Magicdog SDK Development Documentation

发布 *1.2.1-doc1*

MagicLab

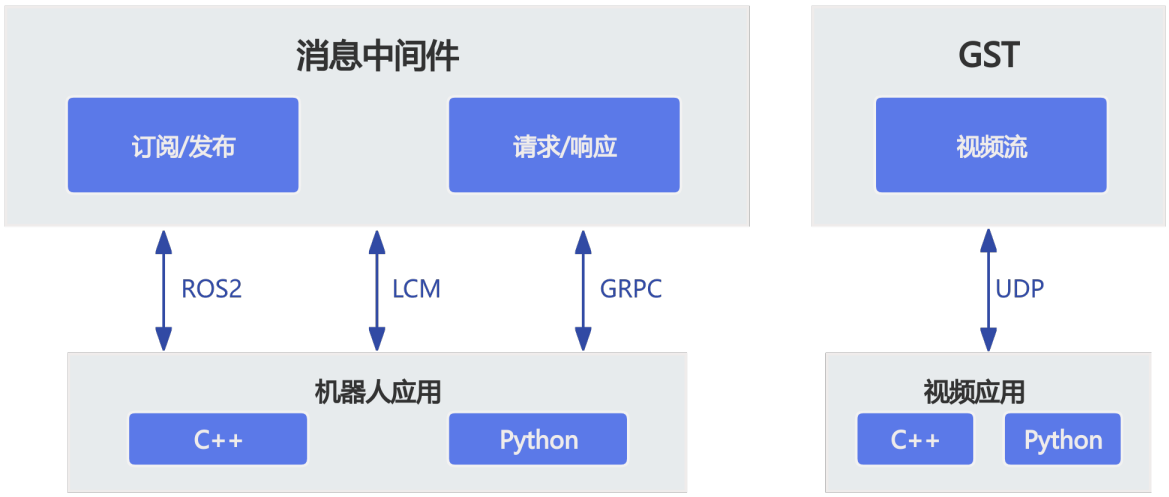
2025 年 12 月 11 日

SDK 介绍:

CHAPTER 1

SDK 概述

1.1 SDK 通信接口介绍



MagicDog 采用 LCM/GRPC/ROS2 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，主要用于中高频率或持续性的数据交互。

- RPC（请求/响应）：采用问答模式，通过请求实现数据获取或操作控制，用于低频率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 调用封装为函数调用，方便用户使用。

1.2 获取 SDK

magicdog-sdk 是魔法原子新一代机器人开发 SDK。该 SDK 将高层运动控制、底层电机控制、语音控制等接口进行了完整封装，并提供了相应的函数式接口。您可以参考我们提供的 SDK 教程，学习机器人控制方法，完成 MagicDog 的二次开发。

1.2.1 SDK 下载地址：

magicdog-sdk

SDK 版本与机器本体软件版本对齐记录，请参考：ChangeLog

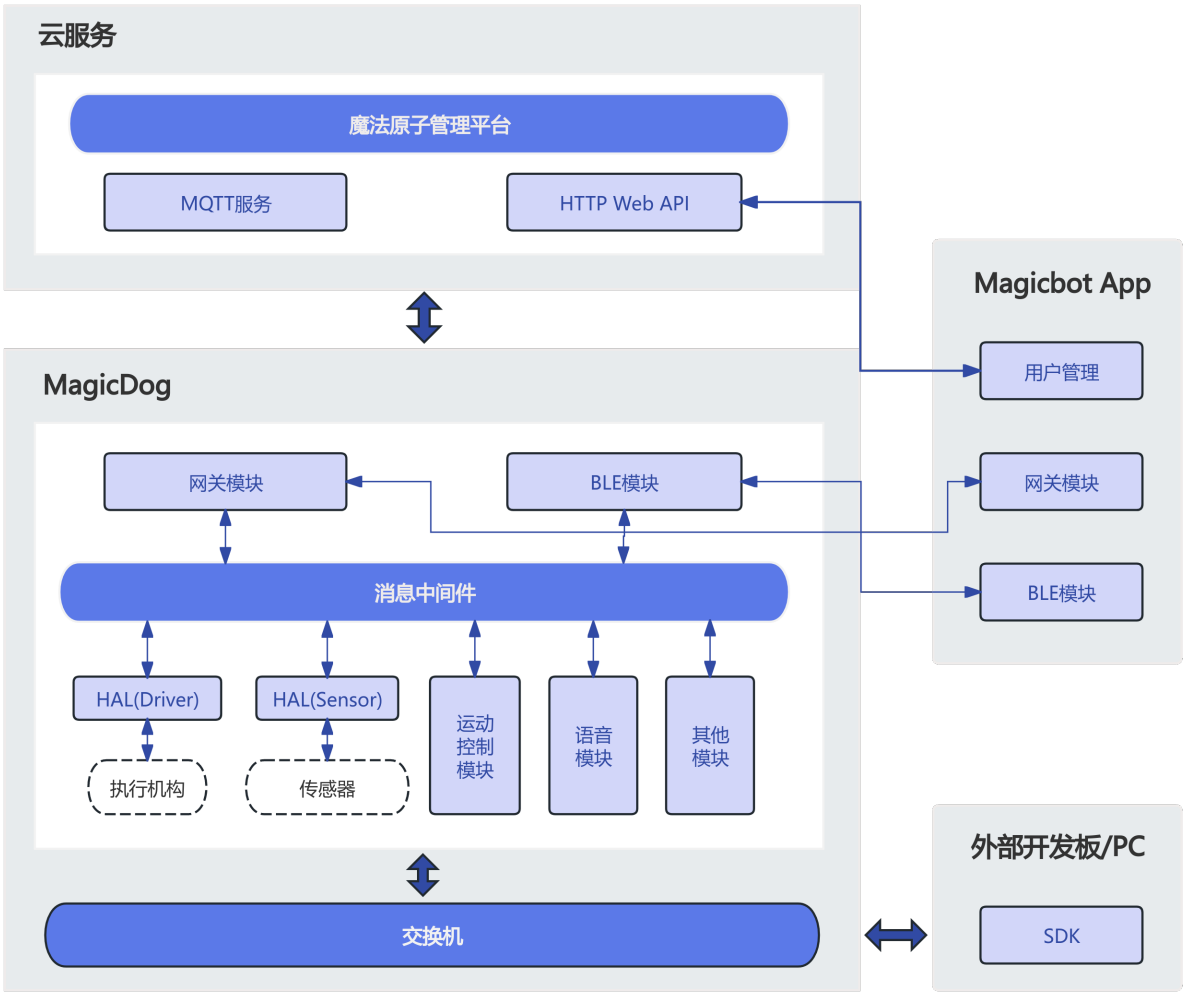
1.2.2 URDF/MJCF 下载地址:

URDF/MJCF

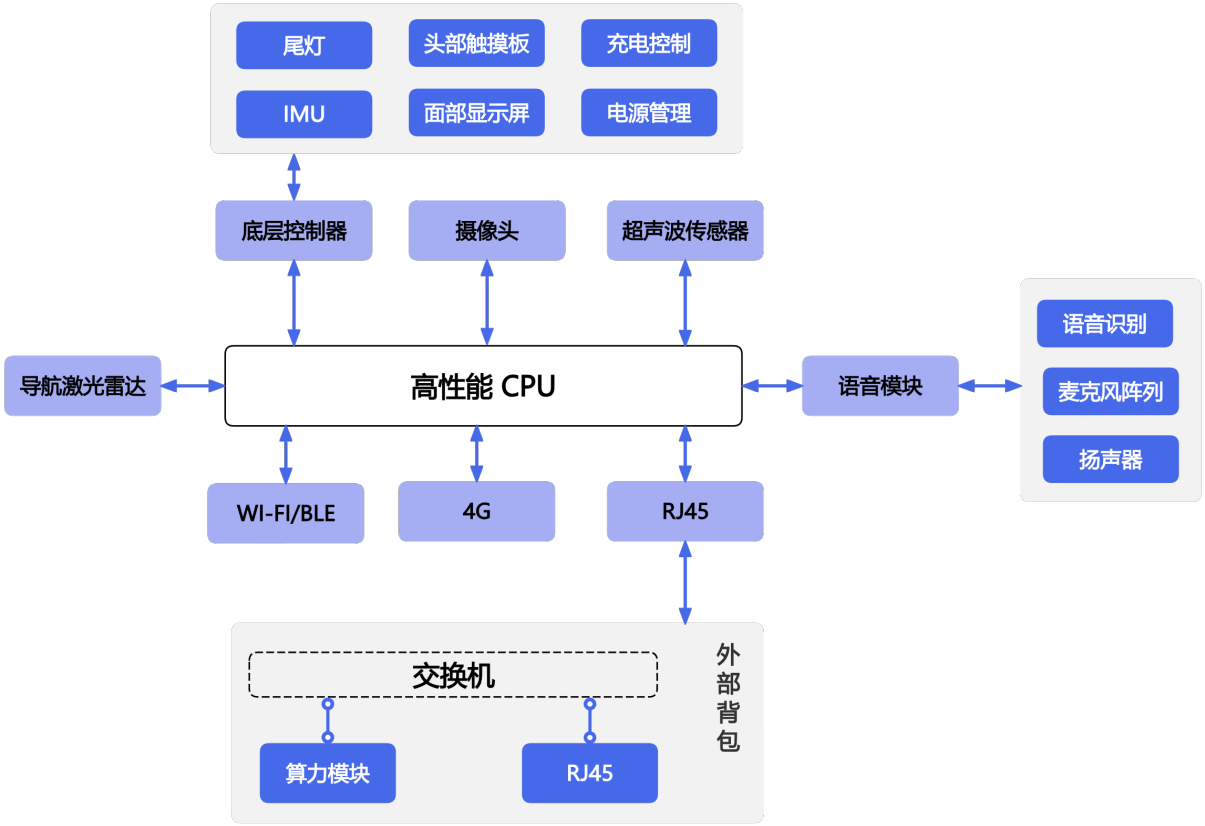
CHAPTER 2

软硬件架构

2.1 软件系统架构图

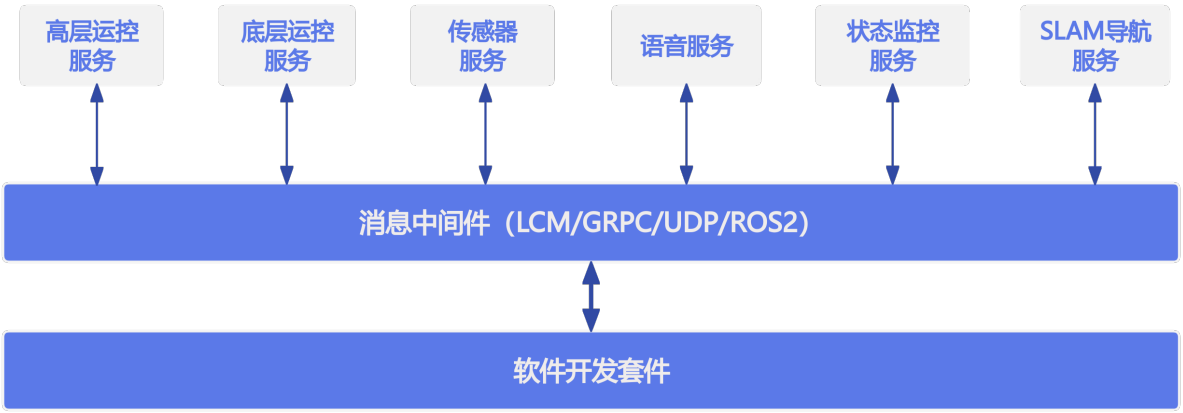

云服务
魔法原子管理平台
MQTT服务
HTTP Web API
MagicDog
网关模块
BLE模块
消息中间件
HAL(Driver)
HAL(Sensor)
运动控制模块
语音模块
其他模块
执行机构
传感器
交换机
Magicbot App
用户管理
网关模块
BLE模块
外部开发板/PC
SDK

2.2 硬件系统架构图


尾灯
头部触摸板
充电控制
IMU
面部显示屏
电源管理
底层控制器
摄像头
超声波传感器
导航激光雷达
高性能 CPU
语音模块
语音识别
麦克风阵列
扬声器
WI-FI/BLE
4G
RJ45
交换机
外部背包
算力模块
RJ45

CHAPTER 3

服务介绍



MagicDog 通过消息中间件（LCM/GRPC/ROS2）提供如下服务：

- **高层运控服务**

基于 MagicDog 内置步态控制器，提供步态切换、特技执行、姿态控制和速度控制等功能（等价于遥控器操作）。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状态、IMU 数据等信息，同时可实时下发关节控制指令。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、语音播放、获取原始语音数据等。语音服务使用 GRPC/LCM 进行通信。

- **传感器服务**

支持雷达、RGBD 相机、双目相机等传感器的数据订阅。传感器服务使用 GRPC/LCM/ROS2 进行通信。

- **状态监控服务**

通过服务接口可订阅机器人本体的软硬件故障信息和 BMS 电池状态。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 4

快速开始

4.1 系统环境

推荐在 Ubuntu 22.04 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，不支持开发环境。

APP 和 SDK 不支持同时控制机器狗

4.1.1 开发环境要求

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

4.1.2 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
```

4.2 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 192.168.55.X 网段下，推荐使用 192.168.55.10。有经验的用户可自行配置网络环境。

假设 SDK 二次开发 PC 与机器人连接的网口为 `eno1`，需要进行如下配置以便 SDK 与机器人的底层通信：

```
sudo ifconfig eno1 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eno1
```

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="eno1"

  # 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：

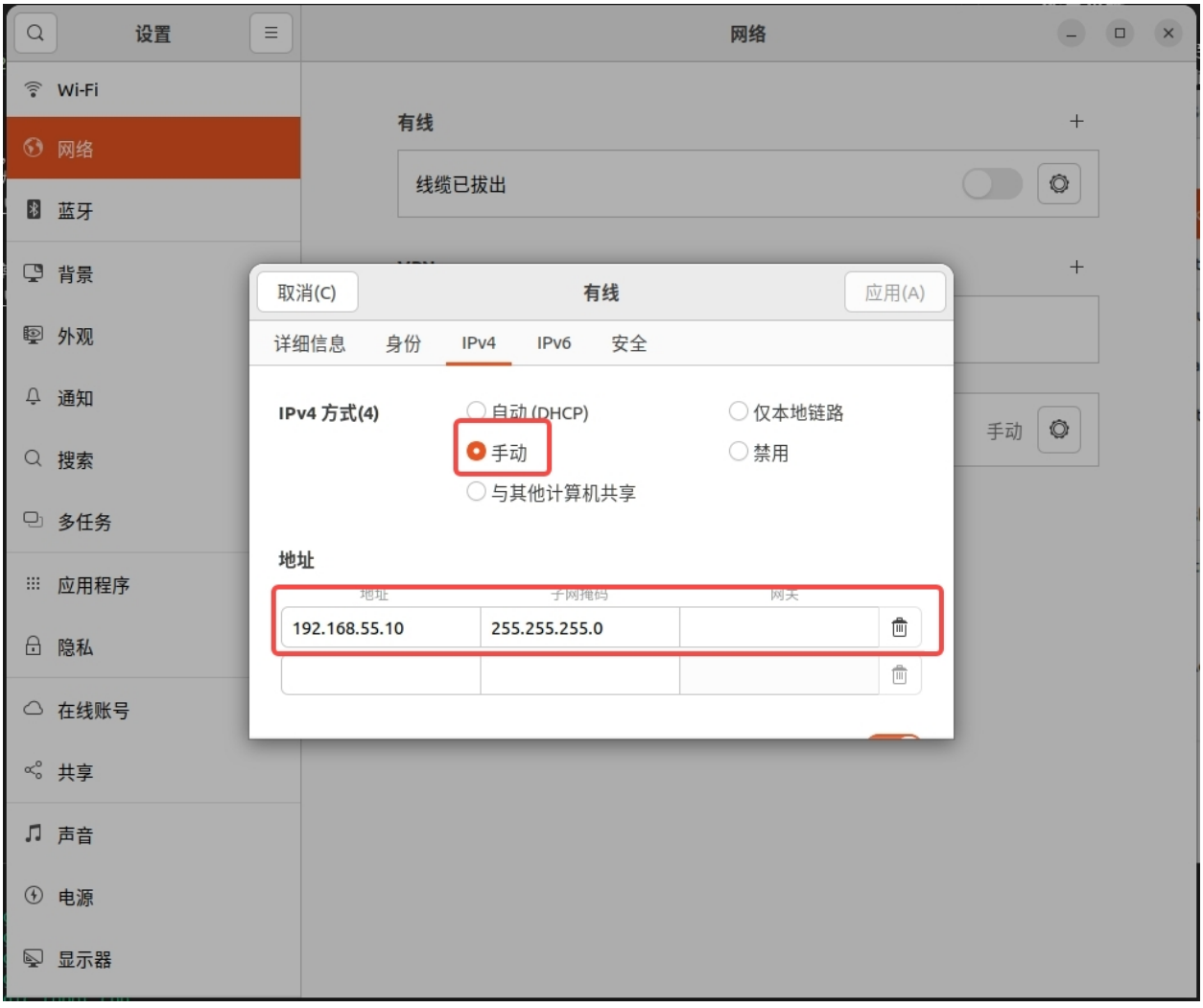
```
sudo systemctl restart NetworkManager
```

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 eno1 scope link
```

4.2.1 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。机器人机载电脑的 IP 地址为 192.168.55.200，所以需要将电脑 ip 设置为同一网段，建议 192.168.55.10:



2. 为了测试通信连接是否正常，可以通过 ping 进行测试：

```
eame@eame:~$ ping 192.168.55.200
PING 192.168.55.200 (192.168.55.200) 56(84) bytes of data.
64 bytes from 192.168.55.200: icmp_seq=1 ttl=64 time=1.61 ms
64 bytes from 192.168.55.200: icmp_seq=2 ttl=64 time=0.960 ms
64 bytes from 192.168.55.200: icmp_seq=3 ttl=64 time=0.807 ms
64 bytes from 192.168.55.200: icmp_seq=4 ttl=64 time=0.823 ms
64 bytes from 192.168.55.200: icmp_seq=5 ttl=64 time=1.07 ms
64 bytes from 192.168.55.200: icmp_seq=6 ttl=64 time=0.984 ms
64 bytes from 192.168.55.200: icmp_seq=7 ttl=64 time=0.786 ms
64 bytes from 192.168.55.200: icmp_seq=8 ttl=64 time=1.09 ms
```

4.3 安装与编译

以下步骤假设工作目录为`/home/magicbot/workspace`

4.3.1 安装 magicdog-sdk

```
cd /home/magicbot/workspace/magicdog_sdk/
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/opt/magic_robotics/magicdog_sdk
make -j8
sudo make install
```

上述命令会把 `magicdog_sdk` 安装到 `/opt/magic_robotics/magicdog_sdk` 目录下

4.3.2 例程编译

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

运行上述命令后，若进度进行到 100% 且没有报错，则意味着编译成功。

```
[  7%] Building CXX object example/monitor_example/CMakeFiles/monitor_example.dir/monitor_example.cpp.o
[ 14%] Linking CXX executable ../../monitor_example
[ 14%] Built target monitor_example
[ 21%] Building CXX object example/low_level_motion_example/CMakeFiles/low_level_motion_example.dir/low_level_motion_example.cpp.o
[ 28%] Linking CXX executable ../../low_level_motion_example
[ 28%] Built target low_level_motion_example
[ 35%] Building CXX object example/high_level_motion_example/CMakeFiles/recovery_stand_example.dir/recovery_stand_example.cpp.o
[ 42%] Linking CXX executable ../../recovery_stand_example
[ 42%] Built target recovery_stand_example
[ 50%] Building CXX object example/high_level_motion_example/CMakeFiles/execute_trick_action_example.dir/execute_trick_action_example.cpp.o
[ 57%] Linking CXX executable ../../execute_trick_action_example
[ 57%] Built target execute_trick_action_example
[ 64%] Building CXX object example/high_level_motion_example/CMakeFiles/joy_control_example.dir/joy_control_example.cpp.o
[ 71%] Linking CXX executable ../../joy_control_example
[ 71%] Built target joy_control_example
[ 78%] Building CXX object example/audio_example/CMakeFiles/audio_example.dir/audio_example.cpp.o
[ 85%] Linking CXX executable ../../audio_example
[ 85%] Built target audio_example
[ 92%] Building CXX object example/sensor_example/CMakeFiles/sensor_example.dir/sensor_example.cpp.o
[100%] Linking CXX executable ../../sensor_example
[100%] Built target sensor_example
Install the project
```

4.3.3 用户模块导入 magicdog-sdk

如果用户需要在自己的模块中引入 `magicdog-sdk`，可以参考 `example/cmake_example/CMakeLists.txt`

4.4 C++ 例程示例

magicdog_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
- **底层运控示例**:
  - low_level_motion_example
- **高层运控示例**:
  - high_level_motion_example
- **slam 示例**:
  - slam_example
- **导航示例**:
  - navigation_example
- **显示示例**:
  - display_example

4.4.1 进入调试模式：

按照操作流程，确保机器人进入调试模式

4.4.2 运行例程

进入 magicdog_sdk/build 目录，执行如下命令：

```
cd /home/magicbot/workspace/magicdog_sdk/build
# 环境配置
sudo ifconfig eno1 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eno1
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$LD_LIBRARY_PATH

# 执行语音示例
./audio_example
```

4.5 Python 例程示例

magicdog_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
- **slam 示例**:
  - slam_example.py
- **导航示例**:
  - navigation_example.py
- **显示示例**:
  - display_example.py

4.5.1 进入调试模式：

按照操作流程，确保机器人进入调试模式

4.5.2 运行例程

进入 magicdog_sdk/example/python 目录，执行如下命令：

```
cd /home/magicbot/workspace/magicdog_sdk/example/python/

# 环境配置
sudo ifconfig eno1 multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eno1
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$LD_LIBRARY_PATH

# 执行语音示例
python3 audio_example.py
```

> 注意：手动执行 `sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eno1` 只会生效一次，开机重启或者网线断连之后，需要重新执行

获取 python api 帮助信息：

```
cd /home/magicbot/workspace/magicdog_sdk/example/python
# 环境配置
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$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Magicdog python SDK接口
>>> import magicdog_python
# 查看所有接口信息
>>> help(magicdog_python)
# 查看CameraInfo结构信息
>>> help(magicdog_python.CameraInfo)
# 查看HighLevelMotionController对象信息
>>> help(magicdog_python.HighLevelMotionController)
# 查看LowLevelMotionController对象信息
>>> help(magicdog_python.LowLevelMotionController)
# 查看SensorController对象信息
>>> help(magicdog_python.SensorController)
# 查看AudioController对象信息
>>> help(magicdog_python.AudioController)
```

(下页继续)

(续上页)

```
# 查看StateMonitor对象信息
>>> help(magicdog_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 magicdog_python
>>> help(magicdog_python.TrickAction)

Help on class TrickAction in module magicdog_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: magicdog_python.TrickAction, /) -> int
 |
 |  __init__(...)
 |      __init__(self: magicdog_python.TrickAction, value: typing.SupportsInt) -> None
 |
 |  __int__(...)
 |      __int__(self: magicdog_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: magicdog_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_ACT_CUTE = <TrickAction.ACTION_ACT_CUTE: 46>
|
|  ACTION_BACK_FLIP = <TrickAction.ACTION_BACK_FLIP: 41>
|
|  ACTION_BACK_HOME = <TrickAction.ACTION_BACK_HOME: 110>
|
|  ACTION_BOXING = <TrickAction.ACTION_BOXING: 47>
|
|  ACTION_CHEER_UP = <TrickAction.ACTION_CHEER_UP: 35>
|
|  ACTION_DANCE = <TrickAction.ACTION_DANCE: 115>
|
|  ACTION_DANCE2 = <TrickAction.ACTION_DANCE2: 91>
|
|  ACTION_EMERGENCY_STOP = <TrickAction.ACTION_EMERGENCY_STOP: 101>
|
|  ACTION_FRONT_FLIP = <TrickAction.ACTION_FRONT_FLIP: 42>
|
|  ACTION_HAPPY_NEW_YEAR = <TrickAction.ACTION_HAPPY_NEW_YEAR: 105>
|
|  ACTION_HIGH_FIVES = <TrickAction.ACTION_HIGH_FIVES: 36>
|
|  ACTION_HIGH_JUMP = <TrickAction.ACTION_HIGH_JUMP: 38>
```

(下页继续)

(续上页)

```
|
|  ACTION_IMITATE = <TrickAction.ACTION_IMITATE: 32>
|
|  ACTION_JUMP_FRONT = <TrickAction.ACTION_JUMP_FRONT: 45>
|
|  ACTION_JUMP_JACK = <TrickAction.ACTION_JUMP_JACK: 30>
|
|  ACTION_LEAP_FROG = <TrickAction.ACTION_LEAP_FROG: 40>
|
|  ACTION_LEAVE_HOME = <TrickAction.ACTION_LEAVE_HOME: 111>
|
|  ACTION_LEFT_SIDE_SOMERSAULT = <TrickAction.ACTION_LEFT_SIDE_SOMERSAULT...
|
|  ACTION_LIE_DOWN = <TrickAction.ACTION_LIE_DOWN: 102>
|
|  ACTION_NONE = <TrickAction.ACTION_NONE: 0>
|
|  ACTION_PUSH_UP = <TrickAction.ACTION_PUSH_UP: 34>
|
|  ACTION_RANDOM_DANCE = <TrickAction.ACTION_RANDOM_DANCE: 49>
|
|  ACTION_RECOVERY_STAND = <TrickAction.ACTION_RECOVERY_STAND: 103>
|
|  ACTION_RIGHT_SIDE_SOMERSAULT = <TrickAction.ACTION_RIGHT_SIDE_SOMERSAU...
|
|  ACTION_ROLL_ABOUT = <TrickAction.ACTION_ROLL_ABOUT: 116>
|
|  ACTION_SCRATCH = <TrickAction.ACTION_SCRATCH: 37>
|
|  ACTION_SHAKE_HEAD = <TrickAction.ACTION_SHAKE_HEAD: 33>
|
|  ACTION_SHAKE_LEFT_HAND = <TrickAction.ACTION_SHAKE_LEFT_HAND: 118>
|
|  ACTION_SHAKE_RIGHT_HAND = <TrickAction.ACTION_SHAKE_RIGHT_HAND: 117>
|
|  ACTION_SIDE_SOMERSAULT = <TrickAction.ACTION_SIDE_SOMERSAULT: 48>
|
|  ACTION_SIT_DOWN = <TrickAction.ACTION_SIT_DOWN: 119>
|
|  ACTION_SLOW_GO_BACK = <TrickAction.ACTION_SLOW_GO_BACK: 109>
|
|  ACTION_SLOW_GO_FRONT = <TrickAction.ACTION_SLOW_GO_FRONT: 108>
|
```

(下页继续)

(续上页)

```
|  ACTION_SPACE_WALK = <TrickAction.ACTION_SPACE_WALK: 31>
|
|  ACTION_SPIN_JUMP_LEFT = <TrickAction.ACTION_SPIN_JUMP_LEFT: 43>
|
|  ACTION_SPIN_JUMP_RIGHT = <TrickAction.ACTION_SPIN_JUMP_RIGHT: 44>
|
|  ACTION_STOMP = <TrickAction.ACTION_STOMP: 29>
|
|  ACTION_STRETCH = <TrickAction.ACTION_STRETCH: 28>
|
|  ACTION_SWING_BODY = <TrickAction.ACTION_SWING_BODY: 27>
|
|  ACTION_SWING_DANCE = <TrickAction.ACTION_SWING_DANCE: 39>
|
|  ACTION_TURN_AROUND = <TrickAction.ACTION_TURN_AROUND: 112>
|
|  ACTION_WIGGLE_HIP = <TrickAction.ACTION_WIGGLE_HIP: 26>
|
|  ----------------------------------------------------------------------
|  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.
```

4.6 其他

4.6.1 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/magicdog_mjrrt.yaml
/tmp/magicdog_mjrrt.yaml
```

4.6.2 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/magicdog_sdk.log
/tmp/logs/magicdog_sdk.log
```

CHAPTER 5

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、音频控制器、状态监控以及传感器控制器等。

5.1 接口定义

MagicRobot 是机器人系统的统一入口类。

5.1.1 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。

5.1.2 ~MagicRobot

项目	内容
函数名	~MagicRobot
函数声明	`~MagicRobot();`
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。

5.1.3 Initialize

项目	内容
函数名	Initialize
函数声明	`bool Initialize(const std::string& 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。

5.1.4 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。

5.1.5 Release

项目	内容
函数名	Release
函数声明	`void Release();`
功能概述	释放机器人系统资源。
备注	调用该函数可手动释放 SDK 内部占用的资源。

5.1.6 Connect

项目	内容
函数名	Connect
函数声明	`Status Connect(int timeout_ms);`
功能概述	与机器人服务建立通信连接。
参数	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功。
备注	阻塞接口，需初始化后调用。

5.1.7 Disconnect

项目	内容
函数名	Disconnect
函数声明	`Status Disconnect(int timeout_ms);`
功能概述	断开与机器人服务的连接。
参数	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	gRPC 调用状态。
备注	阻塞接口，与 Connect 配对使用。

5.1.8 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。

5.1.9 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。

5.1.10 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。

5.1.11 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、遥控等控制功能。

5.1.12 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，可直接控制关节电机等。

5.1.13 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，可播放语音和控制音量。

5.1.14 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 IMU、RGBD 等传感器数据读取。

5.1.15 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。

5.1.16 GetSlamNavController

项目	内容
函数名	GetSlamNavController
函数声明	`SlamNavController& GetSlamNavController();`
功能概述	获取 SLAM/导航控制器对象。
返回值	引用类型，用于访问地图、定位等 SLAM 相关数据。
备注	非阻塞接口，封装地图构建、定位、导航等相关功能的使用。

5.1.17 OpenChannelSwitch

项目	内容
函数名	OpenChannelSwitch
函数声明	`Status OpenChannelSwitch(int timeout_ms);`
功能概述	打开 lcm 通道开关。
参数	`timeout_ms`（可选参数，单位毫秒），操作超时时间，默认 5000 ms。
返回值	`Status` 枚举，表示操作状态。
备注	用于控制 SDK lcm 通信通道的启用。

5.1.18 CloseChannelSwitch

项目	内容
函数名	CloseChannelSwitch
函数声明	`Status CloseChannelSwitch(int timeout_ms);`
功能概述	关闭 lcm 通道开关。
参数	`timeout_ms`（可选参数，单位毫秒），操作超时时间，默认 5000 ms。
返回值	`Status` 枚举，表示操作状态。
备注	用于控制 SDK lcm 通信通道的关闭。

CHAPTER 6

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。

⚠ **Notice:** 某些特技动作，比如前空翻、后空翻对场地要求较大，执行时需要额外注意，建议所有特技动作在大的空场地测试后，再在小的场地使用。

6.1 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、特技等控制操作，封装底层细节以供上层系统调用。

6.1.1 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。

6.1.2 ~HighLevelMotionController

项目	内容
函数名	~HighLevelMotionController
函数声明	`virtual ~HighLevelMotionController();`
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。

6.1.3 Initialize

项目	内容
函数名	Initialize
函数声明	`virtual bool Initialize() override;`
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。

6.1.4 Shutdown

项目	内容
函数名	Shutdown
函数声明	`virtual void Shutdown() override;`
功能概述	关闭控制器并释放资源。
备注	配合 Initialize 使用。

6.1.5 SetGait

项目	内容
函数名	SetGait
函数声明	`Status SetGait(const GaitMode gait_mode, int timeout_ms);`
功能概述	设置机器人步态模式（如位控站立、力控站立、慢跑等）。
参数说明	`gait_mode`：步态控制枚举。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，支持多种步态模式切换。

6.1.6 GetGait

项目	内容
函数名	GetGait
函数声明	`Status GetGait(GaitMode& gait_mode, int timeout_ms);`
功能概述	获取机器人步态模式（如位控站立、力控站立、慢跑等）。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

6.1.7 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（如扭屁股、趴下等）。
参数说明	`trick_action`：特技动作标识。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，需确保机器人当前处于可执行特技的状态。

- 注意事项：特技动作必须在 GaitMode::GAIT_STAND_R(2) 位控站立步态下才能执行

6.1.8 EnableJoyStick

项目	内容
函数名	EnableJoyStick
函数声明	`Status EnableJoyStick();`
功能概述	开启摇杆控制命令。
参数说明	无。
返回值	`Status::OK` 表示操作成功，其他为失败。
备注	用于允许通过摇杆发送实时控制指令。

6.1.9 DisableJoyStick

项目	内容
函数名	DisableJoyStick
函数声明	`Status DisableJoyStick();`
功能概述	关闭摇杆控制命令。
参数说明	无。
返回值	`Status::OK` 表示操作成功，其他为失败。
备注	切换导航模式之前，必须先关闭摇杆控制命令。

6.1.10 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。

6.1.11 GetAllGaitSpeedRatio

项目	内容
函数名	GetAllGaitSpeedRatio
函数声明	`Status GetAllGaitSpeedRatio(AllGaitSpeedRatio& gait_speed_ratios, int timeout_ms);`
功能概述	获取所有步态以及对应前进、横移、旋转速度比例。
参数说明	`gait_speed_ratios`：所有步态以及对应前进、横移、旋转速度比例。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于获取所有步态的速度比例配置。

6.1.12 SetGaitSpeedRatio

项目	内容
函数名	SetGaitSpeedRatio
函数声明	`Status SetGaitSpeedRatio(GaitMode gait_mode, const GaitSpeedRatio& gait_speed_ratio, int timeout_ms);`
功能概述	设置步态以及对应前进、横移、旋转速度比例。
参数说明	`gait_mode`：步态模式 `gait_speed_ratio`：步态以及对应前进、横移、旋转速度比例 `timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置特定步态的速度比例配置。

6.1.13 GetHeadMotorEnabled

项目	内容
函数名	GetHeadMotorEnabled
函数声明	`Status GetHeadMotorEnabled(bool& enabled, int timeout_ms);`
功能概述	获取头部电机使能状态。
参数说明	`enabled`：返回参数，true 表示已使能，false 表示未使能。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于查询头部电机当前使能状态。

6.1.14 EnableHeadMotor

项目	内容
函数名	EnableHeadMotor
函数声明	`Status EnableHeadMotor(int timeout_ms);`
功能概述	使能头部电机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于启用头部电机控制。

6.1.15 DisableHeadMotor

项目	内容
函数名	DisableHeadMotor
函数声明	`Status DisableHeadMotor(int timeout_ms);`
功能概述	关闭头部电机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于禁用头部电机控制。

6.1.16 GetHeadPosition

项目	内容
函数名	GetHeadPosition
函数声明	`Status GetHeadPosition(EulerAngles& euler_angles, int timeout_ms);`
功能概述	获取头部位置。
参数说明	`euler_angles`：返回参数，头部当前绝对位置。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于获取头部位置。

6.1.17 SetHeadPosition

项目	内容
函数名	SetHeadPosition
函数声明	`Status SetHeadPosition(const EulerAngles& euler_angles, int timeout_ms);`
功能概述	设置头部位置。
参数说明	`euler_angles`：头部目标绝对位置。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置头部位置。

6.2 类型定义

6.2.1 GaitSpeedRatio —步态速度比例结构体

字段名	类型	描述
straight_ratio	double	前进速度比例
turn_ratio	double	旋转速度比例
lateral_ratio	double	横移速度比例

6.2.2 AllGaitSpeedRatio —所有步态速度比例结构体

字段名	类型	描述
gait_speed_ratios	std::map<GaitMode, GaitSpeedRatio>	步态模式到速度比例的映射

6.2.3 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）

6.2.4 EulerAngles —欧拉角结构体

字段名	类型	描述
roll	double	横滚角（Roll），绕 X 轴旋转的角度（单位为弧度）
pitch	double	俯仰角（Pitch），绕 Y 轴旋转的角度（单位为弧度）
yaw	double	偏航角（Yaw），绕 Z 轴旋转的角度（单位为弧度）

6.3 枚举类型定义

6.3.1 GaitMode —机器人步态模式枚举

枚举值	数值	描述
GAIT_PASSIVE	0	掉落（关闭电机使能）
GAIT_STAND_R	2	位控站立、RecoveryStand
GAIT_STAND_B	3	力控站立、姿态展示、BalanceStand
GAIT_RUN_FAST	8	快跑
GAIT_DOWN_CLIMB_STAIRS	9	下爬楼梯 => 盲走 => 慢跑
GAIT_TROT	10	小跑
GAIT_PRONK	11	跳跃
GAIT_BOUND	12	前后跳
GAIT_AMBLE	14	交叉步
GAIT_CRAWL	29	爬行
GAIT_LOWLEVL_SDK	30	低层 SDK 步态
GAIT_WALK	39	缓走
GAIT_UP_CLIMB_STAIRS	56	上爬楼梯（全地形）
GAIT_RL_TERRAIN	110	全地形
GAIT_RL_FALL_RECOVERY	111	跌倒爬起
GAIT_RL_HAND_STAND	112	倒立
GAIT_RL_FOOT_STAND	113	正立
GAIT_ENTER_RL	1001	进入 RL
GAIT_DEFAULT	99	默认
GAIT_NONE	9999	无步态

6.3.2 TrickAction —特技动作指令枚举

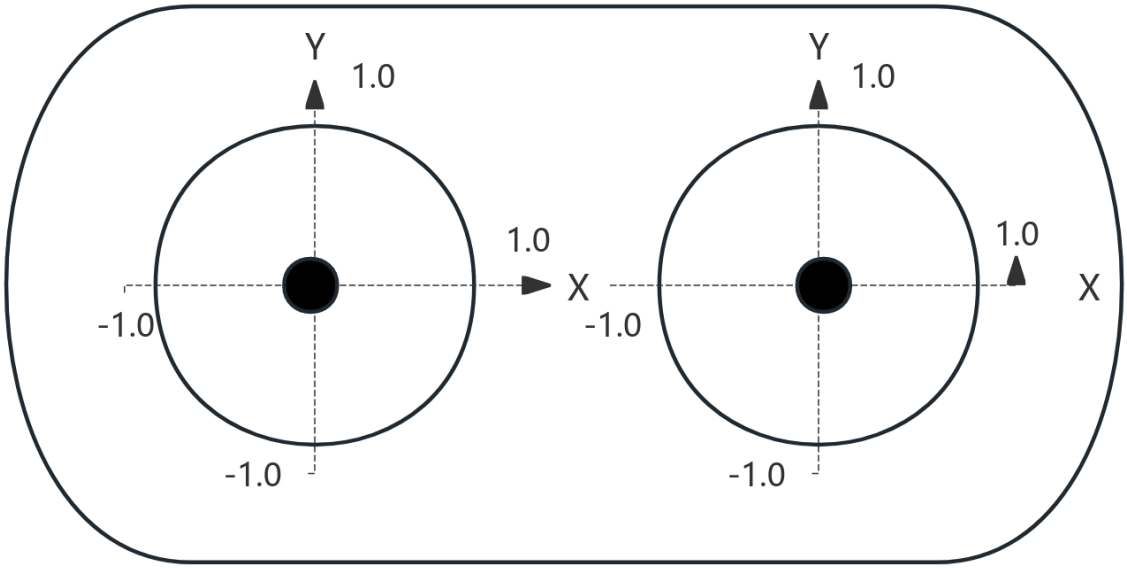
枚举值	数值	描述
ACTION_NONE	0	无动作
ACTION_WIGGLE_HIP	26	扭屁股
ACTION_SWING_BODY	27	甩身
ACTION_STRETCH	28	伸懒腰
ACTION_STOMP	29	跺脚
ACTION_JUMP_JACK	30	开合跳
ACTION_SPACE_WALK	31	太空步
ACTION_IMITATE	32	模仿
ACTION_SHAKE_HEAD	33	摇头

