

---

# Magicbot-Gen1 SDK Development Documentation

*Release 1.2.2-doc2*

**MagicLab**

**Dec 17, 2025**



---

## Contents

---

<b>1</b>	<b>OverView</b>	<b>1</b>
1.1	Product SKUs . . . . .	1
1.2	Electrical interface . . . . .	4
1.3	Onboard computer . . . . .	4
1.4	Joint sequence names and joint limits . . . . .	5
1.5	LiDAR . . . . .	6
1.6	Depth Camera . . . . .	6
1.7	Robot specifications . . . . .	6
<b>2</b>	<b>Remote control button description</b>	<b>7</b>
2.1	Functional Description . . . . .	7
2.2	Mode Switching . . . . .	8
2.3	Remote control check . . . . .	8
2.4	Remote control on and off . . . . .	9
2.5	Remote control operation during power on . . . . .	10
2.6	Walking mode switch . . . . .	10
2.7	Stop and return to standing position . . . . .	10
2.8	Emergency Stop . . . . .	11
2.9	Remote control operation when power off . . . . .	11
<b>3</b>	<b>SDK Overview</b>	<b>13</b>
3.1	SDK Communication Interface Introduction . . . . .	13
3.2	Getting the SDK . . . . .	14
<b>4</b>	<b>Software and Hardware Architecture</b>	<b>15</b>
4.1	Software System Architecture Diagram . . . . .	16
4.2	Hardware System Architecture Diagram . . . . .	17

<b>5 Service Introduction</b>	<b>19</b>
<b>6 Quick Start</b>	<b>21</b>
6.1 System Environment . . . . .	21
6.2 Network Environment . . . . .	22
6.3 Installation and Compilation . . . . .	27
6.4 C++ Example Programs . . . . .	27
6.5 Python Example Programs . . . . .	28
6.6 Others . . . . .	32
<b>7 Robot Main Control Service (C++)</b>	<b>35</b>
7.1 Interface Definition . . . . .	35
<b>8 High-Level Motion Control Service (C++)</b>	<b>41</b>
8.1 Interface Definition . . . . .	41
8.2 Type Definitions . . . . .	44
8.3 Enumeration Type Definitions . . . . .	45
8.4 Joystick Diagram . . . . .	46
8.5 Head Movement Range Diagram . . . . .	46
8.6 High-Level Motion Control Robot State Introduction . . . . .	47
8.7 High-Level Motion Control Interface . . . . .	47
<b>9 Low-Level Motion Control Service (C++)</b>	<b>49</b>
9.1 Interface Definition . . . . .	49
9.2 Type Definitions . . . . .	55
9.3 URDF Reference . . . . .	59
9.4 Low-Level Motion Control Robot State Introduction . . . . .	59
9.5 Notice: . . . . .	59
<b>10 Sensor Control Service (C++)</b>	<b>61</b>
10.1 Interface Definition . . . . .	61
10.2 Type Definitions . . . . .	70
10.3 Notice: . . . . .	72
<b>11 Audio Control Service (C++)</b>	<b>73</b>
11.1 Interface Definition . . . . .	73
11.2 Type Definitions . . . . .	78
11.3 Structure Definitions . . . . .	78
11.4 DOA . . . . .	79
11.5 Notice: . . . . .	79
<b>12 State Monitor Service (C++)</b>	<b>81</b>
12.1 Interface Definition . . . . .	81
12.2 Error Code Mapping Table . . . . .	84

<b>13 SLAM Navigation Control Service (C++)</b>	<b>87</b>
13.1 Interface Definition . . . . .	87
13.2 Type Definitions . . . . .	96
13.3 Enumeration Type Definitions . . . . .	97
13.4 SLAM Navigation Control Introduction . . . . .	98
<b>14 Robot Main Control Service (Python)</b>	<b>101</b>
14.1 Interface Definition . . . . .	101
<b>15 High-Level Motion Control Service (Python)</b>	<b>107</b>
15.1 Interface Definition . . . . .	107
15.2 Type Definitions . . . . .	110
15.3 Enumeration Type Definitions . . . . .	111
15.4 Joystick Diagram . . . . .	112
15.5 Head Movement Range Diagram . . . . .	112
15.6 High-Level Motion Control Robot State Introduction . . . . .	113
15.7 High-Level Motion Control Interface . . . . .	113
<b>16 Low-Level Motion Control Service (Python)</b>	<b>115</b>
16.1 Interface Definition . . . . .	115
16.2 Type Definitions . . . . .	120
16.3 URDF Reference . . . . .	124
16.4 Low-Level Motion Control Robot State Introduction . . . . .	124
16.5 Notice: . . . . .	125
<b>17 Sensor Control Service (Python)</b>	<b>127</b>
17.1 Interface Definition . . . . .	127
17.2 Type Definitions . . . . .	136
17.3 Notice: . . . . .	138
<b>18 Audio Control Service (Python)</b>	<b>139</b>
18.1 Interface Definition . . . . .	139
18.2 Type Definitions . . . . .	144
18.3 Structure Definitions . . . . .	144
18.4 DOA . . . . .	146
18.5 Notice: . . . . .	146
<b>19 State Monitor Service (Python)</b>	<b>147</b>
19.1 Interface Definition . . . . .	147
19.2 Error Code Mapping Table . . . . .	150
<b>20 SLAM Navigation Control Service (Python)</b>	<b>153</b>
20.1 Interface Definition . . . . .	153
20.2 Type Definitions . . . . .	162

20.3	Enumeration Type Definitions . . . . .	164
20.4	SLAM Navigation Control Introduction . . . . .	164
<b>21</b>	<b>High-Level Motion Control Example</b>	<b>167</b>
21.1	C++ . . . . .	167
21.2	Python . . . . .	176
21.3	Running Instructions . . . . .	190
<b>22</b>	<b>Low-Level Motion Control Example</b>	<b>191</b>
22.1	C++ . . . . .	191
22.2	Python . . . . .	195
22.3	Running Instructions . . . . .	204
<b>23</b>	<b>Sensor Control Example</b>	<b>207</b>
23.1	C++ . . . . .	207
23.2	Python . . . . .	222
23.3	Running Instructions . . . . .	240
<b>24</b>	<b>Audio Control Example</b>	<b>243</b>
24.1	C++ . . . . .	243
24.2	Python . . . . .	252
24.3	Running Instructions . . . . .	265
<b>25</b>	<b>State Monitor Example</b>	<b>267</b>
25.1	C++ . . . . .	267
25.2	Python . . . . .	269
25.3	Running Instructions . . . . .	273
<b>26</b>	<b>SLAM Navigation Example</b>	<b>275</b>
26.1	C++ . . . . .	275
26.2	Python . . . . .	304
26.3	Running Instructions . . . . .	343
<b>27</b>	<b>FAQ</b>	<b>347</b>
27.1	Development Environment Preparation . . . . .	347
27.2	Does Magicbot-Gen1 support wireless development? . . . . .	347
27.3	What is the current low-level communication frequency of Magicbot-Gen1? . . . . .	347
27.4	Unable to Subscribe and Publish Topic Data? . . . . .	348
27.5	SDK subscription topic data rate unstable? . . . . .	348
27.6	SDK API Call Error: Deadline Exceeded . . . . .	349
27.7	How to View SDK Internal Configuration and Logs? . . . . .	349
27.8	Video Streaming . . . . .	349
27.9	SLAM Relocation Failed? . . . . .	349
27.10	SLAM Navigation Not Executing? . . . . .	350

27.11 Runtime Error: Invalid IP address . . . . .	350
27.12 Log Error: Call of pthread_setschedparam with insufficient privileges! . . . . .	350
27.13 connect Error: Failed to connect to robot, code: ErrorCode.SERVICE_ERROR, message: failed to connect to all addresses; last error: UNKNOWN: ipv4:192.168.54.119:50051: Failed to connect to remote host: Connection refused . . .	350
27.14 Trick Execution Error: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 优先级过低 . . . . .	351
27.15 Trick Execution Error: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 危险动作 . . . . .	351
27.16 Trick Execution Error: Failed to execute robot trick, code: ErrorCode.SERVICE_ERROR, message: 调用 ROS2 服务失败 . . . . .	351



# CHAPTER 1

---

## OverView

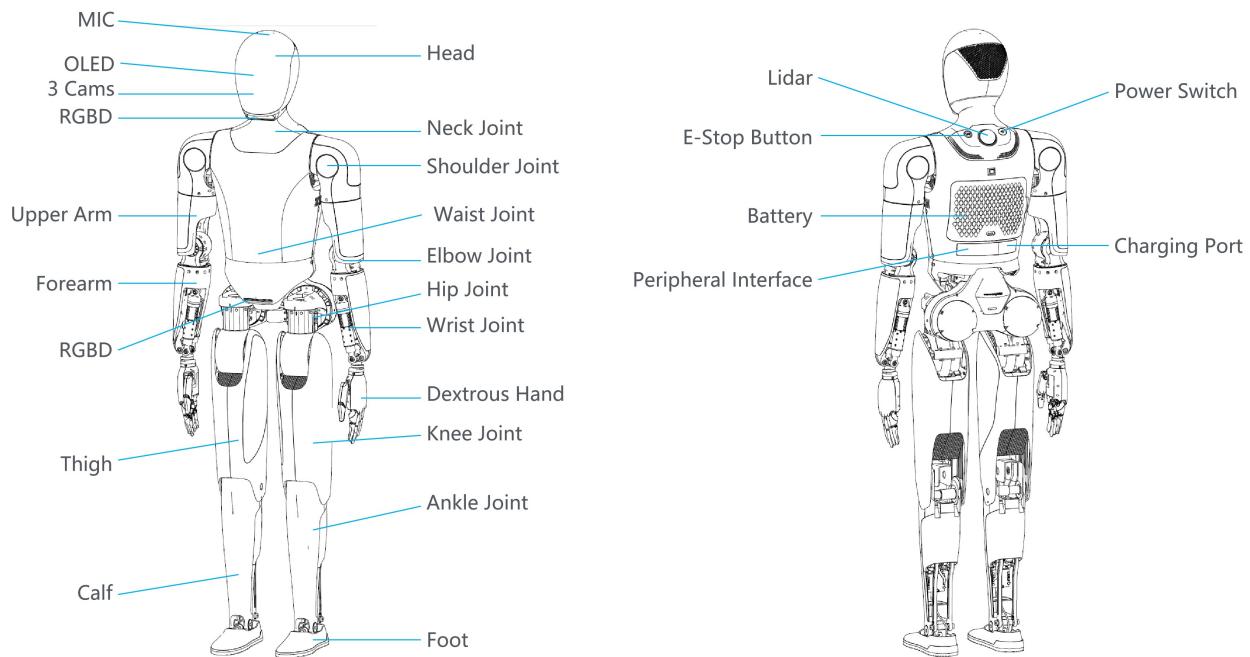
---

---

### 1.1 Product SKUs

MagicBot Gen1 adopts an excellent biomimetic humanoid appearance design, with multiple joint degrees of freedom throughout the body, and is equipped with multiple sensors such as 360 ° LiDAR and high-definition camera, enabling multimodal humanoid interaction. A single arm has 7 degrees of freedom, including shoulder pitch, shoulder roll, shoulder yaw, elbow roll, elbow roll, wrist pitch, and wrist roll. A single leg has 6 degrees of freedom, including hip roll, hip yaw, hip pitch, knee pitch, ankle pitch, and ankle roll. The head has 2 degrees of freedom, including neck joint pitch and neck joint roll. The waist has 2 degrees of freedom, including lumbar joint pitch and lumbar joint roll. The whole machine has a total of 52 degrees of freedom (including passive degrees of freedom), ensuring that the robot can achieve high flexibility in motion and precise attitude control.

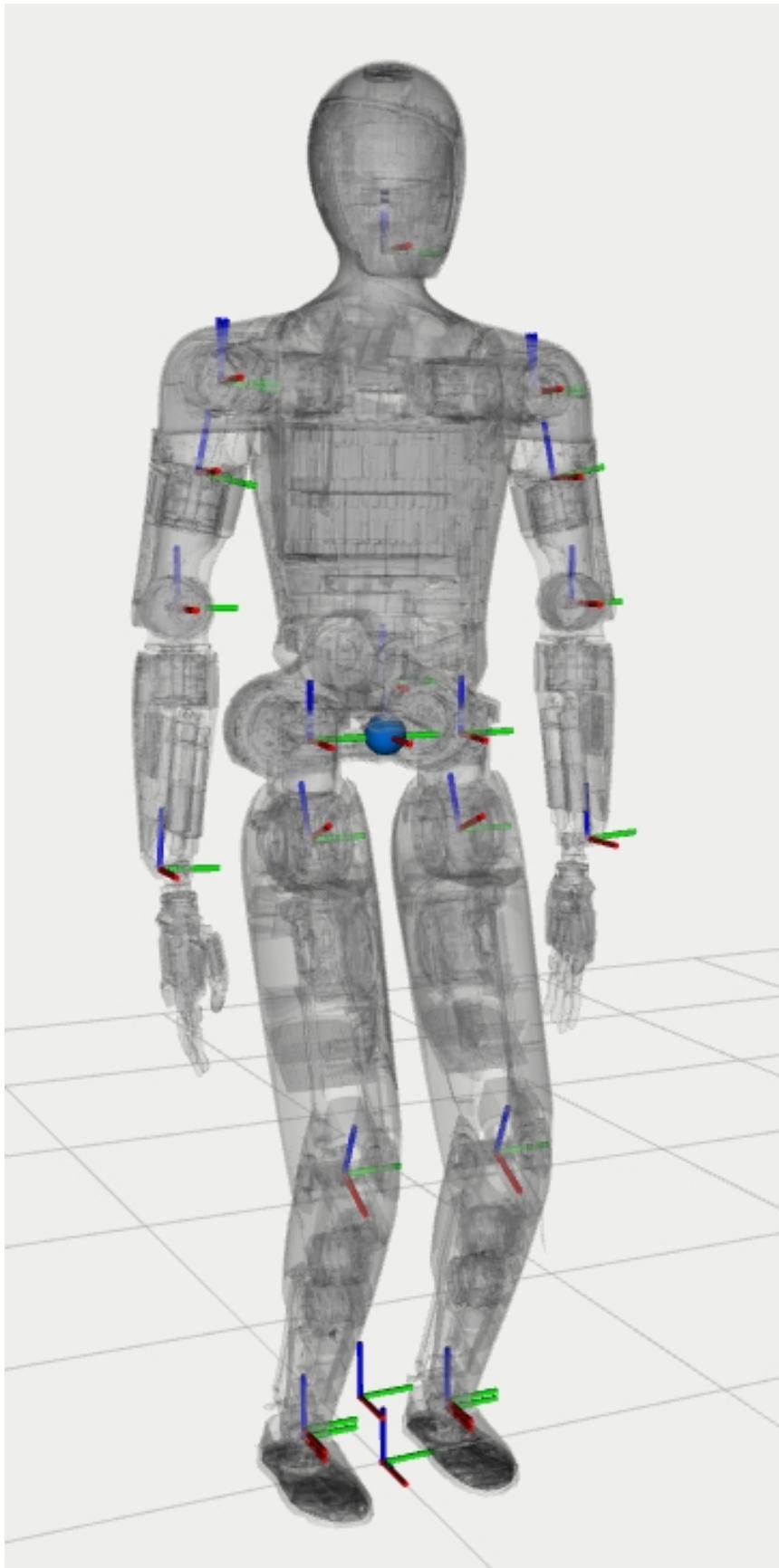
The following is a schematic diagram of the sensor layout. Please refer to the actual product for details.



The degrees of freedom of each part of the system are shown in the following table:

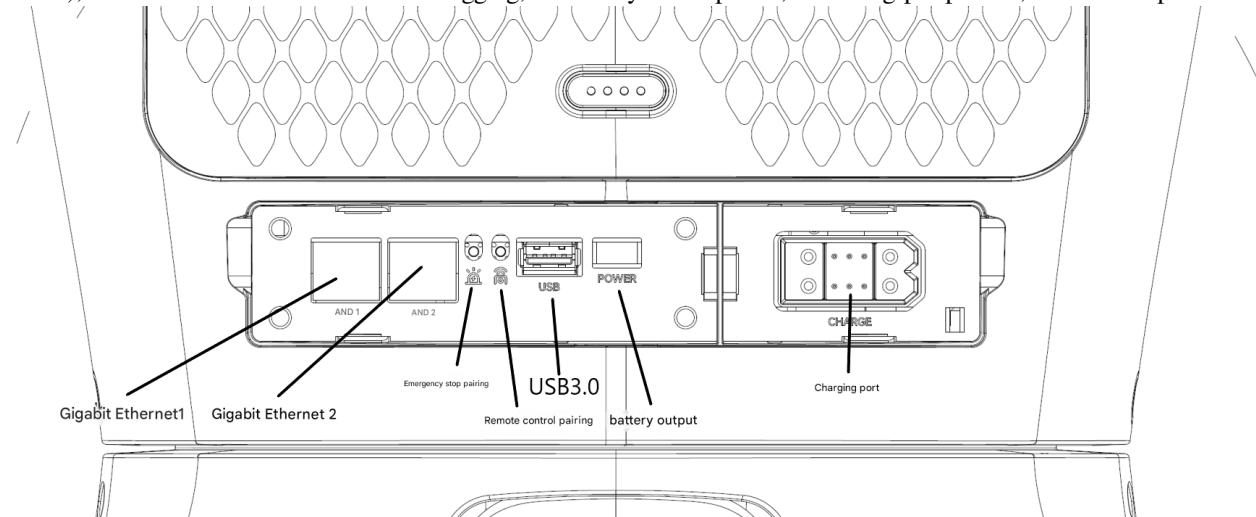
Joint Name	Degrees of Freedom
Head	2
Arm	$7 \times 2$
Dexterous Hand	$11 \times 2$
Waist	2
Leg	$6 \times 2$
Total	52

When all joints are at zero degrees, the coordinate systems are as shown in the figure below, where red is the x-axis, green is the y-axis, and blue is the z-axis.



## 1.2 Electrical interface

The MagicBot Gen1 has multiple types of peripheral interfaces reserved at the back waist (under the battery cover), which can be used for device debugging, secondary development, mounting peripherals, and other operations.



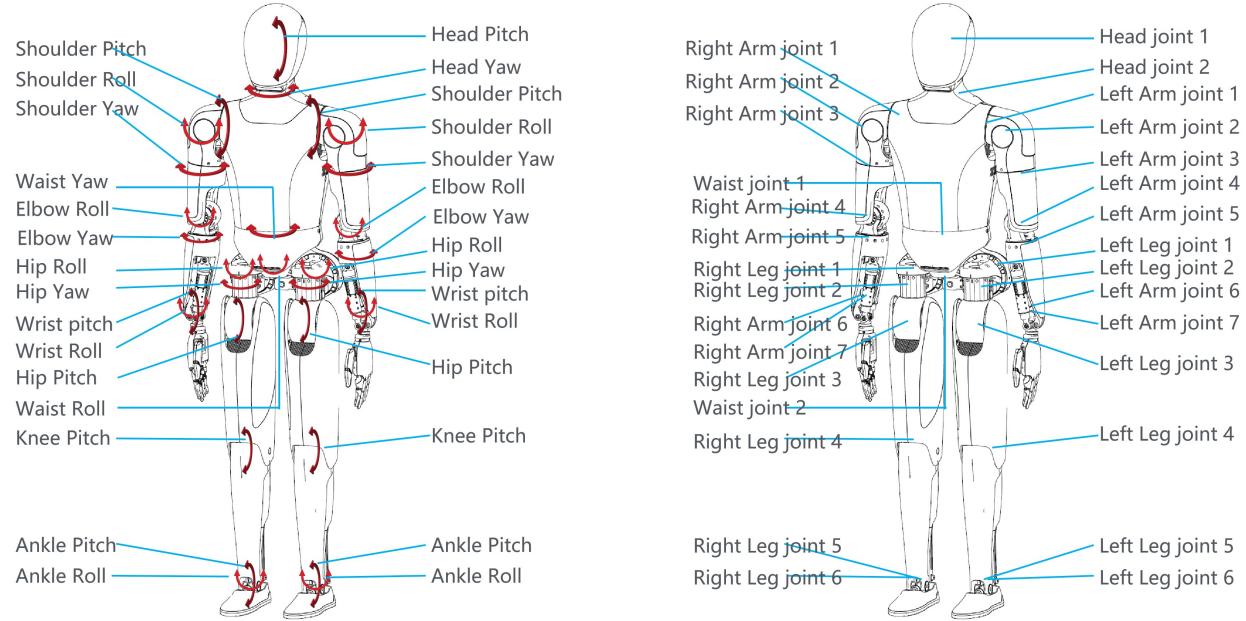
## 1.3 Onboard computer

MagicBot-Gen1 comes standard with one [Operation and Control Computing Unit] and one [Development Computing Unit].

parameter	Development of computing units
model	Jetson Orin NX
CPU	Arm® Cortex®-A78AE
Number of cores	8
Number of threads	8
Maximum turbo frequency	2GHz
Video Memory	16G
Memory	16G
cache	2MB L2 + 4MB L3
storage	512G
GPU	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores
Graphics card maximum dynamic frequency	918MHz
Gaussian and Neural Accelerator	3.0
instruction set	64bit
OpenGL	4.6
OpenGL ES	3.2
Vulkan™	1.1
CUDA	11.4

## 1.4 Joint sequence names and joint limits

The specific joints and numbers of the system are as follows:



Part	Name	Range of Motion
Head	Head Yaw	$\pm 60^\circ$
Head	Head Pitch	$\pm 20^\circ$
Arm	Shoulder Pitch	$\pm 158^\circ$
Arm	Shoulder Roll	-10~165°
Arm	Shoulder Yaw	$\pm 150^\circ$
Arm	Elbow Roll	-10~127°
Arm	Elbow Yaw	$\pm 167^\circ$
Arm	Wrist Pitch	$\pm 26^\circ$
Arm	Wrist Roll	-63~49°
Waist	Waist Yaw	$\pm 90^\circ$
Waist	Waist Roll	$\pm 20^\circ$
Leg	Hip Roll	-20°~+30°
Leg	Hip Yaw	-15°~+30°
Leg	Hip Pitch	-120°~+40°
Leg	Knee Pitch	0°~+150°
Leg	Ankle Pitch	-75°~+30°
Leg	Ankle Roll	-20°~+20°

## 1.5 LiDAR

The MagicBot Gen1 is equipped with a MID-360 laser radar on the back neck, which provides 360 ° blind spot perception with miniaturization and low power consumption characteristics, accurately obtains environmental data, assists in mapping navigation and obstacle avoidance, and can complement vision to adapt to complex scenes.

## 1.6 Depth Camera

The MagicBot Gen1 is equipped with D435 depth cameras at the lower jaw and waist, which have a wide field of view and can comprehensively capture environmental information. The global shutter is adapted to dynamic scenes, and the low light environment is stable with accurate depth perception. It supports real-time environment modeling and obstacle avoidance, and is compact and easy to integrate, enabling the robot to intelligently interact with the surrounding environment.

## 1.7 Robot specifications

### 1.7.1 MagicBot-Gen1

Project Name	Specifications
Height, Width, and Depth (Standing)	174cm × 58cm × 28 cm
Arm Reach	160 cm
Thigh + Calf	95 cm
Total Weight	Approximately 70 kg
Battery	Capacity: 25Ah (1.35Kwh)
Motion Speed	≥2 m/s
Maximum Dual-Arm Load	15 kg
Maximum Joint Torque	≥350 N · m
Maximum Slope Angle	≥20°
Dexterous Hand	11-DOF Haptic Dexterous Hand (Optional)
Single-arm Degrees of Freedom	7 (3 shoulder joints + 1 elbow joint + 3 wrist joints)
Single-leg Degrees of Freedom	6 (3 hip joints + 1 knee joint + 2 ankle joints)
Computing Power Module	Basic computing power: 8-core high-performance CPU Perceived computing power: NVIDIA Jeston Orin
Sensor Configuration	3D LiDAR, depth camera, trinocular camera
Audio Module	Ring microphone, speaker
Communication Module	5G Module(Optional), Bluetooth 5.2, Wi-Fi 6
Reducer	Planetary Reducer, Harmonic Reducer

# CHAPTER 2

---

## Remote control button description

---

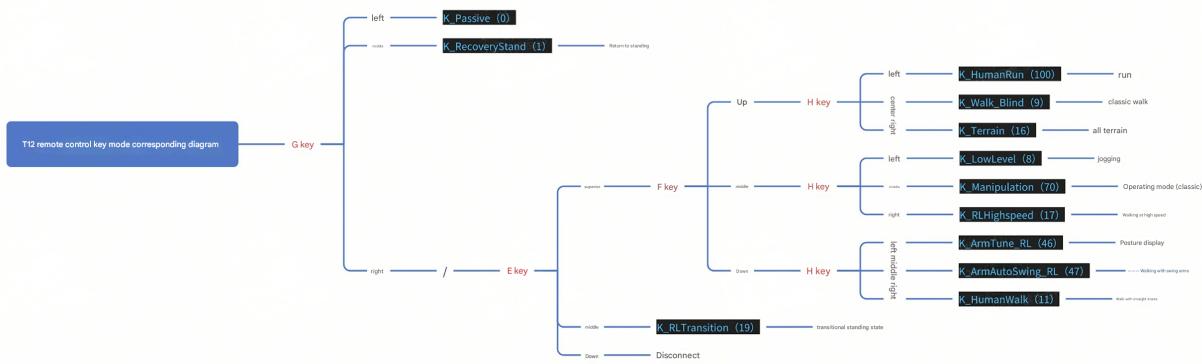
---

### 2.1 Functional Description

parameter	Development of computing units
Remote control power on	Press and hold for 3 seconds to power on the device without human intervention.
Remote control shutdown	Press and hold for 3 seconds to shut down the device without human intervention
Charge	USB-C direct fast charging
APP pairing	Pairing via mobile app
Robot state switching	Switch between damping, recover stand, and RL modes
Robot gait switching	Switch between ArmAutoSwin 47, walking baseline k_lowlevel 8, and ArmTune 46
Robot stunt switching	Switch between opening hand, returning hand, turning around, and turning back
Mobile phone and remote control interaction	Supports clamping the phone to the remote control 1. The remote control has higher priority than the APP: if you use the remote control and the APP at the same time, only the remote control will be controlled 2. Supports controlling the robot separately
Disassembly and storage	The remote control joystick is detachable and storable

---

## 2.2 Mode Switching



## 2.3 Remote control check

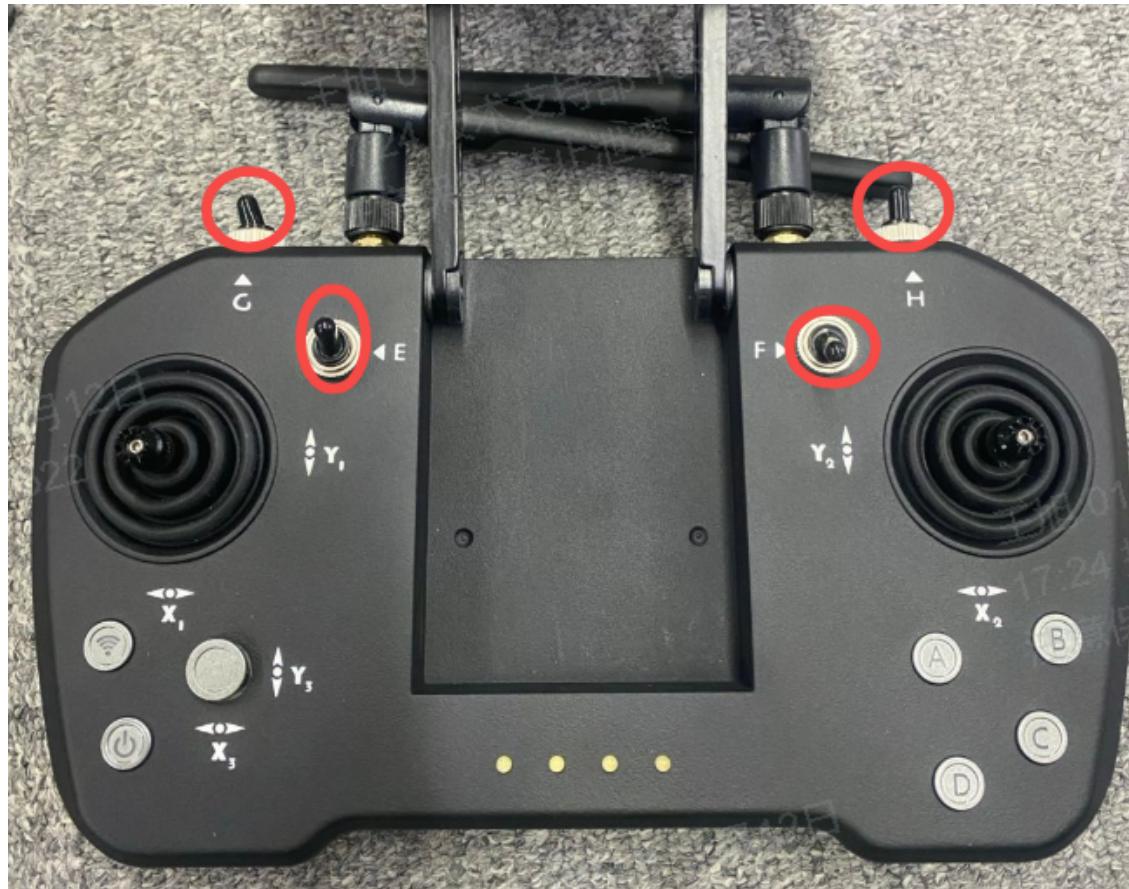
- Check that the joysticks of the remote controller are in the middle position and whether there are any foreign objects such as sand in the joysticks;
- Check whether there is any lag on the buttons of the remote control;

- Restore the remote control's levers to their default settings and check if the remote control's batteries are sufficiently charged. The specific steps are as follows:

- As shown in the figure, short press and then long press the power button on the remote control to turn on the remote control;
- At this time, the indicator light in the center of the lower part of the remote control lights up, indicating the battery status of the remote control from left to right.

When the 1st, 2nd, 3rd and 4th lights are on, the corresponding power levels are 25%, 50%, 75% and 100% respectively;

- If you are using the remote control, before turning on the remote control, you need to return the control levers to the initial position: "G" to the far left, "E" to the top, "F" to the bottom, and "H" to the middle.

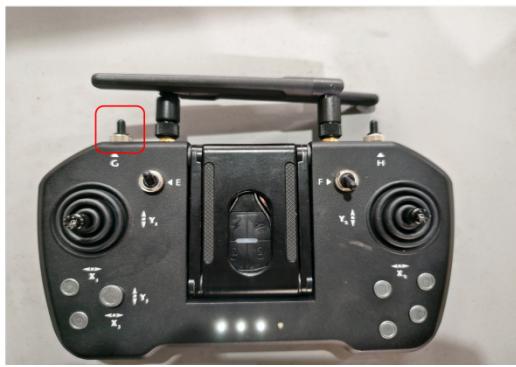


## 2.4 Remote control on and off

- Press the power button briefly and then long press it. When the battery indicator lights up, the device is turned on successfully.
- Press the power button briefly and then long press it until the battery indicator light goes out, which means the device has been shut down successfully.

## 2.5 Remote control operation during power on

- Short press and long press the power button, after the indicator light turns on, the remote control is turned on successfully;
- The robot must be fixed on the protective bracket, and the “G” lever on the remote controller must be moved from the left to the middle;
- It is observed that the robot’s legs and hands have small joint displacements, and the limbs will slowly enter a standing state.
- Control the protective bracket to slowly descend, lower the robot to stand on the ground, and observe whether the robot is standing in balance. If there is no abnormality such as shaking, tilting forward, backward, left, or right, the robot is standing in balance;



## 2.6 Walking mode switch

- Make sure the robot is in a standing position and on the ground;
- Move the “G” lever on the remote control to the far right, the “H” lever to the middle, and the “F” lever to the bottom to enter the swing arm walking mode. In this mode, the robot has a steady gait.
- Move the “G” lever on the remote control to the far right, move the “H” lever to the right, and move the “F” lever to the bottom to enter the anthropomorphic walking mode. In this mode, the robot has a light gait.

## 2.7 Stop and return to standing position

In the swing arm walking mode and anthropomorphic walking mode:

- The remote-controlled robot remains stopped;
- Move the “G” lever on the remote control to the middle and the robot will enter the standing state.

Note: Do not move the “G” lever on the remote control directly to the center or the far left while walking;

## **2.8 Emergency Stop**

In the following emergency situations, you need to immediately move the “G” lever on the remote controller to the far left, which will disable the robot’s joints. At this point, the joints will lose power, and the robot will be unable to maintain any posture and will immediately collapse and fall:

- The robot emits smoke or a burning smell;
- Water or other foreign objects have entered the robot;
- The robot is out of control and cannot be stopped by remote control or other means;
- The robot is damaged and may cause further danger;
- Emergency avoidance, such as working at height or losing control in a dangerous environment;
- Sudden changes in the external environment, such as earthquakes, fires, etc.

Note: The robot must be secured to the protective bracket before moving the “G” lever on the remote controller to the far left.

## **2.9 Remote control operation when power off**

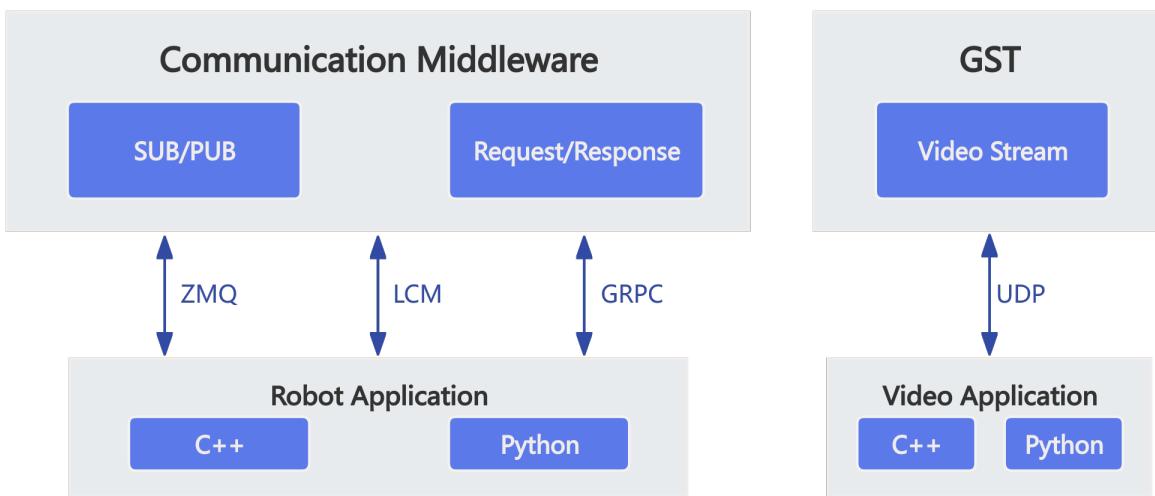
- Make sure to move the “G” lever on the remote control to the middle to put the robot into standing position;
- Fix the robot on the bracket and lift it upwards;
- Press and hold the power button on the back of the robot (about 6 seconds) until the power status light goes out and the robot is powered off.



# CHAPTER 3

## SDK Overview

### 3.1 SDK Communication Interface Introduction



- Current software version does not support GST video stream transmission

Gen1 uses LCM/GRPC/UDP/ZMQ as message middleware, with main data interaction modes: topics (subscribe/publish) and RPC (request/response)

- **Topics (Subscribe/Publish):** Receivers subscribe to specific messages, senders send messages to receivers based on subscription lists, mainly used for medium-high frequency or continuous data interaction.
- **RPC (Request/Response):** Question-answer mode, implementing data acquisition or operations through requests. Used for low-frequency or data interaction during function switching.

Main calling methods for topics and RPC interfaces: functional interfaces

- **Functional Interface:** Encapsulates API calls into function calls for user convenience.

## 3.2 Getting the SDK

**magicbot-gen1\_sdk** is MagicLab's next-generation robot development SDK. The SDK encapsulates high-level motion control, low-level motor control, voice control and other interfaces, and provides related functional interfaces. You can refer to our provided SDK tutorials to learn robot control and complete Gen1 secondary development;

### 3.2.1 SDK Download Address:

[magicbot-gen1\\_sdk](#)

The SDK version must be aligned with the robot firmware version. Please refer to [ChangeLog](#)

### 3.2.2 URDF/MJCF Download Address:

[URDF/MJCF](#)

# CHAPTER 4

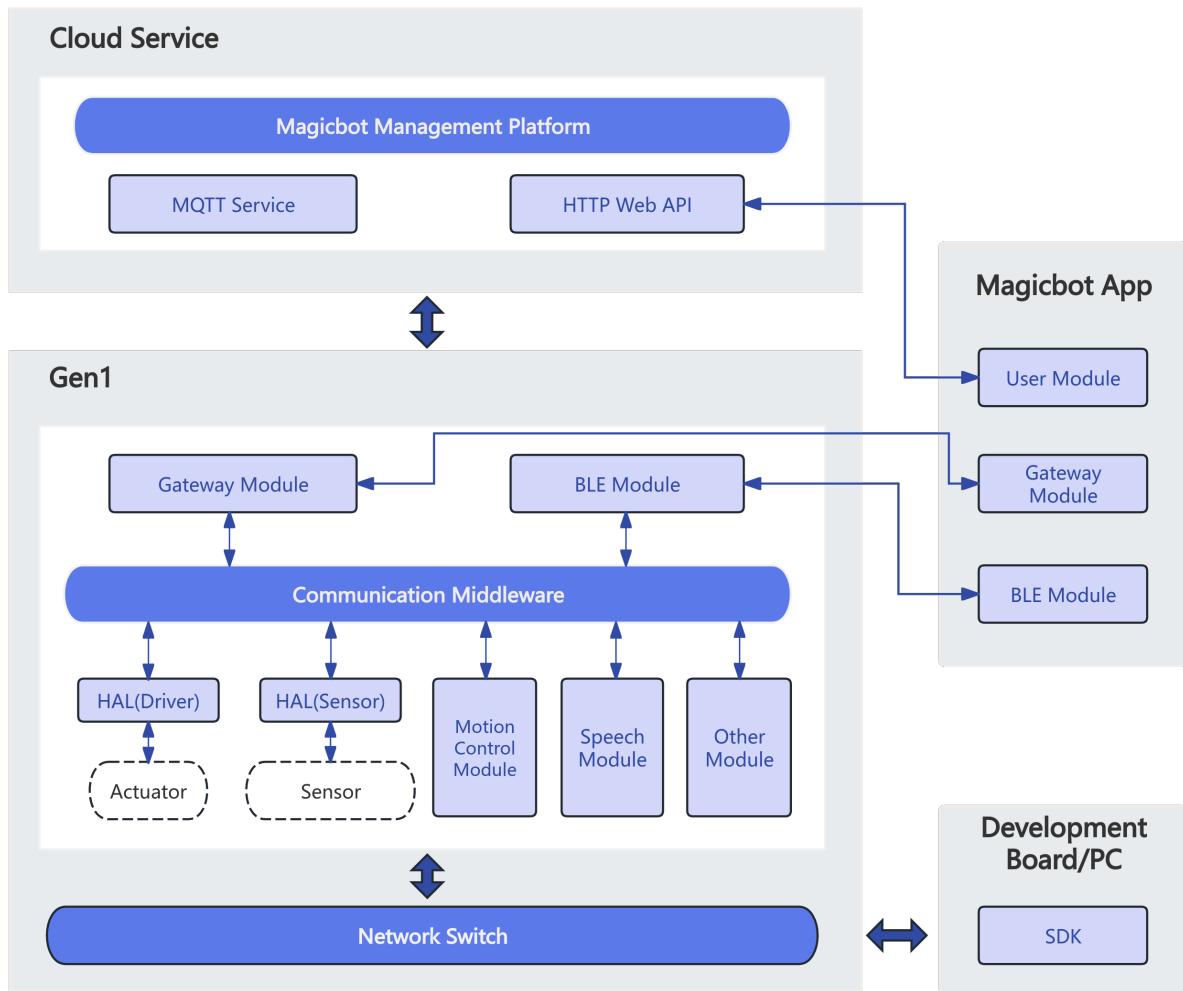
---

Software and Hardware Architecture

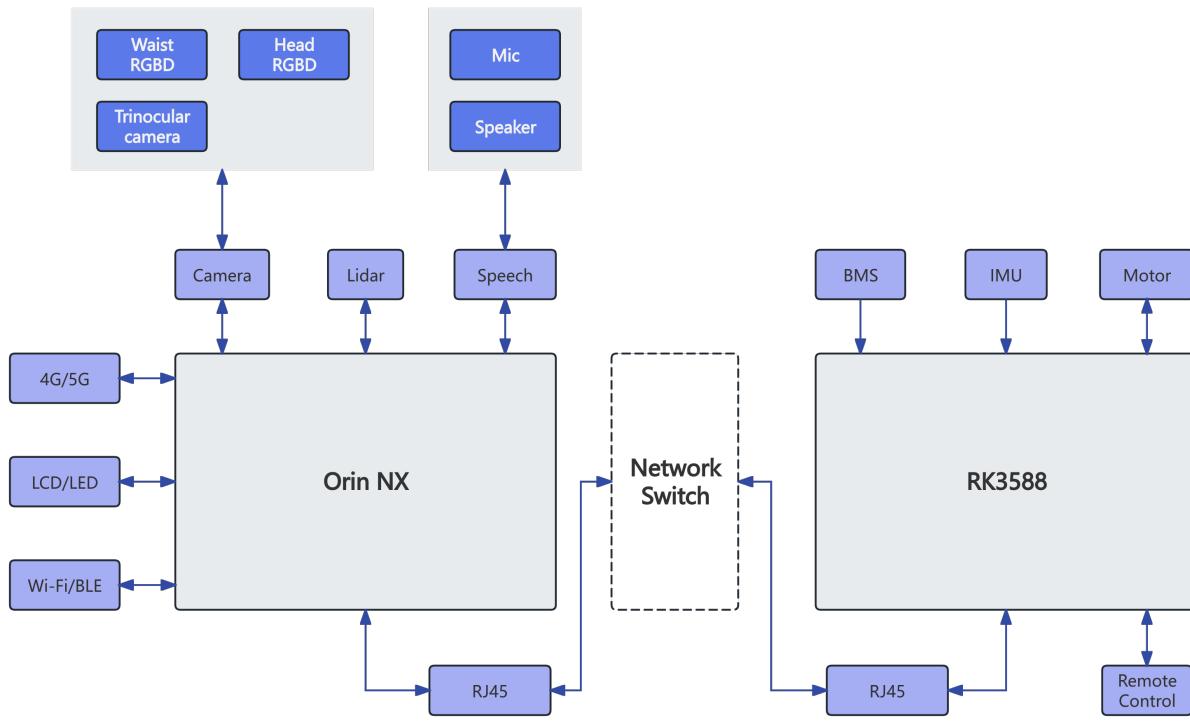
---

---

## 4.1 Software System Architecture Diagram



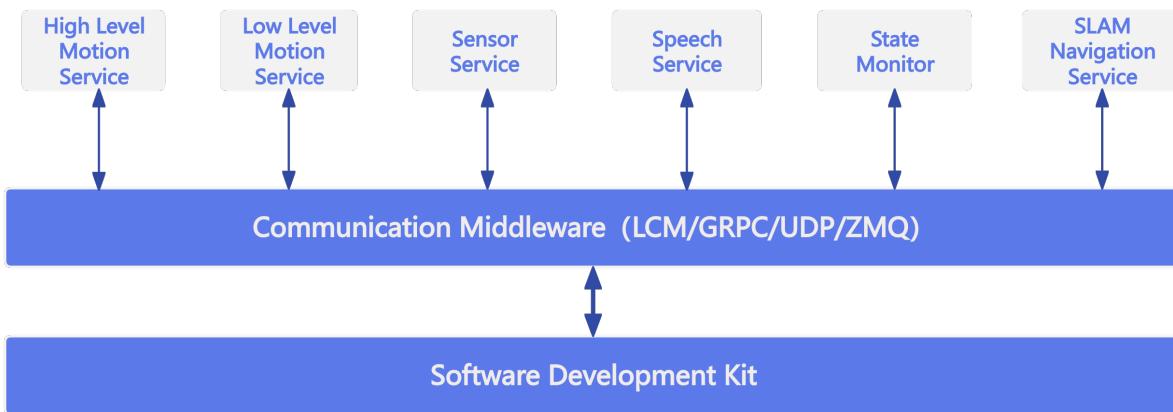
## 4.2 Hardware System Architecture Diagram





# CHAPTER 5

## Service Introduction



Magicbot Gen1 provides the following services through message middleware (LCM/GRPC/UDP/ZMQ):

- **High-Level Motion Control Service**

Based on Gen1's built-in gait controller, it can perform gait switching, trick execution, attitude and velocity control (equivalent to remote control) and other functions. High-level motion control service uses GRPC for communication.

