Magicdog SDK Development Documentation

Release 1.2.1-doc1

MagicLab

Dec 11, 2025

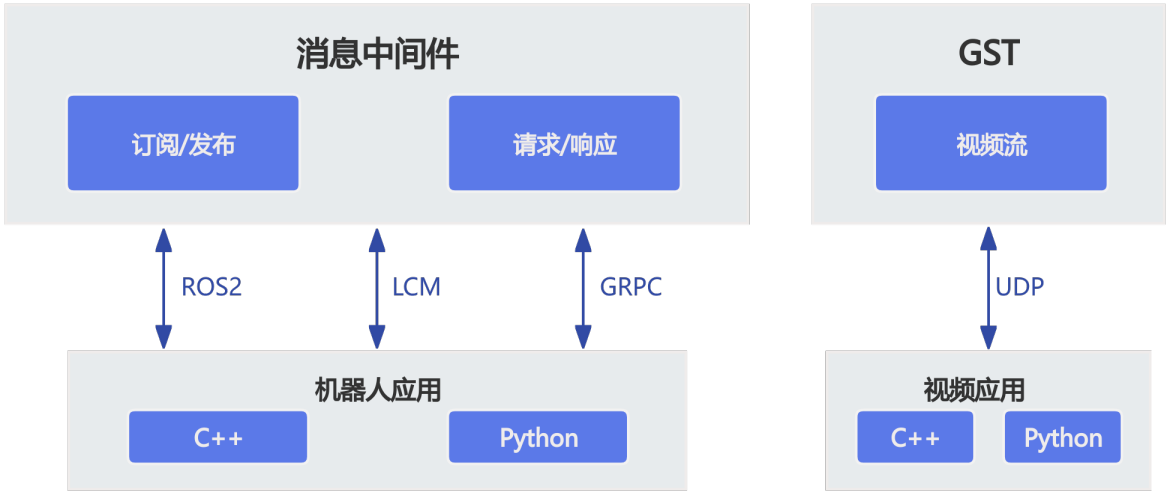
SDK INTRODUCTION:

CHAPTER

ONE

SDK OVERVIEW

1.1 SDK Communication Interface Introduction



MagicDog uses LCM/GRPC/ROS2 as message middleware. The main data interaction modes are: Topic (Subscribe/Publish) and RPC (Request/Response).

- Topic (Subscribe/Publish): The receiver subscribes to a message, and the sender sends messages to the receiver based on the subscription list. This is mainly used for medium to high-frequency or continuous data interactions.
- RPC (Request/Response): Uses a question-and-answer mode, where requests are used to obtain data or perform control operations. This is used for low-frequency or function-switching data interactions.

The main way to call Topic and RPC interfaces: Functional Interface

- Functional Interface: API calls are encapsulated as function calls for user convenience.

1.2 Getting the SDK

magicdog-sdk is the new generation robot development SDK from MagicLab. This SDK fully encapsulates interfaces such as high-level motion control, low-level motor control, and voice control, and provides corresponding functional interfaces. You can refer to our SDK tutorials to learn robot control methods and complete secondary development for MagicDog.

1.2.1 SDK Download Link:

magicdog-sdk

The SDK version must be aligned with the robot firmware version. Please refer to ChangeLog

1.2.2 URDF/MJCF Download Link:

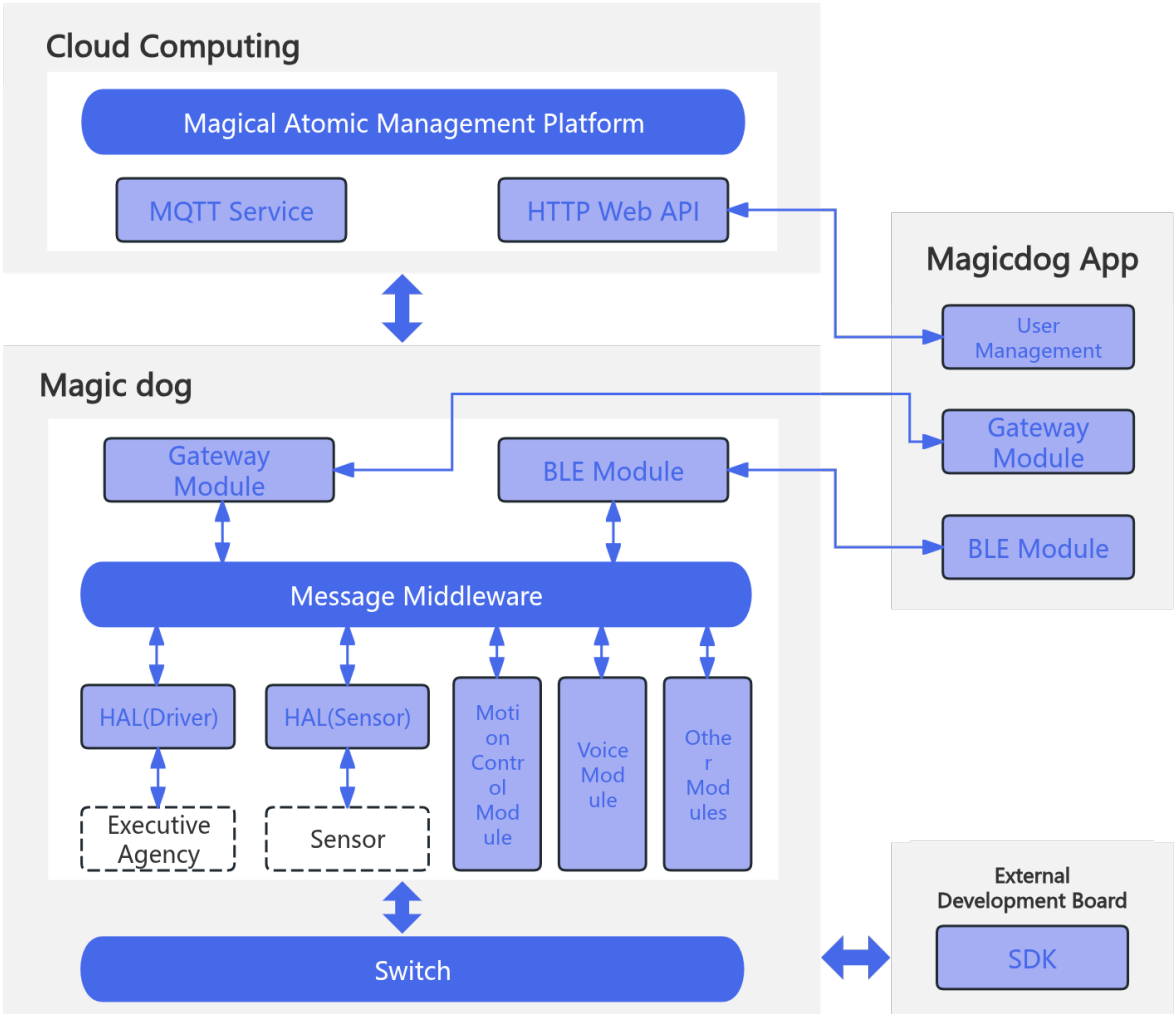
URDF/MJCF

CHAPTER

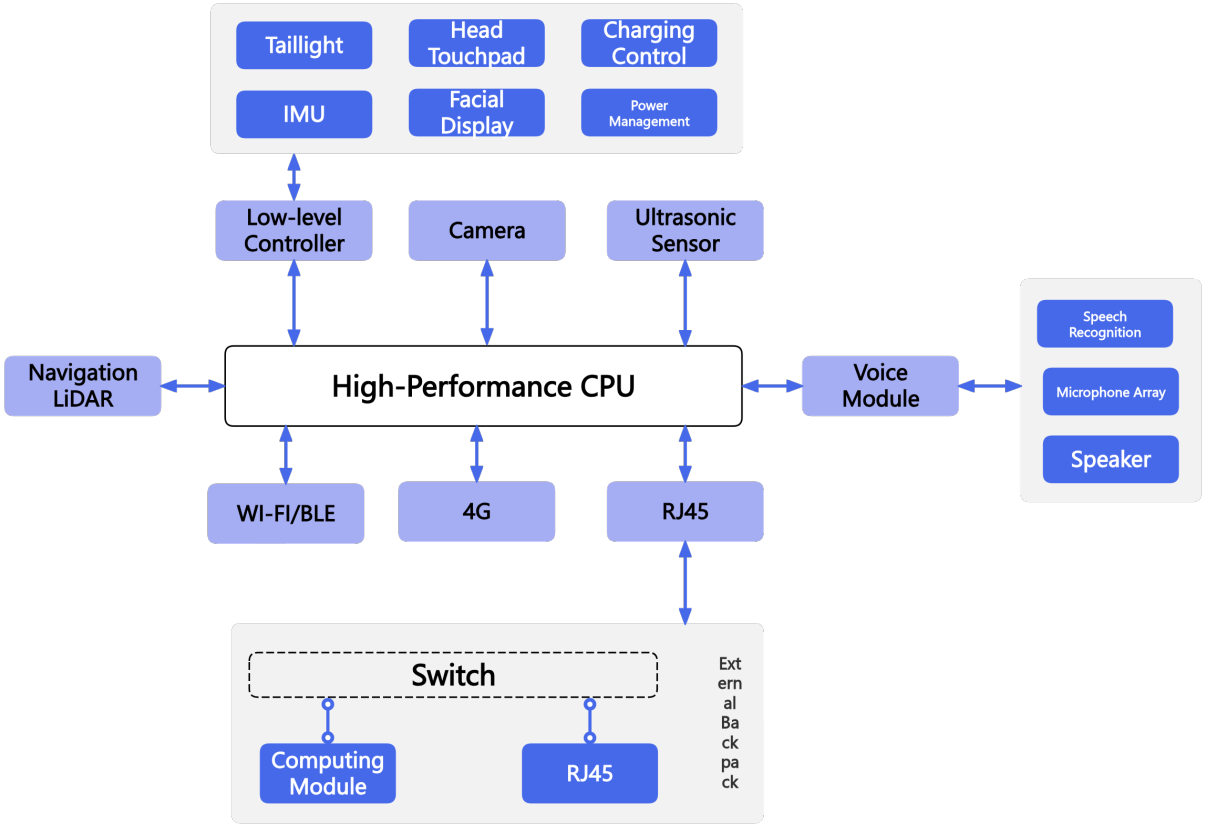
TWO

SOFTWARE AND HARDWARE ARCHITECTURE

2.1 Software System Architecture Diagram



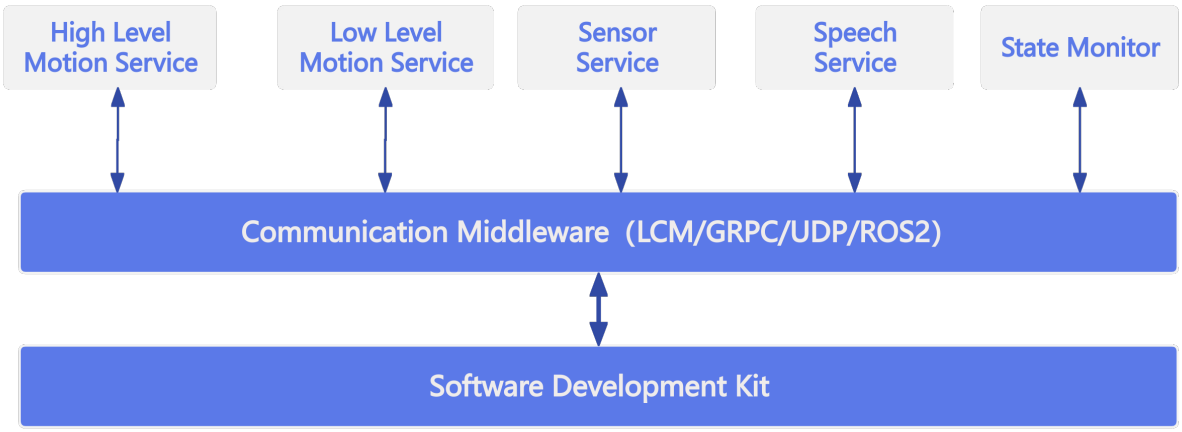
2.2 Hardware System Architecture Diagram



CHAPTER

THREE

SERVICE INTRODUCTION



MagicDog provides the following services through message middleware (LCM/GRPC/ROS2):

- **High-level Motion Control Service**

Based on MagicDog’ s built-in gait controller, this service provides functions such as gait switching, trick execution, posture control, and speed control (equivalent to remote controller operations). The high-level motion control service communicates via GRPC.

- **Low-level Motion Control Service**

Through the service interface, you can obtain joint status, IMU data, and other information in real time, as well as send joint control commands in real time. The low-level motion control service communicates via LCM.

- **Voice Service**

Through the service interface, you can control volume, play voice, and obtain raw voice data. The voice service communicates via GRPC/LCM.

- **Sensor Service**

Supports data subscription for sensors such as LiDAR, RGBD cameras, and binocular cameras. The sensor service communicates via GRPC/LCM/ROS2.

- **Status Monitoring Service**

Through the service interface, you can subscribe to the robot ' s hardware and software fault information and BMS battery status. The status monitoring service communicates via GRPC.

- **SLAM Navigation Service**

Robot SLAM mapping and navigation control can be performed through service interfaces. SLAM navigation service uses GRPC/LCM for communication.

CHAPTER

FOUR

QUICK START

4.1 System Environment

Development is recommended on Ubuntu 22.04 system. Mac/Windows systems are not currently supported for development. The robot’ s onboard PC runs official services and does not support development environments.

APP and SDK cannot suppport control the robot simultaneously

4.1.1 Development Environment Requirements

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

4.1.2 System Configuration

First, to achieve real-time communication for regular users, add the following configuration to the `/etc/security/limits.conf` file:

```
*    -   rtprio   98
```

Second, to increase the receive buffer for each socket connection, add the following configuration to the `/etc/sysctl.conf` file. Execute `sudo sysctl -p` to apply immediately or restart to take effect:

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

(continues on next page)

(continued from previous page)

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

4.2 Network Environment

Connect the user's computer and robot switch to a unified network. It is recommended that new users connect their computer to the robot switch using an Ethernet cable and set the network card communicating with the robot to the 192.168.55.X network segment, preferably 192.168.55.10. Experienced users can configure the network environment themselves.

Assuming the SDK development PC is connected to the robot through network interface `eno1`, the following configuration is needed for SDK to communicate with the robot's underlying system:

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

To avoid reconfiguring the network interface multicast route each time you reconnect the network cable, users can automate this on their development PC by following the steps below (requires Ubuntu 22.04 and NetworkManager as the system network configuration tool):

- Check the service status:

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

  If the output shows `active`, proceed to the next steps; otherwise, skip the following operations.

- Edit the configuration file: `sudo vim /etc/NetworkManager/dispatcher.d/90-add-mcast-route`

  ```
  #!/bin/bash

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

  # Target network interface
  TARGET_IF="eno1"

  # Execute only when the specified interface is carrier on or up
  if [ "$IFACE" = "$TARGET_IF" ] && [[ "$STATUS" = "up" || "$STATUS" = "carrier" ]];
  ↪ then
      # Remove possible old route
      ip route del 224.0.0.0/4 dev "$TARGET_IF" 2>/dev/null
  ```

(continues on next page)

(continued from previous page)

```
    # Add the route you want (with scope link)
    ip route add 224.0.0.0/4 dev "$TARGET_IF" scope link
fi
```

- Add execute permission:

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

- Restart the service to apply:

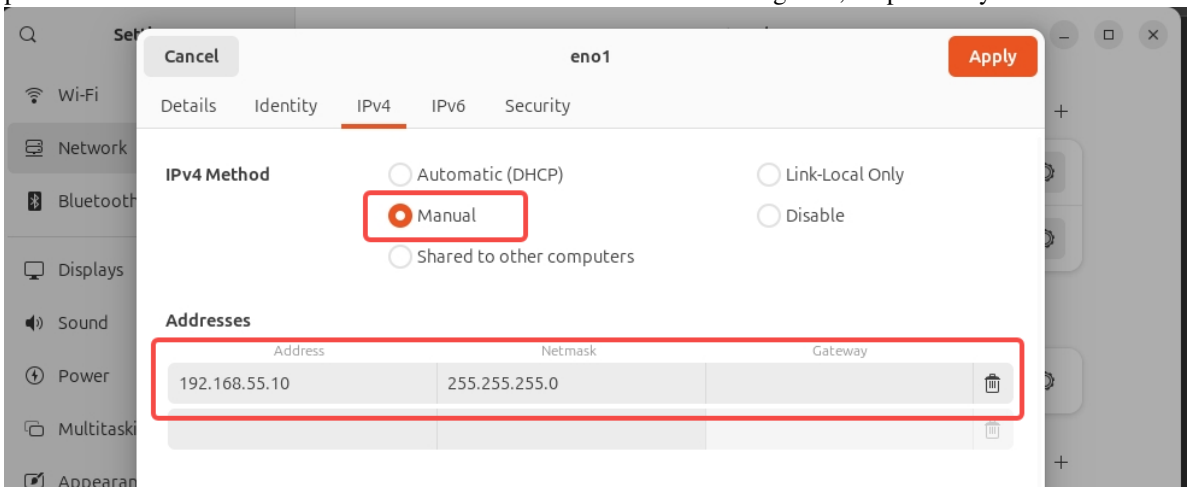
```
sudo systemctl restart NetworkManager
```

Check if the network interface route is configured correctly:

```
$ ip route | grep 224.0.0.0
224.0.0.0/4 dev eno1 scope link
```

4.2.1 Configuration Steps

1. Connect one end of the Ethernet cable to the robot and the other end to the user's computer. The robot's onboard computer IP address is 192.168.55.200, so the computer IP needs to be set to the same network segment, preferably 192.168.55.10.



2. To test if the communication connection is normal, you can use ping for testing:

```
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
```

3. After connectivity is established, execute the following commands:

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

4.3 Installation and Compilation

The following steps assume the working directory is `/home/magicbot/workspace`

4.3.1 Install 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
```

The above commands will install `magicdog_sdk` to the `/opt/magic_robotics/magicdog_sdk` directory

4.3.2 Example Compilation

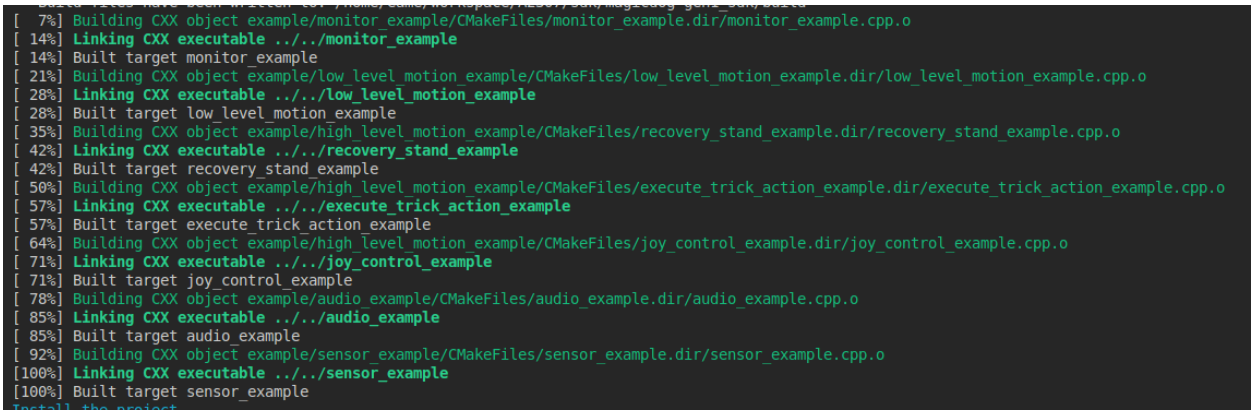
```
cd /home/magicbot/workspace/magicdog_sdk/
mkdir build
cd build
```

(continues on next page)

(continued from previous page)

```
cmake .. -DBUILD_EXAMPLES=ON
make -j8
```

After running the above commands, if the progress reaches 100% without errors, it means compilation was successful.



4.3.3 Import magicdog-sdk in User Modules

If users need to import `magicdog-sdk` in their own modules, they can refer to `example/cmake_example/CMakeLists.txt`

4.4 C++ Example Programs

In the magicdog_sdk/build directory:

- **Audio Examples**:
  - audio_example
- **Sensor Examples**:
  - sensor_example
- **State Monitoring Examples**:
  - monitor_example
- **Low-level Motion Control Examples**:
  - low_level_motion_example
- **High-level Motion Control Examples**:
  - high_level_motion_example
- **SLAM Examples**:
  - slam_example
- **Navigation Examples**:

  - navigation_example

- **Display Examples**:
  - display_example

4.4.1 Enter Debug Mode:

Follow the operation procedures to ensure the robot enters debug mode

4.4.2 Run Examples

Enter the magicdog_sdk/build directory and execute the following commands:

```
cd /home/magicbot/workspace/magicdog_sdk/build
# Environment configuration
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

# execute audio example
./audio_example
```

4.5 Python Example Programs

In the magicdog_sdk/example/python directory:

- **Audio Examples**:
  - audio_example.py
- **Sensor Examples**:
  - sensor_example.py
- **State Monitoring Examples**:
  - monitor_example.py
- **Low-level Motion Control Examples**:
  - low_level_motion_example.py
- **High-level Motion Control Examples**:
  - high_level_motion_example.py
- **SLAM Examples**:
  - slam_example.py

- **Navigation Examples**:
  - navigation_example.py
- **Display Examples**:
  - display_example.py

4.5.1 Enter Debug Mode:

Follow the operation procedures to ensure the robot enters debug mode

4.5.2 Run Examples

Enter the magicdog_sdk/example/python directory and execute the following commands:

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

# Environment configuration
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

# execute audio example
python3 audio_example.py
```

> Note: Manually executing `sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev eno1` only takes effect once. After system restart or network cable disconnection, it needs to be executed again.

Get Python API help information:

```
cd /home/magicbot/workspace/magicdog_sdk/example/python
# Environment configuration
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.
# Import Magicdog Python SDK interface
>>> import magicdog_python
# View all interface information
>>> help(magicdog_python)
# View CameraInfo structure information
```

(continues on next page)

(continued from previous page)

```
>>> help(magicdog_python.CameraInfo)
# View HighLevelMotionController object information
>>> help(magicdog_python.HighLevelMotionController)
# View LowLevelMotionController object information
>>> help(magicdog_python.LowLevelMotionController)
# View SensorController object information
>>> help(magicdog_python.SensorController)
# View AudioController object information
>>> help(magicdog_python.AudioController)
# View StateMonitor object information
>>> help(magicdog_python.StateMonitor)
```

For example, if you want to view the enumeration values of the trick action `TrickAction`:

```
$ 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__(...)
```

(continues on next page)

(continued from previous page)

```
 |      __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
 |
 |  __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>
 |
```

(continues on next page)

(continued from previous page)

```
|  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>
|
|  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>
|
```

(continues on next page)

(continued from previous page)

```
|  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>
|
|  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:
```

(continues on next page)

(continued from previous page)

```
 |
 |  __new__(*args, **kwargs) from pybind11_builtins.pybind11_type
 |      Create and return a new object.  See help(type) for accurate signature.
```

4.6 Others

4.6.1 SDK Configuration File

The SDK generates its default configuration file in the /tmp directory at runtime. If the configuration file already exists, it will use the existing configuration:

```
$ ls /tmp/magicdog_mjrrt.yaml
/tmp/magicdog_mjrrt.yaml
```

4.6.2 SDK Logs

The internal log information of the SDK at runtime is generated by default in the /tmp/logs directory:

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

CHAPTER

FIVE

ROBOT MAIN CONTROL SERVICE (C++)

Provides the robot system main controller. Through MagicRobot, you can perform resource initialization, manage communication connections, and access various sub-controllers such as motion controllers, audio controllers, state monitoring, and sensor controllers.

5.1 Interface Definition

MagicRobot is the unified entry class for the robot system.

5.1.1 MagicRobot

Item	Content
Function Name	MagicRobot
Function Declaration	`MagicRobot();`
Function Overview	Constructor, creates a MagicRobot instance.
Notes	Constructs internal state.

5.1.2 ~MagicRobot

Item	Content
Function Name	~MagicRobot
Function Declaration	`~MagicRobot();`
Function Overview	Destructor, releases MagicRobot instance resources.
Notes	Ensures safe resource release.

5.1.3 Initialize

Item	Content
Function Name	Initialize
Function Declaration	`bool Initialize(const std::string& local_ip);`
Function Overview	Initializes the robot system, including controllers and network modules.
Parameter Description	`local_ip`: Local communication IP address.
Return Value	`true` indicates success, `false` indicates failure.
Notes	Must be called before first use.

5.1.4 Shutdown

Item	Content
Function Name	Shutdown
Function Declaration	`void Shutdown();`
Function Overview	Shuts down the robot system and releases resources.
Notes	Used in conjunction with Initialize.

5.1.5 Connect

Item	Content
Function Name	Connect
Function Declaration	`Status Connect(int timeout_ms);`
Function Overview	Establishes a communication connection with the robot service.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` indicates success.
Notes	Blocking interface, should be called after initialization.

5.1.6 Release

Item	Content
Function Name	Release
Function Declaration	`void Release();`
Function Overview	Releases robot system resources.
Notes	Call this function to manually release resources occupied by the SDK.

5.1.7 Disconnect

Item	Content
Function Name	Disconnect
Function Declaration	`Status Disconnect(int timeout_ms);`
Function Overview	Disconnects from the robot service.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	gRPC call status.
Notes	Blocking interface, used in pair with Connect.

5.1.8 GetSDKVersion

Item	Content
Function Name	GetSDKVersion
Function Declaration	`std::string GetSDKVersion() const;`
Function Overview	Gets the current SDK version.
Return Value	SDK version string (e.g., `0.0.1`).
Notes	Non-blocking interface, useful for debugging or log marking.

5.1.9 GetMotionControlLevel

Item	Content
Function Name	GetMotionControlLevel
Function Declaration	`ControllerLevel GetMotionControlLevel();`
Function Overview	Gets the current motion control level (high/low).
Return Value	`ControllerLevel` enum value.
Notes	Non-blocking interface, used to determine current control permission.

5.1.10 SetMotionControlLevel

Item	Content
Function Name	SetMotionControlLevel
Function Declaration	`Status SetMotionControlLevel(ControllerLevel level);`
Function Overview	Sets the current motion control level.
Parameter Description	`level`: Control permission enum value.
Return Value	`Status::OK` indicates success, others indicate failure.
Notes	Blocking interface, used when switching control modes.

5.1.11 GetHighLevelMotionController

Item	Content
Function Name	GetHighLevelMotionController
Function Declaration	`HighLevelMotionController& GetHighLevelMotionController();`
Function Overview	Gets the high-level motion controller object.
Return Value	Reference type, used to call high-level control interfaces.
Notes	Non-blocking interface, encapsulates gait, stunt, remote control and other control functions.

5.1.12 GetLowLevelMotionController

Item	Content
Function Name	GetLowLevelMotionController
Function Declaration	`LowLevelMotionController& GetLowLevelMotionController();`
Function Overview	Gets the low-level motion controller object.
Return Value	Reference type, used to control joints or motors.
Notes	Non-blocking interface, can directly control joint motors, etc.

5.1.13 GetAudioController

Item	Content
Function Name	GetAudioController
Function Declaration	`AudioController& GetAudioController();`
Function Overview	Gets the audio controller object.
Return Value	Reference type, used for voice control.
Notes	Non-blocking interface, can play voice and control volume.

5.1.14 GetSensorController

Item	Content
Function Name	GetSensorController
Function Declaration	`SensorController& GetSensorController();`
Function Overview	Gets the sensor controller object.
Return Value	Reference type, used to access sensor data.
Notes	Non-blocking interface, encapsulates IMU, RGBD and other sensor data reading.

5.1.15 GetStateMonitor

Item	Content
Function Name	GetStateMonitor
Function Declaration	`StateMonitor& GetStateMonitor();`
Function Overview	Gets the state monitor object.
Return Value	Reference type, used to get the current robot state information.
Notes	Non-blocking interface, encapsulates BMS, fault and other state information reading.

5.1.16 GetSlamNavController

Item	Content
Function Name	GetSlamNavController
Function Declaration	`SlamNavController& GetSlamNavController();`
Function Overview	Gets the SLAM/navigation controller object.
Return Value	Reference type, used to access map, localization, and other SLAM-related data.
Notes	Non-blocking interface, encapsulates features such as map building, localization, and navigation.

5.1.17 OpenChannelSwitch

Item	Content
Function Name	OpenChannelSwitch
Function Declaration	`Status OpenChannelSwitch(int timeout_ms);`
Function Overview	Opens the lcm channel switch.
Parameter	`timeout_ms` (optional parameter, milliseconds), operation timeout, default is 5000 ms.
Return Value	`Status` enum indicating the operation status.
Notes	Used to enable the SDK lcm communication channel.

5.1.18 CloseChannelSwitch

Item	Content
Function Name	CloseChannelSwitch
Function Declaration	`Status CloseChannelSwitch(int timeout_ms);`
Function Overview	Closes the lcm channel switch.
Parameter	`timeout_ms` (optional parameter, milliseconds), operation timeout, default is 5000 ms.
Return Value	`Status` enum indicating the operation status.
Notes	Used to disable the SDK lcm communication channel.

