
                        f"Waist RGBD Color Camera Info: {info.width}x{info.height} (count:
                        {waist_rgbd_color_camera_info_counter})"
                    )

            self.sensor_controller.subscribe_waist_rgbd_color_camera_info(
                waist_rgbd_color_camera_info_callback
            )
            self.subscriptions["waist_rgbd_color_camera_info"] = True
            logging.info("✓ Waist RGBD color camera info subscribed")

    def toggle_waist_rgbd_depth_image_subscription(self):
        """Toggle waist RGBD depth image subscription"""

```

(下页继续)

(续上页)

```

if self.subscriptions["waist_rgbd_depth_image"]:
    self.sensor_controller.unsubscribe_waist_rgbd_depth_image()
    self.subscriptions["waist_rgbd_depth_image"] = False
    logging.info("✓ Waist RGBD depth image unsubscribed")

else:

    def waist_rgbd_depth_image_callback(img):
        global waist_rgbd_depth_image_counter
        waist_rgbd_depth_image_counter += 1
        if waist_rgbd_depth_image_counter % 15 == 0:
            logging.info(
                f"Waist RGBD Depth Image: {len(img.data)} bytes, ({img.width}x{img.
                height}) (count: {waist_rgbd_depth_image_counter})"
            )

        self.sensor_controller.subscribe_waist_rgbd_depth_image(
            waist_rgbd_depth_image_callback
        )
    self.subscriptions["waist_rgbd_depth_image"] = True
    logging.info("✓ Waist RGBD depth image subscribed")

def toggle_waist_rgbd_depth_camera_info_subscription(self):
    """Toggle waist RGBD depth camera info subscription"""
    if self.subscriptions["waist_rgbd_depth_camera_info"]:
        self.sensor_controller.unsubscribe_waist_rgbd_depth_camera_info()
        self.subscriptions["waist_rgbd_depth_camera_info"] = False
        logging.info("✓ Waist RGBD depth camera info unsubscribed")
    else:

        def waist_rgbd_depth_camera_info_callback(info):
            global waist_rgbd_depth_camera_info_counter
            waist_rgbd_depth_camera_info_counter += 1
            if waist_rgbd_depth_camera_info_counter % 15 == 0:
                logging.info(
                    f"Waist RGBD Depth Camera Info: {info.width}x{info.height} (count:
                    {waist_rgbd_depth_camera_info_counter})"
                )

        self.sensor_controller.subscribe_waist_rgbd_depth_camera_info(
            waist_rgbd_depth_camera_info_callback
        )
    self.subscriptions["waist_rgbd_depth_camera_info"] = True
    logging.info("✓ Waist RGBD depth camera info subscribed")

```

(下页继续)

(续上页)

```

# === Trinocular Camera Subscribe/Unsubscribe Methods ===

def toggle_trinocular_image_subscription(self):
    """Toggle trinocular camera image subscription"""
    if self.subscriptions["trinocular_image"]:
        self.sensor_controller.unsubscribe_trinocular_image()
        self.subscriptions["trinocular_image"] = False
        logging.info("✓ Trinocular camera image unsubscribed")
    else:

        def trinocular_image_callback(frame):
            global trinocular_image_counter
            trinocular_image_counter += 1
            if trinocular_image_counter % 15 == 0:
                logging.info(
                    f"Trinocular Camera Frame received (count: {trinocular_image_counter})"
                )

        self.sensor_controller.subscribe_trinocular_image(trinocular_image_callback)
        self.subscriptions["trinocular_image"] = True
        logging.info("✓ Trinocular camera image subscribed")

    def show_status(self):
        """Display current sensor status"""
        logging.info("\n" + "=" * 80)
        logging.info("MAGICBOT GEN1 SENSOR STATUS")
        logging.info("=" * 80)
        logging.info(
            f"LiDAR:           {'OPEN' if self.sensors_state['lidar'] else 'CLOSED'}\n"
        )
        logging.info(
            f"Head RGBD Camera:      {'OPEN' if self.sensors_state['head_rgbd_camera'] else 'CLOSED'}\n"
        )
        logging.info(
            f"Waist RGBD Camera:     {'OPEN' if self.sensors_state['waist_rgbd_camera'] else 'CLOSED'}\n"
        )
        logging.info(
            f"Trinocular Camera:    {'OPEN' if self.sensors_state['trinocular_camera'] else 'CLOSED'}\n"
        )

```

(下页继续)

(续上页)

```

        logging.info("\nLiDAR SUBSCRIPTIONS:")
        logging.info(
            f"    LiDAR IMU:           {'✓ SUBSCRIBED' if self.subscriptions['lidar_imu']}"
        )
        if self.subscriptions['lidar_imu']:
            print("Subscribed to LiDAR IMU")
        else:
            print("Unsubscribed from LiDAR IMU")

        logging.info(
            f"    LiDAR Point Cloud:   {'✓ SUBSCRIBED' if self.subscriptions['lidar_point_cloud']}"
        )
        if self.subscriptions['lidar_point_cloud']:
            print("Subscribed to LiDAR Point Cloud")
        else:
            print("Unsubscribed from LiDAR Point Cloud")

        logging.info("\nHEAD RGBD SUBSCRIPTIONS:")
        logging.info(
            f"    Color Image:         {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_color_image']}"
        )
        if self.subscriptions['head_rgbd_color_image']:
            print("Subscribed to Head RGBD Color Image")
        else:
            print("Unsubscribed from Head RGBD Color Image")

        logging.info(
            f"    Color Camera Info:  {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_color_camera_info']}"
        )
        if self.subscriptions['head_rgbd_color_camera_info']:
            print("Subscribed to Head RGBD Color Camera Info")
        else:
            print("Unsubscribed from Head RGBD Color Camera Info")

        logging.info(
            f"    Depth Image:          {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_depth_image']}"
        )
        if self.subscriptions['head_rgbd_depth_image']:
            print("Subscribed to Head RGBD Depth Image")
        else:
            print("Unsubscribed from Head RGBD Depth Image")

        logging.info(
            f"    Depth Camera Info:   {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_depth_camera_info']}"
        )
        if self.subscriptions['head_rgbd_depth_camera_info']:
            print("Subscribed to Head RGBD Depth Camera Info")
        else:
            print("Unsubscribed from Head RGBD Depth Camera Info")

        logging.info("\nWAIST RGBD SUBSCRIPTIONS:")
        logging.info(
            f"    Color Image:         {'✓ SUBSCRIBED' if self.subscriptions['waist_rgbd_color_image']}"
        )
        if self.subscriptions['waist_rgbd_color_image']:
            print("Subscribed to Waist RGBD Color Image")
        else:
            print("Unsubscribed from Waist RGBD Color Image")

        logging.info(
            f"    Color Camera Info:  {'✓ SUBSCRIBED' if self.subscriptions['waist_rgbd_color_camera_info']}"
        )
        if self.subscriptions['waist_rgbd_color_camera_info']:
            print("Subscribed to Waist RGBD Color Camera Info")
        else:
            print("Unsubscribed from Waist RGBD Color Camera Info")

        logging.info(
            f"    Depth Image:          {'✓ SUBSCRIBED' if self.subscriptions['waist_rgbd_depth_image']}"
        )
        if self.subscriptions['waist_rgbd_depth_image']:
            print("Subscribed to Waist RGBD Depth Image")
        else:
            print("Unsubscribed from Waist RGBD Depth Image")

        logging.info(
            f"    Depth Camera Info:   {'✓ SUBSCRIBED' if self.subscriptions['waist_rgbd_depth_camera_info']}"
        )
        if self.subscriptions['waist_rgbd_depth_camera_info']:
            print("Subscribed to Waist RGBD Depth Camera Info")
        else:
            print("Unsubscribed from Waist RGBD Depth Camera Info")
    )

```

(下页继续)

(续上页)

```

        logging.info("\nTRINOCULAR CAMERA SUBSCRIPTIONS:")
        logging.info(
            f"    Trinocular Image:           {'✓' if self.subscriptions['trinocular_image'] else '✗' } UNSUBSCRIBED"
        )
        logging.info("=" * 80 + "\n")

def print_menu():
    """Print interactive menu"""
    logging.info("\n" + "=" * 80)
    logging.info("MAGICBOT GEN1 SENSOR CONTROL MENU")
    logging.info("=" * 80)
    logging.info("Sensor Open/Close:")
    logging.info("    1 - Open LiDAR             2 - Close LiDAR")
    logging.info("    3 - Open Head RGBD Camera 4 - Close Head RGBD Camera")
    logging.info("    5 - Open Waist RGBD Camera 6 - Close Waist RGBD Camera")
    logging.info("    7 - Open Trinocular Camera 8 - Close Trinocular Camera")
    logging.info("\nLiDAR Subscriptions:")
    logging.info("    i - Toggle LiDAR IMU       p - Toggle LiDAR Point Cloud")
    logging.info("\nHead RGBD Camera Subscriptions:")
    logging.info(
        "    c - Toggle Head Color Image      C - Toggle Head Color Camera Info"
    )
    logging.info(
        "    d - Toggle Head Depth Image     D - Toggle Head Depth Camera Info"
    )
    logging.info("\nWaist RGBD Camera Subscriptions:")
    logging.info(
        "    w - Toggle Waist Color Image    W - Toggle Waist Color Camera Info"
    )
    logging.info(
        "    e - Toggle Waist Depth Image   E - Toggle Waist Depth Camera Info"
    )
    logging.info("\nTrinocular Camera Subscriptions:")
    logging.info("    t - Toggle Trinocular Image")
    logging.info("\nCommands:")
    logging.info(
        "    s - Show Status              ESC - Quit          ? - Help"
    )
    logging.info("=" * 80)

```

(下页继续)

(续上页)

```

def main():
    """Main function"""
    logging.info("\n" + "=" * 80)
    logging.info("MagicBot Gen1 SDK Sensor Interactive Example")
    logging.info("=" * 80 + "\n")

    local_ip = "192.168.55.10"
    robot = magicbot.MagicRobot()

    if not robot.initialize(local_ip):
        logging.error("Robot initialization failed")
        return

    status = robot.connect()
    if status.code != ErrorCode.OK:
        logging.error(f"Robot connection failed: {status.message}")
        robot.shutdown()
        return

    logging.info("✓ Robot connected successfully\n")

    sensor_controller = robot.get_sensor_controller()

    # Initialize sensor controller
    if not sensor_controller.initialize():
        logging.error("Sensor controller initialization failed")
        robot.disconnect()
        robot.shutdown()
        return

    logging.info("✓ Sensor controller initialized successfully\n")

    sensor_manager = SensorManager(sensor_controller)

    print_menu()

    try:
        while True:
            choice = input("\nEnter your choice: ").strip()

            if choice == "\x1b": # ESC key
                logging.info("ESC key pressed, exiting program...")
                break

```

(下页继续)

(续上页)

```

# Sensor open/close control
if choice == "1":
    sensor_manager.open_lidar()
elif choice == "2":
    sensor_manager.close_lidar()
elif choice == "3":
    sensor_manager.open_head_rgbd_camera()
elif choice == "4":
    sensor_manager.close_head_rgbd_camera()
elif choice == "5":
    sensor_manager.open_waist_rgbd_camera()
elif choice == "6":
    sensor_manager.close_waist_rgbd_camera()
elif choice == "7":
    sensor_manager.open_trinocular_camera()
elif choice == "8":
    sensor_manager.close_trinocular_camera()

# LiDAR subscriptions
elif choice == "i":
    sensor_manager.toggle_lidar_imu_subscription()
elif choice == "p":
    sensor_manager.toggle_lidar_point_cloud_subscription()

# Head RGBD subscriptions
elif choice == "c":
    sensor_manager.toggle_head_rgbd_color_image_subscription()
elif choice == "C":
    sensor_manager.toggle_head_rgbd_color_camera_info_subscription()
elif choice == "d":
    sensor_manager.toggle_head_rgbd_depth_image_subscription()
elif choice == "D":
    sensor_manager.toggle_head_rgbd_depth_camera_info_subscription()

# Waist RGBD subscriptions
elif choice == "w":
    sensor_manager.toggle_waist_rgbd_color_image_subscription()
elif choice == "W":
    sensor_manager.toggle_waist_rgbd_color_camera_info_subscription()
elif choice == "e":
    sensor_manager.toggle_waist_rgbd_depth_image_subscription()
elif choice == "E":

```

(下页继续)

(续上页)

```

        sensor_manager.toggle_waist_rgbd_depth_camera_info_subscription()

    # Trinocular camera subscriptions
    elif choice == "t":
        sensor_manager.toggle_trinocular_image_subscription()

    # Commands
    elif choice.lower() == "s":
        sensor_manager.show_status()
    elif choice == "?" or choice.lower() == "help":
        print_menu()
    else:
        logging.warning(f"Invalid choice: '{choice}'. Press '?' for help.")

except KeyboardInterrupt:
    logging.info("\nReceived keyboard interrupt, shutting down...")
except Exception as e:
    logging.error(f"Error occurred: {e}")
    import traceback

    traceback.print_exc()
finally:
    # Cleanup: unsubscribe all and close all sensors
    logging.info("Cleaning up...")

