

LimX Oli EDU User Manual

1. LimX Oli EDU Overview

Safety Notice for LimX Humanoid Robot Operation:

To ensure the safety of both users and the robot, please read the following precautions carefully before use:

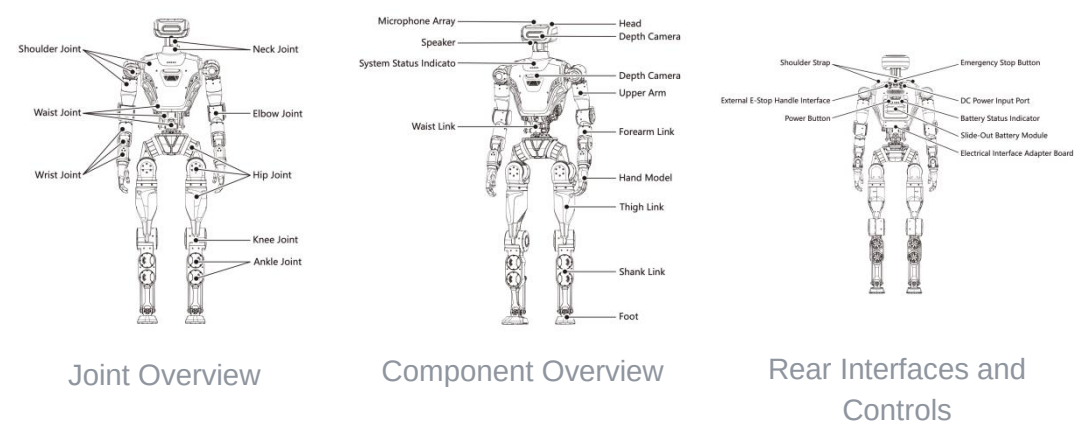
- 1. Professional Use Only:** This product is intended for professional use. Individuals under 18 are not advised to operate the device. The operation should only be performed by persons with relevant technical knowledge or under the supervision of qualified personnel.
- 2. Keep Out of Reach of Children:** Prevent children from approaching or interacting with the robot. In environments where children are present, take extra precautions and ensure the robot operates only in safe and supervised conditions.
- 3. Idle Protection:** Do not leave the robot powered on and unattended. Activate mechanical or software-based suspension protection to prevent unintended movement.
- 4. Safe Operating Environment:** Keep the operating area clear of obstacles at all times. Use safety restraints or tethers when necessary to prevent accidental movement, tipping, or injury.
- 5. Maintain a Safe Distance:** Always keep the robot within your line of sight. Maintain a minimum distance of 1 meter (3 feet) from the robot during operation.
- 6. Handling Precautions:** Do not touch the robot's joints or actuating components immediately after power-on or during transportation, as unexpected movement may cause pinching or impact injuries.
- 7. Low Battery Warning:** When a low battery warning is displayed, power off the robot and recharge the battery immediately to prevent unexpected shutdown, system malfunction, or damage.

1.1 Component Description

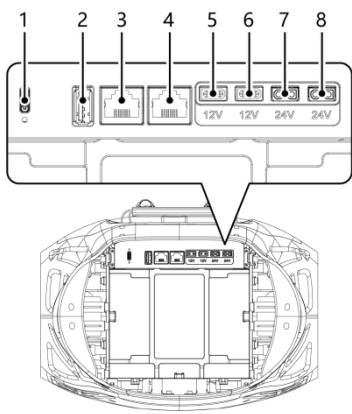
The LimX Oli EDU is a full-sized humanoid robot featuring 31 degrees of freedom (DOF) - including 6 DOF per leg, 7 per arm, 3 in the waist, and 2 in the neck - providing highly biomimetic motion performance.

Its body is constructed from aerospace-grade aluminium and titanium alloys, offering an optimal balance of strength and lightness. The robot weighs approximately 55 kg,

and key exterior areas are fitted with soft protective materials to improve impact and fall resistance.