CHAPTER

SIX

HIGH-LEVEL MOTION CONTROL SERVICE (C++)

Provides high-level motion control services for the robot system. Through HighLevelMotionController, you can control robot gait, tricks, and remote control via RPC communication.

⚠ **Notice:** Some trick actions, such as front flips and back flips, require large spaces. Extra attention is needed during execution. It is recommended to test all trick actions in large open spaces before using them in smaller areas.

6.1 Interface Definition

HighLevelMotionController is a high-level motion controller for semantic control, supporting control operations such as walking and tricks, encapsulating low-level details for upper system calls.

6.1.1 HighLevelMotionController

Item	Content
Function Name	HighLevelMotionController
Function Declaration	`HighLevelMotionController();`
Function Overview	Constructor, initializes high-level controller state.
Notes	Constructs internal control resources.

6.1.2 ~HighLevelMotionController

Item	Content
Function Name	~HighLevelMotionController
Function Declaration	`virtual ~HighLevelMotionController();`
Function Overview	Destructor, releases controller resources.
Notes	Used together with the constructor.

6.1.3 Initialize

Item	Content
Function Name	Initialize
Function Declaration	`virtual bool Initialize() override;`
Function Overview	Initializes the controller and prepares high-level control functions.
Return Value	`true` means success, `false` means failure.
Notes	Must be called before first use.

6.1.4 Shutdown

Item	Content
Function Name	Shutdown
Function Declaration	`virtual void Shutdown() override;`
Function Overview	Shuts down the controller and releases resources.
Notes	Used together with Initialize.

6.1.5 SetGait

Item	Content
Function Name	SetGait
Function Declaration	`Status SetGait(const GaitMode gait_mode, int timeout_ms);`
Function Overview	Set the robot’s gait mode (e.g., position control stand, force control stand, jogging, etc.).
Parameter Description	`gait_mode`: Gait control enumeration.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, supports switching between multiple gait modes.

6.1.6 GetGait

Item	Content
Function Name	GetGait
Function Declaration	`Status GetGait(GaitMode& gait_mode, int timeout_ms);`
Function Overview	Get the robot’s gait mode (e.g., position control stand, force control stand, jogging, etc.).
Parameter Description	`gait_mode`: Gait control enumeration.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, gets the current gait mode.

6.1.7 ExecuteTrick

Item	Content
Function Name	ExecuteTrick
Function Declaration	`Status ExecuteTrick(const TrickAction trick_action, int timeout_ms);`
Function Overview	Execute trick actions (e.g., wiggle hip, lie down, etc.).
Parameter Description	`trick_action`: Trick action identifier.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, ensure the robot is in a state where tricks can be executed.

- Note: Trick actions must be executed under GaitMode::GAIT_STAND_R(2) (position control stand gait).

6.1.8 EnableJoyStick

Item	Content
Function Name	EnableJoyStick
Function Declaration	`Status EnableJoyStick();`
Function Overview	Enable joystick control commands.
Parameter Description	None.
Return Value	`Status::OK` means success, others mean failure.
Notes	Used to allow sending real-time control commands via joystick.

6.1.9 DisableJoyStick

Item	Content
Function Name	DisableJoyStick
Function Declaration	`Status DisableJoyStick();`
Function Overview	Used to disable real-time control commands sent by joystick.
Parameter Description	None.
Return Value	`Status::OK` means operation was successful, others indicate failure.
Notes	You must disable joystick control commands before switching to navigation mode.

6.1.10 SendJoyStickCommand

Item	Content
Function Name	SendJoyStickCommand
Function Declaration	`Status SendJoyStickCommand(const JoystickCommand& joy_command);`
Function Overview	Send real-time joystick control command.
Parameter Description	`joy_command`: Control data containing joystick coordinates.
Return Value	`Status::OK` means success, others mean failure.
Notes	Non-blocking interface, recommended sending frequency is 20Hz.

6.1.11 GetAllGaitSpeedRatio

Item	Content
Function Name	GetAllGaitSpeedRatio
Function Declaration	`Status GetAllGaitSpeedRatio(AllGaitSpeedRatio& gait_speed_ratios, int timeout_ms);`
Function Overview	Get all gaits and their corresponding forward, lateral, and rotational speed ratios.
Parameter Description	`gait_speed_ratios`: All gaits and their corresponding speed ratios.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to get speed ratio configuration for all gaits.

6.1.12 SetGaitSpeedRatio

Item	Content
Function Name	SetGaitSpeedRatio
Function Declaration	`Status SetGaitSpeedRatio(GaitMode gait_mode, const GaitSpeedRatio& gait_speed_ratio, int timeout_ms);`
Function Overview	Set gait and its corresponding forward, lateral, and rotational speed ratios.
Parameter Description	`gait_mode`: Gait mode`gait_speed_ratio`: Gait and its corresponding speed ratios.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to set speed ratio configuration for a specific gait.

6.1.13 GetHeadMotorEnabled

Item	Content
Function Name	GetHeadMotorEnabled
Function Declaration	`Status GetHeadMotorEnabled(bool& enabled, int timeout_ms);`
Function Overview	Get the enable status of the head motor.
Parameter Description	`enabled`: Return parameter, true means enabled, false means not enabled.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to query the current enable status of the head motor.

6.1.14 EnableHeadMotor

Item	Content
Function Name	EnableHeadMotor
Function Declaration	`Status EnableHeadMotor(int timeout_ms);`
Function Overview	Enable the head motor.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to enable head motor control.

6.1.15 DisableHeadMotor

Item	Content
Function Name	DisableHeadMotor
Function Declaration	`Status DisableHeadMotor(int timeout_ms);`
Function Overview	Disable the head motor.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to disable head motor control.

6.1.16 GetHeadPosition

Item	Content
Function Name	GetHeadPosition
Function Declaration	`Status GetHeadPosition(EulerAngles& euler_angles, int timeout_ms);`
Function Overview	Get current head position.
Parameter Description	`euler_angles`: Return parameter, current absolute position of head.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to get current head position.

6.1.17 SetHeadPosition

Item	Content
Function Name	SetHeadPosition
Function Declaration	`Status SetHeadPosition(const EulerAngles& euler_angles, int timeout_ms);`
Function Overview	Set the head position.
Parameter Description	`euler_angles`: Absolute position of head target.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to set the head position.

6.2 Type Definitions

6.2.1 GaitSpeedRatio —Gait Speed Ratio Structure

Field Name	Type	Description
straight_ratio	double	Forward speed ratio
turn_ratio	double	Rotational speed ratio
lateral_ratio	double	Lateral speed ratio

6.2.2 AllGaitSpeedRatio —All Gait Speed Ratio Structure

Field Name	Type	Description
gait_speed_ratios	std::map<GaitMode, GaitSpeedRatio>	Mapping from gait mode to speed ratio

6.2.3 JoystickCommand —High-Level Motion Control Joystick Command Structure

Field Name	Type	Description
left_x_axis	float	X axis value of the left joystick (-1.0: left, 1.0: right)
left_y_axis	float	Y axis value of the left joystick (-1.0: down, 1.0: up)
right_x_axis	float	X axis value of the right joystick (rotation -1.0: left, 1.0: right)
right_y_axis	float	Y axis value of the right joystick (usage not defined yet)

6.2.4 EulerAngles —Euler angles Structure

Field Name	Type	Description
roll	double	Roll angle - rotation around the X-axis(in radians)
pitch	double	Pitch angle - rotation around the Y-axis(in radians)
yaw	double	XYaw angle - rotation around the Z-axis(in radians)

6.3 Enum Type Definitions

6.3.1 GaitMode —Robot Gait Mode Enumeration

Enum Value	Value	Description
GAIT_PASSIVE	0	Drop (disable motor enable)
GAIT_STAND_R	2	Recovery stand, RecoveryStand
GAIT_STAND_B	3	Force control stand, pose display, BalanceStand
GAIT_RUN_FAST	8	Fast run
GAIT_DOWN_CLIMB_STAIRS	9	Downstairs => blind walk => jog
GAIT_TROT	10	Trot
GAIT_PRONK	11	Jump
GAIT_BOUND	12	Jump forward and backward
GAIT_AMBLE	14	Amble
GAIT_CRAWL	29	Crawl
GAIT_LOWLEVL_SDK	30	Low-level SDK gait
GAIT_WALK	39	Walk slowly
GAIT_UP_CLIMB_STAIRS	56	Climb stairs (all terrain)
GAIT_RL_TERRAIN	110	All terrain
GAIT_RL_FALL_RECOVERY	111	Fall recovery
GAIT_RL_HAND_STAND	112	Handstand
GAIT_RL_FOOT_STAND	113	Upright
GAIT_ENTER_RL	1001	Enter RL
GAIT_DEFAULT	99	Default
GAIT_NONE	9999	No gait

6.3.2 TrickAction —Trick Action Command Enumeration

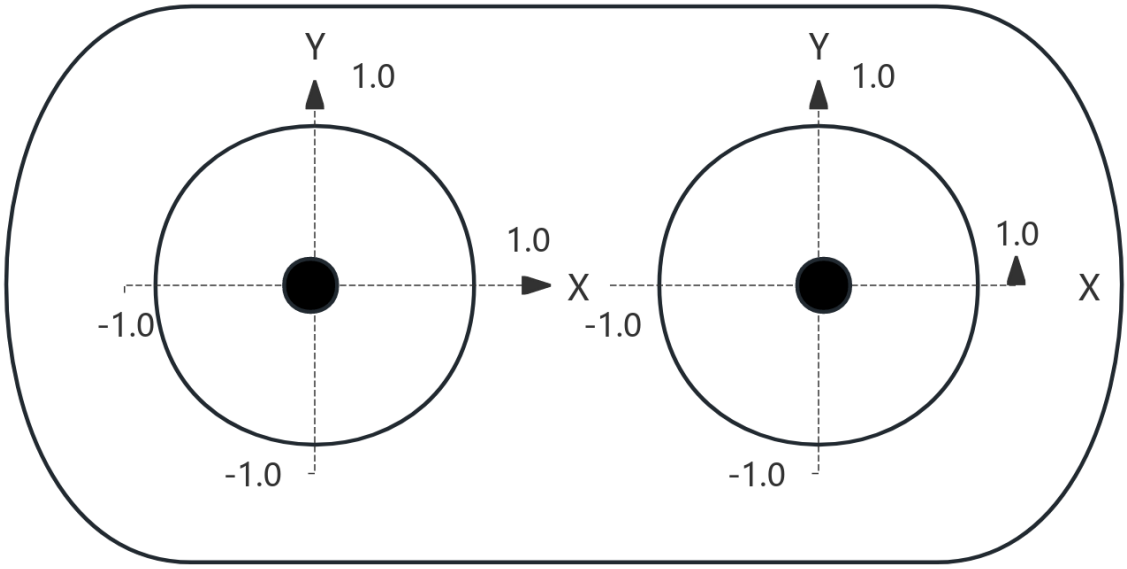
Enum Value	Value	Description
ACTION_NONE	0	No action
ACTION_WIGGLE_HIP	26	Wiggle hip
ACTION_SWING_BODY	27	Swing body
ACTION_STRETCH	28	Stretch
ACTION_STOMP	29	Stomp
ACTION_JUMP_JACK	30	Jumping jack
ACTION_SPACE_WALK	31	Moonwalk
ACTION_IMITATE	32	Imitate
ACTION_SHAKE_HEAD	33	Shake head

continues on next page

Table 1 – continued from previous page

Enum Value	Value	Description
ACTION_PUSH_UP	34	Push-up
ACTION_CHEER_UP	35	Cheer up
ACTION_HIGH_FIVES	36	High fives
ACTION_SCRATCH	37	Scratch
ACTION_HIGH_JUMP	38	High jump
ACTION_SWING_DANCE	39	Swing dance
ACTION_LEAP_FROG	40	Leap frog
ACTION_BACK_FLIP	41	Back flip
ACTION_FRONT_FLIP	42	Front flip
ACTION_SPIN_JUMP_LEFT	43	Spin jump left 70 degrees
ACTION_SPIN_JUMP_RIGHT	44	Spin jump right 70 degrees
ACTION_JUMP_FRONT	45	Jump forward 0.5 meters
ACTION_ACT_CUTE	46	Act cute
ACTION_BOXING	47	Boxing
ACTION_SIDE_SOMERSAULT	48	Side somersault
ACTION_RANDOM_DANCE	49	Random dance
ACTION_LEFT_SIDE_SOMERSAULT	84	Left side somersault
ACTION_RIGHT_SIDE_SOMERSAULT	85	Right side somersault
ACTION_DANCE2	91	Dance 2
ACTION_EMERGENCY_STOP	101	Emergency stop
ACTION_LIE_DOWN	102	Lie down
ACTION_RECOVERY_STAND	103	Stand up
ACTION_HAPPY_NEW_YEAR	105	New Year greeting (bow)
ACTION_SLOW_GO_FRONT	108	Come here
ACTION_SLOW_GO_BACK	109	Go back
ACTION_BACK_HOME	110	Go home
ACTION_LEAVE_HOME	111	Leave home
ACTION_TURN_AROUND	112	Turn around
ACTION_DANCE	115	Dance
ACTION_ROLL_ABOUT	116	Roll about
ACTION_SHAKE_RIGHT_HAND	117	Shake right hand
ACTION_SHAKE_LEFT_HAND	118	Shake left hand
ACTION_SIT_DOWN	119	Sit down

6.4 Joystick Diagram



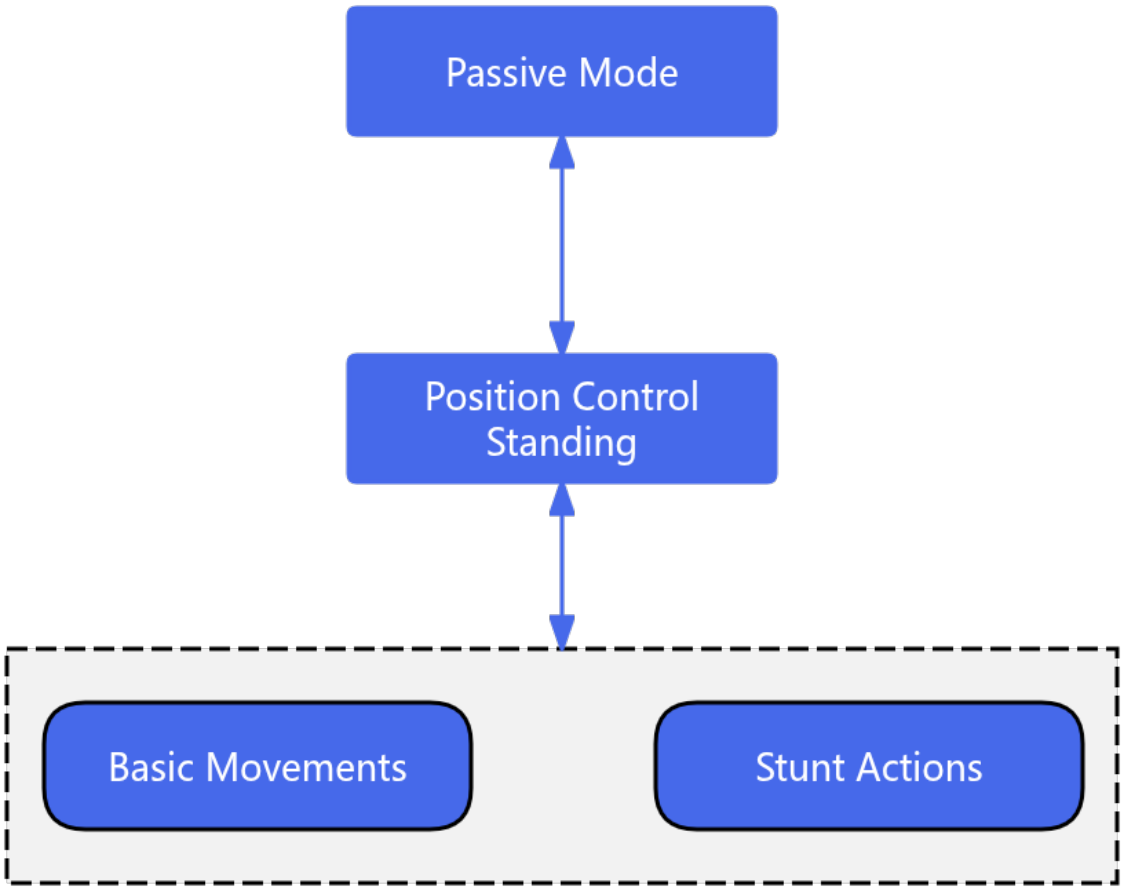
1. The value range of the x and y axes of the left and right joysticks is [-1.0, 1.0];
2. The positive direction of the x and y axes of the left and right joysticks is up/right, as shown in the diagram;

6.5 High-Level Motion Control Robot State Introduction

The robot’ s motion includes position control stand, force control stand, basic motion, and trick action states. During operation, the robot switches between different states through a state machine to accomplish different control tasks. The explanations for each state are as follows:

- Recovery stand: In this state, you can call various SDK interfaces to perform trick actions and basic motion control.
- Force control stand: In this state, it can be used for pose display.
- Basic motion: During motion execution, you can call SDK interfaces to let the robot enter different gaits.
- Trick action: When entering the special action execution state, other motion control services will be suspended. After the current action is completed and the robot returns to balance stand, other services will take effect again.

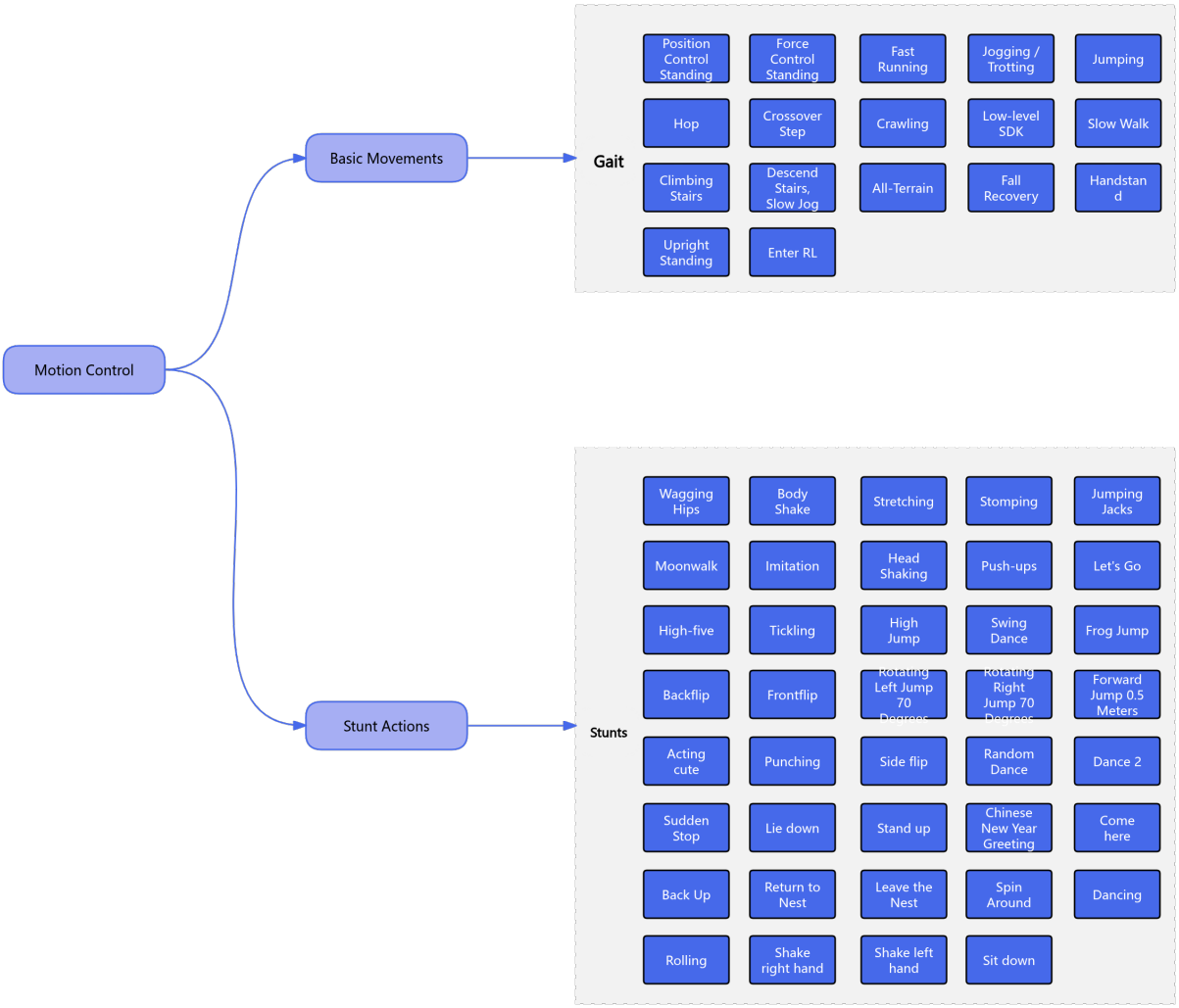
Robot state switching mechanism:



6.6 High-Level Motion Control Interfaces

The robot’ s high-level motion control services are divided into basic motion control and trick action control.

- In the basic motion control service, you can call the corresponding interfaces to switch the robot’ s walking gait according to different terrain scenarios and task requirements.
- In the trick action control service, you can call the corresponding interfaces to perform the robot’ s built-in special tricks, such as wiggle hip, lie down, etc.


Motion Control
Basic Movements
Gait
Position Control Standing
Force Control Standing
Fast Running
Jogging / Trotting
Jumping
Hop
Crossover Step
Crawling
Low-level SDK
Slow Walk
Climbing Stairs
Descend Stairs, Slow Jog
All-Terrain
Fall Recovery
Handstand
Upright Standing
Enter RL
Stunt Actions
Stunts
Wagging Hips
Body Shake
Stretching
Stomping
Jumping Jacks
Moonwalk
Imitation
Head Shaking
Push-ups
Let's Go
High-five
Tickling
High Jump
Swing Dance
Frog Jump
Backflip
Frontflip
Rotating Left Jump 70 Degrees
Rotating Right Jump 70 Degrees
Forward Jump 0.5 Meters
Acting cute
Punching
Side flip
Random Dance
Dance 2
Sudden Stop
Lie down
Stand up
Chinese New Year Greeting
Come here
Back Up
Return to Nest
Leave the Nest
Spin Around
Dancing
Rolling
Shake right hand
Shake left hand
Sit down

CHAPTER

SEVEN

LOW LEVEL MOTION CONTROL SERVICE (C++)

Provides low-level motion control services for the robot system. Through LowLevelMotionController, you can use topic-based communication to send joint commands and receive joint states.

7.1 Interface Definition

LowLevelMotionController is a motion controller for low-level development, supporting direct control and state subscription of leg and other moving parts.

⚠ **Notice:** Only when you call the PublishLegCommand interface will your command be sent to the motion controller. Therefore, you must ensure the calling frequency of PublishLegCommand, which is recommended to be 500 Hz.

7.1.1 LowLevelMotionController

Item	Description
Function Name	LowLevelMotionController
Declaration	`LowLevelMotionController();`
Overview	Constructor, initializes the low-level controller object.
Note	Constructs internal resources.

7.1.2 ~LowLevelMotionController

Item	Description
Function Name	~LowLevelMotionController
Declaration	`virtual ~LowLevelMotionController();`
Overview	Destructor, releases resources.
Note	Cleans up low-level resources.

7.1.3 Initialize

Item	Description
Function Name	Initialize
Declaration	`virtual bool Initialize() override;`
Overview	Initializes the controller and establishes low-level connections.
Return Value	`true` means success, `false` means failure.
Note	Must be initialized on first call.

7.1.4 Shutdown

Item	Description
Function Name	Shutdown
Declaration	`virtual void Shutdown() override;`
Overview	Shuts down the controller and releases low-level resources.
Note	Used together with Initialize.

7.1.5 SubscribeLegState

Item	Description
Function Name	SubscribeLegState
Declaration	`void SubscribeLegState(LegJointStateCallback callback);`
Overview	Subscribe to leg joint state data.
Parameters	callback: Callback function to handle received leg joint state data.
Note	Non-blocking interface.

7.1.6 UnsubscribeLegState

Item	Description
Function Name	UnsubscribeLegState
Declaration	`void UnsubscribeLegState();`
Overview	Unsubscribe from leg joint state data.
Note	Non-blocking interface. Used together with SubscribeLegState.

7.1.7 PublishLegCommand

Item	Description
Function Name	PublishLegCommand
Declaration	`Status PublishLegCommand(const LegJointCommand& command);`
Overview	Publish leg joint control command.
Parameters	command: Leg joint control command containing target position/velocity and other control information.
Return Value	`Status::OK` means success, others mean failure.
Note	Non-blocking interface.

7.2 Type Definitions

7.2.1 SingleLegJointCommand —Control command for a single leg joint

Field Name	Type	Description
`q_des`	`float`	Desired joint position
`dq_des`	`float`	Desired joint velocity
`tau_des`	`float`	Desired feedforward torque
`kp`	`float`	P gain, must be positive
`kd`	`float`	D gain, must be positive

7.2.2 LegJointCommand —Control command for the entire leg

Field Name	Type	Description
`timestamp`	`int64_t`	Timestamp (unit: nanoseconds)
`cmd`	`std::array<SingleLegJointCommand, kLegJointNum>`	Array of control commands

7.2.3 SingleLegJointState —State of a single leg joint

Field Name	Type	Description
q	float	Actual joint position
dq	float	Actual joint velocity
tau_est	float	Estimated torque

7.2.4 LegState —State information for the entire leg

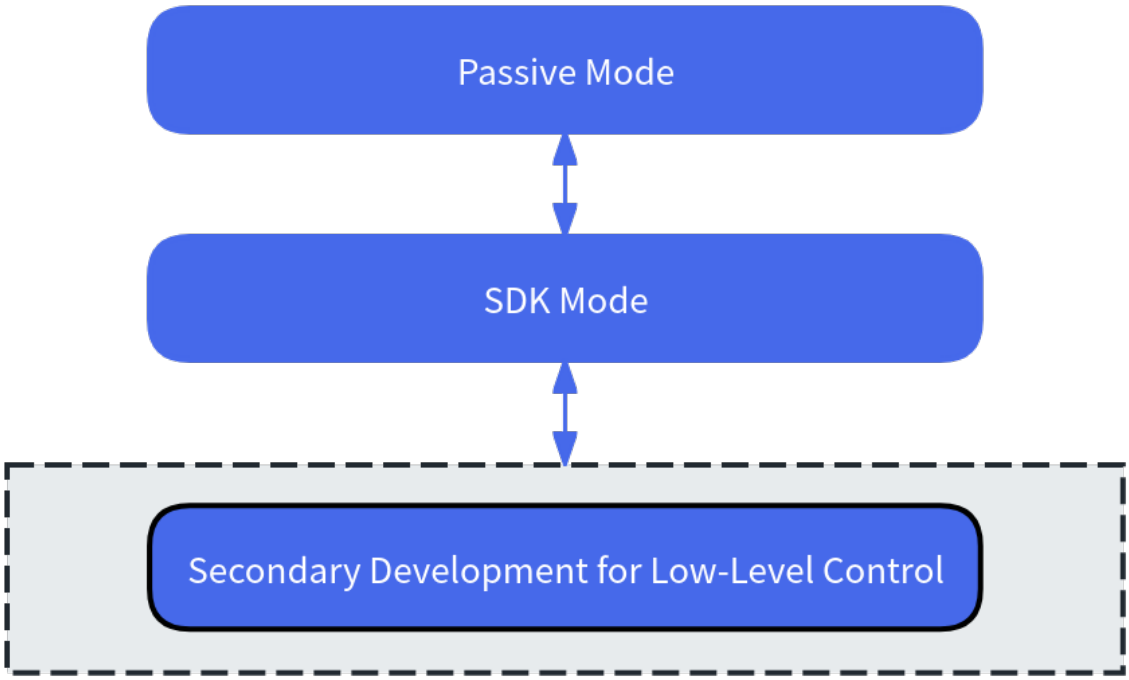
Field Name	Type	Description
timestamp	int64_t	Timestamp (unit: nanoseconds)
state	std::array<SingleLegJointState, kLegJointNum>	All leg joint states

7.3 URDF Reference

Robot URDF

7.4 Introduction to Low Level Motion Control Robot States

The robot’ s low-level motion mainly provides three-loop joint control for developers to enable secondary development of robot motion capabilities. The basic control state switching mechanism:



7.5 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

EIGHT

SENSOR CONTROL SERVICE (C++)

Provides services for robot system sensors (Lidar/RGBD Camera/Binocular Camera). Through the SensorController, you can control robot sensors and obtain their status via RPC and topic mechanisms.

8.1 Interface Definition

SensorController is a management class that encapsulates various robot sensors, supporting initialization, control, and data subscription for Laser Scan, RGBD Camera, and Binocular Camera.

8.1.1 SensorController

Item	Description
Function Name	SensorController
Declaration	`SensorController();`
Overview	Constructor, initializes the sensor controller object.
Note	Constructs internal state.