下页继续

表 1 – 续上页

枚举值	数值	描述
ACTION_PUSH_UP	34	俯卧撑
ACTION_CHEER_UP	35	加油
ACTION_HIGH_FIVES	36	击掌
ACTION_SCRATCH	37	挠痒
ACTION_HIGH_JUMP	38	跳高
ACTION_SWING_DANCE	39	摇摆舞
ACTION_LEAP_FROG	40	蛙跳
ACTION_BACK_FLIP	41	后空翻
ACTION_FRONT_FLIP	42	前空翻
ACTION_SPIN_JUMP_LEFT	43	旋转左跳 70 度
ACTION_SPIN_JUMP_RIGHT	44	旋转右跳 70 度
ACTION_JUMP_FRONT	45	前跳 0.5 米
ACTION_ACT_CUTE	46	撒娇
ACTION_BOXING	47	打拳
ACTION_SIDE_SOMERSAULT	48	侧空翻
ACTION_RANDOM_DANCE	49	随机跳舞
ACTION_LEFT_SIDE_SOMERSAULT	84	左侧侧空翻
ACTION_RIGHT_SIDE_SOMERSAULT	85	右侧侧空翻
ACTION_DANCE2	91	跳舞 2
ACTION_EMERGENCY_STOP	101	急停
ACTION_LIE_DOWN	102	趴下
ACTION_RECOVERY_STAND	103	起立
ACTION_HAPPY_NEW_YEAR	105	拜年 = 作揖
ACTION_SLOW_GO_FRONT	108	过来
ACTION_SLOW_GO_BACK	109	后退
ACTION_BACK_HOME	110	回窝
ACTION_LEAVE_HOME	111	离窝
ACTION_TURN_AROUND	112	转圈
ACTION_DANCE	115	跳舞
ACTION_ROLL_ABOUT	116	打滚
ACTION_SHAKE_RIGHT_HAND	117	握右手
ACTION_SHAKE_LEFT_HAND	118	握左手
ACTION_SIT_DOWN	119	坐下

6.4 遥控器示意图



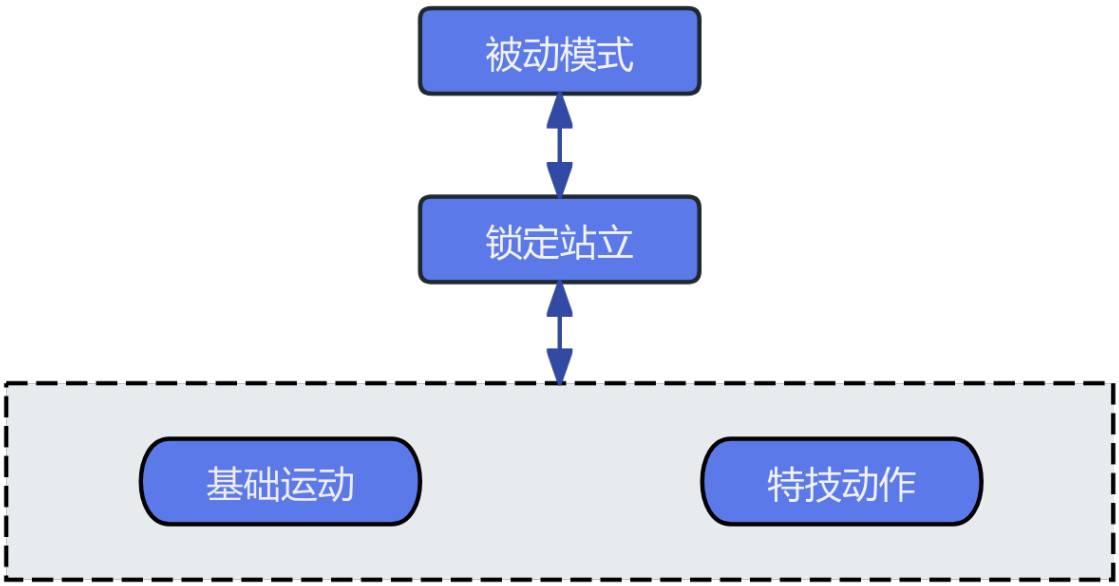
1. 左右摇杆 x 轴和 y 轴的取值范围为 [-1.0, 1.0];
2. 左右摇杆 x 轴和 y 轴的方向上/右为正，如示意图所示；

6.5 高层运动控制机器人状态介绍

机器人的运动包含位控站立、力控站立、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：

- 位控站立：在位控站立状态下，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。
- 力控站立：在力控站立状态下，可以用于姿态展示。
- 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。

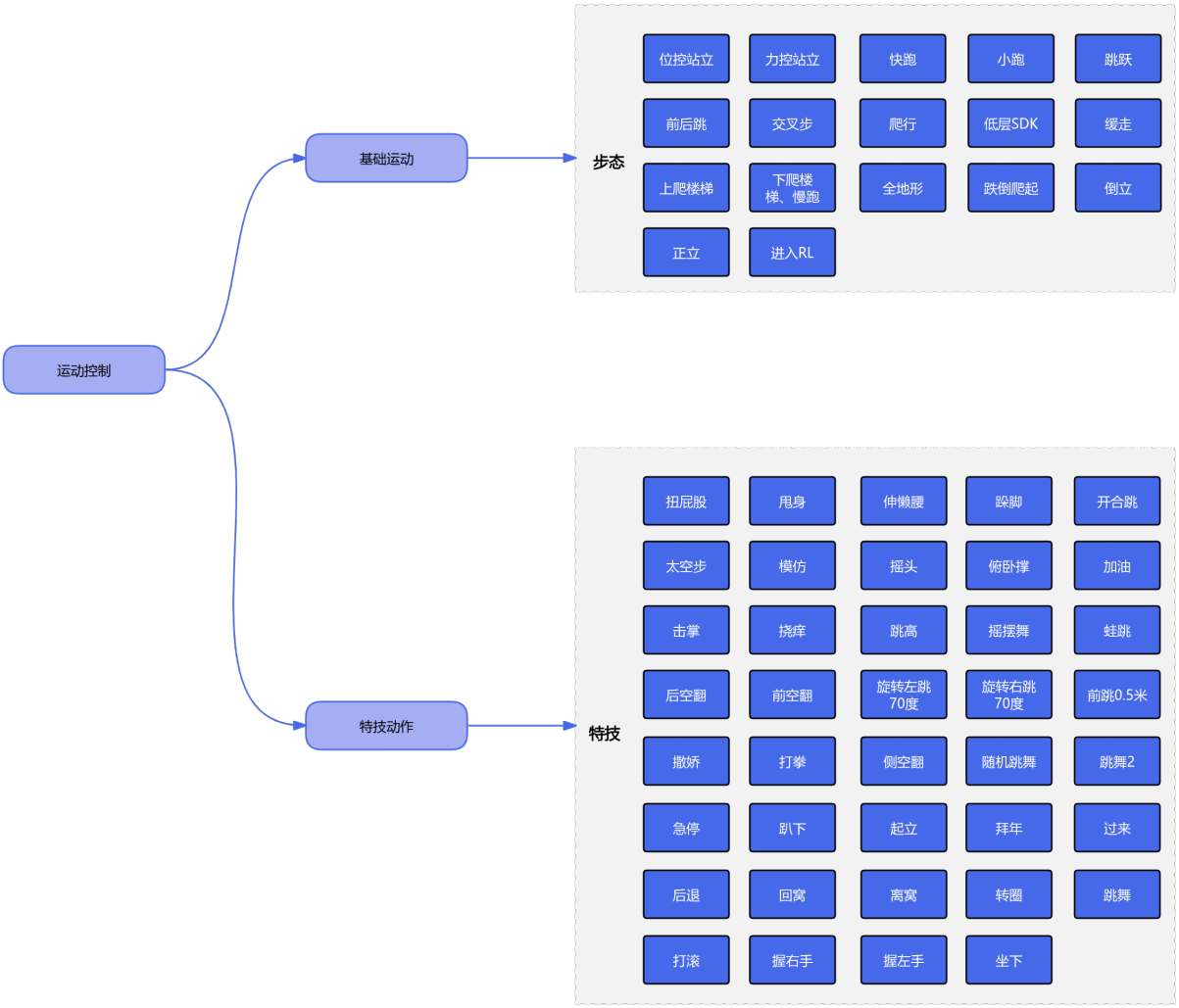
机器人状态切换机制；



6.6 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，比如扭屁股、趴下等。


运动控制
基础运动
步态
位控站立
力控站立
快跑
小跑
跳跃
前后跳
交叉步
爬行
低层SDK
缓走
上爬楼梯
下爬楼梯、慢跑
全地形
跌倒爬起
倒立
正立
进入RL
特技动作
特技
扭屁股
甩身
伸懒腰
跺脚
开合跳
太空步
模仿
摇头
俯卧撑
加油
击掌
挠痒
跳高
摇摆舞
蛙跳
后空翻
前空翻
旋转左跳70度
旋转右跳70度
前跳0.5米
撒娇
打拳
侧空翻
随机跳舞
跳舞2
急停
趴下
起立
拜年
过来
后退
回窝
离窝
转圈
跳舞
打滚
握右手
握左手
坐下

CHAPTER 7

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 可以使用话题通信方式实现对机器人关节的指令控制和状态获取。

7.1 接口定义

LowLevelMotionController 是面向底层开发的运动控制器，支持对腿等运动部件的直接控制与状态订阅。

⚠ **Notice:** 调用 PublishLegCommand 接口，用户命令才会下发到运控，因此需要用户保证调用 PublishLegCommand 的频率，建议是 500 hz。

7.1.1 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。

7.1.2 ~LowLevelMotionController

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

7.1.3 Initialize

项目	内容
函数名	Initialize
函数声明	`virtual bool Initialize() override;`
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。

7.1.4 Shutdown

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

7.1.5 SubscribeLegState

项目	内容
函数名	SubscribeLegState
函数声明	`void SubscribeLegState(LegJointStateCallback callback);`
功能概述	订阅下肢关节状态数据
参数说明	callback：回调函数，用于处理下肢关节状态的接收数据
备注	非阻塞接口。

7.1.6 UnsubscribeLegState

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

7.1.7 PublishLegCommand

项目	内容
函数名	PublishLegCommand
函数声明	`Status PublishLegCommand(const LegJointCommand& command);`
功能概述	发布下肢关节控制指令
参数说明	command：包含目标角度/速度等控制信息的下肢关节控制指令
返回值	`Status::OK` 表示成功，其他为失败。
备注	非阻塞接口。

7.2 类型定义

7.2.1 SingleLegJointCommand —单个下肢关节的控制命令

字段名	类型	描述
`q_des`	`float`	期望关节位置
`dq_des`	`float`	期望关节速度
`tau_des`	`float`	期望前馈力矩
`kp`	`float`	P 增益，必须为正数
`kd`	`float`	D 增益，必须为正数

7.2.2 LegJointCommand —整个下肢控制命令

字段名	类型	描述
`timestamp`	`int64_t`	时间戳（单位：纳秒）
`cmd`	`std::array<SingleLegJointCommand, kLegJointNum>`	控制命令数组

7.2.3 SingleLegJointState —单个下肢关节的状态

字段名	类型	描述
q	float	实际关节位置
dq	float	实际关节速度
tau_est	float	估计力矩

7.2.4 LegState —整个下肢状态信息

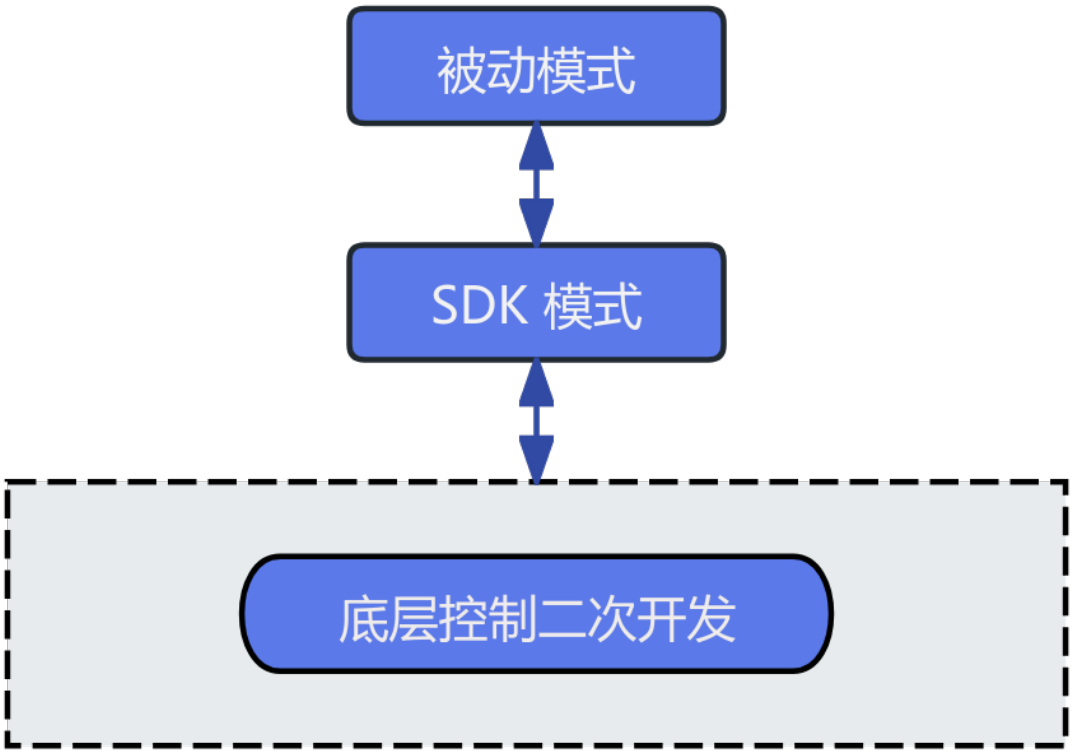
字段名	类型	描述
timestamp	int64_t	时间戳（单位：纳秒）
state	std::array<SingleLegJointState, kLegJointNum>	所有下肢关节状态

7.3 URDF 参考

机器人 URDF

7.4 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：



7.5 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 8

传感器控制服务 C++

提供机器人系统传感器（雷达/RGBD 相机/双目相机）服务，通过 SensorController 可以通过 RPC 和话题方式实现对机器人传感器的指令控制和状态获取。

8.1 接口定义

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

8.1.1 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。

8.1.2 ~SensorController

项目	内容
函数名	~SensorController
函数声明	`virtual ~SensorController();`
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。

8.1.3 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。

8.1.4 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。

8.1.5 OpenLaserScan

项目	内容
函数名	OpenLaserScan
函数声明	`Status OpenLaserScan(int timeout_ms);`
功能概述	打开雷达。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

8.1.6 CloseLaserScan

项目	内容
函数名	CloseLaserScan
函数声明	`Status CloseLaserScan(int timeout_ms);`
功能概述	关闭雷达。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

8.1.7 OpenRgbdCamera

项目	内容
函数名	OpenRgbdCamera
函数声明	`Status OpenRgbdCamera(int timeout_ms);`
功能概述	打开 RGBD 相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

8.1.8 CloseRgbdCamera

项目	内容
函数名	CloseRgbdCamera
函数声明	`Status CloseRgbdCamera(int timeout_ms);`
功能概述	关闭 RGBD 相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

8.1.9 OpenBinocularCamera

项目	内容
函数名	OpenBinocularCamera
函数声明	`Status OpenBinocularCamera(int timeout_ms);`
功能概述	打开双目相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

8.1.10 CloseBinocularCamera

项目	内容
函数名	CloseBinocularCamera
函数声明	`Status CloseBinocularCamera(int timeout_ms);`
功能概述	关闭双目相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

8.1.11 SubscribeUltra

项目	内容
函数名	SubscribeUltra
函数声明	`void SubscribeUltra(const UltraCallback callback);`
功能概述	订阅超声波数据
参数说明	callback：接收到超声波数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.12 UnsubscribeUltra

项目	内容
函数名	UnsubscribeUltra
函数声明	`void UnsubscribeUltra();`
功能概述	取消订阅超声波数据
备注	非阻塞接口，用于停止接收超声波数据。

8.1.13 SubscribeHeadTouch

项目	内容
函数名	SubscribeHeadTouch
函数声明	`void SubscribeHeadTouch(const HeadTouchCallback callback);`
功能概述	订阅头部触摸数据
参数说明	callback：接收到头部触摸数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.14 UnsubscribeHeadTouch

项目	内容
函数名	UnsubscribeHeadTouch
函数声明	`void UnsubscribeHeadTouch();`
功能概述	取消订阅头部触摸数据
备注	非阻塞接口，用于停止接收头部触摸数据。

8.1.15 SubscribeLaserScan

项目	内容
函数名	SubscribeLaserScan
函数声明	`void SubscribeLaserScan(const LaserScanCallback callback);`
功能概述	订阅激光雷达数据
参数说明	callback：接收到激光雷达数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.16 UnsubscribeLaserScan

项目	内容
函数名	UnsubscribeLaserScan
函数声明	`void UnsubscribeLaserScan();`
功能概述	取消订阅激光雷达数据
备注	非阻塞接口，用于停止接收激光雷达数据。

8.1.17 SubscribeRgbDepthCameraInfo

项目	内容
函数名	SubscribeRgbDepthCameraInfo
函数声明	`void SubscribeRgbDepthCameraInfo(const CameraInfoCallback callback);`
功能概述	订阅 RGBD 深度相机内参数据
参数说明	callback：接收到 RGBD 深度相机内参数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.18 UnsubscribeRgbDepthCameraInfo

项目	内容
函数名	UnsubscribeRgbDepthCameraInfo
函数声明	`void UnsubscribeRgbDepthCameraInfo();`
功能概述	取消订阅 RGBD 深度相机内参数据
备注	非阻塞接口，用于停止接收 RGBD 深度相机内参数据。

8.1.19 SubscribeRgbdDepthImage

项目	内容
函数名	SubscribeRgbdDepthImage
函数声明	`void SubscribeRgbdDepthImage(const ImageCallback callback);`
功能概述	订阅 RGBD 深度图像数据
参数说明	callback：接收到 RGBD 深度图像数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.20 UnsubscribeRgbdDepthImage

项目	内容
函数名	UnsubscribeRgbdDepthImage
函数声明	`void UnsubscribeRgbdDepthImage();`
功能概述	取消订阅 RGBD 深度图像数据
备注	非阻塞接口，用于停止接收 RGBD 深度图像数据。

8.1.21 SubscribeRgbdColorCameraInfo

项目	内容
函数名	SubscribeRgbdColorCameraInfo
函数声明	`void SubscribeRgbdColorCameraInfo(const CameraInfoCallback callback);`
功能概述	订阅 RGBD 彩色图像内参数据
参数说明	callback：接收到 RGBD 彩色图像内参数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.22 UnsubscribeRgbdColorCameraInfo

项目	内容
函数名	UnsubscribeRgbdColorCameraInfo
函数声明	`void UnsubscribeRgbdColorCameraInfo();`
功能概述	取消订阅 RGBD 彩色图像内参数据
备注	非阻塞接口，用于停止接收 RGBD 彩色图像内参数据。

8.1.23 SubscribeRgbdColorImage

项目	内容
函数名	SubscribeRgbdColorImage
函数声明	`void SubscribeRgbdColorImage(const ImageCallback callback);`
功能概述	订阅 RGBD 彩色图像数据
参数说明	callback：接收到 RGBD 彩色图像数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.24 UnsubscribeRgbdColorImage

项目	内容
函数名	UnsubscribeRgbdColorImage
函数声明	`void UnsubscribeRgbdColorImage();`
功能概述	取消订阅 RGBD 彩色图像数据
备注	非阻塞接口，用于停止接收 RGBD 彩色图像数据。

8.1.25 SubscribeImu

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

8.1.26 UnsubscribeImu

项目	内容
函数名	UnsubscribeImu
函数声明	`void UnsubscribeImu();`
功能概述	取消订阅 IMU 数据
备注	非阻塞接口，用于停止接收 IMU 数据。

8.1.27 SubscribeLeftBinocularHighImg

项目	内容
函数名	SubscribeLeftBinocularHighImg
函数声明	`void SubscribeLeftBinocularHighImg(const CompressedImageCallback callback);`
功能概述	订阅左侧高质量双目数据
参数说明	callback：接收到左侧高质量双目数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.28 UnsubscribeLeftBinocularHighImg

项目	内容
函数名	UnsubscribeLeftBinocularHighImg
函数声明	`void UnsubscribeLeftBinocularHighImg();`
功能概述	取消订阅左侧高质量双目数据
备注	非阻塞接口，用于停止接收左侧高质量双目数据。

8.1.29 SubscribeLeftBinocularLowImg

项目	内容
函数名	SubscribeLeftBinocularLowImg
函数声明	`void SubscribeLeftBinocularLowImg(const CompressedImageCallback callback);`
功能概述	订阅左侧低质量双目数据
参数说明	callback：接收到左侧低质量双目数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.30 UnsubscribeLeftBinocularLowImg

项目	内容
函数名	UnsubscribeLeftBinocularLowImg
函数声明	`void UnsubscribeLeftBinocularLowImg();`
功能概述	取消订阅左侧低质量双目数据
备注	非阻塞接口，用于停止接收左侧低质量双目数据。

8.1.31 SubscribeRightBinocularLowImg

项目	内容
函数名	SubscribeRightBinocularLowImg
函数声明	`void SubscribeRightBinocularLowImg(const CompressedImageCallback callback);`
功能概述	订阅右侧低质量双目数据
参数说明	callback：接收到右侧低质量双目数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.32 UnsubscribeRightBinocularLowImg

项目	内容
函数名	UnsubscribeRightBinocularLowImg
函数声明	`void UnsubscribeRightBinocularLowImg();`
功能概述	取消订阅右侧低质量双目数据
备注	非阻塞接口，用于停止接收右侧低质量双目数据。

8.1.33 SubscribeDepthImage

项目	内容
函数名	SubscribeDepthImage
函数声明	`void SubscribeDepthImage(const ImageCallback callback);`
功能概述	订阅深度图像
参数说明	callback：接收到深度图像后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

8.1.34 UnsubscribeDepthImage

项目	内容
函数名	UnsubscribeDepthImage
函数声明	`void UnsubscribeDepthImage();`
功能概述	取消订阅深度图像
备注	非阻塞接口，用于停止接收深度图像数据。

8.2 类型定义

8.2.1 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]`	`double[4]`	姿态四元数（w, x, y, z）
`angular_velocity[3]`	`double[3]`	角速度（单位：rad/s）
`linear_acceleration[3]`	`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）

8.2.2 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

8.2.3 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

8.2.4 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	std::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	std::vector<uint8_t>	原始点云字节数据
is_dense	bool	是否稠密点云（无无效点）

8.2.5 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	std::vector<uint8_t>	原始图像字节数据

8.2.6 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	std::vector<double>	畸变参数数组
K	double[9]	相机内参矩阵
R	double[9]	矫正矩阵
P	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

8.2.7 CompressedImage —压缩图像数据结构

字段名	类型	描述
header	Header	消息头
format	std::string	压缩格式（如 jpeg, png）
data	std::vector<uint8_t>	压缩后的图像数据

8.2.8 LaserScan —激光雷达数据结构

字段名	类型	描述
header	Header	消息头
angle_min	int32_t	起始角度（弧度）
angle_max	int32_t	结束角度（弧度）
angle_increment	int32_t	角度增量（弧度）
time_increment	int32_t	时间增量（秒）
scan_time	int32_t	扫描时间（秒）
range_min	int32_t	最小距离（米）
range_max	int32_t	最大距离（米）
ranges	std::vector<double>	距离数据数组（单位：米）
intensities	std::vector<double>	强度数据数组（无量纲）

8.2.9 MultiArrayDimension —多维数组维度描述

字段名	类型	描述
label	std::string	维度标签
size	int32_t	维度大小
stride	int32_t	步长

8.2.10 MultiArrayLayout —多维数组布局描述

字段名	类型	描述
dim_size	int32_t	维度数量
dim	std::vector<MultiArrayDimension>	维度数组
data_offset	int32_t	数据偏移量

8.2.11 Float32MultiArray —浮点数多维数组

字段名	类型	描述
layout	MultiArrayLayout	数组布局信息
data	std::vector<float>	浮点数数据数组

8.2.12 Int8 —8 位整数数据结构

字段名	类型	描述
`data`	`int8_t`	8 位整数数据

8.2.13 HeadTouch —头部触摸数据结构

项目	内容
类型	`Int8`
描述	头部触摸传感器数据，使用 8 位整数格式

8.3 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 9

语音控制服务 C++

提供机器人系统语音服务控制器，通过 AudioController 可以使用 RPC 方式实现对机器人音频的指令控制和状态获取。

9.1 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、查询，订阅语音原始数据等场景。

9.1.1 AudioController

项目	内容
函数名	AudioController
函数声明	`AudioController();`
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。

9.1.2 ~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。

9.1.3 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。

9.1.4 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。

9.1.5 GetVoiceConfig

项目	内容
函数名	GetVoiceConfig
函数声明	`Status GetVoiceConfig(GetSpeechConfig& config, int timeout_ms);`
功能概述	获取语音系统的完整配置信息。
参数说明	`config`：通过引用返回语音系统配置信息。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，获取的配置包含音色、智能体、唤醒、对话等所有子配置，以及当前使用的语音模型类型。

9.1.6 SwitchTtsVoiceModel

项目	内容
函数名	SwitchTtsVoiceModel
函数声明	`Status SwitchTtsVoiceModel(TtsType tts_type, int timeout_ms);`
功能概述	切换 TTS 语音模型。
参数说明	`tts_type`：TTS 语音模型类型 `timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	操作状态。
备注	切换模型属于阻塞操作。

9.1.7 SetVoiceConfig

项目	内容
函数名	SetVoiceConfig
函数声明	`Status SetVoiceConfig(const SetSpeechConfig& config, int timeout_ms);`
功能概述	设置语音系统的完整配置信息。
参数说明	`config`：要设置的语音系统配置信息。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，设置的配置将完全覆盖当前所有语音相关配置。

9.1.8 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。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

9.1.9 Stop

项目	内容
函数名	Stop
函数声明	`Status Stop(int timeout_ms);`
功能概述	停止当前音频播放。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

9.1.10 SetVolume

项目	内容
函数名	SetVolume
函数声明	`Status SetVolume(int volume, int timeout_ms);`
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。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

9.1.11 GetVolume

项目	内容
函数名	GetVolume
函数声明	`Status GetVolume(int& volume, int timeout_ms);`
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。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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`。

9.1.12 ControlVoiceStream

项目	内容
函数名	ControlVoiceStream
函数声明	`Status ControlVoiceStream(bool enable_raw_data, bool enable_bf_data, int timeout_ms);`
功能概述	控制语音数据流。
参数说明	`enable_raw_data`：是否开启原始语音数据流。`enable_bf_data`：是否开启 BF 语音数据流。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	操作状态。`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于控制语音数据流的开启和关闭。

9.1.13 SubscribeOriginVoiceData

项目	内容
函数名	SubscribeOriginVoiceData
函数声明	`void SubscribeOriginVoiceData(const ByteMultiArrayCallback callback);`
功能概述	订阅原始语音数据
参数说明	callback：接收到原始语音数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

9.1.14 UnsubscribeOriginVoiceData

项目	内容
函数名	UnsubscribeOriginVoiceData
函数声明	`void UnsubscribeOriginVoiceData();`
功能概述	取消订阅原始语音数据
备注	非阻塞接口，用于停止接收原始语音数据。

9.1.15 SubscribeBfVoiceData

项目	内容
函数名	SubscribeBfVoiceData
函数声明	`void SubscribeBfVoiceData(const ByteMultiArrayCallback callback);`
功能概述	订阅 BF 语音数据
参数说明	callback：接收到 BF 语音数据后的处理回调
备注	非阻塞接口，回调函数会在数据更新时被调用。

9.1.16 UnsubscribeBfVoiceData

项目	内容
函数名	UnsubscribeBfVoiceData
函数声明	`void UnsubscribeBfVoiceData();`
功能概述	取消订阅 BF 语音数据
备注	非阻塞接口，用于停止接收 BF 语音数据。

9.1.17 ControlSpeechIO

项目	内容
函数名	ControlSpeechIO
函数声明	`Status ControlSpeechIO(bool enable_data, int timeout_ms);`
功能概述	控制语音 io 数据流 (asr 和 tts)。
参数说明	`enable_data`：打开或关闭语音 io 数据流。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于控制语音 io 数据流 (asr 和 tts)。

9.1.18 SubscribeSpeechASRStream

项目	内容
函数名	SubscribeSpeechASRStream
函数声明	`void SubscribeSpeechASRStream(const SpeechASRStreamCallback callback);`
功能概述	订阅语音 asr 流。
参数说明	`callback`：处理接收到的语音 asr 流的回调。
备注	阻塞接口，用于订阅语音 asr 流。

9.1.19 UnsubscribeSpeechASRStream

项目	内容
函数名	UnsubscribeSpeechASRStream
函数声明	`void UnsubscribeSpeechASRStream();`
功能概述	取消订阅语音 asr 流。
备注	阻塞接口，用于取消订阅语音 asr 流。

9.1.20 PublishSpeechTTSStream

项目	内容
函数名	PublishSpeechTTSStream
函数声明	`Status PublishSpeechTTSStream(const SpeechTTSStream& data);`
功能概述	发布语音 tts 流。
参数说明	`data`：语音 tts 流。
备注	阻塞接口，用于发布语音 tts 流。数据 ID 应该与 ASR 输出的请求 ID 一致，起始类型为 begin，中间数据类型为 var，结束类型为 end。

9.2 类型定义

9.2.1 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	数值	描述
`TtsPriority::HIGH`	0	最高优先级，例如：低电告警、紧急提醒
`TtsPriority::MIDDLE`	1	中优先级，例如：系统提示、状态播报
`TtsPriority::LOW`	2	最低优先级，例如：日常语音对话、背景播报

9.2.2 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	数值	描述
`TtsMode::CLEARTOP`	0	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`	1	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`	2	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

9.3 结构体定义

9.3.1 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

9.3.2 CustomBotInfo —自定义智能体配置结构体

描述一个自定义智能体的基本配置信息。

字段名	类型	描述
`name`	`std::string`	智能体名称
`workflow`	`std::string`	工作流 ID
`token`	`std::string`	用户授权 token

9.3.3 CustomBotMap —自定义智能体映射表

项目	内容
类型	std::map<std::string, CustomBotInfo>
描述	自定义智能体映射表，键为智能体 ID，值为智能体配置信息

9.3.4 SetSpeechConfig —语音配置参数结构体

用于设置语音系统的各项配置参数。

字段名	类型	描述
speaker_id	std::string	音色 ID
region	std::string	音色地区
bot_id	std::string	模式 ID
is_front_doa	bool	是否强制正前方识别
is_fullduplex_enable	bool	自然对话开关
is_enable	bool	语音开关
is_doa_enable	bool	是否打开唤醒方位转头
speaker_speed	double	TTS 播放语速，范围 [1,2]
wakeup_name	std::string	唤醒名字
custom_bot	CustomBotMap	自定义智能体配置

9.3.5 SpeakerConfigSelected —音色选中配置结构体

当前音色配置

字段名	类型	描述
region	std::string	选中的地区
speaker_id	std::string	选中的音色 ID

9.3.6 SpeakerConfig —音色配置结构体

音色配置

字段名	类型	描述
data	std::map<std::string, std::vector<std::array<std::string, 2>>>	音色数据：地区-> 音色 ID-> 音色名称
selected	SpeakerConfigSelected	当前选中的音色配置
speaker_sp	double	语速

9.3.7 BotInfo —智能体配置信息结构体

描述一个标准智能体的基本信息。

字段名	类型	描述
name	std::string	工作场景名称
workflow	std::string	工作流 ID

BotConfigSelected —智能体选中项结构体

字段名	类型	描述
bot_id	std::string	选中的智能体 ID

9.3.8 BotConfig —智能体配置结构体

描述智能体相关的配置信息，包括标准智能体和自定义智能体。

字段名	类型	描述
data	std::map<std::string, BotInfo>	标准智能体数据：智能体 ID-> 智能体信息
custom_data	std::map<std::string, CustomBotInfo>	自定义智能体数据：智能体 ID-> 自定义智能体信息
selected	BotConfigSelected	当前选中的智能体配置

9.3.9 WakeupConfig —唤醒配置结构体

描述唤醒相关的配置信息。

字段名	类型	描述
name	std::string	唤醒名称
data	std::map<std::string, std::string>	唤醒词数据：唤醒词-> 拼音

9.3.10 DialogConfig —对话配置结构体

描述对话相关的配置参数。

字段名	类型	描述
is_front_doa	bool	是否强制正前方加强拾音
is_fullduplex_enable	bool	是否打开全双工对话
is_enable	bool	是否打开语音
is_doa_enable	bool	是否打开唤醒方位转头

9.3.11 TtsType —语音模型枚举

描述可选的语音合成模型类型。

枚举值	数值	描述
TtsType::NONE	0	无语音模型
TtsType::DOUBAO	1	豆包语音模型
TtsType::GOOGLE	2	谷歌语音模型

9.3.12 GetSpeechConfig —语音系统完整配置结构体

描述语音系统的完整配置信息，包含所有子配置模块。

字段名	类型	描述
speaker_config	SpeakerConfig	音色配置
bot_config	BotConfig	智能体配置
wakeup_config	WakeupConfig	唤醒配置
dialog_config	DialogConfig	对话配置
tts_type	TtsType	语音模型，默认值为 TtsType::NONE

9.3.13 MultiArrayDimension —多维数组维度描述

字段名	类型	描述
label	std::string	维度标签
size	int32_t	维度大小
stride	int32_t	步长

9.3.14 MultiArrayLayout —多维数组布局描述

字段名	类型	描述
dim_size	int32_t	维度数量
dim	std::vector<MultiArrayDimension>	维度数组
data_offset	int32_t	数据偏移量

9.3.15 ByteMultiArray —字节数组数据结构

字段名	类型	描述
layout	MultiArrayLayout	数组布局信息
data	std::vector<uint8_t>	字节数据数组

9.3.16 SpeechASRStream —机器人 ASR 输出

字段名	类型	描述
id	std::string	ID - 请求 id
type	std::string	Type - 类型 request or cancel
text	std::string	Text - ASR 输出

9.3.17 SpeechTTSStream —机器人 TTS 输入

字段名	类型	描述
id	std::string	ID - 与请求 id 相同
type	std::string	Type - 类型 begin or var or end
text	std::string	Text - TTS 输入
end_session	bool	End Session - 结束会话并关闭 asr

9.4 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 10

状态监控服务 C++

提供机器人状态监控接口类，通过 StateMonitor 可以实现状态查询等功能。

10.1 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。

10.1.1 GetCurrentState

项目	内容
函数名	GetCurrentState
函数声明	`Status GetCurrentState(RobotState& robot_state);`
功能概述	获取当前机器人聚合状态。
参数说明	`robot_state`：用于接收状态结构体。
返回值	`Status::OK` 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 电池状态、故障信息等，后续会持续扩展。

10.1.2 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	std::vector<Fault>	故障信息列表
bms_data	BmsData	电池管理系统数据

10.1.3 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	double	当前电池剩余电量（百分比，0~100）
battery_health	double	电池健康状态（数值越高表示健康状态越好）
battery_state	BatteryState	电池状态（见上方枚举）
power_supply_status	PowerSupplyStatus	电池充放电状态（见上方枚举）

10.1.4 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

10.2 错误码映射表

错误码（十六进制）	错误描述
0x0000	无错误
0x1101	调用 ROS 服务失败
0x1301	中控节点丢失
0x1302	APP 节点丢失
0x1304	音频节点丢失
0x1305	运动节点丢失
0x1306	LCD 节点丢失
0x1307	realsense 节点丢失
0x1308	双目相机节点丢失
0x1309	LDS 节点丢失
0x130A	传感器板节点丢失
0x130B	触摸节点丢失
0x130C	SLAM 节点丢失
0x130D	导航节点丢失
0x130E	AI 节点丢失
0x130F	头部节点丢失
0x1310	云端处理器节点丢失
0x3201	激光无数据
0x3202	双目相机无数据
0x3203	双目相机数据错误
0x3204	双目相机初始化失败
0x320B	里程计无数据
0x320C	IMU 无数据
0x6101	机器人连接 APP 失败
0x6102	与 APP 断开连接
0x9201	打开 LCD 串口失败
0x7201	打开头部串口失败

下页继续

表 1 – 续上页

错误码（十六进制）	错误描述
0x7202	头部无数据
0x8201	打开传感器板串口失败
0x8202	传感器板无数据
0x2201	错误：导航未收到 tf 数据
0x2202	错误：导航未收到地图数据
0x2203	错误：导航未收到定位数据
0x2204	错误：导航未收到超声波数据
0x2205	错误：导航未收到激光数据
0x2206	错误：导航未收到 RGBD 数据
0x2207	错误：导航未收到多线激光数据
0x2208	错误：导航未收到点 TOF 数据
0x2209	错误：导航未收到面 TOF 数据
0x220A	错误：导航未收到里程计数据
0x2101	警告：导航未收到 tf 数据
0x2102	警告：导航未收到地图数据
0x2103	警告：导航未收到定位数据
0x2104	警告：导航未收到超声波数据
0x2105	警告：导航未收到激光数据
0x2106	警告：导航未收到 RGBD TOF 数据
0x2107	警告：导航未收到多线激光数据
0x2108	警告：导航未收到点 TOF 数据
0x2109	警告：导航未收到面 TOF 数据
0x210A	警告：导航未收到里程计数据
0x4201	SLAM 定位错误
0x4102	错误：SLAM 未收到激光数据
0x4103	错误：SLAM 未收到里程计数据
0x4205	SLAM 地图错误

CHAPTER 11

SLAM 导航控制服务 C++

提供机器人系统 SLAM 导航控制服务，通过 SlamNavController 可以实现 SLAM 建图、定位、导航等功能。

11.1 接口定义

SlamNavController 是面向 SLAM 导航控制的控制器，支持建图、定位、导航等操作。

11.1.1 SlamNavController

项目	内容
函数名	SlamNavController
函数声明	`SlamNavController();`
功能概述	构造函数，创建 SlamNavController 实例。
备注	构造内部控制资源。

11.1.2 ~SlamNavController

项目	内容
函数名	~SlamNavController
函数声明	`~SlamNavController();`
功能概述	析构函数，释放 SlamNavController 实例资源。
备注	配合构造函数使用。

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 SLAM 导航控制器。
返回值	`true` 表示成功，`false` 表示失败。
备注	首次使用前必须调用。

11.1.4 Shutdown

项目	内容
函数名	Shutdown
函数声明	`void Shutdown();`
功能概述	释放资源，清理 SLAM 导航控制器。
备注	配合 Initialize 使用。

11.1.5 SwitchToIdle

项目	内容
函数名	SwitchToIdle
函数声明	`Status SwitchToIdle();`
功能概述	切换到空闲模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换到空闲状态。

11.1.6 StartMapping

项目	内容
函数名	StartMapping
函数声明	`Status StartMapping();`
功能概述	开始建图模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于启动建图任务。

11.1.7 CancelMapping

项目	内容
函数名	CancelMapping
函数声明	`Status CancelMapping();`
功能概述	取消当前建图任务。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于取消建图任务。

11.1.8 SwitchToLocation

项目	内容
函数名	SwitchToLocation
函数声明	`Status SwitchToLocation();`
功能概述	切换到定位模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换到定位状态。

11.1.9 SaveMap

项目	内容
函数名	SaveMap
函数声明	`Status SaveMap(const std::string& map_name);`
功能概述	结束建图并保存地图，当处于建图模式时。
参数说明	`map_name`：地图名称。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于保存当前建图结果。

11.1.10 LoadMap

项目	内容
函数名	LoadMap
函数声明	`Status LoadMap(const std::string& map_name);`
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。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于加载已保存的地图。

11.1.11 DeleteMap

项目	内容
函数名	DeleteMap
函数声明	`Status DeleteMap(const std::string& map_name);`
功能概述	删除地图。
参数说明	`map_name`：要删除的地图名称。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于删除不需要的地图。

11.1.12 GetAllMapInfo

项目	内容
函数名	GetAllMapInfo
函数声明	`Status GetAllMapInfo(AllMapInfo& all_map_info);`
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）。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于获取系统中所有地图的列表。

11.1.13 InitPose

项目	内容
函数名	InitPose
函数声明	`Status InitPose(const Pose3DEuler& pose);`
功能概述	初始化位姿。
参数说明	`pose`：要发布的位姿信息。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置机器人的初始位姿。