    # Close all sensors
    if sensor_manager.sensors_state["lidar"]:
        sensor_manager.close_lidar()
    if sensor_manager.sensors_state["head_rgbd_camera"]:
        sensor_manager.close_head_rgbd_camera()
    if sensor_manager.sensors_state["waist_rgbd_camera"]:
        sensor_manager.close_waist_rgbd_camera()
    if sensor_manager.sensors_state["trinocular_camera"]:
        sensor_manager.close_trinocular_camera()

    # Allow time for cleanup
    time.sleep(1)

    sensor_controller.shutdown()
    logging.info("Sensor controller shutdown")

    robot.disconnect()
    logging.info("Robot disconnected")

```

(下页继续)

(续上页)

```

robot.shutdown()
logging.info("Robot shutdown")

if __name__ == "__main__":
    main()

```

23.3 运行说明

23.3.1 环境准备

```

export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

```

23.3.2 运行示例

```

# C++
./sensor_example

# Python
python3 sensor_example.py

```

23.3.3 控制说明

- 传感器开关控制:
 - 1 - 打开激光雷达
 - 2 - 关闭激光雷达
 - 3 - 打开头部 RGBD 相机
 - 4 - 关闭头部 RGBD 相机
 - 5 - 打开腰部 RGBD 相机
 - 6 - 关闭腰部 RGBD 相机
 - 7 - 打开三目相机
 - 8 - 关闭三目相机
- 激光雷达数据订阅:
 - i - 切换 IMU 数据订阅
 - p - 切换点云数据订阅

- 头部 RGBD 相机数据订阅:
 - c - 切换彩色图像订阅
 - C - 切换彩色相机参数订阅
 - d - 切换深度图像订阅
 - D - 切换深度相机参数订阅
- 腰部 RGBD 相机数据订阅:
 - w - 切换彩色图像订阅
 - W - 切换彩色相机参数订阅
 - e - 切换深度图像订阅
 - E - 切换深度相机参数订阅
- 三目相机数据订阅:
 - t - 切换图像订阅
- 其他命令:
 - s - 显示当前状态
 - ? - 显示帮助菜单

23.3.4 停止程序

- 按 ESC 可以安全停止程序
- 程序会自动清理所有资源

CHAPTER 24

语音控制示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行初始化、连接机器人、音频控制（获取音量、设置音量、TTS 播放、音频流订阅）等基本操作。

24.1 C++

示例文件: audio_example.cpp

```
#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <termios.h>
#include <unistd.h>
#include <csignal>

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
```

(下页继续)

(续上页)

```

robot.Shutdown();
// Exit process
exit(signum);
}

void print_help() {
    std::cout << "Key Function Description:\n";
    std::cout << "  Audio Functions:\n";
    std::cout << "    1      Function 1: Get volume\n";
    std::cout << "    2      Function 2: Set volume\n";
    std::cout << "    3      Function 3: Play TTS\n";
    std::cout << "    4      Function 4: Stop playback\n";
    std::cout << "  Audio stream Functions:\n";
    std::cout << "    5      Function 5: Open audio stream\n";
    std::cout << "    6      Function 6: Close audio stream\n";
    std::cout << "    7      Function 7: Subscribe to audio stream\n";
    std::cout << "    8      Function 8: Unsubscribe to audio stream\n";
    std::cout << "  Wakeup Status Functions:\n";
    std::cout << "    q      Function q: Open wakeup status stream\n";
    std::cout << "    w      Function w: Close wakeup status stream\n";
    std::cout << "    e      Function e: Subscribe to wakeup status\n";
    std::cout << "    r      Function r: Unsubscribe to wakeup status\n";
    std::cout << "\n";
    std::cout << "    ?      Function ?: Print help\n";
    std::cout << "    ESC    Exit program\n";
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt); // Get current terminal settings
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO); // Disable buffering and echo
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar(); // Read key press
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt); // Restore settings
    return ch;
}

void GetVolume() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
}

```

(下页继续)

(续上页)

```

// Get volume
int get_volume = 0;
auto status = controller.GetVolume(get_volume);
if (status.code != ErrorCode::OK) {
    std::cerr << "get volume failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}
std::cout << "get volume success, volume: " << std::to_string(get_volume) << std::endl;
}

void SetVolume() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Set volume
    auto status = controller.SetVolume(50);
    if (status.code != ErrorCode::OK) {
        std::cerr << "set volume failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "set volume success" << std::endl;
}

void PlayTts() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Play speech
    TtsCommand tts;
    tts.id = "100000000001";
    tts.content = "How's the weather today!";
    tts.priority = TtsPriority::HIGH;
    tts.mode = TtsMode::CLEARTOP;
    auto status = controller.Play(tts);
    if (status.code != ErrorCode::OK) {
        std::cerr << "play tts failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "play tts success" << std::endl;
}

```

(下页继续)

(续上页)

}

```
void StopTts() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Stop speech playback
    auto status = controller.Stop();
    if (status.code != ErrorCode::OK) {
        std::cerr << "stop tts failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "stop tts success" << std::endl;
}

void OpenAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Open audio stream
    auto status = controller.OpenAudioStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "open audio stream failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "open audio stream success" << std::endl;
}

void CloseAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Close audio stream
    auto status = controller.CloseAudioStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "close audio stream failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "close audio stream success" << std::endl;
}
```

(下页继续)

(续上页)

```

void SubscribeAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();

    // Subscribe to audio streams
    controller.SubscribeOriginAudioStream([](const std::shared_ptr<AudioStream> data) {
        static int32_t counter = 0;
        if (counter++ % 30 == 0) {
            std::cout << "Received origin audio stream data, size: " << data->data_length << std::endl;
        }
    });
}

controller.SubscribeBfAudioStream([](const std::shared_ptr<AudioStream> data) {
    static int32_t counter = 0;
    if (counter++ % 30 == 0) {
        std::cout << "Received bf audio stream data, size: " << data->data_length << std::endl;
    }
});
std::cout << "Subscribed to audio streams" << std::endl;
}

void UnsubscribeAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Unsubscribe from audio streams
    controller.UnsubscribeOriginAudioStream();
    controller.UnsubscribeBfAudioStream();
    std::cout << "Unsubscribed from audio streams" << std::endl;
}

void OpenWakeupStatusStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Open wakeup status stream
    auto status = controller.OpenWakeupStatusStream();
}

void CloseWakeupStatusStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Close wakeup status stream
}

```

(下页继续)

(续上页)

```

auto status = controller.CloseWakeupStatusStream();
}

void SubscribeWakeupStatus() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Subscribe to wakeup status
    controller.SubscribeWakeupStatus([](const std::shared_ptr<WakeupStatus> data) {
        static int32_t counter = 0;
        if (data->is_wakeup) {
            if (data->enable_wakeup_orientation) {
                std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup << ", enable_"
                    << "wakeup_orientation: " << data->enable_wakeup_orientation << ", wakeup_orientation: " << data->
                    << "wakeup_orientation << std::endl;
            } else {
                std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup << std::endl;
            }
        } else {
            std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup << std::endl;
        }
    });
}

void UnsubscribeWakeupStatus() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Unsubscribe from wakeup status
    controller.UnsubscribeWakeupStatus();
}

int main(int argc, char* argv[]) {
    // Bind SIGINT (Ctrl+C)
    signal(SIGINT, signalHandler);

    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;

    print_help();

    std::string local_ip = "192.168.54.111";
    // Configure local IP address for direct network connection and initialize SDK
    if (!robot.Initialize(local_ip)) {
        std::cerr << "robot sdk initialize failed." << std::endl;
        robot.Shutdown();
    }
}

```

(下页继续)

(续上页)

```

    return -1;
}

// Connect to robot
auto status = robot.Connect();
if (status.code != ErrorCode::OK) {
    std::cerr << "connect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
robot.Shutdown();
return -1;
}
std::cout << "Press any key to continue (ESC to exit)..." 
        << std::endl;

// Wait for user input
while (1) {
    int key = getch();
    if (key == 27)
        break; // ESC key ASCII code is 27

    std::cout << "Key ASCII: " << key << ", Character: " << static_cast<char>(key) << std::endl;
    switch (key) {
        // 1. Audio Functions
        case '1': {
            // Get volume
            GetVolume();
            break;
        }
        case '2': {
            // Set volume
            SetVolume();
            break;
        }
        case '3': {
            PlayTts();
            break;
        }
        case '4': {
            StopTts();
            break;
        }
        // 2. Audio stream Functions
    }
}

```

(下页继续)

(续上页)

```

case '5': {
    OpenAudioStream();
    break;
}

case '6': {
    CloseAudioStream();
    break;
}

case '7': {
    SubscribeAudioStream();
    break;
}

case '8': {
    UnsubscribeAudioStream();
    break;
}

// 3. Wakeup Status Functions

case 'Q':
case 'q': {
    OpenWakeupStatusStream();
    break;
}

case 'W':
case 'w': {
    CloseWakeupStatusStream();
    break;
}

case 'E':
case 'e': {
    SubscribeWakeupStatus();
    break;
}

case 'R':
case 'r': {
    UnsubscribeWakeupStatus();
    break;
}

case '?': {
    print_help();
    break;
}

default:
    std::cout << "Unknown key: " << key << std::endl;

```

(下页继续)

(续上页)

```

        break;
    }

    usleep(10000);
}

// Disconnect from robot
status = robot.Disconnect();
if (status.code != ErrorCode::OK) {
    std::cerr << "disconnect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
robot.Shutdown();
return -1;
}

robot.Shutdown();

return 0;
}

```

24.2 Python

示例文件: audio_example.py

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import time
import signal
import threading
import logging
from typing import Optional

import magicbot_gen1_python as magicbot

# Configure logging format and level
logging.basicConfig(
    level=logging.INFO,  # Minimum log level
    format="(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

```

(下页继续)

(续上页)

```

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True

def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global running, robot
    logging.info("Received interrupt signal (%s), exiting...", signum)
    running = False
    if robot:
        robot.shutdown()
    logging.info("Robot shutdown")
    exit(-1)

def print_help():
    """Print help information"""
    logging.info("Key Function Demo Program")
    logging.info("")
    logging.info("Audio Functions:")
    logging.info("  1      Function 1: Get volume")
    logging.info("  2      Function 2: Set volume")
    logging.info("  3      Function 3: Play TTS")
    logging.info("  4      Function 4: Stop playback")
    logging.info("")
    logging.info("Audio stream Functions:")
    logging.info("  5      Function 5: Open audio stream")
    logging.info("  6      Function 6: Close audio stream")
    logging.info("  7      Function 7: Subscribe to audio stream")
    logging.info("  8      Function 8: Unsubscribe to audio stream")
    logging.info("")
    logging.info("Wakeup Status Functions:")
    logging.info("  Q      Function Q: Open wakeup status stream")
    logging.info("  W      Function W: Close wakeup status stream")
    logging.info("  E      Function E: Subscribe to wakeup status")
    logging.info("  R      Function R: Unsubscribe to wakeup status")
    logging.info("")
    logging.info("  ?      Function ?: Print help")
    logging.info("  ESC    Exit program")

def get_volume():

```

(下页继续)

(续上页)

```
"""Get volume"""
global robot
try:
    # Get audio controller
    controller = robot.get_audio_controller()

    # Get volume
    status, volume = controller.get_volume()
    if status.code != magicbot.ErrorCode.OK:
        logging.error(
            "Failed to get volume, code: %s, message: %s",
            status.code,
            status.message,
        )
    return

    logging.info("Successfully got volume, volume: %s", volume)
except Exception as e:
    logging.error("Exception occurred while getting volume: %s", e)

def set_volume(volume):
    """Set volume"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Set volume to 7
        status = controller.set_volume(volume)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set volume, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully set volume")
    except Exception as e:
        logging.error("Exception occurred while setting volume: %s", e)
```

(下页继续)

(续上页)

```

def play_tts(content):
    """Play TTS speech"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Create TTS command
        tts = magicbot.TtsCommand()
        tts.id = "100000000001"
        tts.content = content
        tts.priority = magicbot.TtsPriority.HIGH
        tts.mode = magicbot.TtsMode.CLEARTOP

        # Play speech
        status = controller.play(tts)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to play TTS, code: %s, message: %s", status.code, status.message
            )
    return

    logging.info("Successfully played TTS")
except Exception as e:
    logging.error("Exception occurred while playing TTS: %s", e)

def stop_tts():
    """Stop TTS playback"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Stop speech playback
        status = controller.stop()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to stop TTS, code: %s, message: %s", status.code, status.message
            )
    return

    logging.info("Successfully stopped TTS")

```

(下页继续)

(续上页)

```

except Exception as e:
    logging.error("Exception occurred while stopping TTS: %s", e)

def open_audio_stream():
    """Open audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Open audio stream
        status = controller.open_audio_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to open audio stream, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully opened audio stream")
except Exception as e:
    logging.error("Exception occurred while opening audio stream: %s", e)

def close_audio_stream():
    """Close audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Close audio stream
        status = controller.close_audio_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close audio stream, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

```

(下页继续)

(续上页)

```
logging.info("Successfully closed audio stream")
except Exception as e:
    logging.error("Exception occurred while closing audio stream: %s", e)

def subscribe_audio_stream():
    """Subscribe to audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Audio stream counters
        origin_counter = 0
        bf_counter = 0

        def origin_audio_callback(audio_stream):
            """Original audio stream callback function"""
            nonlocal origin_counter
            if origin_counter % 30 == 0:
                logging.info(
                    "Received original audio stream data, size: %d",
                    audio_stream.data_length,
                )
                sys.stdout.write("\r")
                sys.stdout.flush()
            origin_counter += 1

        def bf_audio_callback(audio_stream):
            """BF audio stream callback function"""
            nonlocal bf_counter
            if bf_counter % 30 == 0:
                logging.info(
                    "Received BF audio stream data, size: %d", audio_stream.data_length
                )
                sys.stdout.write("\r")
                sys.stdout.flush()
            bf_counter += 1

        # Subscribe to audio streams
        controller.subscribe_origin_audio_stream(origin_audio_callback)
        controller.subscribe_bf_audio_stream(bf_audio_callback)
```

(下页继续)

(续上页)

```

        logging.info("Subscribed to audio streams")
    except Exception as e:
        logging.error("Exception occurred while subscribing to audio stream: %s", e)

def unsubscribe_audio_stream():
    """Unsubscribe to audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Unsubscribe to audio stream
        controller.unsubscribe_bf_audio_stream()
        controller.unsubscribe_origin_audio_stream()

        logging.info("Unsubscribed to audio stream")
    except Exception as e:
        logging.error("Exception occurred while unsubscribing to audio stream: %s", e)

def open_wakeup_status_stream():
    """Open wakeup status stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Open wakeup status stream
        status = controller.open_wakeup_status_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to open wakeup status stream, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully opened wakeup status stream")
    except Exception as e:
        logging.error("Exception occurred while opening wakeup status stream: %s", e)

```

(下页继续)

(续上页)

```

def close_wakeup_status_stream():
    """Close wakeup status stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Close wakeup status stream
        status = controller.close_wakeup_status_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close wakeup status stream, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully closed wakeup status stream")
except Exception as e:
    logging.error("Exception occurred while closing wakeup status stream: %s", e)

def unsubscribe_wakeup_status():
    """Unsubscribe to wakeup status"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Unsubscribe to wakeup status
        controller.unsubscribe_wakeup_status()

        logging.info("Unsubscribed to wakeup status")
    except Exception as e:
        logging.error("Exception occurred while unsubscribing to wakeup status: %s", e)

def subscribe_wakeup_status():
    """Subscribe to wakeup status"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

```

(下页继续)