8.1.2 ~SensorController

Item	Description
Function Name	~SensorController
Declaration	`virtual ~SensorController();`
Overview	Destructor, releases all sensor resources.
Note	Sensors should be closed before calling.

8.1.3 Initialize

Item	Description
Function Name	Initialize
Declaration	`bool Initialize();`
Overview	Initializes the controller, including resource allocation and driver loading.
Return Value	`true` for success, `false` for failure.
Note	Object must be constructed before calling.

8.1.4 Shutdown

Item	Description
Function Name	Shutdown
Declaration	`void Shutdown();`
Overview	Closes all sensor connections and releases resources.
Note	Used together with Initialize.

8.1.5 OpenLaserScan

Item	Description
Function Name	OpenLaserScan
Declaration	`Status OpenLaserScan(int timeout_ms);`
Overview	Opens the lidar.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

8.1.6 CloseLaserScan

Item	Description
Function Name	CloseLaserScan
Declaration	`Status CloseLaserScan(int timeout_ms);`
Overview	Closes the lidar.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with the open function.

8.1.7 OpenRgbdCamera

Item	Description
Function Name	OpenRgbdCamera
Declaration	`Status OpenRgbdCamera(int timeout_ms);`
Overview	Opens the RGBD camera.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

8.1.8 CloseRgbdCamera

Item	Description
Function Name	CloseRgbdCamera
Declaration	`Status CloseRgbdCamera(int timeout_ms);`
Overview	Closes the RGBD camera.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with the open function.

8.1.9 OpenBinocularCamera

Item	Description
Function Name	OpenBinocularCamera
Declaration	`Status OpenBinocularCamera(int timeout_ms);`
Overview	Opens the binocular camera.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

8.1.10 CloseBinocularCamera

Item	Description
Function Name	CloseBinocularCamera
Declaration	`Status CloseBinocularCamera(int timeout_ms);`
Overview	Closes the binocular camera.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with the open function.

8.1.11 SubscribeUltra

Item	Description
Function Name	SubscribeUltra
Declaration	`void SubscribeUltra(const UltraCallback callback);`
Overview	Subscribe to ultrasonic data.
Parameters	callback: Callback function to handle received ultrasonic data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.12 UnsubscribeUltra

Item	Description
Function Name	UnsubscribeUltra
Declaration	`void UnsubscribeUltra();`
Overview	Unsubscribe from ultrasonic data.
Note	Non-blocking interface, used to stop receiving ultrasonic data.

8.1.13 SubscribeHeadTouch

Item	Description
Function Name	SubscribeHeadTouch
Declaration	`void SubscribeHeadTouch(const HeadTouchCallback callback);`
Overview	Subscribe to head touch data.
Parameters	callback: Callback function to handle received head touch data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.14 UnsubscribeHeadTouch

Item	Description
Function Name	UnsubscribeHeadTouch
Declaration	`void UnsubscribeHeadTouch();`
Overview	Unsubscribe from head touch data.
Note	Non-blocking interface, used to stop receiving head touch data.

8.1.15 SubscribeLaserScan

Item	Description
Function Name	SubscribeLaserScan
Declaration	`void SubscribeLaserScan(const LaserScanCallback callback);`
Overview	Subscribe to lidar data.
Parameters	callback: Callback function to handle received lidar data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.16 UnsubscribeLaserScan

Item	Description
Function Name	UnsubscribeLaserScan
Declaration	`void UnsubscribeLaserScan();`
Overview	Unsubscribe from lidar data.
Note	Non-blocking interface, used to stop receiving lidar data.

8.1.17 SubscribeRgbDepthCameraInfo

Item	Description
Function Name	SubscribeRgbDepthCameraInfo
Declaration	`void SubscribeRgbDepthCameraInfo(const CameraInfoCallback callback);`
Overview	Subscribe to RGBD depth camera intrinsic data.
Parameters	callback: Callback function to handle received RGBD depth camera intrinsic data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.18 UnsubscribeRgbDepthCameraInfo

Item	Description
Function Name	UnsubscribeRgbDepthCameraInfo
Declaration	`void UnsubscribeRgbDepthCameraInfo();`
Overview	Unsubscribe from RGBD depth camera intrinsic data.
Note	Non-blocking interface, used to stop receiving RGBD depth camera intrinsic data.

8.1.19 SubscribeRgbdDepthImage

Item	Description
Function Name	SubscribeRgbdDepthImage
Declaration	`void SubscribeRgbdDepthImage(const ImageCallback callback);`
Overview	Subscribe to RGBD depth image data.
Parameters	callback: Callback function to handle received RGBD depth image data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.20 UnsubscribeRgbdDepthImage

Item	Description
Function Name	UnsubscribeRgbdDepthImage
Declaration	`void UnsubscribeRgbdDepthImage();`
Overview	Unsubscribe from RGBD depth image data.
Note	Non-blocking interface, used to stop receiving RGBD depth image data.

8.1.21 SubscribeRgbdColorCameraInfo

Item	Description
Function Name	SubscribeRgbdColorCameraInfo
Declaration	`void SubscribeRgbdColorCameraInfo(const CameraInfoCallback callback);`
Overview	Subscribe to RGBD color image intrinsic data.
Parameters	callback: Callback function to handle received RGBD color image intrinsic data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.22 UnsubscribeRgbdColorCameraInfo

Item	Description
Function Name	UnsubscribeRgbdColorCameraInfo
Declaration	`void UnsubscribeRgbdColorCameraInfo();`
Overview	Unsubscribe from RGBD color image intrinsic data.
Note	Non-blocking interface, used to stop receiving RGBD color image intrinsic data.

8.1.23 SubscribeRgbdColorImage

Item	Description
Function Name	SubscribeRgbdColorImage
Declaration	`void SubscribeRgbdColorImage(const ImageCallback callback);`
Overview	Subscribe to RGBD color image data.
Parameters	callback: Callback function to handle received RGBD color image data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.24 UnsubscribeRgbdColorImage

Item	Description
Function Name	UnsubscribeRgbdColorImage
Declaration	`void UnsubscribeRgbdColorImage();`
Overview	Unsubscribe from RGBD color image data.
Note	Non-blocking interface, used to stop receiving RGBD color image data.

8.1.25 SubscribeImu

Item	Description
Function Name	SubscribeImu
Declaration	`void SubscribeImu(const ImuCallback callback);`
Overview	Subscribe to IMU data.
Parameters	callback: Callback function to handle received IMU data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.26 UnsubscribeImu

Item	Description
Function Name	UnsubscribeImu
Declaration	`void UnsubscribeImu();`
Overview	Unsubscribe from IMU data.
Note	Non-blocking interface, used to stop receiving IMU data.

8.1.27 SubscribeLeftBinocularHighImg

Item	Description
Function Name	SubscribeLeftBinocularHighImg
Declaration	`void SubscribeLeftBinocularHighImg(const CompressedImageCallback callback);`
Overview	Subscribe to left high-quality binocular data.
Parameters	callback: Callback function to handle received left high-quality binocular data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.28 UnsubscribeLeftBinocularHighImg

Item	Description
Function Name	UnsubscribeLeftBinocularHighImg
Declaration	`void UnsubscribeLeftBinocularHighImg();`
Overview	Unsubscribe from left high-quality binocular data.
Note	Non-blocking interface, used to stop receiving left high-quality binocular data.

8.1.29 SubscribeLeftBinocularLowImg

Item	Description
Function Name	SubscribeLeftBinocularLowImg
Declaration	`void SubscribeLeftBinocularLowImg(const CompressedImageCallback callback);`
Overview	Subscribe to left low-quality binocular data.
Parameters	callback: Callback function to handle received left low-quality binocular data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.30 UnsubscribeLeftBinocularLowImg

Item	Description
Function Name	UnsubscribeLeftBinocularLowImg
Declaration	`void UnsubscribeLeftBinocularLowImg();`
Overview	Unsubscribe from left low-quality binocular data.
Note	Non-blocking interface, used to stop receiving left low-quality binocular data.

8.1.31 SubscribeRightBinocularLowImg

Item	Description
Function Name	SubscribeRightBinocularLowImg
Declaration	`void SubscribeRightBinocularLowImg(const CompressedImageCallback callback);`
Overview	Subscribe to right low-quality binocular data.
Parameters	callback: Callback function to handle received right low-quality binocular data.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.32 UnsubscribeRightBinocularLowImg

Item	Description
Function Name	UnsubscribeRightBinocularLowImg
Declaration	`void UnsubscribeRightBinocularLowImg();`
Overview	Unsubscribe from right low-quality binocular data.
Note	Non-blocking interface, used to stop receiving right low-quality binocular data.

8.1.33 SubscribeDepthImage

Item	Description
Function Name	SubscribeDepthImage
Declaration	`void SubscribeDepthImage(const ImageCallback callback);`
Overview	Subscribe to depth image.
Parameters	callback: Callback function to handle received depth image.
Note	Non-blocking interface, callback will be invoked when data updates.

8.1.34 UnsubscribeDepthImage

Item	Description
Function Name	UnsubscribeDepthImage
Declaration	`void UnsubscribeDepthImage();`
Overview	Unsubscribe from depth image.
Note	Non-blocking interface, used to stop receiving depth image data.

8.2 Type Definitions

8.2.1 Imu —IMU Data Structure

Field Name	Type	Description
`timestamp`	`int64_t`	Timestamp (unit: nanoseconds)
`orientation[4]`	`double[4]`	Orientation quaternion (w, x, y, z)
`angular_velocity[3]`	`double[3]`	Angular velocity (unit: rad/s)
`linear_acceleration[3]`	`double[3]`	Linear acceleration (unit: m/s^2)
`temperature`	`float`	Temperature (unit: Celsius or other, should be specified)

8.2.2 Header —Common Message Header Structure

Field Name	Type	Description
`stamp`	`int64_t`	Timestamp (unit: nanoseconds)
`frame_id`	`std::string`	Coordinate frame name

8.2.3 PointField —Point Field Description

Field Name	Type	Description
`name`	`std::string`	Field name (e.g., x, y, z, intensity)
`offset`	`int32_t`	Starting byte offset
`datatype`	`int8_t`	Data type (corresponding constant)
`count`	`int32_t`	Number of elements per point for this field

8.2.4 PointCloud2 —Point Cloud Data Structure

Field Name	Type	Description
header	Header	Message header
height	int32_t	Number of rows
width	int32_t	Number of columns
fields	std::vector<PointField>	Array of point fields
is_bigendian	bool	Byte order
point_step	int32_t	Bytes per point
row_step	int32_t	Bytes per row
data	std::vector<uint8_t>	Raw point cloud byte data
is_dense	bool	Is dense point cloud (no invalid points)

8.2.5 Image —Image Data Structure

Field Name	Type	Description
header	Header	Message header
height	int32_t	Image height (pixels)
width	int32_t	Image width (pixels)
encoding	std::string	Encoding type (e.g., rgb8, mono8)
is_bigendian	bool	Is big-endian mode
step	int32_t	Bytes per image row
data	std::vector<uint8_t>	Raw image byte data

8.2.6 CameraInfo —Camera Intrinsic and Distortion Information

Field Name	Type	Description
header	Header	Message header
height	int32_t	Image height
width	int32_t	Image width
distortion_model	std::string	Distortion model (e.g., plumb_bob)
D	std::vector<double>	Distortion parameter array
K	double[9]	Camera intrinsic matrix
R	double[9]	Rectification matrix
P	double[12]	Projection matrix
binning_x	int32_t	Horizontal binning factor
binning_y	int32_t	Vertical binning factor
roi_x_offset	int32_t	ROI starting x coordinate
roi_y_offset	int32_t	ROI starting y coordinate
roi_height	int32_t	ROI height
roi_width	int32_t	ROI width
roi_do_rectify	bool	Whether to rectify

8.2.7 CompressedImage —Compressed Image Data Structure

Field Name	Type	Description
header	Header	Message header
format	std::string	Compression format (e.g., jpeg, png)
data	std::vector<uint8_t>	Compressed image data

8.2.8 LaserScan —Lidar Data Structure

Field Name	Type	Description
header	Header	Message header
angle_min	int32_t	Start angle (radians)
angle_max	int32_t	End angle (radians)
angle_increment	int32_t	Angle increment (radians)
time_increment	int32_t	Time increment (seconds)
scan_time	int32_t	Scan time (seconds)
range_min	int32_t	Minimum range (meters)
range_max	int32_t	Maximum range (meters)
ranges	std::vector<double>	Range data array (meters)
intensities	std::vector<double>	Intensity data array (unitless)

8.2.9 MultiArrayDimension —Multi-dimensional Array Dimension Description

Field Name	Type	Description
label	std::string	Dimension label
size	int32_t	Dimension size
stride	int32_t	Stride

8.2.10 MultiArrayLayout —Multi-dimensional Array Layout Description

Field Name	Type	Description
dim_size	int32_t	Number of dimensions
dim	std::vector<MultiArrayDimension>	Array of dimensions
data_offset	int32_t	Data offset

8.2.11 Float32MultiArray —Float Multi-dimensional Array

Field Name	Type	Description
layout	MultiArrayLayout	Array layout information
data	std::vector<float>	Float data array

8.2.12 Int8 —8-bit Integer Data Structure

Field Name	Type	Description
`data`	`int8_t`	8-bit integer data

8.2.13 HeadTouch —Head Touch Data Structure

Item	Description
Type	`Int8`
Description	Head touch sensor data, using 8-bit integer format

8.3 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

NINE

AUDIO CONTROL SERVICE (C++)

Provides a voice service controller for the robot system. Through AudioController, you can use RPC to control robot audio commands and obtain status.

9.1 Interface Definition

AudioController is a C++ class encapsulating audio control functions, mainly used for audio playback control, TTS playback, volume setting and query, and subscribing to raw voice data.

9.1.1 AudioController

Item	Description
Function Name	AudioController
Declaration	`AudioController();`
Overview	Initializes the audio controller object, constructs internal state, allocates resources, etc.
Note	Constructs internal state.

9.1.2 ~AudioController

Item	Description
Function Name	~AudioController
Declaration	`~AudioController();`
Overview	Releases audio controller resources, ensures playback is stopped and underlying resources are cleaned up.
Note	Ensures resources are safely released.

9.1.3 Initialize

Item	Description
Function Name	Initialize
Declaration	`bool Initialize();`
Overview	Initializes the audio control module, prepares playback resources and devices.
Return Value	`true` for success, `false` for failure.
Note	Used in pair with `Shutdown()`.

9.1.4 Shutdown

Item	Description
Function Name	Shutdown
Declaration	`void Shutdown();`
Overview	Shuts down the audio controller and releases resources.
Note	Be sure to call before destruction.

9.1.5 GetVoiceConfig

Item	Description
Function Name	GetVoiceConfig
Declaration	`Status GetVoiceConfig(GetSpeechConfig& config, int timeout_ms);`
Overview	Gets the complete configuration information of the voice system.
Parameters	`config`: Returns the voice system configuration by reference.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. The configuration includes all sub-configs such as speaker, bot, wakeup, dialog, and the current TTS model type.

9.1.6 SwitchTtsVoiceModel

Item	Description
Function Name	SwitchTtsVoiceModel
Declaration	`Status SwitchTtsVoiceModel(TtsType tts_type, int timeout_ms);`
Overview	Switches the TTS (Text-to-Speech) voice model.
Parameters	`tts_type`: Type of TTS voice model`timeout_ms`: Timeout time (milliseconds), default is 5000.
Return Value	Operation status.
Note	Model switching is a blocking operation.

9.1.7 SetVoiceConfig

Item	Description
Function Name	SetVoiceConfig
Declaration	`Status SetVoiceConfig(const SetSpeechConfig& config, int timeout_ms);`
Overview	Sets the complete configuration information of the voice system.
Parameters	`config`: The voice system configuration to set.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. The set configuration will completely overwrite all current voice-related configurations.

9.1.8 Play

Item	Description
Function Name	Play
Declaration	`Status Play(const TtsCommand& cmd, int timeout_ms);`
Overview	Plays a TTS (Text-to-Speech) command.
Parameters	`cmd`: TTS command, including text, speed, pitch, etc.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Make sure the module is initialized before calling.

9.1.9 Stop

Item	Description
Function Name	Stop
Declaration	`Status Stop(int timeout_ms);`
Overview	Stops the current audio playback.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Usually used to interrupt current speech.

9.1.10 SetVolume

Item	Description
Function Name	SetVolume
Declaration	`Status SetVolume(int volume, int timeout_ms);`
Overview	Sets the audio output volume.
Parameters	`volume`: Volume value, usually in the range 0~100.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Takes effect immediately after setting.

9.1.11 GetVolume

Item	Description
Function Name	GetVolume
Declaration	`Status GetVolume(int& volume, int timeout_ms);`
Overview	Gets the current audio output volume.
Parameters	`volume`: Returns the current volume value by reference.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Check the return value before using `volume`.

9.1.12 ControlVoiceStream

Item	Description
Function Name	ControlVoiceStream
Declaration	`Status ControlVoiceStream(bool enable_raw_data, bool enable_bf_data, int timeout_ms);`
Overview	Controls the voice data stream.
Parameters	`enable_raw_data`: Whether to enable the original voice data stream. `enable_bf_data`: Whether to enable the BF voice data stream. `timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Operation status. `Status::OK` for success, others for failure.
Note	Blocking interface, used to control enabling and disabling of voice data streams.

9.1.13 SubscribeOriginVoiceData

Item	Description
Function Name	SubscribeOriginVoiceData
Declaration	`void SubscribeOriginVoiceData(const ByteMultiArrayCallback callback);`
Overview	Subscribes to raw voice data.
Parameters	callback: Callback to handle received raw voice data.
Note	Non-blocking interface. The callback will be called when data is updated.

9.1.14 UnsubscribeOriginVoiceData

Item	Description
Function Name	UnsubscribeOriginVoiceData
Declaration	`void UnsubscribeOriginVoiceData();`
Overview	Unsubscribes from raw voice data.
Note	Non-blocking interface. Used to stop receiving raw voice data.

9.1.15 SubscribeBfVoiceData

Item	Description
Function Name	SubscribeBfVoiceData
Declaration	`void SubscribeBfVoiceData(const ByteMultiArrayCallback callback);`
Overview	Subscribes to BF voice data.
Parameters	callback: Callback to handle received BF voice data.
Note	Non-blocking interface. The callback will be called when data is updated.

9.1.16 UnsubscribeBfVoiceData

Item	Description
Function Name	UnsubscribeBfVoiceData
Declaration	`void UnsubscribeBfVoiceData();`
Overview	Unsubscribes from BF voice data.
Note	Non-blocking interface. Used to stop receiving BF voice data.

9.1.17 ControlSpeechIO

Item	Content
Function Name	ControlSpeechIO
Function Declaration	`Status ControlSpeechIO(bool enable_data, int timeout_ms);`
Function Overview	Control speech io data stream(asr and tts).
Parameter Description	`enable_data`: Turn the voice io data stream on or off. `timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to control speech io data stream(asr and tts).

9.1.18 SubscribeSpeechASRStream

Item	Content
Function Name	SubscribeSpeechASRStream
Function Declaration	void SubscribeSpeechASRStream(const SpeechASRStreamCallback callback);
Function Overview	Subscribe to speech asr stream.
Parameter Description	`callback`: Callback to handle received speech asr stream.
Notes	Blocking interface, used to subscribe to speech asr stream.

9.1.19 UnsubscribeSpeechASRStream

Item	Content
Function Name	UnsubscribeSpeechASRStream
Function Declaration	void UnsubscribeSpeechASRStream();
Function Overview	Unsubscribe from speech asr stream.
Notes	Blocking interface, used to unsubscribe from speech asr stream.

9.1.20 PublishSpeechTTSStream

Item	Content
Function Name	PublishSpeechTTSStream
Function Declaration	`Status PublishSpeechTTSStream(const SpeechTTSStream& data);`
Function Overview	Publish speech tts stream.
Parameter Description	`data`: Speech TTS stream data.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to publish speech tts stream. The data ID should be consistent with the request ID from the ASR output, with the start type of begin, the middle data type of var and the end type of end.

9.2 Type Definitions

9.2.1 TtsPriority —TTS Playback Priority Level

Used to control interruption behavior between different TTS tasks. Higher priority tasks will interrupt the playback of current lower priority tasks.

Enum Value	Value	Description
`TtsPriority::HIGH`	0	Highest priority, e.g., low battery alert, emergency reminder
`TtsPriority::MIDDLE`	1	Medium priority, e.g., system prompt, status broadcast
`TtsPriority::LOW`	2	Lowest priority, e.g., daily voice dialogue, background broadcast

9.2.2 TtsMode —Task Scheduling Policy at the Same Priority

Used to refine the playback order and clearing logic of multiple TTS tasks under the same priority.

Enum Value	Value	Description
`TtsMode::CLEARTOP`	0	Clear all tasks of the current priority (including playing and waiting queue), play this request immediately
`TtsMode::ADD`	1	Add this request to the end of the current priority queue, play in order (do not interrupt current playback)
`TtsMode::CLEARBUF`	2	Clear unplayed requests in the queue, keep current playback, then play this request

9.3 Struct Definitions

9.3.1 TtsCommand —TTS Playback Command Structure

Describes the complete information of a TTS playback request, supporting unique ID, text content, priority control, and scheduling mode under the same priority.

Field Name	Type	Description
`id`	`std::string`	Unique TTS task ID, e.g., `"id_01"`, used to track playback status
`content`	`std::string`	Text content to play, e.g., `"Hello, welcome to the intelligent voice system."`
`priority`	`TtsPriority`	Playback priority, controls whether to interrupt lower priority speech
`mode`	`TtsMode`	Scheduling policy under the same priority, controls whether to append, overwrite, etc.

9.3.2 CustomBotInfo —Custom Bot Configuration Structure

Describes the basic configuration information of a custom bot.

Field Name	Type	Description
`name`	`std::string`	Bot name
`workflow`	`std::string`	Workflow ID
`token`	`std::string`	User authorization token

9.3.3 CustomBotMap —Custom Bot Mapping Table

Item	Description
Type	`std::map<std::string, CustomBotInfo>`
Description	Custom bot mapping table, key is bot ID, value is bot configuration info

9.3.4 SetSpeechConfig —Speech Configuration Parameter Structure

Used to set various configuration parameters of the voice system.

Field Name	Type	Description
`speaker_id`	`std::string`	Speaker ID
`region`	`std::string`	Speaker region
`bot_id`	`std::string`	Mode ID
`is_front_doa`	`bool`	Force recognition from the front
`is_fullduplex_enable`	`bool`	Natural conversation switch
`is_enable`	`bool`	Voice switch
`is_doa_enable`	`bool`	Enable wakeup direction turning
`speaker_speed`	`double`	TTS playback speed, range [1,2]
`wakeup_name`	`std::string`	Wakeup name
`custom_bot`	`CustomBotMap`	Custom bot configuration

9.3.5 SpeakerConfigSelected —Selected Speaker Configuration Structure

Current speaker configuration

Field Name	Type	Description
`region`	`std::string`	Selected region
`speaker_id`	`std::string`	Selected speaker ID

9.3.6 SpeakerConfig —Speaker Configuration Structure

Speaker configuration

Field Name	Type	Description
data	std::map<std::string, std::vector< std::array<std::string, 2>>>	Speaker data: region->speaker ID->speaker name
selected	SpeakerConfigSelected	Currently selected speaker configuration
speaker_s	double	Speech speed

9.3.7 BotInfo —Bot Configuration Information Structure

Describes the basic information of a standard bot.

Field Name	Type	Description
name	std::string	Work scenario name
workflow	std::string	Workflow ID

BotConfigSelected —Selected Bot Structure

Field Name	Type	Description
bot_id	std::string	Selected bot ID

9.3.8 BotConfig —Bot Configuration Structure

Describes bot-related configuration information, including standard bots and custom bots.

Field Name	Type	Description
data	std::map<std::string, BotInfo>	Standard bot data: bot ID->bot info
custom_data	std::map<std::string, CustomBotInfo>	Custom bot data: bot ID->custom bot info
selected	BotConfigSelected	Currently selected bot configuration

9.3.9 WakeupConfig —Wakeup Configuration Structure

Describes wakeup-related configuration information.

Field Name	Type	Description
`name`	`std::string`	Wakeup name
`data`	`std::map<std::string, std::string>`	Wakeup word data: wakeup word->pinyin

9.3.10 DialogConfig —Dialog Configuration Structure

Describes dialog-related configuration parameters.

Field Name	Type	Description
`is_front_doa`	`bool`	Force enhanced pickup from the front
`is_fullduplex_enable`	`bool`	Enable full-duplex dialog
`is_enable`	`bool`	Enable voice
`is_doa_enable`	`bool`	Enable wakeup direction turning

9.3.11 TtsType —TTS Model Enum

Describes the available TTS model types.

Enum Value	Value	Description
`TtsType::NONE`	0	No TTS model
`TtsType::DOUBAO`	1	Doubao TTS model
`TtsType::GOOGLE`	2	Google TTS model

9.3.12 GetSpeechConfig —Complete Voice System Configuration Structure

Describes the complete configuration information of the voice system, including all sub-configuration modules.

Field Name	Type	Description
`speaker_config`	`SpeakerConfig`	Speaker configuration
`bot_config`	`BotConfig`	Bot configuration
`wakeup_config`	`WakeupConfig`	Wakeup configuration
`dialog_config`	`DialogConfig`	Dialog configuration
`tts_type`	`TtsType`	TTS model, default is `TtsType::NONE`

9.3.13 MultiArrayDimension —Multi-dimensional Array Dimension Description

Field Name	Type	Description
label	std::string	Dimension label
size	int32_t	Dimension size
stride	int32_t	Stride

9.3.14 MultiArrayLayout —Multi-dimensional Array Layout Description

Field Name	Type	Description
dim_size	int32_t	Number of dimensions
dim	std::vector<MultiArrayDimension>	Dimension array
data_offset	int32_t	Data offset

9.3.15 ByteMultiArray —Byte Array Data Structure

Field Name	Type	Description
layout	MultiArrayLayout	Array layout information
data	std::vector<uint8_t>	Byte data array

9.3.16 SpeechASRStream —Robot ASR output

Field Name	Type	Description
id	std::string	ID - request id
type	std::string	Type - request or cancel
text	std::string	Text - asr output

9.3.17 SpeechTTSStream —Robot TTS input

Field Name	Type	Description
id	std::string	ID - same as request id
type	std::string	Type - begin or var or end
text	std::string	Text - tts input
end_session	bool	End Session - end session and close asr

9.4 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

TEN

STATE MONITOR SERVICE (C++)

Provides a robot state monitoring interface class. Through StateMonitor, you can perform state queries and related functions.

10.1 Interface Definition

StateMonitor is the robot system state monitoring management class. This class is typically used for managing the robot's state and supports state queries.

10.1.1 GetCurrentState

Item	Description
Function Name	GetCurrentState
Function Declaration	`Status GetCurrentState(RobotState& robot_state);`
Overview	Get the current aggregated robot state.
Parameter Description	`robot_state`: Used to receive the state struct.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Non-blocking interface. Retrieves the most recently monitored robot state data, including BMS battery status, fault information, etc. More features will be added in the future.