11.1.14 GetCurrentLocalizationInfo

项目	内容
函数名	GetCurrentLocalizationInfo
函数声明	`Status GetCurrentLocalizationInfo(LocalizationInfo& localization_info);`
功能概述	获取当前位姿信息。
参数说明	`localization_info`：当前位置和姿态信息（输出参数）。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于获取机器人当前位姿。

11.1.15 ActivateNavMode

项目	内容
函数名	ActivateNavMode
函数声明	`Status ActivateNavMode(NavMode mode);`
功能概述	激活导航模式。
参数说明	`mode`：目标导航模式（NavMode 枚举类型）。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换导航工作模式。

11.1.16 SetNavTarget

项目	内容
函数名	SetNavTarget
函数声明	`Status SetNavTarget(const NavTarget& goal);`
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。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置导航目标。

11.1.17 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	阻塞接口，用于暂停导航。

11.1.18 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	阻塞接口，用于恢复导航。

11.1.19 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	阻塞接口，用于取消导航。

11.1.20 GetNavTaskStatus

项目	内容
函数名	GetNavTaskStatus
函数声明	`/** * @brief 获取当前导航任务状态 * @param status 导航状态 * @return 操作状态，成功返回 Status::OK */ Status GetNavTaskStatus(NavStatus& status);`
功能概述	获取当前导航任务状态。
参数说明	`status`：导航状态。
返回值	操作状态，成功返回 `Status::OK`。
备注	阻塞接口，用于查询导航任务状态。

11.1.21 SubscribeOdometry

项目	内容
函数名	SubscribeOdometry
函数声明	`Status SubscribeOdometry(OdometryCallback callback);`
功能概述	订阅机器人里程计数据。
参数说明	`callback`：里程计数据回调函数。
返回值	`Status::OK` 表示成功，其他为失败。
备注	非阻塞接口，用于获取实时里程计数据。

11.1.22 UnsubscribeOdometry

项目	内容
函数名	UnsubscribeOdometry
函数声明	`void UnsubscribeOdometry();`
功能概述	取消订阅机器人里程计数据。
返回值	无
备注	`/** @brief Unsubscribe from odometry data */`

11.1.23 GetPointCloudMap

项目	内容
函数名	GetPointCloudMap
函数声明	`Status GetPointCloudMap(PointCloud2&amp; point_cloud_map, int timeout_ms);`
功能概述	获取当前构建的点云地图。
参数说明	`point_cloud_map`：输出参数，获取到的点云地图数据。`timeout_ms`：操作超时 (毫秒)，默认 10000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，若在超时时间内未完成则返回错误状态。

11.2 类型定义

11.2.1 NavMode —导航模式枚举类型

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

11.2.2 Pose3DEuler —3D 位姿结构

字段名	类型	描述
`position`	`std::array&lt;double, 3&gt;`	位置 (x, y, z)，用于表示空间位置
`orientatior`	`std::array&lt;double, 3&gt;`	欧拉角 (roll, pitch, yaw)，用于表示空间姿态，避免欧拉角万向锁问题

11.2.3 NavTarget —全局导航目标点结构

字段名	类型	描述
`id`	`int32_t`	目标点 ID
`frame_id`	`std::string`	目标点坐标系 ID
`goal`	`Pose3DEuler`	目标点位姿

11.2.4 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	结束成功
END_FAILED	3	结束失败
PAUSE	4	暂停
CONTINUE	5	继续
CANCEL	6	取消

11.2.5 NavStatus —导航状态结构

字段名	类型	描述
id	int32_t	目标点 ID，-1 表示无目标点
status	NavStatusType	导航状态
message	std::string	导航状态消息

11.2.6 MapImageData —建图图像数据结构，.pgm 格式

字段名	类型	描述
type	std::string	魔数，”P5”：二进制格式
width	uint32_t	图像宽度
height	uint32_t	图像高度
max_gray_value	uint32_t	最大灰度值，255
image	std::vector<uint8_t>	图像数据

11.2.7 MapMetaData —建图地图元数据结构

字段名	类型	描述
resolution	double	地图分辨率，单位：m/pixel
origin	Pose3DEuler	地图原点，世界坐标系原点相对于地图左下角的位置
map_image_data	MapImageData	图像数据，.pgm 格式图像数据

11.2.8 MapInfo —单个地图信息结构

字段名	类型	描述
map_name	std::string	地图名称
map_meta_data	MapMetaData	地图元数据

11.2.9 AllMapInfo —所有地图信息结构

字段名	类型	描述
current_map_name	std::string	当前地图名称
map_infos	std::vector<MapInfo>	所有地图信息

11.2.10 LocalizationInfo —当前位置信息结构

字段名	类型	描述
is_localization	bool	是否已定位
pose	Pose3DEuler	欧拉角位姿

11.3 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。

11.3.1 适用场景与范围

- 小于 20m * 20m 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

11.3.2 SLAM 功能流程

功能	操作流程
建图	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

11.3.3 导航功能流程

功能	操作流程
定位	加载地图 → 切换到定位模式 → 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

CHAPTER 12

显示控制服务 C++

提供机器人系统显示服务控制器，通过 DisplayController 可以使用 RPC 方式实现对机器人显示的指令控制和状态获取。

12.1 接口定义

DisplayController 是一个封装显示控制功能的 C++ 类，主要用于显示控制等场景。

12.1.1 DisplayController

项目	内容
函数名	DisplayController
函数声明	`DisplayController();`
功能概述	初始化显示控制器对象，构造内部状态，分配资源等。
备注	构造内部状态。

12.1.2 ~DisplayController

项目	内容
函数名	~DisplayController
函数声明	`~DisplayController();`
功能概述	释放显示控制器资源。
备注	确保资源安全释放。

12.1.3 Initialize

项目	内容
函数名	Initialize
函数声明	`bool Initialize();`
功能概述	初始化显示控制模块。
返回值	`true` 表示成功，`false` 表示失败。
备注	与 `Shutdown()` 配对使用。

12.1.4 Shutdown

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

12.1.5 GetAllFaceExpressions

项目	内容
函数名	GetAllFaceExpressions
函数声明	`Status GetAllFaceExpressions(std::vector& data, int timeout_ms);`
功能概述	获取所有面部表情。
参数说明	`data`：所有面部表情。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，获取所有面部表情。

12.1.6 SetFaceExpression

项目	内容
函数名	SetFaceExpression
函数声明	`Status SetFaceExpression(int expression_id, int timeout_ms);`
功能概述	设置面部表情。
参数说明	`expression_id`：表情 ID`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，设置面部表情。

12.1.7 GetFaceExpression

项目	内容
函数名	GetFaceExpression
函数声明	`Status GetFaceExpression(FaceExpression& data, int timeout_ms);`
功能概述	获取当前面部表情。
参数说明	`data`：当前面部表情。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，获取当前面部表情。

12.2 结构体定义

12.2.1 FaceExpression —机器人面部表情结构

描述机器人面部表情的完整信息，支持唯一 ID、名称和描述。

字段名	类型	描述
`id`	`int32_t`	机器人面部表情 ID
`name`	`std::string`	机器人面部表情名称
`description`	`std::string`	机器人面部表情描述

12.3 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 13

机器人主控制服务 Python

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、音频控制器、状态监控以及传感器控制器等。

⚠ **注意事项：使用 Python 接口时，请确保已正确构建 Python 绑定模块，并注意内存管理和线程安全**。

13.1 模块信息

magicdog_python 是 MagicDog SDK 的 Python 绑定模块，提供了完整的机器人控制、传感器数据获取、音频处理等功能接口。

项目	内容
模块名	`magicdog_python`
描述	MagicDog SDK Python 绑定模块
依赖	pybind11, C++ SDK

13.2 接口定义

MagicRobot 是机器人系统的统一入口类。

13.2.1 MagicRobot —主机器人类

项目	内容
类名	`MagicRobot`
功能概述	机器人主控制类，整合所有功能模块
主要功能	系统初始化、连接管理、控制器获取
使用场景	机器人应用开发、系统集成、功能测试

13.2.2 initialize

项目	内容
方法名	`initialize`
方法声明	`bool initialize(const 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。

13.2.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	关闭机器人系统，释放资源。
备注	配合 initialize 使用。

13.2.4 release

项目	内容
方法名	`release`
方法声明	`void release()`
功能概述	释放机器人系统资源。
备注	调用该方法可手动释放 SDK 内部占用的所有资源，通常在程序退出前调用。

13.2.5 connect

项目	内容
方法名	`connect`
方法声明	`Status connect(int timeout_ms)`
功能概述	与机器人服务建立通信连接。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功。
备注	阻塞接口，需初始化后调用。

13.2.6 disconnect

项目	内容
方法名	`disconnect`
方法声明	`Status disconnect(int timeout_ms)`
功能概述	断开与机器人服务的连接。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功。
备注	阻塞接口，与 connect 配对使用。

13.2.7 get_sdk_version

项目	内容
方法名	`get_sdk_version`
方法声明	`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。

13.2.8 get_motion_control_level

项目	内容
方法名	`get_motion_control_level`
方法声明	`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。

13.2.9 set_motion_control_level

项目	内容
方法名	`set_motion_control_level`
方法声明	`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::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。

13.2.10 get_high_level_motion_controller

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

13.2.11 get_low_level_motion_controller

项目	内容
方法名	`get_low_level_motion_controller`
方法声明	`LowLevelMotionController get_low_level_motion_controller()`
功能概述	获取低层运动控制器对象。
返回值	低层运动控制器实例，用于控制关节或电机。
备注	非阻塞接口，可直接控制关节电机等。

13.2.12 get_audio_controller

项目	内容
方法名	`get_audio_controller`
方法声明	`AudioController get_audio_controller()`
功能概述	获取音频控制器对象。
返回值	音频控制器实例，用于语音控制。
备注	非阻塞接口，可播放语音和控制音量。

13.2.13 get_sensor_controller

项目	内容
方法名	`get_sensor_controller`
方法声明	`SensorController get_sensor_controller()`
功能概述	获取传感器控制器对象。
返回值	传感器控制器实例，用于访问传感器数据。
备注	非阻塞接口，封装了 IMU、RGBD 等传感器数据读取。

13.2.14 get_state_monitor

项目	内容
方法名	`get_state_monitor`
方法声明	`StateMonitor get_state_monitor()`
功能概述	获取状态监控器对象。
返回值	状态监控器实例，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。

13.2.15 get_slam_nav_controller

项目	内容
方法名	`get_slam_nav_controller`
方法声明	`SlamNavController get_slam_nav_controller()`
功能概述	获取 SLAM/导航控制器对象。
返回值	SLAM/导航控制器实例，用于访问地图、定位等 SLAM 相关数据。
备注	非阻塞接口，封装地图构建、定位、导航等相关功能的使用。

13.2.16 open_channel_switch

项目	内容
方法名	`open_channel_switch`
方法声明	`Status open_channel_switch(int timeout_ms)`
功能概述	打开 lcm 通道开关。
参数说明	`timeout_ms`（可选参数，单位毫秒），操作超时时间，默认 5000 ms。
返回值	`Status` 结构体，表示操作状态。
备注	用于控制 SDK lcm 通信通道的启用。

13.2.17 close_channel_switch

项目	内容
方法名	`close_channel_switch`
方法声明	`Status close_channel_switch(int timeout_ms)`
功能概述	关闭 lcm 通道开关。
参数说明	`timeout_ms`（可选参数，单位毫秒），操作超时时间，默认 5000 ms。
返回值	`Status` 结构体，表示操作状态。
备注	用于控制 SDK lcm 通信通道的关闭。

13.3 常量定义

常量名	类型	描述
`LEG_JOINT_NUM`	`int`	腿部关节数量
`PERIOD_MS`	`int`	控制周期（毫秒）

13.4 数据结构定义

13.4.1 Status —状态返回结构体

字段名	类型	描述
`code`	`ErrorCode`	错误代码
`message`	`str`	状态消息

13.4.2 ErrorCode —错误代码枚举

枚举值	数值	描述
`OK`	0	操作成功
`SERVICE_NOT_READY`	1	服务未就绪
`TIMEOUT`	2	操作超时
`INTERNAL_ERROR`	3	内部错误
`SERVICE_ERROR`	4	服务错误

CHAPTER 14

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。

⚠ **Notice:** 某些特技动作，比如前空翻、后空翻对场地要求较大，执行时需要额外注意，建议所有特技动作在大的空场地测试后，再在小的场地使用。

14.1 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、特技等控制操作，封装底层细节以供上层系统调用。

14.1.1 HighLevelMotionController —高级运动控制器

项目	内容
类名	`HighLevelMotionController`
功能概述	面向语义控制的高层运动控制器，支持行走、特技等控制操作
主要功能	步态控制、特技执行、摇杆命令发送
使用场景	机器人高层运动控制、特技表演、遥控操作

14.1.2 initialize

项目	内容
方法名	`initialize`
方法声明	`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。

14.1.3 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。

14.1.4 set_gait

项目	内容
方法名	`set_gait`
方法声明	`Status set_gait(GaitMode gait_mode, int timeout_ms)`
功能概述	设置机器人步态模式（如位控站立、力控站立、慢跑等）。
参数说明	`gait_mode`：步态控制枚举。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，支持多种步态模式切换。

14.1.5 get_gait

项目	内容
方法名	`get_gait`
方法声明	`GaitMode get_gait(int timeout_ms)`
功能概述	获取机器人步态模式（如位控站立、力控站立、慢跑等）。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	当前步态模式，失败返回 `GAIT_NONE`。
备注	阻塞接口，获取当前步态模式。

14.1.6 execute_trick

项目	内容
方法名	`execute_trick`
方法声明	`Status execute_trick(TrickAction trick_action, int timeout_ms)`
功能概述	执行特技动作（如扭屁股、趴下等）。
参数说明	`trick_action`：特技动作标识。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，需确保机器人当前处于可执行特技的状态。

注意事项：特技动作必须在 GaitMode::GAIT_STAND_R(2) 位控站立步态下才能执行

14.1.7 enable_joy_stick

方法名	内容
方法名	`enable_joy_stick`
方法声明	`Status enable_joy_stick()`
功能概述	开启摇杆控制命令。
参数说明	无。
返回值	`Status.OK` 表示操作成功，其他为失败。
备注	用于允许通过摇杆发送实时控制指令。

14.1.8 disable_joy_stick

方法名	内容
方法名	`disable_joy_stick`
方法声明	`Status disable_joy_stick()`
功能概述	关闭摇杆控制命令。
参数说明	无。
返回值	`Status.OK` 表示操作成功，其他为失败。
备注	切换导航模式之前，必须先关闭摇杆控制命令。

14.1.9 send_joystick_command

项目	内容
方法名	`send_joystick_command`
方法声明	`Status send_joystick_command(JoystickCommand command)`
功能概述	发送实时摇杆控制指令。
参数说明	`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。

14.1.10 get_all_gait_speed_ratio

项目	内容
方法名	`get_all_gait_speed_ratio`
方法声明	`AllGaitSpeedRatio get_all_gait_speed_ratio(int timeout_ms)`
功能概述	获取所有步态以及对应前进、横移、旋转速度比例。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	成功返回 `AllGaitSpeedRatio` 对象，失败返回空对象。
备注	阻塞接口，用于获取所有步态的速度比例配置。

14.1.11 set_gait_speed_ratio

项目	内容
方法名	`set_gait_speed_ratio`
方法声明	`Status set_gait_speed_ratio(GaitMode gait_mode, GaitSpeedRatio gait_speed_ratio, int timeout_ms)`
功能概述	设置步态以及对应前进、横移、旋转速度比例。
参数说明	`gait_mode`：步态模式；`gait_speed_ratio`：速度比例配置。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置特定步态的速度比例配置。

14.1.12 get_head_motor_enabled

项目	内容
方法名	`get_head_motor_enabled`
方法声明	`bool get_head_motor_enabled(int timeout_ms)`
功能概述	获取头部电机使能状态。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	成功返回 `true`（已使能）或 `false`（未使能），失败返回 `false`。
备注	阻塞接口，用于查询头部电机的使能状态。

14.1.13 enable_head_motor

项目	内容
方法名	`enable_head_motor`
方法声明	`Status enable_head_motor(int timeout_ms)`
功能概述	使能头部电机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于启用头部电机控制。

14.1.14 disable_head_motor

项目	内容
方法名	`disable_head_motor`
方法声明	`Status disable_head_motor(int timeout_ms)`
功能概述	关闭头部电机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于禁用头部电机控制。

14.1.15 get_current_head_position

项目	内容
方法名	get_current_head_position
方法声明	`Status get_current_head_position(EulerAngles euler_angles, int timeout_ms);`
功能概述	获取头部位置。
参数说明	`euler_angles`：返回参数，头部当前绝对位置。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于获取头部位置。

14.1.16 set_head_position

项目	内容
方法名	set_head_position
方法声明	`Status set_head_position(EulerAngles euler_angles, int timeout_ms);`
功能概述	设置头部位置。
参数说明	`euler_angles`：头部目标绝对位置。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置头部位置。

14.2 枚举类型定义

14.2.1 GaitMode —步态模式枚举

枚举值	数值	描述
GAIT_PASSIVE	0	掉落（关闭电机使能）
GAIT_STAND_R	2	位控站立、RecoveryStand
GAIT_STAND_B	3	力控站立、姿态展示、BalanceStand
GAIT_RUN_FAST	8	快跑
GAIT_DOWN_CLIMB_STAIRS	9	下爬楼梯 => 盲走 => 慢跑
GAIT_TROT	10	小跑
GAIT_PRONK	11	跳跃
GAIT_BOUND	12	前后跳
GAIT_AMBLE	14	交叉步
GAIT_CRAWL	29	爬行
GAIT_LOWLEVL_SDK	30	低层 SDK 步态
GAIT_WALK	39	缓走
GAIT_UP_CLIMB_STAIRS	56	上爬楼梯（全地形）
GAIT_RL_TERRAIN	110	全地形
GAIT_RL_FALL_RECOVERY	111	跌倒爬起
GAIT_RL_HAND_STAND	112	倒立
GAIT_RL_FOOT_STAND	113	正立
GAIT_ENTER_RL	1001	进入 RL
GAIT_DEFAULT	99	默认
GAIT_NONE	9999	无步态

14.2.2 TrickAction —特技动作枚举

枚举值	数值	描述
ACTION_NONE	0	无动作
ACTION_WIGGLE_HIP	26	扭屁股
ACTION_SWING_BODY	27	甩身
ACTION_STRETCH	28	伸懒腰
ACTION_STOMP	29	跺脚
ACTION_JUMP_JACK	30	开合跳
ACTION_SPACE_WALK	31	太空步
ACTION_IMITATE	32	模仿
ACTION_SHAKE_HEAD	33	摇头

下页继续

表 1 – 续上页

枚举值	数值	描述
ACTION_PUSH_UP	34	俯卧撑
ACTION_CHEER_UP	35	加油
ACTION_HIGH_FIVES	36	击掌
ACTION_SCRATCH	37	挠痒
ACTION_HIGH_JUMP	38	跳高
ACTION_SWING_DANCE	39	摇摆舞
ACTION_LEAP_FROG	40	蛙跳
ACTION_BACK_FLIP	41	后空翻
ACTION_FRONT_FLIP	42	前空翻
ACTION_SPIN_JUMP_LEFT	43	旋转左跳 70 度
ACTION_SPIN_JUMP_RIGHT	44	旋转右跳 70 度
ACTION_JUMP_FRONT	45	前跳 0.5 米
ACTION_ACT_CUTE	46	撒娇
ACTION_BOXING	47	打拳
ACTION_SIDE_SOMERSAULT	48	侧空翻
ACTION_RANDOM_DANCE	49	随机跳舞
ACTION_LEFT_SIDE_SOMERSAULT	84	左侧侧空翻
ACTION_RIGHT_SIDE_SOMERSAULT	85	右侧侧空翻
ACTION_DANCE2	91	跳舞 2
ACTION_EMERGENCY_STOP	101	急停
ACTION_LIE_DOWN	102	趴下
ACTION_RECOVERY_STAND	103	起立
ACTION_HAPPY_NEW_YEAR	105	拜年 = 作揖
ACTION_SLOW_GO_FRONT	108	过来
ACTION_SLOW_GO_BACK	109	后退
ACTION_BACK_HOME	110	回窝
ACTION_LEAVE_HOME	111	离窝
ACTION_TURN_AROUND	112	转圈
ACTION_DANCE	115	跳舞
ACTION_ROLL_ABOUT	116	打滚
ACTION_SHAKE_RIGHT_HAND	117	握右手
ACTION_SHAKE_LEFT_HAND	118	握左手
ACTION_SIT_DOWN	119	坐下

14.3 数据结构定义

14.3.1 GaitSpeedRatio —步态速度比例结构体

字段名	类型	描述
straight_ratio	double	前进速度比例
turn_ratio	double	旋转速度比例
lateral_ratio	double	横移速度比例

14.3.2 AllGaitSpeedRatio —所有步态速度比例结构体

字段名	类型	描述
gait_speed_ratios	GaitModeGaitSpeedRatioMap	步态模式到速度比例的映射

14.3.3 GaitModeGaitSpeedRatioMap —步态模式到速度比例映射类型

项目	内容
类型名	GaitModeGaitSpeedRatioMap
类型定义	std::map<GaitMode, GaitSpeedRatio>
功能概述	步态模式到速度比例配置的映射容器
键类型	GaitMode - 步态模式枚举
值类型	GaitSpeedRatio - 速度比例结构体
使用场景	存储和管理所有步态的速度比例配置

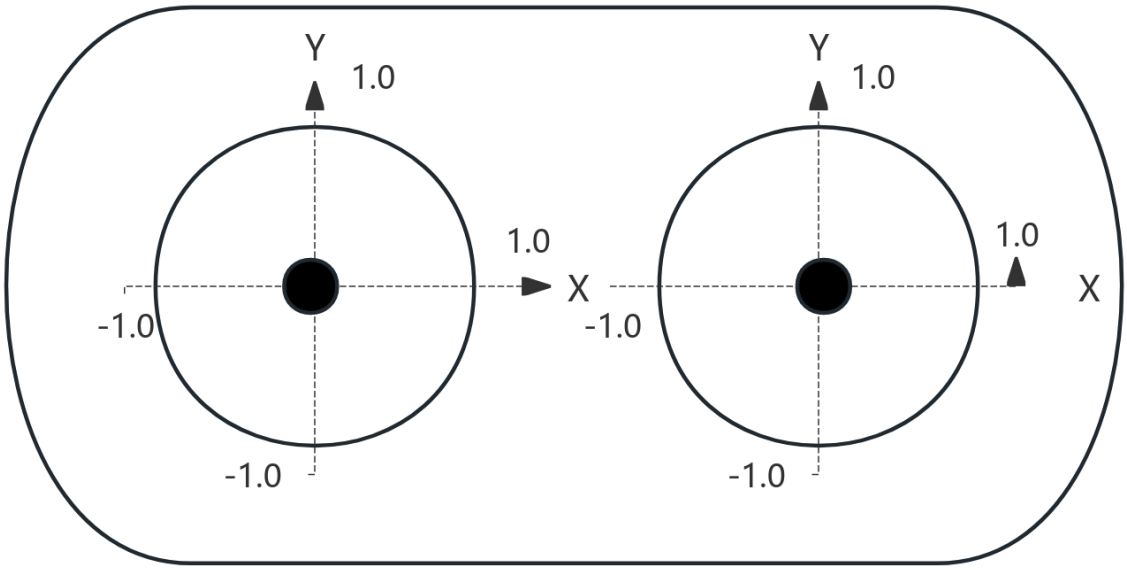
14.3.4 JoystickCommand —摇杆命令结构体

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）

14.3.5 EulerAngles —欧拉角结构体

字段名	类型	描述
roll	double	横滚角（Roll），绕 X 轴旋转的角度（单位为弧度）
pitch	double	俯仰角（Pitch），绕 Y 轴旋转的角度（单位为弧度）
yaw	double	偏航角（Yaw），绕 Z 轴旋转的角度（单位为弧度）

14.4 遥控器示意图



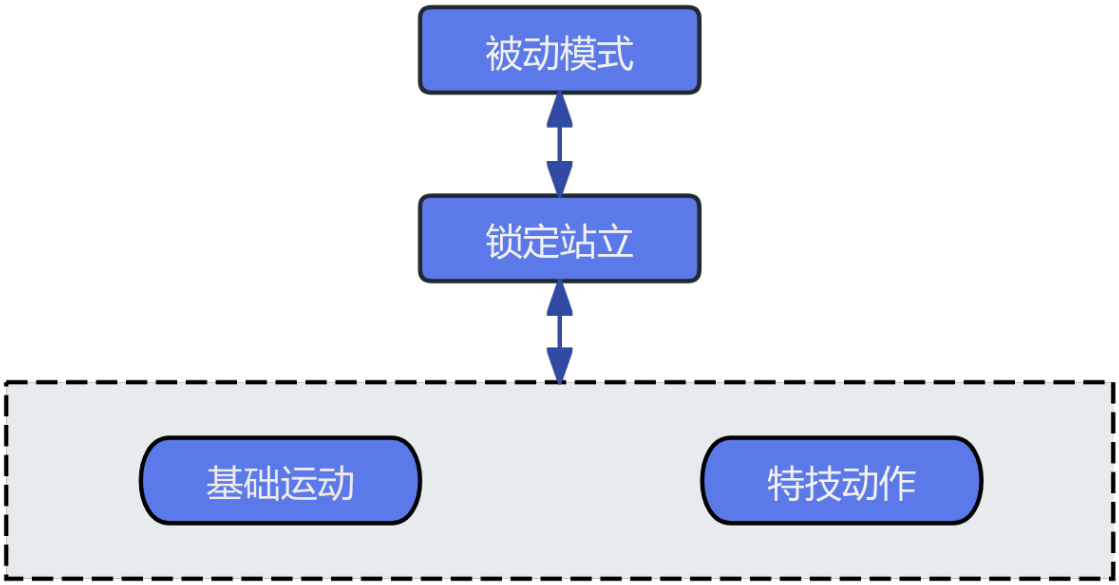
1. 左右摇杆 x 轴和 y 轴的取值范围为 [-1.0, 1.0];
2. 左右摇杆 x 轴和 y 轴的方向上/右为正，如示意图所示；

14.5 高层运动控制机器人状态介绍

机器人的运动包含位控站立、力控站立、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：

- 位控站立：在位控站立状态下，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。
- 力控站立：在力控站立状态下，可以用于姿态展示。
- 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。

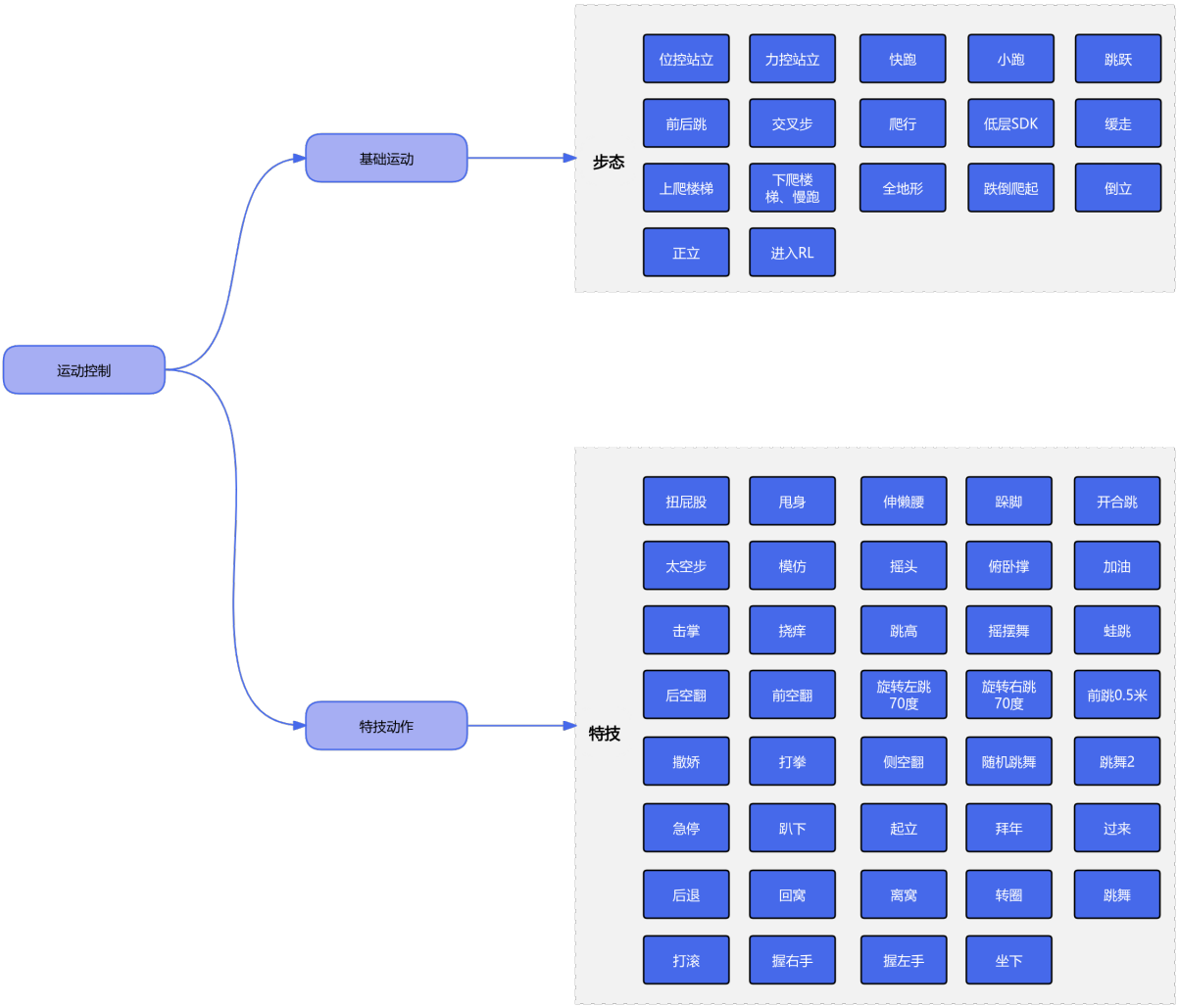
机器人状态切换机制；



14.6 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，比如扭屁股、趴下等。


运动控制
基础运动
特技动作
步态
位控站立
力控站立
快跑
小跑
跳跃
前后跳
交叉步
爬行
低层SDK
缓走
上爬楼梯
下爬楼梯、慢跑
全地形
跌倒爬起
倒立
正立
进入RL
特技
扭屁股
甩身
伸懒腰
跺脚
开合跳
太空步
模仿
摇头
俯卧撑
加油
击掌
挠痒
跳高
摇摆舞
蛙跳
后空翻
前空翻
旋转左跳70度
旋转右跳70度
前跳0.5米
撒娇
打拳
侧空翻
随机跳舞
跳舞2
急停
趴下
起立
拜年
过来
后退
回窝
离窝
转圈
跳舞
打滚
握右手
握左手
坐下

CHAPTER 15

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 可以使用话题通信方式实现对机器人关节的指令控制和状态获取。

15.1 接口定义

LowLevelMotionController 是面向底层开发的运动控制器，支持对腿等运动部件的直接控制与状态订阅。

> ⚠ **Notice:** 调用 publish_leg_command 接口，用户命令才会下发到运控，因此需要用户保证调用 publish_leg_command 的频率，建议是 500 hz。

15.1.1 LowLevelMotionController —低级运动控制器

项目	内容
类名	`LowLevelMotionController`
功能概述	底层关节级运动控制器，提供精确的关节控制
主要功能	关节状态订阅、关节命令发布、控制周期设置
使用场景	精确运动控制、自定义步态、研究开发

15.1.2 initialize

项目	内容
方法名	`initialize`
方法声明	`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。

15.1.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	关闭控制器，释放底层资源。
备注	配合 initialize 使用。

15.1.4 subscribe_leg_state

项目	内容
方法名	`subscribe_leg_state`
方法声明	`void subscribe_leg_state(callback)`
功能概述	订阅下肢关节状态数据。
参数说明	`callback`：回调函数，用于处理下肢关节状态的接收数据。函数签名：callback(data : LegState) -> None
备注	非阻塞接口。

15.1.5 unsubscribe_leg_state

项目	内容
方法名	`unsubscribe_leg_state`
方法声明	`void unsubscribe_leg_state()`
功能概述	取消订阅下肢关节状态数据。
备注	非阻塞接口。与 subscribe_leg_state 配合使用。

15.1.6 publish_leg_command

项目	内容
方法名	`publish_leg_command`
方法声明	`Status publish_leg_command(LegJointCommand command)`
功能概述	发布下肢关节控制指令。
参数说明	`command`：包含目标角度/速度等控制信息的下肢关节控制指令。
返回值	`Status::OK` 表示成功，其他为失败。
备注	非阻塞接口。

15.2 数据结构定义

15.2.1 LegJointCommandArray —腿部关节命令数组

项目	内容
类型	`std::array<magic::dog::SingleLegJointCommand, magic::dog::kLegJointNum>` 的 Python 绑定
功能概述	固定大小的腿部关节命令数组，包含所有腿部关节的控制指令
主要方法	支持索引访问、迭代、长度查询，长度固定为腿部关节数量
使用场景	腿部运动控制、关节协调控制、步态规划

15.2.2 LegJointStateArray —腿部关节状态数组

项目	内容
类型	`std::array<magic::dog::SingleLegJointState, magic::dog::kLegJointNum>` 的 Python 绑定
功能概述	固定大小的腿部关节状态数组，包含所有腿部关节的实时状态信息
主要方法	支持索引访问、迭代、长度查询，长度固定为腿部关节数量
使用场景	关节状态监控、运动反馈、安全检测

15.2.3 SingleLegJointCommand —单腿关节命令结构体

字段名	类型	描述
q_des	float	期望关节角度
dq_des	float	期望关节角速度
tau_des	float	期望关节力矩
kp	float	位置增益
kd	float	速度增益

15.2.4 LegJointCommand —腿部关节命令结构体

字段名	类型	描述
timestamp	int	时间戳
cmd	LegJointCommandArray	关节命令列表

15.2.5 SingleLegJointState —单腿关节状态结构体

字段名	类型	描述
q	float	关节角度
dq	float	关节角速度
tau_est	float	估计关节力矩

15.2.6 LegState —腿部状态结构体

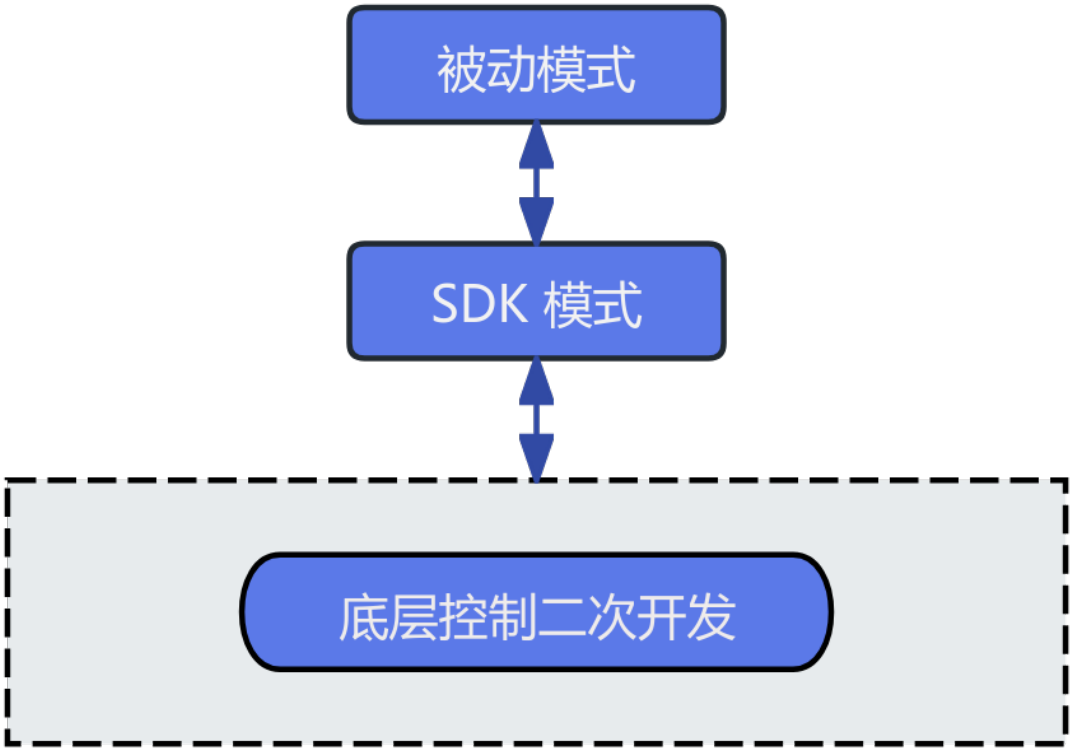
字段名	类型	描述
timestamp	int	时间戳
state	LegJointStateArray	关节状态列表

15.3 URDF 参考

机器人 URDF

15.4 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：



15.5 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 16

传感器控制服务 Python

提供机器人系统传感器数据管理，包括相机、IMU、激光雷达等。

16.1 接口定义

SensorController 是机器人传感器数据管理，包括相机、IMU、激光雷达等。

16.1.1 SensorController —传感器控制器

项目	内容
类名	`SensorController`
功能概述	机器人传感器数据管理，包括相机、IMU、激光雷达等
主要功能	传感器开关控制、数据订阅、多模态感知
使用场景	环境感知、导航定位、状态监控

16.1.2 initialize

项目	内容
方法名	`initialize`
方法声明	`bool initialize()`
功能概述	初始化传感器控制器。
返回值	`true` 表示成功，`false` 表示失败。
备注	首次使用前必须调用。

16.1.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	关闭传感器控制器。
备注	配合 initialize 使用。

16.1.4 open_laser_scan

项目	内容
方法名	`open_laser_scan`
方法声明	`Status open_laser_scan(int timeout_ms)`
功能概述	打开雷达。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

16.1.5 close_laser_scan

项目	内容
方法名	`close_laser_scan`
方法声明	`Status close_laser_scan(int timeout_ms)`
功能概述	关闭雷达。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

16.1.6 open_rgbd_camera

项目	内容
方法名	`open_rgbd_camera`
方法声明	`Status open_rgbd_camera(int timeout_ms)`
功能概述	打开 RGBD 相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

16.1.7 close_rgbd_camera

项目	内容
方法名	`close_rgbd_camera`
方法声明	`Status close_rgbd_camera(int timeout_ms)`
功能概述	关闭 RGBD 相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

16.1.8 open_binocular_camera

项目	内容
方法名	`open_binocular_camera`
方法声明	`Status open_binocular_camera(int timeout_ms)`
功能概述	打开双目相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口。

16.1.9 close_binocular_camera

项目	内容
方法名	`close_binocular_camera`
方法声明	`Status close_binocular_camera(int timeout_ms)`
功能概述	关闭双目相机。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

16.1.10 subscribe_imu

项目	内容
方法名	`subscribe_imu`
方法声明	`void subscribe_imu(callback)`
功能概述	订阅 IMU 数据。
参数说明	`callback`：回调函数，接收 IMU 数据。函数签名为：callback(data : Imu) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.11 unsubscribe_imu

项目	内容
方法名	`unsubscribe_imu`
方法声明	`void unsubscribe_imu()`
功能概述	取消订阅 IMU 数据。
备注	非阻塞接口。与 subscribe_imu 配合使用。

16.1.12 subscribe_ultra

项目	内容
方法名	`subscribe_ultra`
方法声明	`void subscribe_ultra(callback)`
功能概述	订阅超声波数据。
参数说明	`callback`：回调函数，接收超声波数据。函数签名为：callback(data : Float32MultiArray) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.13 unsubscribe_ultra

项目	内容
方法名	`unsubscribe_ultra`
方法声明	`void unsubscribe_ultra()`
功能概述	取消订阅超声波数据。
备注	非阻塞接口。与 subscribe_ultra 配合使用。