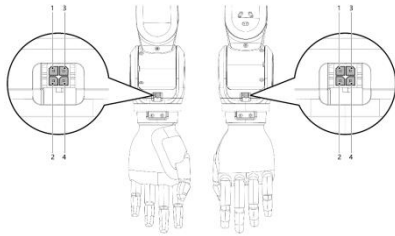
• **Rear Lower Electrical Interfaces**



No.	Interface type	Abbr.	Description
1	Type-C	Type-C	Supports USB3.2 Host 5V/1.5A Power Output
2	USB	USB	Supports USB3.2 Host 5V/1.5A Power Output
3	RJ45	1000 BASE-T	Gigabit Ethernet
4	RJ45	1000 BASE-T	Gigabit Ethernet
5	XT30PW - M30.G.Y	12V	12V/5A Power Output
6	XT30PW - M30.G.Y	12V	12V/5A Power Output
7	XT30PW	24V	24V/5A Power Output

	-F20.G.Y		
8	XT30PW -F20.G.Y	24V	24V/5A Power Output

- **Inner/Outer Hand Interface**



N o.	Interface type	Abbr.	Description
1	MX3.0	DC +24V	Power Positive Terminal 24V
2	MX3.0	DC GND	Power Negative Terminal GND
3	MX3.0	RS48 5_A	RS-485 Differential Positive Logic Line
4	MX3.0	RS48 5_B	RS-485 Differential Negative Logic Line

The inner hand interface is only compatible with BrainCo's Revo 2 dexterous hand.

The outer hand interface is only compatible with Inspire-Robots' EG2-4C2 electric gripper.

1.2 Onboard Computer

Oli EDU is equipped with 1 「Development Computing Unit」 and 1 「Motion Control Computing Unit」 as standard.

Parameters	Development Computing Unit
Model	Jetson Orin NX 16GB
AI performance	157 TOPS
GPU	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores

GPU max frequency	1173MHz
CPU	8-core Arm® Cortex®-A78AE v8.2 64-bit CPU 2MB L2 + 4MB L3
CPU max frequency	2 GHz
DL accelerator	2x NVDLA v2
DLA max frequency	1.23 GHz
Vision Accelerator	1x PVA v2
Memory	16GB 128-bit LPDDR5 102.4GB/s

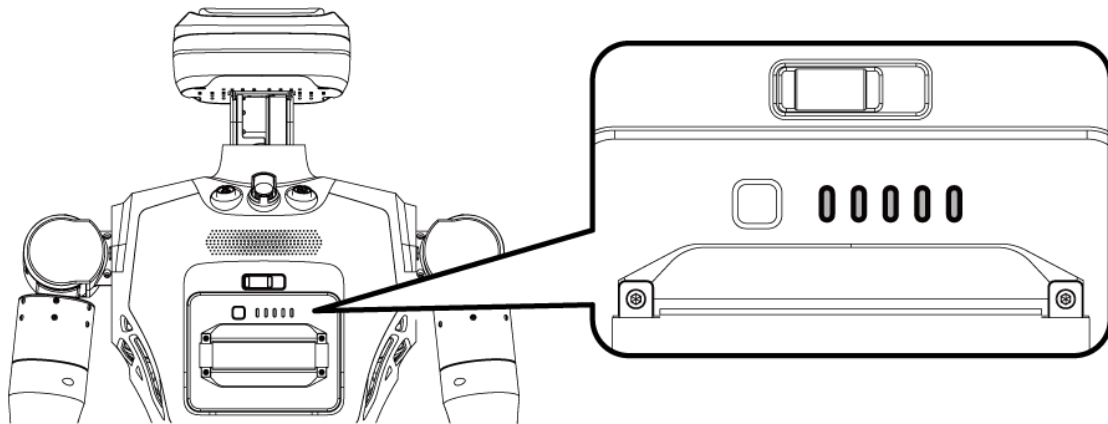
Jetson Orin NX Network Setup:

To establish a stable network connection for the Orin NX module, please follow the steps below:

1. **Prepare a Router:** Ensure a fully functional router is available, equipped with both WAN and LAN ports.
2. **Connect the Robot to the Router:** Use an Ethernet cable to connect the router's LAN port to the robot's lower rear Ethernet port.
3. **Connect the Router to the Internet:** Connect the router's WAN port to an active internet line.
4. **Configure Router LAN Subnet:**
 - a. Access the router management interface and set the LAN subnet to `10.192.1.x`
 - b. Assign the router's LAN IP address to `10.192.1.1`.

Upon completing these steps, the Orin NX module will be able to access external networks through the router.