- **Low-Level Motion Control Service**

Through service interfaces, joint and IMU data can be obtained in real-time, and joint commands can be sent in real-time to control motors. Low-level motion control service uses LCM for communication.

- **Audio Service**

Through service interfaces, volume and voice control can be managed. Audio service uses GRPC/LCM for communication.

- **Sensor Service**

Supports data subscription for lidar, RGBD, triple-camera and other sensors. Sensor service uses GRPC/UDP/LCM/ZMQ for communication.

Notice: Current version only supports lidar data subscription, RGBD/Trinocular-camera is not yet supported

- **Status Monitoring Service**

Through service interfaces, robot body software/hardware faults and BMS status can be subscribed to. State monitoring service uses LCM for communication.

- **SLAM Navigation Service**

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

# CHAPTER 6

---

## Quick Start

---

---

### 6.1 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;

APP and SDK cannot support control the robot simultaneously

#### 6.1.1 Development Environment Requirements

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

#### 6.1.2 System Configuration

First, to achieve real-time communication under 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. Use `sudo sysctl -p` to take effect immediately or restart to take effect:

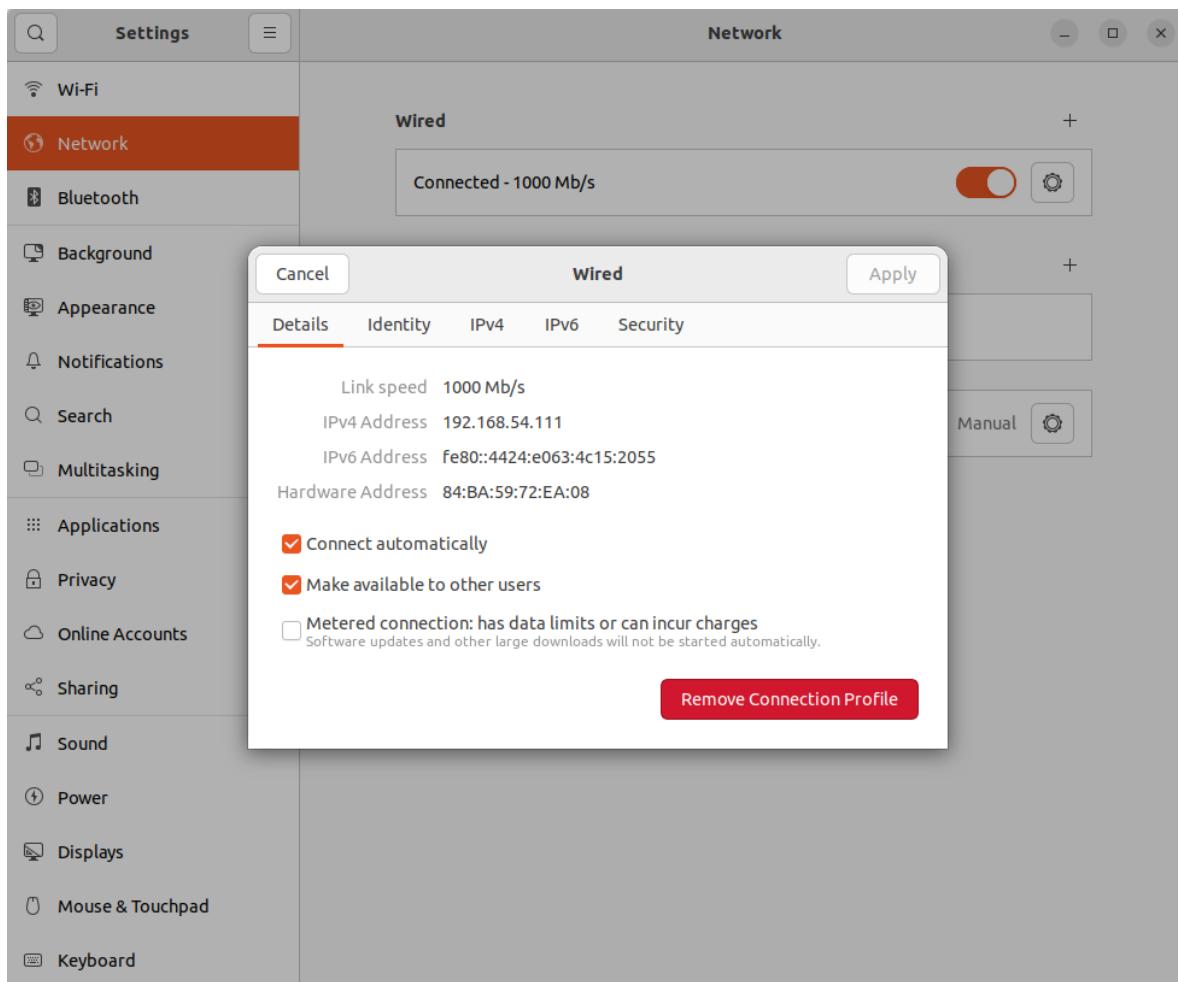
```
net.core.rmem_max=20971520  
net.core.rmem_default=20971520  
net.core.wmem_max=20971520  
net.core.wmem_default=20971520
```

## 6.2 Network Environment

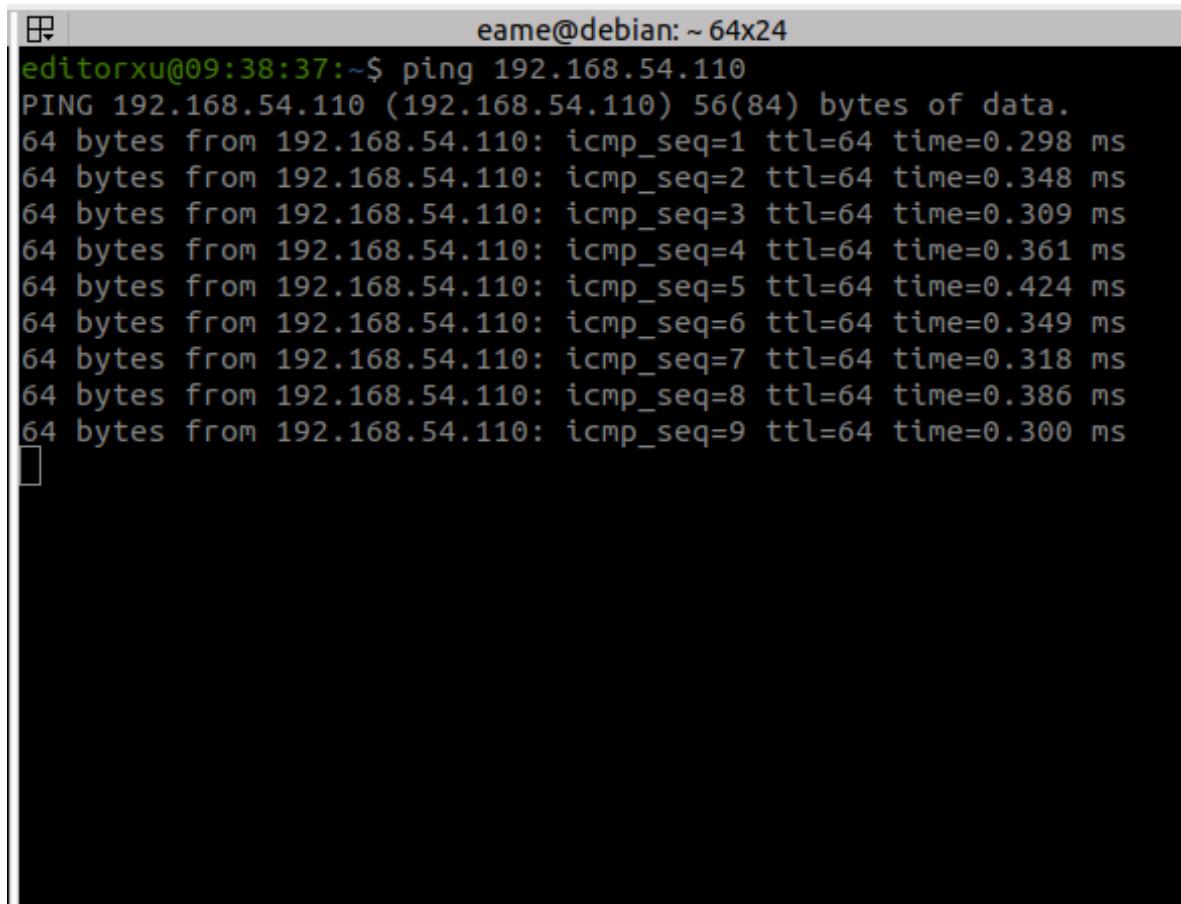
Connect the user computer and robot switch to a unified network. It is recommended that new users use an Ethernet cable to connect the user computer to the robot switch, and set the network card communicating with the robot to the 192.168.54.X network segment, recommended 192.168.54.111. Experienced users can configure the network environment themselves.

### 6.2.1 Configuration Steps

1. Connect one end of the Ethernet cable to the robot and the other end to the user computer. The robot's onboard computer IP addresses are 192.168.54.110 (cerebellum computing board) and 192.168.54.119 (brain computing board), so the computer IP needs to be set to the same network segment, recommended 192.168.54.111.



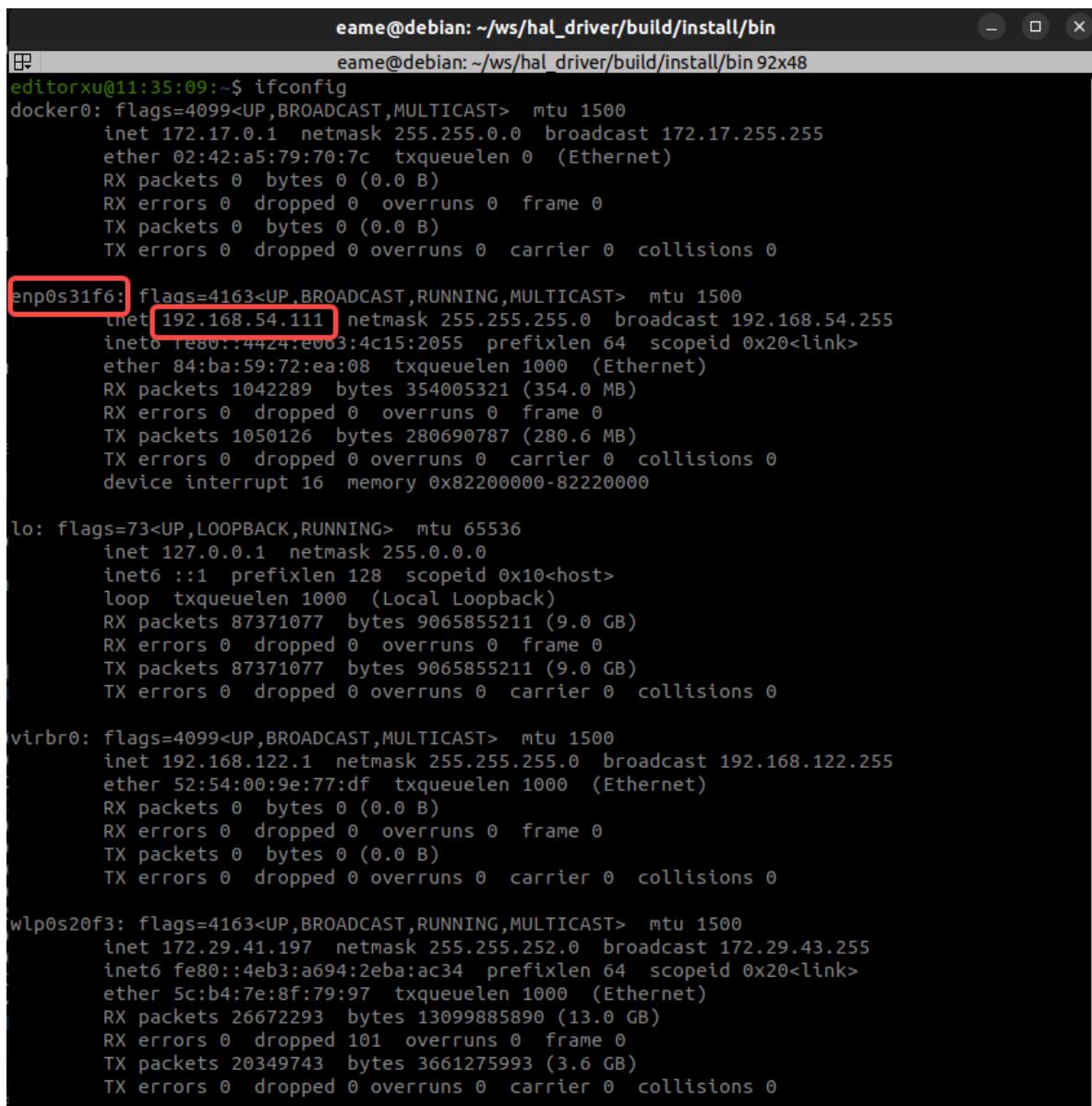
2. To test if the communication connection is normal, you can test with ping:



A terminal window titled 'eame@debian: ~ 64x24' showing the output of a 'ping' command. The command 'ping 192.168.54.110' was run at 09:38:37. The output shows nine ICMP echo requests (seq=1 to 9) being sent to the target address. Each packet is 64 bytes long and has a TTL of 64. The time taken for each response ranges from 0.298 ms to 0.424 ms.

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

3. Check the network card name corresponding to the 192.168.54.111 address by using the ifconfig command to view the network card name, as shown in the figure:



```
eame@debian: ~/ws/hal_driver/build/install/bin
eame@debian: ~/ws/hal_driver/build/install/bin 92x48

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

enp0s31f6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.54.111 netmask 255.255.255.0 broadcast 192.168.54.255
        inet6 fe80::e063:4c15%2055 prefixlen 64 scopeid 0x20<link>
            ether 84:ba:59:72:ea:08 txqueuelen 1000 (Ethernet)
            RX packets 1042289 bytes 354005321 (354.0 MB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1050126 bytes 280690787 (280.6 MB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
            device interrupt 16 memory 0x82200000-82220000

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

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

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

Assuming the network port connecting the SDK secondary development PC to the robot is `enp0s31f6`, the following configuration is needed for SDK to communicate with 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
```

To avoid having to manually reconfigure the network interface multicast route every time you unplug and replug the network cable, you can automate this process on your development PC by following the steps below (system requirements: Ubuntu 22.04, and NetworkManager as the network configuration tool):

- Check the service status:

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

If the output shows active, proceed with the next steps; otherwise, skip the following steps.

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

```
#!/bin/bash

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

# Target network interface
TARGET_IF="enp0s31f6"

# Only execute when the specified interface is 'carrier on' or 'up'
if [ "$IFACE" = "$TARGET_IF" ] && [[ "$STATUS" = "up" || "$STATUS" = "carrier" ]]; then
    # Delete any possible old route
    ip route del 224.0.0.0/4 dev "$TARGET_IF" 2>/dev/null

    # Add the required route (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 take effect:

```
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 enp0s31f6 scope link
```

## 6.3 Installation and Compilation

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

### 6.3.1 Install magicbot-gen1\_sdk:

Installation steps: Enter the magicbot-gen1\_sdk directory and execute the following commands step by step to install the SDK to the /opt/magic\_robots/magicbot\_gen1\_sdk directory for easy find\_package retrieval for secondary development program compilation and linking:

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

### 6.3.2 Example Compilation

Compilation steps: Enter the magicbot-gen1\_sdk directory and execute the following commands step by step to generate example executable programs in the build directory:

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

## 6.4 C++ Example Programs

In the magicbot-gen1\_sdk/build directory:

- **Audio Example:**
  - audio\_example
- **Sensor Example:**
  - sensor\_example
- **Status Monitoring Example:**
  - monitor\_example
- **Low-Level Motion Control Example:**
  - low\_level\_motion\_example

- **High-Level Motion Control Example:**

- high\_level\_motion\_example

- **SLAM Example:**

- slam\_example

- **Navigation Example:**

- navigation\_example

**Notice:** Among these, the high-level motion control examples have sequential dependencies (recovery stand -> balance stand -> execute trick actions/remote control)

#### 6.4.1 Enter Debug Mode:

Follow the operation procedures to ensure the robot enters debug mode

#### 6.4.2 Run Examples

Enter the magicbot-gen1\_sdk/build directory and execute the following commands:

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

# execute audio control example
./audio_example
```

### 6.5 Python Example Programs

In the magicbot-gen1\_sdk/example/python directory:

- **Audio Example:**

- audio\_example.py

- **Sensor Example:**

- sensor\_example.py

- **Status Monitoring Example:**

- monitor\_example.py

- **Low-Level Motion Control Example:**

- low\_level\_motion\_example.py

- **High-Level Motion Control Example:**

- high\_level\_motion\_example.py

- **SLAM Example:**

- slam\_example.py

- **Navigation Example:**

- navigation\_example.py

**Notice:** Among these, the high-level motion control examples have sequential dependencies (recovery stand -> balance stand -> execute trick actions/remote control)

### 6.5.1 Enter Debug Mode:

Follow the operation procedures to ensure the robot enters debug mode

### 6.5.2 Run Examples

Enter the magicbot-gen1\_sdk/example/python directory and execute the following commands:

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

# execute audio control example
python3 audio_example.py
```

Get Python API help information:

```
# Environment configuration
export PYTHONPATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$PYTHONPATH
export LD_LIBRARY_PATH=/opt/magic_robotics/magicbot_gen1_sdk/lib:$LD_LIBRARY_PATH

$ python3
Python 3.10.12 (main, Aug 15 2025, 14:32:43) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
# Import Magic Gen1 python SDK interface
>>> import magicbot_gen1_python
# View all interface information
>>> help(magicbot_gen1_python)
```

(continues on next page)

(continued from previous page)

```
# View CameraInfo structure information
>>> help(magicbot_gen1_python.CameraInfo)

# View HighLevelMotionController object information
>>> help(magicbot_gen1_python.HighLevelMotionController)

# View LowLevelMotionController object information
>>> help(magicbot_gen1_python.LowLevelMotionController)

# View SensorController object information
>>> help(magicbot_gen1_python.SensorController)

# View AudioController object information
>>> help(magicbot_gen1_python.AudioController)

# View StateMonitor object information
>>> help(magicbot_gen1_python.StateMonitor)
```

For example, if you want to view the TrickAction enumeration values:

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

>>> import magicbot_gen1_python
>>> help(magicbot_gen1_python.TrickAction)

Help on class TrickAction in module magicbot_gen1_python:

class TrickAction(pybind11_builtins.pybind11_object)
| Method resolution order:
|     TrickAction
|     pybind11_builtins.pybind11_object
|     builtins.object
|
| Methods defined here:
|
|     __eq__(...)
|         __eq__(self: object, other: object, /) -> bool
|
|     __getstate__(...)
|         __getstate__(self: object, /) -> int
|
|     __hash__(...)
|         __hash__(self: object, /) -> int
| 
```

(continues on next page)

(continued from previous page)

```
| __index__(...)  
|     __index__(self: magicbot_gen1_python.TrickAction, /) -> int  
|  
| __init__(...)  
|     __init__(self: magicbot_gen1_python.TrickAction, value: typing.SupportsInt) ->  
|     None  
|  
| __int__(...)  
|     __int__(self: magicbot_gen1_python.TrickAction, /) -> int  
|  
| __ne__(...)  
|     __ne__(self: object, other: object, /) -> bool  
|  
| __repr__(...)  
|     __repr__(self: object, /) -> str  
|  
| __setstate__(...)  
|     __setstate__(self: magicbot_gen1_python.TrickAction, state: typing.  
|     SupportsInt, /) -> None  
|  
| __str__(...)  
|     __str__(self: object, /) -> str  
|-----  
| Readonly properties defined here:  
|  
| __members__  
|  
| name  
|     name(self: object, /) -> str  
|  
| value  
|-----  
| Data and other attributes defined here:  
|  
| ACTION_CELEBRATE = <TrickAction.ACTION_CELEBRATE: 201>  
|  
| ACTION_CIRCLE_HEAD = <TrickAction.ACTION_CIRCLE_HEAD: 221>  
|
```

(continues on next page)

(continued from previous page)

```

| ACTION_GREETING = <TrickAction.ACTION_GREETING: 301>
|
| ACTION_NOD_HEAD = <TrickAction.ACTION_NOD_HEAD: 219>
|
| ACTION_NONE = <TrickAction.ACTION_NONE: 0>
|
| ACTION_POINT_GROUND = <TrickAction.ACTION_POINT_GROUND: 302>
|
| ACTION_POINT_GROUND_WITH_DRAW = <TrickAction.ACTION_POINT_GROUND_WITH_...
|
| ACTION_RECOVERY_STAND = <TrickAction.ACTION_RECOVERY_STAND: 103>
|
| ACTION_SHAKE_HAND_REACHOUT = <TrickAction.ACTION_SHAKE_HAND_REACHOUT: ...
|
| ACTION_SHAKE_HAND_WITHDRAW = <TrickAction.ACTION_SHAKE_HAND_WITHDRAW: ...
|
| ACTION_SHAKE_HEAD = <TrickAction.ACTION_SHAKE_HEAD: 220>
|
| ACTION_SPREAD_HAND = <TrickAction.ACTION_SPREAD_HAND: 304>
|
| ACTION_SPREAD_HAND_WITH_DRAW = <TrickAction.ACTION_SPREAD_HAND_WITH_DR...
|
| ACTION_TRUN_AWAY_LEFT_INTRODUCE = <TrickAction.ACTION_TRUN_AWAY_LEFT_I...
|
| ACTION_TRUN_BACK_LEFT_INTRODUCE = <TrickAction.ACTION_TRUN_BACK_LEFT_I...
|
| -----
| Static methods inherited from pybind11_builtins.pybind11_object:
|
|     __new__(*args, **kwargs) from pybind11_builtins.pybind11_type
|         Create and return a new object. See help(type) for accurate signature.

```

## 6.6 Others

### 6.6.1 SDK Configuration File

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

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

## 6.6.2 SDK Logs

The SDK's internal logs are generated in the /tmp/logs directory by default during runtime:

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



# CHAPTER 7

---

## Robot Main Control Service (C++)

---

---

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, state monitor, and sensor controllers.

### 7.1 Interface Definition

MagicRobot is the unified entry class for the robot SDK.

#### 7.1.1 MagicRobot

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

### 7.1.2 ~MagicRobot

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

### 7.1.3 Initialize

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

### 7.1.4 Shutdown

Item	Content
Function Name	Shutdown
Function Declaration	<code>void Shutdown();</code>
Function Overview	Shutdown the robot system and release resources.
Notes	Use in conjunction with Initialize.

### 7.1.5 Release

Item	Content
Function Name	Release
Function Declaration	<code>void Release();</code>
Function Overview	Release resources.
Notes	Release topic resources occupied by the SDK.

### 7.1.6 Connect

Item	Content
Function Name	Connect
Function Declaration	Status Connect();
Function Overview	Establish communication connection with robot service.
Return Value	Status::OK indicates success.
Notes	Blocking interface, must be called after initialization.

### 7.1.7 Disconnect

Item	Content
Function Name	Disconnect
Function Declaration	Status Disconnect();
Function Overview	Disconnect from robot service.
Return Value	gRPC call status.
Notes	Blocking interface, used in conjunction with Connect.

### 7.1.8 GetSDKVersion

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

### 7.1.9 GetMotionControlLevel

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

### 7.1.10 SetMotionControlLevel

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

### 7.1.11 GetHighLevelMotionController

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

### 7.1.12 GetLowLevelMotionController

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

### 7.1.13 GetAudioController

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

### 7.1.14 GetSensorController

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

### 7.1.15 GetStateMonitor

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

### 7.1.16 GetSlamNavController

Item	Content
Function Name	GetSlamNavController
Function Declaration	SlamNavController& GetSlamNavController();
Function Overview	Get SLAM and navigation controller object.
Return Value	Reference type, used for mapping and localization.
Notes	Non-blocking interface, used for robot SLAM mapping and autonomous navigation functions.



# CHAPTER 8

---

## High-Level Motion Control Service (C++)

---

---

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

### 8.1 Interface Definition

HighLevelMotionController is a high-level motion controller for semantic control, supporting control operations such as walking, trick actions, head movement, etc., encapsulating low-level details for high-level system calls.

#### 8.1.1 HighLevelMotionController

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

### 8.1.2 ~HighLevelMotionController

Item	Content
Function Name	<code>~HighLevelMotionController</code>
Function Declaration	<code>virtual ~HighLevelMotionController();</code>
Function Overview	Destructor, releases controller resources
Notes	Used in conjunction with constructor

### 8.1.3 Initialize

Item	Content
Function Name	Initialize
Function Declaration	<code>virtual bool Initialize() override;</code>
Function Overview	Initialize controller, prepare high-level control functions
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure
Notes	Must be called before first use

### 8.1.4 Shutdown

Item	Content
Function Name	Shutdown
Function Declaration	<code>virtual void Shutdown() override;</code>
Function Overview	Shutdown controller and release resources
Notes	Used in conjunction with Initialize, safely disconnect

### 8.1.5 SetGait

Item	Content
Function Name	SetGait
Function Declaration	<code>Status SetGait(const GaitMode gait_mode, int timeout_ms);</code>
Function Overview	Set robot gait mode (such as recovery stand, balance stand, humanoid walking, etc.)
Parameter Description	<code>gait_mode</code> : Gait control enumeration
Return Value	<code>Status::OK</code> indicates success, others indicate failure
Notes	Blocking interface, can switch between multiple gait modes

### 8.1.6 GetGait

Item	Content
Function Name	GetGait
Function Declaration	Status GetGait(GaitMode& gait_mode);
Function Overview	Get robot gait mode (such as recovery stand, balance stand, humanoid walking, etc.)
Parameter Description	gait_mode: Gait control enumeration
Return Value	Status::OK indicates success, others indicate failure
Notes	Blocking interface, gets current gait mode

### 8.1.7 ExecuteTrick

Item	Content
Function Name	ExecuteTrick
Function Declaration	Status ExecuteTrick(const TrickAction trick_action, int timeout_ms);
Function Overview	Execute trick actions (such as celebrate, wave, etc.)
Parameter Description	trick_action: Trick action identifier
Return Value	Status::OK indicates success, others indicate failure
Notes	Blocking interface, ensure robot can currently execute tricksNote: Trick actions must be performed in GaitMode::GAIT_BALANCE_STAND(46) gait

- Note: Trick actions must be performed in GaitMode::GAIT\_BALANCE\_STAND(46) gait to execute trick demonstrations

### 8.1.8 SendJoyStickCommand

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

### 8.1.9 HeadMove

Item	Content
Function Name	HeadMove
Function Declaration	Status HeadMove(float shake_angle, float nod_angle, int timeout_ms);
Function Overview	Control robot head shaking and nodding motion
Parameter Description	shake_angle: Head left-right shake angle, direction: negative left/positive right, unit: rad, range: [-0.5236, 0.5236] nod_angle: Head nodding angle, direction: positive up/negative down, unit: rad, range: [-0.3491, 0.3491] timeout_ms: Timeout duration, default 5000 milliseconds
Return Value	Status::OK indicates success, others indicate failure
Notes	Blocking interface, controls robot head left-right shaking and up-down nodding motion

## 8.2 Type Definitions

### 8.2.1 JoystickCommand —High-Level Motion Control Joystick Command Structure

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

## 8.3 Enumeration Type Definitions

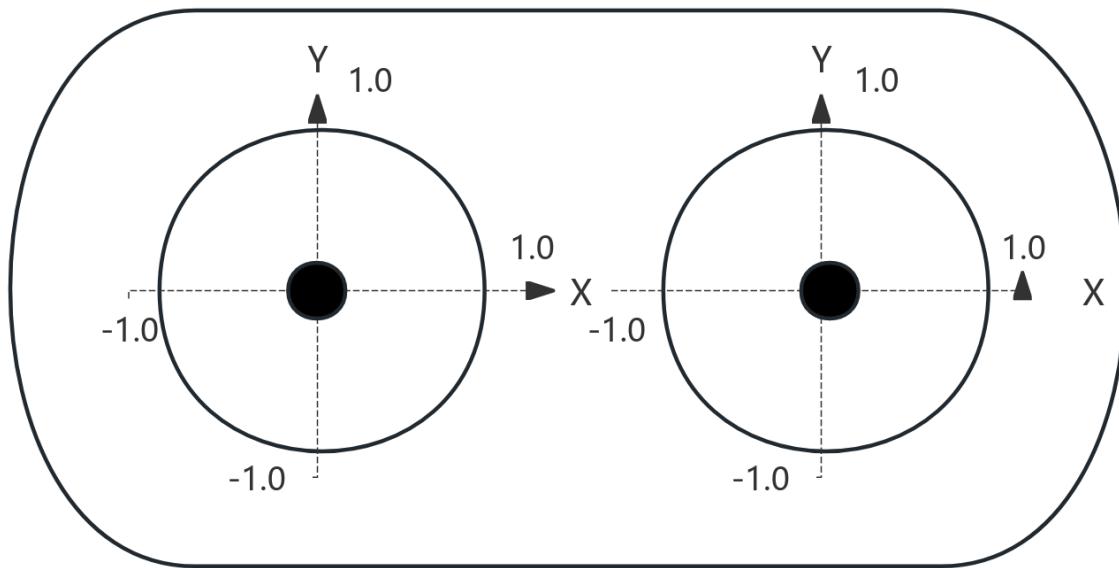
### 8.3.1 GaitMode —Robot Gait Mode Enumeration

Enumeration Value	Value	Description
GAIT_PASSIVE	0	Passive
GAIT_PURE_DAMPER	0	Pure Damper
GAIT_RECOVERY_STAND	1	Recovery stand
GAIT_BALANCE_STAND	46	Balance stand (supports movement)
GAIT_ARM_SWING_WALK	47	Arm swing walking
GAIT_LOWLEVEL_SDK	200	Low-level control SDK mode

### 8.3.2 TrickAction —Trick Action Command Enumeration

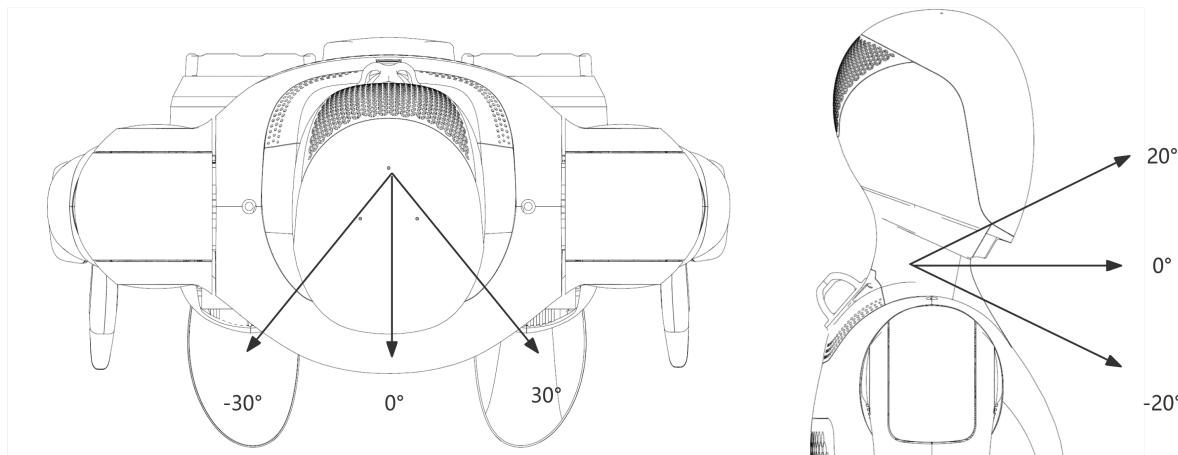
Enumeration Value	Value	Description
ACTION_RECOVERY_STAND	103	Balance stand
ACTION_CELEBRATE	201	Celebrate
ACTION_SHAKE_HAND_REACHOUT	217	Handshake - reach out
ACTION_SHAKE_HAND_WITHDRAW	218	Handshake - withdraw
ACTION_NOD_HEAD	219	Nod head
ACTION_SHAKE_HEAD	220	Shake head
ACTION_CIRCLE_HEAD	221	Circle head
ACTION_GREETING	301	Greeting
ACTION_POINT_GROUND	302	Point ground - reach out
ACTION_POINT_GROUND_WITH_DRAW	303	Point ground - withdraw
ACTION_SPREAD_HAND	304	Spread hand
ACTION_SPREAD_HAND_WITH_DRAW	305	Spread hand - withdraw
ACTION_TURN_AWAY_LEFT_INTRODUCE	306	Turn away left introduce
ACTION_TURN_BACK_LEFT_INTRODUCE	307	Turn back left introduce

## 8.4 Joystick Diagram



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

## 8.5 Head Movement Range Diagram



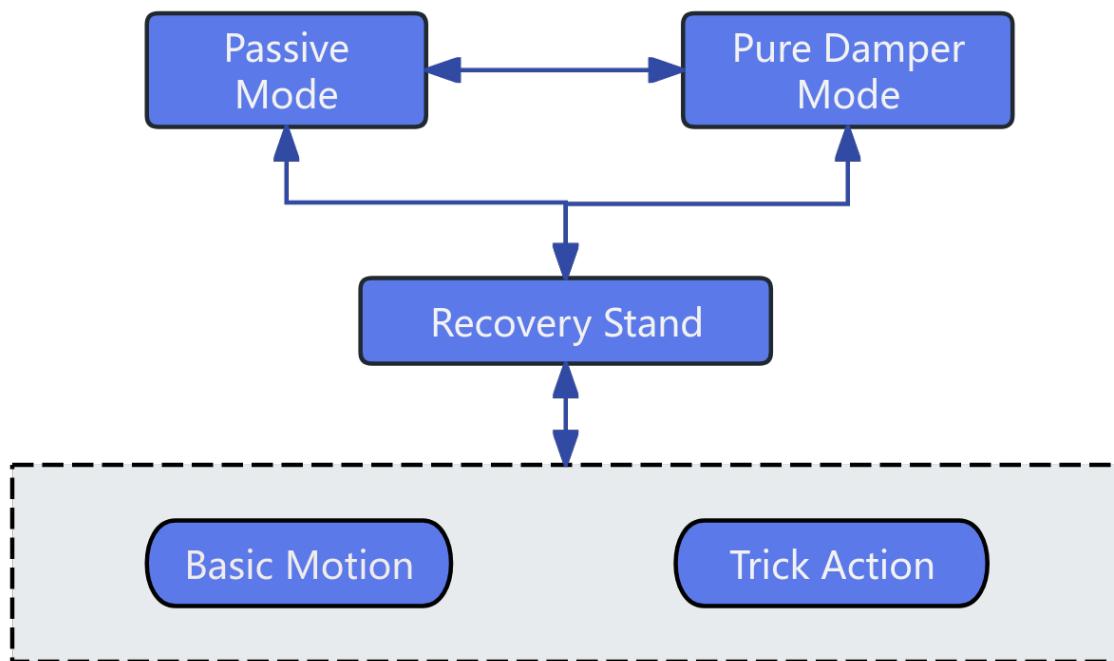
- Head shake range [-30°, 30°] ( $[-0.5236\text{rad}, 0.5236\text{rad}]$ )
- Head nod range [-20°, 20°] ( $[-0.3491\text{rad}, 0.3491\text{rad}]$ )

## 8.6 High-Level Motion Control Robot State Introduction

Robot motion includes recovery stand, balance stand, basic motion, and trick action states. During operation, the robot switches between different states through a state machine to achieve different control tasks. The explanations for each state are as follows:

- **Recovery Stand:** Recovery stand is the state connecting robot suspension landing and balance stand. The robot needs to enter the balance stand state before calling the corresponding motion services to achieve robot control.
- **Balance Stand:** In the balance stand state, you can call various SDK interfaces to implement robot trick actions and basic motion control.
- **Basic Motion:** During motion execution, you can call SDK interfaces to make the robot enter different gaits.
- **Trick Actions:** When entering the special action execution state, other motion control services will be suspended first, waiting for the current action to complete and enter the balance stand state before taking effect again.

Robot state switching mechanism;

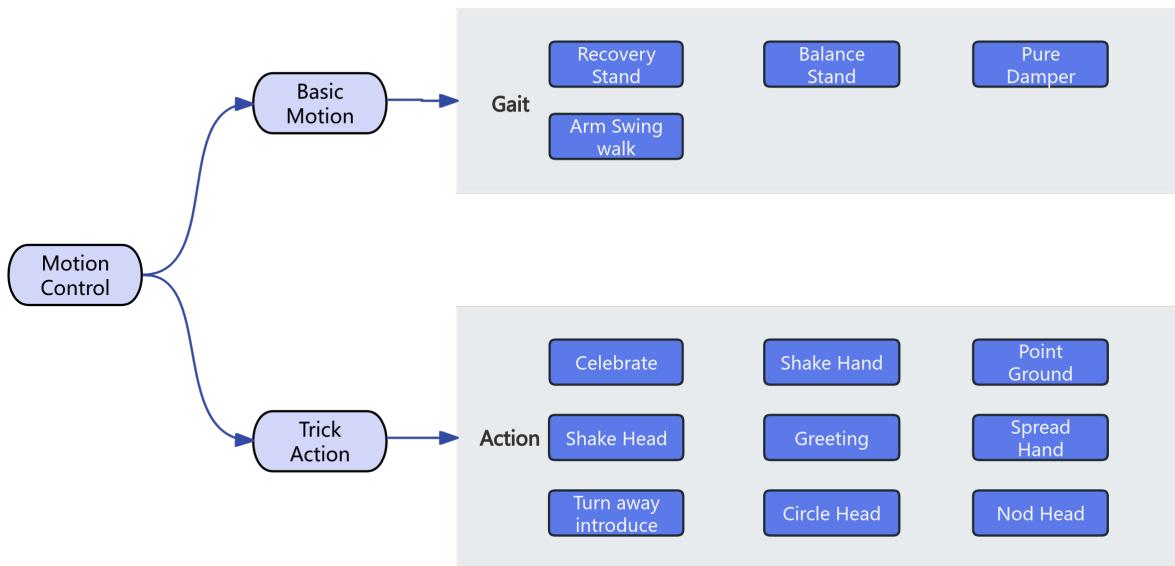


## 8.7 High-Level Motion Control Interface

Robot high-level motion control services can be divided into basic motion control and trick action control.

- In basic motion control services, you can call corresponding interfaces to switch robot walking gaits according to different terrain scenarios and task requirements.