16.1.14 subscribe_head_touch

项目	内容
方法名	`subscribe_head_touch`
方法声明	`void subscribe_head_touch(callback)`
功能概述	订阅头部触摸数据。
参数说明	`callback`：回调函数，接收头部触摸数据。函数签名为：callback(data : HeadTouch) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.15 unsubscribe_head_touch

项目	内容
方法名	`unsubscribe_head_touch`
方法声明	`void unsubscribe_head_touch()`
功能概述	取消订阅头部触摸数据。
备注	非阻塞接口。与 subscribe_head_touch 配合使用。

16.1.16 subscribe_laser_scan

项目	内容
方法名	`subscribe_laser_scan`
方法声明	`void subscribe_laser_scan(callback)`
功能概述	订阅激光雷达数据。
参数说明	`callback`：回调函数，接收激光雷达数据。函数签名为：callback(data : LaserScan) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.17 unsubscribe_laser_scan

项目	内容
方法名	`unsubscribe_laser_scan`
方法声明	`void unsubscribe_laser_scan()`
功能概述	取消订阅激光雷达数据。
备注	非阻塞接口。与 subscribe_laser_scan 配合使用。

16.1.18 subscribe_rgb_depth_camera_info

项目	内容
方法名	`subscribe_rgb_depth_camera_info`
方法声明	`void subscribe_rgb_depth_camera_info(callback)`
功能概述	订阅 RGBD 深度相机内参数据。
参数说明	`callback`：回调函数，接收相机内参数据。函数签名为：callback(data : CameraInfo) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.19 unsubscribe_rgb_depth_camera_info

项目	内容
方法名	`unsubscribe_rgb_depth_camera_info`
方法声明	`void unsubscribe_rgb_depth_camera_info()`
功能概述	取消订阅 RGBD 深度相机内参数据。
备注	非阻塞接口。与 subscribe_rgb_depth_camera_info 配合使用。

16.1.20 subscribe_rgbd_depth_image

项目	内容
方法名	`subscribe_rgbd_depth_image`
方法声明	`void subscribe_rgbd_depth_image(callback)`
功能概述	订阅 RGBD 深度图像数据。
参数说明	`callback`：回调函数，接收深度图像数据。函数签名为：callback(data : Image) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.21 unsubscribe_rgbd_depth_image

项目	内容
方法名	`unsubscribe_rgbd_depth_image`
方法声明	`void unsubscribe_rgbd_depth_image()`
功能概述	取消订阅 RGBD 深度图像数据。
备注	非阻塞接口。与 subscribe_rgbd_depth_image 配合使用。

16.1.22 subscribe_rgbd_color_camera_info

项目	内容
方法名	`subscribe_rgbd_color_camera_info`
方法声明	`void subscribe_rgbd_color_camera_info(callback)`
功能概述	订阅 RGBD 彩色图像内参数据。
参数说明	`callback`：回调函数，接收相机内参数据。函数签名为：callback(data : CameraInfo) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.23 unsubscribe_rgbd_color_camera_info

项目	内容
方法名	`unsubscribe_rgbd_color_camera_info`
方法声明	`void unsubscribe_rgbd_color_camera_info()`
功能概述	取消订阅 RGBD 彩色图像内参数据。
备注	非阻塞接口。与 subscribe_rgbd_color_camera_info 配合使用。

16.1.24 subscribe_rgbd_color_image

项目	内容
方法名	`subscribe_rgbd_color_image`
方法声明	`void subscribe_rgbd_color_image(callback)`
功能概述	订阅 RGBD 彩色图像数据。
参数说明	`callback`：回调函数，接收彩色图像数据。函数签名为：callback(data : Image) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.25 unsubscribe_rgbd_color_image

项目	内容
方法名	`unsubscribe_rgbd_color_image`
方法声明	`void unsubscribe_rgbd_color_image()`
功能概述	取消订阅 RGBD 彩色图像数据。
备注	非阻塞接口。与 subscribe_rgbd_color_image 配合使用。

16.1.26 subscribe_left_binocular_high_img

项目	内容
方法名	`subscribe_left_binocular_high_img`
方法声明	`void subscribe_left_binocular_high_img(callback)`
功能概述	订阅左侧高质量双目数据。
参数说明	`callback`：回调函数，接收左侧高质量双目数据。函数签名为：callback(data : CompressedImage) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.27 unsubscribe_left_binocular_high_img

项目	内容
方法名	`unsubscribe_left_binocular_high_img`
方法声明	`void unsubscribe_left_binocular_high_img()`
功能概述	取消订阅左侧高质量双目数据。
备注	非阻塞接口。与 subscribe_left_binocular_high_img 配合使用。

16.1.28 subscribe_left_binocular_low_img

项目	内容
方法名	`subscribe_left_binocular_low_img`
方法声明	`void subscribe_left_binocular_low_img(callback)`
功能概述	订阅左侧低质量双目数据。
参数说明	`callback`：回调函数，接收左侧低质量双目数据。函数签名为：callback(data : CompressedImage) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.29 unsubscribe_left_binocular_low_img

项目	内容
方法名	`unsubscribe_left_binocular_low_img`
方法声明	`void unsubscribe_left_binocular_low_img()`
功能概述	取消订阅左侧低质量双目数据。
备注	非阻塞接口。与 subscribe_left_binocular_low_img 配合使用。

16.1.30 subscribe_right_binocular_low_img

项目	内容
方法名	`subscribe_right_binocular_low_img`
方法声明	`void subscribe_right_binocular_low_img(callback)`
功能概述	订阅右侧低质量双目数据。
参数说明	`callback`：回调函数，接收右侧低质量双目数据。函数签名为：callback(data : CompressedImage) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.31 unsubscribe_right_binocular_low_img

项目	内容
方法名	`unsubscribe_right_binocular_low_img`
方法声明	`void unsubscribe_right_binocular_low_img()`
功能概述	取消订阅右侧低质量双目数据。
备注	非阻塞接口。与 subscribe_right_binocular_low_img 配合使用。

16.1.32 subscribe_depth_image

项目	内容
方法名	`subscribe_depth_image`
方法声明	`void subscribe_depth_image(callback)`
功能概述	订阅深度图像数据。
参数说明	`callback`：回调函数，接收深度图像数据。函数签名为：callback(data : Image) -> None
备注	非阻塞接口，通过回调函数异步接收数据。

16.1.33 unsubscribe_depth_image

项目	内容
方法名	`unsubscribe_depth_image`
方法声明	`void unsubscribe_depth_image()`
功能概述	取消订阅深度图像数据。
备注	非阻塞接口。与 subscribe_depth_image 配合使用。

16.2 数据结构定义

16.2.1 Uint8Vector —无符号 8 位整数向量

项目	内容
类型	`std::vector<uint8_t>` 的 Python 绑定
功能概述	存储无符号 8 位整数的动态数组，常用于存储原始字节数据
主要方法	支持标准 Python 列表操作：索引访问、切片、迭代、长度查询等
使用场景	图像数据、音频数据、二进制数据存储

16.2.2 PointFieldVector —点云字段向量

项目	内容
类型	std::vector<magic::dog::PointField> 的 Python 绑定
功能概述	存储点云字段描述的动态数组，用于定义点云数据的结构
主要方法	支持标准 Python 列表操作，每个元素为 PointField 结构体
使用场景	点云数据处理、3D 传感器数据解析

16.2.3 MultiArrayDimensionVector —多维数组维度向量

项目	内容
类型	std::vector<magic::dog::MultiArrayDimension> 的 Python 绑定
功能概述	存储多维数组维度信息的动态数组，用于描述数组的形状和布局
主要方法	支持标准 Python 列表操作，每个元素为 MultiArrayDimension 结构体
使用场景	多维传感器数据、矩阵数据处理

16.2.4 FloatVector —浮点数向量

项目	内容
类型	std::vector<float> 的 Python 绑定
功能概述	存储单精度浮点数的动态数组，用于高精度数值计算
主要方法	支持标准 Python 列表操作，支持浮点数运算
使用场景	传感器数据、坐标数据、物理量数据

16.2.5 DoubleVector —双精度浮点数向量

项目	内容
类型	std::vector<double> 的 Python 绑定
功能概述	存储双精度浮点数的动态数组，提供最高精度的数值计算
主要方法	支持标准 Python 列表操作，支持高精度浮点数运算
使用场景	高精度传感器数据、科学计算、坐标变换

16.2.6 Double3Array —三维双精度浮点数数组

项目	内容
类型	std::array<double, 3> 的 Python 绑定
功能概述	固定大小的三维双精度浮点数数组，常用于 3D 坐标表示
主要方法	支持索引访问、迭代、长度查询，长度固定为 3
使用场景	3D 坐标、RGB 值、欧拉角、位置向量

16.2.7 Double4Array —四维双精度浮点数数组

项目	内容
类型	std::array<double, 4> 的 Python 绑定
功能概述	固定大小的四维双精度浮点数数组，常用于四元数表示
主要方法	支持索引访问、迭代、长度查询，长度固定为 4
使用场景	四元数、RGBA 值、姿态表示

16.2.8 Double9Array —九维双精度浮点数数组

项目	内容
类型	std::array<double, 9> 的 Python 绑定
功能概述	固定大小的九维双精度浮点数数组，常用于 3x3 矩阵表示
主要方法	支持索引访问、迭代、长度查询，长度固定为 9
使用场景	3x3 旋转矩阵、协方差矩阵、变换矩阵

16.2.9 Double12Array —十二维双精度浮点数数组

项目	内容
类型	std::array<double, 12> 的 Python 绑定
功能概述	固定大小的十二维双精度浮点数数组，常用于 3x4 投影矩阵表示
主要方法	支持索引访问、迭代、长度查询，长度固定为 12
使用场景	3x4 投影矩阵、相机参数矩阵

16.2.10 Header —头部信息结构体

字段名	类型	描述
stamp	int	时间戳
frame_id	str	坐标系 ID

16.2.11 Image —图像结构体

字段名	类型	描述
header	Header	头部信息
height	int	图像高度
width	int	图像宽度
encoding	str	编码格式
is_bigendian	bool	是否大端序
step	int	步长
data	Uint8Vector	图像数据

16.2.12 Imu —惯性测量单元结构体

字段名	类型	描述
timestamp	int	时间戳
orientation	Double4Array	姿态四元数（只读）
angular_velocity	Double3Array	角速度（只读）
linear_acceleration	Double3Array	线性加速度（只读）
temperature	float	温度

16.2.13 PointField —点云字段描述结构体

字段名	类型	描述
name	str	字段名，如”x”、”y”、”z”、”intensity” 等
offset	int	起始字节偏移
datatype	int	数据类型（对应常量）
count	int	该字段包含的元素数量

16.2.14 PointCloud2 —通用点云数据结构

字段名	类型	描述
header	Header	标准消息头
height	int	行数
width	int	列数
fields	PointFieldVector	点字段数组
is_bigendian	bool	字节序
point_step	int	每个点占用的字节数
row_step	int	每行占用的字节数
data	Uint8Vector	原始点云数据（按字段打包）
is_dense	bool	是否为稠密点云（无无效点）

16.2.15 CameraInfo —相机内参与畸变信息结构体

字段名	类型	描述
header	Header	通用消息头
height	int	图像高度（行数）
width	int	图像宽度（列数）
distortion_model	str	畸变模型，如”plumb_bob”
D	FloatVector	畸变参数数组
K	FloatVector	相机内参矩阵（9 个元素）
R	FloatVector	矫正矩阵（9 个元素）
P	FloatVector	投影矩阵（12 个元素）
binning_x	int	水平 binning 系数
binning_y	int	垂直 binning 系数
roi_x_offset	int	ROI 起始 x
roi_y_offset	int	ROI 起始 y
roi_height	int	ROI 高度
roi_width	int	ROI 宽度
roi_do_rectify	bool	是否进行矫正

16.2.16 CompressedImage —压缩图像结构体

字段名	类型	描述
header	Header	通用消息头
format	str	压缩格式
data	Uint8Vector	压缩后的图像数据

16.2.17 LaserScan —激光雷达数据结构体

字段名	类型	描述
header	Header	消息头
angle_min	int	起始角度（弧度）
angle_max	int	结束角度（弧度）
angle_increment	int	角度增量（弧度）
time_increment	int	时间增量（秒）
scan_time	int	扫描时间（秒）
range_min	int	最小距离（米）
range_max	int	最大距离（米）
ranges	FloatVector	距离数据数组（单位：米）
intensities	FloatVector	强度数据数组（无量纲）

16.2.18 MultiArrayDimension —多维数组维度描述结构体

字段名	类型	描述
label	str	维度标签
size	int	维度大小
stride	int	步长

16.2.19 MultiArrayLayout —多维数组布局描述结构体

字段名	类型	描述
dim_size	int	维度数量
dim	MultiArrayDimensionVector	维度描述数组
data_offset	int	数据偏移

16.2.20 Float32MultiArray —浮点数多维数组结构体

字段名	类型	描述
layout	MultiArrayLayout	数组布局描述
data	FloatVector	浮点数数据数组

16.2.21 ByteMultiArray —字节数组结构体

字段名	类型	描述
layout	MultiArrayLayout	数组布局描述
data	Uint8Vector	字节数据数组

16.2.22 HeadTouch —头部触摸数据结构体

字段名	类型	描述
data	int	触摸状态数据

16.3 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 17

语音控制服务 Python

提供机器人系统音频控制，包括 TTS 语音合成、音频播放、语音配置。

17.1 接口定义

AudioController 是机器人音频控制，包括 TTS 语音合成、音频播放、语音配置。

17.1.1 AudioController —音频控制器

项目	内容
类名	`AudioController`
功能概述	机器人音频控制，包括 TTS 语音合成、音频播放、语音配置
主要功能	语音播放、音量控制、语音模型切换、原始音频数据订阅
使用场景	语音交互、音频播放、语音识别

17.1.2 initialize

项目	内容
方法名	`initialize`
方法声明	`bool initialize()`
功能概述	初始化音频控制器。
返回值	`true` 表示成功，`false` 表示失败。
备注	首次使用前必须调用。

17.1.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	关闭音频控制器。
备注	配合 initialize 使用。

17.1.4 play

项目	内容
方法名	`play`
方法声明	`Status play(TtsCommand command, int timeout_ms)`
功能概述	播放 TTS（文本转语音）语音命令。
参数说明	`command`：TTS 命令，包含文本、语速、语调等。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

17.1.5 stop

项目	内容
方法名	`stop`
方法声明	`Status stop(int timeout_ms)`
功能概述	停止当前音频播放。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

17.1.6 set_volume

项目	内容
方法名	`set_volume`
方法声明	`Status set_volume(int volume, int timeout_ms)`
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。`timeout_ms`：超时时间（毫秒），默认值为 5000。
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。

17.1.7 get_volume

项目	内容
方法名	`get_volume`
方法声明	`tuple[Status, int] get_volume(int timeout_ms)`
功能概述	获取当前音频输出音量。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	成功，返回当前音量值。
备注	非阻塞接口。

17.1.8 switch_tts_voice_model

项目	内容
方法名	`switch_tts_voice_model`
方法声明	`Status switch_tts_voice_model(TtsType tts_type, int timeout_ms)`
功能概述	切换 TTS 语音模型。
参数说明	`tts_type`：TTS 语音模型类型 `timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	操作状态。
备注	切换模型属于阻塞操作。

17.1.9 get_voice_config

项目	内容
方法名	`get_voice_config`
方法声明	`tuple[Status, GetSpeechConfig] get_voice_config(int timeout_ms)`
功能概述	获取语音系统的完整配置信息。
参数说明	`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	语音系统配置信息，失败返回空配置对象。
备注	阻塞接口，获取的配置包含音色、智能体、唤醒、对话等所有子配置。

17.1.10 set_voice_config

项目	内容
方法名	`set_voice_config`
方法声明	`Status set_voice_config(SetSpeechConfig config, int timeout_ms)`
功能概述	设置语音系统的完整配置信息。
参数说明	`config`：要设置的语音系统配置信息。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败状态。
备注	阻塞接口，设置的配置将完全覆盖当前所有语音相关配置。

17.1.11 control_voice_stream

项目	内容
方法名	`control_voice_stream`
方法声明	`Status control_voice_stream(bool raw_data, bool bf_data, int timeout_ms)`
功能概述	控制语音数据流。
参数说明	`raw_data`：是否开启原始语音数据流。`bf_data`：是否开启 BF 语音数据流。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	操作状态。`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于控制语音数据流的开启和关闭。

17.1.12 subscribe_origin_voice_data

项目	内容
方法名	`subscribe_origin_voice_data`
方法声明	`void subscribe_origin_voice_data(callback)`
功能概述	订阅原始语音数据。
参数说明	`callback`：接收到原始语音数据后的处理回调。函数签名为：callback(data : ByteMultiArray) -> 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_origin_voice_data

项目	内容
方法名	`unsubscribe_origin_voice_data`
方法声明	`void unsubscribe_origin_voice_data()`
功能概述	取消订阅原始语音数据。
备注	非阻塞接口。与 subscribe_origin_voice_data 配合使用。

17.1.14 subscribe_bf_voice_data

项目	内容
方法名	`subscribe_bf_voice_data`
方法声明	`void subscribe_bf_voice_data(callback)`
功能概述	订阅 BF 语音数据。
参数说明	`callback`：接收到 BF 语音数据后的处理回调。函数签名为：callback(data : ByteMultiArray) -> 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_bf_voice_data

项目	内容
方法名	`unsubscribe_bf_voice_data`
方法声明	`void unsubscribe_bf_voice_data()`
功能概述	取消订阅 BF 语音数据。
备注	非阻塞接口。与 subscribe_bf_voice_data 配合使用。

17.1.16 control_speech_io

项目	内容
方法名	control_speech_io
方法声明	`Status control_speech_io(bool enable_data, int timeout_ms);`
功能概述	控制语音 io 数据流 (asr 和 tts)。
参数说明	`enable_data`：打开或关闭语音 io 数据流。`timeout_ms`：超时时间（毫秒），默认值为 5000。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于控制语音 io 数据流 (asr 和 tts)。

17.1.17 subscribe_speech_asr

项目	内容
方法名	subscribe_speech_asr
方法声明	`void subscribe_speech_asr(callback);`
功能概述	订阅语音 asr 流。
参数说明	`callback`：处理接收到的语音 asr 流的回调。
备注	阻塞接口，用于订阅语音 asr 流。

17.1.18 unsubscribe_speech_asr

项目	内容
方法名	unsubscribe_speech_asr
方法声明	`void unsubscribe_speech_asr();`
功能概述	取消订阅语音 asr 流。
备注	阻塞接口，用于取消订阅语音 asr 流。

17.1.19 publish_speech_tts

项目	内容
方法名	publish_speech_tts
方法声明	`Status publish_speech_tts(SpeechTTSStream data);`
功能概述	发布语音 tts 流。
参数说明	`data`：语音 tts 流。
备注	阻塞接口，用于发布语音 tts 流。数据 ID 应该与 ASR 输出的请求 ID 一致，起始类型为 begin，中间数据类型为 var，结束类型为 end。

17.2 枚举类型定义

17.2.1 TtsPriority —TTS 优先级枚举

枚举值	数值	描述
HIGH	0	高优先级
MIDDLE	1	中优先级
LOW	2	低优先级

17.2.2 TtsMode —TTS 模式枚举

枚举值	数值	描述
CLEARTOP	0	清除顶部
ADD	1	添加
CLEARBUFFER	2	清除缓冲区

17.2.3 TtsType —TTS 类型枚举

枚举值	数值	描述
NONE	0	无
DOUBAO	1	豆包
GOOGLE	2	谷歌

17.3 数据结构定义

17.3.1 S2DString —二维字符串数组

项目	内容
类型	std::array<std::string, 2> 的 Python 绑定
功能概述	固定大小的二维字符串数组，包含两个字符串元素
主要方法	支持索引访问、迭代、长度查询，长度固定为 2
使用场景	键值对存储、配置参数、状态标识

17.3.2 S2DStringVector —二维字符串数组

项目	内容
类型	std::vector<std::array<std::string, 2>> 的 Python 绑定
功能概述	可变长度的二维字符串数组，每个元素为包含两个字符串的数组
主要方法	支持索引访问、迭代、长度查询，支持动态添加和删除元素，每个元素长度固定为 2
使用场景	批量键值对存储、配置参数集合、状态标识列表等

17.3.3 String2DStringVectorMap —字符串到二维字符串数组向量的映射

项目	内容
类型	std::map<std::string, std::vector<std::array<std::string, 2>>> 的 Python 绑定
功能概述	字符串键到二维字符串数组向量的映射，用于存储配置参数
主要方法	支持标准 Python 字典操作：键值访问、迭代、长度查询等
使用场景	音色配置、智能体配置、系统参数存储

17.3.4 StringBotInfoMap —字符串到智能体信息的映射

项目	内容
类型	std::map<std::string, magic::dog::BotInfo> 的 Python 绑定
功能概述	字符串键到智能体信息的映射，用于管理多个智能体配置
主要方法	支持标准 Python 字典操作，每个值为 BotInfo 结构体
使用场景	智能体管理、配置存储、多智能体系统

17.3.5 StringCustomBotMap —自定义智能体映射

项目	内容
类型	std::map<std::string, magic::dog::CustomBotInfo> 的 Python 绑定
功能概述	字符串键到自定义智能体信息的映射，用于管理用户自定义的智能体
主要方法	支持标准 Python 字典操作，每个值为 CustomBotInfo 结构体
使用场景	自定义智能体管理、用户配置存储

17.3.6 StringStringMap —字符串到字符串的映射

项目	内容
类型	std::map<std::string, std::string> 的 Python 绑定
功能概述	字符串键到字符串值的简单映射，用于存储配置参数
主要方法	支持标准 Python 字典操作，键值均为字符串类型
使用场景	简单配置存储、参数映射、状态标识

17.3.7 TtsCommand —TTS 命令结构体

字段名	类型	描述
id	int	命令 ID
content	str	内容
priority	TtsPriority	优先级
mode	TtsMode	模式

17.3.8 GetSpeechConfig —语音配置获取结构体

字段名	类型	描述
tts_type	TtsType	TTS 类型
speaker_config	SpeakerConfig	音色配置
bot_config	BotConfig	智能体配置
wakeup_config	WakeupConfig	唤醒配置
dialog_config	DialogConfig	对话配置

17.3.9 SetSpeechConfig —语音配置设置结构体

字段名	类型	描述
speaker_id	str	音色 ID
region	str	音色区域
bot_id	str	智能体 ID
is_front_doa	bool	是否启用前向声源定位
is_fullduplex_enable	bool	是否启用全双工对话
is_enable	bool	是否启用对话功能
is_doa_enable	bool	是否启用声源定位
speaker_speed	double	音色语速
wakeup_name	str	唤醒词名称
custom_bot	CustomBotInfo	自定义智能体信息

17.3.10 SpeakerConfig —音色配置结构体

字段名	类型	描述
data	MapStringVectorArray2DString	音色数据映射
selected	SpeakerConfigSelected	选中的音色配置
speaker_speed	double	音色语速

17.3.11 BotConfig —智能体配置结构体

字段名	类型	描述
data	MapStringBotInfo	智能体数据映射
custom_data	CustomBotMap	自定义智能体数据
selected	BotInfo	选中的智能体

17.3.12 WakeupConfig —唤醒配置结构体

字段名	类型	描述
name	str	唤醒词名称
data	MapStringString	唤醒数据映射

17.3.13 DialogConfig —对话配置结构体

字段名	类型	描述
is_front_doa	bool	是否启用前向声源定位
is_fullduplex_enable	bool	是否启用全双工对话
is_enable	bool	是否启用对话功能
is_doa_enable	bool	是否启用声源定位

17.3.14 SpeakerConfigSelected —音色配置选择结构体

字段名	类型	描述
speaker_id	str	音色 ID
region	str	音色区域

17.3.15 BotInfo —智能体信息结构体

字段名	类型	描述
name	str	智能体名称
workflow	str	工作流 ID

17.3.16 CustomBotInfo —自定义智能体信息结构体

字段名	类型	描述
name	str	自定义智能体名称
workflow	str	自定义工作流 ID
token	str	访问令牌

17.3.17 SpeechASRStream —机器人 ASR 输出

字段名	类型	描述
id	str	ID - 请求 id
type	str	Type - 类型 request or cancel
text	str	Text - ASR 输出

17.3.18 SpeechTTSStream —机器人 TTS 输入

字段名	类型	描述
`id`	`str`	ID - 与请求 id 相同
`type`	`str`	Type - 类型 begin or var or end
`text`	`str`	Text - TTS 输入
`end_session`	`bool`	End Session - 结束会话并关闭 asr

17.4 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 18

状态监控服务 Python

提供机器人系统运行状态监控，包括故障检测、电池状态等。

18.1 接口定义

StateMonitor 是机器人运行状态监控，包括故障检测、电池状态等。

18.1.1 StateMonitor —状态监控器

项目	内容
类名	`StateMonitor`
功能概述	机器人运行状态监控，包括故障检测、电池状态等
主要功能	状态查询、故障监控、健康检查
使用场景	系统监控、故障诊断、状态报告

18.1.2 initialize

项目	内容
方法名	`initialize`
方法声明	`bool initialize()`
功能概述	初始化状态监控器。
返回值	`true` 表示成功，`false` 表示失败。
备注	首次使用前必须调用。

18.1.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	关闭状态监控器。
备注	配合 initialize 使用。

18.1.4 get_current_state

项目	内容
方法名	`get_current_state`
方法声明	`RobotState get_current_state()`
功能概述	获取机器人当前状态信息。
返回值	`tuple[Status, RobotState]` 对象
备注	非阻塞接口，用于获取机器人当前状态信息。

18.2 错误码映射表

错误码（十六进制）	错误描述
0x0000	无错误
0x1101	调用 ROS 服务失败
0x1301	中控节点丢失
0x1302	APP 节点丢失
0x1304	音频节点丢失
0x1305	运动节点丢失

下页继续

表 1 – 续上页

错误码（十六进制）	错误描述
0x1306	LCD 节点丢失
0x1307	realsense 节点丢失
0x1308	双目相机节点丢失
0x1309	LDS 节点丢失
0x130A	传感器板节点丢失
0x130B	触摸节点丢失
0x130C	SLAM 节点丢失
0x130D	导航节点丢失
0x130E	AI 节点丢失
0x130F	头部节点丢失
0x1310	云端处理器节点丢失
0x3201	激光无数据
0x3202	双目相机无数据
0x3203	双目相机数据错误
0x3204	双目相机初始化失败
0x320B	里程计无数据
0x320C	IMU 无数据
0x6101	机器人连接 APP 失败
0x6102	与 APP 断开连接
0x9201	打开 LCD 串口失败
0x7201	打开头部串口失败
0x7202	头部无数据
0x8201	打开传感器板串口失败
0x8202	传感器板无数据
0x2201	错误：导航未收到 tf 数据
0x2202	错误：导航未收到地图数据
0x2203	错误：导航未收到定位数据
0x2204	错误：导航未收到超声波数据
0x2205	错误：导航未收到激光数据
0x2206	错误：导航未收到 RGBD 数据
0x2207	错误：导航未收到多线激光数据
0x2208	错误：导航未收到点 TOF 数据
0x2209	错误：导航未收到面 TOF 数据
0x220A	错误：导航未收到里程计数据
0x2101	警告：导航未收到 tf 数据
0x2102	警告：导航未收到地图数据
0x2103	警告：导航未收到定位数据
0x2104	警告：导航未收到超声波数据

下页继续

表 1 – 续上页

错误码（十六进制）	错误描述
0x2105	警告：导航未收到激光数据
0x2106	警告：导航未收到 RGBD TOF 数据
0x2107	警告：导航未收到多线激光数据
0x2108	警告：导航未收到点 TOF 数据
0x2109	警告：导航未收到面 TOF 数据
0x210A	警告：导航未收到里程计数据
0x4201	SLAM 定位错误
0x4102	错误：SLAM 未收到激光数据
0x4103	错误：SLAM 未收到里程计数据
0x4205	SLAM 地图错误

18.3 枚举类型定义

18.3.1 BatteryState —电池状态枚举

枚举值	数值	描述
UNKNOWN	0	未知状态
GOOD	1	良好
OVERHEAT	2	过热
DEAD	3	电量耗尽
OVERVOLTAGE	4	过压
UNSPEC_FAILURE	5	未指定故障
COLD	6	过冷
WATCHDOG_TIMER_EXPIRE	7	看门狗定时器超时
SAFETY_TIMER_EXPIRE	8	安全定时器超时

18.3.2 PowerSupplyStatus —电源状态枚举

枚举值	数值	描述
UNKNOWN	0	未知状态
CHARGING	1	充电中
DISCHARGING	2	放电中
NOTCHARGING	3	未充电
FULL	4	电量满

18.4 数据结构定义

18.4.1 FaultVector —故障信息向量

项目	内容
类型	std::vector<magic::dog::Fault> 的 Python 绑定
功能概述	存储故障信息的动态数组，用于系统状态监控和错误处理
主要方法	支持标准 Python 列表操作，每个元素为 Fault 结构体
使用场景	故障诊断、系统监控、错误日志记录

18.4.2 Fault —故障信息结构体

字段名	类型	描述
error_code	int	错误代码
error_message	str	错误消息

18.4.3 BmsData —电池管理系统数据结构体

字段名	类型	描述
battery_percentage	double	电池电量百分比
battery_health	double	电池健康度
battery_state	BatteryState	电池状态
power_supply_status	PowerSupplyStatus	电源状态

18.4.4 RobotState —机器人状态结构体

字段名	类型	描述
faults	FaultVector	故障列表
bms_data	BmsData	电池数据

CHAPTER 19

SLAM 导航控制服务 Python

提供机器人系统 SLAM 导航控制服务，通过 SlamNavController 可以实现 SLAM 建图、定位、导航等功能。

19.1 接口定义

SlamNavController 是面向 SLAM 导航控制的控制器，支持建图、定位、导航等操作。

19.1.1 SlamNavController —SLAM 导航控制器

项目	内容
类名	`SlamNavController`
功能概述	面向 SLAM 导航控制的控制器，支持建图、定位、导航等操作
主要功能	SLAM 建图、定位、导航、地图管理
使用场景	机器人自主导航、环境建图、路径规划

19.1.2 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。
返回值	`true` 表示成功，`false` 表示失败。
备注	首次使用前必须调用。

19.1.3 shutdown

项目	内容
方法名	`shutdown`
方法声明	`void shutdown()`
功能概述	释放资源，清理 SLAM 导航控制器。
备注	配合 initialize 使用。

19.1.4 switch_to_idle

项目	内容
方法名	`switch_to_idle`
方法声明	`Status switch_to_idle()`
功能概述	切换到空闲模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换到空闲状态。

19.1.5 switch_to_location

项目	内容
方法名	`switch_to_location`
方法声明	`Status switch_to_location()`
功能概述	切换到定位模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换到定位状态。

19.1.6 start_mapping

项目	内容
方法名	`start_mapping`
方法声明	`Status start_mapping()`
功能概述	开始建图模式。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于启动建图任务。

19.1.7 cancel_mapping

项目	内容
方法名	`cancel_mapping`
方法声明	`Status cancel_mapping()`
功能概述	取消当前建图任务。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于取消建图任务。

19.1.8 save_map

项目	内容
方法名	`save_map`
方法声明	`Status save_map(const str& map_name)`
功能概述	结束建图并保存地图，当处于建图模式时。
参数说明	`map_name`：地图名称。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于保存当前建图结果。

19.1.9 load_map

项目	内容
方法名	`load_map`
方法声明	`Status load_map(const str& map_name)`
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。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于加载已保存的地图。

19.1.10 delete_map

项目	内容
方法名	`delete_map`
方法声明	`Status delete_map(const str& map_name)`
功能概述	删除地图。
参数说明	`map_name`：要删除的地图名称。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于删除不需要的地图。

19.1.11 get_all_map_info

项目	内容
方法名	get_all_map_info
方法声明	`tuple[Status, AllMapInfo] 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 `AllMapInfo`：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

19.1.12 init_pose

项目	内容
方法名	`init_pose`
方法声明	`Status init_pose(Pose3DEuler pose)`
功能概述	初始化位姿。
参数说明	`pose`：要发布的位姿信息。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置机器人的初始位姿。

19.1.13 get_current_localization_info

项目	内容
方法名	`get_current_localization_info`
方法声明	`tuple[Status, LocalizationInfo] get_current_localization_info()`
功能概述	获取当前位姿信息。
返回值	`Status::OK` 表示成功，其他为失败。`LocalizationInfo` 对象，失败返回空对象。
备注	阻塞接口，用于获取机器人当前位姿。

19.1.14 activate_nav_mode

项目	内容
方法名	`activate_nav_mode`
方法声明	`Status activate_nav_mode(NavMode mode)`
功能概述	激活导航模式。
参数说明	`mode`：目标导航模式（NavMode 枚举类型）。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于切换导航工作模式。

19.1.15 set_nav_target

项目	内容
方法名	`set_nav_target`
方法声明	`Status set_nav_target(NavTarget goal)`
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。
返回值	`Status::OK` 表示成功，其他为失败。
备注	阻塞接口，用于设置导航目标。

19.1.16 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	阻塞接口，用于暂停导航。

19.1.17 resume_nav_task

项目	内容
方法名	`resume_nav_task`
方法声明	`Status resume_nav_task()`
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	阻塞接口，用于恢复导航。

19.1.18 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	阻塞接口，用于取消导航。

19.1.19 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	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。

19.1.20 subscribe_odometry

项目	内容
方法名	`subscribe_odometry`
方法声明	`Status subscribe_odometry(callback: Callable[[Odometry], None])`
功能概述	订阅机器人里程计数据。
参数说明	`callback`：里程计数据回调函数。
返回值	`Status::OK` 表示成功，其他为失败。
备注	非阻塞接口，用于获取实时里程计数据。

19.1.21 unsubscribe_odometry

项目	内容
方法名	`unsubscribe_odometry`
方法声明	`void unsubscribe_odometry()`
功能概述	取消订阅机器人里程计数据。
返回值	无
备注	非阻塞接口，用于取消订阅里程计数据。

19.1.22 get_point_cloud_map

项目	内容
方法名	`get_point_cloud_map`
方法声明	`tuple[Status, PointCloud2] get_point_cloud_map(int timeout_ms)`
功能概述	获取当前构建的点云地图。
参数说明	`timeout_ms`：操作超时（毫秒），默认 5000。
返回值	返回 `Status` 状态码与 `PointCloud2` 对象（失败时为默认空对象）。
备注	阻塞接口，若在超时内未完成则返回错误状态。

19.2 枚举类型定义

19.2.1 NavMode —导航模式枚举类型

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

19.2.2 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	结束成功
END_FAILED	3	结束失败
PAUSE	4	暂停
CONTINUE	5	继续
CANCEL	6	取消

19.3 数据结构定义

19.3.1 Pose3DEuler —3D 位姿结构

字段名	类型	描述
position	list[double]	位置 (x, y, z)，用于表示空间位置
orientation	list[double]	欧拉角 (roll, pitch, yaw)，用于表示空间姿态，避免欧拉角万向锁问题

19.3.2 NavTarget —全局导航目标点结构

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

19.3.3 NavStatus —导航状态结构

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
message	str	导航状态消息

19.3.4 MapImageData —建图图像数据结构，.pgm 格式

字段名	类型	描述
type	str	魔数，”P5”：二进制格式
width	int	图像宽度
height	int	图像高度
max_gray_value	int	最大灰度值，255
image	list[int]	图像数据

19.3.5 MapMetaData —建图地图元数据结构

字段名	类型	描述
resolution	double	地图分辨率，单位：m/pixel
origin	Pose3DEuler	地图原点，世界坐标系原点相对于地图左下角的位置
map_image_data	MapImageData	图像数据，.pgm 格式图像数据

19.3.6 MapInfo —单个地图信息结构

字段名	类型	描述
map_name	str	地图名称
map_meta_data	MapMetaData	地图元数据

19.3.7 AllMapInfo —所有地图信息结构

字段名	类型	描述
current_map_name	str	当前地图名称
map_infos	list[MapInfo]	所有地图信息

19.3.8 LocalizationInfo —当前位置信息结构

字段名	类型	描述
is_localization	bool	是否已定位
pose	Pose3DEuler	欧拉角位姿

19.4 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。

19.4.1 适用场景与范围

- 小于 20m * 20m 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

19.4.2 SLAM 功能流程

功能	操作流程
建图	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

19.4.3 导航功能流程

功能	操作流程
定位	加载地图 → 切换到定位模式 → 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

CHAPTER 20

显示控制服务 Python

提供机器人系统显示服务控制器，通过 DisplayController 可以使用 RPC 方式实现对机器人显示的指令控制和状态获取。

20.1 接口定义

DisplayController 是显示控制功能，主要用于显示控制等场景。

20.1.1 DisplayController —显示控制器

20.1.2 initialize

20.1.3 shutdown

20.1.4 get_all_face_expressions

20.1.5 set_face_expression

20.1.6 get_current_face_expression

20.2 数据结构定义

20.2.1 FaceExpression —机器人面部表情结构

描述机器人面部表情的完整信息，支持唯一 ID、名称和描述。

20.3 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 21

高层运动控制示例

该示例展示了如何使用 MagicDog 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`

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

#include <termios.h>
#include <unistd.h>
#include <atomic>
#include <chrono>
#include <csignal>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <thread>
#include <math.h>

using namespace magic::dog;
```

(下页继续)

(续上页)

```
// Global variables
std::unique_ptr<MagicRobot> robot = nullptr;
std::atomic<bool> running(true);

void signalHandler(int signum) {
  std::cout << "\nInterrupt signal (" << signum << ") received." << std::endl;
  running = false;
  if (robot) {
    robot->Shutdown();
  }
  exit(signum);
}

void print_help() {
  std::cout << "Key function description:" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Gait and Trick Functions:" << std::endl;
  std::cout << "  1        Function 1: Recovery Stand" << std::endl;
  std::cout << "  2        Function 2: Force control standing" << std::endl;
  std::cout << "  3        Function 3: Execute trick - lie down" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Joystick Functions:" << std::endl;
  std::cout << "  w        Move forward" << std::endl;
  std::cout << "  a        Move left" << std::endl;
  std::cout << "  s        Move backward" << std::endl;
  std::cout << "  d        Move right" << std::endl;
  std::cout << "  t        Turn left" << std::endl;
  std::cout << "  g        Turn right" << std::endl;
  std::cout << "  x        Stop movement" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Head Position Functions:" << std::endl;
  std::cout << "  4        Function 4: Get head position" << std::endl;
  std::cout << "  5        Function 5: Set head position" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "  ?        Function ?: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

int getch() {
  struct termios oldt, newt;
  int ch;
  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
```

(下页继续)

(续上页)

```
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  return ch;
}

// Recovery Stand
void recovery_stand() {
  try {
    auto& high_controller = robot->GetHighLevelMotionController();
    auto status = high_controller.SetGait(GaitMode::GAIT_STAND_R);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set position control standing: " << status.message << std::endl;
    } else {
      std::cout << "Robot set to position control standing" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Error executing position control standing: " << e.what() << std::endl;
  }
}

// Force control standing
void balance_stand() {
  try {
    auto& high_controller = robot->GetHighLevelMotionController();
    auto status = high_controller.SetGait(GaitMode::GAIT_STAND_B);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set force control standing: " << status.message << std::endl;
    } else {
      std::cout << "Robot set to force control standing" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Error executing force control standing: " << e.what() << std::endl;
  }
}

// Execute trick
void execute_trick(const std::string& cmd) {
  try {
    TrickAction action;
    if (cmd == "102") {
      action = TrickAction::ACTION_LIE_DOWN;
```

(下页继续)

(续上页)

```
    } else if (cmd == "103") {
      action = TrickAction::ACTION_RECOVERY_STAND;
    } else if (cmd == "33") {
      action = TrickAction::ACTION_SHAKE_HEAD;
    } else {
      action = TrickAction::ACTION_NONE;
    }

    auto& high_controller = robot->GetHighLevelMotionController();
    auto status = high_controller.ExecuteTrick(action);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to execute trick: " << status.message << std::endl;
    } else {
      std::cout << "Trick executed successfully" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Error executing trick: " << e.what() << std::endl;
  }
}