1.3 Battery Indicator Status



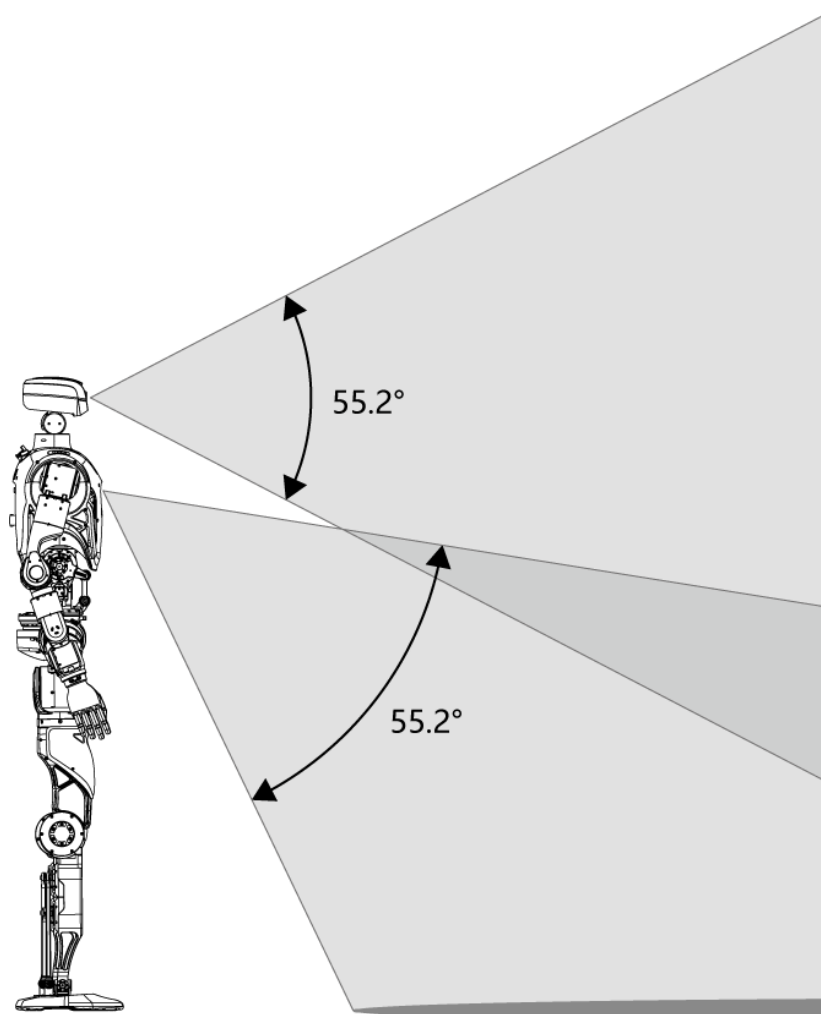
Indicator Light Status (Discharge)	Battery Level	Description
5 green LEDs on	100%	Fully charged
4 green LEDs on, 1 half-lit	81%-99%	Sufficient power
3 green LEDs on, 1 half-lit	61%-80%	Sufficient power
2 green LEDs on, 1 half-lit	41%-60%	Moderate power
1 green LED on, 1 half-lit	21%-40%	Low power, check battery
1 red LED blinking slowly	6%-20%	Insufficient power, recharge soon
1 red LED blinking rapidly	0%-5%	Critically low power, recharge immediately

Indicator Light Status (Recharge)	Battery Level	Description
1 green LED blinking rapidly	0%-5%	Initial charging stage, critically low power
1 green LED blinking slowly	6%-20%	Early charging stage, Insufficient power

1 green LED on, 1 blinking slowly	21%-40%	Low power; not recommended to disconnect
2 green LEDs on, 1 blinking slowly	41%-60%	Moderate power, suggested continuing charging.
3 green LEDs on, 1 blinking slowly	61%-80%	Sufficient power, suggested continuing charging.
4 green LEDs on, 1 blinking slowly	81%-99%	Sufficient power, suggested continuing charging.
5 green LEDs on	100%	Fully charged, safe to disconnect power.

1.4 Field of View of Cameras

Limx Oli EDU is equipped with **Intel RealSense D435i depth cameras** on its head and chest, providing exceptional visual perception capabilities that enable accurate sensing and understanding of the surrounding environment.