10.1.2 RobotState —Robot State Data Structure

Represents the overall state information of the robot:

Field Name	Type	Description
faults	std::vector<Fault>	List of fault information
bms_data	BmsData	Battery Management System data

10.1.3 BmsData —Battery Management System Data Structure

Represents the battery status information:

Field Name	Type	Description
battery_percentage	double	Current battery remaining percentage (0~100)
battery_health	double	Battery health status (higher value means better health)
battery_state	BatteryState	Battery state (see enum below)
power_supply_status	PowerSupplyStatus	Battery charge/discharge status (see enum below)

10.1.4 Enum Type Definitions

BatteryState —Battery State Enum

Represents the current state of the battery, used for determining and handling battery status in the system:

Enum Value	Value	Description
BatteryState::UNKNOWN	0	Unknown state
BatteryState::GOOD	1	Battery is in good condition
BatteryState::OVERHEAT	2	Battery overheated
BatteryState::DEAD	3	Battery damaged
BatteryState::OVERVOLTAGE	4	Battery overvoltage
BatteryState::UNSPEC_FAILURE	5	Unknown failure
BatteryState::COLD	6	Battery too cold
BatteryState::WATCHDOG_TIMER_EXPIRE	7	Watchdog timer expired
BatteryState::SAFETY_TIMER_EXPIRE	8	Safety timer expired

PowerSupplyStatus —Battery Charge/Discharge Status

Represents the current charge/discharge status of the battery:

Enum Value	Value	Description
`PowerSupplyStatus::UNKNOWN`	0	Unknown status
`PowerSupplyStatus::CHARGING`	1	Battery charging
`PowerSupplyStatus::DISCHARGING`	2	Battery discharging
`PowerSupplyStatus::NOTCHARGING`	3	Battery not charging/discharging
`PowerSupplyStatus::FULL`	4	Battery fully charged

10.2 Error Code Mapping Table

Error Code (Hex)	Error Description
0x0000	No error
0x1101	Failed to call ROS service
0x1301	Main control node lost
0x1302	APP node lost
0x1304	Audio node lost
0x1305	Motion node lost
0x1306	LCD node lost
0x1307	Realsense node lost
0x1308	Stereo camera node lost
0x1309	LDS node lost
0x130A	Sensor board node lost
0x130B	Touch node lost
0x130C	SLAM node lost
0x130D	Navigation node lost
0x130E	AI node lost
0x130F	Head node lost
0x1310	Cloud processor node lost
0x3201	No laser data
0x3202	No stereo camera data
0x3203	Stereo camera data error
0x3204	Stereo camera initialization failed
0x320B	No odometry data
0x320C	No IMU data
0x6101	Robot failed to connect to APP
0x6102	Disconnected from APP

continues on next page

Table 1 – continued from previous page

Error Code (Hex)	Error Description
0x9201	Failed to open LCD serial port
0x7201	Failed to open head serial port
0x7202	No head data
0x8201	Failed to open sensor board serial port
0x8202	No sensor board data
0x2201	Error: Navigation did not receive tf data
0x2202	Error: Navigation did not receive map data
0x2203	Error: Navigation did not receive localization data
0x2204	Error: Navigation did not receive ultrasonic data
0x2205	Error: Navigation did not receive laser data
0x2206	Error: Navigation did not receive RGBD data
0x2207	Error: Navigation did not receive multi-line laser data
0x2208	Error: Navigation did not receive point TOF data
0x2209	Error: Navigation did not receive surface TOF data
0x220A	Error: Navigation did not receive odometry data
0x2101	Warning: Navigation did not receive tf data
0x2102	Warning: Navigation did not receive map data
0x2103	Warning: Navigation did not receive localization data
0x2104	Warning: Navigation did not receive ultrasonic data
0x2105	Warning: Navigation did not receive laser data
0x2106	Warning: Navigation did not receive RGBD TOF data
0x2107	Warning: Navigation did not receive multi-line laser data
0x2108	Warning: Navigation did not receive point TOF data
0x2109	Warning: Navigation did not receive surface TOF data
0x210A	Warning: Navigation did not receive odometry data
0x4201	SLAM localization error
0x4102	Error: SLAM did not receive laser data
0x4103	Error: SLAM did not receive odometry data
0x4205	SLAM map error

CHAPTER

ELEVEN

SLAM NAVIGATION CONTROL SERVICE (C++)

Provides SLAM navigation control services for robot systems. Through SlamNavController, SLAM mapping, localization, navigation and other functions can be implemented.

11.1 Interface Definition

SlamNavController is a controller for SLAM navigation control, supporting mapping, localization, navigation and other operations.

11.1.1 SlamNavController

Item	Content
Function Name	SlamNavController
Function Declaration	`SlamNavController();`
Function Overview	Constructor, creates SlamNavController instance.
Remarks	Constructs internal control resources.

11.1.2 ~SlamNavController

Item	Content
Function Name	~SlamNavController
Function Declaration	`~SlamNavController();`
Function Overview	Destructor, releases SlamNavController instance resources.
Remarks	Used in conjunction with constructor.

11.1.3 Initialize

Item	Content
Function Name	Initialize
Function Declaration	`bool Initialize();`
Function Overview	Initializes SLAM navigation controller.
Return Value	`true` indicates success, `false` indicates failure.
Remarks	Must be called before first use.

11.1.4 Shutdown

Item	Content
Function Name	Shutdown
Function Declaration	`void Shutdown();`
Function Overview	Releases resources and cleans up SLAM navigation controller.
Remarks	Used in conjunction with Initialize.

11.1.5 SwitchToIdle

Item	Content
Function Name	SwitchToIdle
Function Declaration	`Status SwitchToIdle();`
Function Overview	Switches to idle mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to switch to idle state.

11.1.6 StartMapping

Item	Content
Function Name	StartMapping
Function Declaration	`Status StartMapping();`
Function Overview	Starts mapping mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to start mapping task.

11.1.7 CancelMapping

Item	Content
Function Name	CancelMapping
Function Declaration	`Status CancelMapping();`
Function Overview	Cancels current mapping task.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to cancel mapping task.

11.1.8 SwitchToLocation

Item	Content
Function Name	SwitchToLocation
Function Declaration	`Status SwitchToLocation();`
Function Overview	Switches to localization mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to switch to localization state.

11.1.9 SaveMap

Item	Content
Function Name	SaveMap
Function Declaration	`Status SaveMap(const std::string& map_name);`
Function Overview	Ends mapping and saves map when in mapping mode.
Parameter Description	`map_name`: Map name.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to save current mapping result.

11.1.10 LoadMap

Item	Content
Function Name	LoadMap
Function Declaration	`Status LoadMap(const std::string& map_name);`
Function Overview	Loads map and sets it as current map.
Parameter Description	`map_name`: Map name.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to load saved map.

11.1.11 DeleteMap

Item	Content
Function Name	DeleteMap
Function Declaration	`Status DeleteMap(const std::string& map_name);`
Function Overview	Deletes map.
Parameter Description	`map_name`: Name of map to delete.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to delete unwanted map.

11.1.12 GetAllMapInfo

Item	Content
Function Name	GetAllMapInfo
Function Declaration	`Status GetAllMapInfo(AllMapInfo& all_map_info);`
Function Overview	Gets all map information.
Parameter Description	`all_map_info`: All map information (output parameter).
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to get list of all maps in the system.

11.1.13 InitPose

Item	Content
Function Name	InitPose
Function Declaration	`Status InitPose(const Pose3DEuler& pose);`
Function Overview	Initializes pose.
Parameter Description	`pose`: Pose information to publish.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to set robot initial pose.

11.1.14 GetCurrentLocalizationInfo

Item	Content
Function Name	GetCurrentLocalizationInfo
Function Declaration	`Status GetCurrentLocalizationInfo(LocalizationInfo& pose_info);`
Function Overview	Gets current pose information.
Parameter Description	`pose_info`: Current position and orientation information (output parameter).
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to get robot current pose.

11.1.15 ActivateNavMode

Item	Content
Function Name	ActivateNavMode
Function Declaration	`Status ActivateNavMode(NavMode mode);`
Function Overview	Activates navigation mode.
Parameter Description	`mode`: Target navigation mode (NavMode enumeration type).
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to switch navigation working mode.

11.1.16 SetNavTarget

Item	Content
Function Name	SetNavTarget
Function Declaration	`Status SetNavTarget(const NavTarget& goal);`
Function Overview	Sets global navigation target point and starts navigation task.
Parameter Description	`goal`: Global coordinates of target point.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to set navigation target.

11.1.17 PauseNavTask

Item	Content
Function Name	PauseNavTask
Function Declaration	`Status PauseNavTask();`
Function Overview	Pauses current navigation task.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to pause navigation.

11.1.18 ResumeNavTask

Item	Content
Function Name	ResumeNavTask
Function Declaration	`Status ResumeNavTask();`
Function Overview	Resumes paused navigation task.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to resume navigation.

11.1.19 CancelNavTask

Item	Content
Function Name	CancelNavTask
Function Declaration	`Status CancelNavTask();`
Function Overview	Cancels current navigation task.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to cancel navigation.

11.1.20 GetNavTaskStatus

Item	Content
Function Name	GetNavTaskStatus
Function Declaration	`/** * @brief Get the current navigation task status * @param status Navigation status * @return Operation status, returns Status::OK if successful */ Status GetNavTaskStatus(NavStatus& status);`
Function Overview	Retrieves the current navigation task status.
Parameter Description	`status`: Navigation status.
Return Value	Operation status, returns `Status::OK` if successful.
Remarks	Blocking interface, used to query the navigation task status.

11.1.21 SubscribeOdometry

Item	Content
Function Name	SubscribeOdometry
Function Declaration	`Status SubscribeOdometry(OdometryCallback callback);`
Function Overview	Subscribes to robot odometry data.
Parameter Description	`callback`: Odometry data callback function.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Non-blocking interface, used to get real-time odometry data.

11.1.22 UnsubscribeOdometry

Item	Content
Function Name	UnsubscribeOdometry
Function Declaration	`void UnsubscribeOdometry();`
Function Overview	Cancels subscription to robot odometry data.
Return Value	None
Remarks	`/** @brief Unsubscribe from odometry data */`

11.1.23 GetPointCloudMap

Item	Content
Function Name	GetPointCloudMap
Function Declaration	`Status GetPointCloudMap(PointCloud2&amp; point_cloud_map, int timeout_ms);`
Function Overview	Retrieves the current point cloud map.
Parameter Description	`point_cloud_map`: Output parameter, receives the acquired point cloud map data. `timeout_ms`: Operation timeout in milliseconds, default is 10000.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, returns an error status if not completed within the timeout period.

11.2 Type Definitions

11.2.1 NavMode —Navigation Mode Enumeration Type

Enum Value	Value	Description
`IDLE`	0	Idle mode
`GRID_MAP`	1	Grid map navigation mode

11.2.2 Pose3DEuler —3D Pose Structure

Field Name	Type	Description
`position`	`std::array<double, 3>`	Position (x, y, z), used to represent spatial position
`orientation`	`std::array<double, 3>`	Euler angles (roll, pitch, yaw), used to represent spatial orientation, avoiding Euler angle gimbal lock issues

11.2.3 NavTarget —Global Navigation Target Point Structure

Field Name	Type	Description
id	int32_t	Target point ID
frame_id	std::string	Target point coordinate frame ID
goal	Pose3DEuler	Target point pose

11.2.4 NavStatusType —Navigation Status Type Enumeration

Enum Value	Value	Description
NONE	0	No status
RUNNING	1	Running
END_SUCCESS	2	Successfully ended
END_FAILED	3	Failed to end
PAUSE	4	Paused
CONTINUE	5	Continue
CANCEL	6	Cancel

11.2.5 NavStatus —Navigation Status Structure

Field Name	Type	Description
id	int32_t	Target point ID, -1 indicates no target point
status	NavStatusType	Navigation status
message	std::string	Navigation status message

11.2.6 MapImageData —Mapping Image Data Structure, .pgm Format

Field Name	Type	Description
type	std::string	Magic number, “P5” : binary format
width	uint32_t	Image width
height	uint32_t	Image height
max_gray_value	uint32_t	Maximum gray value, 255
image	std::vector<uint8_t>	Image data

11.2.7 MapMetaData —Mapping Map Metadata Structure

Field Name	Type	Description
`resolution`	`double`	Map resolution, unit: m/pixel
`origin`	`Pose3DEuler`	Map origin, position of world coordinate system origin relative to bottom-left corner of map
`map_image_dat`	`MapImage-Data`	Image data, .pgm format image data

11.2.8 MapInfo —Single Map Information Structure

Field Name	Type	Description
`map_name`	`std::string`	Map name
`map_meta_data`	`MapMetaData`	Map metadata

11.2.9 AllMapInfo —All Map Information Structure

Field Name	Type	Description
`current_map_name`	`std::string`	Current map name
`map_infos`	`std::vector<MapInfo>`	All map information

11.2.10 LocalizationInfo —Current Position Information Structure

Field Name	Type	Description
`is_localization`	`bool`	Whether localized
`pose`	`Pose3DEuler`	Euler angle pose

11.3 SLAM Navigation Control Introduction

The robot’ s SLAM navigation control service can be divided into SLAM functions and navigation functions.

- In SLAM functions, corresponding interfaces can be called to implement mapping, localization, map management and other functions.
- In navigation functions, corresponding interfaces can be called to implement target navigation, navigation control and other functions.

11.3.1 Applicable Scenarios and Scope

- Static indoor flat areas smaller than 20 m × 20 m with rich features. Do not exceed the recommended range.
- This functionality is applicable to the education and research industry and is not recommended for commercial applications. For commercial use, please contact sales.

11.3.2 SLAM Function Flow

Function	Operation Flow
Mapping	Start mapping → Save map
Map Management	Load map, get map information, delete map, get map absolute path

11.3.3 Navigation Function Flow

Function	Operation Flow
Localization	Load map → Switch to localization mode → Initialize pose → Get localization status
Navigation	Localization success → Activate navigation mode → Set target pose (start navigation task) → Get navigation status
Navigation Control	Set target pose (start navigation task) → Pause navigation → Resume navigation → Cancel navigation

CHAPTER

TWELVE

DISPLAY CONTROL SERVICE (C++)

Provides a display service controller for the robot system. Through DisplayController, you can use RPC to control robot display commands and obtain status.

12.1 Interface Definition

DisplayController is a C++ class encapsulating display control functions, providing interfaces for display query, set, etc.

12.1.1 DisplayController

Item	Description
Function Name	DisplayController
Declaration	`DisplayController();`
Overview	Initializes the display controller object, constructs internal state, allocates resources, etc.
Note	Constructs internal state.

12.1.2 ~DisplayController

Item	Description
Function Name	~DisplayController
Declaration	`~DisplayController();`
Overview	Releases display controller resources.
Note	Ensures resources are safely released.

12.1.3 Initialize

Item	Description
Function Name	Initialize
Declaration	`bool Initialize();`
Overview	Initializes the display control module.
Return Value	`true` for success, `false` for failure.
Note	Used in pair with `Shutdown()`.

12.1.4 Shutdown

Item	Description
Function Name	Shutdown
Declaration	`void Shutdown();`
Overview	Shuts down the display controller and releases resources.
Note	Be sure to call before destruction.

12.1.5 GetAllFaceExpressions

Item	Description
Function Name	GetAllFaceExpressions
Declaration	`Status GetAllFaceExpressions(std::vector& data, int timeout_ms);`
Overview	Get all face expressions.
Parameters	`data`: Face expression data (returned by reference).`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Get all face expressions.

12.1.6 SetFaceExpression

Item	Description
Function Name	SetFaceExpression
Declaration	`Status SetFaceExpression(int expression_id, int timeout_ms);`
Overview	Set face expression.
Parameters	`expression_id`: Face expression ID.`timeout_ms`: Timeout time (milliseconds), default is 5000.
Return Value	Operation status.
Note	Blocking interface. Set face expression.

12.1.7 GetFaceExpression

Item	Description
Function Name	GetFaceExpression
Declaration	`Status GetFaceExpression(FaceExpression& data, int timeout_ms);`
Overview	Get current face expression.
Parameters	`data`: Current face expression (returned by reference).`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Get current face expression.

12.2 Struct Definitions

12.2.1 FaceExpression —Robot Face Expression Structure

Describes the complete information of a robot face expression, supporting unique ID, name, and description.

Field Name	Type	Description
`id`	`int32_t`	Robot face expression id.
`name`	`std::string`	Robot face expression name.
`description`	`std::string`	Robot face expression description.

12.3 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

THIRTEEN

ROBOT MAIN CONTROL SERVICE (PYTHON)

Provides the main controller for the robot system. Through MagicRobot, you can perform resource initialization, manage communication connections, and access various sub-controllers such as motion controllers, audio controllers, state monitors, and sensor controllers.

⚠ **Note: When using the Python interface, please ensure that the Python binding module is properly built, and pay attention to memory management and thread safety.**

13.1 Module Information

magicdog_python is the Python binding module for the MagicDog SDK, providing comprehensive robot control, sensor data acquisition, audio processing, and other functional interfaces.

Item	Content
Module Name	`magicdog_python`
Description	MagicDog SDK Python binding module
Dependencies	pybind11, C++ SDK

13.2 Interface Definition

MagicRobot is the unified entry class for the robot system.

13.2.1 MagicRobot —Main Robot Class

Item	Content
Class Name	`MagicRobot`
Function Overview	Main robot control class, integrates all functional modules
Main Functions	System initialization, connection management, controller access
Use Cases	Robot application development, system integration, functional testing

13.2.2 initialize

Item	Content
Function Name	`initialize`
Method Declaration	`bool initialize(const str& local_ip)`
Function Overview	Initializes the robot system, including controllers and network modules.
Parameter Description	`local_ip`: Local communication IP address.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

13.2.3 shutdown

Item	Content
Function Name	`shutdown`
Method Declaration	`void shutdown()`
Function Overview	Shuts down the robot system and releases resources.
Note	Used together with initialize.

13.2.4 release

Item	Content
Function Name	`release`
Method Declaration	`void release()`
Function Overview	Releases the robot system resources.
Note	Call this method to manually release all resources used by the SDK internally, usually before the program exits.

13.2.5 connect

Item	Content
Function Name	`connect`
Method Declaration	`Status connect(int timeout_ms)`
Function Overview	Establishes a communication connection with the robot service.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` indicates success.
Note	Blocking interface, should be called after initialization.

13.2.6 disconnect

Item	Content
Function Name	`disconnect`
Method Declaration	`Status disconnect(int timeout_ms)`
Function Overview	Disconnects from the robot service.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` indicates success.
Note	Blocking interface, used together with connect.

13.2.7 get_sdk_version

Item	Content
Function Name	`get_sdk_version`
Method Declaration	`str get_sdk_version()`
Function Overview	Gets the current SDK version.
Return Value	SDK version string (e.g., `0.0.1`).
Note	Non-blocking interface, useful for debugging or logging.

13.2.8 get_motion_control_level

Item	Content
Function Name	`get_motion_control_level`
Method Declaration	`ControllerLevel get_motion_control_level()`
Function Overview	Gets the current motion control level (high/low).
Return Value	`ControllerLevel` enum value.
Note	Non-blocking interface, used to determine current control permission.

13.2.9 set_motion_control_level

Item	Content
Function Name	`set_motion_control_level`
Method Declaration	`Status set_motion_control_level(ControllerLevel level)`
Function Overview	Sets the current motion control level.
Parameter Description	`level`: Control permission enum value.
Return Value	`Status::OK` indicates success, others indicate failure.
Note	Blocking interface, used when switching control modes.

13.2.10 get_high_level_motion_controller

Item	Content
Function Name	`get_high_level_motion_controller`
Method Declaration	`HighLevelMotionController get_high_level_motion_controller()`
Function Overview	Gets the high-level motion controller object.
Return Value	High-level motion controller instance, used to call high-level control interfaces.
Note	Non-blocking interface, encapsulates gait, stunt, remote control, and other functions.

13.2.11 get_low_level_motion_controller

Item	Content
Function Name	`get_low_level_motion_controller`
Method Declaration	`LowLevelMotionController get_low_level_motion_controller()`
Function Overview	Gets the low-level motion controller object.
Return Value	Low-level motion controller instance, used to control joints or motors.
Note	Non-blocking interface, can directly control joint motors, etc.

13.2.12 get_audio_controller

Item	Content
Function Name	`get_audio_controller`
Method Declaration	`AudioController get_audio_controller()`
Function Overview	Gets the audio controller object.
Return Value	Audio controller instance, used for voice control.
Note	Non-blocking interface, can play voice and control volume.

13.2.13 get_sensor_controller

Item	Content
Function Name	`get_sensor_controller`
Method Declaration	`SensorController get_sensor_controller()`
Function Overview	Gets the sensor controller object.
Return Value	Sensor controller instance, used to access sensor data.
Note	Non-blocking interface, encapsulates IMU, RGBD, and other sensor data reading.

13.2.14 get_state_monitor

Item	Content
Function Name	`get_state_monitor`
Method Declaration	`StateMonitor get_state_monitor()`
Function Overview	Gets the state monitor object.
Return Value	State monitor instance, used to get the current robot state information.
Note	Non-blocking interface, encapsulates BMS, fault, and other state information reading.

13.2.15 get_slam_nav_controller

Item	Content
Function Name	`get_slam_nav_controller`
Method Declaration	`SlamNavController get_slam_nav_controller()`
Function Overview	Gets the SLAM/Navigation controller object.
Return Value	SLAM/Navigation controller instance, used to access map, localization, and other SLAM-related data.
Note	Non-blocking interface, wraps usage for map building, localization, navigation and related features.

13.2.16 open_channel_switch

Item	Content
Function Name	`open_channel_switch`
Method Declaration	`Status open_channel_switch(int timeout_ms)`
Function Overview	Opens the LCM channel switch.
Parameter Description	`timeout_ms` (optional parameter, in milliseconds), operation timeout, default is 5000 ms.
Return Value	`Status` struct, indicating operation status.
Note	Used to control enabling of the SDK LCM communication channel.

13.2.17 close_channel_switch

Item	Content
Function Name	`close_channel_switch`
Method Declaration	`Status close_channel_switch(int timeout_ms)`
Function Overview	Closes the LCM channel switch.
Parameter Description	`timeout_ms` (optional parameter, in milliseconds), operation timeout, default is 5000 ms.
Return Value	`Status` struct, indicating operation status.
Note	Used to control disabling of the SDK LCM communication channel.

13.3 Constant Definitions

Constant Name	Type	Description
`LEG_JOINT_NUM`	`int`	Number of leg joints
`PERIOD_MS`	`int`	Control period (milliseconds)

13.4 Data Structure Definitions

13.4.1 Status —Status Return Structure

Field Name	Type	Description
`code`	`ErrorCode`	Error code
`message`	`str`	Status message

13.4.2 ErrorCode —Error Code Enum

Enum Value	Value	Description
`OK`	0	Operation successful
`SERVICE_NOT_READY`	1	Service not ready
`TIMEOUT`	2	Operation timed out
`INTERNAL_ERROR`	3	Internal error
`SERVICE_ERROR`	4	Service error

CHAPTER

FOURTEEN

HIGH-LEVEL MOTION CONTROL SERVICE (PYTHON)

Provides high-level motion control services for the robot system. Through the HighLevelMotionController, you can control the robot’s gait, tricks, and remote control via RPC communication.

⚠ **Notice:** Some trick actions, such as front flips and back flips, require large spaces. Extra attention is needed during execution. It is recommended to test all trick actions in large open spaces before using them in smaller areas.

14.1 Interface Definition

HighLevelMotionController is a high-level motion controller oriented towards semantic control. It supports operations such as walking and performing tricks, encapsulating low-level details for upper-layer system calls.

14.1.1 HighLevelMotionController —High-Level Motion Controller

Item	Description
Class Name	`HighLevelMotionController`
Overview	High-level motion controller for semantic control, supports walking, tricks, etc.
Main Functions	Gait control, trick execution, joystick command sending
Use Cases	High-level robot motion control, trick performance, remote operation

14.1.2 initialize

Item	Description
Function Name	`initialize`
Declaration	`bool initialize()`
Overview	Initialize the controller and prepare high-level control functions.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

14.1.3 shutdown

Item	Description
Function Name	`shutdown`
Declaration	`void shutdown()`
Overview	Shut down the controller and release resources.
Note	Use together with initialize.

14.1.4 set_gait

Item	Description
Function Name	`set_gait`
Declaration	`Status set_gait(GaitMode gait_mode, int timeout_ms)`
Overview	Set the robot’ s gait mode (e.g., position-controlled stand, force-controlled stand, trot, etc.).
Parameters	`gait_mode`: Gait control enum.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, supports switching between multiple gait modes.

14.1.5 get_gait

Item	Description
Function Name	`get_gait`
Declaration	`GaitMode get_gait(int timeout_ms)`
Overview	Get the robot’ s current gait mode (e.g., position-controlled stand, force-controlled stand, trot, etc.).
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Current gait mode, returns `GAIT_NONE` on failure.
Note	Blocking interface, gets the current gait mode.

14.1.6 execute_trick

Item	Description
Function Name	`execute_trick`
Declaration	`Status execute_trick(TrickAction trick_action, int timeout_msW)`
Overview	Execute a trick action (e.g., wiggle hip, lie down, etc.).
Parameters	`trick_action`: Trick action identifier.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, ensure the robot is in a state where tricks can be executed.

Note: Trick actions must be executed under GaitMode::GAIT_STAND_R(2) (position-controlled stand gait).

14.1.7 enable_joy_stick

Item	Description
Function Name	`enable_joy_stick`
Declaration	`Status enable_joy_stick()`
Overview	Enable joystick control commands.
Parameter Description	None.
Return Value	`Status.OK` indicates the operation was successful, otherwise failure.
Note	Used to allow sending real-time control commands via the joystick.

14.1.8 disable_joy_stick

Item	Description
Function Name	`disable_joy_stick`
Declaration	`Status disable_joy_stick()`
Overview	Disable joystick control commands.
Parameter Description	None.
Return Value	`Status.OK` indicates the operation was successful, otherwise failure.
Note	You must disable joystick control commands before switching to navigation mode.

14.1.9 send_joystick_command

Item	Description
Function Name	`send_joystick_command`
Declaration	`Status send_joystick_command(JoystickCommand command)`
Overview	Send real-time joystick control commands.
Parameters	`command`: Control data containing joystick coordinates.
Return Value	`Status::OK` for success, others for failure.
Note	Non-blocking interface, recommended sending frequency is 20Hz.

14.1.10 get_all_gait_speed_ratio

Item	Description
Function Name	`get_all_gait_speed_ratio`
Declaration	`AllGaitSpeedRatio get_all_gait_speed_ratio(int timeout_ms)`
Overview	Get all gaits and their corresponding forward, lateral, and rotational speed ratios.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Returns `AllGaitSpeedRatio` object on success, empty object on failure.
Note	Blocking interface, used to get speed ratio configuration for all gaits.

14.1.11 set_gait_speed_ratio

Item	Description
Function Name	`set_gait_speed_ratio`
Declaration	`Status set_gait_speed_ratio(GaitMode gait_mode, GaitSpeedRatio gait_speed_ratio, int timeout_ms)`
Overview	Set the speed ratio for a gait, including forward, lateral, and rotational speed ratios.
Parameters	`gait_mode`: Gait mode; `gait_speed_ratio`: Speed ratio configuration.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used to set speed ratio configuration for a specific gait.

14.1.12 get_head_motor_enabled

Item	Description
Function Name	`get_head_motor_enabled`
Declaration	`bool get_head_motor_enabled(int timeout_ms)`
Overview	Get the enable status of the head motor.
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Returns `true` (enabled) or `false` (disabled) on success, `false` on failure.
Note	Blocking interface, used to query the enable status of the head motor.

14.1.13 enable_head_motor

Item	Description
Function Name	`enable_head_motor`
Declaration	`Status enable_head_motor(int timeout_ms)`
Overview	Enable the head motor.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used to enable head motor control.

14.1.14 disable_head_motor

Item	Description
Function Name	`disable_head_motor`
Declaration	`Status disable_head_motor(int timeout_ms)`
Overview	Disable the head motor.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used to disable head motor control.

14.1.15 get_current_head_position

Item	Content
Function Name	get_current_head_position
Function Declaration	`Status get_current_head_position(EulerAngles euler_angles, int timeout_ms);`
Function Overview	Get current head position.
Parameter Description	`euler_angles`: Return parameter, current absolute position of head.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to get current head position.

14.1.16 set_head_position

Item	Content
Function Name	set_head_position
Function Declaration	`Status set_head_position(EulerAngles euler_angles, int timeout_ms);`
Function Overview	Set the head position.
Parameter Description	`euler_angles`: Absolute position of head target.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to set the head position.