- In trick action control services, you can call corresponding interfaces to implement robot built-in special tricks, such as celebrate, handshake, greeting, etc.



# CHAPTER 9

---

## Low-Level Motion Control Service (C++)

---

---

Provides low-level motion control services for the robot system. Through `LowLevelMotionController`, you can control robot joints and obtain status via topic communication.

### 9.1 Interface Definition

`LowLevelMotionController` is a motion controller for low-level development, supporting direct control and state subscription of motion components such as arms, legs, head, waist, hands, etc., as well as body IMU data acquisition.

#### 9.1.1 `LowLevelMotionController`

Item	Content
Function Name	<code>LowLevelMotionController</code>
Function Declaration	<code>LowLevelMotionController();</code>
Function Overview	Constructor, initializes low-level controller object.
Notes	Constructs internal resources.

### 9.1.2 ~LowLevelMotionController

Item	Content
Function Name	<code>~LowLevelMotionController</code>
Function Declaration	<code>virtual ~LowLevelMotionController();</code>
Function Overview	Destructor, releases resources.
Notes	Cleans up low-level resources.

### 9.1.3 Initialize

Item	Content
Function Name	<code>Initialize</code>
Function Declaration	<code>virtual bool Initialize() override;</code>
Function Overview	Initialize controller, establish low-level connection.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Must be initialized on first call.

### 9.1.4 Shutdown

Item	Content
Function Name	<code>Shutdown</code>
Function Declaration	<code>virtual void Shutdown() override;</code>
Function Overview	Shutdown controller, release low-level resources.
Notes	Used in conjunction with Initialize.

### 9.1.5 SubscribeArmState

Item	Content
Function Name	<code>SubscribeArmState</code>
Function Declaration	<code>void SubscribeArmState(ArmJointStateCallback callback);</code>
Function Overview	Subscribe to arm joint state data
Parameter Description	callback: Data processing function
Notes	Non-blocking interface.

### 9.1.6 UnsubscribeArmState

Item	Content
Function Name	UnsubscribeArmState
Function Declaration	<code>void UnsubscribeArmState();</code>
Function Overview	Unsubscribe from arm joint state data
Notes	Non-blocking interface. Used in conjunction with SubscribeArmState.

### 9.1.7 PublishArmCommand

Item	Content
Function Name	PublishArmCommand
Function Declaration	<code>Status PublishArmCommand(const JointCommand&amp; command);</code>
Function Overview	Publish arm control commands
Parameter Description	command: Target position/velocity etc.
Return Value	true indicates success, false indicates failure.
Notes	Non-blocking interface.

### 9.1.8 SubscribeLegState

Item	Content
Function Name	SubscribeLegState
Function Declaration	<code>void SubscribeLegState(LegJointStateCallback callback);</code>
Function Overview	Subscribe to leg joint state data
Parameter Description	callback: Data processing function
Notes	Non-blocking interface.

### 9.1.9 UnsubscribeLegState

Item	Content
Function Name	UnsubscribeLegState
Function Declaration	<code>void UnsubscribeLegState();</code>
Function Overview	Unsubscribe from leg joint state data
Notes	Non-blocking interface. Used in conjunction with SubscribeLegState.

### 9.1.10 PublishLegCommand

Item	Content
Function Name	PublishLegCommand
Function Declaration	<code>Status PublishLegCommand(const JointCommand&amp; command);</code>
Function Overview	Publish leg control commands
Parameter Description	command: Target position/velocity etc.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Non-blocking interface, called when publishing or subscribing.

### 9.1.11 SubscribeHeadState

Item	Content
Function Name	SubscribeHeadState
Function Declaration	<code>void SubscribeHeadState(HeadJointStateCallback callback);</code>
Function Overview	Subscribe to head joint state data
Parameter Description	callback: Data processing function
Notes	Non-blocking interface.

### 9.1.12 UnsubscribeHeadState

Item	Content
Function Name	UnsubscribeHeadState
Function Declaration	<code>void UnsubscribeHeadState();</code>
Function Overview	Unsubscribe from head joint state data
Notes	Non-blocking interface. Used in conjunction with SubscribeHeadState.

### 9.1.13 PublishHeadCommand

Item	Content
Function Name	PublishHeadCommand
Function Declaration	<code>Status PublishHeadCommand(const JointCommand&amp; command);</code>
Function Overview	Publish head control commands
Parameter Description	command: Target position/velocity etc.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Non-blocking interface.

### 9.1.14 SubscribeWaistState

Item	Content
Function Name	SubscribeWaistState
Function Declaration	<code>void SubscribeWaistState(WaistJointStateCallback callback);</code>
Function Overview	Subscribe to waist joint state data
Parameter Description	callback: Data processing function
Notes	Non-blocking interface.

### 9.1.15 UnsubscribeWaistState

Item	Content
Function Name	UnsubscribeWaistState
Function Declaration	<code>void UnsubscribeWaistState();</code>
Function Overview	Unsubscribe from waist joint state data
Notes	Non-blocking interface. Used in conjunction with SubscribeWaistState.

### 9.1.16 PublishWaistCommand

Item	Content
Function Name	PublishWaistCommand
Function Declaration	<code>Status PublishWaistCommand(const JointCommand&amp; command);</code>
Function Overview	Publish waist control commands
Parameter Description	command: Target position/velocity etc.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Non-blocking interface.

### 9.1.17 SubscribeHandState

Item	Content
Function Name	SubscribeHandState
Function Declaration	<code>void SubscribeHandState(HandStateCallback callback);</code>
Function Overview	Subscribe to hand state data
Parameter Description	callback: Data processing function
Notes	Non-blocking interface.

### 9.1.18 UnsubscribeHandState

Item	Content
Function Name	UnsubscribeHandState
Function Declaration	<code>void UnsubscribeHandState();</code>
Function Overview	Unsubscribe from hand state data
Notes	Non-blocking interface. Used in conjunction with SubscribeHandState.

### 9.1.19 PublishHandCommand

Item	Content
Function Name	PublishHandCommand
Function Declaration	<code>Status PublishHandCommand(const HandCommand&amp; command);</code>
Function Overview	Publish hand control commands
Parameter Description	command: Hand joint target position etc.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Non-blocking interface.

### 9.1.20 SubscribeBodyImu

Item	Content
Function Name	SubscribeBodyImu
Function Declaration	<code>void SubscribeBodyImu(const BodyImuCallback&amp; callback);</code>
Function Overview	Subscribe to body IMU data
Parameter Description	callback: IMU data processing function
Notes	Non-blocking interface.

### 9.1.21 UnsubscribeBodyImu

Item	Content
Function Name	UnsubscribeBodyImu
Function Declaration	<code>void UnsubscribeBodyImu();</code>
Function Overview	Unsubscribe from body IMU data
Notes	Non-blocking interface. Used in conjunction with SubscribeBodyImu.

## 9.2 Type Definitions

### 9.2.1 SingleHandJointCommand —Single Hand Joint Control Command

Field Name	Type	Description
operation_mode	int16_t	Control word
pos	vector<float>;	Desired position array (7 DOF, unit: rad)

- Dexterous hand operation\_mode needs to switch from mode: 200 to mode: 4 to start, and then send commands;
- Dexterous hand joint command range: [0-1000]

### 9.2.2 HandCommand —Complete Hand Control Command

Field Name	Type	Description
times-tamp	int64_t	Timestamp (unit: nanoseconds)
cmd	vector<SingleHandJointCommand>;	Control command array, left hand and right hand in order

### 9.2.3 SingleHandJointState —Single Hand Joint State

Field Name	Type	Description
status_word	int16_t	Status word
pos	vector<float>;	Current position (unit: rad)
toq	vector<float>;	Current torque (unit: Nm)
cur	vector<float>;	Current current (unit: A)
error_code	int16_t	Error code

### 9.2.4 HandState —Complete Hand State Information

Field Name	Type	Description
timestamp	int64_t	Timestamp (unit: nanoseconds)
state	vector<SingleHandJointState>;	All hand joint states (left hand, right hand order)

## 9.2.5 SingleJointCommand —Single Joint Control Command

Field Name	Type	Description
operation_mode	int16_t	Control mode identifier: • 200 = Standby mode • 3 = Parallel triple-loop control (MIT mode) • 4 = Serial triple-loop control
pos	float	Target position (unit: rad)
vel	float	Velocity limit (unit: rad/s)
toq	float	Torque limit (unit: Nm)
kp	float	Position loop proportional gain
kd	float	Velocity loop derivative gain

- For joints 1–5 of the arm, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 6–7 of the arm, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–2 of the head, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–2 of the waist, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–6 of the legs, `operation_mode` must switch from mode 200 (standby) to mode 3 (parallel triple-loop control, MIT mode) before sending commands.

## 9.2.6 JointCommand —Joint Control Command

Lower limbs contain 12 joint items, upper limbs 14, head 2, waist 2:

Field Name	Type	Description
timestamp	int64_t	Timestamp (unit: nanoseconds)
joints	vector<SingleJointCommand>;	Control commands for all joints

## 9.2.7 SingleJointState —Single Joint State Information

Field Name	Type	Description
sta-tus_word	int16_t	Current joint state (custom state machine encoding)
posH	float	Actual position (high precision encoder reading)
posL	float	Actual position (low precision encoder reading, for redundancy check)
vel	float	Current velocity (unit: rad/s or m/s, automatically switches based on joint type)
toq	float	Current torque (unit: Nm)
current	float	Current current (unit: A)
err_code	int16_t	Error code (such as encoder exception, motor overcurrent, etc., see error code reference table)

## 9.2.8 JointState —Joint State Data

Lower limbs contain 12 joint items, upper limbs 14, head 2, waist 2:

Field Name	Type	Description
timestamp	int64_t	Timestamp (unit: nanoseconds)
joints	vector<SingleJointState>	State data for all joints

## 9.2.9 Joint Motor Order

### Head Joints

Index	Joint Name
0	head_yaw_joint
1	head_pitch_joint

**Arm Joints**

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

**Waist Joints**

Index	Joint Name
0	waist_roll_joint
1	waist_yaw_joint

**Leg Joints**

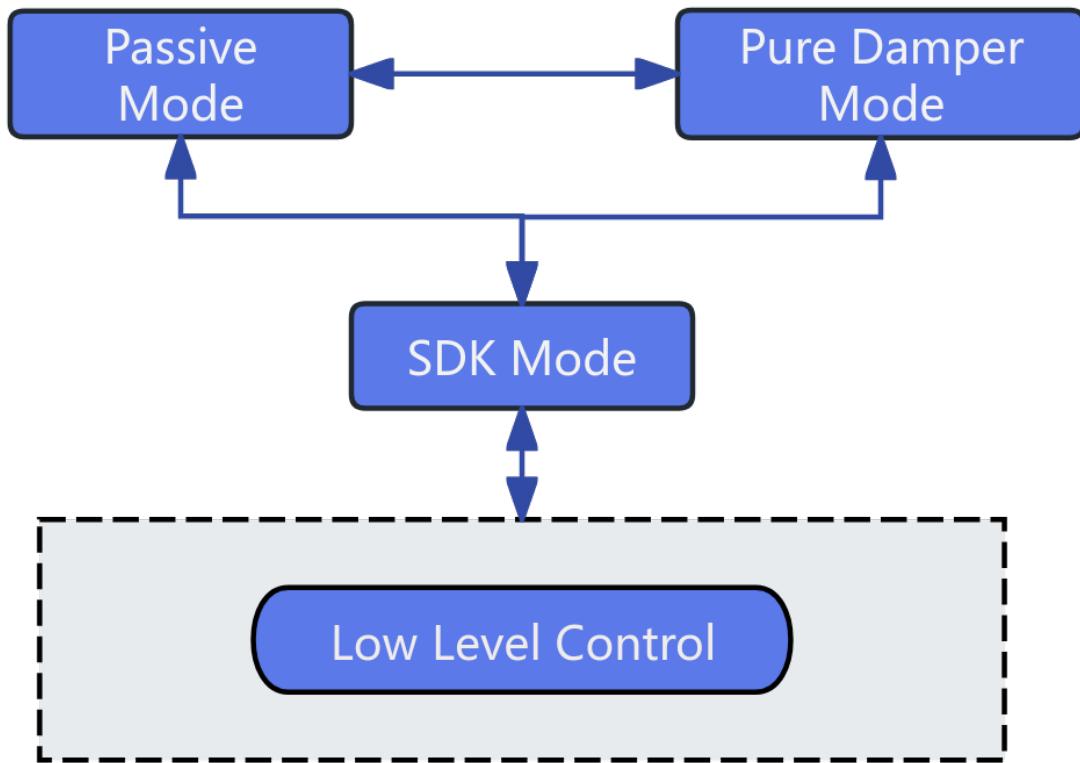
Index	Joint Name
0	left_hip_roll_joint
1	left_hip_yaw_joint
2	left_hip_pitch_joint
3	left_knee_pitch_joint
4	left_ankle_pitch_joint
5	left_ankle_roll_joint
6	right_hip_roll_joint
7	right_hip_yaw_joint
8	right_hip_pitch_joint
9	right_knee_pitch_joint
10	right_ankle_pitch_joint
11	right_ankle_roll_joint

## 9.3 URDF Reference

Robot URDF

## 9.4 Low-Level Motion Control Robot State Introduction

Robot low-level motion mainly develops three-loop control of joints for developers to perform secondary development of robot motion capabilities, basic control state switching mechanism:



## 9.5 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 10

---

## Sensor Control Service (C++)

---

---

Provides robot system sensor (LiDAR/RGBD camera/trinocular camera) services. Through SensorController, you can control robot sensors and obtain status via RPC and topic communication.

### 10.1 Interface Definition

`SensorController` is a management class that encapsulates various robot sensors, supporting initialization, control, and data subscription for LiDAR, RGBD cameras, and trinocular cameras.

**⚠️ Notice:** Current trinocular and RGBD sensor interfaces are not fully support yet, please do not use.

#### 10.1.1 SensorController

Item	Content
Function Name	<code>SensorController</code>
Function Declaration	<code>SensorController();</code>
Function Overview	Constructor, initializes sensor controller object.
Notes	Constructs internal state.

### 10.1.2 ~SensorController

Item	Content
Function Name	<code>~SensorController</code>
Function Declaration	<code>virtual ~SensorController();</code>
Function Overview	Destructor, releases all sensor resources.
Notes	Sensors should be closed before calling.

### 10.1.3 Initialize

Item	Content
Function Name	<code>Initialize</code>
Function Declaration	<code>bool Initialize();</code>
Function Overview	Initialize controller, including resource allocation and driver loading.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Object must be constructed before calling.

### 10.1.4 Shutdown

Item	Content
Function Name	<code>Shutdown</code>
Function Declaration	<code>void Shutdown();</code>
Function Overview	Close all sensor connections and release resources.
Notes	Used in conjunction with Initialize.

### 10.1.5 OpenLidar

Item	Content
Function Name	<code>OpenLidar</code>
Function Declaration	<code>Status OpenLidar();</code>
Function Overview	Open LiDAR.
Return Value	<code>Status::OK</code> indicates success, others indicate failure.
Notes	Blocking interface.

### 10.1.6 CloseLidar

Item	Content
Function Name	CloseLidar
Function Declaration	Status CloseLidar();
Function Overview	Close LiDAR.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, used in conjunction with open function.

### 10.1.7 OpenHeadRgbdCamera

Item	Content
Function Name	OpenHeadRgbdCamera
Function Declaration	Status OpenHeadRgbdCamera();
Function Overview	Open head RGBD camera (head and waist).
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface.

### 10.1.8 CloseHeadRgbdCamera

Item	Content
Function Name	CloseHeadRgbdCamera
Function Declaration	Status CloseHeadRgbdCamera();
Function Overview	Close head RGBD camera.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, used in conjunction with open function.

### 10.1.9 OpenWaistRgbdCamera

Item	Content
Function Name	OpenWaistRgbdCamera
Function Declaration	Status OpenWaistRgbdCamera();
Function Overview	Open waist RGBD camera (head and waist).
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface.

### 10.1.10 CloseWaistRgbdCamera

Item	Content
Function Name	CloseWaistRgbdCamera
Function Declaration	Status CloseWaistRgbdCamera();
Function Overview	Close waist RGBD camera.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, used in conjunction with open function.

### 10.1.11 OpenTrinocularCamera

Item	Content
Function Name	OpenTrinocularCamera
Function Declaration	Status OpenTrinocularCamera();
Function Overview	Open trinocular camera.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface.

### 10.1.12 CloseTrinocularCamera

Item	Content
Function Name	CloseTrinocularCamera
Function Declaration	Status CloseTrinocularCamera();
Function Overview	Close trinocular camera.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface.

### 10.1.13 SubscribeLidarImu

Item	Content
Function Name	SubscribeLidarImu
Function Declaration	void SubscribeLidarImu(const LidarImuCallback& callback);
Function Overview	Subscribe to LiDAR IMU data
Parameter Description	callback: Data processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.14 UnsubscribeLidarImu

Item	Content
Function Name	UnsubscribeLidarImu
Function Declaration	<code>void UnsubscribeLidarImu();</code>
Function Overview	Unsubscribe from LiDAR IMU data
Notes	Non-blocking interface. Used in conjunction with SubscribeLidarImu.

### 10.1.15 SubscribeLidarPointCloud

Item	Content
Function Name	SubscribeLidarPointCloud
Function Declaration	<code>void SubscribeLidarPointCloud(const LidarPointCloudCallback&amp; callback);</code>
Function Overview	Subscribe to LiDAR point cloud data
Parameter Description	callback: Point cloud processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.16 UnsubscribeLidarPointCloud

Item	Content
Function Name	UnsubscribeLidarPointCloud
Function Declaration	<code>void UnsubscribeLidarPointCloud();</code>
Function Overview	Unsubscribe from LiDAR point cloud data
Notes	Non-blocking interface. Used in conjunction with SubscribeLidarPointCloud.

### 10.1.17 SubscribeHeadRgbdColorImage

Item	Content
Function Name	SubscribeHeadRgbdColorImage
Function Declaration	<code>void SubscribeHeadRgbdColorImage(const RgbdImageCallback&amp; callback);</code>
Function Overview	Subscribe to head RGBD color image data
Parameter Description	callback: Image processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.18 UnsubscribeHeadRgbIColorImage

Item	Content
Function Name	UnsubscribeHeadRgbIColorImage
Function Declaration	<code>void UnsubscribeHeadRgbIColorImage();</code>
Function Overview	Unsubscribe from head RGBD color image data
Notes	Non-blocking interface. Used in conjunction with SubscribeHeadRgbIColorImage.

### 10.1.19 SubscribeHeadRgbIColorCameraInfo

Item	Content
Function Name	SubscribeHeadRgbIColorCameraInfo
Function Declaration	<code>void SubscribeHeadRgbIColorCameraInfo(const RgbdCameraInfoCallback&amp; callback);</code>
Function Overview	Subscribe to head RGBD color camera parameters
Parameter Description	callback: Parameter processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.20 UnsubscribeHeadRgbIColorCameraInfo

Item	Content
Function Name	UnsubscribeHeadRgbIColorCameraInfo
Function Declaration	<code>void UnsubscribeHeadRgbIColorCameraInfo();</code>
Function Overview	Unsubscribe from head RGBD color camera parameters
Notes	Non-blocking interface. Used in conjunction with SubscribeHeadRgbIColorCameraInfo.

### 10.1.21 SubscribeHeadRgbDepthImage

Item	Content
Function Name	SubscribeHeadRgbDepthImage
Function Declaration	<code>void SubscribeHeadRgbDepthImage(const RgbdImageCallback&amp; callback);</code>
Function Overview	Subscribe to head RGBD depth image data
Parameter Description	callback: Image processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.22 UnsubscribeHeadRgbDepthImage

Item	Content
Function Name	UnsubscribeHeadRgbDepthImage
Function Declaration	<code>void UnsubscribeHeadRgbDepthImage();</code>
Function Overview	Unsubscribe from head RGBD depth image data
Notes	Non-blocking interface. Used in conjunction with SubscribeHeadRgbDepthImage.

### 10.1.23 SubscribeHeadRgbDepthCameraInfo

Item	Content
Function Name	SubscribeHeadRgbDepthCameraInfo
Function Declaration	<code>void SubscribeHeadRgbDepthCameraInfo(const RgbdCameraInfoCallback&amp; callback);</code>
Function Overview	Subscribe to head RGBD depth camera parameters
Parameter Description	callback: Parameter processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.24 UnsubscribeHeadRgbDepthCameraInfo

Item	Content
Function Name	UnsubscribeHeadRgbDepthCameraInfo
Function Declaration	<code>void UnsubscribeHeadRgbDepthCameraInfo();</code>
Function Overview	Unsubscribe from head RGBD depth camera parameters
Notes	Non-blocking interface. Used in conjunction with SubscribeHeadRgbDepthCameraInfo.

### 10.1.25 SubscribeWaistRgbColorImage

Item	Content
Function Name	SubscribeWaistRgbColorImage
Function Declaration	<code>void SubscribeWaistRgbColorImage(const RgbdImageCallback&amp; callback);</code>
Function Overview	Subscribe to waist RGBD color image data
Parameter Description	callback: Image processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.26 UnsubscribeWaistRgbIColorImage

Item	Content
Function Name	UnsubscribeWaistRgbIColorImage
Function Declaration	<code>void UnsubscribeWaistRgbIColorImage();</code>
Function Overview	Unsubscribe from waist RGBD color image data
Notes	Non-blocking interface. Used in conjunction with SubscribeWaistRgbIColorImage.

### 10.1.27 SubscribeWaistRgbIColorCameraInfo

Item	Content
Function Name	SubscribeWaistRgbIColorCameraInfo
Function Declaration	<code>void SubscribeWaistRgbIColorCameraInfo(const RgbdCameraInfoCallback&amp; callback);</code>
Function Overview	Subscribe to waist RGBD color camera parameters
Parameter Description	callback: Parameter processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.28 UnsubscribeWaistRgbIColorCameraInfo

Item	Content
Function Name	UnsubscribeWaistRgbIColorCameraInfo
Function Declaration	<code>void UnsubscribeWaistRgbIColorCameraInfo();</code>
Function Overview	Unsubscribe from waist RGBD color camera parameters
Notes	Non-blocking interface. Used in conjunction with SubscribeWaistRgbIColorCameraInfo.

### 10.1.29 SubscribeWaistRgbDepthImage

Item	Content
Function Name	SubscribeWaistRgbDepthImage
Function Declaration	<code>void SubscribeWaistRgbDepthImage(const RgbdImageCallback&amp; callback);</code>
Function Overview	Subscribe to waist RGBD depth image data
Parameter Description	callback: Image processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.30 UnsubscribeWaistRgbDepthImage

Item	Content
Function Name	UnsubscribeWaistRgbDepthImage
Function Declaration	<code>void UnsubscribeWaistRgbDepthImage();</code>
Function Overview	Unsubscribe from waist RGBD depth image data
Notes	Non-blocking interface. Used in conjunction with SubscribeWaistRgbDepthImage.

### 10.1.31 SubscribeWaistRgbDepthCameraInfo

Item	Content
Function Name	SubscribeWaistRgbDepthCameraInfo
Function Declaration	<code>void SubscribeWaistRgbDepthCameraInfo(const RgbdCameraInfoCallback&amp; callback);</code>
Function Overview	Subscribe to waist RGBD depth camera parameters
Parameter Description	callback: Parameter processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.32 UnsubscribeWaistRgbDepthCameraInfo

Item	Content
Function Name	UnsubscribeWaistRgbDepthCameraInfo
Function Declaration	<code>void UnsubscribeWaistRgbDepthCameraInfo();</code>
Function Overview	Unsubscribe from waist RGBD depth camera parameters
Notes	Non-blocking interface. Used in conjunction with SubscribeWaistRgbDepthCameraInfo.

### 10.1.33 SubscribeTrinocularImage

Item	Content
Function Name	SubscribeTrinocularImage
Function Declaration	<code>void SubscribeTrinocularImage(const TrinocularImageCallback&amp; callback);</code>
Function Overview	Subscribe to trinocular camera image frames
Parameter Description	callback: Image frame processing function after receiving data
Notes	Non-blocking interface, callback function will be called when data is updated.

### 10.1.34 UnsubscribeTrinocularImage

Item	Content
Function Name	UnsubscribeTrinocularImage
Function Declaration	<code>void UnsubscribeTrinocularImage();</code>
Function Overview	Unsubscribe from trinocular camera image frames
Notes	Non-blocking interface. Used in conjunction with SubscribeTrinocularImage.

## 10.2 Type Definitions

### 10.2.1 Imu —IMU Data Structure

Field Name	Type	Description
timestamp	<code>int64_t</code>	Timestamp (unit: nanoseconds)
orientation[4]	<code>std::array&lt;double, 4&gt;;</code>	Attitude quaternion (w, x, y, z)
angu-	<code>std::array&lt;double, 3&gt;;</code>	Angular velocity (unit: rad/s)
lin-	<code>std::array&lt;double, 3&gt;;</code>	Linear acceleration (unit: m/s <sup>2</sup> )
ear_acceleration[3]	<code>float</code>	Temperature (unit: Celsius or other, unit should be specified)

### 10.2.2 Header —Common Message Header Structure

Field Name	Type	Description
stamp	<code>int64_t</code>	Timestamp (unit: nanoseconds)
frame_id	<code>std::string</code>	Coordinate frame name

### 10.2.3 PointField —Point Field Description

Field Name	Type	Description
name	<code>std::string</code>	Field name (such as x, y, z, intensity)
offset	<code>int32_t</code>	Starting byte offset
datatype	<code>int8_t</code>	Data type (corresponding constant)
count	<code>int32_t</code>	Number of elements this field contains per point

### 10.2.4 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	vector<PointField>;	Point field array
is_bigendian	bool	Byte order
point_step	int32_t	Number of bytes per point
row_step	int32_t	Number of bytes per row
data	vector<uint8_t>;	Raw point cloud byte data
is_dense	bool	Whether dense point cloud

### 10.2.5 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 (such as rgb8, mono8)
is_bigendian	bool	Whether big-endian mode
step	int32_t	Number of bytes per image row
data	vector<uint8_t>;	Raw image byte data

## 10.2.6 CameraInfo —Camera Intrinsic Parameters 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 (such as plumb_bob)
D	vector<double>;	Distortion parameter array
K	std::array<double, 9>;	Camera intrinsic parameter matrix
R	std::array<double, 9>;	Rectification matrix
P	std::array<double, 12>;	Projection matrix
binning_x	int32_t	Horizontal binning coefficient
binning_y	int32_t	Vertical binning coefficient
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 perform rectification

## 10.2.7 TrinocularCameraFrame —Trinocular Camera Frame Data Structure

Field Name	Type	Description
header	Header	Common message header
vin_time	int64_t	Image acquisition time (nanoseconds)
decode_time	int64_t	Image decoding time (nanoseconds)
imgfl_array	vector<uint8_t>;	Left eye image data
imgf_array	vector<uint8_t>;	Center eye image data
imgfr_array	vector<uint8_t>;	Right eye image data

## 10.3 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 11

---

## Audio Control Service (C++)

---

---

Provides robot system audio service controller. Through `AudioController`, you can control robot audio and obtain status via RPC communication.

### 11.1 Interface Definition

`AudioController` is a C++ class that encapsulates audio control functionality, mainly used for audio playback control, TTS playback, volume setting and query, etc.

#### 11.1.1 `AudioController`

Item	Content
Function Name	<code>AudioController</code>
Function Declaration	<code>AudioController();</code>
Function Overview	Initialize audio controller object, construct internal state, allocate resources, etc.
Notes	Constructs internal state.

### 11.1.2 ~AudioController

Item	Content
Function Name	<code>~AudioController</code>
Function Declaration	<code>~AudioController();</code>
Function Overview	Release audio controller resources, ensure playback stops and clean up underlying resources.
Notes	Ensure safe resource release.

### 11.1.3 Initialize

Item	Content
Function Name	<code>Initialize</code>
Function Declaration	<code>bool Initialize();</code>
Function Overview	Initialize audio control module, prepare playback resources and devices.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Used in conjunction with <code>Shutdown()</code> .

### 11.1.4 Shutdown

Item	Content
Function Name	<code>Shutdown</code>
Function Declaration	<code>void Shutdown();</code>
Function Overview	Close audio controller and release resources.
Notes	Ensure called before destruction.

### 11.1.5 Play

Item	Content
Function Name	<code>Play</code>
Function Declaration	<code>Status Play(const TtsCommand&amp; cmd, int timeout_ms);</code>
Function Overview	Play TTS (Text-to-Speech) voice command.
Parameter Description	<code>cmd</code> : TTS command, containing text, speech rate, tone, etc.
Return Value	<code>Status::OK</code> indicates success, others indicate failure.
Notes	Blocking interface, ensure module is initialized before calling.

### 11.1.6 Stop

Item	Content
Function Name	Stop
Function Declaration	Status Stop();
Function Overview	Stop current audio playback.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, typically used to interrupt speech.

### 11.1.7 SetVolume

Item	Content
Function Name	SetVolume
Function Declaration	Status SetVolume(int volume);
Function Overview	Set audio output volume.
Parameter Description	volume: Volume value, typically range 0~100.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, takes effect immediately after setting.

### 11.1.8 GetVolume

Item	Content
Function Name	GetVolume
Function Declaration	Status GetVolume(int& volume);
Function Overview	Get current audio output volume.
Parameter Description	volume: Returns current volume value by reference.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, check return value before using volume.

### 11.1.9 OpenAudioStream

Item	Content
Function Name	OpenAudioStream
Function Declaration	Status OpenAudioStream();
Function Overview	Open audio stream, prepare for audio playback.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, check return value to ensure successful execution

### 11.1.10 CloseAudioStream

Item	Content
Function Name	CloseAudioStream
Function Declaration	Status CloseAudioStream();
Function Overview	Close audio stream.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, check return value to ensure successful execution

### 11.1.11 SubscribeOriginAudioStream

Item	Content
Function Name	SubscribeOriginAudioStream
Function Declaration	void SubscribeOriginAudioStream(const OriginAudioStreamCallback callback);
Function Overview	Subscribe to original audio stream data.
Parameter Description	callback: Audio stream data callback function.
Notes	Non-blocking interface.

### 11.1.12 UnsubscribeOriginAudioStream

Item	Content
Function Name	UnsubscribeOriginAudioStream
Function Declaration	void UnsubscribeOriginAudioStream();
Function Overview	Unsubscribe from original audio stream data.
Notes	Non-blocking interface. Used in conjunction with SubscribeOriginAudioStream.

### 11.1.13 SubscribeBfAudioStream

Item	Content
Function Name	SubscribeBfAudioStream
Function Declaration	void SubscribeBfAudioStream(const BfAudioStreamCallback callback);
Function Overview	Subscribe to BF audio stream data.
Parameter Description	callback: Audio stream data callback function.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Non-blocking interface.

### 11.1.14 OpenWakeupStatusStream

Item	Content
Function Name	OpenWakeupStatusStream
Function Declaration	Status OpenWakeupStatusStream();
Function Overview	Enable voice wake-up status stream.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, check return value to ensure successful execution.

### 11.1.15 CloseWakeupStatusStream

Item	Content
Function Name	CloseWakeupStatusStream
Function Declaration	Status CloseWakeupStatusStream();
Function Overview	Disable voice wake-up status stream.
Return Value	Status::OK indicates success, others indicate failure.
Notes	Blocking interface, used in conjunction with OpenWakeupStatusStream.

### 11.1.16 SubscribeWakeupStatus

Item	Content
Function Name	SubscribeWakeupStatus
Function Declaration	void SubscribeWakeupStatus(const WakeupStatusCallback callback);
Function Overview	Subscribe to voice wake-up status data.
Parameter Description	callback: Processing callback function after receiving wake-up status.
Notes	Non-blocking interface, callback function will be called when wake-up status is updated.

### 11.1.17 UnsubscribeWakeupStatus

Item	Content
Function Name	UnsubscribeWakeupStatus
Function Declaration	void UnsubscribeWakeupStatus();
Function Overview	Unsubscribe from voice wake-up status data.
Notes	Non-blocking interface. Used in conjunction with SubscribeWakeupStatus.

## 11.2 Type Definitions

### 11.2.1 TtsPriority —TTS Playback Priority Level

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

Enumeration Value	Description
TtsPriority::HIGH	Highest priority, e.g.: low battery warning, emergency alerts
TtsPriority::MIDDLE	Medium priority, e.g.: system prompts, status announcements
TtsPriority::LOW	Lowest priority, e.g.: daily voice conversation, background announcements

### 11.2.2 TtsMode —Task Scheduling Strategy Under Same Priority

Used to fine-tune control of playback order and clearing logic for multiple TTS tasks under the same priority condition.

Enumeration Value	Description
TtsMode::CLEARTOP	Clear all tasks of current priority (including currently playing and waiting queue), immediately play this request
TtsMode::ADD	Append this request to the end of current priority queue, play in order (without interrupting current playback)
TtsMode::CLEARBUFF	Clear unplayed requests in queue, keep current playback, then play this request

## 11.3 Structure Definitions

### 11.3.1 TtsCommand —TTS Playback Command Structure

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

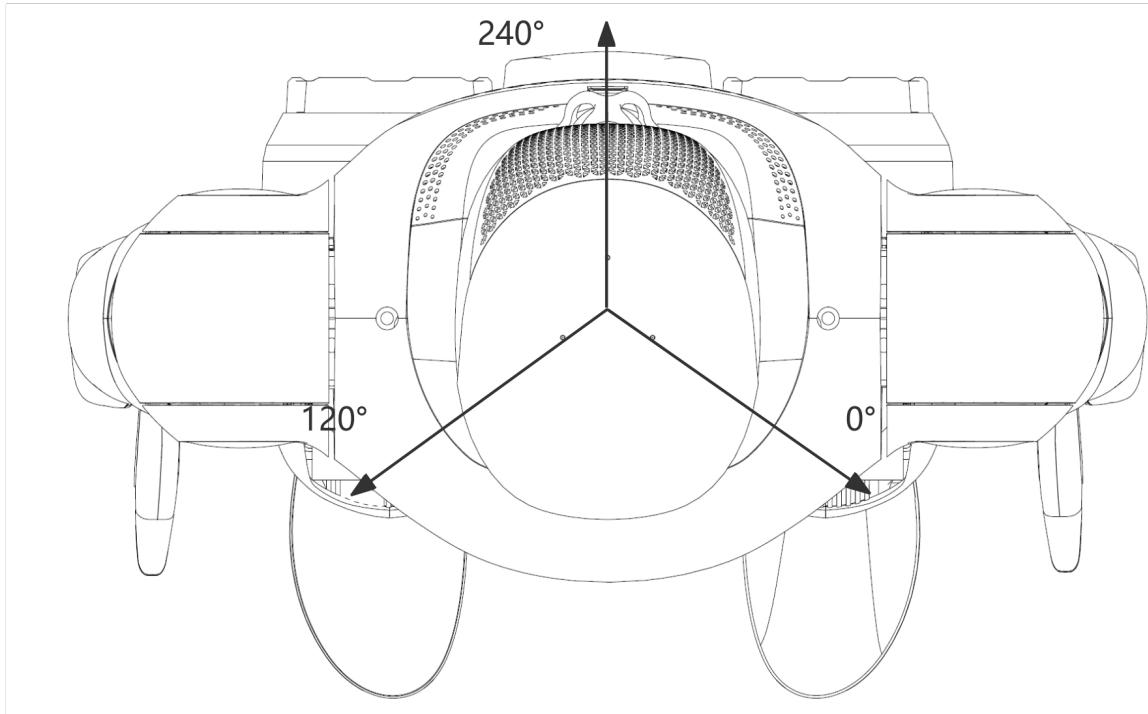
Field Name	Type	Description
id	std::string	TTS task unique ID, e.g. "id_01", used to track playback status
content	std::string	Text content to be played, e.g. "Hello, welcome to the intelligent voice system."
priority	TtsPriority	Playback priority, controls whether to interrupt currently playing lower priority speech
mode	TtsMode	Scheduling strategy under same priority, controls whether tasks are appended, overridden, etc.

### 11.3.2 AudioStream - Audio Stream Structure

Describes original audio stream or BF audio stream data.

Field Name	Type	Description
data_length	int32_t	Data length
raw_data	std::vector<uint8_t>;	Audio stream data

## 11.4 DOA



### 11.5 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 12

---

## State Monitor Service (C++)

---

---

Provides robot state monitoring interface class. Through StateMonitor, you can provide state query and other interfaces.

### 12.1 Interface Definition

`StateMonitor` is a robot system state monitoring management class, typically used to control robot state management and support state queries.

#### 12.1.1 `StateMonitor`

Item	Content
Function Name	<code>StateMonitor</code>
Function Declaration	<code>StateMonitor();</code>
Function Overview	Constructor, initializes state monitor object.
Notes	Constructs internal state.

### 12.1.2 ~StateMonitor

Item	Content
Function Name	<code>~StateMonitor</code>
Function Declaration	<code>virtual ~StateMonitor();</code>
Function Overview	Destructor, releases resources.
Notes	Sensors should be closed before calling.

### 12.1.3 Initialize

Item	Content
Function Name	<code>Initialize</code>
Function Declaration	<code>bool Initialize();</code>
Function Overview	Initialize controller, including resource allocation and driver loading.
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure.
Notes	Object must be constructed before calling.

### 12.1.4 Shutdown

Item	Content
Function Name	<code>Shutdown</code>
Function Declaration	<code>void Shutdown();</code>
Function Overview	Close and release resources.
Notes	Used in conjunction with <code>Initialize</code> .

### 12.1.5 GetCurrentState

Item	Content
Function Name	<code>GetCurrentState</code>
Function Declaration	<code>Status GetcurrentState(RobotState&amp; robot_state);</code>
Function Overview	Get current robot aggregated state
Return Value	<code>Status::OK</code> indicates success, others indicate failure.
Notes	Non-blocking interface, gets recently monitored robot state data, including BMS/fault state monitoring, etc., will be iteratively expanded

### 12.1.6 RobotState —Robot State Data Structure

Used to represent overall robot state information:

Field Name	Type	Description
faults	vector<Fault>;	Fault information list
bms_data	BmsData	Battery management system data

### 12.1.7 BmsData —Battery Management System Data Structure

Represents battery state information:

Field Name	Type	Description
battery_percentage	float	Current battery remaining charge (percentage, 0~100)
battery_health	float	Battery health status (higher is better)
battery_state	BatteryState	Battery state (see enumeration below)
power_supply_status	PowerSupplyStatus	Battery charge/discharge state (see enumeration below)

### 12.1.8 Enumeration Type Definitions

#### BatteryState —Battery State Enumeration Type

Used to represent current battery state, for battery state judgment and processing in the system:

Enumeration Value	Value	Description
BatteryState::UNKNOWN	0	Unknown state
BatteryState::GOOD	1	Battery state good
BatteryState::OVERHEAT	2	Battery overheating
BatteryState::DEAD	3	Battery damaged
BatteryState::OVERVOLTAGE	4	Battery overvoltage
BatteryState::UNSPEC_FAILURE	5	Unknown failure
BatteryState::COLD	6	Battery overcooled
BatteryState::WATCHDOG_TIMER_EXPIRE	7	Watchdog timer expired
BatteryState::SAFETY_TIMER_EXPIRE	8	Safety timer expired

#### PowerSupplyStatus —Battery Charge/Discharge State

Used to represent current battery charge/discharge state:

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

## 12.2 Error Code Mapping Table

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

continues on next page

Table 1 – continued from previous page

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



# CHAPTER 13

---

## SLAM Navigation Control Service (C++)

---

---

Provides SLAM (Simultaneous Localization and Mapping) and navigation control services for robot systems.

Through `SlamNavController`, RPC communication can be used to control robot mapping, localization, navigation and other functions.

### 13.1 Interface Definition

`SlamNavController` is a controller for SLAM and navigation control, supporting control operations such as mapping, localization, navigation, and map management, encapsulating underlying details for upper-level system calls.