1.5 Joint Index and Joint Limits

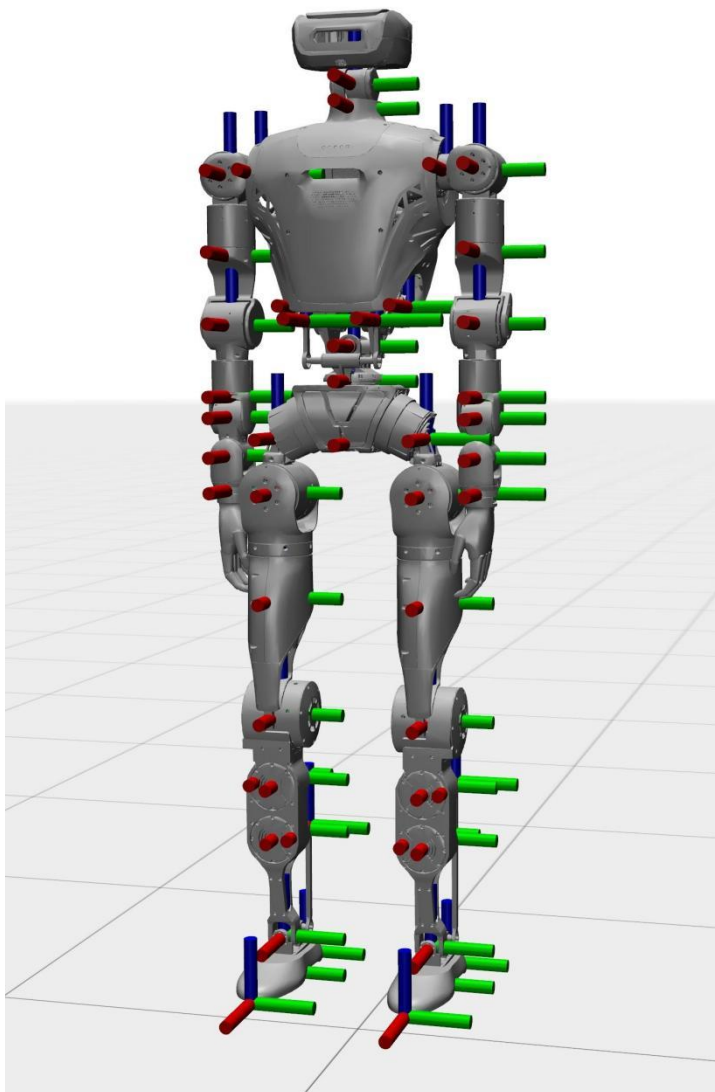
Joint Index	Joint Name	Limit (rad)
1	left_hip_pitch_joint	-2.617994 ~ 2.617994
2	left_hip_roll_joint	-0.872665 ~ 2.094395
3	left_hip_yaw_joint	-2.617994 ~ 2.617994
4	left_knee_joint	-0.087266 ~ 2.583087
5	left_ankle_pitch_joint	-1.012291 ~ 0.610865
6	left_ankle_roll_joint	-0.436332 ~ 0.436332
7	right_hip_pitch_joint	-2.617994 ~ 2.617994
8	right_hip_roll_joint	-2.094395 ~ 0.872665
9	right_hip_yaw_joint	-2.617994 ~ 2.617994
10	right_knee_joint	-0.087266 ~ 2.583087
11	right_ankle_pitch_joint	-1.012291 ~ 0.610865
12	right_ankle_roll_joint	-0.436332 ~ 0.436332
13	waist_yaw_joint	-2.617994 ~ 2.617994
14	waist_roll_joint	-0.523599 ~ 0.523599
15	waist_pitch_joint	-0.523599 ~ 0.785398
16	head_yaw_joint	-0.785398 ~ 0.785398
17	head_pitch_joint	-0.523599 ~ 0.785398
18	left_shoulder_pitch_joint	-2.792527 ~ 2.792527
19	left_shoulder_roll_joint	-0.087266 ~ 3.228859
20	left_shoulder_yaw_joint	-3.403392 ~ 1.570796

21	left_elbow_joint	-2.443 ~ 1.187
22	left_wrist_yaw_joint	-2.356194 ~ 0.785398
23	left_wrist_pitch_joint	-1.570796 ~ 1.570796
24	left_wrist_roll_joint	-1.570796 ~ 1.570796
25	right_shoulder_pitch_joint	-2.792527 ~ 2.792527
26	right_shoulder_roll_joint	-3.228859 ~ 0.087266
27	right_shoulder_yaw_joint	-1.570796 ~ 3.403392
28	right_elbow_joint	-2.443 ~ 1.187
29	right_wrist_yaw_joint	-0.785398 ~ 2.356194
30	right_wrist_pitch_joint	-1.570796 ~ 1.570796
31	right_wrist_roll_joint	-1.570796 ~ 1.570796

1.6 Coordinate Systems, Joint Rotation Axes, and Joint Zero Points

When all joints are at 0 °, the coordinate systems are shown below.