(续上页)

```

# Wakeup status counter
wakeup_counter = 0

def wakeup_status_callback(wakeup_status):
    """Wakeup status callback function"""
    nonlocal wakeup_counter
    if wakeup_status.is_wakeup:
        if wakeup_status.enable_wakeup_orientation:
            logging.info(
                "Voice wakeup detected! Orientation: %.2f radians (%.1f degrees)",
                wakeup_status.wakeup_orientation,
                wakeup_status.wakeup_orientation * 180.0 / 3.14159,
            )
    else:
        logging.info("Voice wakeup detected!")
        sys.stdout.write("\r")
        sys.stdout.flush()
    else:
        if wakeup_counter % 10 == 0: # Log every 50th non-wakeup status
            logging.info(
                "Wakeup status: sleeping, enable_wakeup_orientation: %s, orientation: %.
                ↵2f radians",
                wakeup_status.enable_wakeup_orientation,
                wakeup_status.wakeup_orientation * 180.0 / 3.14159,
            )
            sys.stdout.write("\r")
            sys.stdout.flush()
    wakeup_counter += 1

    # Subscribe to wakeup status
    controller.subscribe_wakeup_status(wakeup_status_callback)

    logging.info("Subscribed to wakeup status stream")
except Exception as e:
    logging.error("Exception occurred while subscribing to wakeup status: %s", e)

def get_user_input():
    """Get user input - Read a single line of data"""
    try:
        # Method 1: Read a line using input() (recommended)
        return input("Enter command: ").strip()

```

(下页继续)

(续上页)

```
except (EOFError, KeyboardInterrupt):
    return ""

def main():
    """Main function"""
    global robot, running

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
            return -1

        # Connect to robot
        status = robot.connect()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to connect to robot, code: %s, message: %s",
                status.code,
                status.message,
            )
            robot.shutdown()
            return -1

        logging.info("Successfully connected to robot")

        # Initialize audio controller
        audio_controller = robot.get_audio_controller()
        if not audio_controller.initialize():
            logging.error("Failed to initialize audio controller")
            robot.disconnect()
            robot.shutdown()

    
```

(下页继续)

(续上页)

```

    return -1

logging.info("Successfully initialized audio controller")
print_help()
logging.info("Press any key to continue (ESC to exit)...")


# Main loop
while running:
    try:
        str_input = get_user_input()

        # Split input parameters by space
        parts = str_input.strip().split()

        if not parts:
            time.sleep(0.01) # Brief delay
            continue

        # Parse parameters
        key = parts[0]
        args = parts[1:] if len(parts) > 1 else []
        if key == "\x1b": # ESC key
            break

        # 1. Audio Functions
        # 1.1 Get volume
        if key == "1":
            get_volume()

        # 1.2 Set volume
        elif key == "2":
            volume = args[0] if args else 50
            set_volume(volume)

        # 1.3 Play TTS
        elif key == "3":
            content = args[0] if args else "How's the weather today!"
            play_tts(content)

        # 1.4 Stop TTS
        elif key == "4":
            stop_tts()

        # 2. Audio Stream Functions
        # 2.1 Open audio stream
        elif key == "5":
            open_audio_stream()

        # 2.2 Close audio stream
    
```

(下页继续)

(续上页)

```

    elif key == "6":
        close_audio_stream()
    # 2.3 Subscribe to audio stream
    elif key == "7":
        subscribe_audio_stream()
    # 2.4 Unsubscribe from audio stream
    elif key.upper() == "8":
        unsubscribe_audio_stream()
    # 3. Wakeup Status Functions
    # 3.1 Open wakeup status stream
    elif key.upper() == "Q":
        open_wakeup_status_stream()
    # 3.2 Close wakeup status stream
    elif key.upper() == "W":
        close_wakeup_status_stream()
    # 3.3 Subscribe to wakeup status stream
    elif key.upper() == "E":
        subscribe_wakeup_status()
    # 3.4 Unsubscribe from wakeup status stream
    elif key.upper() == "R":
        unsubscribe_wakeup_status()
    # 4. Print help information
    elif key.upper() == "?":
        print_help()
    else:
        logging.warning("Unknown key: %s", key)

        time.sleep(0.01) # Brief delay

    except KeyboardInterrupt:
        break
    except Exception as e:
        logging.error("Exception occurred while processing user input: %s", e)
except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close sensor controller
        audio_controller = robot.get_audio_controller()

```

(下页继续)

(续上页)

```

audio_controller.shutdown()
logging.info("Audio controller closed")

# Disconnect
robot.disconnect()
logging.info("Robot connection disconnected")

# Shutdown robot
robot.shutdown()
logging.info("Robot shutdown")

except Exception as e:
    logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())

```

24.3 运行说明

24.3.1 环境准备

```

# 设置环境变量
export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

```

24.3.2 运行示例

```

# C++
./audio_example

# Python
python3 audio_example.py

```

24.3.3 控制说明

- 1: 获取当前音量
- 2: 设置音量
- 3: 播放 TTS 语音
- 4: 停止 TTS 播放
- 5: 打开音频流

- 6: 关闭音频流
- 7: 订阅音频流数据
- 8: 取消订阅音频流数据
- Q: 打开语音唤醒状态流
- W: 关闭语音唤醒状态流
- E: 订阅语音唤醒状态
- R: 取消订阅语音唤醒状态
- ?: 打印帮助信息

24.3.4 停止程序

- 按 ESC 可以安全停止程序
- 程序会自动清理所有资源

CHAPTER 25

状态监控示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行初始化、连接机器人、机器人状态监控（故障、BMS）等基本操作。

25.1 C++

示例文件: monitor_example.cpp

```
#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <unistd.h>
#include <csignal>

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
    robot.Shutdown();
}
```

(下页继续)

(续上页)

```
// Exit process
exit(signum);
}

int main() {
    // Bind SIGINT (Ctrl+C)
    signal(SIGINT, signalHandler);

    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;

    std::string local_ip = "192.168.54.111";
    // Configure local IP address for direct network connection and initialize SDK
    if (!robot.Initialize(local_ip)) {
        std::cerr << "robot sdk initialize failed." << std::endl;
        robot.Shutdown();
        return -1;
    }

    // Connect to robot
    auto status = robot.Connect();
    if (status.code != ErrorCode::OK) {
        std::cerr << "connect robot failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }
    // Wait 5s
    usleep(5000000);

    auto& monitor = robot.GetStateMonitor();

    RobotState state;
    status = monitor.GetCurrentState(state);
    if (status.code != ErrorCode::OK) {
        std::cerr << "get current state failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        robot.Shutdown();
        return -1;
    }

    std::cout << "health: " << state.bms_data.battery_health
```

(下页继续)

(续上页)

```

    << ", percentage: " << state.bms_data.battery_percentage
    << ", state: " << std::to_string((int8_t)state.bms_data.battery_state)
    << ", power_supply_status: " << std::to_string((int8_t)state.bms_data.power_supply_
status)
    << std::endl;

auto& faults = state.faults;
for (auto& [code, msg] : faults) {
    std::cout << "code: " << std::to_string(code)
        << ", message: " << msg;
}

// Disconnect from robot
status = robot.Disconnect();
if (status.code != ErrorCode::OK) {
    std::cerr << "disconnect robot failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
robot.Shutdown();
return -1;
}

robot.Shutdown();

return 0;
}

```

25.2 Python

示例文件: monitor_example.py

```

#!/usr/bin/env python3

import sys
import time
import signal
import logging
from typing import Optional

import magicbot_z1_python as magicbot

# Configure logging format and level
logging.basicConfig(

```

(下页继续)

(续上页)

```

level=logging.INFO,    # Minimum log level
format"%(asctime)s [%(levelname)s] %(message)s",
datefmt="%Y-%m-%d %H:%M:%S",
)

# Global variables
robot: Optional[magicbot.MagicRobot] = None

def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global robot
    logging.info("Received interrupt signal (%s), exiting...", signum)
    if robot:
        robot.disconnect()
        logging.info("Robot disconnected")
        robot.shutdown()
        logging.info("Robot shutdown")
        exit(-1)

def main():
    """Main function"""
    global robot

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
            return -1

        # Connect to robot

```

(下页继续)

(续上页)

```
status = robot.connect()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to connect to robot, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Successfully connected to robot")

# Wait 5 seconds
logging.info("Waiting 5 seconds...")
time.sleep(5)

# Get state monitor
monitor = robot.get_state_monitor()

# Get current state
[status, state] = monitor.get_current_state()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to get current state, code: %s, message: %s",
        status.code,
        status.message,
    )
    return -1

# Print battery information
logging.info("Battery health: %s", state.bms_data.battery_health)
logging.info("Battery percentage: %s%%", state.bms_data.battery_percentage)
logging.info("Battery state: %s", state.bms_data.battery_state)
logging.info("Power supply status: %s", state.bms_data.power_supply_status)

# Print fault information
faults = state.faults
for fault in faults:
    logging.info(
        "Error code: %s, Error message: %s",
        fault.error_code,
        fault.error_message,
    )
```

(下页继续)

(续上页)

```

except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close state monitor
        monitor = robot.get_state_monitor()
        monitor.shutdown()

        # Disconnect
        robot.disconnect()
        logging.info("Robot connection disconnected")

        # Shutdown robot
        robot.shutdown()
        logging.info("Robot shutdown")

    except Exception as e:
        logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())

```

25.3 运行说明

25.3.1 环境准备

```

export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

```

25.3.2 运行示例

```

# C++
./monitor_example
# Python
python3 monitor_example.py

```

25.3.3 停止程序

- 按 `Ctrl+C` 可以安全停止程序
- 程序会自动清理所有资源

CHAPTER 26

SLAM 导航示例

该示例展示了如何使用 Magicbot-Gen1 SDK 进行 SLAM 建图、定位、导航等操作，包括地图管理、位姿获取等功能。

26.1 C++

示例文件：

- slam_example.cpp
- navigation_example.cpp

26.1.1 建图示例

```
#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <termios.h>
#include <unistd.h>
#include <csignal>
#include <ctime>
#include <fstream>
#include <iostream>
#include <string>
```

(下页继续)

(续上页)

```

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
bool running = true;
SlamMode current_slam_mode = SlamMode::IDLE;

void SignalHandler(int signum) {
    std::cout << "Received interrupt signal (" << signum << "), exiting..." << std::endl;
    running = false;
    robotShutdown();
    exit(signum);
}

void PrintHelp() {
    std::cout << "\n=====\n";
    std::cout << "SLAM and Navigation Function Demo Program" << std::endl;
    std::cout << "=====\n";
    std::cout << "preparation Functions:" << std::endl;
    std::cout << " q      Function q: Recovery stand" << std::endl;
    std::cout << " e      Function e: Balance stand" << std::endl;
    std::cout << " w      Function w: Move forward" << std::endl;
    std::cout << " a      Function a: Move left" << std::endl;
    std::cout << " s      Function s: Move backward" << std::endl;
    std::cout << " d      Function d: Move right" << std::endl;
    std::cout << " t      Function t: Turn left" << std::endl;
    std::cout << " g      Function g: Turn right" << std::endl;
    std::cout << " x      Function x: Stop" << std::endl;
    std::cout << "SLAM Functions:" << std::endl;
    std::cout << " 1      Function 1: Switch to mapping mode" << std::endl;
    std::cout << " 2      Function 2: Start mapping" << std::endl;
    std::cout << " 3      Function 3: Cancel mapping" << std::endl;
    std::cout << " 4      Function 4: Save map" << std::endl;
    std::cout << " 5      Function 5: Load map (input map name after pressing 5)" << std::endl;
    std::cout << " 6      Function 6: Delete map (input map name after pressing 6)" << std::endl;
    std::cout << std::endl;
    std::cout << " 7      Function 7: Get all map information and save map image as PGM file" << std::endl;
    std::cout << std::endl;
    std::cout << " 8      Function 8: Get map path (input map name)" << std::endl;
    std::cout << " 9      Function 9: Get SLAM mapping point cloud map" << std::endl;
    std::cout << "Close Functions:" << std::endl;
    std::cout << "  P      Function P: Close SLAM" << std::endl;
    std::cout << "\n ?      Function ?: Print help" << std::endl;
}

```

(下页继续)

(续上页)

```

    std::cout << "  ESC      Exit program" << std::endl;
    std::cout << "=====\\n"
                << std::endl;
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
}

std::string GetUserInput(const std::string& prompt) {
    std::cout << prompt;
    std::string input;
    std::getline(std::cin, input);
    return input;
}

void RecoveryStand() {
    std::cout << "== Executing Recovery Stand ==" << std::endl;
    auto& controller = robot.GetHighLevelMotionController();

    auto status = controller.SetGait(GaitMode::GAIT_RECOVERY_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set robot gait, code: " << status.code
              << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "Successfully executed recovery stand" << std::endl;
}

void BalanceStand() {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();

    // Set posture display gait
    auto status = controller.SetGait(GaitMode::GAIT_BALANCE_STAND, 10000);
}

```

(下页继续)

(续上页)

```

if (status.code != ErrorCode::OK) {
    std::cerr << "set robot gait failed"
    << ", code: " << status.code
    << ", message: " << status.message << std::endl;
    return;
}

std::cout << "robot gait set to GAIT_BALANCE_STAND successfully." << std::endl;
}

void SwitchToMappingMode() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::MAPPING, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to mapping mode, code: " << status.code
        << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::MAPPING;
    std::cout << "Successfully switched to mapping mode" << std::endl;
    std::cout << "Robot is now in mapping mode, ready to create new maps" << std::endl;
}

void StartMapping() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.StartMapping(10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to start mapping, code: " << status.code
        << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully started mapping" << std::endl;
}

void CancelMapping() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CancelMapping(10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to cancel mapping, code: " << status.code
}

```

(下页继续)

(续上页)

```

        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully cancelled mapping" << std::endl;
}

void SaveMap() {
    auto& controller = robot.GetSlamNavController();

    if (current_slam_mode != SlamMode::MAPPING) {
        std::cerr << "Warning: Currently not in mapping mode, may not be able to save map" <<
        std::endl;
    }

    // Generate map name with timestamp
    std::time_t now = std::time(nullptr);
    std::string map_name = "map_" + std::to_string(now);
    std::cout << "Saving map: " << map_name << std::endl;

    auto status = controller.SaveMap(map_name, 20000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to save map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully saved map: " << map_name << std::endl;
}

void LoadMap(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to load is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::cout << "Loading map: " << map_name << std::endl;
    auto status = controller.LoadMap(map_name, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to load map, code: " << status.code
            << ", message: " << status.message << std::endl;
    }
}

```

(下页继续)

(续上页)

```

    return;
}

std::cout << "Successfully loaded map: " << map_name << std::endl;
}

void DeleteMap(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to delete is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::cout << "Deleting map: " << map_name << std::endl;
    auto status = controller.DeleteMap(map_name, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to delete map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully deleted map: " << map_name << std::endl;
}

void SaveMapImageToFile(const MapInfo& map_info) {
    try {
        const auto& map_data = map_info.map_meta_data.map_image_data;
        uint32_t width = map_data.width;
        uint32_t height = map_data.height;
        uint32_t max_gray_value = map_data.max_gray_value;
        const auto& image_bytes = map_data.image;

        std::cout << "Saving map image: " << width << "x" << height
            << ", max_gray: " << max_gray_value << std::endl;

        // Check image data size
        if (image_bytes.size() != width * height) {
            std::cerr << "Image data size mismatch: expected " << (width * height)
                << ", got " << image_bytes.size() << std::endl;
            return;
        }
    }
}