// Switch gait to down climb stairs mode
bool change_gait_to_down_climb_stairs() {
  try {
    GaitMode current_gait;
    auto& high_controller = robot->GetHighLevelMotionController();
    auto status = high_controller.GetGait(current_gait);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to get current gait: " << status.message << std::endl;
      return false;
    }

    if (current_gait != GaitMode::GAIT_DOWN_CLIMB_STAIRS) {
      status = high_controller.SetGait(GaitMode::GAIT_DOWN_CLIMB_STAIRS);
      if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set down climb stairs gait: " << status.message << std::endl;
        return false;
      }

      // Wait for gait switch to complete
      while (current_gait != GaitMode::GAIT_DOWN_CLIMB_STAIRS) {
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
        status = high_controller.GetGait(current_gait);
        if (status.code != ErrorCode::OK) {
```

(下页继续)

(续上页)

```
          std::cerr << "Failed to get gait during transition: " << status.message << std::endl;
          return false;
        }
      }
    }
    std::cout << "Gait changed to down climb stairs" << std::endl;
    return true;
  } catch (const std::exception& e) {
    std::cerr << "Error changing gait: " << e.what() << std::endl;
    return false;
  }
}

void send_joystick_command(float left_x, float left_y, float right_x, float right_y) {
  if (!change_gait_to_down_climb_stairs()) {
    return;
  }

  JoystickCommand joy_command;
  joy_command.left_x_axis = left_x;
  joy_command.left_y_axis = left_y;
  joy_command.right_x_axis = right_x;
  joy_command.right_y_axis = right_y;

  auto& high_controller = robot->GetHighLevelMotionController();
  auto status = high_controller.SendJoyStickCommand(joy_command);
  if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to send joystick command: " << status.message << std::endl;
  }
}

void get_current_head_position() {
  auto& high_controller = robot->GetHighLevelMotionController();

  EulerAngles euler_angles;
  auto status = high_controller.GetHeadPosition(euler_angles);
  if (status.code != ErrorCode::OK) {
    std::cerr << "get current head position failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }

  std::cout << "Head Position Roll Angle: " << euler_angles.roll << std::endl;
```

(下页继续)

(续上页)

```
  std::cout << "Head Position Pitch Angle: " << euler_angles.pitch << std::endl;
  std::cout << "Head Position Yaw Angle: " << euler_angles.yaw << std::endl;
}

void set_head_position(const EulerAngles& euler_angles) {
  auto& high_controller = robot->GetHighLevelMotionController();

  auto status = high_controller.SetHeadPosition(euler_angles);
  if (status.code != ErrorCode::OK) {
    std::cerr << "set head position failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }
  std::cout << "set head position success" << std::endl;
}

// Angle to radian function
constexpr double deg2rad(double deg) {
    return deg * M_PI / 180.0;
}

int main(int argc, char* argv[]) {
  print_help();

  std::cout << "MagicDog SDK C++ Example Program" << std::endl;

  robot = std::make_unique<MagicRobot>();
  if (!robot->Initialize("192.168.55.10")) {
    std::cerr << "Initialization failed" << std::endl;
    return -1;
  }

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

  // Set motion control level to high level
  status = robot->SetMotionControlLevel(ControllerLevel::HighLevel);
  if (status.code != ErrorCode::OK) {
```

(下页继续)

(续上页)

```
    std::cerr << "Failed to set motion control level: " << status.message << std::endl;
    robot->Shutdown();
    return -1;
  }

  std::cout << "Program started, please use keys to control robot..." << std::endl;

  while (running) {
    std::cout << "Enter command: ";
    int key = getch();
    if (key == 27) {  // ESC key
      break;
    }

    if (key == '3') {
      std::string str_input;
      std::cout << "Enter parameters: ";
      std::getline(std::cin, str_input);
      if (str_input.empty()) {
        continue;
      }

      // Parse parameters
      std::string cmd = str_input.substr(0, str_input.find(' '));
      if (cmd.empty()) {
        cmd = "102";
      }
      std::cout << "Execute trick: " << cmd << std::endl;
      execute_trick(cmd);
      continue;
    }

    switch (key) {
      case '1':
        recovery_stand();
        break;
      case '2':
        balance_stand();
        break;
      case 'w':
        send_joystick_command(0.0, 1.0, 0.0, 0.0);
        break;
      case 's':
```

(下页继续)

(续上页)

```
        send_joystick_command(0.0, -1.0, 0.0, 0.0);
        break;
      case 'a':
        send_joystick_command(-1.0, 0.0, 0.0, 0.0);
        break;
      case 'd':
        send_joystick_command(1.0, 0.0, 0.0, 0.0);
        break;
      case 't':
        send_joystick_command(0.0, 0.0, -1.0, 0.0);
        break;
      case 'g':
        send_joystick_command(0.0, 0.0, 1.0, 0.0);
        break;
      case 'x':
        send_joystick_command(0.0, 0.0, 0.0, 0.0);
        break;
      case '4':
        get_current_head_position();
        break;
      case '5':{
        EulerAngles angles;
        double roll_deg, pitch_deg, yaw_deg;
        std::cout << "Please enter three angles (roll pitch yaw, separated by spaces, -60␣
↪degrees to 60 degrees):";
        std::cin >> roll_deg >> pitch_deg >> yaw_deg;
        // Convert to radian
        angles.roll = deg2rad(roll_deg);
        angles.pitch = deg2rad(pitch_deg);
        angles.yaw = deg2rad(yaw_deg);
        // Output result
        std::cout << "\nOutput result:" << std::endl;
        std::cout << "Roll:  " << roll_deg << "° = " << angles.roll << " rad" << std::endl;
        std::cout << "Pitch: " << pitch_deg << "° = " << angles.pitch << " rad" << std::endl;
        std::cout << "Yaw:   " << yaw_deg << "° = " << angles.yaw << " rad" << std::endl;

        set_head_position(angles);
        std::cin.ignore();
      }
        break;
      case '?':
        print_help();
        break;
```

(下页继续)

(续上页)

```
      default:
        std::cout << "Unknown key: " << key << std::endl;
        break;
    }
  }

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

  // Shutdown robot
  robot->Shutdown();
  std::cout << "Robot shutdown" << std::endl;

  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 threading
import tty
import termios
import os
import logging
from typing import Optional
import magicdog_python as magicdog

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)

```
)

running = True
robot = None
high_controller = None


# 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


# Recovery stand
def recovery_stand():
    global high_controller
    try:
        # Set gait to position control standing
        status = high_controller.set_gait(magicdog.GaitMode.GAIT_STAND_R)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set position control standing: {status.message}")
        else:
            logging.info("Robot set to position control standing")
    except Exception as e:
        logging.error(f"Error executing position control standing: {e}")


# Force control standing
def balance_stand():
    global high_controller
    try:
        # Set gait to force control standing
        status = high_controller.set_gait(magicdog.GaitMode.GAIT_STAND_B)
```

(下页继续)

(续上页)

```
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set force control standing: {status.message}")
        else:
            logging.info("Robot set to force control standing")
    except Exception as e:
        logging.error(f"Error executing force control standing: {e}")


# Execute trick
def execute_trick(cmd):
    global high_controller
    try:
        action = get_action(cmd)
        # Execute lie down trick
        status = high_controller.execute_trick(action)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to execute trick: {status.message}")
        else:
            logging.info("Trick executed successfully")
    except Exception as e:
        logging.error(f"Error executing trick: {e}")


# Switch gait to down climb stairs mode
def change_gait_to_down_climb_stairs():
    global high_controller
    try:
        current_gait = high_controller.get_gait()
        if current_gait != magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS:
            status = high_controller.set_gait(magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS)
            if status.code != magicdog.ErrorCode.OK:
                logging.error(f"Failed to set down climb stairs gait: {status.message}")
                return False

            # Wait for gait switch to complete
            while (
                high_controller.get_gait() != magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS
            ):
                time.sleep(0.01)
        logging.info("Gait changed to down climb stairs")
        return True
    except Exception as e:
        logging.error(f"Error changing gait: {e}")
```

(下页继续)

(续上页)

```
        return False


def print_help():
    logging.info("Key function description:")
    logging.info("")
    logging.info("Gait and Trick Functions:")
    logging.info("  1        Function 1: Recovery stand")
    logging.info("  2        Function 2: Force control standing")
    logging.info("  3        Function 3: Execute trick")
    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("  t        Function t: Turn left")
    logging.info("  g        Function g: Turn right")
    logging.info("  x        Function x: Stop movement")
    logging.info("")
    logging.info("  ?        Function ?: Print help")
    logging.info("  ESC      Exit program")


def send_joystick_command(high_controller, left_x, left_y, right_x, right_y):
    if not change_gait_to_down_climb_stairs():
        return
    joy_command = magicdog.JoystickCommand()
    joy_command.left_x_axis = left_x
    joy_command.left_y_axis = left_y
    joy_command.right_x_axis = right_x
    joy_command.right_y_axis = right_y

    status = high_controller.send_joystick_command(joy_command)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to send joystick command: {status.message}")


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

(下页继续)

(续上页)

```
    if robot:
        robot.shutdown()
        logging.info("Robot shutdown")
    exit(-1)


def get_action(cmd):
    if cmd == "102":
        return magicdog.TrickAction.ACTION_LIE_DOWN
    elif cmd == "103":
        return magicdog.TrickAction.ACTION_RECOVERY_STAND
    elif cmd == "33":
        return magicdog.TrickAction.ACTION_SHAKE_HEAD
    else:
        return magicdog.TrickAction.ACTION_NONE


def main():
    """Main function"""
    print_help()

    global robot, high_controller, running

    logging.info("MagicDog SDK Python Example Program")

    robot = magicdog.MagicRobot()
    if not robot.initialize("192.168.55.10"):
        logging.error("Initialization failed")
        return

    if not robot.connect():
        logging.error("Connection failed")
        robot.shutdown()
        return

    # Set motion control level to high level
    status = robot.set_motion_control_level(magicdog.ControllerLevel.HIGH_LEVEL)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to set motion control level: {status.message}")
        robot.shutdown()
        return

    # Get high level motion controller
```

(下页继续)

(续上页)

```
    high_controller = robot.get_high_level_motion_controller()

    logging.info("Program started, please use keys to control robot...")

    while running:
        logging.info("Enter command: ")
        key = getch()
        if key == "\x1b":  # ESC key
            break

        # 1. Gait and Trick Functions
        # 1.1 Recovery stand
        if key == "1":
            recovery_stand()
        # 1.2 Force control standing
        elif key == "2":
            balance_stand()
        # 1.3 Execute trick
        elif key == "3":
            str_input = input("Enter parameters: ").strip()
            if not str_input:
                continue
            # Split input parameters by space
            parts = str_input.strip().split()

            # Parse parameters
            cmd = parts[0] if parts else "102"
            logging.info(f"Execute trick: {cmd}")
            execute_trick(cmd)
        # 2. Joystick Functions
        # 2.1 Move forward
        elif key.lower() == "w":
            left_y = 1.0
            left_x = 0.0
            right_x = 0.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.2 Move backward
        elif key.lower() == "s":
            left_y = -1.0
            left_x = 0.0
            right_x = 0.0
            right_y = 0.0
```

(下页继续)

(续上页)

```
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.3 Move left
        elif key.lower() == "a":
            left_x = -1.0
            left_y = 0.0
            right_x = 0.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.4 Move right
        elif key.lower() == "d":
            left_x = 1.0
            left_y = 0.0
            right_x = 0.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.5 Turn left
        elif key.lower() == "t":
            left_x = 0.0
            left_y = 0.0
            right_x = -1.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.6 Turn right
        elif key.lower() == "g":
            left_x = 0.0
            left_y = 0.0
            right_x = 1.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 2.7 Stop movement
        elif key.lower() == "x":
            left_x = 0.0
            left_y = 0.0
            right_x = 0.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # Help
        elif key.upper() == "?":
            print_help()
        else:
            logging.info(f"Unknown key: {key}")

    # Shutdown high level motion controller
```

(下页继续)

(续上页)

```
    high_controller.shutdown()
    logging.info("High level motion controller shutdown")

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

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


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

21.3 运行说明

21.3.1 环境准备

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$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

- 步态和特技控制:
  - 1: 恢复站立姿态
  - 2: 力控站立
  - 3: 执行特技动作 - 趴下
- 遥控控制:
  - w: 前进
  - s: 后退

  - a: 左移
  - d: 右移
  - t: 左转
  - g: 右转
  - x: 停止移动
- 头部控制:
  - 4: 获取头部位置
  - 5: 设置头部位置
- 其他功能:
  - ?: 打印帮助信息

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 MagicDog SDK 进行初始化、连接机器人、控制机器人重复站立、蹲下等基本操作。

22.1 C++

示例文件：`low_level_motion_example.cpp`

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

#include <unistd.h>
#include <csignal>
#include <iostream>
#include <memory>
#include <mutex>
#include <chrono>
#include <thread>
#include <cstdlib>

using namespace magic::dog;

// Global robot instance
```

(下页继续)

(续上页)

```
std::unique_ptr<MagicRobot> robot = nullptr;

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received." << std::endl;
    if (robot) {
        robot->Shutdown();
    }
    exit(signum);
}

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

    std::cout << "MagicDog SDK C++ Example Program" << std::endl;

    robot = std::make_unique<MagicRobot>();
    if (!robot->Initialize("192.168.55.10")) {
        std::cerr << "Initialization failed" << std::endl;
        return -1;
    }

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

    std::cout << "Setting motion control level to high level" << std::endl;
    status = robot->SetMotionControlLevel(ControllerLevel::HighLevel);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set motion control level: " << status.message << std::endl;
        robot->Shutdown();
        return -1;
    }

    std::cout << "Getting high level motion controller" << std::endl;
    auto& high_controller = robot->GetHighLevelMotionController();

    std::cout << "Setting motion mode to passive" << std::endl;
    status = high_controller.SetGait(GaitMode::GAIT_PASSIVE);
```

(下页继续)

(续上页)

```
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set motion mode: " << status.message << std::endl;
        robot->Shutdown();
        return -1;
    }

    std::cout << "Waiting for motion mode to change to passive" << std::endl;
    GaitMode current_mode = GaitMode::GAIT_DEFAULT;
    while (current_mode != GaitMode::GAIT_PASSIVE) {
        status = high_controller.GetGait(current_mode);
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to get gait: " << status.message << std::endl;
            robot->Shutdown();
            return -1;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    std::this_thread::sleep_for(std::chrono::seconds(2));

    std::cout << "Setting motion control level to low level" << std::endl;
    status = robot->SetMotionControlLevel(ControllerLevel::LowLevel);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set motion control level: " << status.message << std::endl;
        robot->Shutdown();
        return -1;
    }

    std::cout << "Waiting for motion mode to change to low level" << std::endl;
    while (current_mode != GaitMode::GAIT_LOWLEVL_SDK) {
        status = high_controller.GetGait(current_mode);
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to get gait: " << status.message << std::endl;
            robot->Shutdown();
            return -1;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    std::this_thread::sleep_for(std::chrono::seconds(2));

    std::cout << "Getting low level motion controller" << std::endl;
    auto& low_controller = robot->GetLowLevelMotionController();
```

(下页继续)

(续上页)

```
bool is_had_receive_leg_state = false;
std::mutex mut;
LegState receive_state;
int count = 0;

auto leg_state_callback = [&](const std::shared_ptr<LegState> msg) {
    if (!is_had_receive_leg_state) {
        std::lock_guard<std::mutex> guard(mut);
        is_had_receive_leg_state = true;
        receive_state = *msg;
    }
    if (count % 1000 == 0) {
        std::cout << "Received leg state data." << std::endl;
    }
    count++;
};

low_controller.SubscribeLegState(leg_state_callback);

std::cout << "Waiting to receive leg state data" << std::endl;
while (!is_had_receive_leg_state) {
    std::this_thread::sleep_for(std::chrono::milliseconds(2));
}

std::this_thread::sleep_for(std::chrono::seconds(10));

// Joint angles for different poses
double j1[] = {0.0000, 1.0477, -2.0944};  // base height 0.2
double j2[] = {0.0000, 0.7231, -1.4455};  // base height 0.3

// Get initial joint positions
double initial_q[12] = {
    receive_state.state[0].q, receive_state.state[1].q, receive_state.state[2].q,
    receive_state.state[3].q, receive_state.state[4].q, receive_state.state[5].q,
    receive_state.state[6].q, receive_state.state[7].q, receive_state.state[8].q,
    receive_state.state[9].q, receive_state.state[10].q, receive_state.state[11].q
};

LegJointCommand command;
int cnt = 0;

std::cout << "Starting joint control loop..." << std::endl;
```

(下页继续)

(续上页)

```
// 计算第一个周期的时间点
auto interval = std::chrono::milliseconds(2);
auto next_cycle = std::chrono::steady_clock::now() + interval;
while (true) {
    double t;

    if (cnt < 1000) {
        t = 1.0 * cnt / 1000.0;
        t = std::min(std::max(t, 0.0), 1.0);
        for (int i = 0; i < 12; ++i) {
            command.cmd[i].q_des = (1 - t) * initial_q[i] + t * j1[i % 3];
        }
    } else if (cnt < 1750) {
        t = 1.0 * (cnt - 1000) / 700.0;
        t = std::min(std::max(t, 0.0), 1.0);
        for (int i = 0; i < 12; ++i) {
            command.cmd[i].q_des = (1 - t) * j1[i % 3] + t * j2[i % 3];
        }
    } else if (cnt < 2500) {
        t = 1.0 * (cnt - 1750) / 700.0;
        t = std::min(std::max(t, 0.0), 1.0);
        for (int i = 0; i < 12; ++i) {
            command.cmd[i].q_des = (1 - t) * j2[i % 3] + t * j1[i % 3];
        }
    } else {
        cnt = 1000;
    }

    // Set control gains
    for (int i = 0; i < 12; ++i) {
        command.cmd[i].kp = 100;
        command.cmd[i].kd = 1.2;
    }

    low_controller.PublishLegCommand(command);
    // 精确休眠到下一个周期
    std::this_thread::sleep_until(next_cycle);
    // 更新下一个周期的时间点
    next_cycle += interval;
    cnt++;
}
```

(下页继续)

(续上页)

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

    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;

    std::cout << "\nExample program execution completed!" << std::endl;

    return 0;
}
```

22.2 Python

示例文件：`low_level_motion_example.py`

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

import sys
import time
import threading
import logging
from typing import Optional
import magicdog_python as magicdog


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


def main():
    """Main function"""
    logging.info("MagicDog SDK Python Example Program")
```

(下页继续)

(续上页)

```
    robot = magicdog.MagicRobot()
    if not robot.initialize("192.168.55.10"):
        logging.error("Initialization failed")
        return

    if not robot.connect():
        logging.error("Connection failed")
        robot.shutdown()
        return

    logging.info("Setting motion control level to high level")
    status = robot.set_motion_control_level(magicdog.ControllerLevel.HIGH_LEVEL)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to set motion control level: {status.message}")
        robot.shutdown()
        return

    logging.info("Getting high level motion controller")
    high_controller = robot.get_high_level_motion_controller()

    logging.info("Setting motion mode to passive")
    status = high_controller.set_gait(magicdog.GaitMode.GAIT_PASSIVE)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to set motion mode: {status.message}")
        robot.shutdown()
        return

    logging.info("Waiting for motion mode to change to passive")
    current_mode = magicdog.GaitMode.GAIT_DEFAULT
    while current_mode != magicdog.GaitMode.GAIT_PASSIVE:
        current_mode = high_controller.get_gait()
        time.sleep(0.1)

    time.sleep(2)

    logging.info("Setting motion control level to low level")
    status = robot.set_motion_control_level(magicdog.ControllerLevel.LOW_LEVEL)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to set motion control level: {status.message}")
        robot.shutdown()
        return

    logging.info("Waiting for motion mode to change to low level")
```

(下页继续)

(续上页)

```
while current_mode != magicdog.GaitMode.GAIT_LOWLEVL_SDK:
    current_mode = high_controller.get_gait()
    time.sleep(0.1)

time.sleep(2)

logging.info("Getting low level motion controller")
low_controller = robot.get_low_level_motion_controller()

is_had_receive_leg_state = False
mut = threading.Lock()
receive_state = None
count = 0

def leg_state_callback(msg):
    nonlocal is_had_receive_leg_state, receive_state, count
    if not is_had_receive_leg_state:
        with mut:
            is_had_receive_leg_state = True
            receive_state = msg
    if count % 1000 == 0:
        logging.info("Received leg state data.")
    count += 1

low_controller.subscribe_leg_state(leg_state_callback)

logging.info("Waiting to receive leg state data")
while not is_had_receive_leg_state:
    time.sleep(0.002)

time.sleep(10)

j1 = [0.0000, 1.0477, -2.0944]
j2 = [0.0000, 0.7231, -1.4455]
inital_q = [
    receive_state.state[0].q,
    receive_state.state[1].q,
    receive_state.state[2].q,
    receive_state.state[3].q,
    receive_state.state[4].q,
    receive_state.state[5].q,
    receive_state.state[6].q,
    receive_state.state[7].q,
```

(下页继续)

(续上页)

```
        receive_state.state[8].q,
        receive_state.state[9].q,
        receive_state.state[10].q,
        receive_state.state[11].q,
    ]

    command = magicdog.LegJointCommand()
    cnt = 0
    interval = 0.002 # 2ms控制周期
    next_t = time.perf_counter() + interval
    while True:
        if cnt < 1000:
            t = 1.0 * cnt / 1000.0
            t = min(max(t, 0.0), 1.0)
            for i in range(12):
                command.cmd[i].q_des = (1 - t) * inital_q[i] + t * j1[i % 3]
        elif cnt < 1750:
            t = 1.0 * (cnt - 1000) / 700.0
            t = min(max(t, 0.0), 1.0)
            for i in range(12):
                command.cmd[i].q_des = (1 - t) * j1[i % 3] + t * j2[i % 3]
        elif cnt < 2500:
            t = 1.0 * (cnt - 1750) / 700.0
            t = min(max(t, 0.0), 1.0)
            for i in range(12):
                command.cmd[i].q_des = (1 - t) * j2[i % 3] + t * j1[i % 3]
        else:
            cnt = 1000

        for i in range(12):
            command.cmd[i].kp = 100
            command.cmd[i].kd = 1.2

        low_controller.publish_leg_command(command)
        # 等待到下一个执行时间点
        next_t += interval
        sleep_time = next_t - time.perf_counter()
        if sleep_time > 0:
            time.sleep(sleep_time)
        cnt += 1

    robot.disconnect()
    robot.shutdown()
```

(下页继续)

(续上页)

```
    logging.info("\nExample program execution completed!")


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

22.3 运行说明

22.3.1 环境准备

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$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 MagicDog SDK 进行初始化、连接机器人、机器人传感器控制（雷达、RGBD 相机、双目相机）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 <unistd.h>
#include <csignal>
#include <iostream>
#include <memory>
#include <chrono>
#include <thread>
#include <map>
#include <string>
#include <cstdlib>

using namespace magic::dog;

```

(下页继续)

(续上页)

```
// Global robot instance
std::unique_ptr<MagicRobot> robot = nullptr;

// Sensor manager class
class SensorManager {
public:
    SensorManager(SensorController& controller) : controller_(controller) {}

    // Channel Control
    bool open_channel() {
        if (channel_opened_) {
            std::cout << "Channel already opened" << std::endl;
            return true;
        }

        auto status = robot->OpenChannelSwitch();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to open channel: " << status.message << std::endl;
            return false;
        }

        channel_opened_ = true;
        std::cout << "✓ Channel opened successfully" << std::endl;
        return true;
    }

    bool close_channel() {
        if (!channel_opened_) {
            std::cout << "Channel already closed" << std::endl;
            return true;
        }

        auto status = robot->CloseChannelSwitch();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to close channel: " << status.message << std::endl;
            return false;
        }

        channel_opened_ = false;
        std::cout << "✓ Channel closed successfully" << std::endl;
        return true;
    }
```

(下页继续)

(续上页)

```
    // Sensor Open/Close
    bool open_laser_scan() {
        if (sensors_state_["laser_scan"]) {
            std::cout << "Laser scan already opened" << std::endl;
            return true;
        }

        auto status = controller_.OpenLaserScan();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to open laser scan: " << status.message << std::endl;
            return false;
        }

        sensors_state_["laser_scan"] = true;
        std::cout << "✓ Laser scan opened" << std::endl;
        return true;
    }

    bool close_laser_scan() {
        if (!sensors_state_["laser_scan"]) {
            std::cout << "Laser scan already closed" << std::endl;
            return true;
        }

        // Unsubscribe if subscribed
        if (subscriptions_["laser_scan"]) {
            toggle_laser_scan_subscription();
        }

        auto status = controller_.CloseLaserScan();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to close laser scan: " << status.message << std::endl;
            return false;
        }

        sensors_state_["laser_scan"] = false;
        std::cout << "✓ Laser scan closed" << std::endl;
        return true;
    }

    bool open_rgbd_camera() {
        if (sensors_state_["rgbd_camera"]) {
            std::cout << "RGBD camera already opened" << std::endl;
```

(下页继续)

(续上页)

```
            return true;
        }

        auto status = controller_.OpenRgbdCamera();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to open RGBD camera: " << status.message << std::endl;
            return false;
        }

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

    bool close_rgbd_camera() {
        if (!sensors_state_["rgbd_camera"]) {
            std::cout << "RGBD camera already closed" << std::endl;
            return true;
        }

        // Unsubscribe all RGBD subscriptions if subscribed
        if (subscriptions_["rgbd_color_info"]) {
            toggle_rgbd_color_info_subscription();
        }
        if (subscriptions_["rgbd_depth_image"]) {
            toggle_rgbd_depth_image_subscription();
        }
        if (subscriptions_["rgbd_color_image"]) {
            toggle_rgbd_color_image_subscription();
        }
        if (subscriptions_["rgb_depth_info"]) {
            toggle_rgb_depth_info_subscription();
        }

        auto status = controller_.CloseRgbdCamera();
        if (status.code != ErrorCode::OK) {
            std::cerr << "Failed to close RGBD camera: " << status.message << std::endl;
            return false;
        }

        sensors_state_["rgbd_camera"] = false;
        std::cout << "✓ RGBD camera closed" << std::endl;
        return true;
```

(下页继续)

(续上页)

```
}

bool open_binocular_camera() {
    if (sensors_state_["binocular_camera"]) {
        std::cout << "Binocular camera already opened" << std::endl;
        return true;
    }

    auto status = controller_.OpenBinocularCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to open binocular camera: " << status.message << std::endl;
        return false;
    }

    sensors_state_["binocular_camera"] = true;
    std::cout << "✓ Binocular camera opened" << std::endl;
    return true;
}

bool close_binocular_camera() {
    if (!sensors_state_["binocular_camera"]) {
        std::cout << "Binocular camera already closed" << std::endl;
        return true;
    }

    // Unsubscribe all binocular subscriptions if subscribed
    if (subscriptions_["left_binocular_high"]) {
        toggle_left_binocular_high_subscription();
    }
    if (subscriptions_["left_binocular_low"]) {
        toggle_left_binocular_low_subscription();
    }
    if (subscriptions_["right_binocular_low"]) {
        toggle_right_binocular_low_subscription();
    }

    auto status = controller_.CloseBinocularCamera();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close binocular camera: " << status.message << std::endl;
        return false;
    }

    sensors_state_["binocular_camera"] = false;
```

(下页继续)

(续上页)

```
        std::cout << "✓ Binocular camera closed" << std::endl;
        return true;
    }

    // Subscribe/Unsubscribe Methods
    void toggle_ultra_subscription() {
        if (subscriptions_["ultra"]) {
            controller_.UnsubscribeUltra();
            subscriptions_["ultra"] = false;
            std::cout << "✓ Ultra sensor unsubscribed" << std::endl;
        } else {
            controller_.SubscribeUltra([](const std::shared_ptr<Float32MultiArray> ultra) {
                std::cout << "Ultra: " << ultra->data.size() << std::endl;
            });
            subscriptions_["ultra"] = true;
            std::cout << "✓ Ultra sensor subscribed" << std::endl;
        }
    }

    void toggle_head_touch_subscription() {
        if (subscriptions_["head_touch"]) {
            controller_.UnsubscribeHeadTouch();
            subscriptions_["head_touch"] = false;
            std::cout << "✓ Head touch sensor unsubscribed" << std::endl;
        } else {
            controller_.SubscribeHeadTouch([](const std::shared_ptr<HeadTouch> touch) {
                std::cout << "Head Touch: " << int(touch->data) << std::endl;
            });
            subscriptions_["head_touch"] = true;
            std::cout << "✓ Head touch sensor subscribed" << std::endl;
        }
    }

    void toggle_imu_subscription() {
        if (subscriptions_["imu"]) {
            controller_.UnsubscribeImu();
            subscriptions_["imu"] = false;
            std::cout << "✓ IMU sensor unsubscribed" << std::endl;
        } else {
            static int imu_counter = 0;
            controller_.SubscribeImu([&](const std::shared_ptr<Imu> imu) {
                imu_counter++;
                if (imu_counter % 500 == 0) {
```

(下页继续)

(续上页)

```
                    std::cout << "IMU: " << imu->temperature << std::endl;
                }
            });
            subscriptions_["imu"] = true;
            std::cout << "✓ IMU sensor subscribed" << std::endl;
        }
    }

    void toggle_laser_scan_subscription() {
        if (subscriptions_["laser_scan"]) {
            controller_.UnsubscribeLaserScan();
            subscriptions_["laser_scan"] = false;
            std::cout << "✓ Laser scan unsubscribed" << std::endl;
        } else {
            controller_.SubscribeLaserScan([](const std::shared_ptr<LaserScan> scan) {
                std::cout << "Laser Scan: " << scan->ranges.size() << " ranges" << std::endl;
            });
            subscriptions_["laser_scan"] = true;
            std::cout << "✓ Laser scan subscribed" << std::endl;
        }
    }

    void toggle_rgbd_color_info_subscription() {
        if (subscriptions_["rgbd_color_info"]) {
            controller_.UnsubscribeRgbdColorCameraInfo();
            subscriptions_["rgbd_color_info"] = false;
            std::cout << "✓ RGBD color camera info unsubscribed" << std::endl;
        } else {
            controller_.SubscribeRgbdColorCameraInfo([](const std::shared_ptr<CameraInfo> info)
↪{
                std::cout << "RGBD Color Info: K=" << info->K[0] << ", " << info->K[1] << ", " <
↪< info->K[2] << ", " << info->K[3] << ", " << info->K[4] << ", " << info->K[5] << ", " <<␣
↪info->K[6] << ", " << info->K[7] << ", " << info->K[8] << std::endl;
            });
            subscriptions_["rgbd_color_info"] = true;
            std::cout << "✓ RGBD color camera info subscribed" << std::endl;
        }
    }

    void toggle_rgbd_depth_image_subscription() {
        if (subscriptions_["rgbd_depth_image"]) {
            controller_.UnsubscribeRgbdDepthImage();
            subscriptions_["rgbd_depth_image"] = false;
```

(下页继续)

(续上页)

```
            std::cout << "✓ RGBD depth image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeRgbdDepthImage([](const std::shared_ptr<Image> img) {
                std::cout << "RGBD Depth Image: " << img->data.size() << " bytes" << std::endl;
            });
            subscriptions_["rgbd_depth_image"] = true;
            std::cout << "✓ RGBD depth image subscribed" << std::endl;
        }
    }

    void toggle_rgbd_color_image_subscription() {
        if (subscriptions_["rgbd_color_image"]) {
            controller_.UnsubscribeRgbdColorImage();
            subscriptions_["rgbd_color_image"] = false;
            std::cout << "✓ RGBD color image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeRgbdColorImage([](const std::shared_ptr<Image> img) {
                std::cout << "RGBD Color Image: " << img->data.size() << " bytes" << std::endl;
            });
            subscriptions_["rgbd_color_image"] = true;
            std::cout << "✓ RGBD color image subscribed" << std::endl;
        }
    }

    void toggle_rgb_depth_info_subscription() {
        if (subscriptions_["rgb_depth_info"]) {
            controller_.UnsubscribeRgbDepthCameraInfo();
            subscriptions_["rgb_depth_info"] = false;
            std::cout << "✓ RGB depth camera info unsubscribed" << std::endl;
        } else {
            controller_.SubscribeRgbDepthCameraInfo([](const std::shared_ptr<CameraInfo> info) {
                std::cout << "RGB Depth Info: K=" << info->K[0] << ", " << info->K[1] << ", " <
↪< info->K[2] << ", " << info->K[3] << ", " << info->K[4] << ", " << info->K[5] << ", " <<␣
↪info->K[6] << ", " << info->K[7] << ", " << info->K[8] << std::endl;
            });
            subscriptions_["rgb_depth_info"] = true;
            std::cout << "✓ RGB depth camera info subscribed" << std::endl;
        }
    }

    void toggle_left_binocular_high_subscription() {
        if (subscriptions_["left_binocular_high"]) {
            controller_.UnsubscribeLeftBinocularHighImg();
```

(下页继续)

(续上页)

```
            subscriptions_["left_binocular_high"] = false;
            std::cout << "✓ Left binocular high image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeLeftBinocularHighImg([](const std::shared_ptr<CompressedImage>␣
↪img) {
                std::cout << "Left Binocular High: " << img->data.size() << " bytes" <<␣
↪std::endl;
            });
            subscriptions_["left_binocular_high"] = true;
            std::cout << "✓ Left binocular high image subscribed" << std::endl;
        }
    }

    void toggle_left_binocular_low_subscription() {
        if (subscriptions_["left_binocular_low"]) {
            controller_.UnsubscribeLeftBinocularLowImg();
            subscriptions_["left_binocular_low"] = false;
            std::cout << "✓ Left binocular low image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeLeftBinocularLowImg([](const std::shared_ptr<CompressedImage>␣
↪img) {
                std::cout << "Left Binocular Low: " << img->data.size() << " bytes" <<␣
↪std::endl;
            });
            subscriptions_["left_binocular_low"] = true;
            std::cout << "✓ Left binocular low image subscribed" << std::endl;
        }
    }

    void toggle_right_binocular_low_subscription() {
        if (subscriptions_["right_binocular_low"]) {
            controller_.UnsubscribeRightBinocularLowImg();
            subscriptions_["right_binocular_low"] = false;
            std::cout << "✓ Right binocular low image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeRightBinocularLowImg([](const std::shared_ptr<CompressedImage>␣
↪img) {
                std::cout << "Right Binocular Low: " << img->data.size() << " bytes" <<␣
↪std::endl;
            });
            subscriptions_["right_binocular_low"] = true;
            std::cout << "✓ Right binocular low image subscribed" << std::endl;
        }
```

(下页继续)

(续上页)

```
    }

    void toggle_depth_image_subscription() {
        if (subscriptions_["depth_image"]) {
            controller_.UnsubscribeDepthImage();
            subscriptions_["depth_image"] = false;
            std::cout << "✓ Depth image unsubscribed" << std::endl;
        } else {
            controller_.SubscribeDepthImage([](const std::shared_ptr<Image> img) {
                std::cout << "Depth Image: " << img->data.size() << " bytes" << std::endl;
            });
            subscriptions_["depth_image"] = true;
            std::cout << "✓ Depth image subscribed" << std::endl;
        }
    }

    void show_status() {
        std::cout << "\n" << std::string(70, '=') << std::endl;
        std::cout << "SENSOR STATUS" << std::endl;
        std::cout << std::string(70, '=') << std::endl;
        std::cout << "Channel Switch:                " << (channel_opened_ ? "OPEN" : "CLOSED")
↪<< std::endl;
        std::cout << "Laser Scan:                    " << (sensors_state_["laser_scan"] ? "OPEN
↪" : "CLOSED") << std::endl;
        std::cout << "RGBD Camera:                   " << (sensors_state_["rgbd_camera"] ? "OPEN
↪" : "CLOSED") << std::endl;
        std::cout << "Binocular Camera:              " << (sensors_state_["binocular_camera"] ?
↪"OPEN" : "CLOSED") << std::endl;
        std::cout << "\nSUBSCRIPTIONS:" << std::endl;
        std::cout << "  Ultra:                       " << (subscriptions_["ultra"] ? "✓␣
↪SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  Head Touch:                  " << (subscriptions_["head_touch"] ? "✓␣
↪SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  IMU:                         " << (subscriptions_["imu"] ? "✓ SUBSCRIBED
↪" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  Laser Scan:                  " << (subscriptions_["laser_scan"] ? "✓␣
↪SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  RGBD Color Info:             " << (subscriptions_["rgbd_color_info"] ?
↪"✓ SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  RGBD Depth Image:            " << (subscriptions_["rgbd_depth_image"] ?
↪"✓ SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
        std::cout << "  RGBD Color Image:            " << (subscriptions_["rgbd_color_image"] ?
↪"✓ SUBSCRIBED" : "□ UNSUBSCRIBED") << std::endl;
```

(下页继续)

(续上页)

```
        std::cout << "  RGB Depth Info:              " << (subscriptions_["rgb_depth_info"] ?
↪"✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
        std::cout << "  Left Binocular High:         " << (subscriptions_["left_binocular_high
↪"] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
        std::cout << "  Left Binocular Low:          " << (subscriptions_["left_binocular_low"]␣
↪? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
        std::cout << "  Right Binocular Low:         " << (subscriptions_["right_binocular_low
↪"] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
        std::cout << "  Depth Image:                 " << (subscriptions_["depth_image"] ? "✓␣
↪SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
        std::cout << std::string(70, '=') << "\n" << std::endl;
    }

private:
    SensorController& controller_;
    bool channel_opened_ = false;
    std::map<std::string, bool> sensors_state_ = {
        {"laser_scan", false},
        {"rgbd_camera", false},
        {"binocular_camera", false}
    };
    std::map<std::string, bool> subscriptions_ = {
        {"ultra", false},
        {"head_touch", false},
        {"imu", false},
        {"laser_scan", false},
        {"rgbd_color_info", false},
        {"rgbd_depth_image", false},
        {"rgbd_color_image", false},
        {"rgb_depth_info", false},
        {"left_binocular_high", false},
        {"left_binocular_low", false},
        {"right_binocular_low", false},
        {"depth_image", false}
    };
};

void print_menu() {
    std::cout << "\n" << std::string(70, '=') << std::endl;
    std::cout << "SENSOR CONTROL MENU" << std::endl;
    std::cout << std::string(70, '=') << std::endl;
    std::cout << "Channel Control:" << std::endl;
    std::cout << "  1 - Open Channel Switch       2 - Close Channel Switch" << std::endl;
```

(下页继续)

(续上页)

```
    std::cout << "\nSensor Control:" << std::endl;
    std::cout << "  3 - Open Laser Scan            4 - Close Laser Scan" << std::endl;
    std::cout << "  5 - Open RGBD Camera           6 - Close RGBD Camera" << std::endl;
    std::cout << "  7 - Open Binocular Camera      8 - Close Binocular Camera" << std::endl;
    std::cout << "\nSubscription Toggle (lowercase=toggle, UPPERCASE=unsubscribe):" <<␣
↪std::endl;
    std::cout << "  u/U - Ultra                    l/L - Laser Scan Data" << std::endl;
    std::cout << "  h/H - Head Touch               i/I - IMU" << std::endl;
    std::cout << "\nRGBD Subscriptions (lowercase=toggle, UPPERCASE=unsubscribe):" << std::endl;
    std::cout << "  r/R - RGBD Color Info            d/D - RGBD Depth Image" << std::endl;
    std::cout << "  c/C - RGBD Color Image           p/P - RGB Depth Info" << std::endl;
    std::cout << "\nBinocular Subscriptions:" << std::endl;
    std::cout << "  b/B - Left Binocular High        n/N - Left Binocular Low" << std::endl;
    std::cout << "  m/M - Right Binocular Low" << std::endl;
    std::cout << "\nOther Subscriptions:" << std::endl;
    std::cout << "  e/E - Depth Image" << std::endl;
    std::cout << "\nCommands:" << std::endl;
    std::cout << "  s - Show Status                ESC - Quit              ? - Help" <<␣
↪std::endl;
    std::cout << std::string(70, '=') << std::endl;
}

void signalHandler(int signum) {
    std::cout << "\nInterrupt signal (" << signum << ") received." << std::endl;
    if (robot) {
        robot->Shutdown();
    }
    exit(signum);
}