The **red**, **green**, and **blue** axes represent the **X**, **Y**, and **Z** axes, respectively.



1.7 Specifications

Model	Oli EDU
Mechanical Parameters	
Height	≈165cm / 5'5"
Shoulder Width	≈55cm / 21.7in
Arm Length	≈70cm / 27.6in
Weight (Battery Included)	≈55kg / 121.3lbs

Active DoF (Total) [1]	31 ~ 43
Single Leg DoF	6
Single Arm DoF	7
Waist DoF	3
Neck DoF	2
End Effectors	
Hand Model	□
2-Finger Gripper (1 DoF)	□
5-Finger Hand (6 DoF)	o
Self-Developed 3-Finger Gripper (4 DoF)	o
Joint Range of Motion	
Neck Joint	P-30° ~ 45°、Y±45°
Shoulder Joint	P±160°、R-5°~185°、Y-90°~195°
Elbow Joint	P-66°~137°
Wrist Joint	P±90°、R±90°、Y-45°~135°
Waist Joint	P-30°~45°、R±30°、Y±150°
Hip Joint	P±150°、R-50°~120°、Y±150°
Knee Joint	P+5°-148°
Ankle Joint	P-35°~58°、R±25°
Controller	

Remote Controller	☐
Thermal Management	
Air-Cooled Main Control Area	☐ (Optimized)
Communication Interfaces	
WiFi 6	☐
Bluetooth	☐
USB3.0/3.2 (USB Hub Supported)	☐
Gigabit Ethernet (RJ45)	☐
Power Supply Interfaces	
24V 5A	☐ (x2)
12V 5A	☐ (x2)
Perception Sensors	
IMU	6-Axis
Head Depth Camera	☐
Chest Depth Camera	☐
Battery	
Battery Capacity	9500mAH
Battery Life [2]	About 1.5H
Charger	58.8V 10A
Slide-Out Battery Module	☐
Capability	

Maximum Load Capacity (Single Arm) [3]	3kg / 6.6lbs
Maximum Moving Speed	5km/h
Maximum Joint Torque [4]	150N·m
Computing Configuration	Motion Control: • RK3588 SoC + 8G RAM+ 64G Storage Perception : • Orin NX SoC+16G RAM+1T Storage+157TOPS
Feature	
Voice Interaction Module	☐
LED Indicators	☐
System Status Monitoring	☐
Basic Motion Library [5]	☐
Motion Library Expansion Service	☐
Developer Tools	
Custom Development	☐
Remote Control API	☐
Sensor API	• Visual Perception Data • IMU Data
Low-Level Motion Control API [6]	☐
High-Level Motion Control API [7]	☐
OTA Updates	☐
Studio Package (Optional) [8]	o

Note:

***Products are continuously iterated and optimized; appearance and configuration are subject to change. Please refer to the actual product received.**

[1] Total Degrees of Freedom (DoF) refers to the combined DoF of the robot's base body and the installed end-effector (e.g., hand model, two-finger gripper, three-finger gripper, or five-finger dexterous hand).

[2] The data above was measured at the LimX Dynamics laboratory. Actual results may vary depending on environment, usage, device status, and software version. Please refer to the actual operating performance.

[3] The arm's maximum load varies with its extension and posture. Values may differ under different configurations. Please refer to the actual operating performance.

[4] The maximum torque differs among motors. The value shown represents the maximum torque of the highest-torque motor.

[5] The action library includes standing, humanoid walking, waving, bowing, punching, modern dance, and ethnic dance.

[6] Supports both joint-level control and end-effector control.

[7] Supports humanoid walking, mobile operation, stationary operation, and remote operation.

[8] Supports hardware status monitoring, voice configuration, data collection, motion choreography, and one-click simulation verification.