#### 13.1.1 `SlamNavController`

Item	Content
Function Name	<code>SlamNavController</code>
Function Declaration	<code>SlamNavController();</code>
Function Overview	Constructor, initializes SLAM navigation controller state
Remarks	Constructs internal control resources

### 13.1.2 ~SlamNavController

Item	Content
Function Name	<code>~SlamNavController</code>
Function Declaration	<code>~SlamNavController();</code>
Function Overview	Destructor, releases controller resources
Remarks	Used in conjunction with constructor

### 13.1.3 Initialize

Item	Content
Function Name	<code>Initialize</code>
Function Declaration	<code>bool Initialize();</code>
Function Overview	Initializes SLAM navigation controller, prepares SLAM and navigation functions
Return Value	<code>true</code> indicates success, <code>false</code> indicates failure
Remarks	Must be called before first use

### 13.1.4 Shutdown

Item	Content
Function Name	<code>Shutdown</code>
Function Declaration	<code>void Shutdown();</code>
Function Overview	Closes controller and releases resources
Remarks	Used in conjunction with Initialize, safely disconnects

### 13.1.5 ActivateSlamMode

Item	Content
Function Name	ActivateSlamMode
Function Declaration	Status ActivateSlamMode(SlamMode mode, std::string map_path = "", int timeout_ms);
Function Overview	Activates SLAM mode and starts mapping or localization mode
Parameter Description	mode: SLAM mode enumeration map_path: Map path (used in localization mode), can be obtained through GetMapPath timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, supports mapping and localization mode switching

### 13.1.6 StartMapping

Item	Content
Function Name	StartMapping
Function Declaration	Status StartMapping(int timeout_ms);
Function Overview	Starts mapping
Parameter Description	timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, mapping mode must be activated first

### 13.1.7 CancelMapping

Item	Content
Function Name	CancelMapping
Function Declaration	Status CancelMapping(int timeout_ms);
Function Overview	Cancels mapping
Parameter Description	timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, cancels current mapping task

### 13.1.8 SaveMap

Item	Content
Function Name	SaveMap
Function Declaration	Status SaveMap(const std::string& map_name, int timeout_ms);
Function Overview	Ends mapping and saves the map
Parameter Description	map_name: Map name timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, must be called in mapping mode

### 13.1.9 LoadMap

Item	Content
Function Name	LoadMap
Function Declaration	Status LoadMap(const std::string& map_name, int timeout_ms);
Function Overview	Loads map and sets it as current map
Parameter Description	map_name: Map name timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, loads map with specified name

### 13.1.10 DeleteMap

Item	Content
Function Name	DeleteMap
Function Declaration	Status DeleteMap(const std::string& map_name, int timeout_ms);
Function Overview	Deletes map
Parameter Description	map_name: Name of map to delete timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, permanently deletes specified map

### 13.1.11 GetMapPath

Item	Content
Function Name	GetMapPath
Function Declaration	Status GetMapPath(const std::string& map_name, std::vector<std::string>& map_path, int timeout_ms);
Function Overview	Gets map path
Parameter Description	map_name: Map name map_path: Map path (output parameter) timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, gets storage path of specified map

### 13.1.12 GetAllMapInfo

Item	Content
Function Name	GetAllMapInfo
Function Declaration	Status GetAllMapInfo(AllMapInfo& all_map_info, int timeout_ms);
Function Overview	Gets all map information
Parameter Description	all_map_info: All map information (output parameter) timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, gets detailed information of all maps in the system

### 13.1.13 InitPose

Item	Content
Function Name	InitPose
Function Declaration	Status InitPose(const Pose3DEuler& pose, int timeout_ms);
Function Overview	Initializes pose
Parameter Description	pose: Pose information to publish timeout_ms: Timeout time (milliseconds), default value is 15000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, sets robot initial pose; when publishing pose, a fixed -1.57rad offset needs to be applied to the Yaw direction because there is a -1.57rad offset between the mechanical installation of the LiDAR and the robot body

### 13.1.14 GetCurrentLocalizationInfo

Item	Content
Function Name	GetCurrentLocalizationInfo
Function Declaration	Status GetCurrentLocalizationInfo(LocalizationInfo& pose_info, int timeout_ms);
Function Overview	Gets current pose information
Parameter Description	pose_info: Current position and orientation information (output parameter) timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, gets robot current pose status

### 13.1.15 ActivateNavMode

Item	Content
Function Name	ActivateNavMode
Function Declaration	Status ActivateNavMode (NavMode mode, std::string map_path = "", int timeout_ms);
Function Overview	Activates navigation mode
Parameter Description	mode: Target navigation mode map_path: Map path (used in grid map mode), can be obtained through GetMapPath timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, activates specified navigation mode

### 13.1.16 SetNavTarget

Item	Content
Function Name	SetNavTarget
Function Declaration	Status SetNavTarget (const NavTarget& goal, int timeout_ms);
Function Overview	Sets global navigation target point and starts navigation task
Parameter Description	goal: Global coordinates of target point timeout_ms: Timeout time (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, sets navigation target and starts navigation

### 13.1.17 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	Non-blocking interface, pauses navigation task

### 13.1.18 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	Non-blocking interface, resumes paused navigation task

### 13.1.19 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	Non-blocking interface, cancels current navigation task

### 13.1.20 GetNavStatus

Item	Content
Function Name	GetNavStatus
Function Declaration	Status GetNavStatus(NavStatus& nav_status);
Function Overview	Gets navigation status
Parameter Description	nav_status: Navigation status (output parameter)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Non-blocking interface, gets current navigation status information

### 13.1.21 OpenOdometryStream

Item	Content
Function Name	OpenOdometryStream
Function Declaration	Status OpenOdometryStream();
Function Overview	Open odometry stream
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, check return value to ensure successful execution

### 13.1.22 CloseOdometryStream

Item	Content
Function Name	CloseOdometryStream
Function Declaration	Status CloseOdometryStream();
Function Overview	Close odometry stream
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, used in conjunction with OpenOdometryStream

### 13.1.23 SubscribeOdometry

Item	Content
Function Name	SubscribeOdometry
Function Declaration	void SubscribeOdometry(const OdometryCallback callback);
Function Overview	Subscribe to odometry data
Parameter Description	callback: Processing callback function after receiving odometry data
Remarks	Non-blocking interface, callback function will be called when odometry data is updated

### 13.1.24 UnsubscribeOdometry

Item	Content
Function Name	UnsubscribeOdometry
Function Declaration	void UnsubscribeOdometry();
Function Overview	Unsubscribe from odometry data
Remarks	Non-blocking interface. Used in conjunction with SubscribeOdometry

### 13.1.25 GetPointCloudMap

Item	Content
Function Name	GetPointCloudMap
Function Declaration	Status GetPointCloudMap(PointCloud2& point_cloud_map, int timeout_ms);
Function Overview	Get point cloud map data
Parameter Description	point_cloud_map: Point cloud map data (output parameter) timeout_ms: Timeout duration (milliseconds)
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, gets current point cloud map data

## 13.2 Type Definitions

### 13.2.1 Pose3DEuler —3D Pose Structure

Field Name	Type	Description
position	std::array<double, 3>;	Position coordinates (x, y, z)
orientation	std::array<double, 3>;	Euler angles (roll, pitch, yaw)

### 13.2.2 NavTarget —Navigation Target Point Structure

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

### 13.2.3 LocalizationInfo —Pose Information Structure

Field Name	Type	Description
is_localization	bool	Whether localized
pose	Pose3DEuler	Current pose

### 13.2.4 NavStatus —Navigation Status Structure

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

### 13.2.5 PolyRegion —Polygon Region Structure

Field Name	Type	Description
points	std::vector<Point2D>;	Polygon vertex list

### 13.2.6 Point2D —2D Point Structure

Field Name	Type	Description
x	double	X coordinate
y	double	Y coordinate

## 13.3 Enumeration Type Definitions

### 13.3.1 SlamMode —SLAM Mode Enumeration

Enum Value	Value	Description
IDLE	0	Idle mode
LOCALIZATION	1	Localization mode
MAPPING	2	Mapping mode

### 13.3.2 NavMode —Navigation Mode Enumeration

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

### 13.3.3 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	Canceled

## 13.4 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.

### 13.4.1 Applicable Scenarios and Scope

- Static indoor flat areas smaller than 100 m × 100 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.

### 13.4.2 SLAM Function Flow

Function	Operation Flow
Mapping	Activate mapping mode → Start mapping → Save map
Map Management	Load map, get map information, delete map, get map path, etc.

### 13.4.3 Navigation Function Flow

Function	Operation Flow
Localization	Load map → Activate localization mode → Initialize pose → Get localization status
Navigation	Localization success → Activate navigation mode → Set target pose → Get navigation status
Navigation Control	Set target pose → PauseNavTask navigation → ResumeNavTask navigation → CancelNavTask navigation

### 13.4.4 Localization Mode Initialization

When SLAM is in localization mode, an initial pose can be specified to quickly complete relocalization and initialization. For large-scale maps, specifying the initial pose is mandatory.

#### Small Map Mode

满足如下条件：

The following conditions must be met:

- Condition 1: The map should not be too large. For example, in a single room, it is best to keep the map area within 10m × 10m.

- Condition 2: Modify the configuration to adapt to this mode:

```
# Log in to the robot
ssh eame@192.168.54.119

# Modify the following configuration
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
reloc_options: true

# After modifying the YAML file for the first time, restart the SLAM module for the
→ change to take effect:
sudo systemctl restart slam.service
```

### Large Map Mode

The following conditions must be met:

- Condition 1: The robot must be located at the mapping origin (the starting position when mapping began), and the robot's orientation must remain consistent.
- Condition 2: Modify the configuration to adapt to this mode:

When using the SDK for SLAM navigation (unlike the app, which allows convenient visual positioning), you must modify the SLAM configuration to fix the map orientation and facilitate localization:

```
# Log in to the robot
ssh eame@192.168.54.119

# Modify the following configuration
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# After modifying the YAML file for the first time, restart the SLAM module for the
→ change to take effect:
sudo systemctl restart slam.service
```

- Condition 3: When initializing the pose, apply a -90° (-1.57 rad) offset to the yaw angle:

because the LiDAR's mechanical installation differs from the robot body orientation by -90°.

### 13.4.5 Map Noise Removal and Obstacle Addition

- Software Name: GIMP
- Download Link: <https://www.gimp.org/downloads/>
- Operating System: Ubuntu

- File to Edit: /home/eame/cust\_para/maps/\${map\_name}/map/map.pgm Log in to the robot (ssh eame@192.168.54.119), open the .pgm map file with GIMP for editing, and replace the original file after modifications are complete.
- Editing Tutorial Video:

# CHAPTER 14

---

## Robot Main Control Service (Python)

---

---

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

### 14.1 Interface Definition

`MagicRobot` is the unified entry class for the robot SDK.

#### 14.1.1 MagicRobot

Item	Content
Class Name	<code>MagicRobot</code>
Constructor	<code>robot = MagicRobot()</code>
Function Overview	Constructor, creates <code>MagicRobot</code> instance.
Notes	Constructs internal state.

### 14.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize(str local_ip)
Function Overview	Initialize 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.

### 14.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Shutdown robot system and release resources.
Notes	Used in conjunction with initialize.

### 14.1.4 connect

Item	Content
Function Name	connect
Function Declaration	Status connect()
Function Overview	Establish communication connection with robot service.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, must be called after initialization.

### 14.1.5 disconnect

Item	Content
Function Name	disconnect
Function Declaration	Status disconnect()
Function Overview	Disconnect from robot service.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, used in conjunction with connect.

#### 14.1.6 `get_sdk_version`

Item	Content
Function Name	<code>get_sdk_version</code>
Function Declaration	<code>str get_sdk_version()</code>
Function Overview	Get current SDK version number.
Return Value	SDK version string (e.g. <code>0.0.1</code> ).
Notes	Non-blocking interface, convenient for debugging or log marking.

#### 14.1.7 `get_motion_control_level`

Item	Content
Function Name	<code>get_motion_control_level</code>
Function Declaration	<code>ControllerLevel get_motion_control_level()</code>
Function Overview	Get current motion control level (high-level/low-level).
Return Value	<code>ControllerLevel</code> enumeration value.
Notes	Non-blocking interface, used to determine current control permissions.

#### 14.1.8 `set_motion_control_level`

Item	Content
Function Name	<code>set_motion_control_level</code>
Function Declaration	<code>Status set_motion_control_level(ControllerLevel level)</code>
Function Overview	Set current motion control level.
Parameter Description	<code>level</code> : Control permission enumeration value.
Return Value	<code>Status</code> object, <code>Status.code == ErrorCode.OK</code> indicates successful setting.
Notes	Blocking interface, used when switching control modes.

#### 14.1.9 `get_high_level_motion_controller`

Item	Content
Function Name	<code>get_high_level_motion_controller</code>
Function Declaration	<code>HighLevelMotionController get_high_level_motion_controller()</code>
Function Overview	Get high-level motion controller object.
Return Value	<code>HighLevelMotionController</code> object for calling high-level control interfaces.
Notes	Non-blocking interface, encapsulates gait, tricks, remote control, etc.

#### 14.1.10 get\_low\_level\_motion\_controller

Item	Content
Function Name	get_low_level_motion_controller
Function Declaration	LowLevelMotionController get_low_level_motion_controller()
Function Overview	Get low-level motion controller object.
Return Value	LowLevelMotionController object for controlling joints or motors.
Notes	Non-blocking interface, directly controls joint motors, gets IMU data, etc.

#### 14.1.11 get\_audio\_controller

Item	Content
Function Name	get_audio_controller
Function Declaration	AudioController get_audio_controller()
Function Overview	Get audio controller object.
Return Value	AudioController object for voice control.
Notes	Non-blocking interface, plays voice and controls volume.

#### 14.1.12 get\_sensor\_controller

Item	Content
Function Name	get_sensor_controller
Function Declaration	SensorController get_sensor_controller()
Function Overview	Get sensor controller object.
Return Value	SensorController object for accessing sensor data.
Notes	Non-blocking interface, encapsulates IMU, RGBD reading, etc.

#### 14.1.13 get\_state\_monitor

Item	Content
Function Name	get_state_monitor
Function Declaration	StateMonitor get_state_monitor()
Function Overview	Get state monitor object.
Return Value	StateMonitor object for getting current robot state information.
Notes	Non-blocking interface, encapsulates BMS, fault status information reading, etc.

#### 14.1.14 get\_slam\_nav\_controller

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



# CHAPTER 15

---

## High-Level Motion Control Service (Python)

---

---

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

### 15.1 Interface Definition

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

#### 15.1.1 HighLevelMotionController

Item	Content
Class Name	<code>HighLevelMotionController</code>
Constructor	<code>controller = HighLevelMotionController()</code>
Function Overview	Constructor, initializes high-level controller state
Notes	Constructs internal control resources

### 15.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize()
Function Overview	Initialize controller, prepare high-level control functions
Return Value	True indicates success, False indicates failure
Notes	Must be called before first use

### 15.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Close controller and release resources
Notes	Used in conjunction with initialize, safely disconnect

### 15.1.4 set\_gait

Item	Content
Function Name	set_gait
Function Declaration	Status set_gait(GaitMode gait_mode, int timeout_ms)
Function Overview	Set robot gait mode (such as recovery stand, balance stand, humanoid walking, etc.)
Parameter Description	gait_mode: Gait control enumeration
Return Value	Status object, Status.code == ErrorCode.OK indicates success
Notes	Blocking interface, can switch between multiple gait modes

### 15.1.5 get\_gait

Item	Content
Function Name	get_gait
Function Declaration	tuple[Status, GaitMode] get_gait()
Function Overview	Get robot gait mode (such as recovery stand, balance stand, humanoid walking, etc.)
Return Value	Status object containing current gait mode information
Notes	Blocking interface, gets current gait mode

### 15.1.6 execute\_trick

Item	Content
Function	execute_trick
Name	
Function Declaration	Status execute_trick(TrickAction trick_action, int timeout_ms)
Function Overview	Execute trick actions (such as celebrate, wave, etc.)
Parameter Description	trick_action: Trick action identifier
Return Value	Status object, Status.code == ErrorCode.OK indicates success
Notes	Blocking interface, ensure robot can currently execute tricks Note: Trick actions must be performed under GaitMode.GAIT_BALANCE_STAND(46) gait

### 15.1.7 send\_joystick\_command

Item	Content
Function Name	send_joystick_command
Function Declaration	Status send_joystick_command(JoystickCommand joy_command)
Function Overview	Send real-time joystick control commands
Parameter Description	joy_command: Control data containing joystick coordinates
Return Value	Status object, Status.code == ErrorCode.OK indicates success
Notes	Non-blocking interface, recommended sending frequency is 20Hz

### 15.1.8 head\_move

Item	Content
Function Name	head_move
Function Declaration	Status head_move(float shake_angle, float nod_angle, int timeout_ms);
Function Overview	Control robot head shaking and nodding motion
Parameter Description	shake_angle: Head left-right shake angle, direction: negative left/positive right, unit: rad, range: [-0.5236, 0.5236] nod_angle: Head nodding angle, direction: positive up/negative down, unit: rad, range: [-0.3491, 0.3491] timeout_ms: Timeout duration, default 5000 milliseconds
Return Value	Status::OK indicates success, others indicate failure
Notes	Blocking interface, controls robot head left-right shaking and up-down nodding motion

## 15.2 Type Definitions

### 15.2.1 JoystickCommand —High-Level Motion Control Joystick Command Structure

Field Name	Type	Description
x	float	Left joystick X-axis direction value (-1.0: left, 1.0: right)
y	float	Left joystick Y-axis direction value (-1.0: down, 1.0: up)
yaw	float	Right joystick X-axis direction value (rotation -1.0: left, 1.0: right)
z	float	Right joystick Y-axis direction value (purpose not yet defined)

## 15.3 Enumeration Type Definitions

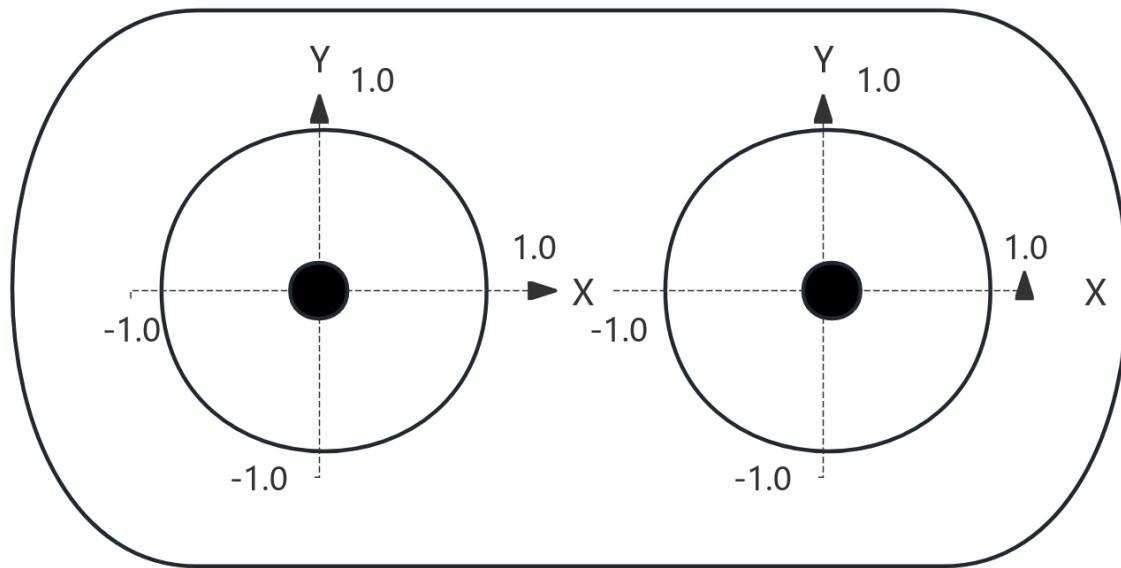
### 15.3.1 GaitMode —Robot Gait Mode Enumeration

Enumeration Value	Value	Description
GAIT_PASSIVE	0	Passive
GAIT_RECOVERY_STAND	1	Recovery Stand
GAIT_BALANCE_STAND	46	Balance Stand (supports movement)
GAIT_ARM_SWING_WALK	47	Arm Swing Walking
GAIT_LOWLEVL_SDK	200	Low-level Control SDK Mode

### 15.3.2 TrickAction —Trick Action Command Enumeration

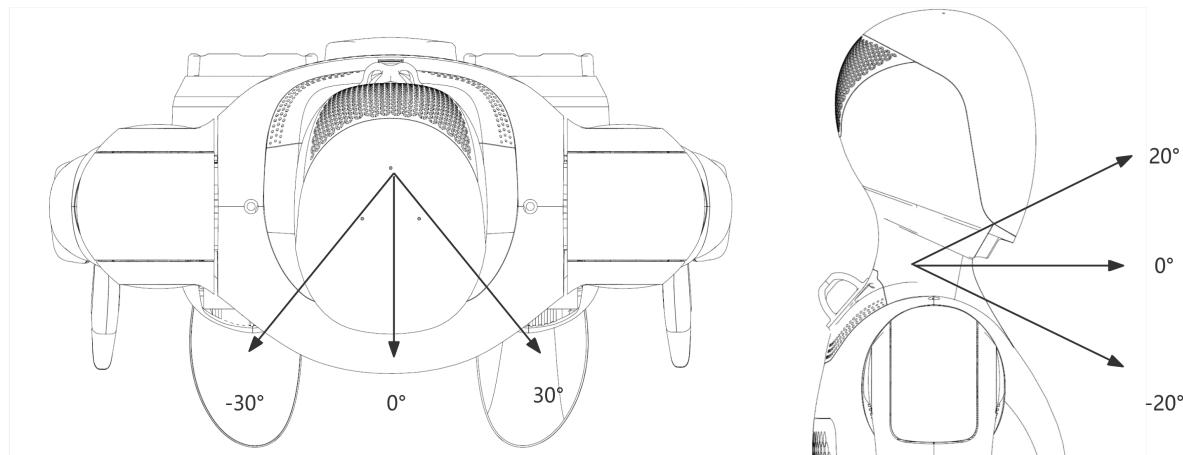
Enumeration Value	Value	Description
ACTION_RECOVERY_STAND	103	Balance Stand
ACTION_CELEBRATE	201	Celebrate
ACTION_SHAKE_HAND_REACHOUT	217	Shake Hand - Reach Out
ACTION_SHAKE_HAND_WITHDRAW	218	Shake Hand - Withdraw
ACTION_NOD_HEAD	219	Nod Head
ACTION_SHAKE_HEAD	220	Shake Head
ACTION_CIRCLE_HEAD	221	Circle Shake Head
ACTION_GREETING	301	Greeting
ACTION_POINT_GROUND	302	Point Ground - Reach Out
ACTION_POINT_GROUND_WITH_DRAW	303	Point Ground - Withdraw
ACTION_SPREAD_HAND	304	Spread Hand
ACTION_SPREAD_HAND_WITH_DRAW	305	Spread Hand - Withdraw
ACTION_TRUN_AWAY_LEFT_INTRODUCE	306	Turn Away Back
ACTION_TRUN_BACK_LEFT_INTRODUCE	307	Turn Back Forward

## 15.4 Joystick Diagram



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

## 15.5 Head Movement Range Diagram



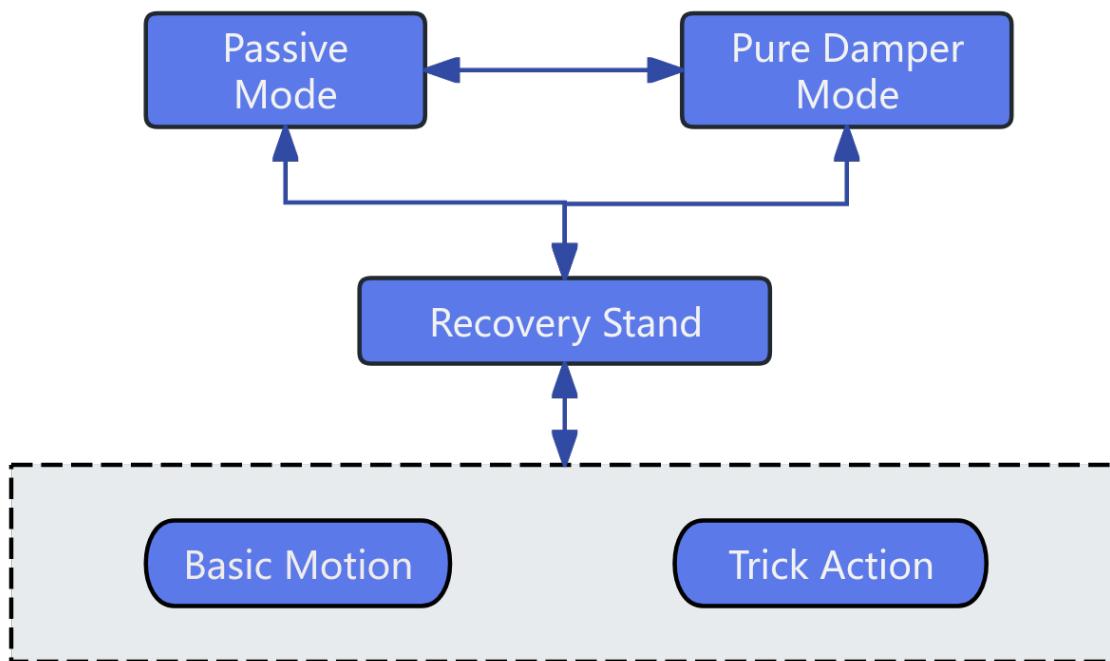
- Head shake range [-30°, 30°] ( $[-0.5236\text{rad}, 0.5236\text{rad}]$ )
- Head nod range [-20°, 20°] ( $[-0.3491\text{rad}, 0.3491\text{rad}]$ )

## 15.6 High-Level Motion Control Robot State Introduction

Robot motion includes recovery stand, balance stand, basic motion, and trick action states. During operation, the robot switches between different states through a state machine to achieve different control tasks. The explanation of each state is as follows:

- **Recovery Stand:** Recovery stand is the state connecting robot suspension landing and balance stand. The robot needs to enter balance stand state before calling corresponding motion services to achieve robot control.
- **Balance Stand:** In balance stand state, you can call various SDK interfaces to achieve robot trick actions and basic motion control.
- **Basic Motion:** During motion execution, you can call SDK interfaces to make the robot enter different gaits.
- **Trick Actions:** When entering special action execution state, other motion control services will be suspended first, waiting for current action to complete and enter balance stand state before taking effect again.

Robot state switching mechanism;

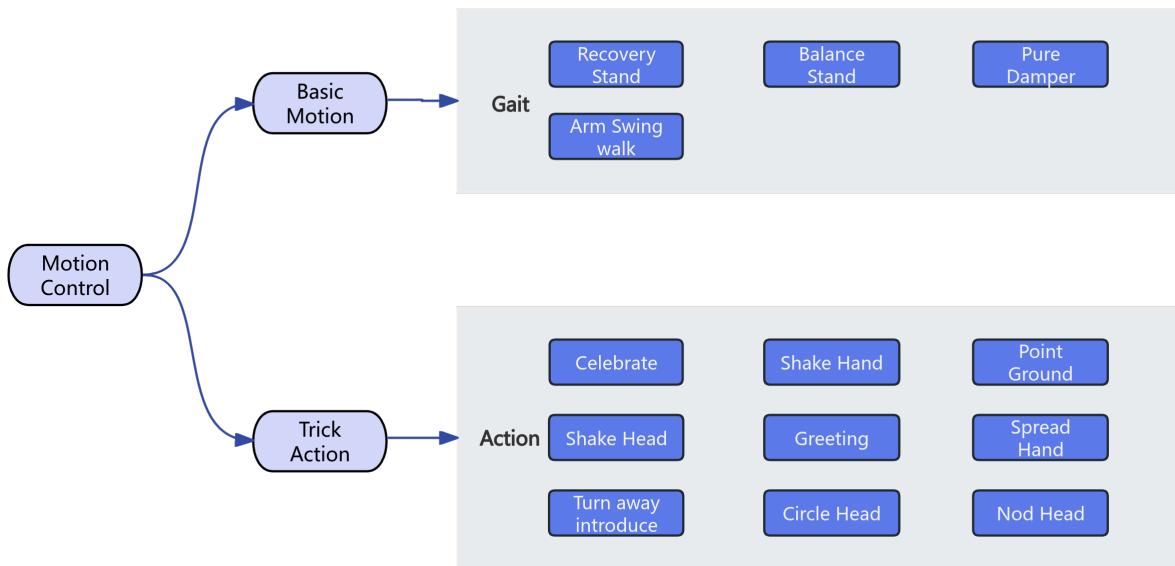


## 15.7 High-Level Motion Control Interface

Robot high-level motion control service can be divided into basic motion control and trick action control.

- In basic motion control service, you can call corresponding interfaces to switch robot walking gaits according to different terrain scenarios and task requirements.

- In trick action control service, you can call corresponding interfaces to achieve robot built-in special tricks, such as celebrate, shake hand, greeting, etc.



# CHAPTER 16

---

## Low-Level Motion Control Service (Python)

---

---

Provides robot system low-level motion control service. Through `LowLevelMotionController`, you can control robot joints and obtain status via topic communication.

### 16.1 Interface Definition

`LowLevelMotionController` is a motion controller for low-level development, supporting direct control and state subscription of motion components such as arms, legs, head, waist, hands, etc., as well as body IMU data acquisition.

#### 16.1.1 LowLevelMotionController

Item	Content
Class Name	<code>LowLevelMotionController</code>
Constructor	<code>controller = LowLevelMotionController()</code>
Function Overview	Constructor, initializes low-level controller object.
Notes	Constructs internal resources.

### 16.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize()
Function Overview	Initialize controller, establish low-level connection.
Return Value	True indicates success, False indicates failure.
Notes	Must be initialized before first call.

### 16.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Close controller, release low-level resources.
Notes	Used in conjunction with initialize.

### 16.1.4 subscribe\_arm\_state

Item	Content
Function Name	subscribe_arm_state
Function Declaration	void subscribe_arm_state(callback)
Function Overview	Subscribe to arm joint state data
Parameter Description	callback: Data processing function, signature is callback(data: JointState) -> None
Notes	Non-blocking interface.

### 16.1.5 unsubscribe\_arm\_state

Item	Content
Function Name	unsubscribe_arm_state
Function Declaration	void unsubscribe_arm_state()
Function Overview	Unsubscribe from arm joint state data
Notes	Non-blocking interface. Used in conjunction with subscribe_arm_state.

### 16.1.6 publish\_arm\_command

Item	Content
Function Name	publish_arm_command
Function Declaration	Status publish_arm_command(JointCommand command)
Function Overview	Publish arm control commands
Parameter Description	command: Target position/velocity etc.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Non-blocking interface.

### 16.1.7 subscribe\_leg\_state

Item	Content
Function Name	subscribe_leg_state
Function Declaration	void subscribe_leg_state(callback)
Function Overview	Subscribe to leg joint state data
Parameter Description	callback: Data processing function, signature is callback(data: JointState) -> None
Notes	Non-blocking interface.

### 16.1.8 unsubscribe\_leg\_state

Item	Content
Function Name	unsubscribe_leg_state
Function Declaration	void unsubscribe_leg_state()
Function Overview	Unsubscribe from leg joint state data
Notes	Non-blocking interface. Used in conjunction with subscribe_leg_state.

### 16.1.9 publish\_leg\_command

Item	Content
Function Name	publish_leg_command
Function Declaration	Status publish_leg_command(JointCommand command)
Function Overview	Publish leg control commands
Parameter Description	command: Target position/velocity etc.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Non-blocking interface, called when publishing or subscribing.

### 16.1.10 subscribe\_head\_state

Item	Content
Function Name	subscribe_head_state
Function Declaration	<code>void subscribe_head_state(callback)</code>
Function Overview	Subscribe to head joint state data
Parameter Description	callback: Data processing function, signature is <code>callback(data: JointState) -&gt; None</code>
Notes	Non-blocking interface.

### 16.1.11 unsubscribe\_head\_state

Item	Content
Function Name	unsubscribe_head_state
Function Declaration	<code>void unsubscribe_head_state()</code>
Function Overview	Unsubscribe from head joint state data
Notes	Non-blocking interface. Used in conjunction with <code>subscribe_head_state</code> .

### 16.1.12 publish\_head\_command

Item	Content
Function Name	publish_head_command
Function Declaration	<code>Status publish_head_command(JointCommand command)</code>
Function Overview	Publish head control commands
Parameter Description	command: Target position/velocity etc.
Return Value	<code>Status object, Status.code == ErrorCode.OK</code> indicates success.
Notes	Non-blocking interface.

### 16.1.13 subscribe\_waist\_state

Item	Content
Function Name	subscribe_waist_state
Function Declaration	<code>void subscribe_waist_state(callback)</code>
Function Overview	Subscribe to waist joint state data
Parameter Description	callback: Data processing function, signature is <code>callback(data: JointState) -&gt; None</code>
Notes	Non-blocking interface.

### 16.1.14 unsubscribe\_waist\_state

Item	Content
Function Name	unsubscribe_waist_state
Function Declaration	void unsubscribe_waist_state()
Function Overview	Unsubscribe from waist joint state data
Notes	Non-blocking interface. Used in conjunction with subscribe_waist_state.

### 16.1.15 publish\_waist\_command

Item	Content
Function Name	publish_waist_command
Function Declaration	Status publish_waist_command(JointCommand command)
Function Overview	Publish waist control commands
Parameter Description	command: Target position/velocity etc.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Non-blocking interface.

### 16.1.16 subscribe\_hand\_state

Item	Content
Function Name	subscribe_hand_state
Function Declaration	void subscribe_hand_state(callback)
Function Overview	Subscribe to hand state data
Parameter Description	callback: Data processing function, signature is callback(data: HandState) -> None
Notes	Non-blocking interface.

### 16.1.17 unsubscribe\_hand\_state

Item	Content
Function Name	unsubscribe_hand_state
Function Declaration	void unsubscribe_hand_state()
Function Overview	Unsubscribe from hand state data
Notes	Non-blocking interface. Used in conjunction with subscribe_hand_state.

### 16.1.18 publish\_hand\_command

Item	Content
Function Name	publish_hand_command
Function Declaration	Status publish_hand_command (HandCommand command)
Function Overview	Publish hand control commands
Parameter Description	command: Hand joint target position etc.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Non-blocking interface.

### 16.1.19 subscribe\_body\_imu

Item	Content
Function Name	subscribe_body_imu
Function Declaration	void subscribe_body_imu(callback)
Function Overview	Subscribe to body IMU data
Parameter Description	callback: IMU data processing function, signature is callback(data: Imu) -> None
Notes	Non-blocking interface.

### 16.1.20 unsubscribe\_body\_imu

Item	Content
Function Name	unsubscribe_body_imu
Function Declaration	void unsubscribe_body_imu()
Function Overview	Unsubscribe from body IMU data
Notes	Non-blocking interface. Used in conjunction with subscribe_body_imu.

## 16.2 Type Definitions

### 16.2.1 SingleHandJointCommand —Single Hand Joint Control Command

Field Name	Type	Description
operation_mode	int	Control word
pos	list[float]	Desired position array (7 DOF, unit: rad)

- Dexterous hand operation\_mode needs to switch from mode: 200 to mode: 4 to start, and then send commands;

- Dexterous hand joint command range: [0-1000]

### 16.2.2 HandCommand —Complete Hand Control Command

Field Name	Type	Description
timestamp	int	Timestamp (unit: nanoseconds)
cmd	list[SingleHandJointCommand]	Control command array, left hand and right hand in order

### 16.2.3 SingleHandJointState —Single Hand Joint State

Field Name	Type	Description
status_word	int	Status word
pos	list[float]	Current position (unit: rad)
toq	list[float]	Current torque (unit: Nm)
cur	list[float]	Current current (unit: A)
error_code	int	Error code

### 16.2.4 HandState —Complete Hand State Information

Field Name	Type	Description
timestamp	int	Timestamp (unit: nanoseconds)
state	list[SingleHandJointState]	All hand joint states (left hand, right hand order)

### 16.2.5 SingleJointCommand —Single Joint Control Command

Field Name	Type	Description
operation_mode	int	Control mode identifier: • 200 = Standby mode • 3 = Parallel triple-loop control (MIT mode) • 4 = Serial triple-loop control
pos	float	Target position (unit: rad)
vel	float	Target velocity (unit: rad/s)
toq	float	Target torque (unit: Nm)
kp	float	Position gain
kd	float	Velocity gain

- For joints 1–5 of the arm, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.

- For joints 6–7 of the arm, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–2 of the head, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–2 of the waist, `operation_mode` must switch from mode 200 (standby) to mode 4 (serial triple-loop control) before sending commands.
- For joints 1–6 of the legs, `operation_mode` must switch from mode 200 (standby) to mode 3 (parallel triple-loop control, MIT mode) before sending commands.

### 16.2.6 `JointCommand` —Joint Control Command

Field Name	Type	Description
<code>timestamp</code>	<code>int</code>	Timestamp (unit: nanoseconds)
<code>joints</code>	<code>list[SingleJointCommand]</code>	Joint control command array

### 16.2.7 `SingleJointState` —Single Joint State

Field Name	Type	Description
<code>operation_mode</code>	<code>int</code>	Current control mode
<code>pos</code>	<code>float</code>	Current position (unit: rad)
<code>vel</code>	<code>float</code>	Current velocity (unit: rad/s)
<code>toq</code>	<code>float</code>	Current torque (unit: Nm)
<code>cur</code>	<code>float</code>	Current current (unit: A)
<code>error_code</code>	<code>int</code>	Error code

### 16.2.8 `JointState` —Joint State

Field Name	Type	Description
<code>timestamp</code>	<code>int</code>	Timestamp (unit: nanoseconds)
<code>joints</code>	<code>list[SingleJointState]</code>	Joint state array

### 16.2.9 Joint Motor Order

## Head Joints

Index	Joint Name
0	head_yaw_joint
1	head_pitch_joint

## Arm Joints

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

## Waist Joints

Index	Joint Name
0	waist_roll_joint
1	waist_yaw_joint

####Leg Joints

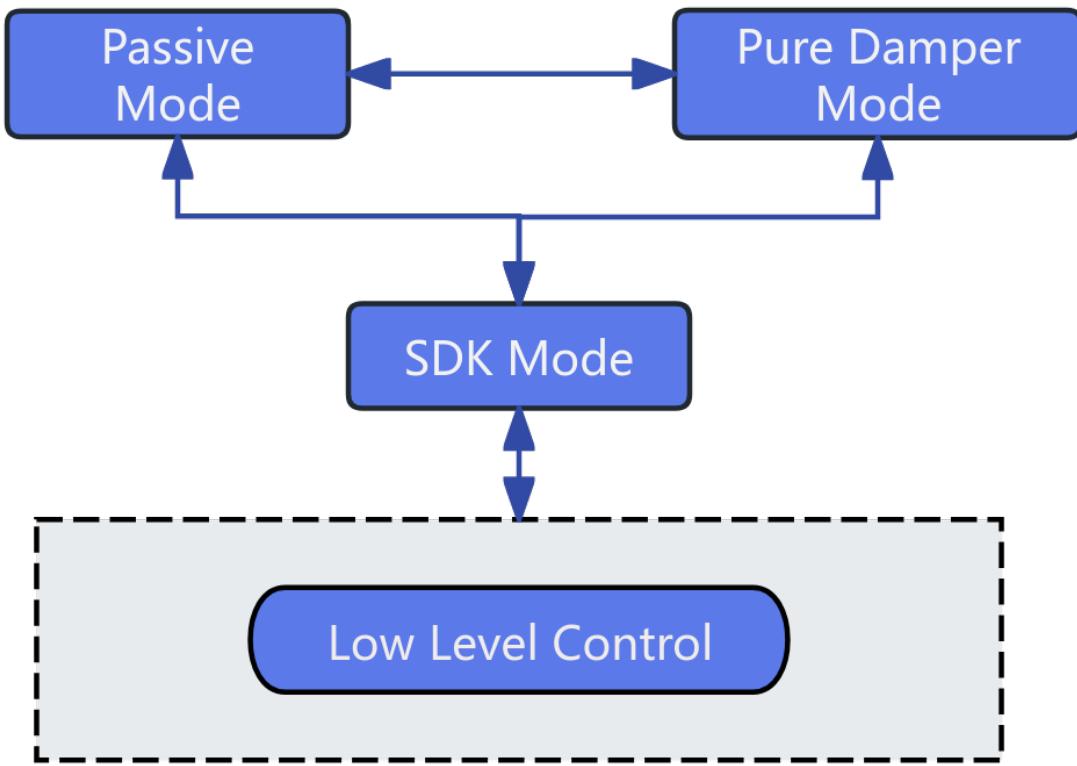
Index	Joint Name
0	left_hip_roll_joint
1	left_hip_yaw_joint
2	left_hip_pitch_joint
3	left_knee_pitch_joint
4	left_ankle_pitch_joint
5	left_ankle_roll_joint
6	right_hip_roll_joint
7	right_hip_yaw_joint
8	right_hip_pitch_joint
9	right_knee_pitch_joint
10	right_ankle_pitch_joint
11	right_ankle_roll_joint

## 16.3 URDF Reference

Robot URDF

## 16.4 Low-Level Motion Control Robot State Introduction

Robot low-level motion mainly develops three-loop control of joints for developers to perform secondary development of robot motion capabilities, basic control state switching mechanism:



## 16.5 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 17

---

## Sensor Control Service (Python)

---

---

Provides robot system sensor (LiDAR/RGBD camera/trinocular camera) service. Through SensorController, you can control robot sensors and obtain status via RPC and topic methods.

### 17.1 Interface Definition

`SensorController` is a management class that encapsulates various robot sensors, supporting initialization, control, and data subscription of LiDAR, RGBD cameras, and trinocular cameras.

**⚠️ Notice:** Current trinocular and RGBD sensor interfaces are not fully support yet, please do not use.

#### 17.1.1 SensorController

Item	Content
Class Name	<code>SensorController</code>
Constructor	<code>controller = SensorController()</code>
Function Overview	Constructor, initializes sensor controller object.
Notes	Constructs internal state.

### 17.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize()
Function Overview	Initialize controller, including resource allocation and driver loading.
Return Value	True indicates success, False indicates failure.
Notes	Object must be constructed before calling.

### 17.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Close all sensor connections and release resources.
Notes	Used in conjunction with initialize.

### 17.1.4 open\_lidar

Item	Content
Function Name	open_lidar
Function Declaration	Status open_lidar()
Function Overview	Open LiDAR.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface.

### 17.1.5 close\_lidar

Item	Content
Function Name	close_lidar
Function Declaration	Status close_lidar()
Function Overview	Close LiDAR.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, used in conjunction with open function.

### 17.1.6 open\_head\_rgbd\_camera

Item	Content
Function Name	open_head_rgbd_camera
Function Declaration	Status open_head_rgbd_camera()
Function Overview	Open head RGBD camera (head and waist).
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface.

### 17.1.7 close\_head\_rgbd\_camera

Item	Content
Function Name	close_head_rgbd_camera
Function Declaration	Status close_head_rgbd_camera()
Function Overview	Close head RGBD camera.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, used in conjunction with open function.

### 17.1.8 open\_waist\_rgbd\_camera

Item	Content
Function Name	open_waist_rgbd_camera
Function Declaration	Status open_waist_rgbd_camera()
Function Overview	Open waist RGBD camera (head and waist).
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface.