14.2 Enum Type Definitions

14.2.1 GaitMode —Gait Mode Enum

Enum Value	Value	Description
GAIT_PASSIVE	0	Drop (disable motors)
GAIT_STAND_R	2	Position-controlled stand, RecoveryStand
GAIT_STAND_B	3	Force-controlled stand, pose display, BalanceStand
GAIT_RUN_FAST	8	Fast run
GAIT_DOWN_CLIMB_STAIRS	9	Downstairs => blind walk => trot
GAIT_TROT	10	Trot
GAIT_PRONK	11	Jump
GAIT_BOUND	12	Front-back jump
GAIT_AMBLE	14	Amble
GAIT_CRAWL	29	Crawl
GAIT_LOWLEVL_SDK	30	Low-level SDK gait
GAIT_WALK	39	Slow walk
GAIT_UP_CLIMB_STAIRS	56	Upstairs (all-terrain)
GAIT_RL_TERRAIN	110	All-terrain
GAIT_RL_FALL_RECOVERY	111	Fall recovery
GAIT_RL_HAND_STAND	112	Handstand
GAIT_RL_FOOT_STAND	113	Upright
GAIT_ENTER_RL	1001	Enter RL
GAIT_DEFAULT	99	Default
GAIT_NONE	9999	No gait

14.2.2 TrickAction —Trick Action Enum

Enum Value	Value	Description
ACTION_NONE	0	No action
ACTION_WIGGLE_HIP	26	Wiggle hip
ACTION_SWING_BODY	27	Swing body
ACTION_STRETCH	28	Stretch
ACTION_STOMP	29	Stomp
ACTION_JUMP_JACK	30	Jumping jack
ACTION_SPACE_WALK	31	Moonwalk
ACTION_IMITATE	32	Imitate
ACTION_SHAKE_HEAD	33	Shake head

continues on next page

Table 1 – continued from previous page

Enum Value	Value	Description
ACTION_PUSH_UP	34	Push-up
ACTION_CHEER_UP	35	Cheer up
ACTION_HIGH_FIVES	36	High fives
ACTION_SCRATCH	37	Scratch
ACTION_HIGH_JUMP	38	High jump
ACTION_SWING_DANCE	39	Swing dance
ACTION_LEAP_FROG	40	Leap frog
ACTION_BACK_FLIP	41	Back flip
ACTION_FRONT_FLIP	42	Front flip
ACTION_SPIN_JUMP_LEFT	43	Spin jump left 70°
ACTION_SPIN_JUMP_RIGHT	44	Spin jump right 70°
ACTION_JUMP_FRONT	45	Jump forward 0.5m
ACTION_ACT_CUTE	46	Act cute
ACTION_BOXING	47	Boxing
ACTION_SIDE_SOMERSAULT	48	Side somersault
ACTION_RANDOM_DANCE	49	Random dance
ACTION_LEFT_SIDE_SOMERSAULT	84	Left side somersault
ACTION_RIGHT_SIDE_SOMERSAULT	85	Right side somersault
ACTION_DANCE2	91	Dance 2
ACTION_EMERGENCY_STOP	101	Emergency stop
ACTION_LIE_DOWN	102	Lie down
ACTION_RECOVERY_STAND	103	Stand up
ACTION_HAPPY_NEW_YEAR	105	New Year greeting (bow)
ACTION_SLOW_GO_FRONT	108	Come here
ACTION_SLOW_GO_BACK	109	Go back
ACTION_BACK_HOME	110	Go home
ACTION_LEAVE_HOME	111	Leave home
ACTION_TURN_AROUND	112	Turn around
ACTION_DANCE	115	Dance
ACTION_ROLL_ABOUT	116	Roll about
ACTION_SHAKE_RIGHT_HAND	117	Shake right hand
ACTION_SHAKE_LEFT_HAND	118	Shake left hand
ACTION_SIT_DOWN	119	Sit down

14.3 Data Structure Definitions

14.3.1 GaitSpeedRatio —Gait Speed Ratio Structure

Field Name	Type	Description
`straight_ratio`	`double`	Forward speed ratio
`turn_ratio`	`double`	Rotational speed ratio
`lateral_ratio`	`double`	Lateral speed ratio

14.3.2 AllGaitSpeedRatio —All Gait Speed Ratio Structure

Field Name	Type	Description
`gait_speed_ratios`	`GaitModeGaitSpeedRatioMap`	Mapping from gait mode to speed ratio

14.3.3 GaitModeGaitSpeedRatioMap —Gait Mode to Speed Ratio Mapping Type

Item	Description
Type Name	`GaitModeGaitSpeedRatioMap`
Type Definition	`std::map<GaitMode, GaitSpeedRatio>`
Overview	Mapping container from gait mode to speed ratio configuration
Key Type	`GaitMode` - Gait mode enum
Value Type	`GaitSpeedRatio` - Speed ratio structure
Use Cases	Store and manage speed ratio configurations for all gaits

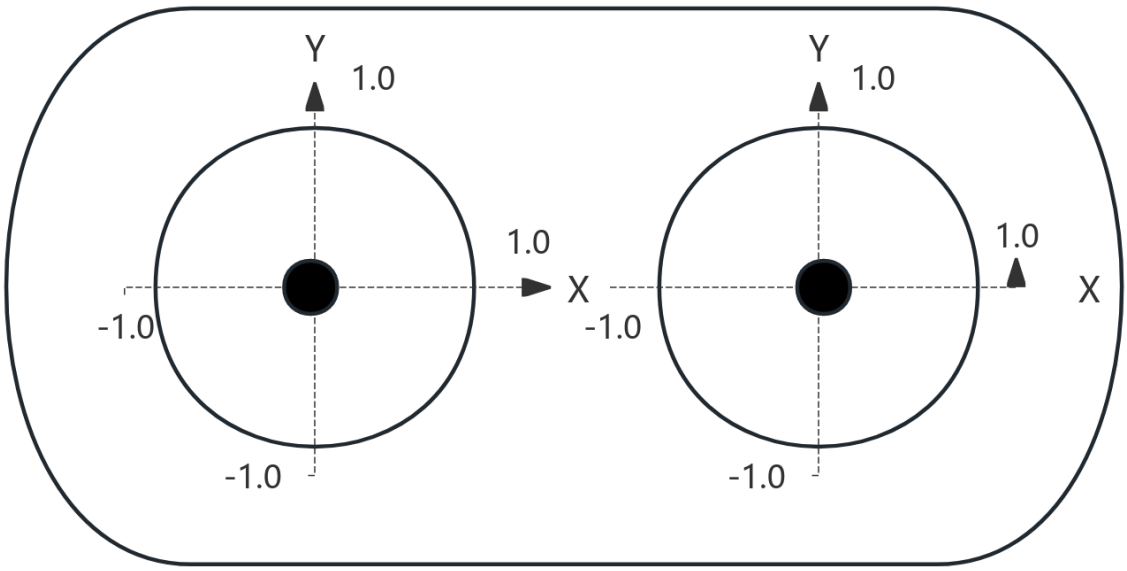
14.3.4 JoystickCommand —Joystick Command Structure

Field Name	Type	Description
`left_x_axis`	`float`	X-axis value of the left joystick (-1.0: left, 1.0: right)
`left_y_axis`	`float`	Y-axis value of the left joystick (-1.0: down, 1.0: up)
`right_x_axis`	`float`	X-axis value of the right joystick (rotation -1.0: left, 1.0: right)
`right_y_axis`	`float`	Y-axis value of the right joystick (usage not defined yet)

14.3.5 EulerAngles —Euler angles Structure

Field Name	Type	Description
`roll`	`double`	Roll angle - rotation around the X-axis(in radians)
`pitch`	`double`	Pitch angle - rotation around the Y-axis(in radians)
`yaw`	`double`	XYaw angle - rotation around the Z-axis(in radians)

14.4 Joystick Diagram



1. The value range of the x and y axes of both joysticks is [-1.0, 1.0];
2. The positive direction of the x and y axes is right/up, as shown in the diagram;

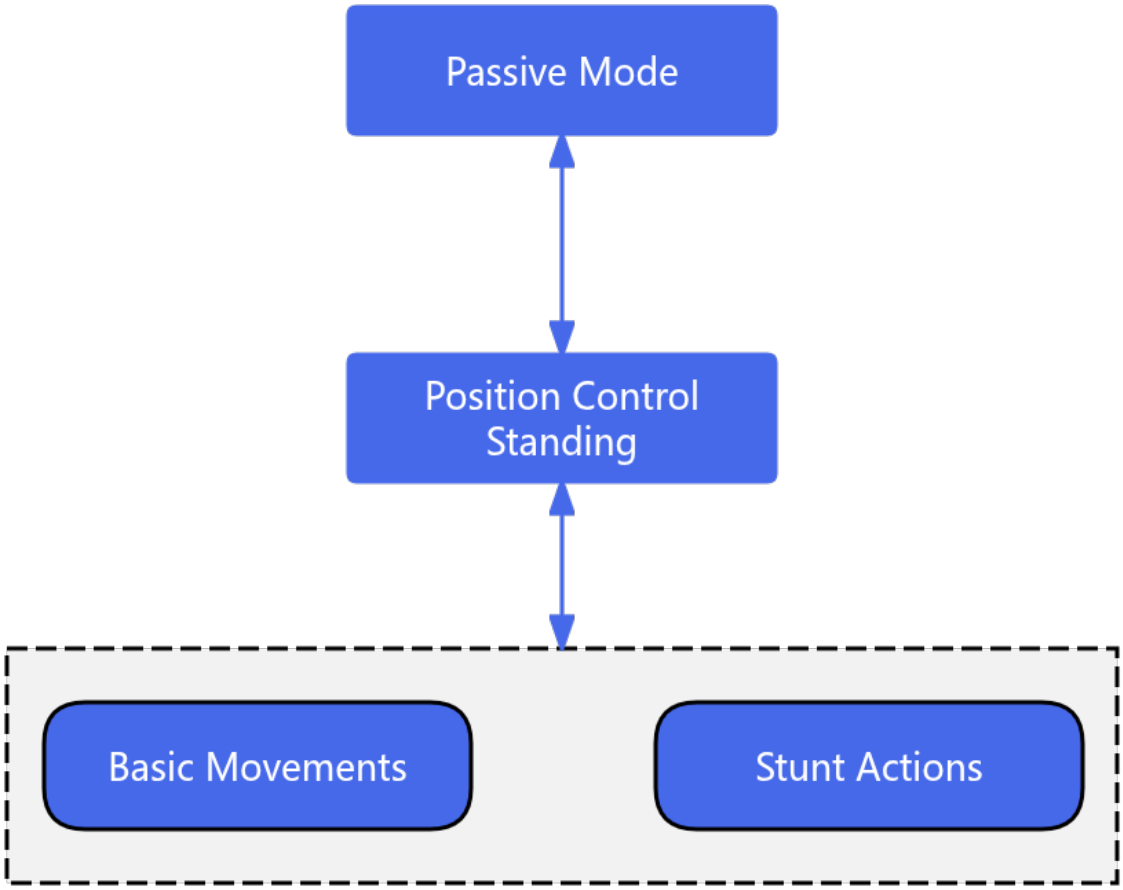
14.5 High-Level Motion Control Robot State Introduction

The robot’ s motion includes position-controlled stand, force-controlled stand, basic motion, and trick action states. During operation, the robot switches between different states via a state machine to accomplish different control tasks. The explanations for each state are as follows:

- Position-controlled stand: In this state, you can call various SDK interfaces to perform tricks and basic motion control.
- Force-controlled stand: In this state, it can be used for pose display.
- Basic motion: During motion execution, you can call SDK interfaces to let the robot enter different gaits.

- Trick action: When entering the special action execution state, other motion control services will be suspended. After the current action is completed and the robot returns to balance stand, other services will take effect.

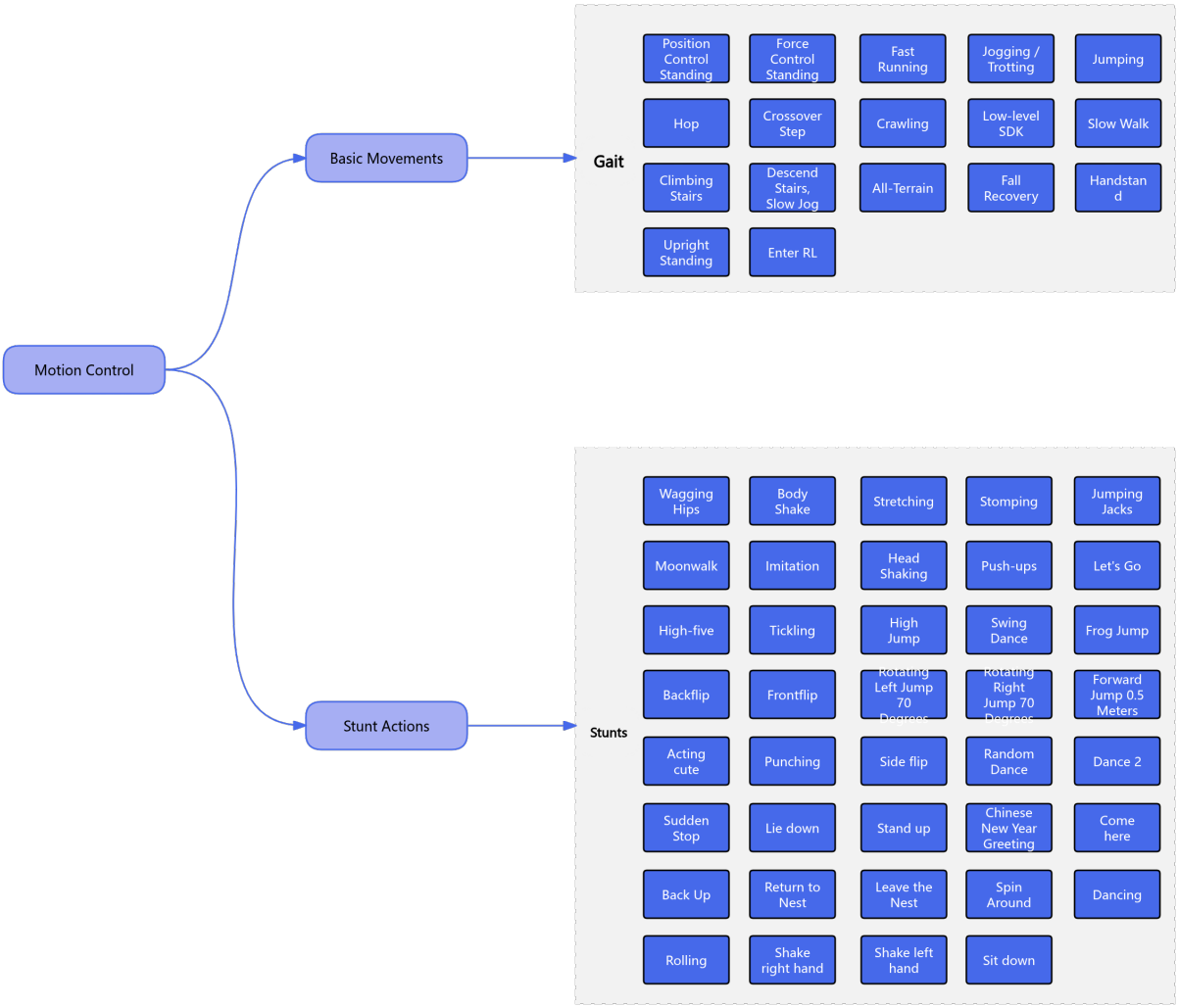
Robot state switching mechanism:



14.6 High-Level Motion Control Interface

The robot’s high-level motion control service can be divided into basic motion control and trick action control.

- In the basic motion control service, you can call the corresponding interface to switch the robot’s walking gait according to different terrain scenarios and task requirements.
- In the trick action control service, you can call the corresponding interface to perform built-in special tricks, such as wiggle hip, lie down, etc.


Motion Control
Basic Movements
Gait
Position Control Standing
Force Control Standing
Fast Running
Jogging / Trotting
Jumping
Hop
Crossover Step
Crawling
Low-level SDK
Slow Walk
Climbing Stairs
Descend Stairs, Slow Jog
All-Terrain
Fall Recovery
Handstand
Upright Standing
Enter RL
Stunt Actions
Stunts
Wagging Hips
Body Shake
Stretching
Stomping
Jumping Jacks
Moonwalk
Imitation
Head Shaking
Push-ups
Let's Go
High-five
Tickling
High Jump
Swing Dance
Frog Jump
Backflip
Frontflip
Rotating Left Jump 70 Degrees
Rotating Right Jump 70 Degrees
Forward Jump 0.5 Meters
Acting cute
Punching
Side flip
Random Dance
Dance 2
Sudden Stop
Lie down
Stand up
Chinese New Year Greeting
Come here
Back Up
Return to Nest
Leave the Nest
Spin Around
Dancing
Rolling
Shake right hand
Shake left hand
Sit down

CHAPTER

FIFTEEN

LOW-LEVEL MOTION CONTROL SERVICE (PYTHON)

Provides low-level motion control services for the robot system. Through the LowLevelMotionController, you can use topic-based communication to send joint commands and receive joint states.

15.1 Interface Definition

LowLevelMotionController is a motion controller designed for low-level development, supporting direct control and state subscription of leg and other moving parts.

⚠ **Notice:** Only when you call the publish_leg_command interface will your command be sent to the motion controller. Therefore, you must ensure the calling frequency of publish_leg_command, which is recommended to be 500 Hz.

15.1.1 LowLevelMotionController —Low-Level Motion Controller

Item	Description
Class Name	`LowLevelMotionController`
Overview	Low-level joint motion controller providing precise joint control
Main Functions	Joint state subscription, joint command publishing, control period setting
Use Cases	Precise motion control, custom gait, research and development

15.1.2 initialize

Item	Description
Function Name	`initialize`
Declaration	`bool initialize()`
Overview	Initialize the controller and establish low-level connections.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

15.1.3 shutdown

Item	Description
Function Name	`shutdown`
Declaration	`void shutdown()`
Overview	Shut down the controller and release low-level resources.
Note	Used together with initialize.

15.1.4 subscribe_leg_state

Item	Description
Function Name	`subscribe_leg_state`
Declaration	`void subscribe_leg_state(callback)`
Overview	Subscribe to leg joint state data.
Parameters	`callback`: Callback function to handle received leg joint state data. Function signature: callback(data : LegState) -> None
Note	Non-blocking interface.

15.1.5 unsubscribe_leg_state

Item	Description
Function Name	`unsubscribe_leg_state`
Declaration	`void unsubscribe_leg_state()`
Overview	Unsubscribe from leg joint state data.
Note	Non-blocking interface. Used together with subscribe_leg_state.

15.1.6 publish_leg_command

Item	Description
Function Name	`publish_leg_command`
Declaration	`Status publish_leg_command(LegJointCommand command)`
Overview	Publish leg joint control commands.
Parameters	`command`: Leg joint control command containing target angles/velocities, etc.
Return Value	`Status::OK` for success, others for failure.
Note	Non-blocking interface.

15.2 Data Structure Definitions

15.2.1 LegJointCommandArray —Leg Joint Command Array

Item	Description
Type	Python binding of `std::array<magic::dog::SingleLegJointCommand, magic::dog::kLegJointNum>`
Overview	Fixed-size array of leg joint commands, containing control commands for all leg joints
Main Methods	Supports index access, iteration, and length query. Length is fixed to the number of leg joints.
Use Cases	Leg motion control, joint coordination control, gait planning

15.2.2 LegJointStateArray —Leg Joint State Array

Item	Description
Type	Python binding of `std::array<magic::dog::SingleLegJointState, magic::dog::kLegJointNum>`
Overview	Fixed-size array of leg joint states, containing real-time state information for all leg joints
Main Methods	Supports index access, iteration, and length query. Length is fixed to the number of leg joints.
Use Cases	Joint state monitoring, motion feedback, safety detection

15.2.3 SingleLegJointCommand —Single Leg Joint Command Structure

Field Name	Type	Description
`q_des`	`float`	Desired joint angle
`dq_des`	`float`	Desired joint angular velocity
`tau_des`	`float`	Desired joint torque
`kp`	`float`	Position gain
`kd`	`float`	Velocity gain

15.2.4 LegJointCommand —Leg Joint Command Structure

Field Name	Type	Description
`timestamp`	`int`	Timestamp
`cmd`	`LegJointCommandArray`	List of joint commands

15.2.5 SingleLegJointState —Single Leg Joint State Structure

Field Name	Type	Description
`q`	`float`	Joint angle
`dq`	`float`	Joint angular velocity
`tau_est`	`float`	Estimated joint torque

15.2.6 LegState —Leg State Structure

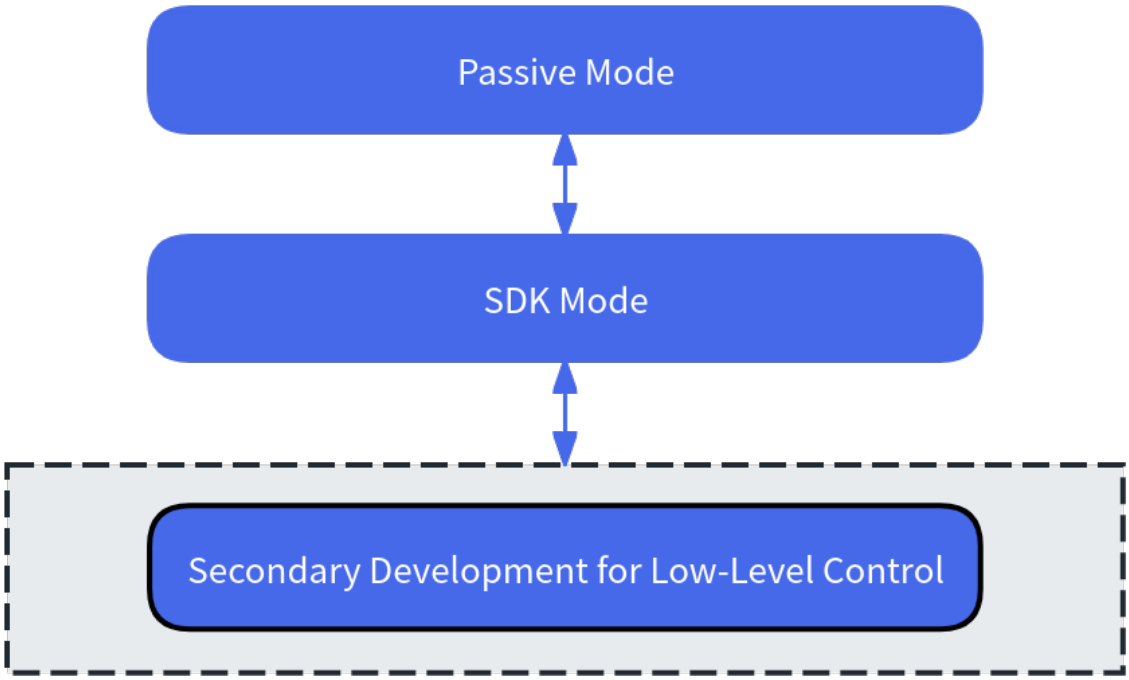
Field Name	Type	Description
`timestamp`	`int`	Timestamp
`state`	`LegJointStateArray`	List of joint states

15.3 URDF Reference

Robot URDF

15.4 Introduction to Low-Level Motion Control Robot States

The robot’s low-level motion mainly refers to the three-loop joint control, enabling developers to further develop the robot’s motion capabilities. The basic control state switching mechanism:



15.5 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

SIXTEEN

SENSOR CONTROL SERVICE (PYTHON)

Provides management of robot system sensor data, including camera, IMU, LiDAR, etc.

16.1 Interface Definition

SensorController is responsible for robot sensor data management, including camera, IMU, LiDAR, etc.

16.1.1 SensorController —Sensor Controller

Item	Description
Class Name	`SensorController`
Overview	Robot sensor data management, including camera, IMU, LiDAR, etc.
Main Functions	Sensor switch control, data subscription, multimodal perception
Use Cases	Environmental perception, navigation and localization, status monitoring

16.1.2 initialize

Item	Description
Function Name	`initialize`
Method Declaration	`bool initialize()`
Overview	Initialize the sensor controller.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

16.1.3 shutdown

Item	Description
Function Name	`shutdown`
Method Declaration	`void shutdown()`
Overview	Shut down the sensor controller.
Note	Used together with initialize.

16.1.4 open_laser_scan

Item	Description
Function Name	`open_laser_scan`
Method Declaration	`Status open_laser_scan(int timeout_ms)`
Overview	Open the LiDAR.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

16.1.5 close_laser_scan

Item	Description
Function Name	`close_laser_scan`
Method Declaration	`Status close_laser_scan(int timeout_ms)`
Overview	Close the LiDAR.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with open function.

16.1.6 open_rgbd_camera

Item	Description
Function Name	`open_rgbd_camera`
Method Declaration	`Status open_rgbd_camera(int timeout_ms)`
Overview	Open the RGBD camera.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

16.1.7 close_rgbd_camera

Item	Description
Function Name	`close_rgbd_camera`
Method Declaration	`Status close_rgbd_camera(int timeout_ms)`
Overview	Close the RGBD camera.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with open function.

16.1.8 open_binocular_camera

Item	Description
Function Name	`open_binocular_camera`
Method Declaration	`Status open_binocular_camera(int timeout_ms)`
Overview	Open the binocular camera.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface.

16.1.9 close_binocular_camera

Item	Description
Function Name	`close_binocular_camera`
Method Declaration	`Status close_binocular_camera(int timeout_ms)`
Overview	Close the binocular camera.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, used together with open function.

16.1.10 subscribe_imu

Item	Description
Function Name	`subscribe_imu`
Method Declaration	`void subscribe_imu(callback)`
Overview	Subscribe to IMU data.
Parameter Description	`callback`: Callback function to receive IMU data. Function signature: callback(data : Imu) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.11 unsubscribe_imu

Item	Description
Function Name	`unsubscribe_imu`
Method Declaration	`void unsubscribe_imu()`
Overview	Unsubscribe from IMU data.
Note	Non-blocking interface. Used together with subscribe_imu.

16.1.12 subscribe_ultra

Item	Description
Function Name	`subscribe_ultra`
Method Declaration	`void subscribe_ultra(callback)`
Overview	Subscribe to ultrasonic data.
Parameter Description	`callback`: Callback function to receive ultrasonic data. Function signature: callback(data : Float32MultiArray) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.13 unsubscribe_ultra

Item	Description
Function Name	`unsubscribe_ultra`
Method Declaration	`void unsubscribe_ultra()`
Overview	Unsubscribe from ultrasonic data.
Note	Non-blocking interface. Used together with subscribe_ultra.

16.1.14 subscribe_head_touch

Item	Description
Function Name	`subscribe_head_touch`
Method Declaration	`void subscribe_head_touch(callback)`
Overview	Subscribe to head touch data.
Parameter Description	`callback`: Callback function to receive head touch data. Function signature: callback(data : HeadTouch) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.15 unsubscribe_head_touch

Item	Description
Function Name	`unsubscribe_head_touch`
Method Declaration	`void unsubscribe_head_touch()`
Overview	Unsubscribe from head touch data.
Note	Non-blocking interface. Used together with subscribe_head_touch.

16.1.16 subscribe_laser_scan

Item	Description
Function Name	`subscribe_laser_scan`
Method Declaration	`void subscribe_laser_scan(callback)`
Overview	Subscribe to LiDAR data.
Parameter Description	`callback`: Callback function to receive LiDAR data. Function signature: callback(data : LaserScan) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.17 unsubscribe_laser_scan

Item	Description
Function Name	`unsubscribe_laser_scan`
Method Declaration	`void unsubscribe_laser_scan()`
Overview	Unsubscribe from LiDAR data.
Note	Non-blocking interface. Used together with subscribe_laser_scan.

16.1.18 subscribe_rgb_depth_camera_info

Item	Description
Function Name	`subscribe_rgb_depth_camera_info`
Method Declaration	`void subscribe_rgb_depth_camera_info(callback)`
Overview	Subscribe to RGBD depth camera intrinsic data.
Parameter Description	`callback`: Callback function to receive camera intrinsic data. Function signature: callback(data : CameraInfo) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.19 unsubscribe_rgb_depth_camera_info

Item	Description
Function Name	`unsubscribe_rgb_depth_camera_info`
Method Declaration	`void unsubscribe_rgb_depth_camera_info()`
Overview	Unsubscribe from RGBD depth camera intrinsic data.
Note	Non-blocking interface. Used together with subscribe_rgb_depth_camera_info.

16.1.20 subscribe_rgbd_depth_image

Item	Description
Function Name	`subscribe_rgbd_depth_image`
Method Declaration	`void subscribe_rgbd_depth_image(callback)`
Overview	Subscribe to RGBD depth image data.
Parameter Description	`callback`: Callback function to receive depth image data. Function signature: callback(data : Image) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.21 unsubscribe_rgbd_depth_image

Item	Description
Function Name	`unsubscribe_rgbd_depth_image`
Method Declaration	`void unsubscribe_rgbd_depth_image()`
Overview	Unsubscribe from RGBD depth image data.
Note	Non-blocking interface. Used together with subscribe_rgbd_depth_image.

16.1.22 subscribe_rgbd_color_camera_info

Item	Description
Function Name	`subscribe_rgbd_color_camera_info`
Method Declaration	`void subscribe_rgbd_color_camera_info(callback)`
Overview	Subscribe to RGBD color image intrinsic data.
Parameter Description	`callback`: Callback function to receive camera intrinsic data. Function signature: callback(data : CameraInfo) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.23 unsubscribe_rgbd_color_camera_info

Item	Description
Function Name	`unsubscribe_rgbd_color_camera_info`
Method Declaration	`void unsubscribe_rgbd_color_camera_info()`
Overview	Unsubscribe from RGBD color image intrinsic data.
Note	Non-blocking interface. Used together with subscribe_rgbd_color_camera_info.

16.1.24 subscribe_rgbd_color_image

Item	Description
Function Name	`subscribe_rgbd_color_image`
Method Declaration	`void subscribe_rgbd_color_image(callback)`
Overview	Subscribe to RGBD color image data.
Parameter Description	`callback`: Callback function to receive color image data. Function signature: callback(data : Image) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.25 unsubscribe_rgbd_color_image

Item	Description
Function Name	`unsubscribe_rgbd_color_image`
Method Declaration	`void unsubscribe_rgbd_color_image()`
Overview	Unsubscribe from RGBD color image data.
Note	Non-blocking interface. Used together with subscribe_rgbd_color_image.

16.1.26 subscribe_left_binocular_high_img

Item	Description
Function Name	`subscribe_left_binocular_high_img`
Method Declaration	`void subscribe_left_binocular_high_img(callback)`
Overview	Subscribe to left high-quality binocular data.
Parameter Description	`callback`: Callback function to receive left high-quality binocular data. Function signature: callback(data : CompressedImage) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.27 unsubscribe_left_binocular_high_img

Item	Description
Function Name	`unsubscribe_left_binocular_high_img`
Method Declaration	`void unsubscribe_left_binocular_high_img()`
Overview	Unsubscribe from left high-quality binocular data.
Note	Non-blocking interface. Used together with subscribe_left_binocular_high_img.

16.1.28 subscribe_left_binocular_low_img

Item	Description
Function Name	`subscribe_left_binocular_low_img`
Method Declaration	`void subscribe_left_binocular_low_img(callback)`
Overview	Subscribe to left low-quality binocular data.
Parameter Description	`callback`: Callback function to receive left low-quality binocular data. Function signature: callback(data : CompressedImage) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.29 unsubscribe_left_binocular_low_img

Item	Description
Function Name	`unsubscribe_left_binocular_low_img`
Method Declaration	`void unsubscribe_left_binocular_low_img()`
Overview	Unsubscribe from left low-quality binocular data.
Note	Non-blocking interface. Used together with subscribe_left_binocular_low_img.