```

(下页继续)

(续上页)

```

// Generate filename based on map name
std::string safe_filename = map_info.map_name;
// Remove invalid characters
safe_filename.erase(
    std::remove_if(safe_filename.begin(), safe_filename.end(),
        [] (char c) { return !std::isalnum(c) && c != '_' && c != '-'; }),
    safe_filename.end());
}

if (safe_filename.empty()) {
    safe_filename = "map_" + std::to_string(std::time(nullptr));
}

// Save as PGM format
std::string pgm_filename = "build/" + safe_filename + ".pgm";
std::ofstream pgm_file(pgm_filename, std::ios::binary);
if (!pgm_file.is_open()) {
    std::cerr << "Failed to open file for writing: " << pgm_filename << std::endl;
    return;
}

// Write PGM header
pgm_file << "P5\n"
    << width << " " << height << "\n"
    << max_gray_value << "\n";

// Write image data
pgm_file.write(reinterpret_cast<const char*>(image_bytes.data()), image_bytes.size());
pgm_file.close();

std::cout << "Map image saved successfully as PGM: " << pgm_filename << std::endl;

} catch (const std::exception& e) {
    std::cerr << "Exception occurred while saving map image: " << e.what() << std::endl;
}
}

void GetAllMapInfo() {
    auto& controller = robot.GetSlamNavController();

    AllMapInfo all_map_info;
    auto status = controller.GetAllMapInfo(all_map_info, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get map information, code: " << status.code
    }
}

```

(下页继续)

(续上页)

```

        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully retrieved map information" << std::endl;
std::cout << "Current map: " << all_map_info.current_map_name << std::endl;
std::cout << "Total maps: " << all_map_info.map_infos.size() << std::endl;

if (!all_map_info.map_infos.empty()) {
    std::cout << "Map details:" << std::endl;
    for (size_t i = 0; i < all_map_info.map_infos.size(); ++i) {
        const auto& map_info = all_map_info.map_infos[i];
        std::cout << "    Map " << (i + 1) << ":" << map_info.map_name << std::endl;
        std::cout << "        Origin: [" << map_info.map_meta_data.origin.position[0] << ", "
            << map_info.map_meta_data.origin.position[1] << ", "
            << map_info.map_meta_data.origin.position[2] << "]" << std::endl;
        std::cout << "        Orientation: [" << map_info.map_meta_data.origin.orientation[0] << ", "
            << map_info.map_meta_data.origin.orientation[1] << ", "
            << map_info.map_meta_data.origin.orientation[2] << "]" << std::endl;
        std::cout << "        Resolution: " << map_info.map_meta_data.resolution << " m/pixel" <<
        std::endl;
        std::cout << "        Size: " << map_info.map_meta_data.map_image_data.width << " x "
            << map_info.map_meta_data.map_image_data.height << std::endl;
        std::cout << "        Max gray value: " << map_info.map_meta_data.map_image_data.max_gray_
        value << std::endl;
        std::cout << "        Image type: " << map_info.map_meta_data.map_image_data.type <<
        std::endl;
    }
}

SaveMapImageToFile(map_info);
}

} else {
    std::cout << "No available maps" << std::endl;
}
}

void GetMapPath(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to get path is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();
}

```

(下页继续)

(续上页)

```

std::vector<std::string> map_path;
auto status = controller.GetMapPath(map_name, map_path, 10000);
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to get map path, code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}

if (map_path.empty()) {
    std::cerr << "No map path found" << std::endl;
    return;
}

for (const auto& path : map_path) {
    std::cout << "Map path: " << path << std::endl;
}
}

void GetPointCloudMap() {
    auto& controller = robot.GetSlamNavController();

    PointCloud2 point_cloud_map;
    auto status = controller.GetPointCloudMap(point_cloud_map, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get SLAM mapping point cloud map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully got SLAM mapping point cloud map" << std::endl;
    std::cout << "Point cloud map - Height: " << point_cloud_map.height
        << ", Width: " << point_cloud_map.width << std::endl;
    std::cout << "Point cloud map data size: " << point_cloud_map.data.size() << " bytes" <<
    std::endl;
}

void JoyStickCommand(float left_x_axis,
                     float left_y_axis,
                     float right_x_axis,
                     float right_y_axis) {
    // Get high-level motion controller
    auto& controller = robot.GetHighLevelMotionController();
}

```

(下页继续)

(续上页)

```

JoystickCommand joy_command;
joy_command.left_x_axis = left_x_axis;
joy_command.left_y_axis = left_y_axis;
joy_command.right_x_axis = right_x_axis;
joy_command.right_y_axis = right_y_axis;
auto status = controller.SendJoyStickCommand(joy_command);
if (status.code != ErrorCode::OK) {
    std::cerr << "execute robot trick failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    usleep(50000);
    return;
}
// Wait 50ms
usleep(50000);
}

void CloseSlam() {
auto& controller = robot.GetSlamNavController();

auto status = controller.ActivateSlamMode(SlamMode::IDLE, "", 10000);
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to close SLAM, code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}

current_slam_mode = SlamMode::IDLE;
std::cout << "Successfully closed SLAM system" << std::endl;
}

int main(int argc, char* argv[]) {
// Bind signal handler
signal(SIGINT, SignalHandler);

std::cout << "\n=====\n" << std::endl;
std::cout << "MagicBot Gen1 SDK SLAM Example" << std::endl;
std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;
std::cout << "=====\\n"
        << std::endl;

PrintHelp();
std::cout << "Press any key to continue (ESC to exit)..." << std::endl;
}

```

(下页继续)

(续上页)

```

// Configure local IP address for direct network connection and initialize SDK
std::string local_ip = "192.168.54.111";
if (!robot.Initialize(local_ip)) {
    std::cerr << "Failed to initialize robot SDK" << std::endl;
    robot.Shutdown();
    return -1;
}

// Connect to robot
auto status = robot.Connect();
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to connect to robot, code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}
std::cout << "Successfully connected to robot" << std::endl;

// Switch motion control controller to high-level controller
status = robot.SetMotionControlLevel(ControllerLevel::HighLevel);
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to switch robot motion control level, code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}

// Initialize SLAM navigation controller
auto& slam_nav_controller = robot.GetSlamNavController();
if (!slam_nav_controller.Initialize()) {
    std::cerr << "Failed to initialize SLAM navigation controller" << std::endl;
    robot.Disconnect();
    robot.Shutdown();
    return -1;
}
std::cout << "Successfully initialized SLAM navigation controller" << std::endl;

// Main loop
while (running) {
    int key = getch();

    if (key == 27) { // ESC key

```

(下页继续)

(续上页)

```

    std::cout << "ESC key pressed, exiting..." << std::endl;
    break;
}

switch (key) {
    // 1. Preparation Functions
    case 'q':
    case 'Q':
        RecoveryStand();
        break;
    case 'e':
    case 'E':
        BalanceStand();
        break;
    case 'w':
    case 'W': {
        JoyStickCommand(0.0, 1.0, 0.0, 0.0); // Forward
        break;
    }
    case 'a':
    case 'A': {
        JoyStickCommand(-1.0, 0.0, 0.0, 0.0); // Left
        break;
    }
    case 's':
    case 'S': {
        JoyStickCommand(0.0, -1.0, 0.0, 0.0); // Backward
        break;
    }
    case 'd':
    case 'D': {
        JoyStickCommand(1.0, 0.0, 0.0, 0.0); // Right
        break;
    }
    case 'x':
    case 'X': {
        JoyStickCommand(0.0, 0.0, 0.0, 0.0); // Stop
        break;
    }
    case 't':
    case 'T': {
        JoyStickCommand(0.0, 0.0, -1.0, 0.0); // Turn left
        break;
    }
}

```

(下页继续)

(续上页)

```

    }

case 'g':
case 'G': {
    JoyStickCommand(0.0, 0.0, 1.0, 0.0); // Turn right
    break;
}

// 2. SLAM Functions

case '1': {
    SwitchToMappingMode();
    break;
}

case '2': {
    StartMapping();
    break;
}

case '3': {
    CancelMapping();
    break;
}

case '4': {
    SaveMap();
    break;
}

case '5': {
    std::string map_name = GetUserInput("Enter map name to load: ");
    LoadMap(map_name);
    break;
}

case '6': {
    std::string map_name = GetUserInput("Enter map name to delete: ");
    DeleteMap(map_name);
    break;
}

case '7': {
    GetAllMapInfo();
    break;
}

case '8': {
    std::string map_name = GetUserInput("Enter map name to get path: ");
    GetMapPath(map_name);
    break;
}

case '9': {
    GetPointCloudMap();
    break;
}

case 'P':
case 'p':
    CloseSlam();
}

```

(下页继续)

(续上页)

```

        break;

    case '?':
        PrintHelp();
        break;

    default:
        std::cout << "Unknown key: " << static_cast<char>(key) << std::endl;
        break;
    }

    usleep(10000); // 10ms delay
}

// Cleanup resources
std::cout << "Clean up resources" << std::endl;

slam_nav_controller.Shutdown();
std::cout << "SLAM navigation controller closed" << std::endl;

robot.Disconnect();
std::cout << "Robot connection disconnected" << std::endl;

robot.Shutdown();
std::cout << "Robot shutdown" << std::endl;

return 0;
}

```

26.1.2 导航示例

```

#include "magic_robot.h"
#include "magic_sdk_version.h"

#include <termios.h>
#include <unistd.h>
#include <atomic>
#include <csignal>
#include <ctime>
#include <iostream>
#include <sstream>
#include <string>

using namespace magic::gen1;

```

(下页继续)

(续上页)

```

magic::gen1::MagicRobot robot;
bool running = true;
SlamMode current_slam_mode = SlamMode::IDLE;
NavMode current_nav_mode = NavMode::IDLE;
std::atomic<int> odometry_counter{0};

void SignalHandler(int signum) {
    std::cout << "Received interrupt signal (" << signum << "), exiting..." << std::endl;
    running = false;
    robotShutdown();
    exit(signum);
}

void PrintHelp() {
    std::cout << "\n======" << std::endl;
    std::cout << "SLAM and Navigation Function Demo Program" << std::endl;
    std::cout << "======" << std::endl;
    std::cout << "\npreparation Functions:" << std::endl;
    std::cout << " Q      Function Q: Recovery stand" << std::endl;
    std::cout << " W      Function W: Balance stand" << std::endl;
    std::cout << " E      Function E: Get map path (input map name)" << std::endl;
    std::cout << "\nLocalization Functions:" << std::endl;
    std::cout << " 1      Function 1: Switch to localization mode (input map path)" << std::endl;
    std::cout << " 2      Function 2: Initialize pose (input x y yaw)" << std::endl;
    std::cout << " 3      Function 3: Get current pose information" << std::endl;
    std::cout << "\nNavigation Functions:" << std::endl;
    std::cout << " 4      Function 4: Switch to navigation mode (input map path)" << std::endl;
    std::cout << " 5      Function 5: Set navigation target goal (input x y yaw)" << std::endl;
    std::cout << " 6      Function 6: Pause navigation" << std::endl;
    std::cout << " 7      Function 7: Resume navigation" << std::endl;
    std::cout << " 8      Function 8: Cancel navigation" << std::endl;
    std::cout << " 9      Function 9: Get navigation status" << std::endl;
    std::cout << "\nOdometry Functions:" << std::endl;
    std::cout << " Z      Function Z: Open odometry stream" << std::endl;
    std::cout << " X      Function X: Close odometry stream" << std::endl;
    std::cout << " C      Function C: Subscribe odometry stream" << std::endl;
    std::cout << " V      Function V: Unsubscribe odometry stream" << std::endl;
    std::cout << "\nClose Functions:" << std::endl;
    std::cout << " P      Function P: Close SLAM" << std::endl;
    std::cout << " L      Function L: Close navigation" << std::endl;
    std::cout << "\n ?      Function ?: Print help" << std::endl;
    std::cout << " ESC    Exit program" << std::endl;
}

```

(下页继续)

(续上页)

```

    std::cout << "=====\n"
    << std::endl;
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
}

std::string GetUserInput(const std::string& prompt) {
    std::cout << prompt;
    std::string input;
    std::getline(std::cin, input);
    return input;
}

void RecoveryStand() {
    std::cout << "==== Executing Recovery Stand ===" << std::endl;
    auto& controller = robot.GetHighLevelMotionController();

    auto status = controller.SetGait(GaitMode::GAIT_RECOVERY_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set robot gait, code: " << status.code
        << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "Successfully executed recovery stand" << std::endl;
}

void BalanceStand() {
    std::cout << "==== Executing Balance Stand ===" << std::endl;
    auto& controller = robot.GetHighLevelMotionController();

    auto status = controller.SetGait(GaitMode::GAIT_BALANCE_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set robot gait, code: " << status.code

```

(下页继续)

(续上页)

```

        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully executed balance stand" << std::endl;
}

void GetMapPath(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to get path is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::vector<std::string> map_path;
    auto status = controller.GetMapPath(map_name, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get map path, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    if (map_path.empty()) {
        std::cerr << "No map path found" << std::endl;
        return;
    }

    for (const auto& path : map_path) {
        std::cout << "Map path: " << path << std::endl;
    }
}

void SwitchToLocalizationMode(const std::string& map_path) {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::LOCALIZATION, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to localization mode, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::LOCALIZATION;
}

```

(下页继续)

(续上页)

```

    std::cout << "Successfully switched to localization mode" << std::endl;
    std::cout << "Robot is now in localization mode, ready to localize on existing maps" <<_
    std::endl;
}

void InitializePose(double x, double y, double yaw) {
    auto& controller = robot.GetSlamNavController();

    Pose3DEuler initial_pose;
    initial_pose.position = {x, y, 0.0};
    initial_pose.orientation = {0.0, 0.0, yaw};

    std::cout << "Initializing robot pose to: [" << x << ", " << y << ", " << yaw << "]" <<_
    std::endl;

    auto status = controller.InitPose(initial_pose, 15000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to initialize pose, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully initialized pose" << std::endl;
    std::cout << "Robot pose has been set to [" << x << ", " << y << ", " << yaw << "]" <<_
    std::endl;
}

void GetCurrentPoseInfo() {
    auto& controller = robot.GetSlamNavController();

    LocalizationInfo pose_info;
    auto status = controller.GetCurrentLocalizationInfo(pose_info);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get current pose information, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully retrieved current pose information" << std::endl;
    std::cout << "Localization status: " << (pose_info.is_localization ? "Localized" : "Not_
    localized") << std::endl;
    std::cout << "Position: [" << pose_info.pose.position[0] << ", "
        << pose_info.pose.position[1] << ", "