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

    std::cout << "\n" << std::string(70, '=') << std::endl;
    std::cout << "MagicDog SDK Sensor Interactive Example" << std::endl;
    std::cout << std::string(70, '=') << "\n" << std::endl;

    robot = std::make_unique<MagicRobot>();
    if (!robot->Initialize("192.168.55.10")) {
        std::cerr << "Robot initialization failed" << std::endl;
        return -1;
    }
```

(下页继续)

(续上页)

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

std::cout << "✓ Robot connected successfully\n" << std::endl;

SensorManager sensor_manager(robot->GetSensorController());

print_menu();

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

        if (choice.empty()) {
            continue;
        }

        char ch = choice[0];

        if (ch == 27) {  // ESC key
            std::cout << "ESC key pressed, exiting program..." << std::endl;
            break;
        }

        // Channel control
        if (choice == "1") {
            sensor_manager.open_channel();
        } else if (choice == "2") {
            sensor_manager.close_channel();
        }
        // Sensor control
        else if (choice == "3") {
            sensor_manager.open_laser_scan();
        } else if (choice == "4") {
            sensor_manager.close_laser_scan();
```

(下页继续)

(续上页)

```
            } else if (choice == "5") {
                sensor_manager.open_rgbd_camera();
            } else if (choice == "6") {
                sensor_manager.close_rgbd_camera();
            } else if (choice == "7") {
                sensor_manager.open_binocular_camera();
            } else if (choice == "8") {
                sensor_manager.close_binocular_camera();
            }
            // Basic sensor subscriptions
            else if (ch == 'u') {  // ~20HZ
                sensor_manager.toggle_ultra_subscription();
            } else if (ch == 'U') {  // ~20HZ
                sensor_manager.toggle_ultra_subscription();
            } else if (ch == 'h') {  // trigger
                sensor_manager.toggle_head_touch_subscription();
            } else if (ch == 'H') {  // trigger
                sensor_manager.toggle_head_touch_subscription();
            } else if (ch == 'i') {  // ~500HZ
                sensor_manager.toggle_imu_subscription();
            } else if (ch == 'I') {  // ~500HZ
                sensor_manager.toggle_imu_subscription();
            } else if (ch == 'l') {
                sensor_manager.toggle_laser_scan_subscription();
            } else if (ch == 'L') {
                sensor_manager.toggle_laser_scan_subscription();
            }
            // RGBD subscriptions
            else if (ch == 'r') {
                sensor_manager.toggle_rgbd_color_info_subscription();
            } else if (ch == 'R') {
                sensor_manager.toggle_rgbd_color_info_subscription();
            } else if (ch == 'd') {
                sensor_manager.toggle_rgbd_depth_image_subscription();
            } else if (ch == 'D') {
                sensor_manager.toggle_rgbd_depth_image_subscription();
            } else if (ch == 'c') {
                sensor_manager.toggle_rgbd_color_image_subscription();
            } else if (ch == 'C') {
                sensor_manager.toggle_rgbd_color_image_subscription();
            } else if (ch == 'p') {
                sensor_manager.toggle_rgb_depth_info_subscription();
            } else if (ch == 'P') {
```

(下页继续)

(续上页)

```
                sensor_manager.toggle_rgb_depth_info_subscription();
            }
            // Binocular subscriptions
            else if (ch == 'b') {
                sensor_manager.toggle_left_binocular_high_subscription();
            } else if (ch == 'B') {
                sensor_manager.toggle_left_binocular_high_subscription();
            } else if (ch == 'n') {
                sensor_manager.toggle_left_binocular_low_subscription();
            } else if (ch == 'N') {
                sensor_manager.toggle_left_binocular_low_subscription();
            } else if (ch == 'm') {
                sensor_manager.toggle_right_binocular_low_subscription();
            } else if (ch == 'M') {
                sensor_manager.toggle_right_binocular_low_subscription();
            } else if (ch == 'e') {
                sensor_manager.toggle_depth_image_subscription();
            } else if (ch == 'E') {
                sensor_manager.toggle_depth_image_subscription();
            }
            // Commands
            else if (ch == 's') {
                sensor_manager.show_status();
            } else if (choice == "?" || choice == "help") {
                print_menu();
            } else {
                std::cout << "Invalid choice: '" << choice << "'. Press '?' for help." <<␣
↪std::endl;
            }
        }
    } catch (const std::exception& e) {
        std::cerr << "Error occurred: " << e.what() << std::endl;
    }

    // Cleanup: unsubscribe all and close all sensors
    std::cout << "Cleaning up..." << std::endl;

    status = robot->Disconnect();
    if (status.code != ErrorCode::OK) {
        std::cerr << "robot disconnect failed, code: " << status.code
                  << ", message: " << status.message << std::endl;
    } else {
        std::cout << "robot disconnect" << std::endl;
```

(下页继续)

(续上页)

```
    }

    robot->Shutdown();
    std::cout << "robot shutdown" << std::endl;

    return 0;
}
```

23.2 Python

示例文件：sensor_example.py

示例文档：Python API/sensor_reference.md

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

import sys
import time
import logging
from typing import Optional, Dict
import magicdog_python as magicdog
from magicdog_python import ErrorCode


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

imu_counter = 0


class SensorManager:
    """Manages sensor subscriptions and channels"""

    def __init__(self, robot, sensor_controller):
        self.robot = robot
        self.sensor_controller = sensor_controller
        self.channel_opened = False
        self.sensors_state = {
            "laser_scan": False,
```

(下页继续)

(续上页)

```
            "rgbd_camera": False,
            "binocular_camera": False,
        }
        self.subscriptions = {
            "ultra": False,
            "head_touch": False,
            "imu": False,
            "laser_scan": False,
            "rgbd_color_info": False,
            "rgbd_depth_image": False,
            "rgbd_color_image": False,
            "rgb_depth_info": False,
            "left_binocular_high": False,
            "left_binocular_low": False,
            "right_binocular_low": False,
            "depth_image": False,
        }

    # === Channel Control ===
    def open_channel(self) -> bool:
        """Open sensor channel switch"""
        if self.channel_opened:
            logging.warning("Channel already opened")
            return True

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

        self.channel_opened = True
        logging.info("✓ Channel opened successfully")
        return True

    def close_channel(self) -> bool:
        """Close sensor channel switch"""
        if not self.channel_opened:
            logging.warning("Channel already closed")
            return True

        status = self.robot.close_channel_switch()
        if status.code != ErrorCode.OK:
            logging.error(f"Failed to close channel: {status.message}")
```

(下页继续)

(续上页)

```
            return False

        self.channel_opened = False
        logging.info("✓ Channel closed successfully")
        return True

    # === Sensor Open/Close ===
    def open_laser_scan(self) -> bool:
        """Open laser scan sensor"""
        if self.sensors_state["laser_scan"]:
            logging.warning("Laser scan already opened")
            return True

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

        self.sensors_state["laser_scan"] = True
        logging.info("✓ Laser scan opened")
        return True

    def close_laser_scan(self) -> bool:
        """Close laser scan sensor"""
        if not self.sensors_state["laser_scan"]:
            logging.warning("Laser scan already closed")
            return True

        # Unsubscribe if subscribed
        if self.subscriptions["laser_scan"]:
            self.toggle_laser_scan_subscription()

        status = self.sensor_controller.close_laser_scan()
        if status.code != ErrorCode.OK:
            logging.error(f"Failed to close laser scan: {status.message}")
            return False

        self.sensors_state["laser_scan"] = False
        logging.info("✓ Laser scan closed")
        return True

    def open_rgbd_camera(self) -> bool:
        """Open RGBD camera"""
```

(下页继续)

(续上页)

```
        if self.sensors_state["rgbd_camera"]:
            logging.warning("RGBD camera already opened")
            return True

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

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

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

        # Unsubscribe all RGBD subscriptions if subscribed
        if self.subscriptions["rgbd_color_info"]:
            self.toggle_rgbd_color_info_subscription()
        if self.subscriptions["rgbd_depth_image"]:
            self.toggle_rgbd_depth_image_subscription()
        if self.subscriptions["rgbd_color_image"]:
            self.toggle_rgbd_color_image_subscription()
        if self.subscriptions["rgb_depth_info"]:
            self.toggle_rgb_depth_info_subscription()

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

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

    def open_binocular_camera(self) -> bool:
        """Open binocular camera"""
        if self.sensors_state["binocular_camera"]:
            logging.warning("Binocular camera already opened")
            return True
```

(下页继续)

(续上页)

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

        self.sensors_state["binocular_camera"] = True
        logging.info("✓ Binocular camera opened")
        return True

    def close_binocular_camera(self) -> bool:
        """Close binocular camera"""
        if not self.sensors_state["binocular_camera"]:
            logging.warning("Binocular camera already closed")
            return True

        # Unsubscribe all binocular subscriptions if subscribed
        if self.subscriptions["left_binocular_high"]:
            self.toggle_left_binocular_high_subscription()
        if self.subscriptions["left_binocular_low"]:
            self.toggle_left_binocular_low_subscription()
        if self.subscriptions["right_binocular_low"]:
            self.toggle_right_binocular_low_subscription()

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

        self.sensors_state["binocular_camera"] = False
        logging.info("✓ Binocular camera closed")
        return True

    # === Subscribe/Unsubscribe Methods ===
    def toggle_ultra_subscription(self):
        """Toggle ultra sensor subscription"""
        if self.subscriptions["ultra"]:
            self.sensor_controller.unsubscribe_ultra()
            self.subscriptions["ultra"] = False
            logging.info("✓ Ultra sensor unsubscribed")
        else:
            self.sensor_controller.subscribe_ultra(
                lambda ultra: logging.info(f"Ultra: {len(ultra.data)}")
```

(下页继续)

(续上页)

```
            )
            self.subscriptions["ultra"] = True
            logging.info("✓ Ultra sensor subscribed")

    def toggle_head_touch_subscription(self):
        """Toggle head touch sensor subscription"""
        if self.subscriptions["head_touch"]:
            self.sensor_controller.unsubscribe_head_touch()
            self.subscriptions["head_touch"] = False
            logging.info("✓ Head touch sensor unsubscribed")
        else:
            self.sensor_controller.subscribe_head_touch(
                lambda touch: logging.info(f"Head Touch: {touch}")
            )
            self.subscriptions["head_touch"] = True
            logging.info("✓ Head touch sensor subscribed")

    def toggle_imu_subscription(self):
        """Toggle IMU sensor subscription"""
        if self.subscriptions["imu"]:
            self.sensor_controller.unsubscribe_imu()
            self.subscriptions["imu"] = False
            logging.info("✓ IMU sensor unsubscribed")
        else:

            def imu_callback(imu):
                global imu_counter
                imu_counter += 1
                if imu_counter % 500 == 0:
                    logging.info(f"IMU: {imu}")

            self.sensor_controller.subscribe_imu(imu_callback)
            self.subscriptions["imu"] = True
            logging.info("✓ IMU sensor subscribed")

    def toggle_laser_scan_subscription(self):
        """Toggle laser scan subscription"""
        if self.subscriptions["laser_scan"]:
            self.sensor_controller.unsubscribe_laser_scan()
            self.subscriptions["laser_scan"] = False
            logging.info("✓ Laser scan unsubscribed")
        else:
            self.sensor_controller.subscribe_laser_scan(
```

(下页继续)

(续上页)

```
            lambda scan: logging.info(f"Laser Scan: {len(scan.ranges)} ranges")
        )
        self.subscriptions["laser_scan"] = True
        logging.info("✓ Laser scan subscribed")

    def toggle_rgbd_color_info_subscription(self):
        """Toggle RGBD color camera info subscription"""
        if self.subscriptions["rgbd_color_info"]:
            self.sensor_controller.unsubscribe_rgbd_color_camera_info()
            self.subscriptions["rgbd_color_info"] = False
            logging.info("✓ RGBD color camera info unsubscribed")
        else:
            self.sensor_controller.subscribe_rgbd_color_camera_info(
                lambda info: logging.info(f"RGBD Color Info: K={info.K}")
            )
            self.subscriptions["rgbd_color_info"] = True
            logging.info("✓ RGBD color camera info subscribed")

    def toggle_rgbd_depth_image_subscription(self):
        """Toggle RGBD depth image subscription"""
        if self.subscriptions["rgbd_depth_image"]:
            self.sensor_controller.unsubscribe_rgbd_depth_image()
            self.subscriptions["rgbd_depth_image"] = False
            logging.info("✓ RGBD depth image unsubscribed")
        else:
            self.sensor_controller.subscribe_rgbd_depth_image(
                lambda img: logging.info(f"RGBD Depth Image: {len(img.data)} bytes")
            )
            self.subscriptions["rgbd_depth_image"] = True
            logging.info("✓ RGBD depth image subscribed")

    def unsubscribe_rgbd_depth_image_subscription(self):
        """Unsubscribe RGBD depth image subscription"""
        self.sensor_controller.unsubscribe_rgbd_depth_image()
        self.subscriptions["rgbd_depth_image"] = False
        logging.info("✓ RGBD depth image unsubscribed")

    def toggle_rgbd_color_image_subscription(self):
        """Toggle RGBD color image subscription"""
        if self.subscriptions["rgbd_color_image"]:
            self.sensor_controller.unsubscribe_rgbd_color_image()
            self.subscriptions["rgbd_color_image"] = False
            logging.info("✓ RGBD color image unsubscribed")
```

(下页继续)

(续上页)

```
        else:
            self.sensor_controller.subscribe_rgbd_color_image(
                lambda img: logging.info(f"RGBD Color Image: {len(img.data)} bytes")
            )
            self.subscriptions["rgbd_color_image"] = True
            logging.info("✓ RGBD color image subscribed")

    def toggle_rgb_depth_info_subscription(self):
        """Toggle RGB depth camera info subscription"""
        if self.subscriptions["rgb_depth_info"]:
            self.sensor_controller.unsubscribe_rgb_depth_camera_info()
            self.subscriptions["rgb_depth_info"] = False
            logging.info("✓ RGB depth camera info unsubscribed")
        else:
            self.sensor_controller.subscribe_rgb_depth_camera_info(
                lambda info: logging.info(f"RGB Depth Info: K={info.K}")
            )
            self.subscriptions["rgb_depth_info"] = True
            logging.info("✓ RGB depth camera info subscribed")

    def toggle_left_binocular_high_subscription(self):
        """Toggle left binocular high image subscription"""
        if self.subscriptions["left_binocular_high"]:
            self.sensor_controller.unsubscribe_left_binocular_high_img()
            self.subscriptions["left_binocular_high"] = False
            logging.info("✓ Left binocular high image unsubscribed")
        else:
            self.sensor_controller.subscribe_left_binocular_high_img(
                lambda img: logging.info(f"Left Binocular High: {len(img.data)} bytes")
            )
            self.subscriptions["left_binocular_high"] = True
            logging.info("✓ Left binocular high image subscribed")

    def toggle_left_binocular_low_subscription(self):
        """Toggle left binocular low image subscription"""
        if self.subscriptions["left_binocular_low"]:
            self.sensor_controller.unsubscribe_left_binocular_low_img()
            self.subscriptions["left_binocular_low"] = False
            logging.info("✓ Left binocular low image unsubscribed")
        else:
            self.sensor_controller.subscribe_left_binocular_low_img(
                lambda img: logging.info(f"Left Binocular Low: {len(img.data)} bytes")
            )
```

(下页继续)

(续上页)

```
            self.subscriptions["left_binocular_low"] = True
            logging.info("✓ Left binocular low image subscribed")

    def toggle_right_binocular_low_subscription(self):
        """Toggle right binocular low image subscription"""
        if self.subscriptions["right_binocular_low"]:
            self.sensor_controller.unsubscribe_right_binocular_low_img()
            self.subscriptions["right_binocular_low"] = False
            logging.info("✓ Right binocular low image unsubscribed")
        else:
            self.sensor_controller.subscribe_right_binocular_low_img(
                lambda img: logging.info(f"Right Binocular Low: {len(img.data)} bytes")
            )
            self.subscriptions["right_binocular_low"] = True
            logging.info("✓ Right binocular low image subscribed")

    def toggle_depth_image_subscription(self):
        """Toggle depth image subscription"""
        if self.subscriptions["depth_image"]:
            self.sensor_controller.unsubscribe_depth_image()
            self.subscriptions["depth_image"] = False
            logging.info("✓ Depth image unsubscribed")
        else:
            self.sensor_controller.subscribe_depth_image(
                lambda img: logging.info(f"Depth Image: {len(img.data)} bytes")
            )
            self.subscriptions["depth_image"] = True
            logging.info("✓ Depth image subscribed")

    def show_status(self):
        """Display current sensor status"""
        logging.info("\n" + "=" * 70)
        logging.info("SENSOR STATUS")
        logging.info("=" * 70)
        logging.info(
            f"Channel Switch:                {'OPEN' if self.channel_opened else 'CLOSED'}"
        )
        logging.info(
            f"Laser Scan:                    {'OPEN' if self.sensors_state['laser_scan'] else
↪'CLOSED'}"
        )
        logging.info(
            f"RGBD Camera:                   {'OPEN' if self.sensors_state['rgbd_camera'] else
```

(下页继续)

(续上页)

```
↪'CLOSED'}"
        )
        logging.info(
            f"Binocular Camera:              {'OPEN' if self.sensors_state['binocular_camera']␣
↪else 'CLOSED'}"
        )
        logging.info("\nSUBSCRIPTIONS:")
        logging.info(
            f"  Ultra:                       {'✓ SUBSCRIBED' if self.subscriptions['ultra']␣
↪else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  Head Touch:                  {'✓ SUBSCRIBED' if self.subscriptions['head_touch
↪'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  IMU:                         {'✓ SUBSCRIBED' if self.subscriptions['imu'] else
↪'✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  Laser Scan:                  {'✓ SUBSCRIBED' if self.subscriptions['laser_scan
↪'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  RGBD Color Info:             {'✓ SUBSCRIBED' if self.subscriptions['rgbd_color_
↪info'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  RGBD Depth Image:            {'✓ SUBSCRIBED' if self.subscriptions['rgbd_depth_
↪image'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  RGBD Color Image:            {'✓ SUBSCRIBED' if self.subscriptions['rgbd_color_
↪image'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  RGB Depth Info:              {'✓ SUBSCRIBED' if self.subscriptions['rgb_depth_
↪info'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  Left Binocular High:         {'✓ SUBSCRIBED' if self.subscriptions['left_
↪binocular_high'] else '✗ UNSUBSCRIBED'}"
        )
```

(下页继续)

(续上页)

```
        logging.info(
            f"  Left Binocular Low:          {'✓ SUBSCRIBED' if self.subscriptions['left_
↪binocular_low'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  Right Binocular Low:         {'✓ SUBSCRIBED' if self.subscriptions['right_
↪binocular_low'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info(
            f"  Depth Image:                 {'✓ SUBSCRIBED' if self.subscriptions['depth_image
↪'] else '✗ UNSUBSCRIBED'}"
        )
        logging.info("=" * 70 + "\n")


def print_menu():
    """Print interactive menu"""
    logging.info("\n" + "=" * 70)
    logging.info("SENSOR CONTROL MENU")
    logging.info("=" * 70)
    logging.info("Channel Control:")
    logging.info("  1 - Open Channel Switch       2 - Close Channel Switch")
    logging.info("\nSensor Control:")
    logging.info("  3 - Open Laser Scan           4 - Close Laser Scan")
    logging.info("  5 - Open RGBD Camera          6 - Close RGBD Camera")
    logging.info("  7 - Open Binocular Camera     8 - Close Binocular Camera")
    logging.info("\nSubscription Toggle (lowercase=toggle, UPPERCASE=unsubscribe):")
    logging.info("  u/U - Ultra                   l/L - Laser Scan Data")
    logging.info("  h/H - Head Touch              i/I - IMU")
    logging.info("\nRGBD Subscriptions (lowercase=toggle, UPPERCASE=unsubscribe):")
    logging.info("  r/R - RGBD Color Info           d/D - RGBD Depth Image")
    logging.info("  c/C - RGBD Color Image          p/P - RGB Depth Info")
    logging.info("\nBinocular Subscriptions:")
    logging.info("  b/B - Left Binocular High       n/N - Left Binocular Low")
    logging.info("  m/M - Right Binocular Low")
    logging.info("\nOther Subscriptions:")
    logging.info("  e/E - Depth Image")
    logging.info("\nCommands:")
    logging.info("  s - Show Status               ESC - Quit              ? - Help")
    logging.info("=" * 70)


def main():
```

(下页继续)

(续上页)

```
    """Main function"""
    logging.info("\n" + "=" * 70)
    logging.info("MagicDog SDK Sensor Interactive Example")
    logging.info("=" * 70 + "\n")

    local_ip = "192.168.55.10"
    robot = magicdog.MagicRobot()

    if not robot.initialize(local_ip):
        logging.error("Robot initialization failed")
        return

    if not robot.connect():
        logging.error("Robot connection failed")
        robot.shutdown()
        return

    logging.info("✓ Robot connected successfully\n")

    sensor_controller = robot.get_sensor_controller()
    sensor_manager = SensorManager(robot, sensor_controller)

    print_menu()

    try:
        while True:
            choice = input("\nEnter your choice: ").strip().lower()

            if choice == "\x1b":  # ESC key
                logging.info("ESC key pressed, exiting program...")
                break

            # Channel control
            if choice == "1":
                sensor_manager.open_channel()
            elif choice == "2":
                sensor_manager.close_channel()

            # Sensor control
            elif choice == "3":
                sensor_manager.open_laser_scan()
            elif choice == "4":
                sensor_manager.close_laser_scan()
```

(下页继续)

(续上页)

```
            elif choice == "5":
                sensor_manager.open_rgbd_camera()
            elif choice == "6":
                sensor_manager.close_rgbd_camera()
            elif choice == "7":
                sensor_manager.open_binocular_camera()
            elif choice == "8":
                sensor_manager.close_binocular_camera()

            # Basic sensor subscriptions
            elif choice.lower() == "u":
                sensor_manager.toggle_ultra_subscription()
            elif choice.lower() == "h":
                sensor_manager.toggle_head_touch_subscription()
            elif choice.lower() == "i":
                sensor_manager.toggle_imu_subscription()
            elif choice.lower() == "l":
                sensor_manager.toggle_laser_scan_subscription()

            # RGBD subscriptions
            elif choice.lower() == "r":
                sensor_manager.toggle_rgbd_color_info_subscription()
            elif choice.lower() == "d":
                sensor_manager.toggle_rgbd_depth_image_subscription()
            elif choice.lower() == "c":
                sensor_manager.toggle_rgbd_color_image_subscription()
            elif choice.lower() == "p":
                sensor_manager.toggle_rgb_depth_info_subscription()
            # Binocular subscriptions
            elif choice.lower() == "b":
                sensor_manager.toggle_left_binocular_high_subscription()
            elif choice.lower() == "n":
                sensor_manager.toggle_left_binocular_low_subscription()
            elif choice.lower() == "m":
                sensor_manager.toggle_right_binocular_low_subscription()
            elif choice.lower() == "e":
                sensor_manager.toggle_depth_image_subscription()

            # Commands
            elif choice.lower() == "s":
                sensor_manager.show_status()
            elif choice == "?" or choice == "help":
                print_menu()
```

(下页继续)

(续上页)

```
            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...")

        # Unsubscribe from all active subscriptions
        for sensor_name, is_subscribed in sensor_manager.subscriptions.items():
            if is_subscribed:
                logging.info(f"Unsubscribing from {sensor_name}...")

        if sensor_manager.sensors_state["laser_scan"]:
            sensor_manager.close_laser_scan()
        if sensor_manager.sensors_state["rgbd_camera"]:
            sensor_manager.close_rgbd_camera()
        if sensor_manager.sensors_state["binocular_camera"]:
            sensor_manager.close_binocular_camera()
        if sensor_manager.channel_opened:
            sensor_manager.close_channel()

        # Allow time for cleanup
        time.sleep(1)

        sensor_controller.shutdown()
        logging.info("sensor controller shutdown")

        robot.disconnect()
        logging.info("robot disconnect")

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

- 通道控制：
  - 按键 `1` - 打开通道开关
  - 按键 `2` - 关闭通道开关
- 传感器控制：
  - 按键 `3` - 打开激光扫描
  - 按键 `4` - 关闭激光扫描
  - 按键 `5` - 打开 RGBD 相机
  - 按键 `6` - 关闭 RGBD 相机
  - 按键 `7` - 打开双目相机
  - 按键 `8` - 关闭双目相机
- 基本传感器订阅：
  - 按键 `u/U` - 切换/取消订阅超声波传感器
  - 按键 `h/H` - 切换/取消订阅头部触摸传感器
  - 按键 `i/I` - 切换/取消订阅 IMU 传感器
  - 按键 `l/L` - 切换/取消订阅激光扫描数据
- RGBD 相机订阅：
  - 按键 `r/R` - 切换/取消订阅 RGBD 彩色信息
  - 按键 `d/D` - 切换/取消订阅 RGBD 深度图像
  - 按键 `c/C` - 切换/取消订阅 RGBD 彩色图像

  - 按键 `p/P` - 切换/取消订阅 RGB 深度信息

- 双目相机订阅：
  - 按键 `b/B` - 切换/取消订阅左眼高分辨率图像
  - 按键 `n/N` - 切换/取消订阅左眼低分辨率图像
  - 按键 `m/M` - 切换/取消订阅右眼低分辨率图像
  - 按键 `e/E` - 切换/取消订阅深度图像
- 其他功能：
  - 按键 `s` - 显示状态
  - 按键 `?` - 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 MagicDog 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）等基本操作。

24.1 C++

示例文件：`audio_example.cpp`

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

#include <termios.h>
#include <unistd.h>
#include <chrono>
#include <csignal>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <thread>

using namespace magic::dog;

// Global robot instance
std::unique_ptr<MagicRobot> robot = nullptr;
```

(下页继续)

(续上页)

```
void signalHandler(int signum) {
  std::cout << "\nInterrupt signal (" << signum << ") received." << std::endl;
  if (robot) {
    robot->Shutdown();
  }
  exit(signum);
}

void print_help() {
  std::cout << "Key Function Demo Program\n"
            << std::endl;
  std::cout << "Key Function Description:" << std::endl;
  std::cout << "Audio Functions:" << std::endl;
  std::cout << "  1        Function 1: Get volume" << std::endl;
  std::cout << "  2        Function 2: Set volume" << std::endl;
  std::cout << "  3        Function 3: Play TTS" << std::endl;
  std::cout << "  4        Function 4: Stop playback" << std::endl;
  std::cout << "  5        Function 5: Get voice config" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Audio stream Functions:" << std::endl;
  std::cout << "  6        Function 6: Open audio stream" << std::endl;
  std::cout << "  7        Function 7: Close audio stream" << std::endl;
  std::cout << "  8        Function 8: Subscribe to audio stream" << std::endl;
  std::cout << "  9        Function 9: Unsubscribe from audio stream" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Speech IO Functions:" << std::endl;
  std::cout << "  d        Function d: Open speech io" << std::endl;
  std::cout << "  e        Function e: Close speech io" << std::endl;
  std::cout << "  f        Function f: Subscribe to speech asr" << std::endl;
  std::cout << "  g        Function g: Unsubscribe from speech asr" << std::endl;
  std::cout << "  h        Function h: Publish speech tts" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "  ?        Function ?: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

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
```

(下页继续)

(续上页)

```
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  ch = getchar();                              // Read key press
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);  // Restore settings
  return ch;
}

void get_volume() {
  auto& audio_controller = robot->GetAudioController();

  int volume = 0;
  auto status = audio_controller.GetVolume(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: " << volume << std::endl;
}

void set_volume(int volume) {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_controller.SetVolume(volume);
  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 play_tts() {
  auto& audio_controller = robot->GetAudioController();

  TtsCommand tts;
  tts.id = "100000000001";
  tts.content = "How's the weather today!";
  tts.priority = TtsPriority::HIGH;
  tts.mode = TtsMode::CLEARTOP;

  auto status = audio_controller.Play(tts);
  if (status.code != ErrorCode::OK) {
    std::cerr << "play tts failed, code: " << status.code
```

(下页继续)

(续上页)

```
              << ", message: " << status.message << std::endl;
    return;
  }
  std::cout << "play tts success" << std::endl;
}

void stop_tts() {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_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 open_audio_stream() {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_controller.ControlVoiceStream(true, true);
  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 close_audio_stream() {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_controller.ControlVoiceStream(false, false);
  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;
}

// Global counters for audio stream callbacks
```

(下页继续)

(续上页)

```cpp
static int origin_counter = 0;
static int bf_counter = 0;

void subscribe_audio_stream() {
  auto& audio_controller = robot->GetAudioController();

  // Subscribe to origin voice data
  audio_controller.SubscribeOriginVoiceData([](const std::shared_ptr<ByteMultiArray> data) {
    if (origin_counter % 30 == 0) {
      std::cout << "Received origin voice data, size: " << data->data.size() << std::endl;
      std::cout << "\r";
      std::cout.flush();
    }
    origin_counter++;
  });

  // Subscribe to bf voice data
  audio_controller.SubscribeBfVoiceData([](const std::shared_ptr<ByteMultiArray> data) {
    if (bf_counter % 30 == 0) {
      std::cout << "Received bf voice data, size: " << data->data.size() << std::endl;
      std::cout << "\r";
      std::cout.flush();
    }
    bf_counter++;
  });

  std::cout << "Subscribed to audio streams" << std::endl;
}

void unsubscribe_audio_stream() {
  auto& audio_controller = robot->GetAudioController();

  audio_controller.UnsubscribeOriginVoiceData();
  audio_controller.UnsubscribeBfVoiceData();
  std::cout << "Unsubscribed from audio streams" << std::endl;
}

void get_voice_config() {
  auto& audio_controller = robot->GetAudioController();

  GetSpeechConfig voice_config;
  auto status = audio_controller.GetVoiceConfig(voice_config);
  if (status.code != ErrorCode::OK) {
```

(下页继续)

(续上页)

```
    std::cerr << "get voice config failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }

  std::cout << "Get voice config success" << std::endl;
  std::cout << "TTS type: " << static_cast<int>(voice_config.tts_type) << std::endl;
  std::cout << "Speaker: " << voice_config.speaker_config.selected.speaker_id << std::endl;
  std::cout << "Bot config: " << voice_config.bot_config.selected.bot_id << std::endl;
  std::cout << "Wake word: " << voice_config.wakeup_config.name << std::endl;
  std::cout << "Dialog config - Front DOA: " << voice_config.dialog_config.is_front_doa <<␣
↪std::endl;
  std::cout << "Dialog config - Full duplex: " << voice_config.dialog_config.is_fullduplex_
↪enable << std::endl;
  std::cout << "Dialog config - Voice enable: " << voice_config.dialog_config.is_enable <<␣
↪std::endl;
  std::cout << "Dialog config - DOA enable: " << voice_config.dialog_config.is_doa_enable <<␣
↪std::endl;
}

void open_speech_io() {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_controller.ControlSpeechIO(true);
  if (status.code != ErrorCode::OK) {
    std::cerr << "open speech io failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }
  std::cout << "open speech io success" << std::endl;
}

void close_speech_io() {
  auto& audio_controller = robot->GetAudioController();

  auto status = audio_controller.ControlSpeechIO(false);
  if (status.code != ErrorCode::OK) {
    std::cerr << "close speech io failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }
  std::cout << "close speech io success" << std::endl;
}
```

(下页继续)

(续上页)

```
void subscribe_speech_asr() {
  auto& audio_controller = robot->GetAudioController();

  audio_controller.SubscribeSpeechASRStream([](const std::shared_ptr<SpeechASRStream> data) {
    std::cout << "Received speech asr data, id: " << data->id << std::endl;
    std::cout << "                          type: " << data->type << std::endl;
    std::cout << "                          text: " << data->text << std::endl;
    std::cout << "\r";
    std::cout.flush();
  });

  std::cout << "Subscribed to speech asr" << std::endl;
}

void unsubscribe_speech_asr() {
  auto& audio_controller = robot->GetAudioController();

  audio_controller.UnsubscribeSpeechASRStream();
  std::cout << "Unsubscribed from speech asr" << std::endl;
}

void publish_speech_tts(const SpeechTTSStream& data) {
  auto& audio_controller = robot->GetAudioController();
  // begin
  // var: Intermediate result
  // end
  audio_controller.PublishSpeechTTSStream(data);
  std::cout << "Published to speech tts" << std::endl;
}

SpeechTTSStream get_speech_tts_input() {
  SpeechTTSStream stream;

  std::cout << "=== Speech TTS Stream Input ===" << std::endl;

  // Enter ID
  std::cout << "Enter request ID: ";
  std::getline(std::cin, stream.id);

  // Enter Type and verify.
  while (true) {
    std::cout << "Enter type (begin/var/end): ";
```

(下页继续)

(续上页)

```
    std::getline(std::cin, stream.type);
    if (stream.type == "begin" || stream.type == "var" || stream.type == "end") {
      break;
    }
    std::cout << "Invalid type! Please enter 'begin', 'var', or 'end'." << std::endl;
  }

  // Enter Text
  std::cout << "Enter TTS text: ";
  std::getline(std::cin, stream.text);

  // Enter End Session
  std::cout << "End session? (y/n): ";
  std::string end_session_input;
  std::getline(std::cin, end_session_input);
  stream.end_session = (end_session_input == "y" || end_session_input == "Y");

  return stream;
}

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

  print_help();

  std::string local_ip = "192.168.55.10";
  robot = std::make_unique<MagicRobot>();

  // 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;
```

(下页继续)

(续上页)

```
}

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

// Wait for user input
while (true) {
  int key = getch();
  if (key == 27) {  // ESC key
    break;
  }

  std::cout << "Key ASCII: " << key << ", Character: " << static_cast<char>(key) << std::endl;

  switch (key) {
    case '1':
      get_volume();
      break;
    case '2': {
      int volume = 50;
      std::cout << "Please input volume: ";
      std::cin >> volume;
      set_volume(volume);
      std::cin.ignore();
    } break;
    case '3':
      play_tts();
      break;
    case '4':
      stop_tts();
      break;
    case '5':
      get_voice_config();
      break;
    case '6':
      open_audio_stream();
      break;
    case '7':
      close_audio_stream();
      break;
    case '8':
      subscribe_audio_stream();
      break;
    case '9':
```

(下页继续)

(续上页)

```
      unsubscribe_audio_stream();
      break;
    case 'd':
      open_speech_io();
      break;
    case 'e':
      close_speech_io();
      break;
    case 'f':
      subscribe_speech_asr();
      break;
    case 'g':
      unsubscribe_speech_asr();
      break;
    case 'h': {
      SpeechTTSStream data;
      data = get_speech_tts_input();
      publish_speech_tts(data);
    } break;
    case '?':
      print_help();
      break;
    default:
      std::cout << "Unknown key: " << key << std::endl;
      break;
  }

  // Sleep 10ms, equivalent to usleep(10000)
  std::this_thread::sleep_for(std::chrono::milliseconds(10));
}

// 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;
}

std::cout << "disconnect robot success" << std::endl;

robot->Shutdown();
```

(下页继续)

(续上页)

```
  std::cout << "robot shutdown" << std::endl;

  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 termios
import tty
import logging
from typing import Optional
import magicdog_python as magicdog
from magicdog_python import TtsCommand, TtsPriority, TtsMode, GetSpeechConfig

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

# Global robot instance
robot = None


def signal_handler(signum, frame):
    """Handle Ctrl+C signal"""
    logging.info(f"\nInterrupt signal ({signum}) received.")
    if robot:
        robot.shutdown()
    sys.exit(signum)


def print_help():
    """Print help information"""
```

(下页继续)

(续上页)

```
    logging.info("Key Function Description:")
    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("  5        Function 5: Get voice config")
    logging.info("")
    logging.info("Audio stream Functions:")
    logging.info("  6        Function 6: Open audio stream")
    logging.info("  7        Function 7: Close audio stream")
    logging.info("  8        Function 8: Subscribe to audio stream")
    logging.info("  9        Function 9: Unsubscribe from audio stream")
    logging.info("")
    logging.info("Speech IO Functions:")
    logging.info("  d        Function d: Open speech io")
    logging.info("  e        Function e: Close speech io")
    logging.info("  f        Function f: Subscribe to speech asr")
    logging.info("  g        Function g: Unsubscribe from speech asr")
    logging.info("  h        Function h: Publish speech tts")
    logging.info("")
    logging.info("  ?        Function ?: Print help")
    logging.info("  ESC      Exit program")


def getch():
    """Get a single character from standard input"""
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(fd)
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch


def get_volume(audio_controller):
    """Get current volume"""
    status, volume = audio_controller.get_volume()
    if status.code != magicdog.ErrorCode.OK:
        logging.error("get volume failed")
```

(下页继续)

(续上页)

```
        return
    logging.info(f"get volume success, volume: {volume}")


def set_volume(audio_controller):
    """Set volume"""
    status = audio_controller.set_volume(50)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"set volume failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("set volume success")


def play_tts(audio_controller):
    """Play TTS"""
    tts = TtsCommand()
    tts.id = "100000000001"
    tts.content = "How's the weather today!"
    tts.priority = TtsPriority.HIGH
    tts.mode = TtsMode.CLEARTOP
    status = audio_controller.play(tts)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"play tts failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("play tts success")


def stop_tts(audio_controller):
    """Stop TTS playback"""
    status = audio_controller.stop()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"stop tts failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("stop tts success")


def open_audio_stream(audio_controller):
```

(下页继续)

(续上页)

```
    """Open audio stream"""
    status = audio_controller.control_voice_stream(True, True)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"open audio stream failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("open audio stream success")


def close_audio_stream(audio_controller):
    """Close audio stream"""
    status = audio_controller.control_voice_stream(False, False)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"close audio stream failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("close audio stream success")


# Global counter for audio stream callbacks
origin_counter = [0]
bf_counter = [0]


def subscribe_audio_stream(audio_controller):
    """Subscribe to audio streams"""

    def origin_callback(data):
        if origin_counter[0] % 30 == 0:
            logging.info(f"Received origin voice data, size: {len(data.data)}")
            sys.stdout.write("\r")
            sys.stdout.flush()
        origin_counter[0] += 1

    def bf_callback(data):
        if bf_counter[0] % 30 == 0:
            logging.info(f"Received bf voice data, size: {len(data.data)}")
            sys.stdout.write("\r")
            sys.stdout.flush()
        bf_counter[0] += 1
```

(下页继续)

(续上页)

```
    audio_controller.subscribe_origin_voice_data(origin_callback)
    audio_controller.subscribe_bf_voice_data(bf_callback)
    logging.info("Subscribed to audio streams")


def unsubscribe_audio_stream(audio_controller):
    """Unsubscribe from audio streams"""
    audio_controller.unsubscribe_origin_voice_data()
    audio_controller.unsubscribe_bf_voice_data()
    logging.info("Unsubscribed from audio streams")


def get_voice_config(audio_controller):
    """Get voice configuration"""
    status, voice_config = audio_controller.get_voice_config()
    if status.code != magicdog.ErrorCode.OK:
        logging.error("get voice config failed")
        return

    logging.info("Get voice config success")
    logging.info(f"voice config tts type: {int(voice_config.tts_type)}")
    logging.info(
        f"voice config speaker config: {voice_config.speaker_config.selected.speaker_id}"
    )
    logging.info(f"voice config bot config: {voice_config.bot_config.selected.bot_id}")
    logging.info(f"voice config wakeup config: {voice_config.wakeup_config.name}")
    logging.info(
        f"voice config dialog config: {voice_config.dialog_config.is_front_doa}"
    )
    logging.info(
        f"voice config dialog config: {voice_config.dialog_config.is_fullduplex_enable}"
    )
    logging.info(f"voice config dialog config: {voice_config.dialog_config.is_enable}")
    logging.info(
        f"voice config dialog config: {voice_config.dialog_config.is_doa_enable}"
    )


def open_speech_io(audio_controller):
    """Open speech IO"""
    status = audio_controller.control_speech_io(True)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
```

(下页继续)