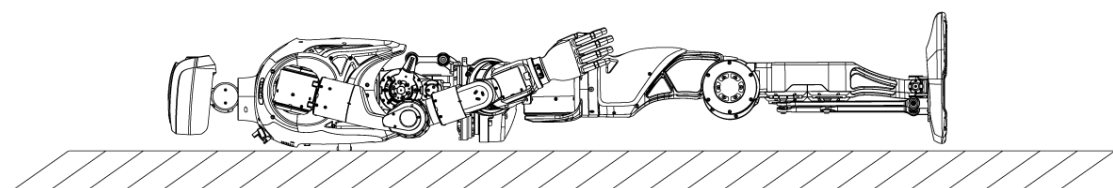
2. Operation guide @Emma

2.1 Startup Procedure

2.1.1 Powering On in Lying Position

Step 1: Positioning the Robot

Ensure Oli is placed flat on a stable and spacious surface. Keep its arms and legs naturally positioned.



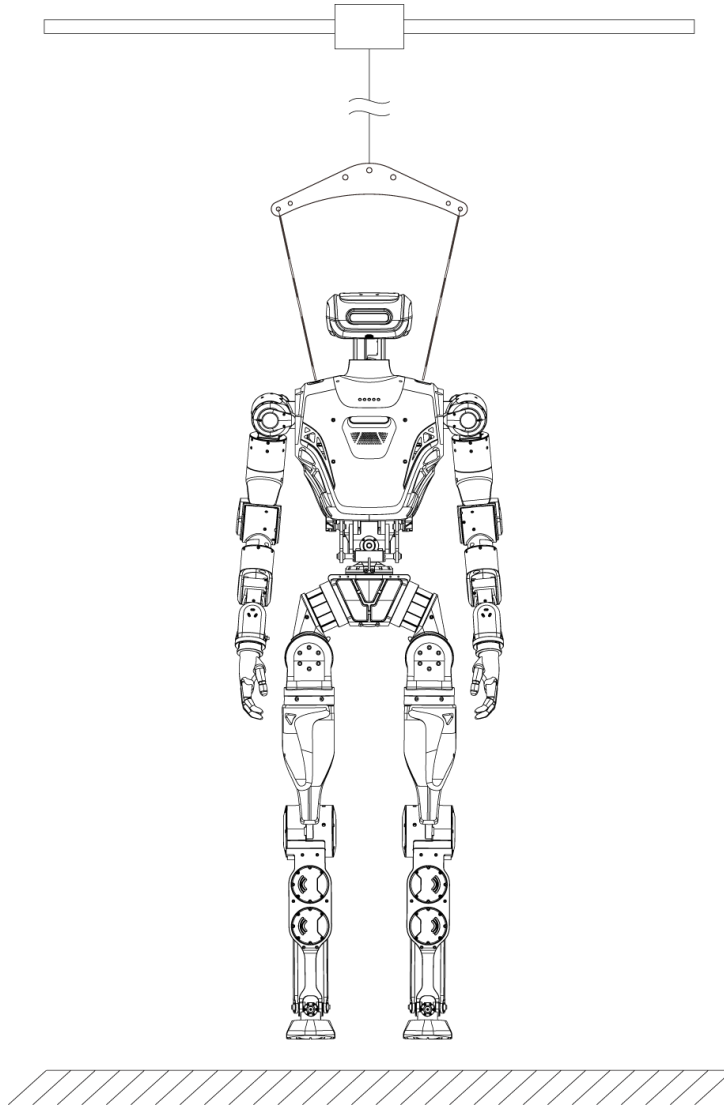
Step 2: Powering On

1. Briefly press and release the power button, then press and hold it for 2 seconds to power on.
2. The chest LED will display a blue breathing light, accompanied by the voice prompt "System starting".
3. When the LED turns solid white, Oli enters **Zero Torque Mode**:
 - a. Press [**L2 + X**] to switch to **Damping Mode**.
 - b. Press [**R1 + Left**] to switch to **Remote Mode** (Oli remains in damping mode), then press [**L2 + Δ**] to command Oli to stand up and enter **Standing Mode**.

2.1.2 Powering On in Hanging Position

Step 1: Suspend the Robot

Secure Oli using the protective frame in a natural hanging position. Ensure the feet are clear of the ground, with a minimum clearance of 15 cm.