16.1.30 subscribe_right_binocular_low_img

Item	Description
Function Name	`subscribe_right_binocular_low_img`
Method Declaration	`void subscribe_right_binocular_low_img(callback)`
Overview	Subscribe to right low-quality binocular data.
Parameter Description	`callback`: Callback function to receive right low-quality binocular data. Function signature: callback(data : CompressedImage) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.31 unsubscribe_right_binocular_low_img

Item	Description
Function Name	`unsubscribe_right_binocular_low_img`
Method Declaration	`void unsubscribe_right_binocular_low_img()`
Overview	Unsubscribe from right low-quality binocular data.
Note	Non-blocking interface. Used together with subscribe_right_binocular_low_img.

16.1.32 subscribe_depth_image

Item	Description
Function Name	`subscribe_depth_image`
Method Declaration	`void subscribe_depth_image(callback)`
Overview	Subscribe to depth image data.
Parameter Description	`callback`: Callback function to receive depth image data. Function signature: callback(data : Image) -> None
Note	Non-blocking interface, data received asynchronously via callback.

16.1.33 unsubscribe_depth_image

Item	Description
Function Name	`unsubscribe_depth_image`
Method Declaration	`void unsubscribe_depth_image()`
Overview	Unsubscribe from depth image data.
Note	Non-blocking interface. Used together with subscribe_depth_image.

16.2 Data Structure Definitions

16.2.1 Uint8Vector —Unsigned 8-bit Integer Vector

Item	Description
Type	Python binding of `std::vector<uint8_t>`
Overview	Dynamic array for storing unsigned 8-bit integers, commonly used for raw byte data
Main Methods	Supports standard Python list operations: indexing, slicing, iteration, length query, etc.
Use Cases	Image data, audio data, binary data storage

16.2.2 PointFieldVector —Point Cloud Field Vector

Item	Description
Type	Python binding of `std::vector<magic::dog::PointField>`
Overview	Dynamic array for storing point cloud field descriptions, used to define point cloud data structure
Main Methods	Supports standard Python list operations, each element is a PointField struct
Use Cases	Point cloud data processing, 3D sensor data parsing

16.2.3 MultiArrayDimensionVector —Multi-dimensional Array Dimension Vector

Item	Description
Type	Python binding of `std::vector<magic::dog::MultiArrayDimension>`
Overview	Dynamic array for storing multi-dimensional array dimension information, used to describe array shape and layout
Main Methods	Supports standard Python list operations, each element is a MultiArrayDimension struct
Use Cases	Multi-dimensional sensor data, matrix data processing

16.2.4 FloatVector —Float Vector

Item	Description
Type	Python binding of `std::vector<float>`
Overview	Dynamic array for storing single-precision floats, used for high-precision numerical computation
Main Methods	Supports standard Python list operations, supports float arithmetic
Use Cases	Sensor data, coordinate data, physical quantity data

16.2.5 DoubleVector —Double Precision Float Vector

Item	Description
Type	Python binding of `std::vector<double>`
Overview	Dynamic array for storing double-precision floats, provides highest precision numerical computation
Main Methods	Supports standard Python list operations, supports high-precision float arithmetic
Use Cases	High-precision sensor data, scientific computation, coordinate transformation

16.2.6 Double3Array —3D Double Precision Float Array

Item	Description
Type	Python binding of `std::array<double, 3>`
Overview	Fixed-size 3D double array, commonly used for 3D coordinate representation
Main Methods	Supports indexing, iteration, length query, fixed length 3
Use Cases	3D coordinates, RGB values, Euler angles, position vectors

16.2.7 Double4Array —4D Double Precision Float Array

Item	Description
Type	Python binding of `std::array<double, 4>`
Overview	Fixed-size 4D double array, commonly used for quaternion representation
Main Methods	Supports indexing, iteration, length query, fixed length 4
Use Cases	Quaternions, RGBA values, pose representation

16.2.8 Double9Array —9D Double Precision Float Array

Item	Description
Type	Python binding of `std::array<double, 9>`
Overview	Fixed-size 9D double array, commonly used for 3x3 matrix representation
Main Methods	Supports indexing, iteration, length query, fixed length 9
Use Cases	3x3 rotation matrix, covariance matrix, transformation matrix

16.2.9 Double12Array —12D Double Precision Float Array

Item	Description
Type	Python binding of `std::array<double, 12>`
Overview	Fixed-size 12D double array, commonly used for 3x4 projection matrix representation
Main Methods	Supports indexing, iteration, length query, fixed length 12
Use Cases	3x4 projection matrix, camera parameter matrix

16.2.10 Header —Header Structure

Field Name	Type	Description
`stamp`	`int`	Timestamp
`frame_id`	`str`	Coordinate frame ID

16.2.11 Image —Image Structure

Field Name	Type	Description
`header`	`Header`	Header information
`height`	`int`	Image height
`width`	`int`	Image width
`encoding`	`str`	Encoding format
`is_bigendian`	`bool`	Is big endian
`step`	`int`	Step size
`data`	`Uint8Vector`	Image data

16.2.12 Imu —Inertial Measurement Unit Structure

Field Name	Type	Description
`timestamp`	`int`	Timestamp
`orientation`	`Double4Array`	Orientation quaternion (read-only)
`angular_velocity`	`Double3Array`	Angular velocity (read-only)
`linear_acceleration`	`Double3Array`	Linear acceleration (read-only)
`temperature`	`float`	Temperature

16.2.13 PointField —Point Cloud Field Description Structure

Field Name	Type	Description
`name`	`str`	Field name, e.g. “x”, “y”, “z”, “intensity”, etc.
`offset`	`int`	Starting byte offset
`datatype`	`int`	Data type (corresponding constant)
`count`	`int`	Number of elements in this field

16.2.14 PointCloud2 —General Point Cloud Data Structure

Field Name	Type	Description
`header`	`Header`	Standard message header
`height`	`int`	Number of rows
`width`	`int`	Number of columns
`fields`	`PointFieldVector`	Point field array
`is_bigendian`	`bool`	Byte order
`point_step`	`int`	Bytes per point
`row_step`	`int`	Bytes per row
`data`	`Uint8Vector`	Raw point cloud data (packed by field)
`is_dense`	`bool`	Is dense point cloud (no invalid points)

16.2.15 CameraInfo —Camera Intrinsic and Distortion Info Structure

Field Name	Type	Description
`header`	`Header`	General message header
`height`	`int`	Image height (rows)
`width`	`int`	Image width (columns)
`distortion_model`	`str`	Distortion model, e.g. “plumb_bob”
`D`	`FloatVector`	Distortion parameter array
`K`	`FloatVector`	Camera intrinsic matrix (9 elements)
`R`	`FloatVector`	Rectification matrix (9 elements)
`P`	`FloatVector`	Projection matrix (12 elements)
`binning_x`	`int`	Horizontal binning factor
`binning_y`	`int`	Vertical binning factor
`roi_x_offset`	`int`	ROI start x
`roi_y_offset`	`int`	ROI start y
`roi_height`	`int`	ROI height
`roi_width`	`int`	ROI width
`roi_do_rectify`	`bool`	Whether to rectify

16.2.16 CompressedImage —Compressed Image Structure

Field Name	Type	Description
`header`	`Header`	General message header
`format`	`str`	Compression format
`data`	`Uint8Vector`	Compressed image data

16.2.17 LaserScan —LiDAR Data Structure

Field Name	Type	Description
`header`	`Header`	General message header
`angle_min`	`int`	Minimum angle (radians)
`angle_max`	`int`	Maximum angle (radians)
`angle_increment`	`int`	Angle increment (radians)
`time_increment`	`int`	Time increment (seconds)
`scan_time`	`int`	Scan time (seconds)
`range_min`	`int`	Minimum range (meters)
`range_max`	`int`	Maximum range (meters)
`ranges`	`FloatVector`	Range data array (meters)
`intensities`	`FloatVector`	Intensity data array (unitless)

16.2.18 MultiArrayDimension —Multi-dimensional Array Dimension Description Structure

Field Name	Type	Description
`label`	`str`	Dimension label
`size`	`int`	Dimension size
`stride`	`int`	Stride

16.2.19 MultiArrayLayout —Multi-dimensional Array Layout Description Structure

Field Name	Type	Description
`dim_size`	`int`	Number of dimensions
`dim`	`MultiArrayDimensionVector`	Dimension description array
`data_offset`	`int`	Data offset

16.2.20 Float32MultiArray —Float Multi-dimensional Array Structure

Field Name	Type	Description
`layout`	`MultiArrayLayout`	Array layout description
`data`	`FloatVector`	Float data array

16.2.21 ByteMultiArray —Byte Array Structure

Field Name	Type	Description
`layout`	`MultiArrayLayout`	Array layout description
`data`	`Uint8Vector`	Byte data array

16.2.22 HeadTouch —Head Touch Data Structure

Field Name	Type	Description
`data`	`int`	Touch state data

16.3 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

SEVENTEEN

AUDIO CONTROL SERVICE (PYTHON)

Provides robot system audio control, including TTS synthesis, audio playback, and audio configuration.

17.1 API Definition

AudioController is responsible for robot audio control, including TTS synthesis, audio playback, and audio configuration.

17.1.1 AudioController —Audio Controller

Item	Description
Class Name	`AudioController`
Overview	Robot audio control, including TTS synthesis, audio playback, and audio configuration
Main Features	Audio playback, volume control, TTS model switching, raw audio data subscription
Use Cases	Voice interaction, audio playback, speech recognition

17.1.2 initialize

Item	Description
Function Name	`initialize`
Declaration	`bool initialize()`
Overview	Initialize the audio controller.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

17.1.3 shutdown

Item	Description
Function Name	`shutdown`
Declaration	`void shutdown()`
Overview	Shut down the audio controller.
Note	Used together with initialize.

17.1.4 play

Item	Description
Function Name	`play`
Declaration	`Status play(TtsCommand command, int timeout_ms)`
Overview	Play a TTS (Text-to-Speech) command.
Parameters	`command`: TTS command, including text, speed, pitch, etc.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, ensure the module is initialized before calling.

17.1.5 stop

Item	Description
Function Name	`stop`
Declaration	`Status stop(int timeout_ms)`
Overview	Stop current audio playback.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, usually used to interrupt current speech.

17.1.6 set_volume

Item	Description
Function Name	`set_volume`
Declaration	`Status set_volume(int volume, int timeout_ms)`
Overview	Set the audio output volume.
Parameters	`volume`: Volume value, usually in the range 0~100.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, takes effect immediately after setting.

17.1.7 get_volume

Item	Description
Function Name	`get_volume`
Declaration	`tuple[Status, int] get_volume(int timeout_ms)`
Overview	Get the current audio output volume.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	success, return current volume value.
Note	Non-blocking interface.

17.1.8 switch_tts_voice_model

Item	Description
Function Name	`switch_tts_voice_model`
Declaration	`Status switch_tts_voice_model(TtsType tts_type, int timeout_ms)`
Overview	Switch TTS voice model.
Parameters	`tts_type`: TTS voice model type`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Operation status.
Note	Switching the model is a blocking operation.

17.1.9 get_voice_config

Item	Description
Function Name	`get_voice_config`
Declaration	`tuple[Status, GetSpeechConfig] get_voice_config(int timeout_ms)`
Overview	Get the complete configuration of the speech system.
Parameters	`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Speech system configuration, returns empty config object on failure.
Note	Blocking interface, the configuration includes all sub-configs such as speaker, bot, wakeup, dialog, etc.

17.1.10 set_voice_config

Item	Description
Function Name	`set_voice_config`
Declaration	`Status set_voice_config(SetSpeechConfig config, int timeout_ms)`
Overview	Set the complete configuration of the speech system.
Parameters	`config`: Speech system configuration to set.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface, the set configuration will completely overwrite all current speech-related configs.

17.1.11 control_voice_stream

Item	Description
Function Name	`control_voice_stream`
Declaration	`Status control_voice_stream(bool raw_data, bool bf_data, int timeout_ms)`
Overview	Control the voice data stream.
Parameters	`raw_data`: Whether to enable the original voice data stream. `bf_data`: Whether to enable the BF voice data stream. `timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	Operation status. `Status::OK` indicates success, others indicate failure.
Note	Blocking interface, used to control enabling and disabling of voice data streams.

17.1.12 subscribe_origin_voice_data

Item	Description
Function Name	`subscribe_origin_voice_data`
Declaration	`void subscribe_origin_voice_data(callback)`
Overview	Subscribe to raw voice data.
Parameters	`callback`: Callback to handle received raw voice data. Function signature: callback(data : ByteMultiArray) -> None
Note	Non-blocking interface, callback will be called when data is updated.

17.1.13 unsubscribe_origin_voice_data

Item	Description
Function Name	`unsubscribe_origin_voice_data`
Declaration	`void unsubscribe_origin_voice_data()`
Overview	Unsubscribe from raw voice data.
Note	Non-blocking interface. Used together with subscribe_origin_voice_data.

17.1.14 subscribe_bf_voice_data

Item	Description
Function Name	`subscribe_bf_voice_data`
Declaration	`void subscribe_bf_voice_data(callback)`
Overview	Subscribe to BF voice data.
Parameters	`callback`: Callback to handle received BF voice data. Function signature: callback(data : ByteMultiArray) -> None
Note	Non-blocking interface, callback will be called when data is updated.

17.1.15 unsubscribe_bf_voice_data

Item	Description
Function Name	`unsubscribe_bf_voice_data`
Declaration	`void unsubscribe_bf_voice_data()`
Overview	Unsubscribe from BF voice data.
Note	Non-blocking interface. Used together with subscribe_bf_voice_data.

17.1.16 control_speech_io

Item	Content
Function Name	control_speech_io
Function Declaration	`Status control_speech_io(bool enable_data, int timeout_ms);`
Function Overview	Control speech io data stream(asr and tts).
Parameter Description	`enable_data`: Enable data stream. `timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to control speech io data stream(asr and tts).

17.1.17 subscribe_speech_asr

Item	Content
Function Name	subscribe_speech_asr
Function Declaration	void subscribe_speech_asr(callback);
Function Overview	Subscribe to speech asr stream.
Parameter Description	`callback`: Callback to handle received speech asr stream.
Notes	Blocking interface, used to subscribe to speech asr stream.

17.1.18 unsubscribe_speech_asr

Item	Content
Function Name	unsubscribe_speech_asr
Function Declaration	void unsubscribe_speech_asr();
Function Overview	Unsubscribe from speech asr stream.
Notes	Blocking interface, used to unsubscribe from speech asr stream.

17.1.19 publish_speech_tts

Item	Content
Function Name	publish_speech_tts
Function Declaration	`Status publish_speech_tts(SpeechTTSStream data);`
Function Overview	Publish speech tts stream.
Parameter Description	`data`: Speech TTS stream data.
Return Value	`Status::OK` means success, others mean failure.
Notes	Blocking interface, used to publish speech tts stream. The data ID should be consistent with the request ID from the ASR output, with the start type of begin, the middle data type of var and the end type of end.

17.2 Enum Type Definitions

17.2.1 TtsPriority —TTS Priority Enum

Enum Value	Value	Description
`HIGH`	0	High priority
`MIDDLE`	1	Medium priority
`LOW`	2	Low priority

17.2.2 TtsMode —TTS Mode Enum

Enum Value	Value	Description
`CLEARTOP`	0	Clear top
`ADD`	1	Add
`CLEARBUFFER`	2	Clear buffer

17.2.3 TtsType —TTS Type Enum

Enum Value	Value	Description
NONE	0	None
DOUBAO	1	Doubao
GOOGLE	2	Google

17.3 Data Structure Definitions

17.3.1 S2DString —2D String Array

Item	Description
Type	Python binding of std::array<std::string, 2>
Overview	Fixed-size 2D string array containing two string elements
Main Methods	Supports index access, iteration, length query, fixed length 2
Use Cases	Key-value storage, configuration parameters, status identification

17.3.2 S2DStringVector —2D String Array Vector

Item	Description
Type	Python binding of std::vector<std::array<std::string, 2>>
Overview	Variable-length 2D string array, each element is an array of two strings
Main Methods	Supports index access, iteration, length query, dynamic add/delete, each element length is 2
Use Cases	Batch key-value storage, configuration parameter sets, status identification lists

17.3.3 String2DStringVectorMap —String to 2D String Array Vector Map

Item	Description
Type	Python binding of std::map<std::string, std::vector<std::array<std::string, 2>>>
Overview	Map from string key to 2D string array vector, used for configuration parameters
Main Methods	Supports standard Python dict operations: key-value access, iteration, length query, etc.
Use Cases	Speaker configuration, bot configuration, system parameter storage

17.3.4 StringBotInfoMap —String to Bot Info Map

Item	Description
Type	Python binding of `std::map<std::string, magic::dog::BotInfo>`
Overview	Map from string key to bot info, used to manage multiple bot configurations
Main Methods	Supports standard Python dict operations, each value is a BotInfo struct
Use Cases	Bot management, configuration storage, multi-bot systems

17.3.5 StringCustomBotMap —Custom Bot Map

Item	Description
Type	Python binding of `std::map<std::string, magic::dog::CustomBotInfo>`
Overview	Map from string key to custom bot info, used to manage user-defined bots
Main Methods	Supports standard Python dict operations, each value is a CustomBotInfo struct
Use Cases	Custom bot management, user configuration storage

17.3.6 StringStringMap —String to String Map

Item	Description
Type	Python binding of `std::map<std::string, std::string>`
Overview	Simple mapping from string key to string value, used for configuration parameters
Main Methods	Supports standard Python dict operations, both key and value are strings
Use Cases	Simple configuration storage, parameter mapping, status identification

17.3.7 TtsCommand —TTS Command Struct

Field Name	Type	Description
`id`	`int`	Command ID
`content`	`str`	Content
`priority`	`TtsPriority`	Priority
`mode`	`TtsMode`	Mode

17.3.8 GetSpeechConfig —Speech Config Get Struct

Field Name	Type	Description
tts_type	TtsType	TTS Type
speaker_config	SpeakerConfig	Speaker configuration
bot_config	BotConfig	Bot configuration
wakeup_config	WakeupConfig	Wakeup configuration
dialog_config	DialogConfig	Dialog configuration

17.3.9 SetSpeechConfig —Speech Config Set Struct

Field Name	Type	Description
speaker_id	str	Speaker ID
region	str	Speaker region
bot_id	str	Bot ID
is_front_doa	bool	Enable front DOA
is_fullduplex_enable	bool	Enable full-duplex dialog
is_enable	bool	Enable dialog function
is_doa_enable	bool	Enable DOA
speaker_speed	float	Speaker speed
wakeup_name	str	Wakeup word name
custom_bot	CustomBotInfo	Custom bot info

17.3.10 SpeakerConfig —Speaker Config Struct

Field Name	Type	Description
data	MapStringVectorArray2DString	Speaker data map
selected	SpeakerConfigSelected	Selected speaker config
speaker_speed	float	Speaker speed

17.3.11 BotConfig —Bot Config Struct

Field Name	Type	Description
data	MapStringBotInfo	Bot data map
custom_data	CustomBotMap	Custom bot data
selected	BotInfo	Selected bot

17.3.12 WakeupConfig —Wakeup Config Struct

Field Name	Type	Description
`name`	`str`	Wakeup word name
`data`	`MapStringString`	Wakeup data map

17.3.13 DialogConfig —Dialog Config Struct

Field Name	Type	Description
`is_front_doa`	`bool`	Enable front DOA
`is_fullduplex_enable`	`bool`	Enable full-duplex dialog
`is_enable`	`bool`	Enable dialog function
`is_doa_enable`	`bool`	Enable DOA

17.3.14 SpeakerConfigSelected —Speaker Config Selected Struct

Field Name	Type	Description
`speaker_id`	`str`	Speaker ID
`region`	`str`	Speaker region

17.3.15 BotInfo —Bot Info Struct

Field Name	Type	Description
`name`	`str`	Bot name
`workflow`	`str`	Workflow ID

17.3.16 CustomBotInfo —Custom Bot Info Struct

Field Name	Type	Description
`name`	`str`	Custom bot name
`workflow`	`str`	Custom workflow ID
`token`	`str`	Access token

17.3.17 SpeechASRStream —Robot ASR output

Field Name	Type	Description
`id`	`str`	ID - request id
`type`	`str`	Type - request or cancel
`text`	`str`	Text - asr output

17.3.18 SpeechTTSStream —Robot TTS input

Field Name	Type	Description
`id`	`str`	ID - same as request id
`type`	`str`	Type - begin or var or end
`text`	`str`	Text - tts input
`end_session`	`bool`	End Session - end session and close asr

17.4 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

EIGHTEEN

STATE MONITOR SERVICE (PYTHON)

Provides monitoring of the robot system's operational status, including fault detection, battery status, etc.

18.1 Interface Definition

StateMonitor is responsible for monitoring the robot's operational status, including fault detection and battery status.

18.1.1 StateMonitor —State Monitor

Item	Description
Class Name	`StateMonitor`
Overview	Monitors the robot's operational status, including fault detection and battery status
Main Functions	Status query, fault monitoring, health check
Use Cases	System monitoring, fault diagnosis, status reporting

18.1.2 initialize

Item	Description
Function Name	`initialize`
Method Declaration	`bool initialize()`
Overview	Initializes the state monitor.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

18.1.3 shutdown

Item	Description
Function Name	`shutdown`
Method Declaration	`void shutdown()`
Overview	Shuts down the state monitor.
Note	Used together with initialize.

18.1.4 get_current_state

Item	Description
Function Name	`get_current_state`
Method Declaration	`RobotState get_current_state()`
Overview	Gets the current state information of the robot.
Return Value	`tuple[Status, RobotState]` object
Note	Non-blocking interface, used to get the current state information of the robot.

18.2 Error Code Mapping Table

Error Code (Hex)	Error Description
0x0000	No error
0x1101	Failed to call ROS service
0x1301	Main control node lost
0x1302	APP node lost
0x1304	Audio node lost
0x1305	Motion node lost
0x1306	LCD node lost
0x1307	realsense node lost
0x1308	Stereo camera node lost
0x1309	LDS node lost
0x130A	Sensor board node lost
0x130B	Touch node lost
0x130C	SLAM node lost
0x130D	Navigation node lost
0x130E	AI node lost
0x130F	Head node lost
0x1310	Cloud processor node lost

continues on next page

Table 1 – continued from previous page

Error Code (Hex)	Error Description
0x3201	No laser data
0x3202	No stereo camera data
0x3203	Stereo camera data error
0x3204	Stereo camera initialization failed
0x320B	No odometry data
0x320C	No IMU data
0x6101	Robot failed to connect to APP
0x6102	Disconnected from APP
0x9201	Failed to open LCD serial port
0x7201	Failed to open head serial port
0x7202	No head data
0x8201	Failed to open sensor board serial port
0x8202	No sensor board data
0x2201	Error: Navigation did not receive tf data
0x2202	Error: Navigation did not receive map data
0x2203	Error: Navigation did not receive localization data
0x2204	Error: Navigation did not receive ultrasonic data
0x2205	Error: Navigation did not receive laser data
0x2206	Error: Navigation did not receive RGBD data
0x2207	Error: Navigation did not receive multi-line laser data
0x2208	Error: Navigation did not receive point TOF data
0x2209	Error: Navigation did not receive area TOF data
0x220A	Error: Navigation did not receive odometry data
0x2101	Warning: Navigation did not receive tf data
0x2102	Warning: Navigation did not receive map data
0x2103	Warning: Navigation did not receive localization data
0x2104	Warning: Navigation did not receive ultrasonic data
0x2105	Warning: Navigation did not receive laser data
0x2106	Warning: Navigation did not receive RGBD TOF data
0x2107	Warning: Navigation did not receive multi-line laser data
0x2108	Warning: Navigation did not receive point TOF data
0x2109	Warning: Navigation did not receive area TOF data
0x210A	Warning: Navigation did not receive odometry data
0x4201	SLAM localization error
0x4102	Error: SLAM did not receive laser data
0x4103	Error: SLAM did not receive odometry data
0x4205	SLAM map error

18.3 Enum Type Definitions

18.3.1 BatteryState —Battery State Enum

Enum Value	Value	Description
UNKNOWN	0	Unknown state
GOOD	1	Good
OVERHEAT	2	Overheat
DEAD	3	Battery depleted
OVERVOLTAGE	4	Overvoltage
UNSPEC_FAILURE	5	Unspecified failure
COLD	6	Too cold
WATCHDOG_TIMER_EXPIRE	7	Watchdog timer expired
SAFETY_TIMER_EXPIRE	8	Safety timer expired

18.3.2 PowerSupplyStatus —Power Supply Status Enum

Enum Value	Value	Description
UNKNOWN	0	Unknown state
CHARGING	1	Charging
DISCHARGING	2	Discharging
NOTCHARGING	3	Not charging
FULL	4	Fully charged

18.4 Data Structure Definitions

18.4.1 FaultVector —Fault Information Vector

Item	Description
Type	Python binding of std::vector<magic::dog::Fault>
Overview	Dynamic array storing fault information, used for system status monitoring and error handling
Main Methods	Supports standard Python list operations, each element is a Fault struct
Use Cases	Fault diagnosis, system monitoring, error log recording

18.4.2 Fault —Fault Information Struct

Field Name	Type	Description
`error_code`	`int`	Error code
`error_message`	`str`	Error message

18.4.3 BmsData —Battery Management System Data Struct

Field Name	Type	Description
`battery_percentage`	`double`	Battery percentage
`battery_health`	`double`	Battery health
`battery_state`	`BatteryState`	Battery state
`power_supply_status`	`PowerSupplyStatus`	Power supply status

18.4.4 RobotState —Robot State Struct

Field Name	Type	Description
`faults`	`FaultVector`	Fault list
`bms_data`	`BmsData`	Battery data

CHAPTER

NINETEEN

SLAM NAVIGATION CONTROL SERVICE (PYTHON)

Provides SLAM navigation control services for robot systems. Through SlamNavController, SLAM mapping, localization, navigation and other functions can be implemented.

19.1 Interface Definition

SlamNavController is a controller for SLAM navigation control, supporting mapping, localization, navigation and other operations.

19.1.1 SlamNavController —SLAM Navigation Controller

Item	Content
Class Name	`SlamNavController`
Function Overview	Controller for SLAM navigation control, supporting mapping, localization, navigation and other operations
Main Functions	SLAM mapping, localization, navigation, map management
Use Cases	Robot autonomous navigation, environment mapping, path planning

19.1.2 initialize

Item	Content
Method Name	`initialize`
Method Declaration	`bool initialize()`
Function Overview	Initializes SLAM navigation controller.
Return Value	`true` indicates success, `false` indicates failure.
Remarks	Must be called before first use.

19.1.3 shutdown

Item	Content
Method Name	`shutdown`
Method Declaration	`void shutdown()`
Function Overview	Releases resources and cleans up SLAM navigation controller.
Remarks	Used in conjunction with initialize.

19.1.4 switch_to_idle

Item	Content
Method Name	`switch_to_idle`
Method Declaration	`Status switch_to_idle()`
Function Overview	Switches to idle mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to switch to idle state.

19.1.5 start_mapping

Item	Content
Method Name	`start_mapping`
Method Declaration	`Status start_mapping()`
Function Overview	Starts mapping mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to start mapping task.

19.1.6 cancel_mapping

Item	Content
Method Name	`cancel_mapping`
Method Declaration	`Status cancel_mapping()`
Function Overview	Cancels current mapping task.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to cancel mapping task.

19.1.7 switch_to_location

Item	Content
Method Name	`switch_to_location`
Method Declaration	`Status switch_to_location()`
Function Overview	Switches to localization mode.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to switch to localization state.

19.1.8 save_map

Item	Content
Method Name	`save_map`
Method Declaration	`Status save_map(const str& map_name)`
Function Overview	Ends mapping and saves map when in mapping mode.
Parameter Description	`map_name`: Map name.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to save current mapping result.

19.1.9 load_map

Item	Content
Method Name	`load_map`
Method Declaration	`Status load_map(const str& map_name)`
Function Overview	Loads map and sets it as current map.
Parameter Description	`map_name`: Map name.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to load saved map.

19.1.10 delete_map

Item	Content
Method Name	`delete_map`
Method Declaration	`Status delete_map(const str& map_name)`
Function Overview	Deletes map.
Parameter Description	`map_name`: Name of map to delete.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Blocking interface, used to delete unwanted map.

19.1.11 get_all_map_info

Item	Content
Method Name	get_all_map_info
Method Declaration	`tuple[Status, AllMapInfo] get_all_map_info(int timeout_ms)`
Function Overview	Get all map information
Parameter Description	`timeout_ms`: Timeout time (milliseconds), default value is 10000
Return Value	`Status::OK`: indicates success, others indicate failure`AllMapInfo`: All map information (output parameter)
Remarks	Blocking interface, gets detailed information of all maps in the system

19.1.12 init_pose

Item	Content
Method Name	`init_pose`
Method Declaration	`Status init_pose(Pose3DEuler pose)`
Function Overview	Initializes pose.
Parameter Description	`pose`: Pose information to publish。
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to set robot initial pose。

19.1.13 get_current_localization_info

Item	Content
Method Name	`get_current_localization_info`
Method Declaration	`tuple[Status, LocalizationInfo] get_current_localization_info()`
Function Overview	Gets current localization information.
Return Value	`Status::OK` indicates success, others indicate failure。`LocalizationInfo` object on success, empty object on failure.
Remarks	Blocking interface, used to get robot current pose.