(续上页)

```
            f"open speech io failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("open speech io success")


def close_speech_io(audio_controller):
    """Close speech IO"""
    status = audio_controller.control_speech_io(False)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"close speech io failed, code: {status.code}, message: {status.message}"
        )
        return
    logging.info("close speech io success")


def subscribe_speech_asr(audio_controller):
    """Subscribe to speech ASR"""

    def asr_callback(data):
        logging.info(
            f"Received speech asr, id: {data.id}, type: {data.type}, text: {data.text}"
        )
        sys.stdout.write("\r")
        sys.stdout.flush()

    audio_controller.subscribe_speech_asr(asr_callback)
    logging.info("Subscribed to speech asr")


def unsubscribe_speech_asr(audio_controller):
    """Unsubscribe from speech ASR"""
    audio_controller.unsubscribe_speech_asr()
    logging.info("Unsubscribed from speech ASR")


def get_speech_tts_input() -> magicdog.SpeechTTSStream:
    """Obtaining TTS stream data from user input"""
    print("=== Speech TTS Stream Input ===")

    # Enter ID
    stream_id = input("Enter request ID: ").strip()
```

(下页继续)

(续上页)

```
    # Enter Type and verify
    while True:
        stream_type = input("Enter type (begin/var/end): ").strip().lower()
        if stream_type in ["begin", "var", "end"]:
            break
        print("Invalid type! Please enter 'begin', 'var', or 'end'.")

    # Enter Text
    text = input("Enter TTS text: ").strip()

    # Enter End Session
    end_session_input = input("End session? (y/n): ").strip().lower()
    end_session = end_session_input in ["y", "yes"]

    stream = magicdog.SpeechTTSStream()  # 无参构造
    stream.id = stream_id
    stream.type = stream_type
    stream.text = text
    stream.end_session = end_session

    return stream


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

    # Bind SIGINT (Ctrl+C)
    signal.signal(signal.SIGINT, signal_handler)

    print_help()

    local_ip = "192.168.55.10"
    robot = magicdog.MagicRobot()

    # Configure local IP address for direct network connection and initialize SDK
    if not robot.initialize(local_ip):
        logging.error("robot sdk initialize failed.")
        robot.shutdown()
        return -1

    # Connect to robot
```

(下页继续)

(续上页)

```
    status = robot.connect()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"connect robot failed, code: {status.code}, message: {status.message}"
        )
        robot.shutdown()
        return -1

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

    audio_controller = robot.get_audio_controller()

    # Wait for user input
    while True:
        key = getch()
        if key == "\x1b":  # ESC key
            break

        key_ascii = ord(key)
        logging.info(f"Key ASCII: {key_ascii}, Character: {key}")

        # 1. Audio Functions
        # 1.1 Get volume
        if key == "1":
            get_volume(audio_controller)
        # 1.2 Set volume
        elif key == "2":
            set_volume(audio_controller)
        # 1.3 Play TTS
        elif key == "3":
            play_tts(audio_controller)
        # 1.4 Stop TTS
        elif key == "4":
            stop_tts(audio_controller)
        # 1.5 Get voice config
        elif key == "5":
            get_voice_config(audio_controller)
        # 2. Audio stream Functions
        # 2.1 Open audio stream
        elif key == "6":
            open_audio_stream(audio_controller)
        # 2.2 Close audio stream
        elif key == "7":
```

(下页继续)

(续上页)

```
            close_audio_stream(audio_controller)
        # 2.3 Subscribe to audio stream
        elif key == "8":
            subscribe_audio_stream(audio_controller)
        # 2.4 Unsubscribe from audio stream
        elif key == "9":
            unsubscribe_audio_stream(audio_controller)
        # 4. Speech IO Functions
        # 4.1 Open speech IO
        elif key == "d":
            open_speech_io(audio_controller)
        # 4.2 Close speech IO
        elif key == "e":
            close_speech_io(audio_controller)
        # 4.3 Subscribe to speech ASR
        elif key == "f":
            subscribe_speech_asr(audio_controller)
        # 4.4 Unsubscribe from speech ASR
        elif key == "g":
            unsubscribe_speech_asr(audio_controller)
        # 4.5 Publish speech TTS
        elif key == "h":
            data = get_speech_tts_input()
            audio_controller.publish_speech_tts(data)
        # Help
        elif key == "?":
            print_help()
        else:
            logging.info(f"Unknown key: {key_ascii}")

        # Sleep 10ms, equivalent to usleep(10000)
        time.sleep(0.01)

    audio_controller.shutdown()
    logging.info("audio controller shutdown")

    # Disconnect from robot
    robot.disconnect()
    logging.info("disconnect robot success")

    robot.shutdown()
    logging.info("robot shutdown")
```

(下页继续)

(续上页)

```
    return 0


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

24.3 运行说明

24.3.1 环境准备

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$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:

- 音频功能:
  - 按键 1 - 获取音量
  - 按键 2 - 设置音量
  - 按键 3 - 播放 TTS 语音合成
  - 按键 4 - 停止播放
  - 按键 5 - 获取语音配置
- 音频流功能:
  - 按键 6 - 打开音频流
  - 按键 7 - 关闭音频流
  - 按键 8 - 订阅音频流
  - 按键 9 - 取消订阅音频流
- 语音 IO 功能:
  - 按键 d - 打开语音 IO

  - 按键 `e` - 关闭语音 IO
  - 按键 `f` - 订阅语音 asr
  - 按键 `g` - 取消订阅语音 asr
  - 按键 `h` - 发布语音 tts
- 其他功能：
  - 按键 `?` - 显示帮助菜单

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 MagicDog 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`

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

#include <unistd.h>
#include <csignal>
#include <iostream>
#include <memory>
#include <chrono>
#include <thread>
#include <cstdlib>

using namespace magic::dog;

// Global robot instance
std::unique_ptr<MagicRobot> robot = nullptr;
```

(下页继续)

(续上页)

```
void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received." << std::endl;
    if (robot) {
        robot->Shutdown();
    }
    exit(signum);
}

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

    std::cout << "MagicDog SDK C++ Example Program" << std::endl;

    robot = std::make_unique<MagicRobot>();
    if (!robot->Initialize("192.168.55.10")) {
        std::cerr << "Initialization failed" << std::endl;
        return -1;
    }

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

    std::this_thread::sleep_for(std::chrono::seconds(5));

    auto& monitor = robot->GetStateMonitor();

    RobotState robot_state;
    status = monitor.GetCurrentState(robot_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: " << robot_state.bms_data.battery_health
              << ", Percentage: " << robot_state.bms_data.battery_percentage
```

(下页继续)

(续上页)

```
              << ", State: " << static_cast<int>(robot_state.bms_data.battery_state)
              << ", Power supply status: " << static_cast<int>(robot_state.bms_data.power_
↪supply_status)
              << std::endl;

    auto& faults = robot_state.faults;
    for (const auto& fault : faults) {
        std::cout << "Code: " << fault.error_code
                  << ", Message: " << fault.error_message << std::endl;
    }

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

    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;

    std::cout << "\nExample program execution completed!" << std::endl;

    return 0;
}
```

25.2 Python

示例文件：monitor_example.py

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

import sys
import time
import logging
from typing import Optional
import magicdog_python as magicdog

logging.basicConfig(
    level=logging.INFO,  # Minimum log level
```

(下页继续)

(续上页)

```
    format="%(asctime)s [%(levelname)s] %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)


def main():
    """Main function"""
    logging.info("MagicDog SDK Python Example Program")

    robot = magicdog.MagicRobot()
    if not robot.initialize("192.168.55.10"):
        logging.error("Initialization failed")
        return

    if not robot.connect():
        logging.error("Connection failed")
        robot.shutdown()
        return

    time.sleep(5)

    monitor = robot.get_state_monitor()

    status, state = monitor.get_current_state()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Get current state failed: {status.message}")
        robot.shutdown()
        return

    logging.info(
        f"health: {state.bms_data.battery_health}, percentage: {state.bms_data.battery_
↪percentage}, state: {state.bms_data.battery_state}, power_supply_status: {state.bms_data.
↪power_supply_status}"
    )

    for fault in state.faults:
        logging.info(f"code: {fault.error_code}, message: {fault.error_message}")

    robot.disconnect()
    robot.shutdown()

    logging.info("\nExample program execution completed!")
```

(下页继续)

(续上页)

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

25.3 运行说明

25.3.1 环境准备

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$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 MagicDog 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 <fcntl.h>
#include <signal.h>
#include <termios.h>
#include <unistd.h>
#include <algorithm>
#include <atomic>
#include <chrono>
```

(下页继续)

(续上页)

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <sstream>
#include <string>
#include <thread>
#include <vector>

using namespace magic::dog;
using namespace std;

// Global variables
std::unique_ptr<MagicRobot> robot = nullptr;
std::atomic<bool> running(true);
std::string current_slam_mode = "IDLE";
NavMode current_nav_mode = NavMode::IDLE;

void signalHandler(int signum) {
  std::cout << "Interrupt signal (" << signum << ") received.\n";
  running = false;
  if (robot) {
    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;
  }
  exit(-1);
}

void printHelp() {
  std::cout << "SLAM Function Demo Program" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "preparation Functions:" << std::endl;
  std::cout << "  1        Function 1: Recovery stand" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "SLAM Functions:" << 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" << std::endl;
  std::cout << "  6        Function 6: Delete map" << std::endl;
  std::cout << "  7        Function 7: Get all map information and save map image as PGM file" <
↪< std::endl;
  std::cout << "" << std::endl;
```

(下页继续)

(续上页)

```
  std::cout << "Joystick Functions:" << std::endl;
  std::cout << "  W        Function W: forward" << std::endl;
  std::cout << "  A        Function A: backward" << std::endl;
  std::cout << "  S        Function S: left" << std::endl;
  std::cout << "  D        Function D: 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 << "" << std::endl;
  std::cout << "Close Functions:" << std::endl;
  std::cout << "  P        Function P: Close SLAM" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "  ?        Function H: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

// ==================== SLAM Functions ====================

void loadMap(const std::string& mapToLoad) {
  try {
    if (mapToLoad.empty()) {
      std::cerr << "Map to load is not provided" << std::endl;
      return;
    }

    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    std::cout << "Loading map: " << mapToLoad << std::endl;
    controller.LoadMap(mapToLoad);
    std::cout << "Successfully loaded map: " << mapToLoad << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while loading map: " << e.what() << std::endl;
  }
}

void startMapping() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Start mapping
    auto status = controller.StartMapping();
```

(下页继续)

(续上页)

```
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to start mapping, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    current_slam_mode = "MAPPING";
    std::cout << "Successfully started mapping" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while starting mapping: " << e.what() << std::endl;
  }
}

void cancelMapping() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Cancel mapping
    auto status = controller.CancelMapping();
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to cancel mapping, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    std::cout << "Successfully cancelled mapping" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while cancelling mapping: " << e.what() << std::endl;
  }
}

void saveMap() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Check if in mapping mode
    if (current_slam_mode != "MAPPING") {
      std::cout << "Warning: Currently not in mapping mode, may not be able to save map" <<␣
↪std::endl;
    }
```

(下页继续)

(续上页)

```
    // Generate map name with timestamp
    auto now = std::chrono::system_clock::now();
    auto timestamp = std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch()).
↪count();
    std::string mapName = "map_" + std::to_string(timestamp);

    std::cout << "Saving map: " << mapName << std::endl;

    // Save map
    auto status = controller.SaveMap(mapName);
    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: " << mapName << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while saving map: " << e.what() << std::endl;
  }
}

void deleteMap(const std::string& mapToDelete) {
  try {
    if (mapToDelete.empty()) {
      std::cerr << "Map to delete is not provided" << std::endl;
      return;
    }

    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    std::cout << "Deleting map: " << mapToDelete << std::endl;

    // Delete map
    auto status = controller.DeleteMap(mapToDelete);
    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: " << mapToDelete << std::endl;
```

(下页继续)

(续上页)

```
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while deleting map: " << e.what() << std::endl;
  }
}

void getAllMapInfo() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Get all map information
    AllMapInfo allMapInfo;
    auto status = controller.GetAllMapInfo(allMapInfo);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to get map information, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    std::cout << "Successfully retrieved map information" << std::endl;
    std::cout << "Current map: " << allMapInfo.current_map_name << std::endl;
    std::cout << "Total maps: " << allMapInfo.map_infos.size() << std::endl;

    if (!allMapInfo.map_infos.empty()) {
      std::cout << "Map details:" << std::endl;
      for (size_t i = 0; i < allMapInfo.map_infos.size(); ++i) {
        const auto& mapInfo = allMapInfo.map_infos[i];
        std::cout << "  Map " << (i + 1) << ": " << mapInfo.map_name << std::endl;
        std::cout << "    Origin: ["
                  << mapInfo.map_meta_data.origin.position[0] << ", "
                  << mapInfo.map_meta_data.origin.position[1] << ", "
                  << mapInfo.map_meta_data.origin.position[2] << "]" << std::endl;
        std::cout << "    Orientation: ["
                  << mapInfo.map_meta_data.origin.orientation[0] << ", "
                  << mapInfo.map_meta_data.origin.orientation[1] << ", "
                  << mapInfo.map_meta_data.origin.orientation[2] << "]" << std::endl;
        std::cout << "    Resolution: " << mapInfo.map_meta_data.resolution << " m/pixel" <<␣
↪std::endl;
        std::cout << "    Size: " << mapInfo.map_meta_data.map_image_data.width
                  << " x " << mapInfo.map_meta_data.map_image_data.height << std::endl;
        std::cout << "    Max gray value: " << mapInfo.map_meta_data.map_image_data.max_gray_
↪value << std::endl;
        std::cout << "    Image type: " << mapInfo.map_meta_data.map_image_data.type <<␣
```

(下页继续)

(续上页)

```
↪std::endl;
      }
    } else {
      std::cout << "No available maps" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while getting map information: " << e.what() << std::endl;
  }
}

void closeSlam() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Switch to idle mode to close SLAM
    auto status = controller.SwitchToIdle();
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to close SLAM, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    current_slam_mode = "IDLE";
    std::cout << "Successfully closed SLAM system" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while closing SLAM: " << e.what() << std::endl;
  }
}

// Switch gait to down climb stairs mode
bool changeGaitToDownClimbStairs() {
  try {
    auto& highController = robot->GetHighLevelMotionController();
    GaitMode currentGait;
    auto status = highController.GetGait(currentGait);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to get gait: " << status.message << std::endl;
      return false;
    }

    if (currentGait != GaitMode::GAIT_DOWN_CLIMB_STAIRS) {
      auto status = highController.SetGait(GaitMode::GAIT_DOWN_CLIMB_STAIRS);
```

(下页继续)

(续上页)

```
      if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set down climb stairs gait: " << status.message << std::endl;
        return false;
      }

      // Wait for gait switch to complete
      while (currentGait != GaitMode::GAIT_DOWN_CLIMB_STAIRS) {
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
        status = highController.GetGait(currentGait);
        if (status.code != ErrorCode::OK) {
          std::cerr << "Failed to get gait during transition: " << status.message << std::endl;
          return false;
        }
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
      }
    }

    std::cout << "Gait changed to down climb stairs" << std::endl;
    return true;
  } catch (const std::exception& e) {
    std::cerr << "Error changing gait: " << e.what() << std::endl;
    return false;
  }
}

void sendJoystickCommand(double leftX, double leftY, double rightX, double rightY) {
  if (!changeGaitToDownClimbStairs()) {
    return;
  }

  auto& highController = robot->GetHighLevelMotionController();
  JoystickCommand joyCommand;
  joyCommand.left_x_axis = leftX;
  joyCommand.left_y_axis = leftY;
  joyCommand.right_x_axis = rightX;
  joyCommand.right_y_axis = rightY;

  auto status = highController.SendJoyStickCommand(joyCommand);
  if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to send joystick command: " << status.message << std::endl;
  }
}
```

(下页继续)

(续上页)

```
// Position control standing
void recoveryStand() {
  try {
    auto& highController = robot->GetHighLevelMotionController();

    // Set gait to position control standing
    auto status = highController.SetGait(GaitMode::GAIT_STAND_R);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set position control standing: " << status.message << std::endl;
    } else {
      std::cout << "Robot set to position control standing" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Error executing position control standing: " << e.what() << std::endl;
  }
}

// Get single character input (no echo)
char getch() {
  char ch;
  struct termios oldt, newt;

  tcgetattr(STDIN_FILENO, &oldt);
  newt = oldt;
  newt.c_lflag &= ~(ICANON | ECHO);
  tcsetattr(STDIN_FILENO, TCSANOW, &newt);

  ch = getchar();

  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);

  std::cout << "Received character: " << ch << std::endl;
  return ch;
}

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

  // Create robot instance
  robot = std::make_unique<MagicRobot>();

  printHelp();
```

(下页继续)

(续上页)

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

try {
  // Configure local IP address for direct network connection and initialize SDK
  std::string localIp = "192.168.55.10";
  if (!robot->Initialize(localIp)) {
    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;

  // Set motion control level to high level
  status = robot->SetMotionControlLevel(ControllerLevel::HighLevel);
  if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to set motion control level: " << status.message << std::endl;
    robot->Shutdown();
    return -1;
  }

  // Initialize SLAM navigation controller
  auto& slamNavController = robot->GetSlamNavController();
  if (!slamNavController.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.load()) {
```

(下页继续)

(续上页)

```
    try {
      std::cout << "Enter command: ";
      char key = getch();
      if (key == 27) {  // ESC key
        break;
      }
      // 1. Preparation Functions
      // 1.1 Recovery stand
      if (key == '1') {
        recoveryStand();
      }
      // 2. SLAM Functions
      // 2.1 Start mapping
      else if (key == '2') {
        startMapping();
      }
      // 2.2 Cancel mapping
      else if (key == '3') {
        cancelMapping();
      }
      // 2.3 Save map
      else if (key == '4') {
        saveMap();
      }
      // 2.4 Load map
      else if (key == '5') {
        std::string strInput;
        std::cout << "Enter parameters: ";
        std::getline(std::cin, strInput);
        if (strInput.empty()) {
          continue;
        }
        // Split input parameters by space
        std::vector<std::string> parts;
        std::stringstream ss(strInput);
        std::string segment;
        while (std::getline(ss, segment, ' ')) {
          parts.push_back(segment);
        }
        // Parse parameters
        std::string mapToLoad = parts.empty() ? "" : parts[0];
        loadMap(mapToLoad);
      }
```

(下页继续)

(续上页)

```
        // 2.5 Delete map
        else if (key == '6') {
          std::string strInput;
          std::cout << "Enter parameters: ";
          std::getline(std::cin, strInput);
          if (strInput.empty()) {
            continue;
          }
          // Split input parameters by space
          std::vector<std::string> parts;
          std::stringstream ss(strInput);
          std::string segment;
          while (std::getline(ss, segment, ' ')) {
            parts.push_back(segment);
          }
          // Parse parameters
          std::string mapToDelete = parts.empty() ? "" : parts[0];
          deleteMap(mapToDelete);
        }
        // 2.6 Get all map information
        else if (key == '7') {
          getAllMapInfo();
        }
        // 4. Joystick Functions
        // 4.1 Move forward
        else if (key == 'w' || key == 'W') {
          sendJoystickCommand(0.0, 1.0, 0.0, 0.0);  // Move forward
        }
        // 4.2 Move backward
        else if (key == 's' || key == 'S') {
          sendJoystickCommand(0.0, -1.0, 0.0, 0.0);  // Move backward
        }
        // 4.3 Move left
        else if (key == 'a' || key == 'A') {
          sendJoystickCommand(-1.0, 0.0, 0.0, 0.0);  // Move left
        }
        // 4.4 Move right
        else if (key == 'd' || key == 'D') {
          sendJoystickCommand(1.0, 0.0, 0.0, 0.0);  // Move right
        }
        // 4.5 Turn left
        else if (key == 't' || key == 'T') {
          sendJoystickCommand(0.0, 0.0, -1.0, 0.0);  // Turn left
```

(下页继续)

(续上页)

```
        }
        // 4.6 Turn right
        else if (key == 'g' || key == 'G') {
          sendJoystickCommand(0.0, 0.0, 1.0, 0.0);  // Turn right
        }
        // 4.7 Stop movement
        else if (key == 'x' || key == 'X') {
          sendJoystickCommand(0.0, 0.0, 0.0, 0.0);  // Stop movement
        }
        // 5. Close Functions
        // 5.1 Close SLAM
        else if (key == 'p' || key == 'P') {
          closeSlam();
        }
        // Help
        else if (key == '?' || key == 'h' || key == 'H') {
          printHelp();
        } else {
          std::cout << "Unknown key: " << key << std::endl;
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(10));  // Brief delay

      } catch (const std::exception& e) {
        std::cerr << "Exception occurred while processing user input: " << e.what() <<␣
↪std::endl;
      }
    }
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred during program execution: " << e.what() << std::endl;
    return -1;
  }

  // Clean up resources
  try {
    std::cout << "Clean up resources" << std::endl;

    // Close SLAM navigation controller
    auto& slamNavController = robot->GetSlamNavController();
    slamNavController.Shutdown();
    std::cout << "SLAM navigation controller closed" << std::endl;

    // Disconnect
```

(下页继续)

(续上页)

```
    robot->Disconnect();
    std::cout << "Robot connection disconnected" << std::endl;

    // Shutdown robot
    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;

  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while cleaning up resources: " << e.what() << std::endl;
  }

  return 0;
}
```

26.1.2 导航示例

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

#include <signal.h>
#include <algorithm>
#include <atomic>
#include <chrono>
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <sstream>
#include <string>
#include <thread>
#include <vector>

using namespace magic::dog;
using namespace std;

// Global variables
std::unique_ptr<MagicRobot> robot = nullptr;
std::atomic<bool> running(true);
std::string current_slam_mode = "IDLE";
NavMode current_nav_mode = NavMode::IDLE;
int odometry_counter = 0;

void signalHandler(int signum) {
```

(下页继续)

(续上页)

```
  std::cout << "Interrupt signal (" << signum << ") received.\n";
  running = false;
  if (robot) {
    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;
  }
  exit(-1);
}

void printHelp() {
  std::cout << "SLAM and Navigation Function Demo Program" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "preparation Functions:" << std::endl;
  std::cout << "  1        Function 1: Recovery stand" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Localization Functions:" << std::endl;
  std::cout << "  2        Function 2: Switch to localization mode" << std::endl;
  std::cout << "  4        Function 4: Initialize pose" << std::endl;
  std::cout << "  5        Function 5: Get current pose information" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Navigation Functions:" << std::endl;
  std::cout << "  3        Function 3: Switch to navigation mode" << std::endl;
  std::cout << "  6        Function 6: Set navigation target goal" << std::endl;
  std::cout << "  7        Function 7: Pause navigation" << std::endl;
  std::cout << "  8        Function 8: Resume navigation" << std::endl;
  std::cout << "  9        Function 9: Cancel navigation" << std::endl;
  std::cout << "  0        Function 0: Get navigation status" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Odometry Functions:" << std::endl;
  std::cout << "  C        Function C: Subscribe odometry stream" << std::endl;
  std::cout << "  V        Function V: Unsubscribe odometry stream" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "Close Functions:" << std::endl;
  std::cout << "  L        Function L: Close navigation" << std::endl;
  std::cout << "  P        Function P: Close SLAM" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "  ?        Function ?: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

// ==================== Navigation Functions ====================

// Position control standing
```

(下页继续)

(续上页)

```
void recoveryStand() {
  try {
    auto& highController = robot->GetHighLevelMotionController();

    // Set gait to position control standing
    auto status = highController.SetGait(GaitMode::GAIT_STAND_R);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set position control standing: " << status.message << std::endl;
    } else {
      std::cout << "Robot set to position control standing" << std::endl;
    }
  } catch (const std::exception& e) {
    std::cerr << "Error executing position control standing: " << e.what() << std::endl;
  }
}

void switchToLocalizationMode() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Switch to localization mode
    auto status = controller.SwitchToLocation();
    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 = "LOCALIZATION";
    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;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while switching to localization mode: " << e.what() <<␣
↪std::endl;
  }
}

void initializePose(double x, double y, double yaw) {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();
```

(下页继续)

(续上页)

```
    // Create initial pose (set to origin)
    Pose3DEuler initialPose;
    initialPose.position = {x, y, 0.0};          // x, y, z
    initialPose.orientation = {0.0, 0.0, yaw};  // roll, pitch, yaw

    std::cout << "Initializing robot pose to origin..." << std::endl;

    // Initialize pose
    auto status = controller.InitPose(initialPose);
    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 origin (" << x << ", " << y << ", " << yaw << ")" <
↪< std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while initializing pose: " << e.what() << std::endl;
  }
}

void getCurrentLocalizationInfo() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Get current pose information
    LocalizationInfo poseInfo;
    auto status = controller.GetCurrentLocalizationInfo(poseInfo);
    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: " << (poseInfo.is_localization ? "Localized" : "Not␣
↪localized") << std::endl;
    std::cout << "Position: [" << poseInfo.pose.position[0] << ", "
              << poseInfo.pose.position[1] << ", " << poseInfo.pose.position[2] << "]" <<␣
```

(下页继续)

(续上页)

```
↪std::endl;
    std::cout << "Orientation: [" << poseInfo.pose.orientation[0] << ", "
              << poseInfo.pose.orientation[1] << ", " << poseInfo.pose.orientation[2] << "]" <<␣
↪std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while getting current pose information: " << e.what() <<␣
↪std::endl;
  }
}

void switchToNavigationMode() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Switch to navigation mode
    auto status = controller.ActivateNavMode(NavMode::GRID_MAP);
    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;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while switching to navigation mode: " << e.what() <<␣
↪std::endl;
  }
}

void setNavigationTarget(double x, double y, double yaw) {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();
    auto& highController = robot->GetHighLevelMotionController();

    // Disable joy stick
    highController.DisableJoyStick();
    std::cout << "Successfully disabled joy stick" << std::endl;

    // change gait to slow
    // To use Navigation, you must switch to the down stairs gait
```

(下页继续)

(续上页)

```
    auto status = highController.SetGait(GaitMode::GAIT_DOWN_CLIMB_STAIRS);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set gait to slow: " << status.message << std::endl;
      return;
    }
    std::cout << "Successfully set gait to slow" << std::endl;

    // Create target goal
    NavTarget targetGoal;
    targetGoal.id = 1;
    targetGoal.frame_id = "map";

    targetGoal.goal.position = {x, y, 0.0};
    targetGoal.goal.orientation = {0.0, 0.0, yaw};

    // Set target goal
    status = controller.SetNavTarget(targetGoal);
    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=("
              << targetGoal.goal.position[0] << ", " << targetGoal.goal.position[1] << ", "
              << targetGoal.goal.position[2] << "), orientation=("
              << targetGoal.goal.orientation[0] << ", " << targetGoal.goal.orientation[1] << ",
↪"
              << targetGoal.goal.orientation[2] << ")" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while setting navigation target: " << e.what() <<␣
↪std::endl;
  }
}

void pauseNavigation() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Pause navigation
    auto status = controller.PauseNavTask();
    if (status.code != ErrorCode::OK) {
```

(下页继续)

(续上页)

```
      std::cerr << "Failed to pause navigation, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    std::cout << "Successfully paused navigation" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while pausing navigation: " << e.what() << std::endl;
  }
}

void resumeNavigation() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Resume navigation
    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;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while resuming navigation: " << e.what() << std::endl;
  }
}

void cancelNavigation() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Cancel navigation
    auto status = controller.CancelNavTask();
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to cancel navigation, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }
```

(下页继续)

(续上页)

```
    std::cout << "Successfully cancelled navigation" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while cancelling navigation: " << e.what() << std::endl;
  }
}

void getNavigationStatus() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Get navigation status
    NavStatus navStatus;
    auto status = controller.GetNavTaskStatus(navStatus);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to get navigation status, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }

    // Display navigation status information
    std::cout << "=== Navigation Status ===" << std::endl;
    std::cout << "Target ID: " << navStatus.id << std::endl;
    std::cout << "Status: " << static_cast<int>(navStatus.status) << std::endl;
    std::cout << "Message: " << navStatus.message << std::endl;

    // Provide status interpretation
    std::string statusMeaning;
    switch (navStatus.status) {
      case NavStatusType::NONE:
        statusMeaning = "No navigation target set";
        break;
      case NavStatusType::RUNNING:
        statusMeaning = "Navigation is running";
        break;
      case NavStatusType::END_SUCCESS:
        statusMeaning = "Navigation completed successfully";
        break;
      case NavStatusType::END_FAILED:
        statusMeaning = "Navigation failed";
        break;
      case NavStatusType::PAUSE:
        statusMeaning = "Navigation is paused";
```

(下页继续)

(续上页)

```
        break;
      default:
        statusMeaning = "Unknown status value";
        break;
    }

    std::cout << "Status meaning: " << statusMeaning << std::endl;
    std::cout << "========================" << std::endl;

  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while getting navigation status: " << e.what() <<␣
↪std::endl;
  }
}

void closeNavigation() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Switch to idle mode to close navigation
    auto status = controller.ActivateNavMode(NavMode::IDLE);
    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;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while closing navigation: " << e.what() << std::endl;
  }
}

void openOdometryStream() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Open odometry stream
    auto status = robot->OpenChannelSwitch();
    if (status.code != ErrorCode::OK) {
```

(下页继续)

(续上页)

```
      std::cerr << "Failed to open odometry stream, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }
    std::cout << "Successfully opened odometry stream" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while opening odometry stream: " << e.what() << std::endl;
  }
}

void closeOdometryStream() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Close odometry stream
    auto status = robot->CloseChannelSwitch();
    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;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while closing odometry stream: " << e.what() << std::endl;
  }
}

void subscribeOdometryStream() {
  try {
    // Open channel switch
    robot->OpenChannelSwitch();

    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Define odometry callback
    auto odometryCallback = [](const std::shared_ptr<Odometry> data) {
      if (odometry_counter % 10 == 0) {
        std::cout << "Odometry position data: " << data->position[0] << ", "
                  << data->position[1] << ", " << data->position[2] << std::endl;
        std::cout << "Odometry orientation data: " << data->orientation[0] << ", "
                  << data->orientation[1] << ", " << data->orientation[2] << ", "
```

(下页继续)

(续上页)

```
                << data->orientation[3] << std::endl;
        std::cout << "Odometry linear velocity data: " << data->linear_velocity[0] << ", "
                << data->linear_velocity[1] << ", " << data->linear_velocity[2] << std::endl;
        std::cout << "Odometry angular velocity data: " << data->angular_velocity[0] << ", "
                << data->angular_velocity[1] << ", " << data->angular_velocity[2] <<␣
↪std::endl;
      }
      odometry_counter++;
    };

    // Subscribe odometry stream
    controller.SubscribeOdometry(odometryCallback);
    std::cout << "Successfully subscribed odometry stream" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while subscribing odometry stream: " << e.what() <<␣
↪std::endl;
  }
}

void unsubscribeOdometryStream() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();
    robot->CloseChannelSwitch();  // Close channel switch
    std::cout << "Successfully unsubscribed odometry stream" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while unsubscribing odometry stream: " << e.what() <<␣
↪std::endl;
  }
}

void closeSlam() {
  try {
    // Get SLAM navigation controller
    auto& controller = robot->GetSlamNavController();

    // Switch to idle mode to close SLAM
    auto status = controller.SwitchToIdle();
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to close SLAM, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }
```

(下页继续)

(续上页)

```
    current_slam_mode = "IDLE";
    std::cout << "Successfully closed SLAM system" << std::endl;
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while closing SLAM: " << e.what() << std::endl;
  }
}

// ==================== Utility Functions ====================

std::string getUserInput() {
  std::string input;
  std::cout << "Enter command: ";
  std::getline(std::cin, input);
  return input;
}

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

  // Create robot instance
  robot = std::make_unique<MagicRobot>();

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

  try {
    // Configure local IP address for direct network connection and initialize SDK
    std::string localIp = "192.168.55.10";
    if (!robot->Initialize(localIp)) {
      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;
```

(下页继续)

(续上页)

```
    }

    std::cout << "Successfully connected to robot" << std::endl;

    // Set motion control level to high level
    status = robot->SetMotionControlLevel(ControllerLevel::HighLevel);
    if (status.code != ErrorCode::OK) {
      std::cerr << "Failed to set motion control level: " << status.message << std::endl;
      robot->Shutdown();
      return -1;
    }

    // Initialize SLAM navigation controller
    auto& slamNavController = robot->GetSlamNavController();
    if (!slamNavController.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.load()) {
      try {
        std::string strInput = getUserInput();

        // Split input parameters by space
        std::vector<std::string> parts;
        std::stringstream ss(strInput);
        std::string segment;
        while (std::getline(ss, segment, ' ')) {
          parts.push_back(segment);
        }

        if (parts.empty()) {
          std::this_thread::sleep_for(std::chrono::milliseconds(10));  // Brief delay
          continue;
        }

        // Parse parameters
        std::string key = parts[0];
```

(下页继续)

(续上页)

```
        std::vector<std::string> args(parts.begin() + 1, parts.end());

        if (key == "\x1b") {  // ESC key
          break;
        }

        // 1. Preparation Functions
        if (key == "1") {
          // recovery stand
          recoveryStand();
        }
        // 2. Localization Functions
        // 2.1 Switch to localization mode
        else if (key == "2") {
          switchToLocalizationMode();
        }
        // 2.2 Initialize pose
        else if (key == "4") {
          double x = args.empty() ? 0.0 : std::stod(args[0]);
          double y = args.size() < 2 ? 0.0 : std::stod(args[1]);
          double yaw = args.size() < 3 ? 0.0 : std::stod(args[2]);
          std::cout << "input initial pose, x: " << x << ", y: " << y << ", yaw: " << yaw <<␣
↪std::endl;
          initializePose(x, y, yaw);
        }
        // 2.3 Get current localization information
        else if (key == "5") {
          getCurrentLocalizationInfo();
        }
        // 3. Navigation Functions
        // 3.1 Switch to navigation mode
        else if (key == "3") {
          switchToNavigationMode();
        }
        // 3.2 Set navigation target
        else if (key == "6") {
          double x = args.empty() ? 0.0 : std::stod(args[0]);
          double y = args.size() < 2 ? 0.0 : std::stod(args[1]);
          double yaw = args.size() < 3 ? 0.0 : std::stod(args[2]);
          std::cout << "input navigation target, x: " << x << ", y: " << y << ", yaw: " << yaw <
↪< std::endl;
          setNavigationTarget(x, y, yaw);
        }
```

(下页继续)

(续上页)

```
        // 3.3 Pause navigation
        else if (key == "7") {
          pauseNavigation();
        }
        // 3.4 Resume navigation
        else if (key == "8") {
          resumeNavigation();
        }
        // 3.5 Cancel navigation
        else if (key == "9") {
          cancelNavigation();
        }
        // 3.6 Get navigation status
        else if (key == "0") {
          getNavigationStatus();
        }
        // 4. Odometry Functions
        // 4.1 Subscribe odometry stream
        else if (key == "C" || key == "c") {
          subscribeOdometryStream();
        }
        // 4.2 Unsubscribe odometry stream
        else if (key == "V" || key == "v") {
          unsubscribeOdometryStream();
        }
        // 5. Close Functions
        // 5.1 Close navigation
        else if (key == "L" || key == "l") {
          closeNavigation();
        }
        // 5.2 Close SLAM
        else if (key == "P" || key == "p") {
          closeSlam();
        }
        // Help
        else if (key == "?") {
          printHelp();
        } else {
          std::cout << "Unknown key: " << key << std::endl;
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(10));  // Brief delay
```

(下页继续)

(续上页)

```
      } catch (const std::exception& e) {
        std::cerr << "Exception occurred while processing user input: " << e.what() <<␣
↪std::endl;
      }
    }
  } catch (const std::exception& e) {
    std::cerr << "Exception occurred during program execution: " << e.what() << std::endl;
    return -1;
  }

  // Clean up resources
  try {
    std::cout << "Clean up resources" << std::endl;

    // Close SLAM navigation controller
    auto& slamNavController = robot->GetSlamNavController();
    slamNavController.Shutdown();
    std::cout << "SLAM navigation controller closed" << std::endl;

    // Disconnect
    robot->Disconnect();
    std::cout << "Robot connection disconnected" << std::endl;

    // Shutdown robot
    robot->Shutdown();
    std::cout << "Robot shutdown" << std::endl;

  } catch (const std::exception& e) {
    std::cerr << "Exception occurred while cleaning up resources: " << e.what() << 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

import tty
import termios
import sys
import time
import signal
import threading
import logging
from typing import Optional
import numpy as np
import cv2
import random
import magicdog_python as magicdog

# 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[magicdog.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicdog.NavMode.IDLE
high_controller = None


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("  1        Function 1: Recovery stand")
    logging.info("")
    logging.info("SLAM Functions:")
    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("")
    logging.info("Joystick Functions:")
    logging.info("  W        Function W: forward")
    logging.info("  A        Function A: backward")
    logging.info("  S        Function S: left")
    logging.info("  D        Function D: right")
    logging.info("  T        Function T: turn left")
    logging.info("  G        Function G: turn right")
    logging.info("  X        Function X: stop")
    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 ====================


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()
```

(下页继续)

(续上页)

```
        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)


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 != magicdog.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 != magicdog.ErrorCode.OK:
            logging.error(
```

(下页继续)

(续上页)

```
                "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)


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)
        if status.code != magicdog.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)
```

(下页继续)

(续上页)

```
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()

        # 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 != magicdog.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 != magicdog.ErrorCode.OK:
            logging.error(
                "Failed to get map information, code: %s, message: %s",
                status.code,
                status.message,
            )
```

(下页继续)

(续上页)

```
            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):
                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.origin.orientation[0],
                    map_info.map_meta_data.origin.orientation[1],
                    map_info.map_meta_data.origin.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:
```

(下页继续)

(续上页)

```
        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
        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)
```

(下页继续)

(续上页)

```
        )

        # 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())}"

        # 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 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.switch_to_idle()
        if status.code != magicdog.ErrorCode.OK:
            logging.error(
                "Failed to close SLAM, code: %s, message: %s",
                status.code,
                status.message,
            )
```

(下页继续)

(续上页)

```
            return

        current_slam_mode = "IDLE"
        logging.info("Successfully closed SLAM system")
    except Exception as e:
        logging.error("Exception occurred while closing SLAM: %s", e)


# Switch gait to down climb stairs mode
def change_gait_to_down_climb_stairs():
    global high_controller
    try:
        current_gait = high_controller.get_gait()
        if current_gait != magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS:
            status = high_controller.set_gait(magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS)
            if status.code != magicdog.ErrorCode.OK:
                logging.error(f"Failed to set down climb stairs gait: {status.message}")
                return False

            # Wait for gait switch to complete
            while (
                high_controller.get_gait() != magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS
            ):
                time.sleep(0.01)
        logging.info("Gait changed to down climb stairs")
        return True
    except Exception as e:
        logging.error(f"Error changing gait: {e}")
        return False


def send_joystick_command(high_controller, left_x, left_y, right_x, right_y):
    if not change_gait_to_down_climb_stairs():
        return
    joy_command = magicdog.JoystickCommand()
    joy_command.left_x_axis = left_x
    joy_command.left_y_axis = left_y
    joy_command.right_x_axis = right_x
    joy_command.right_y_axis = right_y

    status = high_controller.send_joystick_command(joy_command)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to send joystick command: {status.message}")
```

(下页继续)

(续上页)

```
# Position control standing
def recovery_stand(high_controller):
    try:
        # Set gait to position control standing
        status = high_controller.set_gait(magicdog.GaitMode.GAIT_STAND_R)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set position control standing: {status.message}")
        else:
            logging.info("Robot set to position control standing")
    except Exception as e:
        logging.error(f"Error executing position control standing: {e}")


# 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


# ==================== Utility Functions ====================


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

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

    # Create robot instance
```

(下页继续)

(续上页)

```
    robot = magicdog.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 = "192.168.55.10"
        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 != magicdog.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")

        # Set motion control level to high level
        status = robot.set_motion_control_level(magicdog.ControllerLevel.HIGH_LEVEL)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set motion control level: {status.message}")
            robot.shutdown()
            return

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

        # 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
```

(下页继续)

(续上页)

```
    logging.info("Successfully initialized SLAM navigation controller")

    # Main loop
    while running:
        try:
            logging.info("Enter command: ")
            key = getch()
            if key == "\x1b":  # ESC key
                break