Step 2: Powering On

1. Briefly press and release the power button, then press and hold it for 2 seconds to power on.
2. The chest LED will display a blue breathing light, accompanied by the voice prompt "System starting".
3. When the LED turns solid white, Oli enters **Zero Torque Mode**:
 - a. Press **[L2 + X]** to switch to **Damping Mode**.
 - b. Press **[R1 + Left]** to switch to **Remote Mode** (Oli remains in damping mode), then press **[L2 + Δ]** to command Oli to stand up and enter **Standing Mode**.

Step 3: Lowering the Suspension

After Oli enters Standing Mode, gradually lower the suspension rope until both feet are firmly on the ground.

Step 4: Detach the Suspension

Once Oli maintains stable motion, carefully detach the suspension rope. Oli is now

ready for operation.

- If Oli encounters an unexpected condition, press **[L2 + X]** to switch to **Damping Mode**. Oli will gently lower itself to the ground in a controlled manner.
- If the battery level is below **5%** during startup, the system will prevent normal powering on. Please charge it immediately.

2.1.3 Power On Self-Test

During power-on, Oli will automatically perform a self-test. The chest LED will transition from a blue breathing light to a slow white flash:

- If a malfunction is detected, the chest LED will flash red, and a voice prompt will announce the specific faulty component.
- If everything is functioning well, the chest LED will transition from a blue breathing light to a solid white, with the voice prompt "System ready".

Power-On Self-Test Status Description

Status Phase	Chest LED State	Voice Prompt	Description
Power-on → Self-testing	Breathing Blue	"System starting"	The system is starting and preparing for a self-test.
Self-testing (Transition)	Slow-flashing White	-	Self-test in progress.
Self-test → function well	Solid White	"System Ready"	The system functions well and is ready for operation.
Self-test → Malfunction	Flashing Red	Specific component fault (e.g., "Right knee motor error")	A malfunction has been detected; troubleshooting or repair is required.

2.1.4 Remote Mode

When Oli is in Zero Torque Mode, press **[R1 + Left]** to switch to **Remote Mode**.

2.1.5 Developer Mode

- **Switching to Developer Mode**

When Oli is suspended and in **Zero Torque Mode**:

- a. Press **[R1 + Down]** to switch to **Low-Level Developer Mode** → access low-level interfaces.
- b. Press **[R1 + Up]** to enter **High-Level Developer Mode** → access high-level interfaces.

- **Exiting Developer Mode**

While in Developer Mode, press **[L2 + ○]** to exit.

2.2 Shutdown Procedure

2.2.1 Shutdown in Lying Position

Ensure Oli is stably positioned on the ground. Briefly press and release the power button, then press and hold for **2 seconds** to initiate shutdown. Oli will announce **"System shutting down"**.

2.2.2 Shutdown in Standing Position

Regardless of Oli's current posture (standing, walking, performing an action, or other), first transition it to a lying position (gently lower the limbs to ensure stable contact with the ground), then execute the shutdown procedure as described above.

- If performing any action, press **[L2 + Δ]** to return to Standing Mode, then press **[L2 + □]** to switch to Lying Mode.
- If in Standing Mode, press **[L2 + □]** to switch to Lying Mode.

2.2.3 Shutdown in Hanging Position

Press **[L2 + Δ]** to switch to Standing Mode, or press **[L2 + X]** to switch to **Damping Mode**, then execute the shutdown procedure.

2.3 Remote Control Instructions

2.3.1 User Manual

Please visit the Limx Dynamics Support Center - [Documentation] to access the *Humanoid Robot Remote Control User Manual*.

- The remote controller is pre-paired with the robot before shipment and is ready for use immediately after startup.
- To control another Oli robot, please refer to the "Remote Controller Pairing" tutorial video available on the Limx Dynamics Support Center.
- If the remote controller has been paired with another robot, it must be re-paired with the original unit to regain control.