```

(下页继续)

(续上页)

```

    << pose_info.pose.position[2] << "]" << std::endl;
std::cout << "Orientation: [" << pose_info.pose.orientation[0] << ", "
    << pose_info.pose.orientation[1] << ", "
    << pose_info.pose.orientation[2] << "]" << std::endl;
}

void SwitchToNavigationMode(const std::string& map_path) {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateNavMode(NavMode::GRID_MAP, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to navigation mode, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_nav_mode = NavMode::GRID_MAP;
    std::cout << "Successfully switched to navigation mode" << std::endl;
}

void SetNavigationTarget(double x, double y, double yaw) {
    auto& controller = robot.GetSlamNavController();

    NavTarget target_goal;
    target_goal.id = 1;
    target_goal.frame_id = "map";
    target_goal.goal.position = {x, y, 0.0};
    target_goal.goal.orientation = {0.0, 0.0, yaw};

    auto status = controller.SetNavTarget(target_goal, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set navigation target, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully set navigation target: position=(" << x << ", " << y << ", 0.0), "
        << "orientation=(0.0, 0.0, " << yaw << ")" << std::endl;
}

void PauseNavigation() {
    auto& controller = robot.GetSlamNavController();
}

```

(下页继续)

(续上页)

```
auto status = controller.PauseNavTask();

if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to pause navigation, code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully paused navigation" << std::endl;
}

void ResumeNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ResumeNavTask();

    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to resume navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully resumed navigation" << std::endl;
}

void CancelNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CancelNavTask();

    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to cancel navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully cancelled navigation" << std::endl;
}

void GetNavigationStatus() {
    auto& controller = robot.GetSlamNavController();

    NavStatus nav_status;
    auto status = controller.GetNavTaskStatus(nav_status);

    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get navigation status, code: " << status.code
    }
}
```

(下页继续)

(续上页)

```

        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "==== Navigation Status ===" << std::endl;
std::cout << "Target ID: " << nav_status.id << std::endl;
std::cout << "Status: " << static_cast<int>(nav_status.status) << std::endl;
std::cout << "Error code: " << nav_status.error_code << std::endl;
std::cout << "Error description: " << nav_status.error_desc << std::endl;

// Status interpretation
std::string status_meaning;
switch (nav_status.status) {
    case NavStatusType::NONE:
        status_meaning = "No navigation target set";
        break;
    case NavStatusType::RUNNING:
        status_meaning = "Navigation is running";
        break;
    case NavStatusType::END_SUCCESS:
        status_meaning = "Navigation completed successfully";
        break;
    case NavStatusType::END_FAILED:
        status_meaning = "Navigation failed";
        break;
    case NavStatusType::PAUSE:
        status_meaning = "Navigation is paused";
        break;
    case NavStatusType::CONTINUE:
        status_meaning = "Navigation resumed from pause";
        break;
    case NavStatusType::CANCEL:
        status_meaning = "Navigation was cancelled";
        break;
    default:
        status_meaning = "Unknown status";
        break;
}

std::cout << "Status meaning: " << status_meaning << std::endl;
std::cout << "===== " << std::endl;
}

```

(下页继续)

(续上页)

```

void OpenOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.OpenOdometryStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to open odometry stream, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully opened odometry stream" << std::endl;
}

void CloseOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CloseOdometryStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close odometry stream, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully closed odometry stream" << std::endl;
}

void SubscribeOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto callback = [](const std::shared_ptr<Odometry> odometry) {
        if (odometry_counter % 30 == 0) {
            std::cout << "Odometry position: " << odometry->position[0] << ", "
                << odometry->position[1] << ", " << odometry->position[2] << std::endl;
            std::cout << "Odometry orientation: " << odometry->orientation[0] << ", "
                << odometry->orientation[1] << ", " << odometry->orientation[2] << ", "
                << odometry->orientation[3] << std::endl;
            std::cout << "Odometry linear velocity: " << odometry->linear_velocity[0] << ", "
                << odometry->linear_velocity[1] << ", " << odometry->linear_velocity[2] <<
            std::endl;
            std::cout << "Odometry angular velocity: " << odometry->angular_velocity[0] << ", "
                << odometry->angular_velocity[1] << ", " << odometry->angular_velocity[2] <<
            std::endl;
        }
    }
}

```

(下页继续)

(续上页)

```

    odometry_counter++;
}

controller.SubscribeOdometry(callback);
std::cout << "Successfully subscribed odometry stream" << std::endl;
}

void UnsubscribeOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    controller.UnsubscribeOdometry();
    std::cout << "Successfully unsubscribed odometry stream" << std::endl;
}

void CloseSlam() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::IDLE, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close SLAM, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::IDLE;
    std::cout << "Successfully closed SLAM system" << std::endl;
}

void CloseNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateNavMode(NavMode::IDLE, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_nav_mode = NavMode::IDLE;
    std::cout << "Successfully closed navigation system" << std::endl;
}

int main(int argc, char* argv[]) {

```

(下页继续)

(续上页)

```

// Bind signal handler
signal(SIGINT, SignalHandler);

std::cout << "\n=====\n" << std::endl;
std::cout << "MagicBot Gen1 SDK Navigation Example" << std::endl;
std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;
std::cout << "=====\\n"
      << std::endl;

PrintHelp();
std::cout << "Press any key to continue (ESC to exit)..." << std::endl;

// Configure local IP address for direct network connection and initialize SDK
std::string local_ip = "192.168.54.111";
if (!robot.Initialize(local_ip)) {
    std::cerr << "Failed to initialize robot SDK" << std::endl;
    robot.Shutdown();
    return -1;
}

// Connect to robot
auto status = robot.Connect();
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to connect to robot, code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}
std::cout << "Successfully connected to robot" << std::endl;

// Switch motion control controller to high-level controller
status = robot.SetMotionControlLevel(ControllerLevel::HighLevel);
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to switch robot motion control level, code: " << status.code
        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}
std::cout << "Successfully switched robot motion control level to high-level" << std::endl;

// Initialize SLAM navigation controller
auto& slam_nav_controller = robot.GetSlamNavController();
if (!slam_nav_controller.Initialize()) {

```

(下页继续)

(续上页)

```

    std::cerr << "Failed to initialize SLAM navigation controller" << std::endl;
    robot.Disconnect();
    robotShutdown();
    return -1;
}

std::cout << "Successfully initialized SLAM navigation controller" << std::endl;

// Main loop
while (running) {
    int key = getch();

    if (key == 27) { // ESC key
        std::cout << "ESC key pressed, exiting..." << std::endl;
        break;
    }

    switch (key) {
        // 1. preparation Functions
        case 'q':
        case 'Q':
            RecoveryStand();
            break;
        case 'w':
        case 'W':
            BalanceStand();
            break;

        case 'e':
        case 'E': {
            std::string map_name = GetUserInput("Enter map name to get path: ");
            GetMapPath(map_name);
            break;
        }
        // 2. Localization Functions
        case '1': {
            std::string map_path = GetUserInput("Enter map path for localization: ");
            SwitchToLocalizationMode(map_path);
            break;
        }
        case '2': {
            std::string input = GetUserInput("Enter pose (x y yaw): ");
            std::istringstream iss(input);
            double x = 0.0, y = 0.0, yaw = 0.0;
        }
    }
}

```

(下页继续)

(续上页)

```

iss >> x >> y >> yaw;
std::cout << "input pose, x: " << x << ", y: " << y << ", yaw: " << yaw << std::endl;
InitializePose(x, y, yaw);
break;
}

case '3':
GetCurrentPoseInfo();
break;

// 3. Navigation Functions

case '4': {
    std::string map_path = GetUserInput("Enter map path for navigation: ");
    SwitchToNavigationMode(map_path);
    break;
}

case '5': {
    std::string input = GetUserInput("Enter target (x y yaw): ");
    std::istringstream iss(input);
    double x = 0.0, y = 0.0, yaw = 0.0;
    iss >> x >> y >> yaw;
    SetNavigationTarget(x, y, yaw);
    break;
}

case '6':
PauseNavigation();
break;

case '7':
ResumeNavigation();
break;

case '8':
CancelNavigation();
break;

case '9':
GetNavigationStatus();
break;

// 4. Odometry Functions

case 'z':
case 'Z':
OpenOdometryStream();
break;

case 'x':
case 'X':
CloseOdometryStream();
}

```

(下页继续)

(续上页)

```
        break;
    case 'c':
    case 'C':
        SubscribeOdometryStream();
        break;
    case 'v':
    case 'V':
        UnsubscribeOdometryStream();
        break;
    // 5. Close Functions
    case 'l':
    case 'L':
        CloseNavigation();
        break;
    case 'p':
    case 'P':
        CloseSlam();
        break;

    case '?':
        PrintHelp();
        break;
    default:
        std::cout << "Unknown key: " << static_cast<char>(key) << std::endl;
        break;
    }

    usleep(10000); // 10ms delay
}

// Cleanup resources
std::cout << "Clean up resources" << std::endl;

slam_nav_controller.Shutdown();
std::cout << "SLAM navigation controller closed" << std::endl;

robot.Disconnect();
std::cout << "Robot connection disconnected" << std::endl;

robot.Shutdown();
std::cout << "Robot shutdown" << std::endl;

return 0;
}
```

26.2 Python

示例文件:

- slam_example.py
- navigation_example.py

26.2.1 建图示例

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import time
import signal
import threading
import logging
import termios
import tty
from typing import Optional
import numpy as np
import cv2
import random

import magicbot_gen1_python as magicbot

# Configure logging format and level
logging.basicConfig(
    level=logging.INFO,  # Minimum log level
    format="(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicbot.NavMode.IDLE


def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global running, robot
    logging.info("Received interrupt signal (%s), exiting...", signum)
```

(下页继续)

(续上页)

```

running = False
if robot:
    robot.shutdown()
    logging.info("Robot shutdown")
exit(-1)

def print_help():
    """Print help information"""
    logging.info("SLAM and Navigation Function Demo Program")
    logging.info("preparation Functions:")
    logging.info(" Q      Function Q: Recovery stand")
    logging.info(" E      Function E: Balance stand")
    logging.info(" W      Function W: Move forward")
    logging.info(" A      Function A: Move left")
    logging.info(" S      Function S: Move backward")
    logging.info(" D      Function D: Move right")
    logging.info(" T      Function T: Turn left")
    logging.info(" G      Function G: Turn right")
    logging.info(" X      Function X: Stop move")
    logging.info("")

    logging.info("SLAM Functions:")
    logging.info(" 1      Function 1: Switch to mapping mode")
    logging.info(" 2      Function 2: Start mapping")
    logging.info(" 3      Function 3: Cancel mapping")
    logging.info(" 4      Function 4: Save map")
    logging.info(" 5      Function 5: Load map")
    logging.info(" 6      Function 6: Delete map")
    logging.info("")

    " 7      Function 7: Get all map information and save map image as PGM file"
)
logging.info(" 8      Function 8: Get map path")
logging.info(" 9      Function 9: Get SLAM mapping point cloud map")
logging.info("")

logging.info("Close Functions:")
logging.info(" P      Function P: Close SLAM")
logging.info("")

logging.info(" ?      Function ?: Print help")
logging.info(" ESC     Exit program")

# ===== SLAM Functions =====

```

(下页继续)

(续上页)

```

def switch_to_mapping_mode():
    """Switch to mapping mode"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to mapping mode
        status = controller.activate_slam_mode(magicbot.SlamMode.MAPPING)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to switch to mapping mode, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    current_slam_mode = "MAPPING"
    logging.info("Successfully switched to mapping mode")
    logging.info("Robot is now in mapping mode, ready to create new maps")
except Exception as e:
    logging.error("Exception occurred while switching to mapping mode: %s", e)

def load_map(map_to_load):
    """Load map"""
    global robot
    try:
        if not map_to_load:
            logging.error("Map to load is not provided")
            return
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        logging.info("Loading map: %s", map_to_load)
        controller.load_map(map_to_load)
    except Exception as e:
        logging.error("Exception occurred while loading map: %s", e)
        return

    logging.info("Successfully loaded map: %s", map_to_load)

```

(下页继续)

(续上页)

```
def start_mapping():
    """Start mapping"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Start mapping
        status = controller.start_mapping()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to start mapping, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully started mapping")
except Exception as e:
    logging.error("Exception occurred while starting mapping: %s", e)

def cancel_mapping():
    """Cancel mapping"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Cancel mapping
        status = controller.cancel_mapping()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to cancel mapping, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully cancelled mapping")
except Exception as e:
    logging.error("Exception occurred while cancelling mapping: %s", e)
```

(下页继续)

(续上页)

```

def save_map():
    """Save map"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Check if in mapping mode
        if current_slam_mode != "MAPPING":
            logging.warning(
                "Warning: Currently not in mapping mode, may not be able to save map"
            )

        # Generate map name with timestamp
        map_name = f"map_{int(time.time())}"
        logging.info("Saving map: %s", map_name)

        # Save map
        status = controller.save_map(map_name, 20000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to save map, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully saved map: %s", map_name)
    except Exception as e:
        logging.error("Exception occurred while saving map: %s", e)

def delete_map(map_to_delete):
    """Delete map"""
    global robot
    try:
        if not map_to_delete:
            logging.error("Map to delete is not provided")
            return
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

```

(下页继续)

(续上页)

```

# Delete the first map as an example
logging.info("Deleting map: %s", map_to_delete)

# Delete map
status = controller.delete_map(map_to_delete)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to delete map, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully deleted map: %s", map_to_delete)
except Exception as e:
    logging.error("Exception occurred while deleting map: %s", e)


def get_all_map_info():
    """Get all map information"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get all map information
        status, all_map_info = controller.get_all_map_info()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get map information, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        logging.info("Successfully retrieved map information")
        logging.info("Current map: %s", all_map_info.current_map_name)
        logging.info("Total maps: %d", len(all_map_info.map_infos))

        if all_map_info.map_infos:
            logging.info("Map details:")
            for i, map_info in enumerate(all_map_info.map_infos):

```

(下页继续)

(续上页)