            # 1. Preparation Functions
            # 1.1 Recovery stand
            if key == "1":
                # recovery stand
                recovery_stand(high_controller)
            # 2. SLAM Functions
            # 2.1 Start mapping
            elif key == "2":
                start_mapping()
            # 2.2 Cancel mapping
            elif key == "3":
                cancel_mapping()
            # 2.3 Save map
            elif key == "4":
                save_map()
            # 2.4 Load map
            elif key == "5":
                str_input = input("Enter parameters: ").strip()
                if not str_input:
                    continue
                # Split input parameters by space
                parts = str_input.strip().split()
                # Parse parameters
                map_to_load = parts[0]
                load_map(map_to_load)
            # 2.5 Delete map
            elif key == "6":
                str_input = input("Enter parameters: ").strip()
                if not str_input:
                    continue
                # Split input parameters by space
                parts = str_input.strip().split()
```

(下页继续)

(续上页)

```
                # Parse parameters
                map_to_delete = parts[0]
                delete_map(map_to_delete)
            # 2.6 Get all map information
            elif key == "7":
                get_all_map_info()
            # 3. Joystick Functions
            # 3.1 Move forward
            elif key.lower() == "w":
                left_y = 1.0
                left_x = 0.0
                right_x = 0.0
                right_y = 0.0
                send_joystick_command(
                    high_controller, left_x, left_y, right_x, right_y
                )
            # 3.2 Move backward
            elif key.lower() == "s":
                left_y = -1.0
                left_x = 0.0
                right_x = 0.0
                right_y = 0.0
                send_joystick_command(
                    high_controller, left_x, left_y, right_x, right_y
                )
            # 3.3 Move left
            elif key.lower() == "a":
                left_x = -1.0
                left_y = 0.0
                right_x = 0.0
                right_y = 0.0
                send_joystick_command(
                    high_controller, left_x, left_y, right_x, right_y
                )
            # 3.4 Move right
            elif key.lower() == "d":
                left_x = 1.0
                left_y = 0.0
                right_x = 0.0
                right_y = 0.0
                send_joystick_command(
                    high_controller, left_x, left_y, right_x, right_y
                )
```

(下页继续)

(续上页)

```
                # 3.5 Turn left
                elif key.lower() == "t":
                    left_x = 0.0
                    left_y = 0.0
                    right_x = -1.0
                    right_y = 0.0
                    send_joystick_command(
                        high_controller, left_x, left_y, right_x, right_y
                    )
                # 3.6 Turn right
                elif key.lower() == "g":
                    left_x = 0.0
                    left_y = 0.0
                    right_x = 1.0
                    right_y = 0.0
                    send_joystick_command(
                        high_controller, left_x, left_y, right_x, right_y
                    )
                # 3.7 Stop movement
                elif key.lower() == "x":
                    left_x = 0.0
                    left_y = 0.0
                    right_x = 0.0
                    right_y = 0.0
                    send_joystick_command(
                        high_controller, left_x, left_y, right_x, right_y
                    )
                # 4. Close Functions
                # 4.1 Close SLAM
                elif key.upper() == "P":
                    close_slam()
                # Help
                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)
```

(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 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

import sys
import time
import signal
import threading
import logging
from typing import Optional
import numpy as np
import cv2
import random
```

(下页继续)

(续上页)

```
import magicdog_python as magicdog

# 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[magicdog.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicdog.NavMode.IDLE
high_controller = None
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"""
    logging.info("SLAM and Navigation Function Demo Program")
    logging.info("")
    logging.info("preparation Functions:")
    logging.info("  1        Function 1: Recovery stand")
    logging.info("")
    logging.info("Localization Functions:")
    logging.info("  2        Function 2: Switch to localization mode")
    logging.info("  4        Function 4: Initialize pose")
    logging.info("  5        Function 5: Get current pose information")
    logging.info("")
    logging.info("Navigation Functions:")
    logging.info("  3        Function 3: Switch to navigation mode")
```

(下页继续)

(续上页)

```
    logging.info("  6        Function 6: Set navigation target goal")
    logging.info("  7        Function 7: Pause navigation")
    logging.info("  8        Function 8: Resume navigation")
    logging.info("  9        Function 9: Cancel navigation")
    logging.info("  0        Function 0: Get navigation status")
    logging.info("")
    logging.info("Odometry Functions:")
    logging.info("  C        Function C: Subscribe odometry stream")
    logging.info("  V        Function V: Unsubscribe odometry stream")
    logging.info("")
    logging.info("Close Functions:")
    logging.info("  L        Function L: Close navigation")
    logging.info("  P        Function P: Close SLAM")
    logging.info("")
    logging.info("  ?        Function ?: Print help")
    logging.info("  ESC      Exit program")


# Recovery stand
def recovery_stand(high_controller):
    try:
        # Set gait to position control standing
        status = high_controller.set_gait(magicdog.GaitMode.GAIT_STAND_R)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set position control standing: {status.message}")
        else:
            logging.info("Robot set to position control standing")
    except Exception as e:
        logging.error(f"Error executing position control standing: {e}")


# switch to down climb stairs gait
def switch_to_down_climb_stairs_gait(high_controller):
    try:
        status = high_controller.set_gait(magicdog.GaitMode.GAIT_DOWN_CLIMB_STAIRS)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set down climb stairs gait: {status.message}")
        else:
            logging.info("Successfully set down climb stairs gait")
    except Exception as e:
        logging.error(f"Error setting down climb stairs gait: {e}")
```

(下页继续)

(续上页)

```
def switch_to_localization_mode():
    """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.switch_to_location()
        if status.code != magicdog.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"
        )
    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 = magicdog.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 != magicdog.ErrorCode.OK:
```

(下页继续)

(续上页)

```
            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
        status, localization_info = controller.get_current_localization_info()
        if status.code != magicdog.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 localization_info.is_localization else "Not localized",
        )
        logging.info(
            "Position: [%.3f, %.3f, %.3f]",
            localization_info.pose.position[0],
            localization_info.pose.position[1],
            localization_info.pose.position[2],
        )
        logging.info(
            "Orientation: [%.3f, %.3f, %.3f]",
```

(下页继续)

(续上页)

```
            localization_info.pose.orientation[0],
            localization_info.pose.orientation[1],
            localization_info.pose.orientation[2],
        )
    except Exception as e:
        logging.error(
            "Exception occurred while getting current pose information: %s", e
        )


def switch_to_navigation_mode():
    """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(magicdog.NavMode.GRID_MAP)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(
                "Failed to switch to navigation mode, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        current_nav_mode = magicdog.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, high_controller
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Disable joy stick
        high_controller.disable_joy_stick()
        logging.info("Successfully disabled joy stick")
```

(下页继续)

(续上页)

```
        # change gait to down climb stairs
        # To use Navigation, you must switch to the down stairs gait
        switch_to_down_climb_stairs_gait(high_controller)

        # Create target goal
        target_goal = magicdog.NavTarget()
        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]

        # Set target goal
        status = controller.set_nav_target(target_goal)
        if status.code != magicdog.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)


def pause_navigation():
    """Pause navigation"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()
```

(下页继续)

(续上页)

```
        # Pause navigation
        status = controller.pause_nav_task()
        if status.code != magicdog.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 != magicdog.ErrorCode.OK:
            logging.error(
                "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
```

(下页继续)

(续上页)

```
        controller = robot.get_slam_nav_controller()

        # Cancel navigation
        status = controller.cancel_nav_task()
        if status.code != magicdog.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 != magicdog.ErrorCode.OK:
            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("Message: %s", nav_status.message)

        # Provide status interpretation
        status_meaning = {
            magicdog.NavStatusType.NONE: "No navigation target set",
```

(下页继续)

(续上页)

```
            magicdog.NavStatusType.RUNNING: "Navigation is running",
            magicdog.NavStatusType.END_SUCCESS: "Navigation completed successfully",
            magicdog.NavStatusType.END_FAILED: "Navigation failed",
            magicdog.NavStatusType.PAUSE: "Navigation is paused",
        }

        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()

        # Switch to idle mode to close navigation
        status = controller.activate_nav_mode(magicdog.NavMode.IDLE)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(
                "Failed to close navigation, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        current_nav_mode = magicdog.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
```

(下页继续)

(续上页)

```
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Open odometry stream
        status = controller.open_odometry_stream()
        if status.code != magicdog.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()

        # Close odometry stream
        status = controller.close_odometry_stream()
        if status.code != magicdog.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:
```

(下页继续)

(续上页)

```
        # Open channel switch
        robot.open_channel_switch()
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        def odometry_callback(data):
            global odometry_counter
            if odometry_counter % 10 == 0:
                logging.info(
                    "Odometry position data: %f, %f, %f",
                    data.position[0],
                    data.position[1],
                    data.position[2],
                )
                logging.info(
                    "Odometry orientation data: %f, %f, %f, %f",
                    data.orientation[0],
                    data.orientation[1],
                    data.orientation[2],
                    data.orientation[3],
                )
                logging.info(
                    "Odometry linear velocity data: %f, %f, %f",
                    data.linear_velocity[0],
                    data.linear_velocity[1],
                    data.linear_velocity[2],
                )
                logging.info(
                    "Odometry angular velocity data: %f, %f, %f",
                    data.angular_velocity[0],
                    data.angular_velocity[1],
                    data.angular_velocity[2],
                )
            odometry_counter += 1

        # Subscribe odometry stream
        controller.subscribe_odometry(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_stream():
```

(下页继续)

(续上页)

```
    """Unsubscribe odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()
        robot.close_channel_switch()  # Close channel switch

        # 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 disable_joy_stick(high_controller):
    """Disable joy stick"""
    high_controller.disable_joy_stick()
    logging.info("Successfully disabled joy stick")


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.switch_to_idle()
        if status.code != magicdog.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)
```

(下页继续)

(续上页)

```
# ==================== Utility Functions ====================


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


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

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

    # Create robot instance
    robot = magicdog.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 = "192.168.55.10"
        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 != magicdog.ErrorCode.OK:
            logging.error(
                "Failed to connect to robot, code: %s, message: %s",
                status.code,
                status.message,
            )
            robot.shutdown()
```

(下页继续)

(续上页)

```
        return -1

    logging.info("Successfully connected to robot")

    # Set motion control level to high level
    status = robot.set_motion_control_level(magicdog.ControllerLevel.HIGH_LEVEL)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(f"Failed to set motion control level: {status.message}")
        robot.shutdown()
        return

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

    # 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
            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 Functions
```

(下页继续)

(续上页)

```
            if key.upper() == "1":
                # recovery stand
                recovery_stand(high_controller)
            # 2. Localization Functions
            # 2.1 Switch to localization mode
            elif key == "2":
                switch_to_localization_mode()
            # 2.2 Initialize pose
            elif key == "4":
                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 initial pose, x: %f, y: %f, yaw: %f", x, y, yaw)
                initialize_pose(x, y, yaw)
            # 2.3 Get current localization information
            elif key == "5":
                get_current_localization_info()
            # 3. Navigation Functions
            # 3.1 Switch to navigation mode
            elif key.upper() == "3":
                switch_to_navigation_mode()
            # 3.2 Set navigation target
            elif key.upper() == "6":
                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 navigation target, x: %f, y: %f, yaw: %f", x, y, yaw
                )
                set_navigation_target(x, y, yaw)
            # 3.3 Pause navigation
            elif key.upper() == "7":
                pause_navigation()
            # 3.4 Resume navigation
            elif key.upper() == "8":
                resume_navigation()
            # 3.5 Cancel navigation
            elif key.upper() == "9":
                cancel_navigation()
            # 3.6 Get navigation status
            elif key.upper() == "0":
                get_navigation_status()
            # 4. Odometry Functions
```

(下页继续)

(续上页)

```
            # 4.1 Subscribe odometry stream
            elif key.upper() == "C":
                subscribe_odometry_stream()
            # 4.2 Unsubscribe odometry stream
            elif key.upper() == "V":
                unsubscribe_odometry_stream()
            # 5. Close Functions
            # 5.1 Close navigation
            elif key.upper() == "L":
                close_navigation()
            # 5.2 Close SLAM
            elif key.upper() == "P":
                close_slam()
            # Help
            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
```

(下页继续)

(续上页)

```
            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

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$LD_LIBRARY_PATH
```

26.3.2 运行示例

```
# C++
./slam_example
./navigation_example
# Python
python3 slam_example.py
python3 navigation_example.py
```

26.3.3 控制说明

建图示例

- 准备工作:
  - 按键 1 - 恢复站立
- SLAM 功能:
  - 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

- 手柄控制功能：
  - 按键 W - 前进
  - 按键 A - 后退
  - 按键 S - 左移
  - 按键 D - 右移
  - 按键 T - 左转
  - 按键 G - 右转
  - 按键 X - 停止
- 关闭功能：
  - 按键 P - 关闭 SLAM
- 系统命令：
  - 按键 ? - 显示帮助信息

导航

- 准备工作：
  - 按键 1 - 恢复站立
- 定位功能：
  - 按键 2 - 切换到定位模式
  - 按键 4 - 初始化位姿
  - 按键 5 - 获取当前位置信息
- 导航功能：
  - 按键 3 - 切换到导航模式
  - 按键 6 - 设置导航目标点
  - 按键 7 - 暂停导航
  - 按键 8 - 恢复导航
  - 按键 9 - 取消导航
  - 按键 0 - 获取导航状态
- 里程计功能：
  - 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 `L` - 关闭导航
  - 按键 `P` - 关闭 SLAM
- 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并保存当前状态

26.3.5 注意事项

- 建图示例中，至少需要切换到 Recovery_stand 才可以进行建图，可以 Recovery_stand 模式下推着机器人进行建图，也可以 Balance_stand 模式下遥控控制机器人运动进行建图；
- 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 (`SwitchToLocation()`) 和导航模式 (`ActivateNavMode(NavMode::GRID_MAP)`)；
- 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 (`SwitchToIdle()`)，导航也切换到 IDLE 模式 (`ActivateNavMode(NavMode::IDLE)`)

CHAPTER 27

显示示例

该示例展示了如何使用 MagicDog SDK 进行初始化、连接机器人、显示控制等基本操作。

27.1 C++

示例文件：`display_example.cpp`

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

#include <termios.h>
#include <unistd.h>
#include <chrono>
#include <csignal>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <thread>

using namespace magic::dog;

// Global robot instance
std::unique_ptr<MagicRobot> robot = nullptr;
```

(下页继续)

(续上页)

```
void signalHandler(int signum) {
  std::cout << "\nInterrupt signal (" << signum << ") received." << std::endl;
  if (robot) {
    robot->Shutdown();
  }
  exit(signum);
}

void print_help() {
  std::cout << "Key Function Demo Program\n"
            << std::endl;
  std::cout << "Key Function Description:" << std::endl;
  std::cout << "Face expression Functions:" << std::endl;
  std::cout << "  1        Function 1: Get all face expressions" << std::endl;
  std::cout << "  2        Function 2: Set face expression" << std::endl;
  std::cout << "  3        Function 3: Get current face expression" << std::endl;
  std::cout << "" << std::endl;
  std::cout << "  ?        Function ?: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

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 get_all_face_expressions() {
  auto& display_controller = robot->GetDisplayController();

  std::vector<FaceExpression> face_expressions;
  auto status = display_controller.GetAllFaceExpressions(face_expressions);
  if (status.code != ErrorCode::OK) {
    std::cerr << "get all face expressions failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }
```

(下页继续)

(续上页)

```
  for (auto& face_expression : face_expressions) {
    std::cout << "Face Expression ID: " << face_expression.id << std::endl;
    std::cout << "Face Expression Name: " << face_expression.name << std::endl;
    std::cout << "Face Expression Description: " << face_expression.description << std::endl;
    std::cout << std::endl;
  }
}

void get_current_face_expression() {
  auto& display_controller = robot->GetDisplayController();

  FaceExpression face_expression;
  auto status = display_controller.GetFaceExpression(face_expression);
  if (status.code != ErrorCode::OK) {
    std::cerr << "get current face expression failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }

  std::cout << "Face Expression ID: " << face_expression.id << std::endl;
  std::cout << "Face Expression Name: " << face_expression.name << std::endl;
  std::cout << "Face Expression Description: " << face_expression.description << std::endl;
}

void set_face_expression(int face_expression_id) {
  auto& display_controller = robot->GetDisplayController();

  auto status = display_controller.SetFaceExpression(face_expression_id);
  if (status.code != ErrorCode::OK) {
    std::cerr << "set face expression failed, code: " << status.code
              << ", message: " << status.message << std::endl;
    return;
  }
  std::cout << "set face expression success" << std::endl;
}

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

  print_help();
```

(下页继续)

(续上页)

```
std::string local_ip = "192.168.55.10";
robot = std::make_unique<MagicRobot>();

// 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;
}

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

// Wait for user input
while (true) {
  int key = getch();
  if (key == 27) {  // ESC key
    break;
  }

  std::cout << "Key ASCII: " << key << ", Character: " << static_cast<char>(key) << std::endl;

  switch (key) {
    case '1':
      get_all_face_expressions();
      break;
    case '2':{
      int face_expression_id = 16;
      std::cout << "Please input face expression id: ";
      std::cin >> face_expression_id;
      set_face_expression(face_expression_id);
      std::cin.ignore();
    }
      break;
    case '3':
```

(下页继续)

(续上页)

```
        get_current_face_expression();
        break;
      case '?':
        print_help();
        break;
      default:
        std::cout << "Unknown key: " << key << std::endl;
        break;
    }

    // Sleep 10ms, equivalent to usleep(10000)
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
  }

  // 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;
  }

  std::cout << "disconnect robot success" << std::endl;

  robot->Shutdown();
  std::cout << "robot shutdown" << std::endl;

  return 0;
}
```

27.2 Python

示例文件：`display_example.py`

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

import sys
import time
import signal
import termios
import tty
```

(下页继续)

(续上页)

```
import logging
from typing import Optional
import magicdog_python as magicdog

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

# Global robot instance
robot = None


def signal_handler(signum, frame):
    """Handle Ctrl+C signal"""
    logging.info(f"\nInterrupt signal ({signum}) received.")
    if robot:
        robot.shutdown()
    sys.exit(signum)


def print_help():
    """Print help information"""
    logging.info("Key Function Description:")
    logging.info("")
    logging.info("Face expression Functions:")
    logging.info("  1        Function 1: Get all face expressions")
    logging.info("  2        Function 2: Set face expression")
    logging.info("  3        Function 3: Get current face expression")
    logging.info("")
    logging.info("  ?        Function ?: Print help")
    logging.info("  ESC      Exit program")


def getch():
    """Get a single character from standard input"""
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(fd)
        ch = sys.stdin.read(1)
```

(下页继续)

(续上页)

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


def get_all_face_expressions(display_controller):
    """Get all face expressions"""
    status, all_face_expressions = display_controller.get_all_face_expressions()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            "Failed to get all face expressions, code: %s, message: %s",
            status.code,
            status.message,
        )
        return
    logging.info("Total face expressions: %d", len(all_face_expressions))

    for i, face_expression in enumerate(all_face_expressions):
        logging.info("Face expression %d: %d", i + 1, face_expression.id)
        logging.info("  Name: %s", face_expression.name)
        logging.info("  Description: %s", face_expression.description)


def set_face_expression(display_controller, face_expression_id):
    """Set face expression"""
    status = display_controller.set_face_expression(face_expression_id)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            "Failed to set face expression, code: %s, message: %s",
            status.code,
            status.message,
        )
        return


def get_current_face_expression(display_controller):
    """Get all face expressions"""
    status, face_expression = display_controller.get_current_face_expression()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            "Failed to get current face expression, code: %s, message: %s",
            status.code,
            status.message,
```

(下页继续)

(续上页)

```
        )
        return

    logging.info("Face expression: %d", face_expression.id)
    logging.info("  Name: %s", face_expression.name)
    logging.info("  Description: %s", face_expression.description)


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

    # Bind SIGINT (Ctrl+C)
    signal.signal(signal.SIGINT, signal_handler)

    print_help()

    local_ip = "192.168.55.10"
    robot = magicdog.MagicRobot()

    # Configure local IP address for direct network connection and initialize SDK
    if not robot.initialize(local_ip):
        logging.error("robot sdk initialize failed.")
        robot.shutdown()
        return -1

    # Connect to robot
    status = robot.connect()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            f"connect robot failed, code: {status.code}, message: {status.message}"
        )
        robot.shutdown()
        return -1

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

    display_controller = robot.get_display_controller()

    # Wait for user input
    while True:
        key = getch()
        if key == "\x1b":  # ESC key
```

(下页继续)

(续上页)

```
            break

        key_ascii = ord(key)
        logging.info(f"Key ASCII: {key_ascii}, Character: {key}")

        # 1. Face expression Functions
        # 1.1 Get all face expressions
        if key == "1":
            get_all_face_expressions(display_controller)
        # 1.2 Set face expression
        elif key == "2":
            face_expression_id = 16
            try:
                user_input = input("Please input face expression id: ")
                face_expression_id = int(user_input)
            except ValueError:
                logging.info("invalid input, using default value 16.")
            set_face_expression(display_controller, face_expression_id)
        # 1.3 Get current face expression
        elif key == "3":
            get_current_face_expression(display_controller)
        # Help
        elif key == "?":
            print_help()
        else:
            logging.info(f"Unknown key: {key_ascii}")

        # Sleep 10ms, equivalent to usleep(10000)
        time.sleep(0.01)

    display_controller.shutdown()
    logging.info("display controller shutdown")

    # Disconnect from robot
    robot.disconnect()
    logging.info("disconnect robot success")

    robot.shutdown()
    logging.info("robot shutdown")

    return 0
```

(下页继续)

(续上页)

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

27.3 运行说明

27.3.1 环境准备

```
export PYTHONPATH=/opt/magic_robotics/magicdog_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicdog_sdk/lib:$LD_LIBRARY_PATH
```

27.3.2 运行示例

```
# C++
./display_example
# Python
python3 display_example.py
```

27.3.3 控制说明:

- 面部表情功能:
  - 按键 `1` - 获取所有面部表情
  - 按键 `2` - 设置面部表情
  - 按键 `3` - 获取当前面部表情
- 其他功能:
  - 按键 `?` - 显示帮助菜单

27.3.4 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 28

ROS2 SDK 使用说明

28.1 SDK 介绍

MagicDog_sdk 是魔法原子为新一代机器人开发的 SDK。SDK 将底层电机控制、高层运动控制、雷达点云数据、音视频图传、SLAM、里程计等接口进行封装，并提供相关的函数接口。您可以按照我们提供的接口和例程，完成机器狗的二次开发。

注：ROS2 SDK 后续不再更新，如果想要进行二开的话，建议使用 C++ API/Python API 开发

28.2 SDK 支持情况

机器人型号	支持状态	说明
MagicDog EDU 版	✓ 支持	可使用该 SDK 进行二次开发
其他型号或版本	✗ 不支持	不支持使用该 SDK 进行二次开发

注意： ROS2 SDK 后续不再更新，如果想要进行二开的话，建议使用 C++ API/Python API 开发

28.3 SDK 下载地址

类型	地址
GitHub	https://github.com/MagiclabRobotics/magicdog_ros2_sdk

28.4 环境准备

28.4.1 安装 ROS 2

ROS 2 版本	Ubuntu 版本	推荐状态
ROS 2 Foxy	Ubuntu 20.04	✅ 推荐
ROS 2 Iron	Ubuntu 22.04	✅ 支持

28.4.2 安装 cyclone-dds

```
sudo apt install ros-foxy-rmw-cyclonedx-cpp
```

28.5 编译步骤

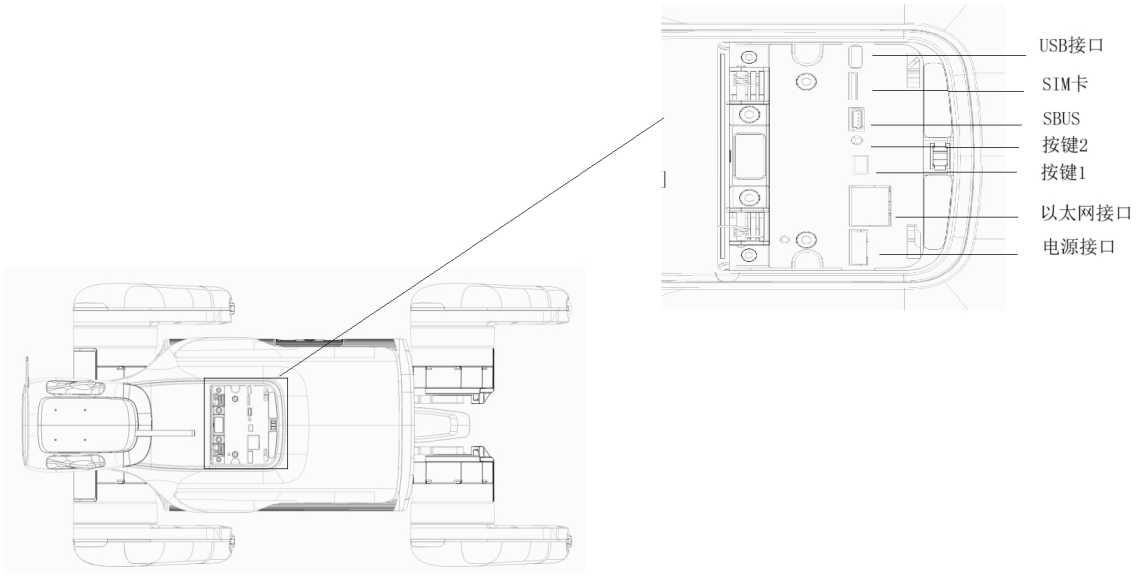
步骤	命令
1. 创建工作空间	`mkdir -p ~/ros2_ws/src && cd ~/ros2_ws/src`
2. 克隆代码	`git clone https://github.com/MagiclabRobotics/magicdog_ros2_sdk.git`
3. 解决依赖	`cd ~/ros2_ws && rosdep install -i --from-path src --rosdistro foxy -y`
4. 编译	`colcon build`
5. 环境配置	`source install/setup.bash`

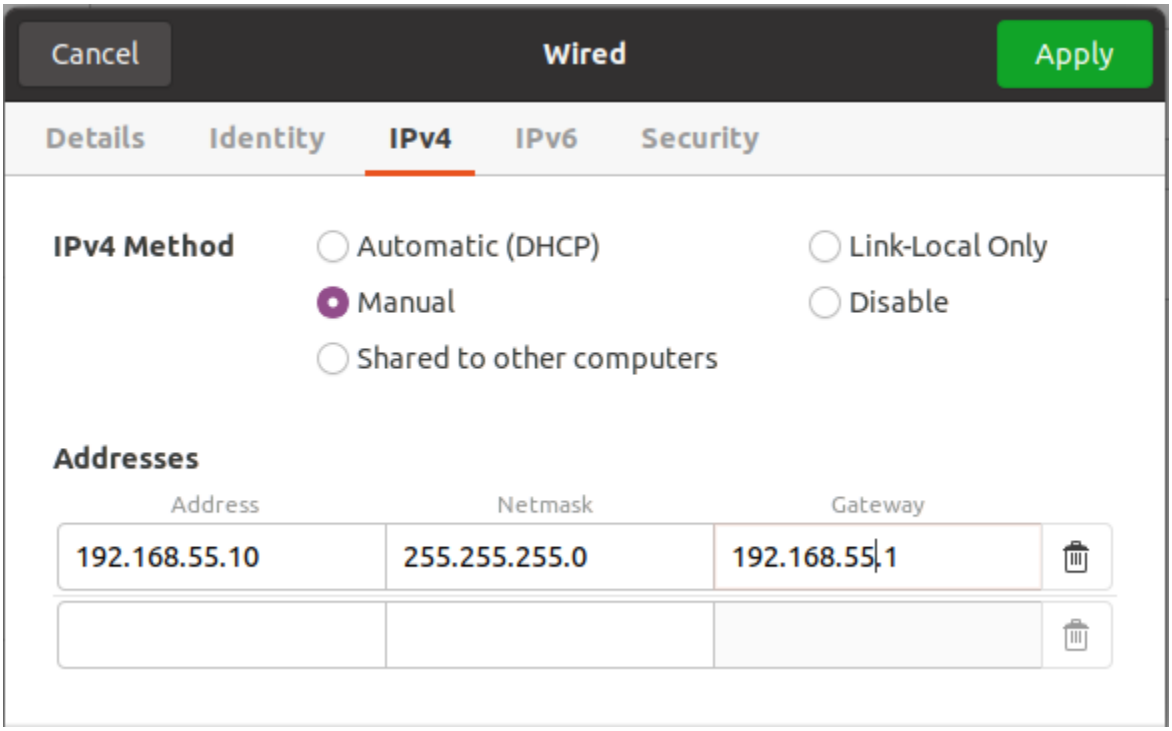
28.6 连接配置

28.6.1 个人电脑连接机器人

步骤	操作说明
1. 关闭服务	关闭个人电脑上的 wifi 和 dockers
2. 检查 Docker	‘ifconfig
3. 关闭 Docker	`sudo ifconfig docker0 down`
4. 物理连接	个人电脑和机器人使用数据线连接

28.6.2 连接端口说明


USB接口
SIM卡
SBUS
按键2
按键1
以太网接口
电源接口


Cancel
Wired
Apply
Details
Identity
IPv4
IPv6
Security
IPv4 Method
Automatic (DHCP)
Link-Local Only
Manual
Disable
Shared to other computers
Addresses
Address
Netmask
Gateway
192.168.55.10
255.255.255.0
192.168.55.1

28.7 环境配置

配置类型	命令
临时配置	`export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp`
临时配置	`export ROS_DOMAIN_ID=25`
永久配置	`echo "export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp" >> ~/.bashrc`
永久配置	`echo "export ROS_DOMAIN_ID=25" >> ~/.bashrc`

28.8 验证连接

使用 ROS 2 指令验证多机通讯是否配置成功：

```
ros2 node list
```

注意：记得 source 不同工作空间中的 setup 文件

CHAPTER 29

机器狗 ROS2 SDK 文档

提供机器人系统 ROS2 接口服务，通过 ROS2 话题和服务方式实现对机器人的传感器数据获取、语音控制、相机管理、导航控制等功能。

注：ROS2 SDK 后续不再更新，如果想要进行二开的话，建议使用 C++ API/Python API 开发

29.1 文档概述

本手册面向机器人集成商和工程师，系统性地介绍机器狗 ROS2 SDK 的功能与接口，方便用户在 Ubuntu+ROS2 平台上进行二次开发和集成。

项目	说明
适用对象	机器人集成商、行业开发者、科研人员
主要内容	系统结构、通讯协议、开发环境配置、接口说明、实用案例、常见问题
覆盖场景	工厂巡检、教育科研、定制应用等

29.2 系统架构

机器狗系统由底层硬件（主控、执行器、传感器）、嵌入式软件（固件）、中间件（ROS2 + DDS）、以及上层应用共同组成。

组件	功能说明
主控板	运算与调度核心，运行 ROS2
传感器	包含 TOF、超声、IMU、鱼眼相机、激光雷达等
执行器	用于运动与交互（如电机、舵机、显示等）
上位机/外部应用	通过有线或 Wi-Fi 与主控进行交互、远程调试与监控

29.3 通信协议

SDK 所有通信基于 ROS2 的分布式架构，底层采用 DDS(Data Distribution Service) 中间件，实现高效可靠的消息发布/订阅与服务调用。

通信方式	应用场景
Topic	适用于传感器数据、状态信息等” 发布-订阅” 模式的数据流转
Service	用于参数配置、控制指令、触发操作等一对一请求-响应事务
Action	适合需要过程反馈和结果通知的复杂任务（如导航、动作执行）
QoS	可根据实际需求灵活设置，保障消息的实时性与可靠性

29.4 开发环境说明

环境要求	说明
操作系统	Ubuntu 20.04 LTS（长期支持版）
ROS2 版本	Foxy/Galactic/Humble 等均兼容，建议使用 Foxy 及以上
基础依赖	Python3、C++ 编译环境、colcon 等构建工具
网络配置	推荐有线直连，便于多播和大数据流

29.5 开发环境配置

29.5.1 SDK 安装与节点配置

步骤	操作说明
1. 上传 SDK	将 SDK 源码/安装包上传至主控或开发 PC
2. 编译	进入工作空间，使用 `colcon build` 编译
3. 环境配置	source 工作空间环境脚本，如 `source install/setup.bash`

29.5.2 ROS2 环境变量设置

```
export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp
export ROS_DOMAIN_ID=25  # 如有多机器人并网，须区分domain_id
```

29.6 ROS2 接口列表与示例

29.6.1 5.1 传感器接口

5.1.1 TOF 输出

飞行时间 (TOF) 传感器阵列，实时输出前方和腹部距离，常用于避障、地图构建等。

话题	消息类型	index/顺序	位置说明
/tof	std_msgs/Float32MultiArray	0	左前
/tof	std_msgs/Float32MultiArray	1	右前
/tof	std_msgs/Float32MultiArray	2	腹部

5.1.2 超声波输出

覆盖侧向与正前方低速避障场景。

话题	消息类型	index/顺序	位置说明
/ultra	std_msgs/Float32MultiArray	0	左侧
/ultra	std_msgs/Float32MultiArray	1	前方
/ultra	std_msgs/Float32MultiArray	2	右侧

5.1.3 头部触摸传感器

用于人机交互，可区分多种手势事件。

话题	消息类型	值	手势类型
/head_touch_state	std_msgs/Int8	0	未触发
/head_touch_state	std_msgs/Int8	1	单击
/head_touch_state	std_msgs/Int8	2	双击
/head_touch_state	std_msgs/Int8	3	上划
/head_touch_state	std_msgs/Int8	4	下划
/head_touch_state	std_msgs/Int8	5	长按
/head_touch_state	std_msgs/Int8	16	错误

5.1.4 电池信息

电池状态监控，包括电压、电流、温度、SOC 等，实时保障系统安全。

话题	消息类型	主要字段
/motion/bms_dat	motion_msg BMSRespons	batt_volt（电压）batt_curr（电流）batt_temp（温度）batt_soc（剩余电量）status（电源状态）key（关机信号）batt_health（健康度）batt_loop_number（循环次数）power_board_status（电源板故障位）

29.6.2 5.2 语音接口

本章节介绍如何通过 ROS2 服务方式调用语音相关能力，包括音量查询、音量设置、语音识别、TTS 控制等。

功能	服务调用
获取音量	`ros2 service call /voice/get_volume voice_msgs/srv/GetVolume "{}"`
设置音量	`ros2 service call /voice/set_volume voice_msgs/srv/SetVolume "{volume: 7}"`
获取语音识别状态	`ros2 service call /voice_config_get voice_msgs/srv/GetConfig "{}"`
设置语音识别	`ros2 service call /voice_config voice_msgs/srv/Config "{enable_voice: true}"`
停止 TTS	`ros2 service call /voice/stop_tts std_srvs/srv/Trigger "{}"`
播放 TTS	`ros2 service call /voice/set_tts voice_msgs/srv/SetTTS "{tts_id: 'id_1', content: '你好', priority: 1, mode: 0}"`

29.6.3 5.3 相机接口

本章节介绍各种相机（如主摄像头、鱼眼、双目等）相关的开启、关闭、参数设置、数据流接入方式等接口。

话题名	消息类型	说明
/sensor_msgs	sensor_msgs::msg::Image	主摄原始图像
/depth	sensor_msgs::msg::Image + /depth/camera_info	深度图像 + 相机内参
/color	sensor_msgs::msg::Image + /color/camera_info	彩色图像 + 相机内参

29.6.4 5.4 导航接口

导航相关接口涉及多种服务与 topic，包括模式切换、目标点设置、速度指令、状态反馈等。

功能	服务调用
切换导航模式	`ros2 service call /switch_navigate_mode pp_msgs/srv/ChangeNavigateMode "{set_mode: 0}"`
深度点云转换	`ros2 service call /cloud_processor/switch_statue std_srvs/srv/SetBool "data: true"`
速度输入	`ros2 topic pub -r 10 /vel_app motion_msgs/msg/VelCommand "{...}"`
全局目标点	`ros2 topic pub -1 /goal_pose geometry_msgs/msg/PoseStamped "{...}"`

29.6.5 5.5 激光雷达接口

本节介绍如何通过 ROS2 节点启动/关闭激光雷达扫描、执行建图与定位。

功能	服务调用
启动/关闭雷达	`ros2 service call /lds/enable std_srvs/srv/SetBool "data: true"`
启动建图	`ros2 service call /set_slam_model cartographer_ros_msgs/srv/SetSlamModel "{slam_model:1,path:'/home/eame/cust_para/maps/123'}"`
保存地图	`ros2 service call /save_map cartographer_ros_msgs/srv/SaveMap "{path:'/home/eame/maps/test'}"`
切换定位模式	`ros2 service call /set_slam_model cartographer_ros_msgs/srv/SetSlamModel "{slam_model:3,path:'/home/eame/cust_para/maps/test'}"`

29.7 常见问题与故障排查

问题类型	解决方案
接口无响应	检查 topic/service 名称、参数格式，以及相关依赖节点是否已正常启动
数据异常	使用 `ros2 topic echo` 调试原始输出，关注是否有异常值（如 nan、inf）
网络丢包	优先采用有线连接，或调整 QoS 策略以提升可靠性
权限/认证失败	确认当前账户具备访问硬件和串口的足够权限

CHAPTER 30

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

30.1 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》

30.2 支持无线开发吗？

不支持，MagicDog 目前只支持有线连接进行开发

30.3 目前底层通讯频率是多少？

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

30.4 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。

30.5 报错：Failed to disable SDK.

检查组播是否正常配置

```
ping 192.168.55.200 # 确认网卡配置是否正确，与机器狗连线是否正常
ifconifg # 确认与机器狗直连的 ethercat 网卡名称
sudo ifconfig <与机器狗直连的 ethercat 网卡名称> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <与机器狗直连的 ethercat 网卡名称>
```

30.6 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/magicdog_mjrrt.yaml` 配置文件，在/tmp/logs 目录下生成 `magicdog_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；

30.7 调用 high_level_motion_example 时，打印以下错误 Fail initializing after {} seconds. \nPlease check the lcm connection between remote and hardware PC then try again.\n

检查组播是否正常配置

```
ping 192.168.55.200 # 确认网卡配置是否正确，与机器狗连线是否正常
ifconifg # 确认与机器狗直连的 ethercat 网卡名称
sudo ifconfig <与机器狗直连的 ethercat 网卡名称> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <与机器狗直连的 ethercat 网卡名称>
```

30.8 调用 audio_example 或者 sensor_example 无法拿到对应的数据

检查组播是否正常配置

```
ping 192.168.55.200 # 确认网卡配置是否正确，与机器狗连线是否正常
ifconifg # 确认与机器狗直连的 ethercat 网卡名称
sudo ifconfig <与机器狗直连的 ethercat 网卡名称> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <与机器狗直连的 ethercat 网卡名称>
```

30.9 如何检查双目相机图像延时

`CompressedImage` 或者 `Image` 数据结构的 `header.stamp` 字段记录图片发出时间，单位：纳纱

30.10 如何获取鱼眼相机或者 4K 相机的数据

```
# 鱼眼相机：
ffplay rtsp://192.168.12.1:8080/
# 中央的4k相机：
ffplay rtsp://192.168.12.1:8082/
```

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

30.11 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：

- `SwitchToLocation()`
- `ActivateNavMode(NavMode::GRID_MAP)`

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` 获取的当前位姿进行设置目标点位姿；

30.12 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；

30.13 SDK 无法订阅话题数据或者话题数据不生效？

- 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 `OpenLaserScan` 打开雷达数据；

- 检查 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
  ```

- 检查 SDK 版本与本体软件版本是否对齐：

  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/magicdog_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 `magicdog_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。

30.14 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
```

30.15 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
```

30.16 connect 返回报错：Failed to connect to robot, code: ErrorCode.SERVICE_ERROR, message: failed to connect to all addresses; last error: UNKNOWN: ipv4:192.168.55.200:50051: Failed to connect to remote host: Connection refused

1. 无法 ping 通 `192.168.55.200`：

- 网线连接异常，检查硬件连接；

- 算力包 IP 配置不是 `192.168.55.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；