19.1.14 activate_nav_mode

Item	Content
Method Name	`activate_nav_mode`
Method Declaration	`Status activate_nav_mode(NavMode mode)`
Function Overview	Activates navigation mode.
Parameter Description	`mode`: Target navigation mode (NavMode enumeration type).
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to switch navigation working mode.

19.1.15 set_nav_target

Item	Content
Method Name	`set_nav_target`
Method Declaration	`Status set_nav_target(NavTarget goal)`
Function Overview	Sets global navigation target point and starts navigation task.
Parameter Description	`goal`: Global coordinates of target point.
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to set navigation target.

19.1.16 pause_nav_task

Item	Content
Method Name	`pause_nav_task`
Method Declaration	`Status pause_nav_task()`
Function Overview	Pauses current navigation task.
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to pause navigation.

19.1.17 resume_nav_task

Item	Content
Method Name	`resume_nav_task`
Method Declaration	`Status resume_nav_task()`
Function Overview	Resumes paused navigation task.
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to resume navigation.

19.1.18 cancel_nav_task

Item	Content
Method Name	`cancel_nav_task`
Method Declaration	`Status cancel_nav_task()`
Function Overview	Cancels current navigation task.
Return Value	`Status::OK` indicates success, others indicate failure。
Remarks	Blocking interface, used to cancel navigation.

19.1.19 get_nav_task_status

Item	Content
Method Name	`get_nav_task_status`
Method Declaration	`tuple[Status, NavStatus] get_nav_task_status()`
Function Overview	Retrieves the current status information of the navigation task.
Return Value	`Status::OK` indicates success; others indicate failure. Returns a `NavStatus` object (empty on failure).
Remarks	This is a blocking interface for querying the navigation task status. Returns a `Status` code along with a `NavStatus` object, which can be used to check navigation progress and status.

19.1.20 subscribe_odometry

Item	Content
Method Name	`subscribe_odometry`
Method Declaration	`Status subscribe_odometry(callback: Callable[[Odometry], None])`
Function Overview	Subscribe to robot odometry data.
Parameters	`callback`: Callback function for receiving odometry data.
Return Value	`Status::OK` indicates success, others indicate failure.
Remarks	Non-blocking interface, used to receive real-time odometry updates.

19.1.21 unsubscribe_odometry

Item	Content
Method Name	`unsubscribe_odometry`
Method Declaration	`void unsubscribe_odometry()`
Function Overview	Unsubscribe from robot odometry data.
Return Value	None
Remarks	Non-blocking interface, used to cancel odometry data subscription.

19.1.22 get_point_cloud_map

Item	Content
Method Name	`get_point_cloud_map`
Method Declaration	`tuple[Status, PointCloud2] get_point_cloud_map(int timeout_ms)`
Function Overview	Get the currently built point cloud map.
Parameters	`timeout_ms`: Timeout in milliseconds, default is 5000.
Return Value	Returns a `Status` code and a `PointCloud2` object (empty object on failure).
Remarks	Blocking interface; returns an error status if not completed within the timeout.

19.2 Enumeration Type Definitions

19.2.1 NavMode —Navigation Mode Enumeration Type

Enum Value	Value	Description
`IDLE`	0	Idle mode
`GRID_MAP`	1	Grid map navigation mode

19.2.2 NavStatusType —Navigation Status Type Enumeration

Enum Value	Value	Description
`NONE`	0	No status
`RUNNING`	1	Running
`END_SUCCESS`	2	Successfully ended
`END_FAILED`	3	Failed to end
`PAUSE`	4	Paused
`CONTINUE`	5	Continue
`CANCEL`	6	Cancel

19.3 Data Structure Definitions

19.3.1 Pose3DEuler —3D Pose Structure

Field Name	Type	Description
`position`	`list[double`	Position (x, y, z), used to represent spatial position
`orientation`	`list[double`	Euler angles (roll, pitch, yaw), used to represent spatial orientation, avoiding Euler angle gimbal lock issues

19.3.2 NavTarget —Global Navigation Target Point Structure

Field Name	Type	Description
`id`	`int`	Target point ID
`frame_id`	`str`	Target point coordinate frame ID
`goal`	`Pose3DEuler`	Target point pose

19.3.3 NavStatus —Navigation Status Structure

Field Name	Type	Description
`id`	`int`	Target point ID, -1 indicates no target point
`status`	`NavStatusType`	Navigation status
`message`	`str`	Navigation status message

19.3.4 MapImageData —Mapping Image Data Structure, .pgm Format

Field Name	Type	Description
`type`	`str`	Magic number, “P5”：binary format
`width`	`int`	Image width
`height`	`int`	Image height
`max_gray_value`	`int`	Maximum gray value, 255
`image`	`list[int]`	Image data

19.3.5 MapMetaData —Mapping Map Metadata Structure

Field Name	Type	Description
`resolution`	`double`	Map resolution, unit: m/pixel
`origin`	`Pose3DEuler`	Map origin, position of world coordinate system origin relative to bottom-left corner of map
`map_image_dat`	`MapImage-Data`	Image data, .pgm format image data

19.3.6 MapInfo —Single Map Information Structure

Field Name	Type	Description
`map_name`	`str`	Map name
`map_meta_data`	`MapMetaData`	Map metadata

19.3.7 AllMapInfo —All Map Information Structure

Field Name	Type	Description
`current_map_name`	`str`	Current map name
`map_infos`	`list[MapInfo]`	All map information

19.3.8 LocalizationInfo —Current Position Information Structure

Field Name	Type	Description
`is_localization`	`bool`	Whether localized
`pose`	`Pose3DEuler`	Euler angle pose

19.4 SLAM Navigation Control Introduction

The robot’ s SLAM navigation control service can be divided into SLAM functions and navigation functions.

- In SLAM functions, corresponding interfaces can be called to implement mapping, localization, map management and other functions.
- In navigation functions, corresponding interfaces can be called to implement target navigation, navigation control and other functions.

19.4.1 Applicable Scenarios and Scope

- Static indoor flat areas smaller than 20 m × 20 m with rich features. Do not exceed the recommended range.
- This functionality is applicable to the education and research industry and is not recommended for commercial applications. For commercial use, please contact sales.

19.4.2 SLAM Function Flow

Function	Operation Flow
Mapping	Start mapping → Save map
Map Management	Load map, get map information, delete map, get map absolute path

19.4.3 Navigation Function Flow

Function	Operation Flow
Localization	Load map → Switch to localization mode → Initialize pose → Get localization status
Navigation	Localization success → Activate navigation mode → Set target pose (start navigation task) → Get navigation status
Navigation Control	Set target pose (start navigation task) → Pause navigation → Resume navigation → Cancel navigation

CHAPTER

TWENTY

DISPLAY CONTROL SERVICE (PYTHON)

Provides a display service controller for the robot system. Through DisplayController, you can use RPC to control robot display commands and obtain status.

20.1 API Definition

DisplayController is responsible for robot display control, including face expression control.

20.1.1 DisplayController —Display Controller

Item	Description
Class Name	`DisplayController`
Overview	Robot display control, including face expression control
Main Features	Face expression control
Use Cases	Face expression control

20.1.2 initialize

Item	Description
Function Name	`initialize`
Declaration	`bool initialize()`
Overview	Initialize the display controller.
Return Value	`true` for success, `false` for failure.
Note	Must be called before first use.

20.1.3 shutdown

Item	Description
Function Name	`shutdown`
Declaration	`void shutdown()`
Overview	Shut down the display controller.
Note	Used together with initialize.

20.1.4 get_all_face_expressions

Item	Description
Function Name	get_all_face_expressions
Declaration	`Status get_all_face_expressions(FaceExpressionVector data, int timeout_ms);`
Overview	Get all face expressions.
Parameters	`data`: Face expression data.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Get all face expressions.

20.1.5 set_face_expression

Item	Description
Function Name	set_face_expression
Declaration	`Status set_face_expression(int expression_id, int timeout_ms);`
Overview	Set face expression.
Parameters	`expression_id`: Face expression ID.`timeout_ms`: Timeout time (milliseconds), default is 5000.
Return Value	Operation status.
Note	Blocking interface. Set face expression.

20.1.6 get_current_face_expression

Item	Description
Function Name	get_current_face_expression
Declaration	`Status get_current_face_expression(FaceExpression data, int timeout_ms);`
Overview	Get current face expression.
Parameters	`data`: Current face expression.`timeout_ms`: Timeout time (milliseconds), default value is 5000.
Return Value	`Status::OK` for success, others for failure.
Note	Blocking interface. Get current face expression.

20.2 Data Structure Definitions

20.2.1 FaceExpression —Robot Face Expression Structure

Describes the complete information of a robot face expression, supporting unique ID, name, and description.

Field Name	Type	Description
`id`	`int`	Robot face expression id.
`name`	`str`	Robot face expression name.
`description`	`str`	Robot face expression description.

20.3 Notice:

When using subscription-type SDK interfaces such as Subscribe, do not perform blocking operations within the callback function. Doing so may cause message congestion, processing delays, or even unexpected errors.

CHAPTER

TWENTYONE

HIGH-LEVEL MOTION CONTROL EXAMPLE

This example demonstrates how to use the MagicDog SDK for initialization, robot connection, high-level motion control (gait, tricks, remote control), and other basic operations.

21.1 C++

Example file: `high_level_motion_example.cpp`

Reference documentation: `C++ API/high_level_motion_reference.md`

High-level motion control example:

```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;

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

(continues on next page)

(continued from previous page)

```
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;
```

(continues on next page)

(continued from previous page)

```
  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;
  }
}
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
        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) {
          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;
  }
```

(continues on next page)

(continued from previous page)

```
}

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;
  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;
```

(continues on next page)

(continued from previous page)

```
  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) {
    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;
      }
```

(continues on next page)

(continued from previous page)

```
    // 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':
      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();
```

(continues on next page)

(continued from previous page)

```
        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;
      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;
```

(continues on next page)

(continued from previous page)

```
  }

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

  return 0;
}
```

21.2 Python

Example file: `high_level_motion_example.py`

Reference documentation: `Python API/high_level_motion_reference.md`

High-level motion control example:

```
#!/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 math
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",
)

running = True
robot = None
high_controller = None
```

(continues on next page)

(continued from previous page)

```
# 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)
        if status.code != magicdog.ErrorCode.OK:
            logging.error(f"Failed to set force control standing: {status.message}")
```

(continues on next page)

(continued from previous page)

```
        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")
```

(continues on next page)

(continued from previous page)

```
        return True
    except Exception as e:
        logging.error(f"Error changing gait: {e}")
        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("Head Position Functions:")
    logging.info("  4        Function 4: Get head position")
    logging.info("  5        Function 5: Set head position")
    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
```

(continues on next page)

(continued from previous page)

```
    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 get_current_head_position(high_controller):
    status, euler_angles = high_controller.get_current_head_position()
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            "Failed to get current head position, code: %s, message: %s",
            status.code,
            status.message,
        )
        return

    logging.info("Head Position Roll Angle: %f", euler_angles.roll)
    logging.info("  Pitch Angle: %f", euler_angles.pitch)
    logging.info("  Yaw Angle: %f", euler_angles.yaw)


def set_head_position(high_controller, euler_angles):
    status = high_controller.set_head_position(euler_angles)
    if status.code != magicdog.ErrorCode.OK:
        logging.error(
            "Failed to set head position, code: %s, message: %s",
            status.code,
            status.message,
        )
        return


def deg2rad(deg):
    """Angle to radian"""
    return deg * math.pi / 180.0


def signal_handler(signum, frame):
    """Signal handler function for graceful exit"""
    global robot, running
    running = False
```

(continues on next page)

(continued from previous page)

```
    logging.info("Received interrupt signal (%s), exiting...", signum)
    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}")
```

(continues on next page)

(continued from previous page)

```
        robot.shutdown()
        return

    # Get high level motion controller
    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
```

(continues on next page)

(continued from previous page)

```
            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
            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
```

(continues on next page)

(continued from previous page)

```
            right_x = 0.0
            right_y = 0.0
            send_joystick_command(high_controller, left_x, left_y, right_x, right_y)
        # 3. Head Position Functions
        # 3.1 Get current head position
        elif key == "4":
            get_current_head_position(high_controller)
        # 3.2 Set head position
        elif key == "5":
            # Processing head position setting command
            try:
                # Get user input
                user_input = input(
                    "Please enter three angles (roll pitch yaw, separated by spaces, -
↪60 degrees to 60 degrees):"
                )
                values = user_input.split()
                if len(values) != 3:
                    logging.error("Error: Three numbers are required.")
                    return
                # Convert to floating point number
                roll_deg = float(values[0])
                pitch_deg = float(values[1])
                yaw_deg = float(values[2])
                # Verification angle range
                if (
                    not (-60 <= roll_deg <= 60)
                    or not (-60 <= pitch_deg <= 60)
                    or not (-60 <= yaw_deg <= 60)
                ):
                    logging.error("Warning: The angle should be in the range of -60␣
↪to 60 degrees.")
                # Convert to radian
                euler_angles = magicdog.EulerAngles()
                euler_angles.roll = deg2rad(roll_deg)
                euler_angles.pitch = deg2rad(pitch_deg)
                euler_angles.yaw = deg2rad(yaw_deg)
                # Output result
                logging.info("\nConversion result:")
                logging.info(f"Roll:  {roll_deg}° = {euler_angles.roll:.6f} rad")
```

(continues on next page)

(continued from previous page)

```
                logging.info(f"Pitch: {pitch_deg}° = {euler_angles.pitch:.6f} rad")
                logging.info(f"Yaw:   {yaw_deg}° = {euler_angles.yaw:.6f} rad")
                # Set the head position
                set_head_position(high_controller, euler_angles)
            except ValueError:
                logging.error("Error: Please enter a valid number.")
            except Exception as e:
                logging.error(f"An error occurred:{e}")
        # Help
        elif key.upper() == "?":
            print_help()
        else:
            logging.info(f"Unknown key: {key}")

    # Shutdown high level motion controller
    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 Running Instructions

21.3.1 Environment Setup:

```
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 Run Example:

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

21.3.3 Control Instructions:

- Gait and Trick Control:
  - `1`: Resume standing posture
  - `2`: Force-controlled standing
  - `3`: Execute trick action - Lie down
- Remote Control:
  - `w`: Move forward
  - `s`: Move backward
  - `a`: Move left
  - `d`: Move right
  - `t`: Turn left
  - `g`: Turn right
  - `x`: Stop movement
- Head Position Control:
  - `4`: Get head position
  - `5`: Set head position
- Other Functions:
  - `?`: Print help information

21.3.4 Stop the Program:

- Press `ESC` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYTWO

LOW-LEVEL MOTION CONTROL EXAMPLE

This example demonstrates how to use the MagicDog SDK to initialize, connect to the robot, and perform basic operations such as repeatedly standing up and squatting down.

22.1 C++

Example file: `low_level_motion_example.cpp`

Reference documentation: `C++ API/low_level_motion_reference.md`

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

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

(continues on next page)

(continued from previous page)

```
        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);
    if (status.code != ErrorCode::OK) {
```

(continues on next page)

(continued from previous page)

```
        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));
```

(continues on next page)

(continued from previous page)

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

    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
```

(continues on next page)

(continued from previous page)

```
};

LegJointCommand command;
int cnt = 0;

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

// 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;
    }
```

(continues on next page)

(continued from previous page)

```
        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++;
    }

    // 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

Example file: `low_level_motion_example.py`

Reference documentation: `Python API/low_level_motion_reference.md`

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

import sys
import time
import threading
import logging
from typing import Optional
```

(continues on next page)

(continued from previous page)

```
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")

    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
```

(continues on next page)

(continued from previous page)

```
    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")
    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
```

(continues on next page)

(continued from previous page)

```
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,
    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):
```

(continues on next page)

(continued from previous page)

```
                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()

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


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

22.3 Running Instructions

22.3.1 Environment Setup:

```
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 Run Example:

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

22.3.3 Stop the Program:

- Press `Ctrl+C` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYTHREE

SENSOR CONTROL EXAMPLE

This example demonstrates how to use the MagicDog SDK to initialize, connect to the robot, and perform basic operations such as robot sensor control (LiDAR, RGBD camera, binocular camera).

23.1 C++

Example file: `sensor_example.cpp`

Reference documentation: `C++ API/sensor_reference.md`

```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;

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

// Sensor manager class
class SensorManager {
```

(continues on next page)

(continued from previous page)

```
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;
    }

    // Sensor Open/Close
    bool open_laser_scan() {
```

(continues on next page)

(continued from previous page)

```
        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() {
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
↪std::endl;
            return false;
        }

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

    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();
        }
```

(continues on next page)

(continued from previous page)

```
        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;
        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)
↪{
```

(continues on next page)

(continued from previous page)

```
                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) {
                    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;
        }
    }
```

(continues on next page)

(continued from previous page)

```
    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;
            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) {
```

(continues on next page)

(continued from previous page)

```
                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();
            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;
        }
    }
```

(continues on next page)

(continued from previous page)

```
    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;
        }
    }

    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) {
```

(continues on next page)

(continued from previous page)

```
                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;
        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_
```

(continues on next page)

(continued from previous page)

```
↪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;
    std::cout << "\nSensor Control:" << std::endl;
```

(continues on next page)

(continued from previous page)

```
    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);
```

(continues on next page)

(continued from previous page)

```
    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;
    }

    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;
            }
```

(continues on next page)

(continued from previous page)

```
        // 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();
        } 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
```

(continues on next page)

(continued from previous page)

```
            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') {
                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();
```

(continues on next page)

(continued from previous page)

```
            } 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;
    }

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

    return 0;
}
```

23.2 Python

Example file: `sensor_example.py`

Reference documentation: `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
```

(continues on next page)

(continued from previous page)

```
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,
            "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 ===
```

(continues on next page)

(continued from previous page)

```
    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}")
            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}")
```

(continues on next page)

(continued from previous page)

```
            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"""
        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
```

(continues on next page)

(continued from previous page)

```
    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

        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
```

(continues on next page)

(continued from previous page)

```
    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)}")
            )
            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"]:
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
    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")
        else:
```

(continues on next page)

(continued from previous page)

```
            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:
```

(continues on next page)

(continued from previous page)

```
            self.sensor_controller.subscribe_left_binocular_low_img(
                lambda img: logging.info(f"Left Binocular Low: {len(img.data)} bytes")
            )
            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
```

(continues on next page)

(continued from previous page)

```
↪'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 '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(
```

(continues on next page)

(continued from previous page)

```
            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'}"
        )
        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")
```

(continues on next page)

(continued from previous page)

```
    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():
    """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:
```

(continues on next page)

(continued from previous page)

```
            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()
            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":
```

(continues on next page)

(continued from previous page)

```
                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()
            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"]:
```

(continues on next page)

(continued from previous page)

```
            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 Running Instructions

23.3.1 Environment Setup:

```
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 Run Example:

```
# C++
./sensor_example
# Python
python3 sensor_example.py
```

23.3.3 Control Instructions:

- Channel Control:
  - Key `1` - Open channel switch
  - Key `2` - Close channel switch
- Sensor Control:
  - Key `3` - Open laser scan
  - Key `4` - Close laser scan
  - Key `5` - Open RGBD camera
  - Key `6` - Close RGBD camera
  - Key `7` - Open binocular camera
  - Key `8` - Close binocular camera
- Basic Sensor Subscription:
  - Key `u/U` - Toggle/cancel ultrasonic sensor subscription
  - Key `h/H` - Toggle/cancel head touch sensor subscription
  - Key `i/I` - Toggle/cancel IMU sensor subscription
  - Key `l/L` - Toggle/cancel laser scan data subscription
- RGBD Camera Subscription:
  - Key `r/R` - Toggle/cancel RGBD color information
  - Key `d/D` - Toggle/cancel RGBD depth image
  - Key `c/C` - Toggle/cancel RGBD color image
  - Key `p/P` - Toggle/cancel RGB depth information
- Binocular Camera Subscription:
  - Key `b/B` - Toggle/cancel left eye high resolution image
  - Key `n/N` - Toggle/cancel left eye low resolution image
  - Key `m/M` - Toggle/cancel right eye low resolution image
  - Key `e/E` - Toggle/cancel depth image
- Other Functions:
  - Key `s` - Display status
  - Key `?` - Display help menu

23.3.4 Stop the Program:

- Press `ESC` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYFOUR

AUDIO CONTROL EXAMPLE

This example demonstrates how to use the MagicDog SDK to perform basic operations such as initialization, connecting to the robot, and audio control (getting volume, setting volume, TTS playback).

24.1 C++

Example file: `audio_example.cpp`

Reference documentation: `C++ API/audio_reference.md`

```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;

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

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
  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) {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "play tts failed, code: " << status.code
              << ", 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;
  }
```

(continues on next page)

(continued from previous page)

```
  std::cout << "close audio stream success" << std::endl;
}

// Global counters for audio stream callbacks
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();
```

(continues on next page)

(continued from previous page)

```
  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) {
    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;
  }
```

(continues on next page)

(continued from previous page)

```
  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;
}

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
```

(continues on next page)

(continued from previous page)

```
  // 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): ";
    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)
```

(continues on next page)

(continued from previous page)

```
  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;
  }

  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': {
```

(continues on next page)

(continued from previous page)

```
        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':
        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': {
```

(continues on next page)

(continued from previous page)

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

  return 0;
}
```

24.2 Python

Example file: `audio_example.py`

Reference documentation: `Python API/audio_reference.md`

```
#!/usr/bin/env python3
```

(continues on next page)

(continued from previous page)

```
# -*- 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"""
    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")
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
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):
    """Open audio stream"""
```

(continues on next page)

(continued from previous page)

```
    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)}")
```

(continues on next page)

(continued from previous page)

```
            sys.stdout.write("\r")
            sys.stdout.flush()
        bf_counter[0] += 1

    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}
```

(continues on next page)

(continued from previous page)

```
↪")
    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(
            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)
```

(continues on next page)

(continued from previous page)

```
    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()

    # 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():
```

(continues on next page)

(continued from previous page)

```
    """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)...")

    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
```

(continues on next page)

(continued from previous page)

```
        # 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":
            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":
```

(continues on next page)

(continued from previous page)

```
            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")

    return 0


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

24.3 Running Instructions

24.3.1 Environment Setup:

```
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 Run Example:

```
# C++
./audio_example
# Python
python3 audio_example.py
```

24.3.3 Control Instructions:

- Audio Functions:
  - Key `1` - Get volume
  - Key `2` - Set volume
  - Key `3` - Play TTS synthesis
  - Key `4` - Stop playback
  - Key `5` - Get audio configuration
- Audio Stream Functions:
  - Key `6` - Open audio stream
  - Key `7` - Close audio stream
  - Key `8` - Subscribe to audio stream
  - Key `9` - Unsubscribe from audio stream
- Speech IO Functions:
  - Key `d` - Open speech io
  - Key `e` - Close speech io
  - Key `f` - Subscribe to speech asr
  - Key `g` - Unsubscribe from speech asr
  - Key `h` - Publish speech tts
- Other Functions:
  - Key `?` - Display help menu
  - Key `ESC` - Exit program

24.3.4 Stop the Program:

- Press `ESC` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYFIVE

STATE MONITOR EXAMPLE

This example demonstrates how to use the MagicDog SDK for initialization, connecting to the robot, and basic operations such as robot state monitoring (faults, BMS).

25.1 C++

Example file: `monitor_example.cpp`

Reference documentation: `C++ API/monitor_reference.md`

```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;

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

(continues on next page)

(continued from previous page)

```
    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
              << ", State: " << static_cast<int>(robot_state.bms_data.battery_state)
              << ", Power supply status: " << static_cast<int>(robot_state.bms_data.
```

(continues on next page)

(continued from previous page)

```
↪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

Example file: `monitor_example.py`

Reference documentation: `Python API/monitor_reference.md`

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

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

(continues on next page)

(continued from previous page)

```
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")

    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}")
```

(continues on next page)

(continued from previous page)

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

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


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

25.3 Running Instructions

25.3.1 Environment Setup:

```
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 Run Example:

```
# C++
./monitor_example
# Python
python3 monitor_example.py
```

25.3.3 Stop the Program:

- Press `Ctrl+C` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYSIX

SLAM NAVIGATION EXAMPLE

This example demonstrates how to use the MagicDog SDK for SLAM mapping, localization, navigation and other operations, including map management, pose acquisition, path planning and other functions.

26.1 C++

Example file:

- `slam_example.cpp`
- `navigation_example.cpp`

Reference documentation: `C++ API/slam_navigation_reference.md`

26.1.1 Slam Mapping Example

```cpp
#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>
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <sstream>
#include <string>
```

(continues on next page)

(continued from previous page)

```
#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;
  std::cout << "Joystick Functions:" << std::endl;
  std::cout << "  W        Function W: forward" << std::endl;
  std::cout << "  A        Function A: backward" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  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();
```

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
    if (current_slam_mode != "MAPPING") {
      std::cout << "Warning: Currently not in mapping mode, may not be able to save␣
↪map" << std::endl;
    }

    // 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
```

(continues on next page)

(continued from previous page)

```
    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;
  } 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;
```

(continues on next page)

(continued from previous page)

```
        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 <
↪< 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;
```

(continues on next page)

(continued from previous page)

```
  }
}

// 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);
      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;
```

(continues on next page)

(continued from previous page)

```
  }
}

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;
  }
}

// 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;
  }
}
```

(continues on next page)

(continued from previous page)

```
// 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();
  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) {
```

(continues on next page)

(continued from previous page)

```
    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()) {
    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();
```

(continues on next page)

(continued from previous page)

```
    }
    // 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);
    }
    // 2.5 Delete map
    else if (key == '6') {
      std::string strInput;
      std::cout << "Enter parameters: ";
      std::getline(std::cin, strInput);
      if (strInput.empty()) {
        continue;
```

(continues on next page)

(continued from previous page)

```
        }
        // 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
      }
      // 4.6 Turn right
      else if (key == 'g' || key == 'G') {
        sendJoystickCommand(0.0, 0.0, 1.0, 0.0);  // Turn right
```

(continues on next page)

(continued from previous page)

```
        }
        // 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
```

(continues on next page)

(continued from previous page)

```
    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 Slam Navigation Example

```
#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";
```

(continues on next page)

(continued from previous page)

```
NavMode current_nav_mode = NavMode::IDLE;
int odometry_counter = 0;

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 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;
```

(continues on next page)

(continued from previous page)

```
  std::cout << "  ?        Function ?: Print help" << std::endl;
  std::cout << "  ESC      Exit program" << std::endl;
}

// ==================== Navigation Functions ====================

// 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;
  }
}

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;
```

(continues on next page)

(continued from previous page)

```
    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();

    // 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
```

(continues on next page)

(continued from previous page)

```
    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] << "]
↪" << 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;
```

(continues on next page)

(continued from previous page)

```
    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
    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;
```

(continues on next page)

(continued from previous page)

```
    }

    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) {
      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();
```

(continues on next page)

(continued from previous page)

```
    // 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;
    }

    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();
```

(continues on next page)

(continued from previous page)

```
    // 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";
        break;
      default:
        statusMeaning = "Unknown status value";
        break;
    }

    std::cout << "Status meaning: " << statusMeaning << std::endl;
    std::cout << "========================" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  } 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) {
      std::cerr << "Failed to open odometry stream, code: " << status.code
                << ", message: " << status.message << std::endl;
      return;
    }
```

(continues on next page)

(continued from previous page)

```
    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] << ", "
```

(continues on next page)

(continued from previous page)

```
                  << data->orientation[1] << ", " << data->orientation[2] << ", "
                  << 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();
```

(continues on next page)

(continued from previous page)

```
    // 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;
  }
}

// ==================== 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();
```

(continues on next page)

(continued from previous page)

```
    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()) {
    try {
      std::string strInput = getUserInput();

      // Split input parameters by space
```

(continues on next page)

(continued from previous page)

```
        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];
        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
```

(continues on next page)

(continued from previous page)

```
        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);
        }
        // 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") {
```

(continues on next page)

(continued from previous page)

```
          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

      } 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;
```

(continues on next page)

(continued from previous page)

```
    // 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

Example file:

- `slam_example.py`
- `navigation_example.py`

Reference documentation: `Python API/slam_navigation_reference.md`

26.2.1 Slam Mapping Example

```
#!/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
```

(continues on next page)

(continued from previous page)

```
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"""
    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")
```

(continues on next page)

(continued from previous page)

```
    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()

        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
```

(continues on next page)

(continued from previous page)

```
    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(
                "Failed to cancel mapping, code: %s, message: %s",
                status.code,
                status.message,
```

(continues on next page)

(continued from previous page)

```
            )
            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)
```

(continues on next page)

(continued from previous page)

```
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",
```

(continues on next page)

(continued from previous page)

```
                status.code,
                status.message,
            )
            return

        logging.info("Successfully retrieved map information")
        logging.info("Current map: %s", all_map_info.current_map_name)
        logging.info("Total maps: %d", len(all_map_info.map_infos))

        if all_map_info.map_infos:
            logging.info("Map details:")
            for i, map_info in enumerate(all_map_info.map_infos):
                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
                )
```

(continues on next page)