### 17.1.9 close\_waist\_rgbd\_camera

Item	Content
Function Name	close_waist_rgbd_camera
Function Declaration	Status close_waist_rgbd_camera()
Function Overview	Close waist RGBD camera.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, used in conjunction with open function.

### 17.1.10 open\_trinocular\_camera

Item	Content
Function Name	open_trinocular_camera
Function Declaration	Status open_trinocular_camera()
Function Overview	Open trinocular camera.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface.

### 17.1.11 close\_trinocular\_camera

Item	Content
Function Name	close_trinocular_camera
Function Declaration	Status close_trinocular_camera()
Function Overview	Close trinocular camera.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface.

### 17.1.12 subscribe\_lidar\_imu

Item	Content
Function Name	subscribe_lidar_imu
Function Declaration	void subscribe_lidar_imu(callback)
Function Overview	Subscribe to LiDAR IMU data
Parameter Description	callback: Data processing function after receiving data, signature is callback(data: Imu) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.13 unsubscribe\_lidar\_imu

Item	Content
Function Name	unsubscribe_lidar_imu
Function Declaration	void unsubscribe_lidar_imu()
Function Overview	Unsubscribe from LiDAR IMU data
Notes	Non-blocking interface. Used in conjunction with subscribe_lidar_imu.

### 17.1.14 subscribe\_lidar\_point\_cloud

Item	Content
Function Name	subscribe_lidar_point_cloud
Function Declaration	void subscribe_lidar_point_cloud(callback)
Function Overview	Subscribe to LiDAR point cloud data
Parameter Description	callback: Point cloud processing function after receiving data, signature is callback(data: PointCloud2) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.15 unsubscribe\_lidar\_point\_cloud

Item	Content
Function Name	unsubscribe_lidar_point_cloud
Function Declaration	void unsubscribe_lidar_point_cloud()
Function Overview	Unsubscribe from LiDAR point cloud data
Notes	Non-blocking interface. Used in conjunction with subscribe_lidar_point_cloud.

### 17.1.16 subscribe\_head\_rgbd\_color\_image

Item	Content
Function Name	subscribe_head_rgbd_color_image
Function Declaration	void subscribe_head_rgbd_color_image(callback)
Function Overview	Subscribe to head RGBD color image data
Parameter Description	callback: Image processing function after receiving data, signature is callback(data: Image) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.17 unsubscribe\_head\_rgbd\_color\_image

Item	Content
Function Name	unsubscribe_head_rgbd_color_image
Function Declaration	void unsubscribe_head_rgbd_color_image()
Function Overview	Unsubscribe from head RGBD color image data
Notes	Non-blocking interface. Used in conjunction with subscribe_head_rgbd_color_image.

### 17.1.18 subscribe\_head\_rgbd\_color\_camera\_info

Item	Content
Function Name	subscribe_head_rgbd_color_camera_info
Function Declaration	void subscribe_head_rgbd_color_camera_info(callback)
Function Overview	Subscribe to head RGBD color camera parameters
Parameter Description	callback: Parameter processing function after receiving data, signature is callback(data: CameraInfo) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.19 unsubscribe\_head\_rgbd\_color\_camera\_info

Item	Content
Function Name	unsubscribe_head_rgbd_color_camera_info
Function Declaration	void unsubscribe_head_rgbd_color_camera_info()
Function Overview	Unsubscribe from head RGBD color camera parameters
Notes	Non-blocking interface. Used in conjunction with subscribe_head_rgbd_color_camera_info.

### 17.1.20 subscribe\_head\_rgbd\_depth\_image

Item	Content
Function Name	subscribe_head_rgbd_depth_image
Function Declaration	void subscribe_head_rgbd_depth_image(callback)
Function Overview	Subscribe to head RGBD depth image data
Parameter Description	callback: Image processing function after receiving data, signature is callback(data: Image) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.21 unsubscribe\_head\_rgbd\_depth\_image

Item	Content
Function Name	unsubscribe_head_rgbd_depth_image
Function Declaration	void unsubscribe_head_rgbd_depth_image()
Function Overview	Unsubscribe from head RGBD depth image data
Notes	Non-blocking interface. Used in conjunction with subscribe_head_rgbd_depth_image.

### 17.1.22 subscribe\_head\_rgbd\_depth\_camera\_info

Item	Content
Function Name	subscribe_head_rgbd_depth_camera_info
Function Declaration	void subscribe_head_rgbd_depth_camera_info(callback)
Function Overview	Subscribe to head RGBD depth camera parameters
Parameter Description	callback: Parameter processing function after receiving data, signature is callback(data: CameraInfo) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.23 unsubscribe\_head\_rgbd\_depth\_camera\_info

Item	Content
Function Name	unsubscribe_head_rgbd_depth_camera_info
Function Declaration	void unsubscribe_head_rgbd_depth_camera_info()
Function Overview	Unsubscribe from head RGBD depth camera parameters
Notes	Non-blocking interface. Used in conjunction with subscribe_head_rgbd_depth_camera_info.

### 17.1.24 subscribe\_waist\_rgbd\_color\_image

Item	Content
Function Name	subscribe_waist_rgbd_color_image
Function Declaration	void subscribe_waist_rgbd_color_image(callback)
Function Overview	Subscribe to waist RGBD color image data
Parameter Description	callback: Image processing function after receiving data, signature is callback(data: Image) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.25 unsubscribe\_waist\_rgbd\_color\_image

Item	Content
Function Name	unsubscribe_waist_rgbd_color_image
Function Declaration	void unsubscribe_waist_rgbd_color_image()
Function Overview	Unsubscribe from waist RGBD color image data
Notes	Non-blocking interface. Used in conjunction with subscribe_waist_rgbd_color_image.

### 17.1.26 subscribe\_waist\_rgbd\_color\_camera\_info

Item	Content
Function Name	subscribe_waist_rgbd_color_camera_info
Function Declaration	void subscribe_waist_rgbd_color_camera_info(callback)
Function Overview	Subscribe to waist RGBD color camera parameters
Parameter Description	callback: Parameter processing function after receiving data, signature is callback(data: CameraInfo) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.27 unsubscribe\_waist\_rgbd\_color\_camera\_info

Item	Content
Function Name	unsubscribe_waist_rgbd_color_camera_info
Function Declaration	void unsubscribe_waist_rgbd_color_camera_info()
Function Overview	Unsubscribe from waist RGBD color camera parameters
Notes	Non-blocking interface. Used in conjunction with subscribe_waist_rgbd_color_camera_info.

### 17.1.28 subscribe\_waist\_rgbd\_depth\_image

Item	Content
Function Name	subscribe_waist_rgbd_depth_image
Function Declaration	void subscribe_waist_rgbd_depth_image(callback)
Function Overview	Subscribe to waist RGBD depth image data
Parameter Description	callback: Image processing function after receiving data, signature is callback(data: Image) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.29 unsubscribe\_waist\_rgbd\_depth\_image

Item	Content
Function Name	unsubscribe_waist_rgbd_depth_image
Function Declaration	void unsubscribe_waist_rgbd_depth_image()
Function Overview	Unsubscribe from waist RGBD depth image data
Notes	Non-blocking interface. Used in conjunction with subscribe_waist_rgbd_depth_image.

### 17.1.30 subscribe\_waist\_rgbd\_depth\_camera\_info

Item	Content
Function Name	subscribe_waist_rgbd_depth_camera_info
Function Declaration	void subscribe_waist_rgbd_depth_camera_info(callback)
Function Overview	Subscribe to waist RGBD depth camera parameters
Parameter Description	callback: Parameter processing function after receiving data, signature is callback(data: CameraInfo) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.31 unsubscribe\_waist\_rgbd\_depth\_camera\_info

Item	Content
Function Name	unsubscribe_waist_rgbd_depth_camera_info
Function Declaration	void unsubscribe_waist_rgbd_depth_camera_info()
Function Overview	Unsubscribe from waist RGBD depth camera parameters
Notes	Non-blocking interface. Used in conjunction with subscribe_waist_rgbd_depth_camera_info.

### 17.1.32 subscribe\_trinocular\_image

Item	Content
Function Name	subscribe_trinocular_image
Function Declaration	void subscribe_trinocular_image(callback)
Function Overview	Subscribe to trinocular camera image frames
Parameter Description	callback: Image frame processing function after receiving data, signature is callback(data: TrinocularCameraFrame) -> None
Notes	Non-blocking interface, callback function will be called when data is updated.

### 17.1.33 unsubscribe\_trinocular\_image

Item	Content
Function Name	unsubscribe_trinocular_image
Function Declaration	void unsubscribe_trinocular_image()
Function Overview	Unsubscribe from trinocular camera image frames
Notes	Non-blocking interface. Used in conjunction with subscribe_trinocular_image.

## 17.2 Type Definitions

### 17.2.1 Imu —IMU Data Structure

Field Name	Type	Description
timestamp	int64	Timestamp (unit: nanoseconds)
orientation	list[float]	Attitude quaternion (w, x, y, z)
angular_velocity	list[float]	Angular velocity (unit: rad/s)
linear_acceleration	list[float]	Linear acceleration (unit: m/s <sup>2</sup> )
temperature	float	Temperature (unit: Celsius or other, unit should be specified)

### 17.2.2 Header —Common Message Header Structure

Field Name	Type	Description
stamp	int64	Timestamp (unit: nanoseconds)
frame_id	str	Coordinate frame name

### 17.2.3 PointField —Point Field Description

Field Name	Type	Description
name	str	Field name (such as x, y, z, intensity)
offset	int	Starting byte offset
datatype	int8	Data type (corresponding constant)
count	int	Number of elements this field contains per point

#### 17.2.4 PointCloud2 —Point Cloud Data Structure

Field Name	Type	Description
header	Header	Message header
height	int	Number of rows
width	int	Number of columns
fields	list[PointField]	Point field array
is_bigendian	bool	Byte order
point_step	int	Number of bytes per point
row_step	int	Number of bytes per row
data	bytes	Raw point cloud byte data
is_dense	bool	Whether dense point cloud

#### 17.2.5 Image —Image Data Structure

Field Name	Type	Description
header	Header	Message header
height	int	Image height (pixels)
width	int	Image width (pixels)
encoding	str	Encoding type (such as rgb8, mono8)
is_bigendian	bool	Whether big-endian mode
step	int	Number of bytes per image row
data	bytes	Raw image byte data

## 17.2.6 CameraInfo —Camera Intrinsic Parameters and Distortion Information

Field Name	Type	Description
header	Header	Message header
height	int	Image height
width	int	Image width
distortion_model	str	Distortion model (such as plumb_bob)
D	list[float]	Distortion parameter array
K	list[float]	Camera intrinsic parameter matrix (3x3)
R	list[float]	Rectification matrix (3x3)
P	list[float]	Projection matrix (3x4)
binning_x	int	Horizontal binning coefficient
binning_y	int	Vertical binning coefficient
roi_x_offset	int	ROI starting x coordinate
roi_y_offset	int	ROI starting y coordinate
roi_height	int	ROI height
roi_width	int	ROI width
roi_do_rectify	bool	Whether to perform rectification

## 17.2.7 TrinocularCameraFrame —Trinocular Camera Frame Data Structure

Field Name	Type	Description
header	Header	Common message header
vin_time	int64	Image acquisition time (nanoseconds)
decode_time	int64	Image decoding time (nanoseconds)
imgfl_array	bytes	Left eye image data
imgf_array	bytes	Center eye image data
imgfr_array	bytes	Right eye image data

## 17.3 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 18

---

## Audio Control Service (Python)

---

---

Provides robot system audio service controller. Through `AudioController`, you can control robot audio and obtain status via RPC communication.

### 18.1 Interface Definition

`AudioController` is a Python class that encapsulates audio control functionality, mainly used for audio playback control, TTS playback, volume setting and query, etc.

#### 18.1.1 `AudioController`

Item	Content
Class Name	<code>AudioController</code>
Constructor	<code>controller = AudioController()</code>
Function Overview	Initialize audio controller object, construct internal state, allocate resources, etc.
Notes	Constructs internal state.

### 18.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize()
Function Overview	Initialize audio control module, prepare playback resources and devices.
Return Value	True indicates success, False indicates failure.
Notes	Used in conjunction with shutdown().

### 18.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Close audio controller and release resources.
Notes	Ensure called before destruction.

### 18.1.4 play

Item	Content
Function Name	play
Function Declaration	Status play(TtsCommand cmd, int timeout_ms)
Function Overview	Play TTS (Text-to-Speech) voice command.
Parameter Description	cmd: TTS command, containing text, speech rate, tone, etc.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, ensure module is initialized before calling.

### 18.1.5 stop

Item	Content
Function Name	stop
Function Declaration	Status stop()
Function Overview	Stop current audio playback.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, typically used to interrupt current speech.

### 18.1.6 set\_volume

Item	Content
Function Name	set_volume
Function Declaration	Status set_volume(int volume)
Function Overview	Set audio output volume.
Parameter Description	volume: Volume value, typically range 0~100.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, takes effect immediately after setting.

### 18.1.7 get\_volume

Item	Content
Function Name	get_volume
Function Declaration	tuple[Status, int] get_volume()
Function Overview	Get current audio output volume.
Return Value	Returns tuple (Status, volume), Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, check return value before using volume.

### 18.1.8 open\_audio\_stream

Item	Content
Function Name	open_audio_stream
Function Declaration	Status open_audio_stream()
Function Overview	Open audio stream, prepare for audio playback.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, check return value to ensure successful execution

### 18.1.9 close\_audio\_stream

Item	Content
Function Name	close_audio_stream
Function Declaration	Status close_audio_stream()
Function Overview	Close audio stream.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, check return value to ensure successful execution

### 18.1.10 subscribe\_origin\_audio\_stream

Item	Content
Function Name	subscribe_origin_audio_stream
Function Declaration	void subscribe_origin_audio_stream(callback)
Function Overview	Subscribe to original audio stream data.
Parameter Description	callback: Data processing function after receiving data, signature is callback(data: AudioStream) -> None
Notes	Non-blocking interface.

### 18.1.11 unsubscribe\_origin\_audio\_stream

Item	Content
Function Name	unsubscribe_origin_audio_stream
Function Declaration	void unsubscribe_origin_audio_stream()
Function Overview	Unsubscribe from original audio stream data.
Notes	Non-blocking interface. Used in conjunction with subscribe_origin_audio_stream.

### 18.1.12 subscribe\_bf\_audio\_stream

Item	Content
Function Name	subscribe_bf_audio_stream
Function Declaration	void subscribe_bf_audio_stream(callback)
Function Overview	Subscribe to BF audio stream data.
Parameter Description	callback: Data processing function after receiving data, signature is callback(data: AudioStream) -> None
Notes	Non-blocking interface.

### 18.1.13 unsubscribe\_bf\_audio\_stream

Item	Content
Function Name	unsubscribe_bf_audio_stream
Function Declaration	void unsubscribe_bf_audio_stream()
Function Overview	Unsubscribe from BF audio stream data.
Notes	Non-blocking interface. Used in conjunction with subscribe_bf_audio_stream.

### 18.1.14 open\_wakeup\_status\_stream

Item	Content
Function Name	open_wakeup_status_stream
Function Declaration	Status open_wakeup_status_stream()
Function Overview	Enable voice wake-up status stream.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, check return value to ensure successful execution.

### 18.1.15 close\_wakeup\_status\_stream

Item	Content
Function Name	close_wakeup_status_stream
Function Declaration	Status close_wakeup_status_stream()
Function Overview	Disable voice wake-up status stream.
Return Value	Status object, Status.code == ErrorCode.OK indicates success.
Notes	Blocking interface, used in conjunction with open_wakeup_status_stream.

### 18.1.16 subscribe\_wakeup\_status

Item	Content
Function Name	subscribe_wakeup_status
Function Declaration	void subscribe_wakeup_status(callback)
Function Overview	Subscribe to voice wake-up status data.
Parameter Description	callback: Processing function after receiving wake-up status, signature is callback(data: WakeupStatus) -> None
Notes	Non-blocking interface, callback function will be called when wake-up status is updated.

### 18.1.17 unsubscribe\_wakeup\_status

Item	Content
Function Name	unsubscribe_wakeup_status
Function Declaration	void unsubscribe_wakeup_status()
Function Overview	Unsubscribe from voice wake-up status data.
Notes	Non-blocking interface. Used in conjunction with subscribe_wakeup_status.

## 18.2 Type Definitions

### 18.2.1 `TtsPriority` —TTS Playback Priority Level

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

Enumeration Value	Description
<code>TtsPriority.HIGH</code>	Highest priority, e.g.: low battery warning, emergency alerts
<code>TtsPriority.MIDDLE</code>	Medium priority, e.g.: system prompts, status announcements
<code>TtsPriority.LOW</code>	Lowest priority, e.g.: daily voice conversation, background announcements

### 18.2.2 `TtsMode` —Task Scheduling Strategy Under Same Priority

Used to fine-tune control of playback order and clearing logic for multiple TTS tasks under the same priority condition.

Enumeration Value	Description
<code>TtsMode.CLEARTOP</code>	Clear all tasks of current priority (including currently playing and waiting queue), immediately play this request
<code>TtsMode.ADD</code>	Append this request to the end of current priority queue, play in order (without interrupting current playback)
<code>TtsMode.CLEARBUFFER</code>	Clear unplayed requests in queue, keep current playback, then play this request

## 18.3 Structure Definitions

### 18.3.1 `TtsCommand` —TTS Playback Command Structure

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

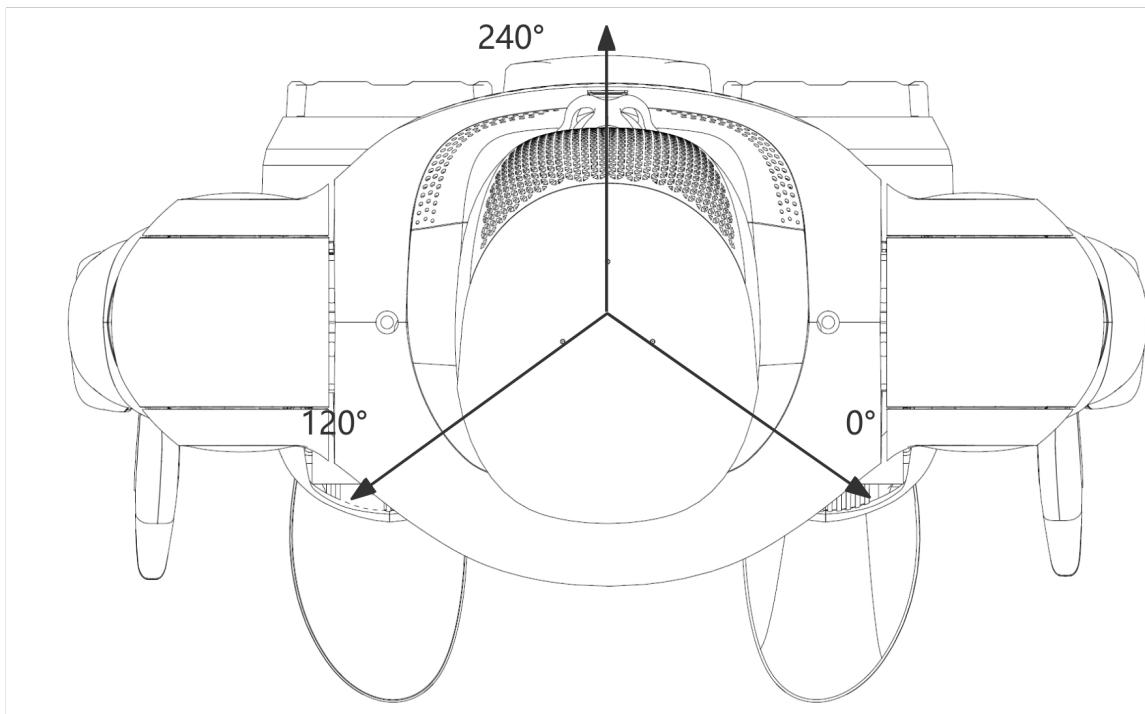
Field Name	Type	Description
id	str	TTS task unique ID, e.g. "id_01", used to track playback status
content	str	Text content to be played, e.g. "Hello, welcome to the intelligent voice system."
priority	TtsPriority	Playback priority, controls whether to interrupt currently playing lower priority speech
mode	TtsMode	Scheduling strategy under same priority, controls whether tasks are appended, overridden, etc.

### 18.3.2 AudioStream - Audio Stream Structure

Describes original audio stream or BF audio stream data.

Field Name	Type	Description
data_length	int	Data length
raw_data	bytes	Audio stream data

## 18.4 DOA



## 18.5 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 19

---

## State Monitor Service (Python)

---

---

Provides robot state monitoring interface class. Through StateMonitor, you can provide state query and other interfaces.

### 19.1 Interface Definition

`StateMonitor` is a robot system state monitoring management class, typically used to control robot state management and support state queries.

#### 19.1.1 `StateMonitor`

Item	Content
Class Name	<code>StateMonitor</code>
Constructor	<code>controller = StateMonitor()</code>
Function Overview	Constructor, initializes controller object.
Notes	Constructs internal state.

### 19.1.2 initialize

Item	Content
Function Name	initialize
Function Declaration	bool initialize()
Function Overview	Initialize controller, including resource allocation and driver loading.
Return Value	True indicates success, False indicates failure.
Notes	Object must be constructed before calling.

### 19.1.3 shutdown

Item	Content
Function Name	shutdown
Function Declaration	void shutdown()
Function Overview	Close all connections and release resources.
Notes	Used in conjunction with initialize.

### 19.1.4 get\_current\_state

Item	Content
Function Name	get_current_state
Function Declaration	RobotState get_current_state()
Function Overview	Get current robot aggregated state
Return Value	tuple[Status, RobotState] object
Notes	Non-blocking interface, gets recently monitored robot state data, including BMS/fault state monitoring, etc., will be iteratively expanded

### 19.1.5 RobotState —Robot State Data Structure

Used to represent overall robot state information:

Field Name	Type	Description
faults	list[Fault]	Fault information list
bms_data	BmsData	Battery management system data

### 19.1.6 `BmsData` —Battery Management System Data Structure

Represents battery state information:

Field Name	Type	Description
battery_percentage	float	Current battery remaining charge (percentage, 0~100)
battery_health	float	Battery health status (higher is better)
battery_state	BatteryState	Battery state (see enumeration below)
power_supply_status	PowerSupplyStatus	Battery charge/discharge state (see enumeration below)

### 19.1.7 Enumeration Type Definitions

#### `BatteryState` —Battery State Enumeration Type

Used to represent current battery state, for battery state judgment and processing in the system:

Enumeration Value	Value	Description
BatteryState.UNKNOWN	0	Unknown state
BatteryState.GOOD	1	Battery state good
BatteryState.OVERHEAT	2	Battery overheating
BatteryState.DEAD	3	Battery damaged
BatteryState.OVERTVOLTAGE	4	Battery overvoltage
BatteryState.UNSPEC_FAILURE	5	Unknown failure
BatteryState.COLD	6	Battery overcooled
BatteryState.WATCHDOG_TIMER_EXPIRE	7	Watchdog timer expired
BatteryState.SAFETY_TIMER_EXPIRE	8	Safety timer expired

#### `PowerSupplyStatus` —Battery Charge/Discharge State

Used to represent current battery charge/discharge state:

Enumeration Value	Value	Description
PowerSupplyStatus.UNKNOWN	0	Unknown state
PowerSupplyStatus.CHARGING	1	Battery charging
PowerSupplyStatus.DISCHARGING	2	Battery discharging
PowerSupplyStatus.NOTCHARGING	3	Battery not charging/discharging
PowerSupplyStatus.FULL	4	Battery fully charged

## 19.2 Error Code Mapping Table

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

continues on next page

Table 1 – continued from previous page

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



# CHAPTER 20

---

## SLAM Navigation Control Service (Python)

---

---

Provides SLAM (Simultaneous Localization and Mapping) and navigation control services for robot systems. Through `SlamNavController`, RPC communication can be used to control robot mapping, localization, navigation and other functions.

### 20.1 Interface Definition

`SlamNavController` is a controller for SLAM and navigation control, supporting control operations such as mapping, localization, navigation, and map management, encapsulating underlying details for upper-level system calls.

#### 20.1.1 `SlamNavController`

Item	Content
Class Name	<code>SlamNavController</code>
Class Declaration	<code>SlamNavController()</code>
Function Overview	Constructor, initializes SLAM navigation controller state
Remarks	Constructs internal control resources

### 20.1.2 initialize

Item	Content
Method Name	initialize
Method Declaration	bool initialize()
Function Overview	Initializes SLAM navigation controller, prepares SLAM and navigation functions
Return Value	True indicates success, False indicates failure
Remarks	Must be called before first use

### 20.1.3 shutdown

Item	Content
Method Name	shutdown
Method Declaration	void shutdown()
Function Overview	Closes controller and releases resources
Remarks	Used in conjunction with initialize, safely disconnects

### 20.1.4 activate\_slam\_mode

Item	Content
Method Name	activate_slam_mode
Method Declaration	Status activate_slam_mode(SlamMode mode, str map_path = "", int timeout_ms)
Function Overview	Activates SLAM mode and starts mapping or localization mode
Parameter Description	mode: SLAM mode enumeration map_path: Map path (used in localization mode), can be obtained through get_map_path timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, supports mapping and localization mode switching

### 20.1.5 start\_mapping

Item	Content
Method Name	start_mapping
Method Declaration	Status start_mapping(int timeout_ms)
Function Overview	Starts mapping
Parameter Description	timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, mapping mode must be activated first

### 20.1.6 cancel\_mapping

Item	Content
Method Name	cancel_mapping
Method Declaration	Status cancel_mapping(int timeout_ms)
Function Overview	Cancels mapping
Parameter Description	timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, cancels current mapping task

### 20.1.7 save\_map

Item	Content
Method Name	save_map
Method Declaration	Status save_map(str map_name, int timeout_ms)
Function Overview	Ends mapping and saves the map
Parameter Description	map_name: Map name timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, must be called in mapping mode

## 20.1.8 load\_map

Item	Content
Method Name	load_map
Method Declaration	Status load_map(str map_name, int timeout_ms)
Function Overview	Loads map and sets it as current map
Parameter Description	map_name: Map name timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, loads map with specified name

## 20.1.9 delete\_map

Item	Content
Method Name	delete_map
Method Declaration	Status delete_map(str map_name, int timeout_ms)
Function Overview	Deletes map
Parameter Description	map_name: Name of map to delete timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, permanently deletes specified map

## 20.1.10 get\_map\_path

Item	Content
Method Name	get_map_path
Method Declaration	tuple[Status, List[str]] get_map_path(str map_name, int timeout_ms)
Function Overview	Get map path
Parameter Description	map_name: Map name timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure List[str]: Map path (output parameter)
Remarks	Blocking interface, gets the storage path of the specified map

### 20.1.11 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

### 20.1.12 init\_pose

Item	Content
Method Name	init_pose
Method Declaration	Status init_pose(Pose3DEuler pose, int timeout_ms)
Function Overview	Initializes pose
Parameter Description	pose: Pose information to publish timeout_ms: Timeout time (milliseconds), default value is 15000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, sets robot initial pose; when publishing pose, a fixed -1.57rad offset needs to be applied to the Yaw direction because there is a -1.57rad offset between the mechanical installation of the LiDAR and the robot body

### 20.1.13 get\_current\_localization\_info

Item	Content
Method Name	get_current_localization_info
Method Declaration	Status get_current_localization_info(LocalizationInfo pose_info, int timeout_ms)
Function Overview	Gets current pose information
Parameter Description	pose_info: Current position and orientation information (output parameter) timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, gets robot current pose status

### 20.1.14 activate\_nav\_mode

Item	Content
Method Name	activate_nav_mode
Method Declaration	Status activate_nav_mode(NavMode mode, str map_path = "", int timeout_ms)
Function Overview	Activates navigation mode
Parameter Description	mode: Target navigation mode map_path: Map path (used in grid map mode), can be obtained through get_map_path timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, activates specified navigation mode

### 20.1.15 set\_nav\_target

Item	Content
Method Name	set_nav_target
Method Declaration	Status set_nav_target (NavTarget goal, int timeout_ms)
Function Overview	Sets global navigation target point and starts navigation task
Parameter Description	goal: Global coordinates of target point timeout_ms: Timeout time (milliseconds), default value is 10000
Return Value	Status::OK indicates success, others indicate failure
Remarks	Blocking interface, sets navigation target and starts navigation

### 20.1.16 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	Non-blocking interface, pauses navigation task

### 20.1.17 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	Non-blocking interface, resumes paused navigation task

### 20.1.18 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	Non-blocking interface, cancels current navigation task

### 20.1.19 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.

### 20.1.20 open\_odometry\_stream

Item	Content
Function Name	open_odometry_stream
Function Declaration	Status open_odometry_stream()
Function Overview	Open odometry stream
Return Value	Status object, Status.code == ErrorCode.OK indicates success
Remarks	Blocking interface, check return value to ensure successful execution

### 20.1.21 close\_odometry\_stream

Item	Content
Function Name	close_odometry_stream
Function Declaration	Status close_odometry_stream()
Function Overview	Close odometry stream
Return Value	Status object, Status.code == ErrorCode.OK indicates success
Remarks	Blocking interface, used in conjunction with open_odometry_stream

### 20.1.22 subscribe\_odometry

Item	Content
Function Name	subscribe_odometry
Function Declaration	void subscribe_odometry(callback)
Function Overview	Subscribe to odometry data
Parameter Description	callback: Processing function after receiving odometry data, signature is callback(data: Odometry) -> None
Remarks	Non-blocking interface, callback function will be called when odometry data is updated

### 20.1.23 unsubscribe\_odometry

Item	Content
Function Name	unsubscribe_odometry
Function Declaration	void unsubscribe_odometry()
Function Overview	Unsubscribe from odometry data
Remarks	Non-blocking interface. Used in conjunction with subscribe_odometry

### 20.1.24 get\_point\_cloud\_map

Item	Content
Function Name	get_point_cloud_map
Function Declaration	tuple[Status, PointCloud2] get_point_cloud_map(int timeout_ms)
Function Overview	Get point cloud map data
Parameter Description	timeout_ms: Timeout duration (milliseconds)
Return Value	Returns a tuple, first element is a Status object, Status.code == ErrorCode.OK indicates success; second element is the point cloud map data
Remarks	Blocking interface, gets current point cloud map data

### 20.1.25 unsubscribe\_localization\_point\_cloud\_map

Item	Content
Function Name	unsubscribe_localization_point_cloud_map
Function Declaration	void unsubscribe_localization_point_cloud_map()
Function Overview	Unsubscribe from localization point cloud map data
Remarks	Non-blocking interface. Used in conjunction with subscribe_localization_point_cloud_map

## 20.2 Type Definitions

### 20.2.1 Pose3DEuler —3D Pose Structure

Field Name	Type	Description
position	List[float]	Position coordinates (x, y, z)
orientation	List[float]	Euler angles (roll, pitch, yaw)

## 20.2.2 NavTarget —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

## 20.2.3 LocalizationInfo —Pose Information Structure

Field Name	Type	Description
is_localization	bool	Whether localized
pose	Pose3DEuler	Current pose

## 20.2.4 NavStatus —Navigation Status Structure

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

## 20.2.5 PolyRegion —Polygon Region Structure

Field Name	Type	Description
points	List[Point2D]	Polygon vertex list

## 20.2.6 Point2D —2D Point Structure

Field Name	Type	Description
x	float	X coordinate
y	float	Y coordinate

## 20.3 Enumeration Type Definitions

### 20.3.1 `SlamMode` —SLAM Mode Enumeration

Enum Value	Value	Description
IDLE	0	Idle mode
LOCALIZATION	1	Localization mode
MAPPING	2	Mapping mode

### 20.3.2 `NavMode` —Navigation Mode Enumeration

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

### 20.3.3 `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	Canceled

## 20.4 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.

### 20.4.1 Applicable Scenarios and Scope

- Static indoor flat areas smaller than 100 m × 100 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.

### 20.4.2 SLAM Function Flow

Function	Operation Flow
Mapping	Activate mapping mode → Start mapping → Save map
Map Management	Load map, get map information, delete map, get map path, etc.

### 20.4.3 Navigation Function Flow

Function	Operation Flow
Localization	Load map → Activate localization mode → Initialize pose → Get localization status
Navigation	Localization success → Activate navigation mode → Set target pose → Get navigation status
Navigation Control	Set target pose → PauseNavTask navigation → ResumeNavTask navigation → CancelNavTask navigation

### 20.4.4 Localization Mode Initialization

When SLAM is in localization mode, an initial pose can be specified to quickly complete relocalization and initialization. For large-scale maps, specifying the initial pose is mandatory.

#### Small Map Mode

满足如下条件：

The following conditions must be met:

- Condition 1: The map should not be too large. For example, in a single room, it is best to keep the map area within 10m × 10m.
- Condition 2: Modify the configuration to adapt to this mode:

```
# Log in to the robot
ssh eame@192.168.54.119
# Modify the following configuration
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
reloc_options: true
```

(continues on next page)

(continued from previous page)

```
# After modifying the YAML file for the first time, restart the SLAM module for the
→change to take effect:
sudo systemctl restart slam.service
```

## Large Map Mode

The following conditions must be met:

- Condition 1: The robot must be located at the mapping origin (the starting position when mapping began), and the robot’s orientation must remain consistent.
- Condition 2: Modify the configuration to adapt to this mode:

When using the SDK for SLAM navigation (unlike the app, which allows convenient visual positioning), you must modify the SLAM configuration to fix the map orientation and facilitate localization:

```
# Log in to the robot
ssh eame@192.168.54.119
# Modify the following configuration
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# After modifying the YAML file for the first time, restart the SLAM module for the
→change to take effect:
sudo systemctl restart slam.service
```

- Condition 3: When initializing the pose, apply a -90° (-1.57 rad) offset to the yaw angle:

because the LiDAR’s mechanical installation differs from the robot body orientation by -90°.

## 20.4.5 Map Noise Removal and Obstacle Addition

- Software Name: GIMP
- Download Link: <https://www.gimp.org/downloads/>
- Operating System: Ubuntu
- File to Edit: /home/eame/cust\_para/maps/\${map\_name}/map/map.pgm Log in to the robot (ssh eame@192.168.54.119), open the .pgm map file with GIMP for editing, and replace the original file after modifications are complete.
- Editing Tutorial Video:

# CHAPTER 21

---

## High-Level Motion Control Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for initialization, robot connection, high-level motion control (gait, trick actions, remote control) and other basic operations.

### 21.1 C++

Example file: `high_level_motion_example.cpp`

Reference documentation: `high_level_motion_reference.md`

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

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

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
```

(continues on next page)

(continued from previous page)

```

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

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

void print_help() {
    std::cout << "Key Function Description:\n";
    std::cout << "Gait and Trick Functions:\n";
    std::cout << "  1      Function 1: Recovery stand\n";
    std::cout << "  2      Function 2: Balance stand\n";
    std::cout << "  3      Function 3: Execute trick - celebrate action\n";
    std::cout << "\n";
    std::cout << "Joystick Functions:\n";
    std::cout << "  w      Function 4: Move forward\n";
    std::cout << "  a      Function 5: Move left\n";
    std::cout << "  s      Function 6: Move backward\n";
    std::cout << "  d      Function 7: Move right\n";
    std::cout << "  x      Function 8: Stop\n";
    std::cout << "  t      Function 9: Turn left\n";
    std::cout << "  g      Function 10: Turn right\n";
    std::cout << "\n";
    std::cout << "Head Functions:\n";
    std::cout << "  b      Function 11: Head look up\n";
    std::cout << "  j      Function 12: Head look down\n";
    std::cout << "  k      Function 13: Head turn left\n";
    std::cout << "  l      Function 14: Head turn right\n";
    std::cout << "  u      Function 15: Head reset\n";
    std::cout << "\n";
    std::cout << "  ESC    Exit program\n";
    std::cout << "  ?      Function ?: Print help\n";
}

int getch() {
    struct termios oldt, newt;
    int ch;

    tcgetattr(STDIN_FILENO, &oldt); // Get current terminal settings
    newt = oldt;
}

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

    // Head look up
}

```

(continues on next page)

(continued from previous page)

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

    print_help();

    std::string local_ip = "192.168.54.111";
    // Configure local IP address for direct network connection and initialize SDK
}

```

(continues on next page)

(continued from previous page)

```

if (!robot.Initialize(local_ip)) {
    std::cerr << "robot sdk initialize failed." << std::endl;
    robotShutdown();
    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;
    robotShutdown();
    return -1;
}

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

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

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

    std::cout << "Key ASCII: " << key << ", Character: " << static_cast<char>(key) <<
→std::endl;
    switch (key) {
        // 1. Gait and Trick Functions

```

(continues on next page)

(continued from previous page)

```

case '1': {
    RecoveryStand();
    break;
}

case '2': {
    BalanceStand();
    break;
}

case '3': {
    ExecuteTrick();
    break;
}

// 2. Joystick Functions

case 'w':
case 'W': {
    JoyStickCommand(0.0, 1.0, 0.0, 0.0); // Forward
    break;
}

case 'a':
case 'A': {
    JoyStickCommand(-1.0, 0.0, 0.0, 0.0); // Left
    break;
}

case 's':
case 'S': {
    JoyStickCommand(0.0, -1.0, 0.0, 0.0); // Backward
    break;
}

case 'd':
case 'D': {
    JoyStickCommand(1.0, 0.0, 0.0, 0.0); // Right
    break;
}

case 'x':
case 'X': {
    JoyStickCommand(0.0, 0.0, 0.0, 0.0); // Stop
    break;
}

case 't':
case 'T': {

```

(continues on next page)

(continued from previous page)

```
JoyStickCommand(0.0, 0.0, -1.0, 0.0); // Turn left
break;
}
case 'g':
case 'G': {
    JoyStickCommand(0.0, 0.0, 1.0, 0.0); // Turn right
    break;
}
// 3. Head Functions
case 'b': {
    HeadLookUp();
    break;
}
case 'j': {
    HeadLookDown();
    break;
}
case 'k': {
    HeadTurnLeft();
    break;
}
case 'l': {
    HeadTurnRight();
    break;
}
case 'u': {
    HeadReset();
    break;
}
case '?': {
    print_help();
    break;
}
default:
    std::cout << "Unknown key: " << key << std::endl;
    break;
}
usleep(10000);
}
```

(continues on next page)

(continued from previous page)

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

robot.Shutdown();

return 0;
}
```

## 21.2 Python

Example file: `high_level_motion_example.py`

Reference documentation: `high_level_motion_reference.md`

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

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

import magicbot_gen1_python as magicbot

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

(continues on next page)

(continued from previous page)

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

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

def print_help():
    """Print help information"""
    logging.info("High-Level Motion Control Function Demo Program")
    logging.info("")
    logging.info("Gait and Trick Functions:")
    logging.info("  0      Function 0: Pure damper")
    logging.info("  1      Function 1: Recovery stand")
    logging.info("  2      Function 2: Balance stand")
    logging.info("  3      Function 3: Execute trick - celebrate action")
    logging.info("")
    logging.info("Joystick Functions:")
    logging.info("  w      Function 4: Move forward")
    logging.info("  a      Function 5: Move left")
    logging.info("  s      Function 6: Move backward")
    logging.info("  d      Function 7: Move right")
    logging.info("  x      Function 8: Stop move")
    logging.info("  t      Function 9: Turn left")
    logging.info("  g      Function 10: Turn right")
    logging.info("")
    logging.info("Head Functions:")
    logging.info("  b      Function 11: Head look up")
    logging.info("  j      Function 12: Head look down")
    logging.info("  k      Function 13: Head turn left")
```

(continues on next page)

(continued from previous page)

```

logging.info(" l           Function 14: Head turn right")
logging.info(" u           Function 15: Head reset")
logging.info("")
logging.info(" ?          Function ?: Print help")
logging.info(" ESC        Exit program")

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```
return False

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

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

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

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

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

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

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

        # Set gait to balance stand
```

(continues on next page)

(continued from previous page)

```

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

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

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


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

```

(continues on next page)

(continued from previous page)

```

elif cmd == "306":
    return magicbot.TrickAction.ACTION_TRUN_AWAY_LEFT_INTRODUCE
elif cmd == "307":
    return magicbot.TrickAction.ACTION_TRUN_BACK_LEFT_INTRODUCE
else:
    return magicbot.TrickAction.ACTION_NONE


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

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

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

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

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


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

```

(continues on next page)

(continued from previous page)

```

controller = robot.get_high_level_motion_controller()

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

def move_right():
    """Move right"""
    logging.info("==== Moving Right ====")
    return joystick_command(1.0, 0.0, 0.0, 0.0)

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

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

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

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

        # Move head
        status = controller.head_move(shake_angle, nod_angle, 5000)

```

(continues on next page)

(continued from previous page)

```

if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to move head, code: %s, message: %s",
        status.code,
        status.message,
    )
    return False

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

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


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


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


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

```

(continues on next page)

(continued from previous page)

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

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

# Get single character input (no echo)
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
        logging.info(f"Received character: {ch}")

        sys.stdout.write("\r")
        sys.stdout.flush()
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

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

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

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

    # Create robot instance
```

(continues on next page)

(continued from previous page)

```
robot = magicbot.MagicRobot()

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

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

    logging.info("Successfully connected to robot")

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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


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

        # 1. Gait and Trick Functions
        # 1.1 Pure damper mode
        if key == "0":
            pure_damper()
        # 1.2 Recovery stand
        elif key == "1":
            recovery_stand()
        # 1.3 Balance stand
        elif key == "2":
            balance_stand()
        # 1.4 Execute trick action
        elif key == "3":
            str_input = get_user_input()
            # Split input parameters by space
            parts = str_input.strip().split()
            # Parse parameters
            cmd = parts[0] if parts else "201"
            print("cmd: ", cmd)
            execute_trick_action(cmd)

# 2. Joystick Functions

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

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

# 3. Head Functions

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

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

            time.sleep(0.01) # Brief delay

    except KeyboardInterrupt:
        break

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

    return 0

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

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

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

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

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

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

## 21.3 Running Instructions

### 21.3.1 Environment Setup

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

### 21.3.2 Run Examples

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

### 21.3.3 Control Instructions

- 0: Pure damper mode
- 1: Recovery stand
- 2: Balance stand
- 3: Execute celebrate action
- W/A/S/D/X: Move forward/left/backward/right/stop
- T/G: Turn left/right
- B/J/K/L/U: Head look up/down/left/right/reset
- ?: Print help information

Note: The humanoid robot must land on the ground in the `recovery stand` state. After landing, switch to `balance stand`, and only then can perform tricks or other actions.