```

        logging.info(" Map %d: %s", i + 1, map_info.map_name)
        logging.info(
            "     Origin: [%f, %f, %f]",
            map_info.map_meta_data.origin.position[0],
            map_info.map_meta_data.origin.position[1],
            map_info.map_meta_data.origin.position[2],
        )
        logging.info(
            "     Orientation: [%f, %f, %f]",
            map_info.map_meta_data.orientation[0],
            map_info.map_meta_data.orientation[1],
            map_info.map_meta_data.orientation[2],
        )
        logging.info(
            "     Resolution: %f m/pixel", map_info.map_meta_data.resolution
        )
        logging.info(
            "     Size: %d x %d",
            map_info.map_meta_data.map_image_data.width,
            map_info.map_meta_data.map_image_data.height,
        )
        logging.info(
            "     Max gray value: %d",
            map_info.map_meta_data.map_image_data.max_gray_value,
        )
        logging.info(
            "     Image type: %s", map_info.map_meta_data.map_image_data.type
        )

    save_map_image_to_file(map_info)

else:
    logging.info("No available maps")
except Exception as e:
    logging.error("Exception occurred while getting map information: %s", e)

def save_map_image_to_file(map_info):
    """Save map image to current directory"""
    try:
        # Extract image data
        map_data = map_info.map_meta_data.map_image_data
        width = map_data.width

```

(下页继续)

(续上页)

```

height = map_data.height
max_gray_value = map_data.max_gray_value
image_bytes = map_data.image

logging.info(
    "Saving map image: %dx%d, max_gray: %d", width, height, max_gray_value
)

# Convert bytes to numpy array
if len(image_bytes) != width * height:
    logging.error(
        "Image data size mismatch: expected %d, got %d",
        width * height,
        len(image_bytes),
    )
    return

# Convert Uint8Vector to bytes for numpy processing
try:
    # Try to convert Uint8Vector to bytes
    if hasattr(image_bytes, "__iter__") and not isinstance(
        image_bytes, (str, bytes)
    ):
        # Convert Uint8Vector or similar iterable to bytes
        image_bytes_data = bytes(image_bytes)
    else:
        image_bytes_data = image_bytes
except Exception as e:
    logging.error("Failed to convert image data to bytes: %s", e)
    return

# Create numpy array from image data
image_array = np.frombuffer(image_bytes_data, dtype=np.uint8).reshape(
    (height, width)
)

# Generate filename based on map name
safe_filename = "".join(
    c for c in map_info.map_name if c.isalnum() or c in (" ", "-", "_")
).rstrip()
safe_filename = safe_filename.replace(" ", "_")
if not safe_filename:
    safe_filename = f"map_{int(time.time())}"

```

(下页继续)

(续上页)

```

# Save as PGM format using OpenCV
pgm_filename = f"build/{safe_filename}.pgm"
success = cv2.imwrite(pgm_filename, image_array)

if success:
    logging.info("Map image saved successfully as PGM: %s", pgm_filename)
else:
    logging.error("Failed to save map image as PGM: %s", pgm_filename)

except ImportError:
    logging.error("OpenCV not available, cannot save image")
    logging.info(
        "Image data: %dx%d pixels, %d bytes", width, height, len(image_bytes)
    )
except Exception as e:
    logging.error("Exception occurred while saving map image: %s", e)

def get_map_path(map_to_get_path):
    """Get map path"""
    global robot
    try:
        if not map_to_get_path:
            logging.error("Map to get path is not provided")
            return
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get map path
        status, map_path = controller.get_map_path(map_to_get_path)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get map path, code: %s, message: %s",
                status.code,
                status.message,
            )
            return
        if len(map_path) == 0:
            logging.error("No map path found")
            return

        for path in map_path:

```

(下页继续)

(续上页)

```

        logging.info("Map path: %s", path)
    except Exception as e:
        logging.error("Exception occurred while getting map path: %s", e)
        return

def get_point_cloud_map():
    """Get SLAM mapping point cloud map"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get SLAM mapping point cloud map
        status, point_cloud_map = controller.get_point_cloud_map()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get SLAM mapping point cloud map, code: %s, message: %s",
                status.code,
                status.message,
            )
        return
    logging.info("Successfully got SLAM mapping point cloud map")

    if point_cloud_map:
        logging.info(
            "Point cloud map - Height: %d, Width: %d, Data size: %d bytes",
            point_cloud_map.height,
            point_cloud_map.width,
            len(point_cloud_map.data),
        )

except Exception as e:
    logging.error(
        "Exception occurred while getting SLAM mapping point cloud map: %s",
        e,
    )

def close_slam():
    """Close SLAM system"""
    global robot, current_slam_mode
    try:

```

(下页继续)

(续上页)

```

# Get SLAM navigation controller
controller = robot.get_slam_nav_controller()

# Switch to idle mode to close SLAM
status = controller.activate_slam_mode(magicbot.SlamMode.IDLE)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to close SLAM, code: %s, message: %s",
        status.code,
        status.message,
    )
return

current_slam_mode = "IDLE"
logging.info("Successfully closed SLAM system")
except Exception as e:
    logging.error("Exception occurred while closing SLAM: %s", e)

def recovery_stand():
    """Recovery stand"""
    global robot
    try:
        logging.info("== Executing Recovery Stand ==")
        controller = robot.get_high_level_motion_controller()
        status = controller.set_gait(magicbot.GaitMode.GAIT_RECOVERY_STAND, 10000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set robot gait, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully executed recovery stand")
        return
    except Exception as e:
        logging.error("Exception occurred while executing recovery stand: %s", e)
        return

def balance_stand():
    """Balance stand"""

```

(下页继续)

(续上页)

```

global robot
try:
    logging.info("== Executing Balance Stand ==")

    # Get high-level motion controller
    controller = robot.get_high_level_motion_controller()

    # Set gait to balance stand
    status = controller.set_gait(magicbot.GaitMode.GAIT_BALANCE_STAND, 10000)
    if status.code != magicbot.ErrorCode.OK:
        logging.error(
            "Failed to set robot gait, code: %s, message: %s",
            status.code,
            status.message,
        )
        return False

    logging.info("Robot gait set to balance stand (supports movement)")
    return True

except Exception as e:
    logging.error("Exception occurred while executing balance stand: %s", e)
    return False


def joystick_command(left_x_axis, left_y_axis, right_x_axis, right_y_axis):
    """Send joystick control command"""
    global robot
    try:
        # Get high-level motion controller
        controller = robot.get_high_level_motion_controller()

        # Create joystick command
        joy_command = magicbot.JoystickCommand()
        joy_command.left_x_axis = left_x_axis
        joy_command.left_y_axis = left_y_axis
        joy_command.right_x_axis = right_x_axis
        joy_command.right_y_axis = right_y_axis

        # Send joystick command
        status = controller.send_joystick_command(joy_command)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(

```

(下页继续)

(续上页)

```

        "Failed to send joystick command, code: %s, message: %s",
        status.code,
        status.message,
    )
    time.sleep(0.05)  # Wait 50ms
    return False

    # Wait 50ms
    time.sleep(0.05)
    return True

except Exception as e:
    logging.error("Exception occurred while sending joystick command: %s", e)
    return False


def move_forward():
    """Move forward"""
    logging.info("==== Moving Forward ====")
    return joystick_command(0.0, 1.0, 0.0, 0.0)


def move_backward():
    """Move backward"""
    logging.info("==== Moving Backward ====")
    return joystick_command(0.0, -1.0, 0.0, 0.0)


def move_left():
    """Move left"""
    logging.info("==== Moving Left ====")
    return joystick_command(-1.0, 0.0, 0.0, 0.0)


def move_right():
    """Move right"""
    logging.info("==== Moving Right ====")
    return joystick_command(1.0, 0.0, 0.0, 0.0)


def turn_left():
    """Turn left"""
    logging.info("==== Turning Left ====")

```

(下页继续)

(续上页)

```

    return joystick_command(0.0, 0.0, -1.0, 0.0)

def turn_right():
    """Turn right"""
    logging.info("==== Turning Right ====")
    return joystick_command(0.0, 0.0, 1.0, 0.0)

def stop_move():
    """Stop Move"""
    logging.info("==== Stop Move ====")
    return joystick_command(0.0, 0.0, 0.0, 0.0)

# ===== Utility Functions =====

def get_user_input():
    """Get user input - Read a single line of data"""
    try:
        # Method 1: Read a line using input() (recommended)
        return input("Enter command: ").strip()
    except (EOFError, KeyboardInterrupt):
        return ""

# Get single character input (no echo)
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
        logging.info(f"Received character: {ch}")

        sys.stdout.write("\r")
        sys.stdout.flush()
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

```

(下页继续)

(续上页)

```
def main():
    """Main function"""
    global robot, running

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    print_help()
    logging.info("Press any key to continue (ESC to exit) ...")

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
            return -1

        # Connect to robot
        status = robot.connect()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to connect to robot, code: %s, message: %s",
                status.code,
                status.message,
            )
            robot.shutdown()
            return -1

        logging.info("Successfully connected to robot")

        # Switch motion control controller to high-level controller
        status = robot.set_motion_control_level(magicbot.ControllerLevel.HighLevel)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to switch robot motion control level, code: %s, message: %s",
                status.code,
                status.message,
            )

    except KeyboardInterrupt:
        logging.info("Keyboard interrupt received, exiting...")
```

(下页继续)

(续上页)

```

        )
    robot.shutdown()
    return -1

# Initialize SLAM navigation controller
slam_nav_controller = robot.get_slam_nav_controller()
if not slam_nav_controller.initialize():
    logging.error("Failed to initialize SLAM navigation controller")
    robot.disconnect()
    robot.shutdown()
    return -1

logging.info("Successfully initialized SLAM navigation controller")

# Main loop
while running:
    try:
        key = getch()
        if key == "\x1b": # ESC key
            break

        # 1. Preparation Functions
        # 1.1 Execute recovery stand first. Two mapping methods: 1) Suspended in air, ↵
can push robot for mapping in recovery_stand state; 2) Switch to balance_stand state for ↵
remote-controlled mapping
        if key.upper() == "Q":
            recovery_stand()
        elif key.upper() == "E":
            balance_stand()
        elif key.upper() == "W":
            move_forward()
        elif key.upper() == "A":
            move_left()
        elif key.upper() == "S":
            move_backward()
        elif key.upper() == "D":
            move_right()
        elif key.upper() == "T":
            turn_left()
        elif key.upper() == "G":
            turn_right()
        elif key.upper() == "X":
            stop_move()
    
```

(下页继续)

(续上页)

```

# 2. Mapping Mode
# 2.1 Switch to mapping mode
elif key == "1":
    switch_to_mapping_mode()

# 2.2 Start mapping
elif key == "2":
    start_mapping()

# 2.3 Cancel mapping
elif key == "3":
    cancel_mapping()

# 2.4 Save map
elif key == "4":
    save_map()

# 2.5 Load map
elif key == "5":
    str_input = get_user_input()
    # Split input parameters by space
    parts = str_input.strip().split()
    # Parse parameters
    map_to_load = parts[0] if parts else ""
    load_map(map_to_load)

# 2.6 Delete map
elif key == "6":
    str_input = get_user_input()
    # Split input parameters by space
    parts = str_input.strip().split()
    # Parse parameters
    map_to_delete = parts[0] if parts else ""
    delete_map(map_to_delete)

# 2.7 Get all map information
elif key == "7":
    get_all_map_info()

# 2.8 Get map path
elif key.upper() == "8":
    str_input = get_user_input()
    # Split input parameters by space
    parts = str_input.strip().split()
    # Parse parameters
    map_to_get_path = parts[0] if parts else ""
    get_map_path(map_to_get_path)

# 2.9 Get SLAM mapping point cloud map
elif key.upper() == "9":
    get_point_cloud_map()

```

(下页继续)

(续上页)

```
# 3. Close Functions
# 3.1 Close SLAM

elif key.upper() == "P":
    close_slam()

elif key.upper() == "?":
    print_help()

else:
    logging.warning("Unknown key: %s", key)

time.sleep(0.01) # Brief delay

except KeyboardInterrupt:
    break
except Exception as e:
    logging.error("Exception occurred while processing user input: %s", e)
except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close SLAM navigation controller
        slam_nav_controller = robot.get_slam_nav_controller()
        slam_nav_controller.shutdown()
        logging.info("SLAM navigation controller closed")

        # Disconnect
        robot.disconnect()
        logging.info("Robot connection disconnected")

        # Shutdown robot
        robot.shutdown()
        logging.info("Robot shutdown")

    except Exception as e:
        logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())
```

26.2.2 导航示例

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import time
import signal
import threading
import logging
from typing import Optional
import numpy as np
import cv2
import random

import magicbot_gen1_python as magicbot

# Configure logging format and level
logging.basicConfig(
    level=logging.INFO, # Minimum log level
    format="%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicbot.NavMode.IDLE
odometry_counter = 0


def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global running, robot
    logging.info("Received interrupt signal (%s), exiting...", signum)
    running = False
    if robot:
        robot.shutdown()
        logging.info("Robot shutdown")
    exit(-1)


def print_help():
    """Print help information"""

```

(下页继续)

(续上页)

```

logging.info("SLAM and Navigation Function Demo Program")
logging.info("")
logging.info("preparation Functions:")
logging.info(" Q      Function Q: Recovery stand")
logging.info(" W      Function W: Balance stand")
logging.info(" E      Function E: Get map path")
logging.info("")
logging.info("Localization Functions:")
logging.info(" 1      Function 1: Switch to localization mode")
logging.info(" 2      Function 2: Initialize pose")
logging.info(" 3      Function 3: Get current pose information")
logging.info("")
logging.info("Navigation Functions:")
logging.info(" 4      Function 4: Switch to navigation mode")
logging.info(" 5      Function 5: Set navigation target goal")
logging.info(" 6      Function 6: Pause navigation")
logging.info(" 7      Function 7: Resume navigation")
logging.info(" 8      Function 8: Cancel navigation")
logging.info(" 9      Function 9: Get navigation status")
logging.info("")
logging.info("Odometry Functions:")
logging.info(" Z      Function Z: Open odometry stream")
logging.info(" X      Function X: Close odometry stream")
logging.info(" C      Function C: Subscribe odometry stream")
logging.info(" V      Function V: Unsubscribe odometry stream")
logging.info("")
logging.info("Close Functions:")
logging.info(" P      Function P: Close SLAM")
logging.info(" L      Function L: Close navigation")
logging.info("")
logging.info(" ?      Function ?: Print help")
logging.info(" ESC    Exit program")

```

```

# ===== Navigation Functions =====
def get_map_path(map_to_get_path):
    """Get map path"""
    global robot
    try:
        if not map_to_get_path:
            logging.error("Map to get path is not provided")
            return
        # Get SLAM navigation controller

```

(下页继续)