(continued from previous page)

```
                save_map_image_to_file(map_info)

        else:
            logging.info("No available maps")
    except Exception as e:
        logging.error("Exception occurred while getting map information: %s", e)


def save_map_image_to_file(map_info):
    """Save map image to current directory"""
    try:
        # Extract image data
        map_data = map_info.map_meta_data.map_image_data
        width = map_data.width
        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)
```

(continues on next page)

(continued from previous page)

```
            else:
                image_bytes_data = image_bytes
        except Exception as e:
            logging.error("Failed to convert image data to bytes: %s", e)
            return

        # Create numpy array from image data
        image_array = np.frombuffer(image_bytes_data, dtype=np.uint8).reshape(
            (height, width)
        )

        # Generate filename based on map name
        safe_filename = "".join(
            c for c in map_info.map_name if c.isalnum() or c in (" ", "-", "_")
        ).rstrip()
        safe_filename = safe_filename.replace(" ", "_")
        if not safe_filename:
            safe_filename = f"map_{int(time.time())}"

        # 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
```

(continues on next page)

(continued from previous page)

```
    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)


# 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
```

(continues on next page)

(continued from previous page)

```
    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}")


# 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)
```

(continues on next page)

(continued from previous page)

```
        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
    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,
```

(continues on next page)

(continued from previous page)

```
            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

    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)
```

(continues on next page)

(continued from previous page)

```
            # 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()
                # 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
```

(continues on next page)

(continued from previous page)

```
                    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
                    )
                # 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
```

(continues on next page)

(continued from previous page)

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

    finally:
        # Clean up resources
        try:
```

(continues on next page)

(continued from previous page)

```
            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 Slam Navigation Example

```
#!/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

import magicdog_python as magicdog

# Configure logging format and level
logging.basicConfig(
```

(continues on next page)

(continued from previous page)

```
    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")
    logging.info("  6        Function 6: Set navigation target goal")
```

(continues on next page)

(continued from previous page)

```
    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:
```

(continues on next page)

(continued from previous page)

```
        logging.error(f"Error setting down climb stairs gait: {e}")


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
```

(continues on next page)

(continued from previous page)

```
        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:
            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",
```

(continues on next page)

(continued from previous page)

```
            "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]",
            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:
```

(continues on next page)

(continued from previous page)

```
        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")

        # 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],
```

(continues on next page)

(continued from previous page)

```
            target_goal.goal.position[1],
            target_goal.goal.position[2],
            target_goal.goal.orientation[0],
            target_goal.goal.orientation[1],
            target_goal.goal.orientation[2],
        )
    except Exception as e:
        logging.error("Exception occurred while setting navigation target: %s", e)


def pause_navigation():
    """Pause navigation"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Pause navigation
        status = controller.pause_nav_task()
        if status.code != 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
```

(continues on next page)

(continued from previous page)

```
        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
        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:
```

(continues on next page)

(continued from previous page)

```
        # 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",
            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():
```

(continues on next page)

(continued from previous page)

```
    """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
    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:
```

(continues on next page)

(continued from previous page)

```
        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:
        # 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],
```

(continues on next page)

(continued from previous page)

```
                    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():
    """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()
```

(continues on next page)

(continued from previous page)

```
        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)


# ==================== Utility Functions ====================


def get_user_input():
    """Get user input - Read a line of data"""
    try:
```

(continues on next page)

(continued from previous page)

```
        # 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()
            return -1

        logging.info("Successfully connected to robot")
```

(continues on next page)

(continued from previous page)

```
        # 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
```

(continues on next page)

(continued from previous page)

```
                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()
```

(continues on next page)

(continued from previous page)

```
                # 3.6 Get navigation status
                elif key.upper() == "0":
                    get_navigation_status()
                # 4. Odometry Functions
                # 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()
```

(continues on next page)

(continued from previous page)

```
            logging.info("SLAM navigation controller closed")

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

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

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


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

26.3 Running Instructions

26.3.1 Environment Setup:

```
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 Running Examples:

```
# C++
./slam_example
./navigation_example
# Python
python3 slam_example.py
python3 navigation_example.py
```

26.3.3 Control Instructions:

Mapping Example:

- Preparation:
  - Key `1` - Recover standing

- SLAM Functions:
  - Key `2` - Start mapping
  - Key `3` - Cancel mapping
  - Key `4` - Save map
  - Key `5` - Load map
  - Key `6` - Delete map
  - Key `7` - Get all map info and save as PGM file
- Joystick Control:
  - Key `W` - Move forward
  - Key `A` - Move backward
  - Key `S` - Move left
  - Key `D` - Move right
  - Key `T` - Turn left
  - Key `G` - Turn right
  - Key `X` - Stop
- Close Functions:
  - Key `P` - Close SLAM
- System Commands:
  - Key `?` - Show help info

Navigation:

- Preparation:
  - Key `1` - Recover standing
- Localization Functions:
  - Key `2` - Switch to localization mode
  - Key `4` - Initialize pose
  - Key `5` - Get current position info
- Navigation Functions:
  - Key `3` - Switch to navigation mode
  - Key `6` - Set navigation target
  - Key `7` - Pause navigation

  - Key `8` - Resume navigation
  - Key `9` - Cancel navigation
  - Key `0` - Get navigation status
- Odometry Functions:
  - Key `C` - Subscribe to odometry stream
  - Key `V` - Unsubscribe from odometry stream
- Close Functions:
  - Key `L` - Close navigation
  - Key `P` - Close SLAM
- System Commands:
  - Key `?` - Show help info

26.3.4 Stop Program:

- Press `ESC` to safely stop the program
- The program will automatically clean up all resources and save current state

26.3.5 Important Notes:

- For mapping, you must switch to Recovery_stand mode first. You can either push the robot in Recovery_stand mode or use remote control in Balance_stand mode to create maps.
- You can use `GetAllMapInfo` to get information about maps stored on the robot (including 2D PGM maps, map size, resolution, etc.).
- Use `SaveMap`/`DeleteMap` for map management. Try to keep no more than 3 maps stored on the robot.
- For navigation, switch to Balance_stand mode first.
- Navigation targets should be set based on current localization status and pose obtained from `GetCurrentLocalizationInfo`.
- Before starting navigation tasks, properly switch to localization mode (`SwitchToLocation()`) and navigation mode (`ActivateNavMode(NavMode::GRID_MAP)`).
- Both localization and navigation mode switches require absolute map paths, which can be obtained using `GetMapPath`.
- After finishing mapping or navigation, switch SLAM to IDLE mode (`SwitchToIdle()`) and navigation to IDLE mode (`ActivateNavMode(NavMode::IDLE)`).

CHAPTER

TWENTYSEVEN

DISPLAY CONTROL EXAMPLE

This example demonstrates how to use the MagicDog SDK to perform basic operations such as initialization, connecting to the robot, and display control.

27.1 C++

Example file: `display_example.cpp`

Reference documentation: `C++ API/display_reference.md`

```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;

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

(continues on next page)

(continued from previous page)

```
  }
  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;
  }
```

(continues on next page)

(continued from previous page)

```
  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)
```

(continues on next page)

(continued from previous page)

```
  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;
  }

  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':{
```

(continues on next page)

(continued from previous page)

```
        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':
        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

Example file: `display_example.py`

Reference documentation: `Python API/display_reference.md`

```python
#!/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

# 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:")
```

(continues on next page)

(continued from previous page)

```
    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)
    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"""
```

(continues on next page)

(continued from previous page)

```
    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,
        )
        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):
```

(continues on next page)

(continued from previous page)

```
        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
            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
```

(continues on next page)

(continued from previous page)

```
        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


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

27.3 Running Instructions

27.3.1 Environment Setup:

```
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 Run Example:

```
# C++
./display_example
# Python
```

(continues on next page)

(continued from previous page)

```
python3 display_example.py
```

27.3.3 Control Instructions:

- Face expression Functions:
  - Key `1` - Get all face expressions
  - Key `2` - Set face expression
  - Key `3` - Get current face expression
- Other Functions:
  - Key `?` - Display help menu
  - Key `ESC` - Exit program

27.3.4 Stop the Program:

- Press `ESC` to safely stop the program
- The program will automatically clean up all resources

CHAPTER

TWENTYEIGHT

ROS2 SDK USER GUIDE

28.1 SDK Introduction

MagicDog_sdk is a software development kit created by MagicLab Robotics for the new generation of robots. The SDK encapsulates interfaces such as low-level motor control, high-level motion control, lidar point cloud data, audio and video streaming, SLAM, odometry, and more, providing related function interfaces. You can follow the interfaces and examples we provide to perform secondary development for the quadruped robot.

> Note: The ROS2 SDK will no longer be updated. For additional development, it is recommended to use the C++ API or Python API.

28.2 SDK Support

Robot Model	Support Status	Remarks
MagicDog EDU Edition	✓ Supported	Can use this SDK for secondary development
Other models or versions	✗ Not supported	Not supported for secondary development using this SDK

Note: The ROS2 SDK will no longer be updated. For further development, it is recommended to use the C++ API or Python API.

28.3 SDK Download Address

Type	Address
GitHub	https://github.com/MagiclabRobotics/magicdog_ros2_sdk

28.4 Environment Preparation

28.4.1 Install ROS 2

ROS 2 Version	Ubuntu Version	Recommendation
ROS 2 Foxy	Ubuntu 20.04	✅ Recommended
ROS 2 Iron	Ubuntu 22.04	✅ Supported

28.4.2 Install cyclone-dds

```
sudo apt install ros-foxy-rmw-cyclonedx-cpp
```

28.5 Build Steps

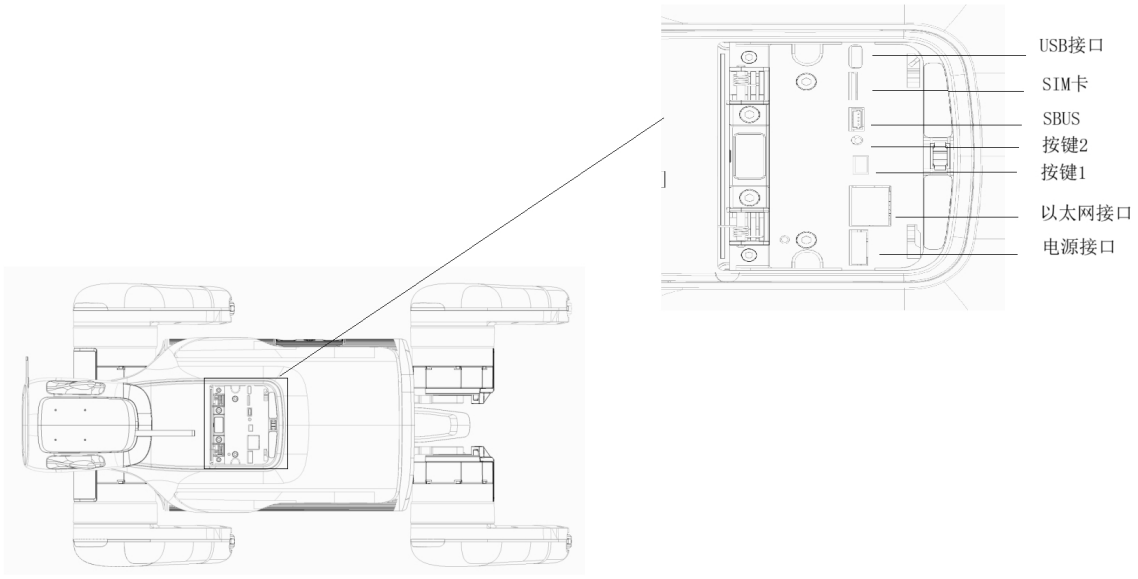
Step	Command
1. Create workspace	`mkdir -p ~/ros2_ws/src && cd ~/ros2_ws/src`
2. Clone the code	`git clone https://github.com/MagiclabRobotics/magicdog_ros2_sdk.git`
3. Install dependencies	`cd ~/ros2_ws && rosdep install -i --from-path src --rosdistro foxy -y`
4. Build	`colcon build`
5. Source environment	`source install/setup.bash`

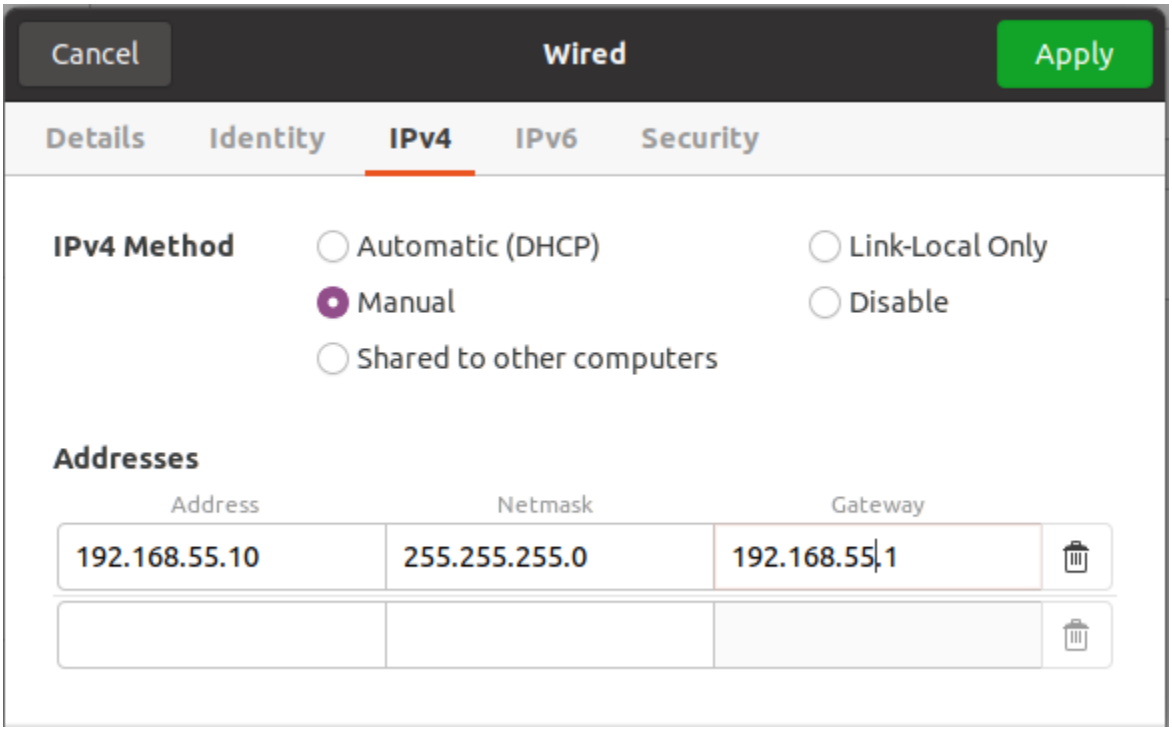
28.6 Connection Configuration

28.6.1 Connecting a Personal Computer to the Robot

Step	Instructions
1. Turn off services	Disable WiFi and Docker on your PC
2. Check Docker	`ifconfig
3. Disable Docker network	`sudo ifconfig docker0 down`
4. Physical connection	Connect your PC and the robot using a data cable

28.6.2 Port Description for Connection





28.7 Environment Configuration

Configuration Type	Command
Temporary	`export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp`
Temporary	`export ROS_DOMAIN_ID=25`
Permanent	`echo "export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp" >> ~/.bashrc`
Permanent	`echo "export ROS_DOMAIN_ID=25" >> ~/.bashrc`

28.8 Connection Verification

Use the ROS 2 command below to verify whether multi-machine communication is successfully set up:

```
ros2 node list
```

Note: Remember to source the setup file for each workspace.

CHAPTER

TWENTYNINE

QUADRUPED ROBOT ROS2 SDK DOCUMENTATION

This documentation provides ROS2 API services for the robotic system. Interfaces via ROS2 topics and services enable sensory data acquisition, voice control, camera management, navigation, and other functions.

Note: The ROS2 SDK will not be updated further. For secondary development, it is recommended to use the C++ API/Python API.

29.1 Documentation Overview

This manual is intended for robot integrators and engineers. It systematically introduces the features and interfaces of the Quadruped Robot ROS2 SDK, facilitating secondary development and integration on the Ubuntu+ROS2 platform.

Item	Description
Target Users	Robot integrators, industry developers, researchers
Main Content	System architecture, communication protocol, development environment setup, interface description, practical examples, FAQs
Application Scenarios	Factory inspection, educational research, customized applications, etc.

29.2 System Architecture

The quadruped robot system consists of underlying hardware (main controller, actuators, sensors), embedded software (firmware), middleware (ROS2 + DDS), and upper-level applications.

Component	Function Description
Main Control Board	Core computation and scheduling unit, runs ROS2
Sensors	Includes TOF, ultrasonic, IMU, fisheye camera, LiDAR, etc.
Actuators	For motion and interaction (such as motors, servos, display, etc.)
Host Computer/External Applications	Interact, remotely debug, and monitor the robot via wired or Wi-Fi connection to the main controller

29.3 Communication Protocol

All SDK communications are based on the distributed architecture of ROS2. DDS (Data Distribution Service) middleware ensures efficient and reliable message publish/subscribe and service calls.

Communication Type	Application Scenario
Topic	For sensor data, status info, and other data flows suitable for publish–subscribe pattern
Service	For parameter configuration, control commands, trigger operations, and other one-to-one request–response transactions
Action	For complex tasks needing feedback and result notification (e.g., navigation, action execution)
QoS	Configurable as required to ensure real-time performance and reliability

29.4 Development Environment

Requirement	Description
Operating System	Ubuntu 20.04 LTS (Long Term Support)
ROS2 Version	Compatible with Foxy/Galactic/Humble; Foxy and above recommended
Basic Dependencies	Python3, C++ build environment, colcon, and other build tools
Network Setup	Wired connection recommended for multicast and large data flows

29.5 Environment Setup

29.5.1 SDK Installation & Node Configuration

Step	Instruction
1. Upload SDK	Upload the SDK source code or installation package to the main controller or development PC
2. Build	Enter workspace, run `colcon build` to build
3. Environment Configuration	Source the workspace setup script, e.g. `source install/setup.bash`

29.5.2 ROS2 Environment Variables

```
export RMW_IMPLEMENTATION=rmw_cyclonedx-cpp
export ROS_DOMAIN_ID=25  # For multiple robots in the same network, each needs a␣
↪unique domain_id
```

29.6 ROS2 Interfaces & Examples

29.6.1 5.1 Sensor Interfaces

5.1.1 TOF Output

Time-of-Flight (TOF) sensor array outputs distances for the front and underside in real time, commonly used for obstacle avoidance and mapping.

Topic	Message Type	Index	Location
`/tof`	`std_msgs/Float32MultiArray`	0	Front Left
`/tof`	`std_msgs/Float32MultiArray`	1	Front Right
`/tof`	`std_msgs/Float32MultiArray`	2	Underside

5.1.2 Ultrasonic Output

Covers low-speed obstacle avoidance scenarios at the sides and front.

Topic	Message Type	Index	Location
`/ultra`	`std_msgs/Float32MultiArray`	0	Left
`/ultra`	`std_msgs/Float32MultiArray`	1	Front
`/ultra`	`std_msgs/Float32MultiArray`	2	Right

5.1.3 Head Touch Sensor

For human-computer interaction, can distinguish multiple gesture events.

Topic	Message Type	Value	Gesture Type
/head_touch_state	std_msgs/Int8	0	Not Triggered
/head_touch_state	std_msgs/Int8	1	Single Tap
/head_touch_state	std_msgs/Int8	2	Double Tap
/head_touch_state	std_msgs/Int8	3	Swipe Up
/head_touch_state	std_msgs/Int8	4	Swipe Down
/head_touch_state	std_msgs/Int8	5	Long Press
/head_touch_state	std_msgs/Int8	16	Error

5.1.4 Battery Information

Monitor battery status, including voltage, current, temperature, SOC, etc., in real-time to ensure system safety.

Topic	Message Type	Main Fields
/motion/bms_dat	motion_msgs BMSRespon	batt_volt (Voltage) batt_curr (Current) batt_temp (Temperature) batt_soc (Remaining Capacity) status (Power Status) key (Shutdown Signal) batt_health (Health) batt_loop_number (Number of Cycles) power_board_status (Power Board Fault)

29.6.2 5.2 Voice Interface

This section introduces how to call voice-related features via ROS2 services, including volume query, volume setting, speech recognition, and TTS control.

Function	Service Call
Get Volume	ros2 service call /voice/get_volume voice_msgs/srv/GetVolume "{}"
Set Volume	ros2 service call /voice/set_volume voice_msgs/srv/SetVolume "{volume: 7}"
Query Speech Recognition Status	ros2 service call /voice_config_get voice_msgs/srv/GetConfig "{}"
Set Speech Recognition	ros2 service call /voice_config voice_msgs/srv/Config "{enable_voice: true}"
Stop TTS	ros2 service call /voice/stop_tts std_srvs/srv/Trigger "{}"
Play TTS	ros2 service call /voice/set_tts voice_msgs/srv/SetTTS "{tts_id: 'id_1', content: 'Hello', priority: 1, mode: 0}"

29.6.3 5.3 Camera Interface

This section introduces interfaces for various cameras (such as main camera, fisheye, stereo, etc.), including enable/disable, parameter configuration, and data stream access.

Topic Name	Message Type	Description
/sensor_msgs	sensor_msgs::msg::Image	Main camera raw image
/depth	sensor_msgs::msg::Image + /depth/camera_info	Depth image + camera intrinsic
/color	sensor_msgs::msg::Image + /color/camera_info	Color image + camera intrinsic

29.6.4 5.4 Navigation Interface

Navigation-related interfaces include various services and topics for mode switching, goal setting, velocity command, status feedback, and more.

Function	Service Call
Switch Navigation Mode	ros2 service call /switch_navigate_mode pp_msgs/srv/ChangeNavigateMode "{set_mode: 0}"
Point Cloud Conversion	ros2 service call /cloud_processor/switch_statue std_srvs/srv/SetBool "data: true"
Velocity Command	ros2 topic pub -r 10 /vel_app motion_msgs/msg/VelCommand "{...}"
Global Goal Point	ros2 topic pub -1 /goal_pose geometry_msgs/msg/PoseStamped "{...}"

29.6.5 5.5 LiDAR Interface

This section introduces how to start/stop scanning, perform mapping, and localization via ROS2 nodes.

Function	Service Call
Start/Stop LiDAR	ros2 service call /lds/enable std_srvs/srv/SetBool "data: true"
Start Mapping	ros2 service call /set_slam_model cartographer_ros_msgs/srv/SetSlamModel "{slam_model:1,path:'/home/eame/cust_para/maps/123'}"
Save Map	ros2 service call /save_map cartographer_ros_msgs/srv/SaveMap "{path:'/home/eame/maps/test'}"
Switch Localization Mode	ros2 service call /set_slam_model cartographer_ros_msgs/srv/SetSlamModel "{slam_model:3,path:'/home/eame/cust_para/maps/test'}"

29.7 Common Issues & Troubleshooting

Issue Type	Solution
No Interface Response	Check topic/service names, parameter format, and if related nodes are running properly
Data Abnormality	Use `ros2 topic echo` to debug raw output, check for abnormal values (e.g., nan, inf)
Network Packet Loss	Prefer wired connection, or adjust QoS policies to improve reliability
Permission/Authentication Failure	Ensure the current user has sufficient permission to access hardware and serial ports

© 2025 MagicLab Robotics

CHAPTER

THIRTY

FAQ

When the SDK becomes unavailable, first verify whether the SDK version meets the requirements. The SDK version tag must correspond to the robot firmware version. Refer to ChangeLog

30.1 Development Environment Preparation

Operating System: Ubuntu 22.04. For detailed setup, please refer to Quick Start.

30.2 Does MagicDog support wireless development?

No. MagicDog currently supports only wired connections for development.

30.3 What is the current low-level communication frequency of MagicDog?

The default reporting frequency for joint status communication at the low level is 500 Hz. The publishing frequency(Suggested frequency: 500 Hz) of joint command transmission is controlled by the user.

30.4 SDK API Call Error: Deadline Exceeded

The default timeout for internal RPC calls within the SDK APIs is 5 seconds. For certain gait or skill execution APIs, the execution time may exceed 5 seconds. If this issue occurs, you can resolve it by configuring the timeout parameter for the specific API.

30.5 How to View SDK Internal Configuration and Logs?

When the SDK program is executed, it will by default generate a configuration file at `/tmp/magicdog_mjrrt.yaml` and a log file at `/tmp/logs/magicdog_sdk.log`. Any internal error messages of the SDK can be checked in this log file.

30.6 报错：Failed to disable SDK.

Check if multicast is properly configured

```
ping 192.168.55.200 # Confirm network card configuration is correct and connection to␣
↪robot dog is normal
ifconfig # Confirm the ethercat network card name directly connected to the robot dog
sudo ifconfig <ethercat network card name directly connected to robot dog> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <ethercat network card name␣
↪directly connected to robot dog>
```

30.7 When calling `high_level_motion_example`, the following error is printed: `Fail initializing after {} seconds. \nPlease check the lcm connection between remote and hardware PC then try again.\n`

Check if multicast is properly configured

```
ping 192.168.55.200 # Confirm network card configuration is correct and connection to␣
↪robot dog is normal
ifconfig # Confirm the ethercat network card name directly connected to the robot dog
sudo ifconfig <ethercat network card name directly connected to robot dog> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <ethercat network card name␣
↪directly connected to robot dog>
```

30.8 Unable to get corresponding data when calling `audio_example` or `sensor_example`

Check if multicast is properly configured

```
ping 192.168.55.200 # Confirm network card configuration is correct and connection to␣
↪robot dog is normal
ifconfig # Confirm the ethercat network card name directly connected to the robot dog
sudo ifconfig <ethercat network card name directly connected to robot dog> multicast
sudo route add -net 224.0.0.0 netmask 240.0.0.0 dev <ethercat network card name␣
↪directly connected to robot dog>
```

30.9 How to check binocular camera image delay

The `header.stamp` field of the `CompressedImage` or `Image` data structure records the image transmission time, unit: nanoseconds

30.10 How to get fisheye camera or 4K camera data

```
# Fisheye camera:
ffplay rtsp://192.168.12.1:8080/
# Central 4K camera:
ffplay rtsp://192.168.12.1:8082/
```

You can use `ffplay` for testing in demonstrations. If you want to get each frame image, you need to code it yourself

30.11 SLAM navigation not executing?

1. First prerequisite for SLAM navigation - enter SLAM localization mode and navigation mode:

- `SwitchToLocation()`
- `ActivateNavMode(NavMode::GRID_MAP)`

2. Second prerequisite for SLAM navigation - successful relocation:

Use `InitPose` for relocation and call `GetCurrentLocalizationInfo` to get current relocation status. If `is_localization=true`, relocation is successful; otherwise, relocation failed - check map and relocation pose, then retry `InitPose` for localization.

3. Setting SLAM navigation target points:

Since the relocation pose set by `InitPose` is estimated, it is recommended to set target position points based on the current pose obtained from `GetCurrentLocalizationInfo`.

30.12 Runtime Error: Invalid IP address

The current robot does not support simultaneous control by an APP and SDK, nor does it support multiple SDK clients controlling the robot at the same time. If an `Invalid IP address` error occurs, please check if there are other APPs or SDK clients connected.

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 Unable to Subscribe and Publish Topic Data?

- Check whether the corresponding topic data stream is enabled.

  Some interfaces require calling a specific function to activate the data stream, such as calling `OpenLaserScan` to enable LiDAR data.

- UDP multicast configuration:

  Assuming the network interface connecting the SDK development PC and the robot is enp0s31f6, configure it as follows to ensure communication between the SDK and the robot at the low level:

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

- SDK version does not match the robot firmware version:

  The SDK version tag must correspond to the robot’s firmware version. For reference, see: ChangeLog.

- Clean residual SDK configuration files:

  The SDK automatically generates its default configuration at `/tmp/magicdog_mjrrt.yaml`. Under normal circumstances, this file is cleared on reboot. However, if during a single boot you first use an older version of the SDK (which generates an old configuration file in `/tmp`), and then switch to a newer SDK version, the newer SDK will still load the old configuration by default. This is because if the `magicdog_mjrrt.yaml` file already exists, the SDK will not recreate it, in order to facilitate debugging. As a result, topic subscription and publication may behave incorrectly.

Finally, please note that when using subscription-type SDK interfaces such as Subscribe, avoid performing blocking operations within the callback function. Otherwise, it may lead to message backlog, processing delays, or even unexpected errors.

30.15 SDK subscription topic data rate unstable?

Check the system configuration of the socket receive buffer on the SDK host in `/etc/sysctl.conf`.Apply the changes immediately using `sysctl -p`, or reboot the host.

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

30.16 Log Error: `Call of pthread_setschedparam with insufficient privileges!`

The main reason for this error is that the SDK is being executed under a regular user account. This warning does not fundamentally affect program operation. To resolve this error, you need to configure the system file `/etc/security/limits.conf` under the regular user to ensure the process has permission to set thread priorities:

```
*    -   rtprio   98
```

30.17 connect Error: `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. Cannot ping `192.168.55.200`:
   - Network cable connection problem, check hardware connections.
   - The compute box IP configuration is not in the `192.168.55.XX` subnet.
2. Robot gateway service exception:
   - Try restarting to restore normal robot software operation.