### 21.3.4 Stop Program

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

# CHAPTER 22

---

## Low-Level Motion Control Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for initialization, robot connection, low-level motion control (hands, arms, legs, waist, head), IMU sensor data reading and other basic operations.

### 22.1 C++

Example file: `low_level_motion_example.cpp`

Reference documentation: `low_level_motion_reference.md`

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

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

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
```

(continues on next page)

(continued from previous page)

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

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

    running = false;

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

        std::cout << "error_code: " << msg->joints[0].err_code << std::endl;
    }
    // TODO: handle arm joint data
});

// Using arm joint control as an example:
// Subsequent joint control commands, joint operation mode is 1, indicating the_
// joint is in position control mode
auto now = std::chrono::steady_clock::now();
while (running.load()) {
    // Left arm joints, refer to documentation:
    // Left arm or right arm joints 1-5 operation_mode needs to switch from mode: 200_
    // to mode: 4 (series PID mode) for command execution;
    JointCommand arm_command;
    arm_command.joints.resize(kArmJointNum);
    for (int ii = 0; ii < kArmJointNum; ii++) {
        // Set joint to ready state
        arm_command.joints[ii].operation_mode = 200;
        // TODO: Set target position, velocity, torque and gains
        arm_command.joints[ii].pos = 0.0;
        arm_command.joints[ii].vel = 0.0;
        arm_command.joints[ii].toq = 0.0;
        arm_command.joints[ii].kp = 0.0;
        arm_command.joints[ii].kd = 0.0;
    }
    // Publish control command
    controller.PublishArmCommand(arm_command);

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

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

```

(continues on next page)

(continued from previous page)

```

    return -1;
}

robot.Shutdown();

return 0;
}

```

## 22.2 Python

Example file: low\_level\_motion\_example.py

Reference documentation: low\_level\_motion\_reference.md

```

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

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

import magicbot_gen1_python as magicbot

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

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

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

```

(continues on next page)

(continued from previous page)

```
waist_state_counter = 0

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

def body_imu_callback(imu_data):
    """Body IMU data callback function"""
    global body_imu_counter
    if body_imu_counter % 1000 == 0:
        logging.info("++++++ Received body IMU data")
        # Print IMU data
        logging.info("Received body IMU data, timestamp: %d", imu_data.timestamp)
        logging.info(
            "Received body IMU data, orientation: [%f, %f, %f, %f],",
            imu_data.orientation[0],
            imu_data.orientation[1],
            imu_data.orientation[2],
            imu_data.orientation[3],
        )
        logging.info(
            "Received body IMU data, angular_velocity: [%f, %f, %f],",
            imu_data.angular_velocity[0],
            imu_data.angular_velocity[1],
            imu_data.angular_velocity[2],
        )
        logging.info(
            "Received body IMU data, linear_acceleration: [%f, %f, %f],",
            imu_data.linear_acceleration[0],
            imu_data.linear_acceleration[1],
            imu_data.linear_acceleration[2],
        )
    
```

(continues on next page)

(continued from previous page)

```

        logging.info("Received body IMU data, temperature: %s", imu_data.temperature)
body_imu_counter += 1

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

def leg_state_callback(joint_state):
    """Leg joint state callback function"""
    global leg_state_counter

```

(continues on next page)

(continued from previous page)

```

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

```

```

def head_state_callback(joint_state):
    """Head joint state callback function"""
    global head_state_counter
    if head_state_counter % 1000 == 0:
        logging.info("++++++ Received head joint state data")
        # Print joint state data
        logging.info(
            "Received head joint state data, status_word: %d",
            joint_state.joints[0].status_word,
        )

```

(continues on next page)

(continued from previous page)

```

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

    # Create robot instance
    robot = magicbot.MagicRobot()

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

        # Connect to robot
        status = robot.connect()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(

```

(continues on next page)

(continued from previous page)

```
"Failed to connect to robot, code: %s, message: %s",
status.code,
status.message,
)
robot.shutdown()
return -1

logging.info("Successfully connected to robot")

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

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

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

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

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

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

# Subscribe to head joint state
controller.subscribe_head_state(head_state_callback)
```

(continues on next page)

(continued from previous page)

```
logging.info("Subscribed to head joint state")

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

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

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

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

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

    for i in range(magicbot.HEAD_JOINT_NUM):
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

finally:
    # Clean up resources

    try:
        logging.info("Clean up resources")
        # Close low-level motion controller
        controller = robot.get_low_level_motion_controller()
        controller.shutdown()
        logging.info("Low-level motion controller closed")

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

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

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

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

```

## 22.3 Running Instructions

### 22.3.1 Environment Setup

```

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

```

### 22.3.2 Run Examples

```

# C++
./low_level_motion_example

# Python
python3 low_level_motion_example.py

```

### 22.3.3 Stop Program

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



# CHAPTER 23

---

## Sensor Control Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for initialization, robot connection, robot sensor control (lidar, RGBD camera, trinocular camera) and other basic operations.

### 23.1 C++

Example file: `sensor_example.cpp`

Reference documentation: `sensor_reference.md`

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

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

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

(continues on next page)

(continued from previous page)

```

using namespace magic::gen1;

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

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

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

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

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

(continues on next page)

```

(continued from previous page)

```

    return false;
}

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

    return false;
}

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

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

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

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

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

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

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

    auto status = sensor_controller_.OpenWaistRgbdCamera();
    if (status.code != ErrorCode::OK) {

```

(continues on next page)

(continued from previous page)

```

        std::cerr << "[ERROR] Failed to open waist RGBD camera: " << status.message <<
→std::endl;
    return false;
}

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

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

    if (subscriptions_[ "waist_rgbd_color_image" ]) ←
→ToggleWaistRgbdColorImageSubscription();
    if (subscriptions_[ "waist_rgbd_depth_image" ]) ←
→ToggleWaistRgbdDepthImageSubscription();

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

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

// === Trinocular Camera Control ===

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

// === LiDAR Subscriptions ===

void ToggleLidarImuSubscription() {
    if (subscriptions_["lidar_imu"]) {
        sensor_controller_.UnsubscribeLidarImu();
        subscriptions_["lidar_imu"] = false;
        std::cout << "[INFO] ✓ LiDAR IMU unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeLidarImu([] (const std::shared_ptr<Imu> msg) {

```

(continues on next page)

(continued from previous page)

```

        lidar_imu_counter++;

    if (lidar_imu_counter % 100 == 0) {
        std::cout << "[DATA] LiDAR IMU received (count: " << lidar_imu_counter << ")"
        <> std::endl;
    }
}

subscriptions_["lidar_imu"] = true;
std::cout << "[INFO] ✓ LiDAR IMU subscribed" << std::endl;
}

}

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

// === Head RGBD Subscriptions ===
void ToggleHeadRgbdColorImageSubscription() {
    if (subscriptions_["head_rgbd_color_image"]) {
        sensor_controller_.UnsubscribeHeadRgbdColorImage();
        subscriptions_["head_rgbd_color_image"] = false;
        std::cout << "[INFO] ✓ Head RGBD color image unsubscribed" << std::endl;
    } else {
        sensor_controller_.SubscribeHeadRgbdColorImage([] (const std::shared_ptr<Image>_
        &msg) {

```

(continues on next page)

(continued from previous page)

```

        head_rgbd_color_counter++;

    if (head_rgbd_color_counter % 15 == 0) {
        std::cout << "[DATA] Head RGBD Color Image: " << msg->data.size()
            << " bytes, " << msg->width << "x" << msg->height
            << " (count: " << head_rgbd_color_counter << ")" << std::endl;
    }
}

subscriptions_["head_rgbd_color_image"] = true;
std::cout << "[INFO] ✓ Head RGBD color image subscribed" << std::endl;
}

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

// === Waist RGBD Subscriptions ===

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

```

(continues on next page)

(continued from previous page)

```

→msg) {
    waist_rgbd_color_counter++;
    if (waist_rgbd_color_counter % 15 == 0) {
        std::cout << "[DATA] Waist RGBD Color Image: " << msg->data.size()
            << " bytes, " << msg->width << "x" << msg->height
            << " (count: " << waist_rgbd_color_counter << ")" << std::endl;
    }
};

subscriptions_["waist_rgbd_color_image"] = true;
std::cout << "[INFO] ✓ Waist RGBD color image subscribed" << std::endl;
}

}

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

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

```

(continues on next page)

(continued from previous page)

```

sensor_controller_.SubscribeTrinocularImage([](const std::shared_ptr<
    <TrinocularCameraFrame> msg) {
    trinocular_image_counter++;

    if (trinocular_image_counter % 15 == 0) {
        std::cout << "[DATA] Trinocular Camera Frame received (count: "
            << trinocular_image_counter << ")" << std::endl;
    }
});

subscriptions_[ "trinocular_image" ] = true;
std::cout << "[INFO] ✓ Trinocular camera image subscribed" << std::endl;
}

}

void ShowStatus() {
    std::cout << "\n"
        << std::string(80, '=') << std::endl;

    std::cout << "MAGICBOT GEN1 SENSOR STATUS" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
    std::cout << "LiDAR: " << (sensors_state_[ "lidar" ] ? "OPEN"
        : "CLOSED") << std::endl;
    std::cout << "Head RGBD Camera: " << (sensors_state_[ "head_rgbd_"
        "camera" ] ? "OPEN" : "CLOSED") << std::endl;
    std::cout << "Waist RGBD Camera: " << (sensors_state_[ "waist_rgbd_"
        "camera" ] ? "OPEN" : "CLOSED") << std::endl;
    std::cout << "Trinocular Camera: " << (sensors_state_[ "trinocular_"
        "camera" ] ? "OPEN" : "CLOSED") << std::endl;
    std::cout << "\nLIDAR SUBSCRIPTIONS:" << std::endl;
    std::cout << " LiDAR IMU: " << (subscriptions_[ "lidar_imu" ] ?
        "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " LiDAR Point Cloud: " << (subscriptions_[ "lidar_point_"
        "cloud" ] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nHEAD RGBD SUBSCRIPTIONS:" << std::endl;
    std::cout << " Color Image: " << (subscriptions_[ "head_rgbd_"
        "color_image" ] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " Depth Image: " << (subscriptions_[ "head_rgbd_"
        "depth_image" ] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nWAIST RGBD SUBSCRIPTIONS:" << std::endl;
    std::cout << " Color Image: " << (subscriptions_[ "waist_rgbd_"
        "color_image" ] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << " Depth Image: " << (subscriptions_[ "waist_rgbd_"
        "depth_image" ] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
}

```

(continues on next page)

(continued from previous page)

```

→depth_image"] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << "\nTRINOCULAR CAMERA SUBSCRIPTIONS:" << std::endl;
    std::cout << "  Trinocular Image:           " << (subscriptions_["trinocular_
→image"] ? "✓ SUBSCRIBED" : "✗ UNSUBSCRIBED") << std::endl;
    std::cout << std::string(80, '=') << "\n"
                << std::endl;
}

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

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

void PrintMenu() {
    std::cout << "\n"
                << std::string(80, '=') << std::endl;
    std::cout << "MAGICBOT GEN1 SENSOR CONTROL MENU" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
    std::cout << "Sensor Open/Close:" << std::endl;
    std::cout << "  1 - Open LiDAR           2 - Close LiDAR" << std::endl;
    std::cout << "  3 - Open Head RGBD Camera 4 - Close Head RGBD Camera" <<
→std::endl;
    std::cout << "  5 - Open Waist RGBD Camera 6 - Close Waist RGBD Camera" <<
→std::endl;
    std::cout << "  7 - Open Trinocular Camera 8 - Close Trinocular Camera" <<
→std::endl;
    std::cout << "\nLiDAR Subscriptions:" << std::endl;
    std::cout << "  i - Toggle LiDAR IMU          p - Toggle LiDAR Point Cloud" <<
→std::endl;
    std::cout << "\nHead RGBD Camera Subscriptions:" << std::endl;
    std::cout << "  c - Toggle Head Color Image   d - Toggle Head Depth Image" <<
→std::endl;
}

```

(continues on next page)

(continued from previous page)

```

    std::cout << "\nWaist RGBD Camera Subscriptions:" << std::endl;
    std::cout << "  w - Toggle Waist Color Image      e - Toggle Waist Depth Image" << std::endl;
    std::cout << "\nTrinocular Camera Subscriptions:" << std::endl;
    std::cout << "  t - Toggle Trinocular Image" << std::endl;
    std::cout << "\nCommands:" << std::endl;
    std::cout << "  s - Show Status           ESC - Exit program     ? - Help" << std::endl;
    std::cout << std::string(80, '=') << std::endl;
}

magic::gen1::MagicRobot robot;
SensorManager* g_sensor_manager = nullptr;

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

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

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

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

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

    std::string local_ip = "192.168.54.111";

    // Initialize SDK
}

```

(continues on next page)

(continued from previous page)

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

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

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

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

PrintMenu();

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

        if (choice.empty()) continue;

        char cmd = choice[0];

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

(continues on next page)

(continued from previous page)

```
// Sensor open/close control
switch (cmd) {
    case '1':
        sensor_manager.OpenLidar();
        break;
    case '2':
        sensor_manager.CloseLidar();
        break;
    case '3':
        sensor_manager.OpenHeadRgbdCamera();
        break;
    case '4':
        sensor_manager.CloseHeadRgbdCamera();
        break;
    case '5':
        sensor_manager.OpenWaistRgbdCamera();
        break;
    case '6':
        sensor_manager.CloseWaistRgbdCamera();
        break;
    case '7':
        sensor_manager.OpenTrinocularCamera();
        break;
    case '8':
        sensor_manager.CloseTrinocularCamera();
        break;

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

    // Head RGBD subscriptions
    case 'c':
        sensor_manager.ToggleHeadRgbdColorImageSubscription();
        break;
```

(continues on next page)

(continued from previous page)

```
case 'd':
    sensor_manager.ToggleHeadRgbdDepthImageSubscription();
    break;

// Waist RGBD subscriptions
case 'w':
    sensor_manager.ToggleWaistRgbdColorImageSubscription();
    break;
case 'e':
    sensor_manager.ToggleWaistRgbdDepthImageSubscription();
    break;

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

// Commands
case 's':
case 'S':
    sensor_manager.ShowStatus();
    break;
case '?':
    PrintMenu();
    break;

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

}

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

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

(continues on next page)

(continued from previous page)

```
sleep(1); // Allow time for cleanup

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

robot.Shutdown();
std::cout << "[INFO] Robot shutdown" << std::endl;

return 0;
}
```

## 23.2 Python

Example file: sensor\_example.py

Reference documentation: sensor\_reference.md

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

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

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

(continues on next page)

(continued from previous page)

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

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

    def __init__(self, sensor_controller):
        self.sensor_controller = sensor_controller
        self.sensors_state = {
            "lidar": False,
            "head_rgbd_camera": False,
            "waist_rgbd_camera": False,
            "trinocular_camera": False,
        }
        self.subscriptions = {
            # LiDAR subscriptions
            "lidar_imu": False,
            "lidar_point_cloud": False,
            # Head RGBD subscriptions
            "head_rgbd_color_image": False,
            "head_rgbd_color_camera_info": False,
            "head_rgbd_depth_image": False,
            "head_rgbd_depth_camera_info": False,
            # Waist RGBD subscriptions
            "waist_rgbd_color_image": False,
            "waist_rgbd_color_camera_info": False,
            "waist_rgbd_depth_image": False,
            "waist_rgbd_depth_camera_info": False,
            # Trinocular camera subscriptions
        }
```

(continues on next page)

(continued from previous page)

```
"trinocular_image": False,  
}  
  
# === LiDAR Control ===  
  
def open_lidar(self) -> bool:  
    """Open LiDAR"""  
  
    if self.sensors_state["lidar"]:  
        logging.warning("LiDAR already opened")  
        return True  
  
    status = self.sensor_controller.open_lidar()  
    if status.code != ErrorCode.OK:  
        logging.error(f"Failed to open LiDAR: {status.message}")  
        return False  
  
    self.sensors_state["lidar"] = True  
    logging.info("✓ LiDAR opened successfully")  
    return True  
  
def close_lidar(self) -> bool:  
    """Close LiDAR"""  
  
    if not self.sensors_state["lidar"]:  
        logging.warning("LiDAR already closed")  
        return True  
  
    # Unsubscribe if subscribed  
    if self.subscriptions["lidar_imu"]:  
        self.toggle_lidar_imu_subscription()  
    if self.subscriptions["lidar_point_cloud"]:  
        self.toggle_lidar_point_cloud_subscription()  
  
    status = self.sensor_controller.close_lidar()  
    if status.code != ErrorCode.OK:  
        logging.error(f"Failed to close LiDAR: {status.message}")  
        return False  
  
    self.sensors_state["lidar"] = False  
    logging.info("✓ LiDAR closed")  
    return True
```

(continues on next page)

(continued from previous page)

```

# === Head RGBD Camera Control ===

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

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

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


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

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

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

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

```

(continues on next page)

(continued from previous page)

```

# === Waist RGBD Camera Control ===

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

    return True

# === Trinocular Camera Control ===

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

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

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

def close_trinocular_camera(self) -> bool:
    """Close trinocular camera"""
    if not self.sensors_state["trinocular_camera"]:
        logging.warning("Trinocular camera already closed")
        return True

    # Unsubscribe if subscribed
    if self.subscriptions["trinocular_image"]:
        self.toggle_trinocular_image_subscription()

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

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

# === LiDAR Subscribe/Unsubscribe Methods ===

def toggle_lidar_imu_subscription(self):
    """Toggle LiDAR IMU subscription"""

```

(continues on next page)

(continued from previous page)

```

if self.subscriptions["lidar_imu"]:
    self.sensor_controller.unsubscribe_lidar_imu()
    self.subscriptions["lidar_imu"] = False
    logging.info("✓ LiDAR IMU unsubscribed")

else:

    def lidar_imu_callback(imu):
        global lidar_imu_counter
        lidar_imu_counter += 1
        if lidar_imu_counter % 100 == 0:
            logging.info(f"LiDAR IMU received (count: {lidar_imu_counter})")

    self.sensor_controller.subscribe_lidar_imu(lidar_imu_callback)
    self.subscriptions["lidar_imu"] = True
    logging.info("✓ LiDAR IMU subscribed")

def toggle_lidar_point_cloud_subscription(self):
    """Toggle LiDAR point cloud subscription"""
    if self.subscriptions["lidar_point_cloud"]:
        self.sensor_controller.unsubscribe_lidar_point_cloud()
        self.subscriptions["lidar_point_cloud"] = False
        logging.info("✓ LiDAR point cloud unsubscribed")
    else:

        def lidar_pointcloud_callback(pointcloud):
            global lidar_pointcloud_counter
            lidar_pointcloud_counter += 1
            if lidar_pointcloud_counter % 10 == 0:
                logging.info(
                    f"LiDAR Point Cloud: {len(pointcloud.data)} bytes (count:
                    ↵{lidar_pointcloud_counter})"
                )

        self.sensor_controller.subscribe_lidar_point_cloud(
            lidar_pointcloud_callback
        )
        self.subscriptions["lidar_point_cloud"] = True
        logging.info("✓ LiDAR point cloud subscribed")

# === Head RGBD Subscribe/Unsubscribe Methods ===

```

(continues on next page)

(continued from previous page)

```

def toggle_head_rgbd_color_image_subscription(self):
    """Toggle head RGBD color image subscription"""
    if self.subscriptions["head_rgbd_color_image"]:
        self.sensor_controller.unsubscribe_head_rgbd_color_image()
        self.subscriptions["head_rgbd_color_image"] = False
        logging.info("✓ Head RGBD color image unsubscribed")
    else:

        def head_rgbd_color_image_callback(img):
            global head_rgbd_color_counter
            head_rgbd_color_counter += 1
            if head_rgbd_color_counter % 15 == 0:
                logging.info(
                    f"Head RGBD Color Image: {len(img.data)} bytes, {img.width}x"
                    f"{img.height} (count: {head_rgbd_color_counter})"
                )

            self.sensor_controller.subscribe_head_rgbd_color_image(
                head_rgbd_color_image_callback
            )
        self.subscriptions["head_rgbd_color_image"] = True
        logging.info("✓ Head RGBD color image subscribed")

    def toggle_head_rgbd_color_camera_info_subscription(self):
        """Toggle head RGBD color camera info subscription"""
        if self.subscriptions["head_rgbd_color_camera_info"]:
            self.sensor_controller.unsubscribe_head_rgbd_color_camera_info()
            self.subscriptions["head_rgbd_color_camera_info"] = False
            logging.info("✓ Head RGBD color camera info unsubscribed")
        else:
            self.sensor_controller.subscribe_head_rgbd_color_camera_info(
                lambda info: logging.info(
                    f"Head RGBD Color Camera Info: {info.width}x{info.height}"
                )
            )
        self.subscriptions["head_rgbd_color_camera_info"] = True
        logging.info("✓ Head RGBD color camera info subscribed")

    def toggle_head_rgbd_depth_image_subscription(self):
        """Toggle head RGBD depth image subscription"""

```

(continues on next page)

(continued from previous page)

```

if self.subscriptions["head_rgbd_depth_image"]:
    self.sensor_controller.unsubscribe_head_rgbd_depth_image()
    self.subscriptions["head_rgbd_depth_image"] = False
    logging.info("✓ Head RGBD depth image unsubscribed")

else:

    def head_rgbd_depth_image_callback(img):
        global head_rgbd_depth_image_counter
        head_rgbd_depth_image_counter += 1
        if head_rgbd_depth_image_counter % 15 == 0:
            logging.info(
                f"Head RGBD Depth Image: {len(img.data)} bytes, {img.width}x"
                f"{img.height} (count: {head_rgbd_depth_image_counter})"
            )

    self.sensor_controller.subscribe_head_rgbd_depth_image(
        head_rgbd_depth_image_callback
    )
    self.subscriptions["head_rgbd_depth_image"] = True
    logging.info("✓ Head RGBD depth image subscribed")

def toggle_head_rgbd_depth_camera_info_subscription(self):
    """Toggle head RGBD depth camera info subscription"""
    if self.subscriptions["head_rgbd_depth_camera_info"]:
        self.sensor_controller.unsubscribe_head_rgbd_depth_camera_info()
        self.subscriptions["head_rgbd_depth_camera_info"] = False
        logging.info("✓ Head RGBD depth camera info unsubscribed")
    else:

        def head_rgbd_depth_camera_info_callback(info):
            global head_rgbd_depth_camera_info_counter
            head_rgbd_depth_camera_info_counter += 1
            if head_rgbd_depth_camera_info_counter % 15 == 0:
                logging.info(
                    f"Head RGBD Depth Camera Info: {info.width}x{info.height}x"
                    f"(count: {head_rgbd_depth_camera_info_counter})"
                )

        self.sensor_controller.subscribe_head_rgbd_depth_camera_info(
            head_rgbd_depth_camera_info_callback

```

(continues on next page)

(continued from previous page)

```

        )

    self.subscriptions["head_rgbd_depth_camera_info"] = True
    logging.info("✓ Head RGBD depth camera info subscribed")

# === Waist RGBD Subscribe/Unsubscribe Methods ===

def toggle_waist_rgbd_color_image_subscription(self):
    """Toggle waist RGBD color image subscription"""
    if self.subscriptions["waist_rgbd_color_image"]:
        self.sensor_controller.unsubscribe_waist_rgbd_color_image()
        self.subscriptions["waist_rgbd_color_image"] = False
        logging.info("✓ Waist RGBD color image unsubscribed")
    else:

        def waist_rgbd_color_image_callback(img):
            global waist_rgbd_color_counter
            waist_rgbd_color_counter += 1
            if waist_rgbd_color_counter % 15 == 0:
                logging.info(
                    f"Waist RGBD Color Image: {len(img.data)} bytes, {img.width}x"
                    f"{img.height} (count: {waist_rgbd_color_counter})"
                )

            self.sensor_controller.subscribe_waist_rgbd_color_image(
                waist_rgbd_color_image_callback
            )
        self.subscriptions["waist_rgbd_color_image"] = True
        logging.info("✓ Waist RGBD color image subscribed")

    def toggle_waist_rgbd_color_camera_info_subscription(self):
        """Toggle waist RGBD color camera info subscription"""
        if self.subscriptions["waist_rgbd_color_camera_info"]:
            self.sensor_controller.unsubscribe_waist_rgbd_color_camera_info()
            self.subscriptions["waist_rgbd_color_camera_info"] = False
            logging.info("✓ Waist RGBD color camera info unsubscribed")
        else:

            def waist_rgbd_color_camera_info_callback(info):
                global waist_rgbd_color_camera_info_counter
                waist_rgbd_color_camera_info_counter += 1
                if waist_rgbd_color_camera_info_counter % 15 == 0:

```

(continues on next page)

(continued from previous page)

```

        logging.info(
            f"Waist RGBD Color Camera Info: {info.width}x{info.height} ↵
↪ (count: {waist_rgbd_color_camera_info_counter})"
        )

        self.sensor_controller.subscribe_waist_rgbd_color_camera_info(
            waist_rgbd_color_camera_info_callback
        )
        self.subscriptions["waist_rgbd_color_camera_info"] = True
        logging.info("✓ Waist RGBD color camera info subscribed")

    def toggle_waist_rgbd_depth_image_subscription(self):
        """Toggle waist RGBD depth image subscription"""
        if self.subscriptions["waist_rgbd_depth_image"]:
            self.sensor_controller.unsubscribe_waist_rgbd_depth_image()
            self.subscriptions["waist_rgbd_depth_image"] = False
            logging.info("✓ Waist RGBD depth image unsubscribed")
        else:

            def waist_rgbd_depth_image_callback(img):
                global waist_rgbd_depth_image_counter
                waist_rgbd_depth_image_counter += 1
                if waist_rgbd_depth_image_counter % 15 == 0:
                    logging.info(
                        f"Waist RGBD Depth Image: {len(img.data)} bytes, {img.width}x
↪ {img.height} (count: {waist_rgbd_depth_image_counter})"
                    )

            self.sensor_controller.subscribe_waist_rgbd_depth_image(
                waist_rgbd_depth_image_callback
            )
            self.subscriptions["waist_rgbd_depth_image"] = True
            logging.info("✓ Waist RGBD depth image subscribed")

    def toggle_waist_rgbd_depth_camera_info_subscription(self):
        """Toggle waist RGBD depth camera info subscription"""
        if self.subscriptions["waist_rgbd_depth_camera_info"]:
            self.sensor_controller.unsubscribe_waist_rgbd_depth_camera_info()
            self.subscriptions["waist_rgbd_depth_camera_info"] = False
            logging.info("✓ Waist RGBD depth camera info unsubscribed")

```

(continues on next page)

(continued from previous page)

```

else:

    def waist_rgbd_depth_camera_info_callback(info):
        global waist_rgbd_depth_camera_info_counter
        waist_rgbd_depth_camera_info_counter += 1
        if waist_rgbd_depth_camera_info_counter % 15 == 0:
            logging.info(
                f"Waist RGBD Depth Camera Info: {info.width}x{info.height} -"
                f" (count: {waist_rgbd_depth_camera_info_counter})"
            )

        self.sensor_controller.subscribe_waist_rgbd_depth_camera_info(
            waist_rgbd_depth_camera_info_callback
        )
        self.subscriptions["waist_rgbd_depth_camera_info"] = True
        logging.info("✓ Waist RGBD depth camera info subscribed")

# === Trinocular Camera Subscribe/Unsubscribe Methods ===
def toggle_trinocular_image_subscription(self):
    """Toggle trinocular camera image subscription"""
    if self.subscriptions["trinocular_image"]:
        self.sensor_controller.unsubscribe_trinocular_image()
        self.subscriptions["trinocular_image"] = False
        logging.info("✓ Trinocular camera image unsubscribed")
    else:

        def trinocular_image_callback(frame):
            global trinocular_image_counter
            trinocular_image_counter += 1
            if trinocular_image_counter % 15 == 0:
                logging.info(
                    f"Trinocular Camera Frame received (count: {trinocular_image_"
                    f"counter})"
                )

            self.sensor_controller.subscribe_trinocular_image(trinocular_image_
callback)
            self.subscriptions["trinocular_image"] = True
            logging.info("✓ Trinocular camera image subscribed")

```

(continues on next page)

(continued from previous page)

```

def show_status(self):
    """Display current sensor status"""
    logging.info("\n" + "=" * 80)
    logging.info("MAGICBOT GEN1 SENSOR STATUS")
    logging.info("=" * 80)
    logging.info(
        f"LiDAR:           {'OPEN' if self.sensors_state['lidar'] else 'CLOSED'}"
    )
    logging.info(
        f"Head RGBD Camera:      {'OPEN' if self.sensors_state['head_rgbd_camera'] else 'CLOSED'}"
    )
    logging.info(
        f"Waist RGBD Camera:     {'OPEN' if self.sensors_state['waist_rgbd_camera'] else 'CLOSED'}"
    )
    logging.info(
        f"Trinocular Camera:    {'OPEN' if self.sensors_state['trinocular_camera'] else 'CLOSED'}"
    )
    logging.info("\nLIDAR SUBSCRIPTIONS:")
    logging.info(
        f"  LiDAR IMU:          {'✓ SUBSCRIBED' if self.subscriptions['lidar_imu'] else 'UNSUBSCRIBED'}"
    )
    logging.info(
        f"  LiDAR Point Cloud:   {'✓ SUBSCRIBED' if self.subscriptions['lidar_point_cloud'] else 'UNSUBSCRIBED'}"
    )
    logging.info("\nHEAD RGBD SUBSCRIPTIONS:")
    logging.info(
        f"  Color Image:         {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_color_image'] else 'UNSUBSCRIBED'}"
    )
    logging.info(
        f"  Color Camera Info:   {'✓ SUBSCRIBED' if self.subscriptions['head_rgbd_color_camera_info'] else 'UNSUBSCRIBED'}"
    )

```

(continues on next page)

(continued from previous page)

```

        f" Depth Image:           {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'head_rgbd_depth_image'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info(
        f" Depth Camera Info:      {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'head_rgbd_depth_camera_info'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info("\nWAIST RGBD SUBSCRIPTIONS:")
    logging.info(
        f" Color Image:            {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'waist_rgbd_color_image'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info(
        f" Color Camera Info:      {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'waist_rgbd_color_camera_info'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info(
        f" Depth Image:             {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'waist_rgbd_depth_image'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info(
        f" Depth Camera Info:       {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'waist_rgbd_depth_camera_info'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info("\nTRINOCULAR CAMERA SUBSCRIPTIONS:")
    logging.info(
        f" Trinocular Image:        {'✓ SUBSCRIBED' if self.subscriptions[
    ↪'trinocular_image'] else '✗ UNSUBSCRIBED'}"
    )
    logging.info("=" * 80 + "\n")

```

```

def print_menu():
    """Print interactive menu"""
    logging.info("\n" + "=" * 80)
    logging.info("MAGICBOT GEN1 SENSOR CONTROL MENU")
    logging.info("=" * 80)
    logging.info("Sensor Open/Close:")
    logging.info("  1 - Open LiDAR           2 - Close LiDAR")
    logging.info("  3 - Open Head RGBD Camera 4 - Close Head RGBD Camera")

```

(continues on next page)

(continued from previous page)

```

logging.info(" 5 - Open Waist RGBD Camera      6 - Close Waist RGBD Camera")
logging.info(" 7 - Open Trinocular Camera       8 - Close Trinocular Camera")
logging.info("\nLiDAR Subscriptions:")
logging.info(" i - Toggle LiDAR IMU           p - Toggle LiDAR Point Cloud")
logging.info("\nHead RGBD Camera Subscriptions:")
logging.info(
    " c - Toggle Head Color Image     C - Toggle Head Color Camera Info"
)
logging.info(
    " d - Toggle Head Depth Image    D - Toggle Head Depth Camera Info"
)
logging.info("\nWaist RGBD Camera Subscriptions:")
logging.info(
    " w - Toggle Waist Color Image   W - Toggle Waist Color Camera Info"
)
logging.info(
    " e - Toggle Waist Depth Image  E - Toggle Waist Depth Camera Info"
)
logging.info("\nTrinocular Camera Subscriptions:")
logging.info(" t - Toggle Trinocular Image")
logging.info("\nCommands:")
logging.info(
    " s - Show Status                ESC - Quit                  ? - Help"
)
logging.info("= " * 80)

```

```

def main():
    """Main function"""
    logging.info("\n" + "=" * 80)
    logging.info("MagicBot Gen1 SDK Sensor Interactive Example")
    logging.info("= " * 80 + "\n")

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

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

```

(continues on next page)

(continued from previous page)

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

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

sensor_controller = robot.get_sensor_controller()

# Initialize sensor controller
if not sensor_controller.initialize():
    logging.error("Sensor controller initialization failed")
    robot.disconnect()
    robot.shutdown()
    return

logging.info("✓ Sensor controller initialized successfully\n")

sensor_manager = SensorManager(sensor_controller)

print_menu()

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

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

# Sensor open/close control
        if choice == "1":
            sensor_manager.open_lidar()
        elif choice == "2":
            sensor_manager.close_lidar()
        elif choice == "3":
            sensor_manager.open_head_rgbd_camera()
        elif choice == "4":
            sensor_manager.close_head_rgbd_camera()
```

(continues on next page)

(continued from previous page)

```

elif choice == "5":
    sensor_manager.open_waist_rgbd_camera()
elif choice == "6":
    sensor_manager.close_waist_rgbd_camera()
elif choice == "7":
    sensor_manager.open_trinocular_camera()
elif choice == "8":
    sensor_manager.close_trinocular_camera()

# LiDAR subscriptions
elif choice == "i":
    sensor_manager.toggle_lidar_imu_subscription()
elif choice == "p":
    sensor_manager.toggle_lidar_point_cloud_subscription()

# Head RGBD subscriptions
elif choice == "c":
    sensor_manager.toggle_head_rgbd_color_image_subscription()
elif choice == "C":
    sensor_manager.toggle_head_rgbd_color_camera_info_subscription()
elif choice == "d":
    sensor_manager.toggle_head_rgbd_depth_image_subscription()
elif choice == "D":
    sensor_manager.toggle_head_rgbd_depth_camera_info_subscription()

# Waist RGBD subscriptions
elif choice == "w":
    sensor_manager.toggle_waist_rgbd_color_image_subscription()
elif choice == "W":
    sensor_manager.toggle_waist_rgbd_color_camera_info_subscription()
elif choice == "e":
    sensor_manager.toggle_waist_rgbd_depth_image_subscription()
elif choice == "E":
    sensor_manager.toggle_waist_rgbd_depth_camera_info_subscription()

# Trinocular camera subscriptions
elif choice == "t":
    sensor_manager.toggle_trinocular_image_subscription()

# Commands

```

(continues on next page)

(continued from previous page)

```
        elif choice.lower() == "s":
            sensor_manager.show_status()
        elif choice == "?" or choice.lower() == "help":
            print_menu()
        else:
            logging.warning(f"Invalid choice: '{choice}'. Press '?' for help.")

    except KeyboardInterrupt:
        logging.info("\nReceived keyboard interrupt, shutting down...")
    except Exception as e:
        logging.error(f"Error occurred: {e}")
        import traceback

        traceback.print_exc()
    finally:
        # Cleanup: unsubscribe all and close all sensors
        logging.info("Cleaning up...")

        # Close all sensors
        if sensor_manager.sensors_state["lidar"]:
            sensor_manager.close_lidar()
        if sensor_manager.sensors_state["head_rgbd_camera"]:
            sensor_manager.close_head_rgbd_camera()
        if sensor_manager.sensors_state["waist_rgbd_camera"]:
            sensor_manager.close_waist_rgbd_camera()
        if sensor_manager.sensors_state["trinocular_camera"]:
            sensor_manager.close_trinocular_camera()

        # Allow time for cleanup
        time.sleep(1)

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

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

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

(continues on next page)

(continued from previous page)

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

## 23.3 Running Instructions

### 23.3.1 Environment Setup

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

### 23.3.2 Run Examples

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

### 23.3.3 控制说明

- Sensor Control:
  - 1 - Open LiDAR
  - 2 - Close LiDAR
  - 3 - Open Head RGBD Camera
  - 4 - Close Head RGBD Camera
  - 5 - Open Waist RGBD Camera
  - 6 - Close Waist RGBD Camera
  - 7 - Open Trinocular Camera
  - 8 - Close Trinocular Camera
- LiDAR Data Subscriptions:
  - i - Toggle IMU Data Subscription
  - p - Toggle Point Cloud Subscription
- Head RGBD Camera Subscriptions:

- `c` - Toggle Color Image Subscription
- `C` - Toggle Color Camera Info Subscription
- `d` - Toggle Depth Image Subscription
- `D` - Toggle Depth Camera Info Subscription
- Waist RGBD Camera Subscriptions:
  - `w` - Toggle Color Image Subscription
  - `W` - Toggle Color Camera Info Subscription
  - `e` - Toggle Depth Image Subscription
  - `E` - Toggle Depth Camera Info Subscription
- Trinocular Camera Subscriptions:
  - `t` - Toggle Image Subscription
- Other Commands:
  - `s` - Show Current Status
  - `?` - Show Help Menu

#### **23.3.4 Stop Program**

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



# CHAPTER 24

---

## Audio Control Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for initialization, robot connection, audio control (get volume, set volume, TTS playback, audio stream subscription) and other basic operations.