(续上页)

```

controller = robot.get_slam_nav_controller()

# Get map path
status, map_path = controller.get_map_path(map_to_get_path)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to get map path, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

if len(map_path) == 0:
    logging.error("No map path found")
    return

for path in map_path:
    logging.info("Map path: %s", path)
except Exception as e:
    logging.error("Exception occurred while getting map path: %s", e)
    return


def switch_to_localization_mode(map_path):
    """Switch to localization mode"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to localization mode
        status = controller.activate_slam_mode(magicbot.SlamMode.LOCALIZATION, map_path)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to switch to localization mode, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        current_slam_mode = "LOCALIZATION"
        logging.info("Successfully switched to localization mode")
        logging.info(
            "Robot is now in localization mode, ready to localize on existing maps"
    
```

(下页继续)

(续上页)

```

        )
except Exception as e:
    logging.error("Exception occurred while switching to localization mode: %s", e)

def initialize_pose(x, y, yaw):
    """Initialize pose"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Create initial pose (set to origin)
        initial_pose = magicbot.Pose3DEuler()
        initial_pose.position = [x, y, 0.0]  # x, y, z
        initial_pose.orientation = [0.0, 0.0, yaw]  # roll, pitch, yaw

        logging.info("Initializing robot pose to origin...")

        # Initialize pose
        status = controller.init_pose(initial_pose)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to initialize pose, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully initialized pose")
        logging.info("Robot pose has been set to origin (0, 0, 0)")

    except Exception as e:
        logging.error("Exception occurred while initializing pose: %s", e)

def get_current_localization_info():
    """Get current pose information"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get current pose information

```

(下页继续)

(续上页)

```

status, pose_info = controller.get_current_localization_info()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to get current pose information, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully retrieved current pose information")
logging.info(
    "Localization status: %s",
    "Localized" if pose_info.is_localization else "Not localized",
)
logging.info(
    "Position: [%f, %f, %f]",
    pose_info.pose.position[0],
    pose_info.pose.position[1],
    pose_info.pose.position[2],
)
logging.info(
    "Orientation: [%f, %f, %f]",
    pose_info.pose.orientation[0],
    pose_info.pose.orientation[1],
    pose_info.pose.orientation[2],
)
except Exception as e:
    logging.error(
        "Exception occurred while getting current pose information: %s", e
    )

def switch_to_navigation_mode(map_path):
    """Switch to navigation mode"""
    global robot, current_nav_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to navigation mode
        status = controller.activate_nav_mode(magicbot.NavMode.GRID_MAP, map_path)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(

```

(下页继续)

(续上页)

```

        "Failed to switch to navigation mode, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

current_nav_mode = magicbot.NavMode.GRID_MAP
logging.info("Successfully switched to navigation mode")
except Exception as e:
    logging.error("Exception occurred while switching to navigation mode: %s", e)

def set_navigation_target(x, y, yaw):
    """Set navigation target goal"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Create target goal
        target_goal = magicbot.NavTarget()
        target_goal.id = 1
        target_goal.frame_id = "map"

        # Set target pose (example: move 2 meters forward)
        target_goal.goal.position = [x, y, 0.0]
        # Set target orientation (example: no rotation)
        target_goal.goal.orientation = [0.0, 0.0, yaw]

        # Set target goal
        status = controller.set_nav_target(target_goal)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set navigation target, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info(
            "Successfully set navigation target: position=(%.2f, %.2f, %.2f), orientation=(%.2f, %.2f, %.2f)",
            target_goal.goal.position[0],
            target_goal.goal.position[1],
            target_goal.goal.position[2],
            target_goal.goal.orientation[0],
            target_goal.goal.orientation[1],
            target_goal.goal.orientation[2]
        )
    except Exception as e:
        logging.error("Exception occurred while setting navigation target: %s", e)

```

(下页继续)

(续上页)

```

        target_goal.goal.position[1],
        target_goal.goal.position[2],
        target_goal.goal.orientation[0],
        target_goal.goal.orientation[1],
        target_goal.goal.orientation[2],
    )
except Exception as e:
    logging.error("Exception occurred while setting navigation target: %s", e)

def pause_navigation():
    """Pause navigation"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Pause navigation
        status = controller.pause_nav_task()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to pause navigation, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully paused navigation")
except Exception as e:
    logging.error("Exception occurred while pausing navigation: %s", e)

def resume_navigation():
    """Resume navigation"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Resume navigation
        status = controller.resume_nav_task()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(

```

(下页继续)

(续上页)

```

        "Failed to resume navigation, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

    logging.info("Successfully resumed navigation")
except Exception as e:
    logging.error("Exception occurred while resuming navigation: %s", e)

def cancel_navigation():
    """Cancel navigation"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Cancel navigation
        status = controller.cancel_nav_task()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to cancel navigation, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully cancelled navigation")
except Exception as e:
    logging.error("Exception occurred while cancelling navigation: %s", e)

def get_navigation_status():
    """Get current navigation status"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get navigation status
        status, nav_status = controller.get_nav_task_status()
        if status.code != magicbot.ErrorCode.OK:

```

(下页继续)

(续上页)

```

    logging.error(
        "Failed to get navigation status, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

# Display navigation status information
logging.info("==== Navigation Status ====")
logging.info("Target ID: %d", nav_status.id)
logging.info("Status: %s", nav_status.status)
logging.info("Error Code: %d", nav_status.error_code)
logging.info("Error Description: %s", nav_status.error_desc)

# Provide status interpretation
status_meaning = {
    magicbot.NavStatusType.NONE: "No navigation target set",
    magicbot.NavStatusType.RUNNING: "Navigation is running",
    magicbot.NavStatusType.END_SUCCESS: "Navigation completed successfully",
    magicbot.NavStatusType.END_FAILED: "Navigation failed",
    magicbot.NavStatusType.PAUSE: "Navigation is paused",
    magicbot.NavStatusType.CONTINUE: "Navigation resumed from pause",
    magicbot.NavStatusType.CANCEL: "Navigation was cancelled",
}
}

if nav_status.status in status_meaning:
    logging.info("Status meaning: %s", status_meaning[nav_status.status])
else:
    logging.warning("Unknown status value: %s", nav_status.status)

logging.info("=====")

except Exception as e:
    logging.error("Exception occurred while getting navigation status: %s", e)

def close_navigation():
    """Close navigation system"""
    global robot, current_nav_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

```

(下页继续)

(续上页)

```

# Switch to idle mode to close navigation
status = controller.activate_nav_mode(magicbot.NavMode.IDLE)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to close navigation, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

current_nav_mode = magicbot.NavMode.IDLE
logging.info("Successfully closed navigation system")
except Exception as e:
    logging.error("Exception occurred while closing navigation: %s", e)

def open_odometry_stream():
    """Open odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Open odometry stream
        status = controller.open_odometry_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to open odometry stream, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        logging.info("Successfully opened odometry stream")
    except Exception as e:
        logging.error("Exception occurred while opening odometry stream: %s", e)

def close_odometry_stream():
    """Close odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

```

(下页继续)

(续上页)

```

# Close odometry stream
status = controller.close_odometry_stream()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to close odometry stream, code: %s, message: %s",
        status.code,
        status.message,
    )
    return
logging.info("Successfully closed odometry stream")

except Exception as e:
    logging.error("Exception occurred while closing odometry stream: %s", e)

def subscribe_odometry_stream():
    """Subscribe odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        def callback(odometry: magicbot.Odometry):
            global odometry_counter
            if odometry_counter % 30 == 0:
                logging.info(
                    "Odometry position: %f, %f, %f",
                    odometry.position[0],
                    odometry.position[1],
                    odometry.position[2],
                )
                logging.info(
                    "Odometry orientation: %f, %f, %f, %f",
                    odometry.orientation[0],
                    odometry.orientation[1],
                    odometry.orientation[2],
                    odometry.orientation[3],
                )
                logging.info(
                    "Odometry linear velocity: %f, %f, %f",
                    odometry.linear_velocity[0],
                    odometry.linear_velocity[1],
                    odometry.linear_velocity[2],
                )
    
```

(下页继续)

(续上页)

```

        )
    logging.info(
        "Odometry angular velocity: %f, %f, %f",
        odometry.angular_velocity[0],
        odometry.angular_velocity[1],
        odometry.angular_velocity[2],
    )
    odometry_counter += 1

    # Subscribe odometry stream
    controller.subscribe_odometry(callback)
    logging.info("Successfully subscribed odometry stream")
except Exception as e:
    logging.error("Exception occurred while subscribing odometry stream: %s", e)

def unsubscribe_odometry():
    """Unsubscribe odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Unsubscribe odometry stream
        controller.unsubscribe_odometry()
        logging.info("Successfully unsubscribed odometry stream")
    except Exception as e:
        logging.error("Exception occurred while unsubscribing odometry stream: %s", e)

def close_slam():
    """Close SLAM system"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to idle mode to close SLAM
        status = controller.activate_slam_mode(magicbot.SlamMode.IDLE)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close SLAM, code: %s, message: %s",
                status.code,

```

(下页继续)

(续上页)

```

        status.message,
    )
    return

current_slam_mode = "IDLE"
logging.info("Successfully closed SLAM system")
except Exception as e:
    logging.error("Exception occurred while closing SLAM: %s", e)

def recovery_stand():
    """Recovery stand"""
    global robot
    try:
        logging.info("== Executing Recovery Stand ==")
        controller = robot.get_high_level_motion_controller()
        status = controller.set_gait(magicbot.GaitMode.GAIT_RECOVERY_STAND, 10000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set robot gait, code: %s, message: %s",
                status.code,
                status.message,
            )
    return
    logging.info("Successfully executed recovery stand")
    return
except Exception as e:
    logging.error("Exception occurred while executing recovery stand: %s", e)
    return

def balance_stand():
    """Balance stand"""
    global robot
    try:
        logging.info("== Executing Balance Stand ==")
        controller = robot.get_high_level_motion_controller()
        status = controller.set_gait(magicbot.GaitMode.GAIT_BALANCE_STAND, 10000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set robot gait, code: %s, message: %s",
                status.code,
                status.message,
            )
    
```

(下页继续)

(续上页)

```

        )
    return
logging.info("Successfully executed balance stand")
return
except Exception as e:
    logging.error("Exception occurred while executing balance stand: %s", e)
    return

# ===== Utility Functions =====

def get_user_input():
    """Get user input - Read a single line of data"""
    try:
        # Method 1: Read a line using input() (recommended)
        return input("Enter command: ").strip()
    except (EOFError, KeyboardInterrupt):
        return ""

def main():
    """Main function"""
    global robot, running

    # Bind signal handler
    signal.signal(signal.SIGINT, signal_handler)

    logging.info("Robot model: %s", magicbot.get_robot_model())

    # Create robot instance
    robot = magicbot.MagicRobot()

    print_help()
    logging.info("Press any key to continue (ESC to exit)...")

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"
        if not robot.initialize(local_ip):
            logging.error("Failed to initialize robot SDK")
            robot.shutdown()
        return -1
    
```

(下页继续)

(续上页)

```
# Connect to robot
status = robot.connect()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to connect to robot, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Successfully connected to robot")

# Switch motion control controller to high-level controller
status = robot.set_motion_control_level(magicbot.ControllerLevel.HighLevel)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to switch robot motion control level, code: %s, message: %s",
        status.code,
        status.message,
    )
    robot.shutdown()
    return -1

logging.info("Successfully switched robot motion control level to high-level")

# Initialize SLAM navigation controller
slam_nav_controller = robot.get_slam_nav_controller()
if not slam_nav_controller.initialize():
    logging.error("Failed to initialize SLAM navigation controller")
    robot.disconnect()
    robot.shutdown()
    return -1

logging.info("Successfully initialized SLAM navigation controller")

# Main loop
while running:
    try:
        str_input = get_user_input()

        # Split input parameters by space
```

(下页继续)

(续上页)

```

parts = str_input.strip().split()

if not parts:
    time.sleep(0.01)  # Brief delay
    continue

# Parse parameters
key = parts[0]
args = parts[1:] if len(parts) > 1 else []
if key == "\x1b":  # ESC key
    break

# 1. Preparation
# 1.1 Execute recovery stand first
if key.upper() == "Q":
    recovery_stand()

# 1.2 Switch to balance stand, allowing robot to transition to walking gait
elif key.upper() == "W":
    balance_stand()

# 1.3 Get current map absolute path
elif key.upper() == "E":
    map_to_get_path = args[0] if args else ""
    get_map_path(map_to_get_path)

# 2. Enable Localization Mode and Initialize Pose
# 2.2 Based on map absolute path, enable localization mode
elif key == "1":
    map_path = args[0] if args else ""
    switch_to_localization_mode(map_path)

# 2.3 Based on current map, initialize pose
elif key == "2":
    x = float(args[0]) if args else 0.0
    y = float(args[1]) if args else 0.0
    yaw = float(args[2]) if args else 0.0
    logging.info("input pose, x: %f, y: %f, yaw: %f", x, y, yaw)
    initialize_pose(x, y, yaw)

# 2.4 Get current initialized pose status, check if localization succeeded
elif key == "3":
    get_current_localization_info()

# 3. Start Navigation
# 3.1 Based on map absolute path, enable navigation mode
elif key.upper() == "4":
    map_path = args[0] if args else ""
    switch_to_navigation_mode(map_path)

```

(下页继续)

(续上页)

```

# 3.2 Input target point, start navigation task
# Due to the lidar being installed with a -1.57rad offset relative to the robot
→'s front,
    # the desired yaw orientation needs to be offset by -1.57rad, otherwise the
→robot's pose initialization may fail
    # Please input yaw orientation, needs to be offset by -1.57rad
elif key.upper() == "5":
    x = float(args[0]) if args else 0.0
    y = float(args[1]) if args else 0.0
    yaw = float(args[2]) if args else 0.0
    logging.info("input target, x: %f, y: %f, yaw: %f", x, y, yaw)
    set_navigation_target(x, y, yaw)

# 3.3 Pause navigation task
elif key.upper() == "6":
    pause_navigation()

# 3.4 Resume navigation task
elif key.upper() == "7":
    resume_navigation()

# 3.5 Cancel navigation task
elif key.upper() == "8":
    cancel_navigation()

# 3.6 Get navigation task status
elif key.upper() == "9":
    get_navigation_status()

# 4. Subscribe to Odometry Data
# 4.1 Open odometry stream
elif key.upper() == "Z":
    open_odometry_stream()

# 4.2 Close odometry stream
elif key.upper() == "X":
    close_odometry_stream()

# 4.3 Subscribe to odometry stream
elif key.upper() == "C":
    subscribe_odometry_stream()

# 4.4 Unsubscribe from odometry stream
elif key.upper() == "V":
    unsubscribe_odometry_stream()

# 5. Close SLAM and Navigation
# 5.1 Close SLAM
elif key.upper() == "P":
    close_slam()

# 5.2 Close navigation
elif key.upper() == "L":