### 24.1 C++

Example file: `audio_example.cpp`

Reference documentation: `audio_reference.md`

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

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

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
```

(continues on next page)

(continued from previous page)

```

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

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

void print_help() {
    std::cout << "Key Function Description:\n";
    std::cout << "  Audio Functions:\n";
    std::cout << "    1      Function 1: Get volume\n";
    std::cout << "    2      Function 2: Set volume\n";
    std::cout << "    3      Function 3: Play TTS\n";
    std::cout << "    4      Function 4: Stop playback\n";
    std::cout << "  Audio stream Functions:\n";
    std::cout << "    5      Function 5: Open audio stream\n";
    std::cout << "    6      Function 6: Close audio stream\n";
    std::cout << "    7      Function 7: Subscribe to audio stream\n";
    std::cout << "    8      Function 8: Unsubscribe to audio stream\n";
    std::cout << "  Wakeup Status Functions:\n";
    std::cout << "    q      Function q: Open wakeup status stream\n";
    std::cout << "    w      Function w: Close wakeup status stream\n";
    std::cout << "    e      Function e: Subscribe to wakeup status\n";
    std::cout << "    r      Function r: Unsubscribe to wakeup status\n";
    std::cout << "\n";
    std::cout << "  ?      Function ?: Print help\n";
    std::cout << "  ESC     Exit program\n";
}

int getch() {
    struct termios oldt, newt;
    int ch;

    tcgetattr(STDIN_FILENO, &oldt); // Get current terminal settings
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO); // Disable buffering and echo
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);

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

```

(continues on next page)

(continued from previous page)

```

}

void GetVolume() {
    // Get audio controller
    auto& controller = robot.GetAudioController();

    // Get volume
    int get_volume = 0;
    auto status = controller.GetVolume(get_volume);
    if (status.code != ErrorCode::OK) {
        std::cerr << "get volume failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "get volume success, volume: " << std::to_string(get_volume) << std::endl;
}

void SetVolume() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Set volume
    auto status = controller.SetVolume(50);
    if (status.code != ErrorCode::OK) {
        std::cerr << "set volume failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "set volume success" << std::endl;
}

void PlayTts() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Play speech
    TtsCommand tts;
    tts.id = "100000000001";
    tts.content = "How's the weather today!";
}

```

(continues on next page)

(continued from previous page)

```
tts.priority = TtsPriority::HIGH;
tts.mode = TtsMode::CLEARTOP;
auto status = controller.Play(tts);
if (status.code != ErrorCode::OK) {
    std::cerr << "play tts failed"
        << ", code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}
std::cout << "play tts success" << std::endl;
}

void StopTts() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Stop speech playback
    auto status = controller.Stop();
    if (status.code != ErrorCode::OK) {
        std::cerr << "stop tts failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
    return;
}
std::cout << "stop tts success" << std::endl;
}

void OpenAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Open audio stream
    auto status = controller.OpenAudioStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "open audio stream failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
    return;
}
std::cout << "open audio stream success" << std::endl;
}
```

(continues on next page)

(continued from previous page)

```

void CloseAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Close audio stream
    auto status = controller.CloseAudioStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "close audio stream failed"
            << ", code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "close audio stream success" << std::endl;
}

void SubscribeAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();

    // Subscribe to audio streams
    controller.SubscribeOriginAudioStream([](const std::shared_ptr<AudioStream> data) {
        static int32_t counter = 0;
        if (counter++ % 30 == 0) {
            std::cout << "Received origin audio stream data, size: " << data->data_length <
            << std::endl;
        }
    });
}

controller.SubscribeBfAudioStream([](const std::shared_ptr<AudioStream> data) {
    static int32_t counter = 0;
    if (counter++ % 30 == 0) {
        std::cout << "Received bf audio stream data, size: " << data->data_length <<
        std::endl;
    }
});
std::cout << "Subscribed to audio streams" << std::endl;
}

void UnsubscribeAudioStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
}

```

(continues on next page)

(continued from previous page)

```

// Unsubscribe from audio streams
controller.UnsubscribeOriginAudioStream();
controller.UnsubscribeBfAudioStream();
std::cout << "Unsubscribed from audio streams" << std::endl;
}

void OpenWakeupStatusStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Open wakeup status stream
    auto status = controller.OpenWakeupStatusStream();
}

void CloseWakeupStatusStream() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Close wakeup status stream
    auto status = controller.CloseWakeupStatusStream();
}

void SubscribeWakeupStatus() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Subscribe to wakeup status
    controller.SubscribeWakeupStatus([] (const std::shared_ptr<WakeupStatus> data) {
        static int32_t counter = 0;
        if (data->is_wakeup) {
            if (data->enable_wakeup_orientation) {
                std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup <<
                ↪, enable_wakeup_orientation: " << data->enable_wakeup_orientation << ", wakeup_
                ↪orientation: " << data->wakeup_orientation << std::endl;
            } else {
                std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup <<
                ↪std::endl;
            }
        } else {
            std::cout << "Received wakeup status data, is_wakeup: " << data->is_wakeup <<
            ↪std::endl;
        }
    });
}

```

(continues on next page)

(continued from previous page)

```
}
```

```
void UnsubscribeWakeupStatus() {
    // Get audio controller
    auto& controller = robot.GetAudioController();
    // Unsubscribe from wakeup status
    controller.UnsubscribeWakeupStatus();
}
```

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

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

    print_help();

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

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

    // Wait for user input
    while (1) {
        int key = getch();
```

(continues on next page)

(continued from previous page)

```
if (key == 27)
    break; // ESC key ASCII code is 27

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

switch (key) {
    // 1. Audio Functions
    case '1': {
        // Get volume
        GetVolume();
        break;
    }
    case '2': {
        // Set volume
        SetVolume();
        break;
    }
    case '3': {
        PlayTts();
        break;
    }
    case '4': {
        StopTts();
        break;
    }
    // 2. Audio stream Functions
    case '5': {
        OpenAudioStream();
        break;
    }
    case '6': {
        CloseAudioStream();
        break;
    }
    case '7': {
        SubscribeAudioStream();
        break;
    }
    case '8': {
        UnsubscribeAudioStream();
    }
}
```

(continues on next page)

(continued from previous page)

```

break;
}

// 3. Wakeup Status Functions

case 'Q':
case 'q': {
    OpenWakeupStatusStream();
    break;
}

case 'W':
case 'w': {
    CloseWakeupStatusStream();
    break;
}

case 'E':
case 'e': {
    SubscribeWakeupStatus();
    break;
}

case 'R':
case 'r': {
    UnsubscribeWakeupStatus();
    break;
}

case '?': {
    print_help();
    break;
}

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

usleep(10000);
}

// Disconnect from robot

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

```

(continues on next page)

(continued from previous page)

```
    robot.Shutdown();  
    return -1;  
}  
  
robot.Shutdown();  
  
return 0;  
}
```

## 24.2 Python

Example file: audio\_example.py

Reference documentation: audio\_reference.md

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
import sys  
import time  
import signal  
import threading  
import logging  
from typing import Optional  
  
import magicbot_gen1_python as magicbot  
  
# Configure logging format and level  
logging.basicConfig(  
    level=logging.INFO, # Minimum log level  
    format="%(asctime)s [%(levelname)s] %(message)s",  
    datefmt="%Y-%m-%d %H:%M:%S",  
)  
  
# Global variables  
robot: Optional[magicbot.MagicRobot] = None  
running = True  
  
def signal_handler(signum, frame):
```

(continues on next page)

(continued from previous page)

```

"""Signal handler function for graceful exit"""

global running, robot
logging.info("Received interrupt signal (%s), exiting...", signum)
running = False
if robot:
    robot.shutdown()
    logging.info("Robot shutdown")
exit(-1)

def print_help():
    """Print help information"""
    logging.info("Key Function Demo Program")
    logging.info("")
    logging.info("Audio Functions:")
    logging.info("  1      Function 1: Get volume")
    logging.info("  2      Function 2: Set volume")
    logging.info("  3      Function 3: Play TTS")
    logging.info("  4      Function 4: Stop playback")
    logging.info("")
    logging.info("Audio stream Functions:")
    logging.info("  5      Function 5: Open audio stream")
    logging.info("  6      Function 6: Close audio stream")
    logging.info("  7      Function 7: Subscribe to audio stream")
    logging.info("  8      Function 8: Unsubscribe to audio stream")
    logging.info("")
    logging.info("Wakeup Status Functions:")
    logging.info("  Q      Function Q: Open wakeup status stream")
    logging.info("  W      Function W: Close wakeup status stream")
    logging.info("  E      Function E: Subscribe to wakeup status")
    logging.info("  R      Function R: Unsubscribe to wakeup status")
    logging.info("")
    logging.info("  ?: Print help")
    logging.info("  ESC   Exit program")

def get_volume():
    """Get volume"""
    global robot
    try:

```

(continues on next page)

(continued from previous page)

```
# Get audio controller
controller = robot.get_audio_controller()

# Get volume
status, volume = controller.get_volume()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to get volume, code: %s, message: %s",
        status.code,
        status.message,
    )
return

logging.info("Successfully got volume, volume: %s", volume)
except Exception as e:
    logging.error("Exception occurred while getting volume: %s", e)

def set_volume(volume):
    """Set volume"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Set volume to 7
        status = controller.set_volume(volume)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to set volume, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully set volume")
    except Exception as e:
        logging.error("Exception occurred while setting volume: %s", e)
```

(continues on next page)

(continued from previous page)

```

def play_tts(content):
    """Play TTS speech"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Create TTS command
        tts = magicbot.TtsCommand()
        tts.id = "100000000001"
        tts.content = content
        tts.priority = magicbot.TtsPriority.HIGH
        tts.mode = magicbot.TtsMode.CLEARTOP

        # Play speech
        status = controller.play(tts)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to play TTS, code: %s, message: %s", status.code, status.
                message
            )
        return

        logging.info("Successfully played TTS")
    except Exception as e:
        logging.error("Exception occurred while playing TTS: %s", e)

def stop_tts():
    """Stop TTS playback"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Stop speech playback
        status = controller.stop()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to stop TTS, code: %s, message: %s", status.code, status.
            message
        )

```

(continues on next page)

(continued from previous page)

```
    ↵message
    )
    return

    logging.info("Successfully stopped TTS")
except Exception as e:
    logging.error("Exception occurred while stopping TTS: %s", e)

def open_audio_stream():
    """Open audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Open audio stream
        status = controller.open_audio_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to open audio stream, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    logging.info("Successfully opened audio stream")
except Exception as e:
    logging.error("Exception occurred while opening audio stream: %s", e)

def close_audio_stream():
    """Close audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Close audio stream
        status = controller.close_audio_stream()
```

(continues on next page)

(continued from previous page)

```
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to close audio stream, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully closed audio stream")
except Exception as e:
    logging.error("Exception occurred while closing audio stream: %s", e)

def subscribe_audio_stream():
    """Subscribe to audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Audio stream counters
        origin_counter = 0
        bf_counter = 0

        def origin_audio_callback(audio_stream):
            """Original audio stream callback function"""
            nonlocal origin_counter
            if origin_counter % 30 == 0:
                logging.info(
                    "Received original audio stream data, size: %d",
                    audio_stream.data_length,
                )
                sys.stdout.write("\r")
                sys.stdout.flush()
            origin_counter += 1

        def bf_audio_callback(audio_stream):
            """BF audio stream callback function"""
            nonlocal bf_counter
            if bf_counter % 30 == 0:
```

(continues on next page)

(continued from previous page)

```
        logging.info(
            "Received BF audio stream data, size: %d", audio_stream.data_
→length
        )
        sys.stdout.write("\r")
        sys.stdout.flush()
        bf_counter += 1

    # Subscribe to audio streams
    controller.subscribe_origin_audio_stream(origin_audio_callback)
    controller.subscribe_bf_audio_stream(bf_audio_callback)

    logging.info("Subscribed to audio streams")
except Exception as e:
    logging.error("Exception occurred while subscribing to audio stream: %s", e)

def unsubscribe_audio_stream():
    """Unsubscribe to audio stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Unsubscribe to audio stream
        controller.unsubscribe_bf_audio_stream()
        controller.unsubscribe_origin_audio_stream()

        logging.info("Unsubscribed to audio stream")
    except Exception as e:
        logging.error("Exception occurred while unsubscribing to audio stream: %s", e)

def open_wakeup_status_stream():
    """Open wakeup status stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()
```

(continues on next page)

(continued from previous page)

```
# Open wakeup status stream
status = controller.open_wakeup_status_stream()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to open wakeup status stream, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully opened wakeup status stream")
except Exception as e:
    logging.error("Exception occurred while opening wakeup status stream: %s", e)

def close_wakeup_status_stream():
    """Close wakeup status stream"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Close wakeup status stream
        status = controller.close_wakeup_status_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close wakeup status stream, code: %s, message: %s",
                status.code,
                status.message,
            )
    except Exception as e:
        logging.error("Exception occurred while closing wakeup status stream: %s", e)

    logging.info("Successfully closed wakeup status stream")
    return

def unsubscribe_wakeup_status():
    """Unsubscribe to wakeup status"""
    global robot
```

(continues on next page)

(continued from previous page)

```

try:
    # Get audio controller
    controller = robot.get_audio_controller()

    # Unsubscribe to wakeup status
    controller.unsubscribe_wakeup_status()

    logging.info("Unsubscribed to wakeup status")

except Exception as e:
    logging.error("Exception occurred while unsubscribing to wakeup status: %s", e)

```

```

def subscribe_wakeup_status():
    """Subscribe to wakeup status"""
    global robot
    try:
        # Get audio controller
        controller = robot.get_audio_controller()

        # Wakeup status counter
        wakeup_counter = 0

        def wakeup_status_callback(wakeup_status):
            """Wakeup status callback function"""
            nonlocal wakeup_counter
            if wakeup_status.is_wakeup:
                if wakeup_status.enable_wakeup_orientation:
                    logging.info(
                        "Voice wakeup detected! Orientation: %.2f radians (%.1f",
                        wakeup_status.wakeup_orientation,
                        wakeup_status.wakeup_orientation * 180.0 / 3.14159,
                    )
            else:
                logging.info("Voice wakeup detected!")
                sys.stdout.write("\r")
                sys.stdout.flush()
    else:
        if wakeup_counter % 10 == 0: # Log every 50th non-wakeup status

```

(continues on next page)

(continued from previous page)

```

        logging.info(
            "Wakeup status: sleeping, orientation: %.2f radians",
            wakeup_status.wakeup_orientation,
        )
        sys.stdout.write("\r")
        sys.stdout.flush()
    wakeup_counter += 1

    # Subscribe to wakeup status
    controller.subscribe_wakeup_status(wakeup_status_callback)

    logging.info("Subscribed to wakeup status stream")
except Exception as e:
    logging.error("Exception occurred while subscribing to wakeup status: %s", e)

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

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

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

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

    # Create robot instance
    robot = magicbot.MagicRobot()

    try:
        # Configure local IP address for direct network connection and initialize SDK
        local_ip = "192.168.54.111"

```

(continues on next page)

(continued from previous page)

```
if not robot.initialize(local_ip):
    logging.error("Failed to initialize robot SDK")
    robot.shutdown()
    return -1

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

logging.info("Successfully connected to robot")

# Initialize audio controller
audio_controller = robot.get_audio_controller()
if not audio_controller.initialize():
    logging.error("Failed to initialize audio controller")
    robot.disconnect()
    robot.shutdown()
    return -1

logging.info("Successfully initialized audio controller")
print_help()
logging.info("Press any key to continue (ESC to exit)...")


# Main loop
while running:
    try:
        str_input = get_user_input()

        # Split input parameters by space
parts = str_input.strip().split()

        if not parts:
            time.sleep(0.01) # Brief delay
```

(continues on next page)

(continued from previous page)

```
continue

# Parse parameters
key = parts[0]
args = parts[1:] if len(parts) > 1 else []
if key == "\x1b": # ESC key
    break
# 1. Audio Functions
# 1.1 Get volume
if key == "1":
    get_volume()
# 1.2 Set volume
elif key == "2":
    volume = args[0] if args else 50
    set_volume(volume)
# 1.3 Play TTS
elif key == "3":
    content = args[0] if args else "How's the weather today!"
    play_tts(content)
# 1.4 Stop TTS
elif key == "4":
    stop_tts()
# 2. Audio Stream Functions
# 2.1 Open audio stream
elif key == "5":
    open_audio_stream()
# 2.2 Close audio stream
elif key == "6":
    close_audio_stream()
# 2.3 Subscribe to audio stream
elif key == "7":
    subscribe_audio_stream()
# 2.4 Unsubscribe from audio stream
elif key.upper() == "8":
    unsubscribe_audio_stream()
# 3. Wakeup Status Functions
# 3.1 Open wakeup status stream
elif key.upper() == "Q":
    open_wakeup_status_stream()
# 3.2 Close wakeup status stream
```

(continues on next page)

(continued from previous page)

```

    elif key.upper() == "W":
        close_wakeup_status_stream()
    # 3.3 Subscribe to wakeup status stream
    elif key.upper() == "E":
        subscribe_wakeup_status()
    # 3.4 Unsubscribe from wakeup status stream
    elif key.upper() == "R":
        unsubscribe_wakeup_status()
    # 4. Print help information
    elif key.upper() == "?":
        print_help()
    else:
        logging.warning("Unknown key: %s", key)

        time.sleep(0.01) # Brief delay

    except KeyboardInterrupt:
        break
    except Exception as e:
        logging.error("Exception occurred while processing user input: %s", e)
    except Exception as e:
        logging.error("Exception occurred during program execution: %s", e)
    return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close sensor controller
        audio_controller = robot.get_audio_controller()
        audio_controller.shutdown()
        logging.info("Audio controller closed")

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

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

```

(continues on next page)

(continued from previous page)

```
except Exception as e:  
    logging.error("Exception occurred while cleaning up resources: %s", e)  
  
if __name__ == "__main__":  
    sys.exit(main())
```

## 24.3 Running Instructions

### 24.3.1 Environment Setup

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

### 24.3.2 Run Examples

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

### 24.3.3 Control Instructions

- 1: Get current volume
- 2: Set volume
- 3: Play TTS audio
- 4: Stop TTS playback
- 5: Open audio stream
- 6: Close audio stream
- 7: Subscribe to audio stream data
- 8: Unsubscribe from audio stream data
- Q: Open voice wake-up status stream
- W: Close voice wake-up status stream
- E: Subscribe to voice wake-up status

- R: Unsubscribe from voice wake-up status
- ?: Print help information

#### 24.3.4 Stop Program

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

# CHAPTER 25

---

## State Monitor Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for initialization, robot connection, robot status monitoring (faults, BMS) and other basic operations.

### 25.1 C++

Example file: monitor\_example.cpp

Reference documentation: monitor\_reference.md

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

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

#include <iostream>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;

void signalHandler(int signum) {
```

(continues on next page)

(continued from previous page)

```
    std::cout << "Interrupt signal (" << signum << ") received.\n";\n\n    robot.Shutdown();\n    // Exit process\n    exit(signum);\n}\n\nint main() {\n    // Bind SIGINT (Ctrl+C)\n    signal(SIGINT, signalHandler);\n\n    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;\n\n    std::string local_ip = "192.168.54.111";\n    // Configure local IP address for direct network connection and initialize SDK\n    if (!robot.Initialize(local_ip)) {\n        std::cerr << "robot sdk initialize failed." << std::endl;\n        robot.Shutdown();\n        return -1;\n    }\n\n    // Connect to robot\n    auto status = robot.Connect();\n    if (status.code != ErrorCode::OK) {\n        std::cerr << "connect robot failed"\n            << ", code: " << status.code\n            << ", message: " << status.message << std::endl;\n        robot.Shutdown();\n        return -1;\n    }\n\n    // Wait 5s\n    usleep(5000000);\n\n    auto& monitor = robot.GetStateMonitor();\n\n    RobotState state;\n    status = monitor.GetCurrentState(state);\n    if (status.code != ErrorCode::OK) {\n        std::cerr << "get current state failed"\n            << ", code: " << status.code\n    }\n}
```

(continues on next page)

(continued from previous page)

```

        << ", message: " << status.message << std::endl;
    robot.Shutdown();
    return -1;
}

std::cout << "health: " << state.bms_data.battery_health
        << ", percentage: " << state.bms_data.battery_percentage
        << ", state: " << std::to_string((int8_t)state.bms_data.battery_state)
        << ", power_supply_status: " << std::to_string((int8_t)state.bms_data.
→power_supply_status)
        << std::endl;

auto& faults = state.faults;
for (auto& [code, msg] : faults) {
    std::cout << "code: " << std::to_string(code)
        << ", message: " << msg;
}

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

robot.Shutdown();

return 0;
}

```

## 25.2 Python

Example file: monitor\_example.py

Reference documentation: monitor\_reference.md

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

(continues on next page)

(continued from previous page)

```
# -*- coding: utf-8 -*-

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

import magicbot_gen1_python as magicbot

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
# Create robot instance
robot = magicbot.MagicRobot()

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

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

    logging.info("Successfully connected to robot")

    # Wait 5 seconds
    logging.info("Waiting 5 seconds...")
    time.sleep(5)

    # Get state monitor
    monitor = robot.get_state_monitor()

    # Get current state
    status, state = monitor.get_current_state()
    if status.code != magicbot.ErrorCode.OK:
        logging.error(
            "Failed to get current state, code: %s, message: %s",
            status.code,
            status.message,
        )

```

(continues on next page)

(continued from previous page)

```
    return -1

    # Print battery information
    logging.info("Battery health: %s", state.bms_data.battery_health)
    logging.info("Battery percentage: %s%%", state.bms_data.battery_percentage)
    logging.info("Battery state: %s", state.bms_data.battery_state)
    logging.info("Power supply status: %s", state.bms_data.power_supply_status)

    # Print fault information
    faults = state.faults
    for fault in faults:
        logging.info(
            "Error code: %s, Error message: %s",
            fault.error_code,
            fault.error_message,
        )

    return 0

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

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close state monitor
        monitor = robot.get_state_monitor()
        monitor.shutdown()
        # Disconnect
        robot.disconnect()
        logging.info("Robot connection disconnected")

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

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

(continues on next page)

(continued from previous page)

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

## 25.3 Running Instructions

### 25.3.1 Environment Setup

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

### 25.3.2 Run Examples

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

### 25.3.3 Stop Program

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



# CHAPTER 26

---

## SLAM Navigation Example

---

---

This example demonstrates how to use the Magicbot-Gen1 SDK for SLAM mapping, localization, navigation and other operations, including map management, pose acquisition and other functions.

### 26.1 C++

Example files:

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

Reference documentation:

- C++ API/`slam_navigation_reference.md`

#### 26.1.1 Mapping Example

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

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

(continues on next page)

(continued from previous page)

```

#include <fstream>
#include <iostream>
#include <string>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
bool running = true;
SlamMode current_slam_mode = SlamMode::IDLE;

void SignalHandler(int signum) {
    std::cout << "Received interrupt signal (" << signum << "), exiting..." <<
    std::endl;
    running = false;
    robotShutdown();
    exit(signum);
}

void PrintHelp() {
    std::cout << "\n=====" << std::endl;
    std::cout << "SLAM and Navigation Function Demo Program" << std::endl;
    std::cout << "=====" << std::endl;
    std::cout << "preparation Functions:" << std::endl;
    std::cout << " q      Function q: Recovery stand" << std::endl;
    std::cout << " e      Function e: Balance stand" << std::endl;
    std::cout << " w      Function w: Move forward" << std::endl;
    std::cout << " a      Function a: Move left" << std::endl;
    std::cout << " s      Function s: Move backward" << std::endl;
    std::cout << " d      Function d: Move right" << std::endl;
    std::cout << " t      Function t: Turn left" << std::endl;
    std::cout << " g      Function g: Turn right" << std::endl;
    std::cout << " x      Function x: Stop" << std::endl;
    std::cout << "SLAM Functions:" << std::endl;
    std::cout << " 1      Function 1: Switch to mapping mode" << std::endl;
    std::cout << " 2      Function 2: Start mapping" << std::endl;
    std::cout << " 3      Function 3: Cancel mapping" << std::endl;
    std::cout << " 4      Function 4: Save map" << std::endl;
    std::cout << " 5      Function 5: Load map (input map name after pressing 5)" <<
    std::endl;
    std::cout << " 6      Function 6: Delete map (input map name after pressing 6)" <<
}

```

(continues on next page)

(continued from previous page)

```

→< std::endl;
    std::cout << " 7      Function 7: Get all map information and save map image as"
→PGM file" << std::endl;
    std::cout << " 8      Function 8: Get map path (input map name)" << std::endl;
    std::cout << " 9      Function 9: Get SLAM mapping point cloud map" << std::endl;
    std::cout << "Close Functions:" << std::endl;
    std::cout << " P      Function P: Close SLAM" << std::endl;
    std::cout << "\n ?      Function ?: Print help" << std::endl;
    std::cout << " ESC     Exit program" << std::endl;
    std::cout << "=====\\n"
                << std::endl;
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
}

std::string GetUserInput(const std::string& prompt) {
    std::cout << prompt;
    std::string input;
    std::getline(std::cin, input);
    return input;
}

void RecoveryStand() {
    std::cout << "==== Executing Recovery Stand ===" << std::endl;
    auto& controller = robot.GetHighLevelMotionController();

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

```

(continues on next page)

(continued from previous page)

```

    return;
}

std::cout << "Successfully executed recovery stand" << std::endl;
}

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

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

void SwitchToMappingMode() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::MAPPING, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to mapping mode, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::MAPPING;
    std::cout << "Successfully switched to mapping mode" << std::endl;
    std::cout << "Robot is now in mapping mode, ready to create new maps" << std::endl;
}

void StartMapping() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.StartMapping(10000);
    if (status.code != ErrorCode::OK) {
}

```

(continues on next page)

(continued from previous page)

```

    std::cerr << "Failed to start mapping, code: " << status.code
                << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully started mapping" << std::endl;
}

void CancelMapping() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CancelMapping(10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to cancel mapping, code: " << status.code
                    << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully cancelled mapping" << std::endl;
}

void SaveMap() {
    auto& controller = robot.GetSlamNavController();

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

    // Generate map name with timestamp
    std::time_t now = std::time(nullptr);
    std::string map_name = "map_" + std::to_string(now);
    std::cout << "Saving map: " << map_name << std::endl;

    auto status = controller.SaveMap(map_name, 20000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to save map, code: " << status.code
                    << ", message: " << status.message << std::endl;
        return;
    }
}

```

(continues on next page)

(continued from previous page)

```
    std::cout << "Successfully saved map: " << map_name << std::endl;
}

void LoadMap(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to load is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::cout << "Loading map: " << map_name << std::endl;
    auto status = controller.LoadMap(map_name, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to load map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully loaded map: " << map_name << std::endl;
}

void DeleteMap(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to delete is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::cout << "Deleting map: " << map_name << std::endl;
    auto status = controller.DeleteMap(map_name, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to delete map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully deleted map: " << map_name << std::endl;
```

(continues on next page)

(continued from previous page)

}

```

void SaveMapImageToFile(const MapInfo& map_info) {
    try {
        const auto& map_data = map_info.map_meta_data.map_image_data;
        uint32_t width = map_data.width;
        uint32_t height = map_data.height;
        uint32_t max_gray_value = map_data.max_gray_value;
        const auto& image_bytes = map_data.image;

        std::cout << "Saving map image: " << width << "x" << height
            << ", max_gray: " << max_gray_value << std::endl;

        // Check image data size
        if (image_bytes.size() != width * height) {
            std::cerr << "Image data size mismatch: expected " << (width * height)
                << ", got " << image_bytes.size() << std::endl;
            return;
        }

        // Generate filename based on map name
        std::string safe_filename = map_info.map_name;
        // Remove invalid characters
        safe_filename.erase(
            std::remove_if(safe_filename.begin(), safe_filename.end(),
                [](char c) { return !std::isalnum(c) && c != '_' && c != '-'; }
            ),
            safe_filename.end());
    }

    if (safe_filename.empty()) {
        safe_filename = "map_" + std::to_string(std::time(nullptr));
    }

    // Save as PGM format
    std::string pgm_filename = "build/" + safe_filename + ".pgm";
    std::ofstream pgm_file(pgm_filename, std::ios::binary);
    if (!pgm_file.is_open()) {
        std::cerr << "Failed to open file for writing: " << pgm_filename << std::endl;
        return;
    }
}

```

(continues on next page)

(continued from previous page)

```

// Write PGM header
pgm_file << "P5\n"
    << width << " " << height << "\n"
    << max_gray_value << "\n";

// Write image data
pgm_file.write(reinterpret_cast<const char*>(image_bytes.data()), image_bytes.
→size());
pgm_file.close();

std::cout << "Map image saved successfully as PGM: " << pgm_filename << std::endl;

} catch (const std::exception& e) {
    std::cerr << "Exception occurred while saving map image: " << e.what() <<
→std::endl;
}

void GetAllMapInfo() {
    auto& controller = robot.GetSlamNavController();

    AllMapInfo all_map_info;
    auto status = controller.GetAllMapInfo(all_map_info, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get map information, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully retrieved map information" << std::endl;
    std::cout << "Current map: " << all_map_info.current_map_name << std::endl;
    std::cout << "Total maps: " << all_map_info.map_infos.size() << std::endl;

    if (!all_map_info.map_infos.empty()) {
        std::cout << "Map details:" << std::endl;
        for (size_t i = 0; i < all_map_info.map_infos.size(); ++i) {
            const auto& map_info = all_map_info.map_infos[i];
            std::cout << "  Map " << (i + 1) << ":" << map_info.map_name << std::endl;
            std::cout << "    Origin: [" << map_info.map_meta_data.origin.position[0] << ", "

```

(continues on next page)

(continued from previous page)

```

    <> "Orientation: [" << map_info.map_meta_data.origin.
    <> orientation[0] << ", "
        << map_info.map_meta_data.origin.orientation[1] << ", "
        << map_info.map_meta_data.origin.orientation[2] << "]"
        << std::endl;
    std::cout << "    Resolution: " << map_info.map_meta_data.resolution << " m/
    <> pixel" << std::endl;
    std::cout << "    Size: " << map_info.map_meta_data.map_image_data.width << " x
    <> "
        << map_info.map_meta_data.map_image_data.height << std::endl;
    std::cout << "    Max gray value: " << map_info.map_meta_data.map_image_data.
    <> max_gray_value << std::endl;
    std::cout << "    Image type: " << map_info.map_meta_data.map_image_data.type <
    <> std::endl;

    SaveMapImageToFile(map_info);
}

} else {
    std::cout << "No available maps" << std::endl;
}
}

void GetMapPath(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to get path is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::vector<std::string> map_path;
    auto status = controller.GetMapPath(map_name, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get map path, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
}

```

(continues on next page)

(continued from previous page)

```

if (map_path.empty()) {
    std::cerr << "No map path found" << std::endl;
    return;
}

for (const auto& path : map_path) {
    std::cout << "Map path: " << path << std::endl;
}
}

void GetPointCloudMap() {
    auto& controller = robot.GetSlamNavController();

    PointCloud2 point_cloud_map;
    auto status = controller.GetPointCloudMap(point_cloud_map, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get SLAM mapping point cloud map, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully got SLAM mapping point cloud map" << std::endl;
    std::cout << "Point cloud map - Height: " << point_cloud_map.height
        << ", Width: " << point_cloud_map.width << std::endl;
    std::cout << "Point cloud map data size: " << point_cloud_map.data.size() << " bytes"
    ↵" << std::endl;
}

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

    JoystickCommand joy_command;
    joy_command.left_x_axis = left_x_axis;
    joy_command.left_y_axis = left_y_axis;
    joy_command.right_x_axis = right_x_axis;
    joy_command.right_y_axis = right_y_axis;
}

```

(continues on next page)

(continued from previous page)

```

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

void CloseSlam() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::IDLE, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close SLAM, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::IDLE;
    std::cout << "Successfully closed SLAM system" << std::endl;
}

int main(int argc, char* argv[]) {
    // Bind signal handler
    signal(SIGINT, SignalHandler);

    std::cout << "\n=====\n" << std::endl;
    std::cout << "MagicBot Gen1 SDK SLAM Example" << std::endl;
    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;
    std::cout << "=====\n"
        << std::endl;

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

    // Configure local IP address for direct network connection and initialize SDK
}

```

(continues on next page)

(continued from previous page)

```

    std::string local_ip = "192.168.54.111";
    if (!robot.Initialize(local_ip)) {
        std::cerr << "Failed to initialize robot SDK" << std::endl;
        robotShutdown();
        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;
        robotShutdown();
        return -1;
    }
    std::cout << "Successfully connected to robot" << std::endl;

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

    // Initialize SLAM navigation controller
    auto& slam_nav_controller = robot.GetSlamNavController();
    if (!slam_nav_controller.Initialize()) {
        std::cerr << "Failed to initialize SLAM navigation controller" << std::endl;
        robotDisconnect();
        robotShutdown();
        return -1;
    }
    std::cout << "Successfully initialized SLAM navigation controller" << std::endl;

    // Main loop
    while (running) {
        int key = getch();

```

(continues on next page)

(continued from previous page)

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

switch (key) {
    // 1. Preparation Functions
    case 'q':
    case 'Q':
        RecoveryStand();
        break;
    case 'e':
    case 'E':
        BalanceStand();
        break;
    case 'w':
    case 'W':
        JoyStickCommand(0.0, 1.0, 0.0, 0.0); // Forward
        break;
    case 'a':
    case 'A':
        JoyStickCommand(-1.0, 0.0, 0.0, 0.0); // Left
        break;
    case 's':
    case 'S':
        JoyStickCommand(0.0, -1.0, 0.0, 0.0); // Backward
        break;
    case 'd':
    case 'D':
        JoyStickCommand(1.0, 0.0, 0.0, 0.0); // Right
        break;
    case 'x':
    case 'X':
        JoyStickCommand(0.0, 0.0, 0.0, 0.0); // Stop
        break;
}
```

(continues on next page)

(continued from previous page)

```

case 't':
case 'T': {
    JoyStickCommand(0.0, 0.0, -1.0, 0.0); // Turn left
    break;
}
case 'g':
case 'G': {
    JoyStickCommand(0.0, 0.0, 1.0, 0.0); // Turn right
    break;
}
// 2. SLAM Functions
case '1': {
    SwitchToMappingMode();
    break;
}
case '2': {
    StartMapping();
    break;
}
case '3': {
    CancelMapping();
    break;
}
case '4': {
    SaveMap();
    break;
}
case '5': {
    std::string map_name = GetUserInput("Enter map name to load: ");
    LoadMap(map_name);
    break;
}
case '6': {
    std::string map_name = GetUserInput("Enter map name to delete: ");
    DeleteMap(map_name);
    break;
}
case '7': {
    GetAllMapInfo();
    break;
}
case '8': {
    std::string map_name = GetUserInput("Enter map name to get path: ");
    GetMapPath(map_name);
    break;
}

```

(continues on next page)

(continued from previous page)

```

    }

    case '9':
        GetPointCloudMap();
        break;
    case 'p':
    case 'P':
        CloseSlam();
        break;
    case '?':
        PrintHelp();
        break;
default:
    std::cout << "Unknown key: " << static_cast<char>(key) << std::endl;
    break;
}

usleep(10000); // 10ms delay
}

// Cleanup resources
std::cout << "Clean up resources" << std::endl;

slam_nav_controller.Shutdown();
std::cout << "SLAM navigation controller closed" << std::endl;

robot.Disconnect();
std::cout << "Robot connection disconnected" << std::endl;

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

return 0;
}

```

## 26.1.2 Navigation Example

```

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

#include <termios.h>

```

(continues on next page)

(continued from previous page)

```
#include <unistd.h>
#include <atomic>
#include <csignal>
#include <ctime>
#include <iostream>
#include <sstream>
#include <string>

using namespace magic::gen1;

magic::gen1::MagicRobot robot;
bool running = true;
SlamMode current_slam_mode = SlamMode::IDLE;
NavMode current_nav_mode = NavMode::IDLE;
std::atomic<int> odometry_counter{0};

void SignalHandler(int signum) {
    std::cout << "Received interrupt signal (" << signum << "), exiting..." <<
    std::endl;
    running = false;
    robotShutdown();
    exit(signum);
}

void PrintHelp() {
    std::cout << "\n=====\n";
    std::cout << "SLAM and Navigation Function Demo Program" << std::endl;
    std::cout << "=====\n";
    std::cout << "\npreparation Functions:" << std::endl;
    std::cout << " Q      Function Q: Recovery stand" << std::endl;
    std::cout << " W      Function W: Balance stand" << std::endl;
    std::cout << " E      Function E: Get map path (input map name)" << std::endl;
    std::cout << "\nLocalization Functions:" << std::endl;
    std::cout << " 1      Function 1: Switch to localization mode (input map path)" <<
    std::endl;
    std::cout << " 2      Function 2: Initialize pose (input x y yaw)" << std::endl;
    std::cout << " 3      Function 3: Get current pose information" << std::endl;
    std::cout << "\nNavigation Functions:" << std::endl;
    std::cout << " 4      Function 4: Switch to navigation mode (input map path)" <<
    std::endl;
}
```

(continues on next page)

(continued from previous page)

```

    std::cout << " 5      Function 5: Set navigation target goal (input x y yaw)" <<
→std::endl;
    std::cout << " 6      Function 6: Pause navigation" << std::endl;
    std::cout << " 7      Function 7: Resume navigation" << std::endl;
    std::cout << " 8      Function 8: Cancel navigation" << std::endl;
    std::cout << " 9      Function 9: Get navigation status" << std::endl;
    std::cout << "\nOdometry Functions:" << std::endl;
    std::cout << " Z      Function Z: Open odometry stream" << std::endl;
    std::cout << " X      Function X: Close odometry stream" << std::endl;
    std::cout << " C      Function C: Subscribe odometry stream" << std::endl;
    std::cout << " V      Function V: Unsubscribe odometry stream" << std::endl;
    std::cout << "\nClose Functions:" << std::endl;
    std::cout << " P      Function P: Close SLAM" << std::endl;
    std::cout << " L      Function L: Close navigation" << std::endl;
    std::cout << "\n ?      Function ?: Print help" << std::endl;
    std::cout << " ESC     Exit program" << std::endl;
    std::cout << "=====\\n"
                  << std::endl;
}

int getch() {
    struct termios oldt, newt;
    int ch;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    ch = getchar();
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    return ch;
}

std::string GetUserInput(const std::string& prompt) {
    std::cout << prompt;
    std::string input;
    std::getline(std::cin, input);
    return input;
}

void RecoveryStand() {

```

(continues on next page)

(continued from previous page)

```

std::cout << "==== Executing Recovery Stand ===" << std::endl;
auto& controller = robot.GetHighLevelMotionController();

auto status = controller.SetGait(GaitMode::GAIT_RECOVERY_STAND, 10000);
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to set robot gait, code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}
std::cout << "Successfully executed recovery stand" << std::endl;
}

void BalanceStand() {
    std::cout << "==== Executing Balance Stand ===" << std::endl;
    auto& controller = robot.GetHighLevelMotionController();

    auto status = controller.SetGait(GaitMode::GAIT_BALANCE_STAND, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set robot gait, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
    std::cout << "Successfully executed balance stand" << std::endl;
}

void GetMapPath(const std::string& map_name) {
    if (map_name.empty()) {
        std::cerr << "Map to get path is not provided" << std::endl;
        return;
    }

    auto& controller = robot.GetSlamNavController();

    std::vector<std::string> map_path;
    auto status = controller.GetMapPath(map_name, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get map path, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
}

```

(continues on next page)

(continued from previous page)

```
if (map_path.empty()) {
    std::cerr << "No map path found" << std::endl;
    return;
}

for (const auto& path : map_path) {
    std::cout << "Map path: " << path << std::endl;
}
}

void SwitchToLocalizationMode(const std::string& map_path) {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::LOCALIZATION, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to localization mode, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::LOCALIZATION;
    std::cout << "Successfully switched to localization mode" << std::endl;
    std::cout << "Robot is now in localization mode, ready to localize on existing maps"
    << std::endl;
}

void InitializePose(double x, double y, double yaw) {
    auto& controller = robot.GetSlamNavController();

    Pose3DEuler initial_pose;
    initial_pose.position = {x, y, 0.0};
    initial_pose.orientation = {0.0, 0.0, yaw};

    std::cout << "Initializing robot pose to: [" << x << ", " << y << ", " << yaw << "]"
    << std::endl;

    auto status = controller.InitPose(initial_pose, 15000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to initialize pose, code: " << status.code
    }
}
```

(continues on next page)

(continued from previous page)

```

        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully initialized pose" << std::endl;
std::cout << "Robot pose has been set to [" << x << ", " << y << ", " << yaw << "]"
-><< std::endl;
}

void GetCurrentPoseInfo() {
    auto& controller = robot.GetSlamNavController();

    LocalizationInfo pose_info;
    auto status = controller.GetCurrentLocalizationInfo(pose_info);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get current pose information, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully retrieved current pose information" << std::endl;
    std::cout << "Localization status: " << (pose_info.is_localization ? "Localized" :
->"Not localized") << std::endl;
    std::cout << "Position: [" << pose_info.pose.position[0] << ", "
        << pose_info.pose.position[1] << ", "
        << pose_info.pose.position[2] << "]" << std::endl;
    std::cout << "Orientation: [" << pose_info.pose.orientation[0] << ", "
        << pose_info.pose.orientation[1] << ", "
        << pose_info.pose.orientation[2] << "]" << std::endl;
}
}

void SwitchToNavigationMode(const std::string& map_path) {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateNavMode(NavMode::GRID_MAP, map_path, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch to navigation mode, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
}
}

```

(continues on next page)

(continued from previous page)

```
current_nav_mode = NavMode::GRID_MAP;
std::cout << "Successfully switched to navigation mode" << std::endl;
}

void SetNavigationTarget(double x, double y, double yaw) {
    auto& controller = robot.GetSlamNavController();

    NavTarget target_goal;
    target_goal.id = 1;
    target_goal.frame_id = "map";
    target_goal.goal.position = {x, y, 0.0};
    target_goal.goal.orientation = {0.0, 0.0, yaw};

    auto status = controller.SetNavTarget(target_goal, 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to set navigation target, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully set navigation target: position=(" << x << ", " << y <<
    ↵", 0.0), "
        << "orientation=(0.0, 0.0, " << yaw << ")" << std::endl;
}

void PauseNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.PauseNavTask();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to pause navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully paused navigation" << std::endl;
}

void ResumeNavigation() {
```

(continues on next page)

(continued from previous page)

```

auto& controller = robot.GetSlamNavController();

auto status = controller.ResumeNavTask();
if (status.code != ErrorCode::OK) {
    std::cerr << "Failed to resume navigation, code: " << status.code
        << ", message: " << status.message << std::endl;
    return;
}

std::cout << "Successfully resumed navigation" << std::endl;
}

void CancelNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CancelNavTask();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to cancel navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully cancelled navigation" << std::endl;
}

void GetNavigationStatus() {
    auto& controller = robot.GetSlamNavController();

    NavStatus nav_status;
    auto status = controller.GetNavTaskStatus(nav_status);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to get navigation status, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "==== Navigation Status ===" << std::endl;
    std::cout << "Target ID: " << nav_status.id << std::endl;
    std::cout << "Status: " << static_cast<int>(nav_status.status) << std::endl;
    std::cout << "Error code: " << nav_status.error_code << std::endl;
}

```

(continues on next page)

(continued from previous page)

```

    std::cout << "Error description: " << nav_status.error_desc << std::endl;

    // Status interpretation
    std::string status_meaning;
    switch (nav_status.status) {
        case NavStatusType::NONE:
            status_meaning = "No navigation target set";
            break;
        case NavStatusType::RUNNING:
            status_meaning = "Navigation is running";
            break;
        case NavStatusType::END_SUCCESS:
            status_meaning = "Navigation completed successfully";
            break;
        case NavStatusType::END_FAILED:
            status_meaning = "Navigation failed";
            break;
        case NavStatusType::PAUSE:
            status_meaning = "Navigation is paused";
            break;
        case NavStatusType::CONTINUE:
            status_meaning = "Navigation resumed from pause";
            break;
        case NavStatusType::CANCEL:
            status_meaning = "Navigation was cancelled";
            break;
        default:
            status_meaning = "Unknown status";
            break;
    }

    std::cout << "Status meaning: " << status_meaning << std::endl;
    std::cout << "======" << std::endl;
}

void OpenOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.OpenOdometryStream();
    if (status.code != ErrorCode::OK) {

```

(continues on next page)

(continued from previous page)

```

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

std::cout << "Successfully opened odometry stream" << std::endl;
}

void CloseOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.CloseOdometryStream();
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close odometry stream, code: " << status.code
                    << ", message: " << status.message << std::endl;
        return;
    }

    std::cout << "Successfully closed odometry stream" << std::endl;
}

void SubscribeOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    auto callback = [](const std::shared_ptr<Odometry> odometry) {
        if (odometry_counter % 30 == 0) {
            std::cout << "Odometry position: " << odometry->position[0] << ", "
                        << odometry->position[1] << ", " << odometry->position[2] <<_
                        std::endl;
            std::cout << "Odometry orientation: " << odometry->orientation[0] << ", "
                        << odometry->orientation[1] << ", " << odometry->orientation[2] << ",
                        "
                        << odometry->orientation[3] << std::endl;
            std::cout << "Odometry linear velocity: " << odometry->linear_velocity[0] << "
                        "
                        << odometry->linear_velocity[1] << ", " << odometry->linear_
                        velocity[2] << std::endl;
            std::cout << "Odometry angular velocity: " << odometry->angular_velocity[0] <<
                        "
                        << odometry->angular_velocity[1] << ", " << odometry->angular_
                        velocity[2] << std::endl;
        }
    };
}

```

(continues on next page)

(continued from previous page)

```
    ↵velocity[2] << std::endl;
    }
    odometry_counter++;
}

controller.SubscribeOdometry(callback);
std::cout << "Successfully subscribed odometry stream" << std::endl;
}

void UnsubscribeOdometryStream() {
    auto& controller = robot.GetSlamNavController();

    controller.UnsubscribeOdometry();
    std::cout << "Successfully unsubscribed odometry stream" << std::endl;
}

void CloseSlam() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateSlamMode(SlamMode::IDLE, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close SLAM, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }

    current_slam_mode = SlamMode::IDLE;
    std::cout << "Successfully closed SLAM system" << std::endl;
}

void CloseNavigation() {
    auto& controller = robot.GetSlamNavController();

    auto status = controller.ActivateNavMode(NavMode::IDLE, "", 10000);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to close navigation, code: " << status.code
            << ", message: " << status.message << std::endl;
        return;
    }
}
```

(continues on next page)

(continued from previous page)

```

current_nav_mode = NavMode::IDLE;
std::cout << "Successfully closed navigation system" << std::endl;
}

int main(int argc, char* argv[]) {
    // Bind signal handler
    signal(SIGINT, SignalHandler);

    std::cout << "\n=====" << std::endl;
    std::cout << "MagicBot Gen1 SDK Navigation Example" << std::endl;
    std::cout << "SDK Version: " << SDK_VERSION_STRING << std::endl;
    std::cout << "=====\n"
        << std::endl;

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

    // Configure local IP address for direct network connection and initialize SDK
    std::string local_ip = "192.168.54.111";
    if (!robot.Initialize(local_ip)) {
        std::cerr << "Failed to initialize robot SDK" << std::endl;
        robotShutdown();
        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;
        robotShutdown();
        return -1;
    }
    std::cout << "Successfully connected to robot" << std::endl;

    // Switch motion control controller to high-level controller
    status = robot.SetMotionControlLevel(ControllerLevel::HighLevel);
    if (status.code != ErrorCode::OK) {
        std::cerr << "Failed to switch robot motion control level, code: " << status.code
            << ", message: " << status.message << std::endl;
    }
}

```

(continues on next page)

(continued from previous page)

```

    robot.Shutdown();
    return -1;
}

std::cout << "Successfully switched robot motion control level to high-level" <<  

→std::endl;

// Initialize SLAM navigation controller
auto& slam_nav_controller = robot.GetSlamNavController();

if (!slam_nav_controller.Initialize()) {
    std::cerr << "Failed to initialize SLAM navigation controller" << std::endl;
    robot.Disconnect();
    robot.Shutdown();
    return -1;
}
std::cout << "Successfully initialized SLAM navigation controller" << std::endl;

// Main loop
while (running) {
    int key = getch();

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

    switch (key) {
        // 1. preparation Functions
        case 'q':
        case 'Q':
            RecoveryStand();
            break;
        case 'w':
        case 'W':
            BalanceStand();
            break;

        case 'e':
        case 'E': {
            std::string map_name = GetUserInput("Enter map name to get path: ");
            GetMapPath(map_name);
    }
}

```

(continues on next page)

(continued from previous page)

```

break;
}

// 2. Localization Functions

case '1':
    std::string map_path = GetUserInput("Enter map path for localization: ");
    SwitchToLocalizationMode(map_path);
    break;
}

case '2':
    std::string input = GetUserInput("Enter pose (x y yaw): ");
    std::istringstream iss(input);
    double x = 0.0, y = 0.0, yaw = 0.0;
    iss >> x >> y >> yaw;
    std::cout << "input pose, x: " << x << ", y: " << y << ", yaw: " << yaw <<  

std::endl;
    InitializePose(x, y, yaw);
    break;
}

case '3':
    GetCurrentPoseInfo();
    break;

// 3. Navigation Functions

case '4':
    std::string map_path = GetUserInput("Enter map path for navigation: ");
    SwitchToNavigationMode(map_path);
    break;
}

case '5':
    std::string input = GetUserInput("Enter target (x y yaw): ");
    std::istringstream iss(input);
    double x = 0.0, y = 0.0, yaw = 0.0;
    iss >> x >> y >> yaw;
    SetNavigationTarget(x, y, yaw);
    break;
}

case '6':
    PauseNavigation();
    break;
case '7':

```

(continues on next page)

(continued from previous page)

```
ResumeNavigation();

break;

case '8':
    CancelNavigation();
    break;

case '9':
    GetNavigationStatus();
    break;

// 4. Odometry Functions

case 'z':
case 'Z':
    OpenOdometryStream();
    break;

case 'x':
case 'X':
    CloseOdometryStream();
    break;

case 'c':
case 'C':
    SubscribeOdometryStream();
    break;

case 'v':
case 'V':
    UnsubscribeOdometryStream();
    break;

// 5. Close Functions

case 'l':
case 'L':
    CloseNavigation();
    break;

case 'p':
case 'P':
    CloseSlam();
    break;

case '?':
    PrintHelp();
    break;

default:
    std::cout << "Unknown key: " << static_cast<char>(key) << std::endl;
```

(continues on next page)

(continued from previous page)

```

        break;
    }

    usleep(10000); // 10ms delay
}

// Cleanup resources
std::cout << "Clean up resources" << std::endl;

slam_nav_controller.Shutdown();
std::cout << "SLAM navigation controller closed" << std::endl;

robot.Disconnect();
std::cout << "Robot connection disconnected" << std::endl;

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

return 0;
}

```

## 26.2 Python

Example files:

- slam\_example.py
- navigation\_example.py

Reference documentation:

- Python API/slam\_navigation\_reference.md

### 26.2.1 Mapping Example

```

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

import sys
import time
import signal
import threading

```

(continues on next page)

(continued from previous page)

```

import logging
import termios
import tty
from typing import Optional
import numpy as np
import cv2
import random

import magicbot_gen1_python as magicbot

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

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicbot.NavMode.IDLE

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

def print_help():
    """Print help information"""
    logging.info("SLAM and Navigation Function Demo Program")
    logging.info("preparation Functions:")
    logging.info(" Q           Function Q: Recovery stand")

```

(continues on next page)

(continued from previous page)

```

logging.info(" E      Function E: Balance stand")
logging.info(" W      Function W: Move forward")
logging.info(" A      Function A: Move left")
logging.info(" S      Function S: Move backward")
logging.info(" D      Function D: Move right")
logging.info(" T      Function T: Turn left")
logging.info(" G      Function G: Turn right")
logging.info(" X      Function X: Stop move")
logging.info("")
logging.info("SLAM Functions:")
logging.info(" 1      Function 1: Switch to mapping mode")
logging.info(" 2      Function 2: Start mapping")
logging.info(" 3      Function 3: Cancel mapping")
logging.info(" 4      Function 4: Save map")
logging.info(" 5      Function 5: Load map")
logging.info(" 6      Function 6: Delete map")
logging.info(
    " 7      Function 7: Get all map information and save map image as PGM file
    ↵"
)
logging.info(" 8      Function 8: Get map path")
logging.info(" 9      Function 9: Get SLAM mapping point cloud map")
logging.info("")
logging.info("Close Functions:")
logging.info(" P      Function P: Close SLAM")
logging.info("")
logging.info(" ?      Function ?: Print help")
logging.info(" ESC     Exit program")

# ===== SLAM Functions =====

def switch_to_mapping_mode():
    """Switch to mapping mode"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

```

(continues on next page)

(continued from previous page)

```

# Switch to mapping mode
status = controller.activate_slam_mode(magicbot.SlamMode.MAPPING)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to switch to mapping mode, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

current_slam_mode = "MAPPING"
logging.info("Successfully switched to mapping mode")
logging.info("Robot is now in mapping mode, ready to create new maps")
except Exception as e:
    logging.error("Exception occurred while switching to mapping mode: %s", e)

def load_map(map_to_load):
    """Load map"""
    global robot
    try:
        if not map_to_load:
            logging.error("Map to load is not provided")
            return
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        logging.info("Loading map: %s", map_to_load)
        controller.load_map(map_to_load)
    except Exception as e:
        logging.error("Exception occurred while loading map: %s", e)
        return

    logging.info("Successfully loaded map: %s", map_to_load)

def start_mapping():
    """Start mapping"""
    global robot
    try:

```

(continues on next page)

(continued from previous page)

```
# Get SLAM navigation controller
controller = robot.get_slam_nav_controller()

# Start mapping
status = controller.start_mapping()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to start mapping, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully started mapping")
except Exception as e:
    logging.error("Exception occurred while starting mapping: %s", e)

def cancel_mapping():
    """Cancel mapping"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Cancel mapping
        status = controller.cancel_mapping()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to cancel mapping, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        logging.info("Successfully cancelled mapping")
    except Exception as e:
        logging.error("Exception occurred while cancelling mapping: %s", e)
```

(continues on next page)

(continued from previous page)

```

def save_map():
    """Save map"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Check if in mapping mode
        if current_slam_mode != "MAPPING":
            logging.warning(
                "Warning: Currently not in mapping mode, may not be able to save map"
            )

        # Generate map name with timestamp
        map_name = f"map_{int(time.time())}"
        logging.info("Saving map: %s", map_name)

        # Save map
        status = controller.save_map(map_name, 20000)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to save map, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully saved map: %s", map_name)
    except Exception as e:
        logging.error("Exception occurred while saving map: %s", e)

def delete_map(map_to_delete):
    """Delete map"""
    global robot
    try:
        if not map_to_delete:
            logging.error("Map to delete is not provided")
        return
        # Get SLAM navigation controller

```

(continues on next page)

(continued from previous page)

```

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 != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to delete map, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully deleted map: %s", map_to_delete)
except Exception as e:
    logging.error("Exception occurred while deleting map: %s", e)

def get_all_map_info():
    """Get all map information"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get all map information
        status, all_map_info = controller.get_all_map_info()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get map information, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

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

(continues on next page)

(continued from previous page)

```

if all_map_info.map_infos:
    logging.info("Map details:")
    for i, map_info in enumerate(all_map_info.map_infos):
        logging.info("  Map %d: %s", i + 1, map_info.map_name)
        logging.info(
            "    Origin: [%f, %f, %f]",
            map_info.map_meta_data.origin.position[0],
            map_info.map_meta_data.origin.position[1],
            map_info.map_meta_data.origin.position[2],
        )
        logging.info(
            "    Orientation: [%f, %f, %f]",
            map_info.map_meta_data.origin.orientation[0],
            map_info.map_meta_data.origin.orientation[1],
            map_info.map_meta_data.origin.orientation[2],
        )
        logging.info(
            "    Resolution: %f m/pixel", map_info.map_meta_data.resolution
        )
        logging.info(
            "    Size: %d x %d",
            map_info.map_meta_data.map_image_data.width,
            map_info.map_meta_data.map_image_data.height,
        )
        logging.info(
            "    Max gray value: %d",
            map_info.map_meta_data.map_image_data.max_gray_value,
        )
        logging.info(
            "    Image type: %s", map_info.map_meta_data.map_image_data.type
        )

    save_map_image_to_file(map_info)

else:
    logging.info("No available maps")

except Exception as e:
    logging.error("Exception occurred while getting map information: %s", e)

```

(continues on next page)

(continued from previous page)

```

def save_map_image_to_file(map_info):
    """Save map image to current directory"""

    try:
        # Extract image data
        map_data = map_info.map_meta_data.map_image_data
        width = map_data.width
        height = map_data.height
        max_gray_value = map_data.max_gray_value
        image_bytes = map_data.image

        logging.info(
            "Saving map image: %dx%d, max_gray: %d", width, height, max_gray_value
        )

        # Convert bytes to numpy array
        if len(image_bytes) != width * height:
            logging.error(
                "Image data size mismatch: expected %d, got %d",
                width * height,
                len(image_bytes),
            )
        return

        # Convert Uint8Vector to bytes for numpy processing
        try:
            # Try to convert Uint8Vector to bytes
            if hasattr(image_bytes, "__iter__") and not isinstance(
                image_bytes, (str, bytes)
            ):
                # Convert Uint8Vector or similar iterable to bytes
                image_bytes_data = bytes(image_bytes)

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

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

```

(continues on next page)

(continued from previous page)

```

        (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 get_map_path(map_to_get_path):
    """Get map path"""
    global robot
    try:
        if not map_to_get_path:
            logging.error("Map to get path is not provided")
            return
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get map path

```

(continues on next page)

(continued from previous page)

```

status, map_path = controller.get_map_path(map_to_get_path)

if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to get map path, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

if len(map_path) == 0:
    logging.error("No map path found")
    return

for path in map_path:
    logging.info("Map path: %s", path)

except Exception as e:
    logging.error("Exception occurred while getting map path: %s", e)
    return


def get_point_cloud_map():
    """Get SLAM mapping point cloud map"""

    global robot

    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get SLAM mapping point cloud map
        status, point_cloud_map = controller.get_point_cloud_map()

        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get SLAM mapping point cloud map, code: %s, message: %s",
                status.code,
                status.message,
            )
            return

        logging.info("Successfully got SLAM mapping point cloud map")

        if point_cloud_map:
            logging.info(
                "Point cloud map - Height: %d, Width: %d, Data size: %d bytes",
                point_cloud_map.height,
                point_cloud_map.width,
                point_cloud_map.data_size,
            )
    except Exception as e:
        logging.error("Exception occurred while getting SLAM mapping point cloud map: %s", e)
        return

```

(continues on next page)

(continued from previous page)

```

        point_cloud_map.height,
        point_cloud_map.width,
        len(point_cloud_map.data),
    )

except Exception as e:
    logging.error(
        "Exception occurred while getting SLAM mapping point cloud map: %s",
        e,
    )

def close_slam():
    """Close SLAM system"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to idle mode to close SLAM
        status = controller.activate_slam_mode(magicbot.SlamMode.IDLE)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close SLAM, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        current_slam_mode = "IDLE"
        logging.info("Successfully closed SLAM system")
    except Exception as e:
        logging.error("Exception occurred while closing SLAM: %s", e)

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

```

(continues on next page)

(continued from previous page)

```

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

logging.info("Successfully executed recovery stand")
return

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


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

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

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

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

    except Exception as e:

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

def move_forward():
    """Move forward"""

```

(continues on next page)

(continued from previous page)

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

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

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

def move_right():
    """Move right"""
    logging.info("==== Moving Right ====")
    return joystick_command(1.0, 0.0, 0.0, 0.0)

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

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

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

(continues on next page)

(continued from previous page)

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

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

# Get single character input (no echo)
def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
        logging.info(f"Received character: {ch}")
        sys.stdout.write("\r")
        sys.stdout.flush()
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

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

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

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

    # Create robot instance
    robot = magicbot.MagicRobot()
```

(continues on next page)

(continued from previous page)

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

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

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

    logging.info("Successfully connected to robot")

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

    # Initialize SLAM navigation controller
    slam_nav_controller = robot.get_slam_nav_controller()
    if not slam_nav_controller.initialize():
        logging.error("Failed to initialize SLAM navigation controller")
        robot.disconnect()
```

(continues on next page)

(continued from previous page)

```

robot.shutdown()

return -1

logging.info("Successfully initialized SLAM navigation controller")

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

# 1. Preparation Functions
# 1.1 Execute recovery stand first. Two mapping methods: 1) Suspended
→in air, can push robot for mapping in recovery_stand state; 2) Switch to balance_
→stand state for remote-controlled mapping
        if key.upper() == "Q":
            recovery_stand()
        elif key.upper() == "E":
            balance_stand()
        elif key.upper() == "W":
            move_forward()
        elif key.upper() == "A":
            move_left()
        elif key.upper() == "S":
            move_backward()
        elif key.upper() == "D":
            move_right()
        elif key.upper() == "T":
            turn_left()
        elif key.upper() == "G":
            turn_right()
        elif key.upper() == "X":
            stop_move()

# 2. Mapping Mode
# 2.1 Switch to mapping mode
        elif key == "1":
            switch_to_mapping_mode()
# 2.2 Start mapping
        elif key == "2":

```

(continues on next page)

(continued from previous page)

```

        start_mapping()

    # 2.3 Cancel mapping
    elif key == "3":
        cancel_mapping()

    # 2.4 Save map
    elif key == "4":
        save_map()

    # 2.5 Load map
    elif key == "5":
        str_input = get_user_input()
        # Split input parameters by space
        parts = str_input.strip().split()
        # Parse parameters
        map_to_load = parts[0] if parts else ""
        load_map(map_to_load)

    # 2.6 Delete map
    elif key == "6":
        str_input = get_user_input()
        # Split input parameters by space
        parts = str_input.strip().split()
        # Parse parameters
        map_to_delete = parts[0] if parts else ""
        delete_map(map_to_delete)

    # 2.7 Get all map information
    elif key == "7":
        get_all_map_info()

    # 2.8 Get map path
    elif key.upper() == "8":
        str_input = get_user_input()
        # Split input parameters by space
        parts = str_input.strip().split()
        # Parse parameters
        map_to_get_path = parts[0] if parts else ""
        get_map_path(map_to_get_path)

    # 2.9 Get SLAM mapping point cloud map
    elif key.upper() == "9":
        get_point_cloud_map()

    # 3. Close Functions
    # 3.1 Close SLAM
    elif key.upper() == "P":

```

(continues on next page)

(continued from previous page)

```
        close_slam()

    elif key.upper() == "?":
        print_help()

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

        time.sleep(0.01) # Brief delay

except KeyboardInterrupt:
    break
except Exception as e:
    logging.error("Exception occurred while processing user input: %s", e)
except Exception as e:
    logging.error("Exception occurred during program execution: %s", e)
return -1

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close SLAM navigation controller
        slam_nav_controller = robot.get_slam_nav_controller()
        slam_nav_controller.shutdown()
        logging.info("SLAM navigation controller closed")

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

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

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

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

## 26.2.2 Navigation Example

```

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

import sys
import time
import signal
import threading
import logging
from typing import Optional
import numpy as np
import cv2
import random

import magicbot_gen1_python as magicbot

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

# Global variables
robot: Optional[magicbot.MagicRobot] = None
running = True
current_slam_mode = None
current_nav_mode = magicbot.NavMode.IDLE
odometry_counter = 0


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

```

(continues on next page)

(continued from previous page)

```

def print_help():
    """Print help information"""
    logging.info("SLAM and Navigation Function Demo Program")
    logging.info("")
    logging.info("preparation Functions:")
    logging.info(" Q      Function Q: Recovery stand")
    logging.info(" W      Function W: Balance stand")
    logging.info(" E      Function E: Get map path")
    logging.info("")
    logging.info("Localization Functions:")
    logging.info(" 1      Function 1: Switch to localization mode")
    logging.info(" 2      Function 2: Initialize pose")
    logging.info(" 3      Function 3: Get current pose information")
    logging.info("")
    logging.info("Navigation Functions:")
    logging.info(" 4      Function 4: Switch to navigation mode")
    logging.info(" 5      Function 5: Set navigation target goal")
    logging.info(" 6      Function 6: Pause navigation")
    logging.info(" 7      Function 7: Resume navigation")
    logging.info(" 8      Function 8: Cancel navigation")
    logging.info(" 9      Function 9: Get navigation status")
    logging.info("")
    logging.info("Odometry Functions:")
    logging.info(" Z      Function Z: Open odometry stream")
    logging.info(" X      Function X: Close odometry stream")
    logging.info(" C      Function C: Subscribe odometry stream")
    logging.info(" V      Function V: Unsubscribe odometry stream")
    logging.info("")
    logging.info("Close Functions:")
    logging.info(" P      Function P: Close SLAM")
    logging.info(" L      Function L: Close navigation")
    logging.info("")
    logging.info(" ?      Function ?: Print help")
    logging.info(" ESC     Exit program")

# ===== Navigation Functions =====
def get_map_path(map_to_get_path):

```

(continues on next page)

(continued from previous page)

```

"""Get map path"""
global robot
try:
    if not map_to_get_path:
        logging.error("Map to get path is not provided")
        return
    # Get SLAM navigation controller
    controller = robot.get_slam_nav_controller()

    # Get map path
    status, map_path = controller.get_map_path(map_to_get_path)
    if status.code != magicbot.ErrorCode.OK:
        logging.error(
            "Failed to get map path, code: %s, message: %s",
            status.code,
            status.message,
        )
        return
    if len(map_path) == 0:
        logging.error("No map path found")
        return

    for path in map_path:
        logging.info("Map path: %s", path)
except Exception as e:
    logging.error("Exception occurred while getting map path: %s", e)
    return


def switch_to_localization_mode(map_path):
    """Switch to localization mode"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to localization mode
        status = controller.activate_slam_mode(magicbot.SlamMode.LOCALIZATION, map_
        ↪path)
        if status.code != magicbot.ErrorCode.OK:
            logging.error("Failed to switch to localization mode, code: %s, message: %s", status.code, status.message)
            return
        current_slam_mode = magicbot.SlamMode.LOCALIZATION
    except Exception as e:
        logging.error("Exception occurred while switching to localization mode: %s", e)
        return

```

(continues on next page)

(continued from previous page)

```

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)
    return

def initialize_pose(x, y, yaw):
    """Initialize pose"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Create initial pose (set to origin)
        initial_pose = magicbot.Pose3DEuler()
        initial_pose.position = [x, y, 0.0]  # x, y, z
        initial_pose.orientation = [0.0, 0.0, yaw]  # roll, pitch, yaw

        logging.info("Initializing robot pose to origin...")

        # Initialize pose
        status = controller.init_pose(initial_pose)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to initialize pose, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

```

(continues on next page)

(continued from previous page)

```
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, pose_info = controller.get_current_localization_info()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get current pose information, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        logging.info("Successfully retrieved current pose information")
        logging.info(
            "Localization status: %s",
            "Localized" if pose_info.is_localization else "Not localized",
        )
        logging.info(
            "Position: [%f, %f, %f]",
            pose_info.pose.position[0],
            pose_info.pose.position[1],
            pose_info.pose.position[2],
        )
        logging.info(
            "Orientation: [%f, %f, %f]",
            pose_info.pose.orientation[0],
            pose_info.pose.orientation[1],
            pose_info.pose.orientation[2],
        )

(continues on next page)
```

(continued from previous page)

```

        )

except Exception as e:
    logging.error(
        "Exception occurred while getting current pose information: %s", e
    )

def switch_to_navigation_mode(map_path):
    """Switch to navigation mode"""
    global robot, current_nav_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to navigation mode
        status = controller.activate_nav_mode(magicbot.NavMode.GRID_MAP, map_path)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to switch to navigation mode, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

    current_nav_mode = magicbot.NavMode.GRID_MAP
    logging.info("Successfully switched to navigation mode")

except Exception as e:
    logging.error("Exception occurred while switching to navigation mode: %s", e)

def set_navigation_target(x, y, yaw):
    """Set navigation target goal"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Create target goal
        target_goal = magicbot.NavTarget()
        target_goal.id = 1
    
```

(continues on next page)

(continued from previous page)

```

target_goal.frame_id = "map"

# Set target pose (example: move 2 meters forward)
target_goal.goal.position = [x, y, 0.0]
# Set target orientation (example: no rotation)
target_goal.goal.orientation = [0.0, 0.0, yaw]

# Set target goal
status = controller.set_nav_target(target_goal)
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to set navigation target, code: %s, message: %s",
        status.code,
        status.message,
    )
return

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

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()

```

(continues on next page)

(continued from previous page)

```
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to pause navigation, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully paused navigation")
except Exception as e:
    logging.error("Exception occurred while pausing navigation: %s", e)

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

        # Resume navigation
        status = controller.resume_nav_task()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "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
```

(continues on next page)

(continued from previous page)

```
controller = robot.get_slam_nav_controller()

# Cancel navigation
status = controller.cancel_nav_task()
if status.code != magicbot.ErrorCode.OK:
    logging.error(
        "Failed to cancel navigation, code: %s, message: %s",
        status.code,
        status.message,
    )
    return

logging.info("Successfully cancelled navigation")

except Exception as e:
    logging.error("Exception occurred while cancelling navigation: %s", e)

def get_navigation_status():
    """Get current navigation status"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Get navigation status
        status, nav_status = controller.get_nav_task_status()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to get navigation status, code: %s, message: %s",
                status.code,
                status.message,
            )
        return

        # Display navigation status information
        logging.info("==== Navigation Status ====")
        logging.info("Target ID: %d", nav_status.id)
        logging.info("Status: %s", nav_status.status)
        logging.info("Error Code: %d", nav_status.error_code)
        logging.info("Error Description: %s", nav_status.error_desc)
    
```

(continues on next page)

(continued from previous page)

```

# Provide status interpretation
status_meaning = {
    magicbot.NavStatusType.NONE: "No navigation target set",
    magicbot.NavStatusType.RUNNING: "Navigation is running",
    magicbot.NavStatusType.END_SUCCESS: "Navigation completed successfully",
    magicbot.NavStatusType.END_FAILED: "Navigation failed",
    magicbot.NavStatusType.PAUSE: "Navigation is paused",
    magicbot.NavStatusType.CONTINUE: "Navigation resumed from pause",
    magicbot.NavStatusType.CANCEL: "Navigation was cancelled",
}

if nav_status.status in status_meaning:
    logging.info("Status meaning: %s", status_meaning[nav_status.status])
else:
    logging.warning("Unknown status value: %s", nav_status.status)

logging.info("=====")

except Exception as e:
    logging.error("Exception occurred while getting navigation status: %s", e)

def close_navigation():
    """Close navigation system"""
    global robot, current_nav_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to idle mode to close navigation
        status = controller.activate_nav_mode(magicbot.NavMode.IDLE)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close navigation, code: %s, message: %s",
                status.code,
                status.message,
            )
    return

```

(continues on next page)

(continued from previous page)

```

        current_nav_mode = magicbot.NavMode.IDLE
        logging.info("Successfully closed navigation system")

    except Exception as e:
        logging.error("Exception occurred while closing navigation: %s", e)

def open_odometry_stream():
    """Open odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Open odometry stream
        status = controller.open_odometry_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to open odometry stream, code: %s, message: %s",
                status.code,
                status.message,
            )
        return
    logging.info("Successfully opened odometry stream")
except Exception as e:
    logging.error("Exception occurred while opening odometry stream: %s", e)

def close_odometry_stream():
    """Close odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Close odometry stream
        status = controller.close_odometry_stream()
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close odometry stream, code: %s, message: %s",
                status.code,
            )
    
```

(continues on next page)

(continued from previous page)

```
        status.message,
    )
    return

logging.info("Successfully closed odometry stream")
except Exception as e:
    logging.error("Exception occurred while closing odometry stream: %s", e)

def subscribe_odometry_stream():
    """Subscribe odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        def callback(odometry: magicbot.Odometry):
            global odometry_counter
            if odometry_counter % 30 == 0:
                logging.info(
                    "Odometry position: %f, %f, %f",
                    odometry.position[0],
                    odometry.position[1],
                    odometry.position[2],
                )
                logging.info(
                    "Odometry orientation: %f, %f, %f, %f",
                    odometry.orientation[0],
                    odometry.orientation[1],
                    odometry.orientation[2],
                    odometry.orientation[3],
                )
                logging.info(
                    "Odometry linear velocity: %f, %f, %f",
                    odometry.linear_velocity[0],
                    odometry.linear_velocity[1],
                    odometry.linear_velocity[2],
                )
                logging.info(
                    "Odometry angular velocity: %f, %f, %f",
                    odometry.angular_velocity[0],

```

(continues on next page)

(continued from previous page)

```

        odometry.angular_velocity[1],
        odometry.angular_velocity[2],
    )
odometry_counter += 1

# Subscribe odometry stream
controller.subscribe_odometry(callback)
logging.info("Successfully subscribed odometry stream")
except Exception as e:
    logging.error("Exception occurred while subscribing odometry stream: %s", e)

def unsubscribe_odometry():
    """Unsubscribe odometry stream"""
    global robot
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Unsubscribe odometry stream
        controller.unsubscribe_odometry()
        logging.info("Successfully unsubscribed odometry stream")
    except Exception as e:
        logging.error("Exception occurred while unsubscribing odometry stream: %s", e)

def close_slam():
    """Close SLAM system"""
    global robot, current_slam_mode
    try:
        # Get SLAM navigation controller
        controller = robot.get_slam_nav_controller()

        # Switch to idle mode to close SLAM
        status = controller.activate_slam_mode(magicbot.SlamMode.IDLE)
        if status.code != magicbot.ErrorCode.OK:
            logging.error(
                "Failed to close SLAM, code: %s, message: %s",
                status.code,
                status.message,
            )
    
```

(continues on next page)

(continued from previous page)

```
)  
return  
  
current_slam_mode = "IDLE"  
logging.info("Successfully closed SLAM system")  
except Exception as e:  
    logging.error("Exception occurred while closing SLAM: %s", e)  
  
  
def recovery_stand():  
    """Recovery stand"""  
global robot  
try:  
    logging.info("== Executing Recovery Stand ==")  
    controller = robot.get_high_level_motion_controller()  
    status = controller.set_gait(magicbot.GaitMode.GAIT_RECOVERY_STAND, 10000)  
    if status.code != magicbot.ErrorCode.OK:  
        logging.error(  
            "Failed to set robot gait, code: %s, message: %s",  
            status.code,  
            status.message,  
        )  
    return  
    logging.info("Successfully executed recovery stand")  
    return  
except Exception as e:  
    logging.error("Exception occurred while executing recovery stand: %s", e)  
    return  
  
  
def balance_stand():  
    """Balance stand"""  
global robot  
try:  
    logging.info("== Executing Balance Stand ==")  
    controller = robot.get_high_level_motion_controller()  
    status = controller.set_gait(magicbot.GaitMode.GAIT_BALANCE_STAND, 10000)  
    if status.code != magicbot.ErrorCode.OK:  
        logging.error(  
            "Failed to set robot gait, code: %s, message: %s",
```

(continues on next page)

(continued from previous page)

```

        status.code,
        status.message,
    )
    return
logging.info("Successfully executed balance stand")
return
except Exception as e:
    logging.error("Exception occurred while executing balance stand: %s", e)
    return

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

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

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

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

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

    # Create robot instance
    robot = magicbot.MagicRobot()

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

    try:
        # Configure local IP address for direct network connection and initialize SDK

```

(continues on next page)

(continued from previous page)

```
local_ip = "192.168.54.111"

if not robot.initialize(local_ip):
    logging.error("Failed to initialize robot SDK")
    robot.shutdown()
    return -1

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

logging.info("Successfully connected to robot")

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

logging.info("Successfully switched robot motion control level to high-level")

# Initialize SLAM navigation controller
slam_nav_controller = robot.get_slam_nav_controller()
if not slam_nav_controller.initialize():
    logging.error("Failed to initialize SLAM navigation controller")
    robot.disconnect()
    robot.shutdown()
    return -1
```

(continues on next page)

(continued from previous page)

```

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
        # 1.1 Execute recovery stand first
        if key.upper() == "Q":
            recovery_stand()
        # 1.2 Switch to balance stand, allowing robot to transition to_
        ↪walking gait
        elif key.upper() == "W":
            balance_stand()
        # 1.3 Get current map absolute path
        elif key.upper() == "E":
            map_to_get_path = args[0] if args else ""
            get_map_path(map_to_get_path)
        # 2. Enable Localization Mode and Initialize Pose
        # 2.2 Based on map absolute path, enable localization mode
        elif key == "1":
            map_path = args[0] if args else ""
            switch_to_localization_mode(map_path)
        # 2.3 Based on current map, initialize pose
        elif key == "2":
            x = float(args[0]) if args else 0.0

```

(continues on next page)

(continued from previous page)

```

y = float(args[1]) if args else 0.0
yaw = float(args[2]) if args else 0.0
logging.info("input pose, x: %f, y: %f, yaw: %f", x, y, yaw)
initialize_pose(x, y, yaw)

# 2.4 Get current initialized pose status, check if localization_
↪succeeded

elif key == "3":
    get_current_localization_info()
    # 3. Start Navigation
    # 3.1 Based on map absolute path, enable navigation mode
elif key.upper() == "4":
    map_path = args[0] if args else ""
    switch_to_navigation_mode(map_path)
    # 3.2 Input target point, start navigation task
    # Due to the lidar being installed with a -1.57rad offset relative to_
↪the robot's front,
    # the desired yaw orientation needs to be offset by -1.57rad,_
↪otherwise the robot's pose initialization may fail
    # Please input yaw orientation, needs to be offset by -1.57rad
elif key.upper() == "5":
    x = float(args[0]) if args else 0.0
    y = float(args[1]) if args else 0.0
    yaw = float(args[2]) if args else 0.0
    logging.info("input target, x: %f, y: %f, yaw: %f", x, y, yaw)
    set_navigation_target(x, y, yaw)
    # 3.3 Pause navigation task
elif key.upper() == "6":
    pause_navigation()
    # 3.4 Resume navigation task
elif key.upper() == "7":
    resume_navigation()
    # 3.5 Cancel navigation task
elif key.upper() == "8":
    cancel_navigation()
    # 3.6 Get navigation task status
elif key.upper() == "9":
    get_navigation_status()
    # 4. Subscribe to Odometry Data
    # 4.1 Open odometry stream
elif key.upper() == "Z":

```

(continues on next page)

(continued from previous page)

```

        open_odometry_stream()

    # 4.2 Close odometry stream
    elif key.upper() == "X":
        close_odometry_stream()

    # 4.3 Subscribe to odometry stream
    elif key.upper() == "C":
        subscribe_odometry_stream()

    # 4.4 Unsubscribe from odometry stream
    elif key.upper() == "V":
        unsubscribe_odometry_stream()

    # 5. Close SLAM and Navigation
    # 5.1 Close SLAM
    elif key.upper() == "P":
        close_slam()

    # 5.2 Close navigation
    elif key.upper() == "L":
        close_navigation()

    elif key.upper() == "?":
        print_help()

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

        time.sleep(0.01) # Brief delay

    except KeyboardInterrupt:
        break

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

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

finally:
    # Clean up resources
    try:
        logging.info("Clean up resources")
        # Close SLAM navigation controller
        slam_nav_controller = robot.get_slam_nav_controller()
        slam_nav_controller.shutdown()
        logging.info("SLAM navigation controller closed")
    
```

(continues on next page)

(continued from previous page)

```

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

```

### 26.3.2 Run Examples

```

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

```

### 26.3.3 Control Instructions

#### Mapping

- Preparation Functions:
  - Press Q - Recovery stand
- SLAM Functions:
  - Press 1 - Switch to mapping mode

- Press 2 - Start mapping
- Press 3 - Cancel mapping
- Press 4 - Save map
- Press 5 - Load map
- Press 6 - Delete map
- Press 7 - Get all map information and save map image as PGM file
- Press 8 - Get map path
- Press 9 - Get SLAM mapping point cloud map
- Close Functions:
  - Press P - Close SLAM
- System Commands:
  - Press ? - Show help information

## Navigation

- Preparation Functions:
  - Press Q - Recovery stand
  - Press W - Balance stand
  - Press E - Get map path
- Localization Functions:
  - Press 1 - Switch to localization mode
  - Press 2 - Initialize pose
  - Press 3 - Get current position information
- Navigation Functions:
  - Press 4 - Switch to navigation mode
  - Press 5 - Set navigation target goal
  - Press 6 - Pause navigation
  - Press 7 - Resume navigation
  - Press 8 - Cancel navigation
  - Press 9 - Get navigation status
- Odometry Functions:
  - Press Z - Open odometry stream

- Press X - Close odometry stream
- Press C - Subscribe odometry stream
- Press V - Unsubscribe odometry stream
- Close Functions:
  - Press P - Close SLAM
  - Press L - Close navigation
- System Commands:
  - Press ? - Show help information

### 26.3.4 Stop Program

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

### 26.3.5 Notes

- In the mapping and navigation examples, you need to switch the robot to high-level motion control mode in advance (the default control mode is high-level motion control).
- In the mapping example, you need to switch to Recovery\_stand mode before mapping. You can either push the robot in Recovery\_stand mode or use remote control in Balance\_stand mode to move the robot for mapping (本示例是展示机器人 Recovery\_stand 模式下推着机器人进行建图) .
- In the mapping example, you can use GetAllMapInfo to get information about maps stored on the robot (including 2D map PGM, map size, resolution, etc.).
- In the mapping example, you can manage maps using SaveMap/DeleteMap. Try not to store more than 3 maps on the robot.
- In the navigation example, you need to switch the robot to Balance\_stand mode first.
- In the navigation example, it's best to set navigation target points based on the current localization status and pose obtained from GetCurrentLocalizationInfo.
- In the navigation example, before starting navigation tasks, you need to properly switch to localization mode (ActivateSlamMode(SlamMode::LOCALIZATION, map\_path)) and navigation mode (ActivateNavMode(NavMode::GRID\_MAP, map\_path)).
- In the navigation example, both localization mode and navigation mode switches require the absolute path of the map, which can be obtained using GetMapPath.
- After finishing mapping or navigation, switch SLAM to IDLE mode (ActivateSlamMode(SlamMode::IDLE)) and navigation to IDLE mode (ActivateNavMode(NavMode::IDLE))



# CHAPTER 27

---

## 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](#)

### 27.1 Development Environment Preparation

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

### 27.2 Does Magicbot-Gen1 support wireless development?

No. Magicbot-Gen1 currently supports only wired connections for development.

### 27.3 What is the current low-level communication frequency of Magicbot-Gen1?

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.

## 27.4 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 `OpenLidar` 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 244.0.0.0 netmask 240.0.0.0 dev enp0s31f6
```

- SDK version not aligned with the robot firmware version:

The SDK version tag must match the robot firmware version. Refer to the [ChangeLog](#) for the version mapping.

- Clean residual SDK configuration file:

The SDK automatically generates the default configuration file at `/tmp/magicbot_gen1_mjrrt.yaml`, which is normally cleared after a reboot. However, if during the same boot session you first use an older SDK version (which generates an old configuration file in `/tmp`), and then switch to a newer SDK version, the newer SDK will still load the legacy configuration by default. This is because the SDK will not recreate the configuration file if `magicbot_gen1_mjrrt.yaml` already exists, for debugging convenience. As a result, topic subscription and publication may malfunction.

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.

## 27.5 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
```

## 27.6 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.

## 27.7 How to View SDK Internal Configuration and Logs?

When the SDK program is executed, it will by default generate a configuration file at `/tmp/magicbot_gen1_mjrrt.yaml` and a log file at `/tmp/logs/magicbot_gen1_sdk.log`. Any internal error messages of the SDK can be checked in this log file.

## 27.8 Video Streaming

How to access trinocular camera video stream:

```
# Wireless (requires connecting to the robot's hotspot)
ffplay rtsp://192.168.12.1:8082/
# Wired
ffplay rtsp://192.168.54.119:8082/
```

You can use ffplay for testing during demonstration. If you want to capture individual frames, you need to implement your own encoding.

## 27.9 SLAM Relocation Failed?

SLAM navigation via SDK differs from the App's convenient visual localization. You need to modify the SLAM configuration to fix the map orientation for easier SLAM localization:

```
# Log into the robot
ssh eame@192.168.54.119
# Modify the following configuration
# vim /opt/eame/eamegrapher_3d/share/eamegrapher_3d/config/params.yaml
align_map: false

# After modifying the yaml file for the first time, restart the SLAM module for
# changes to take effect with:
sudo systemctl restart slam.service
```

## 27.10 SLAM Navigation Not Executing?

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

- `ActivateSlamMode(SlamMode::LOCALIZATION, map_path)`
- `ActivateNavMode(NavMode::GRID_MAP, map_path)`

Where `map_path` is the absolute path to the current map, which can be obtained via `GetMapPath`

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`.

## 27.11 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.

## 27.12 Log Error: Call of `pthread_setschedparam` with insufficient privileges!

The main reason for this error is that the SDK is being executed with 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` for the regular user to ensure the process has permission to set thread priorities:

```
*      -      rtprio    98
```

## 27.13 connect Error: Failed to connect to robot, code: ErrorCode.SERVICE\_ERROR, message: failed to connect to all addresses; last error: UNKNOWN: ipv4:192.168.54.119:50051: Failed to connect to remote host: Connection refused

1. Unable to ping 192.168.54.119:

- Network cable connection problem, check hardware connections.
  - The compute box IP configuration is not in the 192.168.54.XX subnet.
2. Robot gateway service exception:
- Try restarting to restore normal robot software operation.

## 27.14 Trick Execution Error: Failed to execute robot trick, code:

**ErrorCode.SERVICE\_ERROR, message:** 优先级过低

Reason 1: The trick command ID called is not within the TrickAction enumeration range.

Reason 2: The robot needs to be on the ground and must switch to the balance\_stand gait.

Note: For humanoid robots, the robot must first land on the ground in the recovery\_stand state. After landing, switch to balance\_stand, and then you can perform tricks and other actions.

## 27.15 Trick Execution Error: Failed to execute robot trick, code:

**ErrorCode.SERVICE\_ERROR, message:** 危险动作

The possible reason for this error may be that the robot's current initial position differs greatly from the required starting position for the trick.

Solution:

- Try executing recovery\_stand and then balance\_stand in order to restore and retry.

## 27.16 Trick Execution Error: Failed to execute robot trick, code:

**ErrorCode.SERVICE\_ERROR, message:** 调用 ROS2 服务失败

Possible cause: There is an exception in the internal software service of the robot.

Solution:

- Restart the robot to try to recover. If restarting does not work, please contact after-sales support.