```

(下页继续)

(续上页)

```
        close_navigation()

    elif key.upper() == "?":
        print_help()

    else:
        logging.warning("Unknown key: %s", key)

        time.sleep(0.01) # Brief delay

except KeyboardInterrupt:
    break
except Exception as e:
    logging.error("Exception occurred while processing user input: %s", e)
except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close SLAM navigation controller
        slam_nav_controller = robot.get_slam_nav_controller()
        slam_nav_controller.shutdown()
        logging.info("SLAM navigation controller closed")

        # Disconnect
        robot.disconnect()
        logging.info("Robot connection disconnected")

        # Shutdown robot
        robot.shutdown()
        logging.info("Robot shutdown")

    except Exception as e:
        logging.error("Exception occurred while cleaning up resources: %s", e)

if __name__ == "__main__":
    sys.exit(main())
```

26.3 运行说明

26.3.1 环境准备

```
# 设置环境变量
export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH
```

26.3.2 运行示例

```
# C++
./slam_navigation_example
# Python
python3 slam_navigation_example.py
```

26.3.3 控制说明

建图

- 准备工作:
 - 按键 Q - 恢复站立
 - 按键 W/A/D/D/X - 前进/左移/后退/右移/停止移动
 - 按键 T/G - 左转/右转
- SLAM 功能:
 - 按键 1 - 切换到建图模式
 - 按键 2 - 开始建图
 - 按键 3 - 取消建图
 - 按键 4 - 保存地图
 - 按键 5 - 加载地图
 - 按键 6 - 删除地图
 - 按键 7 - 获取所有地图信息并保存为 PGM 文件
 - 按键 8 - 获取地图路径
 - 按键 9 - 获取 SLAM 建图点云地图
- 关闭功能:
 - 按键 P - 关闭 SLAM
- 系统命令:

- 按键 ? - 显示帮助信息

导航

- 准备功能:
 - 按键 Q - 恢复站立
 - 按键 W - 平衡站立
 - 按键 E - 获取地图路径
- 定位功能:
 - 按键 1 - 切换到定位模式
 - 按键 2 - 初始化位姿
 - 按键 3 - 获取当前位置信息
- 导航功能:
 - 按键 4 - 切换到导航模式
 - 按键 5 - 设置导航目标点
 - 按键 6 - 暂停导航
 - 按键 7 - 恢复导航
 - 按键 8 - 取消导航
 - 按键 9 - 获取导航状态
- 里程计功能:
 - 按键 Z - 开启里程计流
 - 按键 X - 关闭里程计流
 - 按键 C - 订阅里程计流
 - 按键 V - 取消订阅里程计流
- 关闭功能:
 - 按键 P - 关闭 SLAM
 - 按键 L - 关闭导航
- 系统命令:
 - 按键 ? - 显示帮助信息

26.3.4 停止程序

- 按 ESC 可以安全停止程序
- 程序会自动清理所有资源

26.3.5 注意事项

- 建图和导航示例中，需要提前将机器人切换到高层运动控制模式（默认控制模式是高层运动控制）；
- 建图示例中，至少需要切换到 Recovery_stand 才可以进行建图，可以在 Recovery_stand 模式下推着机器人进行建图，也可以在 Balance_stand 模式下遥控控制机器人运动进行建图（本示例是展示机器人 Recovery_stand 模式下推着机器人进行建图）；
- 建图示例中，可以通过 GetAllMapInfo 获取当前机器人上存储地图的信息（包括二维地图 pgm, 地图大小，分辨率等信息）；
- 建图示例中，可以通过 SaveMap/DeleteMap 进行地图管理，机器人上存储地图尽量不要超过 3 张地图；
- 导航示例中，需要提前将机器人切换到 Balance_stand 模式；
- 导航示例中，导航目标点最好是基于 GetCurrentLocalizationInfo 获取的当前定位状态和当前定位位姿进行设置；
- 导航示例中，进行导航任务前需要正常切换到定位模式 (ActivateSlamMode (SlamMode::LOCALIZATION, map_path)) 和导航模式 (ActivateNavMode (NavMode::GRID_MAP, map_path))；
- 导航示例中，定位模式切换和导航模式切换都需要传入地图的绝对路径，可以通过 GetMapPath 获取指定地图的路径信息；
- 建图或者导航结束后，需要切换将 SLAM 切换到 IDLE 模式 (ActivateSlamMode (SlamMode::IDLE))，导航也切换到 IDLE 模式 (ActivateNavMode (NavMode::IDLE))

CHAPTER 27

FAQ

当出现 SDK 不可用时，请首先检查 SDK 版本是否符合要求，SDK 版本 tag 与机器人本体版本号对应记录，参考: [ChangeLog](#)

27.1 开发准备环境

操作系统: Ubuntu22.04，具体开发环境可以参考:《快速开始》

27.2 支持无线开发吗？

不支持，MagicBot-Gen1 目前只支持有线连接进行开发

27.3 目前底层控制接口通讯频率是多少？

底层关节状态上报通信频率默认是 500Hz，底层关节指令下发 Publish 频率由用户控制，建议 500HZ

27.4 SDK 无法订阅和发布话题数据？

- 检查相关话题数据流是否打开；
部分接口需要调用对应接口进行打开，比如 openLidar 打开雷达数据；
- udp 多播配置：

假设 SDK 二次开发 PC 与机器人链接的网口为 `enp0s31f6`, 需要进行如下配置以便 SDK 与机器人的底层通信:

```
sudo ifconfig enp0s31f6 multicast  
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev enp0s31f6
```

- SDK 版本与本体软件版本不对齐:

SDK 版本 tag 与机器人本体版本号对应记录, 参考: [ChangeLog](#)

- 清理残留 SDK 配置文件:

SDK 默认内置配置生成在`/tmp/magicbot_gen1_mjrrt.yaml` 文件, 正常开机重启会自动清理掉; 但是如果本次开机, 先使用旧版本的 SDK, 在`/tmp` 目录下生成旧版配置文件, 后使用新版本 SDK 使用, 默认会读取旧版本的配置 (SDK 内部如果检查 `magicbot_gen1_mjrrt.yaml` 文件存在, 则不重新创建, 为的是方便调试)。从而导致话题订阅与发布存在问题。

最后, 需要注意的是, 在使用 `Subscribe` 等订阅类 SDK 接口时, 请避免在回调函数中执行阻塞式的数据处理操作。否则, 可能导致消息堆积、处理延迟, 甚至引发不可预期的错误。

27.5 SDK 订阅话题数据频率不稳定 ?

检查 SDK 运行主机 socket 链接的接收缓存系统配置`/etc/sysctl.conf`, `sysctl -p` 或者重启主机立即生效:

```
net.core.rmem_max=20971520  
net.core.rmem_default=20971520  
net.core.wmem_max=20971520  
net.core.wmem_default=20971520
```

27.6 SDK 接口调用报错: Deadline Exceeded

SDK 接口内部 RPC 调用默认超时时间是 5s, 部分步态或者特技的执行接口的执行时间可能不止 5s, 如果出现该问题, 可以通过设置对应接口的超时时间参数来规避该问题。

27.7 SDK 内部配置和日志如何查看 ?

SDK 程序执行时, 默认会在`/tmp` 目录下生成`/tmp/magicbot_gen1_mjrrt.yaml` 配置文件, 在`/tmp/logs` 目录下生成 `magicbot_gen1_sdk.log` 日志文件, SDK 内部的错误信息可以通过该日志文件查看;

27.8 视频推流

三目相机视频流获取方式：

```
# 无线，前提是需要连接到机器人热点
ffplay rtsp://192.168.12.1:8082/
# 有线
ffplay rtsp://192.168.54.119:8082/
```

演示上可以使用 ffplay 测试，如果想要获取每帧图像，需要自行编码

27.9 SLAM 重定位失败？

SDK 方式 SLAM 导航不同于 App 端方便可视化进行定位，需要修改 SLAM 配置，固定地图方向，方便 SLAM 定位：

```
# 登录机器人本体
ssh eame@192.168.54.119
# 修改如下配置
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# 首次更改yaml文件后需要重启SLAM模块生效，重启指令为：
sudo systemctl restart slam.service
```

27.10 SLAM 导航不执行？

1. SLAM 导航执行前提条件一，需要进入 SLAM 定位模式和导航模式：

- ActivateSlamMode(SlamMode::LOCALIZATION, map_path)
- ActivateNavMode(NavMode::GRID_MAP, map_path);

其中 map_path 为当前地图的绝对路径，可以通过 GetMapPath 进行获取

2. SLAM 导航执行前提条件二，重定位成功：

需要通过 InitPose 进行重定位，并调取 GetCurrentLocalizationInfo 获取当前重定位状态，如果 is_localization=true，则代表重定位成功；否则，重定位失败，需要检查地图和重定位位姿，重新 InitPose 进行定位；

3. SLAM 导航目标点设置：

因为 InitPose 设置的重定位位姿是预估的，所以目标位置点建议基于 GetCurrentLocalizationInfo 获得的当前位置位姿进行设置目标点位姿；

27.11 运行报错: Invalid IP address

当前机器人不支持 APP 与 SDK 同时控制机器人，也不支持多个 SDK 客户端同时控制机器人呢，当出现 Invalid IP address 请求错误时，请检查是否有其他 APP 或 SDK 客户端连接；

27.12 日志报错打印: Call of pthread_setschedparam with insufficient privileges!

主要原因是 SDK 进行在普通用户执行，该报错根本上不影响程序运行，为解决该报错，需要在普通用户下配置系统文件/etc/security/limits.conf，确保进程拥有线程优先级设置权限：

```
* - rtprio 98
```

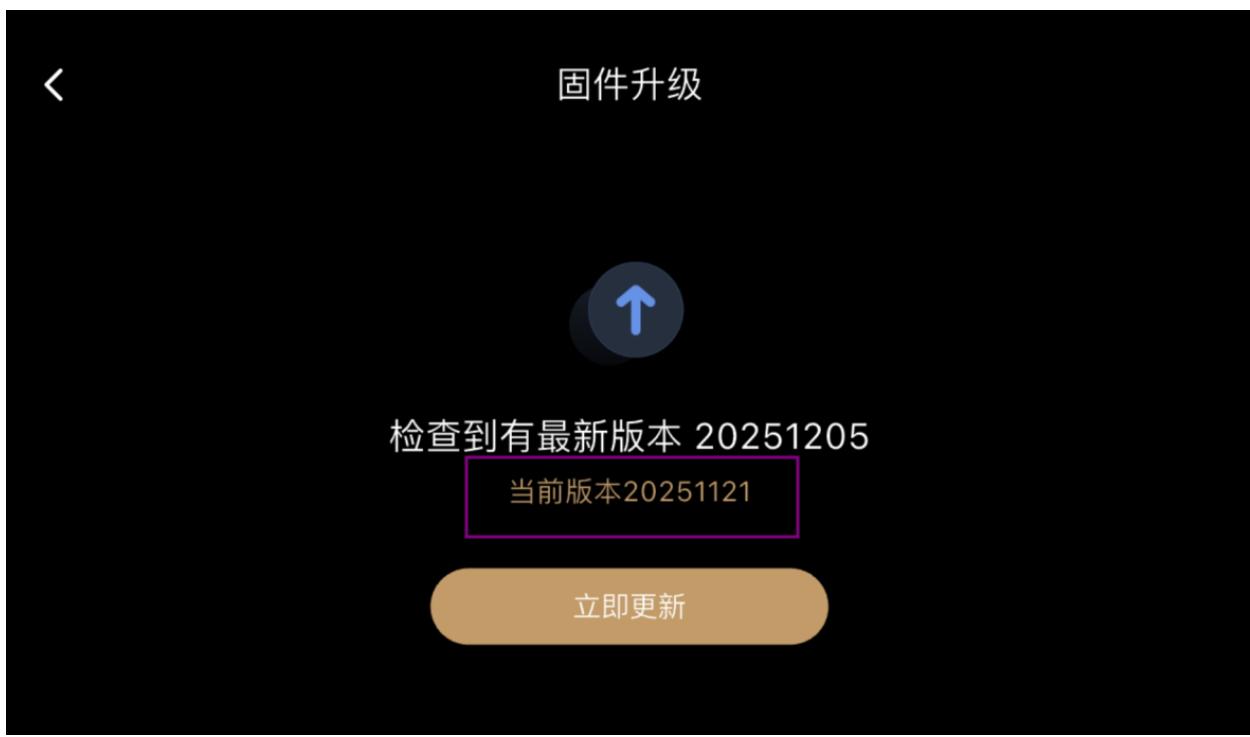
27.13 connect 返回报错: Failed to connect to robot, code: ErrorCode.SERVICE_ERROR, message: failed to connect to all addresses; last error: UNKNOWN: ipv4:192.168.54.119:50051: Failed to connect to remote host: Connection refused

1. 无法 ping 通 192.168.54.119：
 - 网线连接异常，检查硬件连接；
 - 算力包 IP 配置不是 192.168.54.xx 网段；
2. 本体网关服务异常：
 - 尝试重启，恢复本体软件正常运行；

27.14 如何查看机器本体软件版本？

方法一：通过 APP 查看机器人 OTA 软件版本：





方法二：登录到机器本体内查看 OTA 软件版本：

```
# 登录到机器本体  
ssh eame@192.168.54.119  
# 查看机器上次OTA软件版本
```

(下页继续)

(续上页)

```
cat /home/eame/devID/local_version.json
```

27.15 执行特技报错: Failed to execute robot trick, code:

ErrorCode.SERVICE_ERROR, message: 优先级过低

原因一: 调用的特技指令 ID 不在 TrickAction 枚举范围内;

原因二: 机器需要处于落地状态, 并且切换到 balance_stand 步态下;

注意: 人形机器人要 recovery_stand 状态下落地, 落地之后才能切 balance_stand, 之后才能做特技等其他动作

27.16 执行特技报错: Failed to execute robot trick, code:

ErrorCode.SERVICE_ERROR, message: 危险动作

可能原因是, 当前机器人的初始位置与机器人特技的起始位置相差较大导致;

解决方法:

- 重新按顺序执行 recovery_stand 和 balance_stand, 尝试恢复;

27.17 执行特技报错: Failed to execute robot trick, code:

ErrorCode.SERVICE_ERROR, message: 调用 ROS2 服务失败

可能原因是, 当前机器人本体内部软件服务异常导致;

解决方法:

- 重启机器尝试恢复, 如果重启不能恢复, 麻烦联系售后;