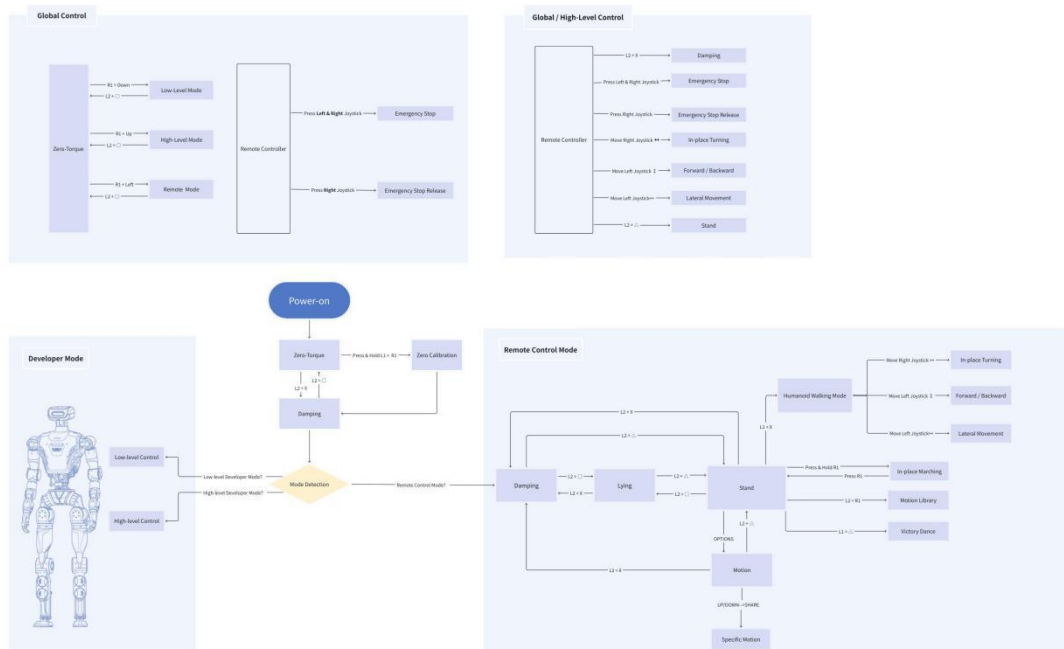
2.3.2 Motion Control Status, Chest LED and Buzzer Indications

Control Status / Mode	LED State	Buzzer Prompt
Powering On	□ White slow flash	—
Zero Torque Mode	□ Solid white	—
Remote Mode	□ Solid blue	—
Developer Mode	□ Solid green	—
Emergency Stop	□ Orange fast flash	—
Low Battery Warning ($\leq 20\%$)	□ Red slow flash	—
Critical Battery Warning ($\leq 5\%$)	□ Red fast flash	Intermittent "beep-beep"
System Abnormal Warning	□ Solid red	—
System Critical Fault	□ Red fast flash	Continuous "beep-beep-beep"
Voice Wake-up (Standby)	□ White breathing light	—
Voice Recognition (Receiving)	□ Blue breathing light	—
Voice Response	□ Green breathing	—

(Speaking)

light

2.3.3 Mode Switching



- Before switching modes, ensure the robot is in Zero Torque Mode, then follow the global control instructions to switch modes.
- When switching to Remote Mode, the robot automatically enters Damping Mode; when switching to Developer Mode, it defaults to Low-level Control Mode.
- Before switching to High-level or Low-level Control Modes, please enter Developer Mode first.
- Oli will automatically enter Zero Torque Mode on startup if no mode is set, then switch to damping mode for mode selection. The selected mode will be automatically saved and loaded at the next startup.
- The global/high-level control buttons remain functional regardless of the robot's current state.

Please operate the robot on a flat, level surface when performing forward/backward movement, lateral movement, or in-place turning/marching to prevent potential damage.

2.3.4 Button Functions

Type	Mode/Motion	Description	Button	Prerequisite States
Mode /State Switching	Remote Mode	Enables robot movement/action control via the remote controller. (Voice interaction is available in this mode.)	R1 + Left	—
	Developer Mode - High-Level Control	Allows advanced development, including high-level command execution and low-level joint control. Ensure Oli is securely suspended using a protective frame or equivalent equipment during development.	R1 + Up	Developer Mode only
	Developer Mode - Low-Level Control		R1 + Down	Developer Mode only
	Zero-Torque Mode	All joints are torque-free and can be manually adjusted.	L2 + ○	—
	Zero-Point Calibration	All joints reset to the mechanical zero position.	Press & Hold L1 + R1	Zero Torque Mode
	Damping Mode	Applies damping torque to joints to prevent free swinging.	L2 + X	Zero Torque Mode
	Lying Mode	Transitions the robot to a lying position (following a zero-torque or damping transition).	L2 + □	Damping Mode / Standing Mode
	Standing Mode	Engages preload torque to maintain balance and await commands.	L2 + △	—
Move	Move	Moves the robot forward	Left	Humanoid Walking

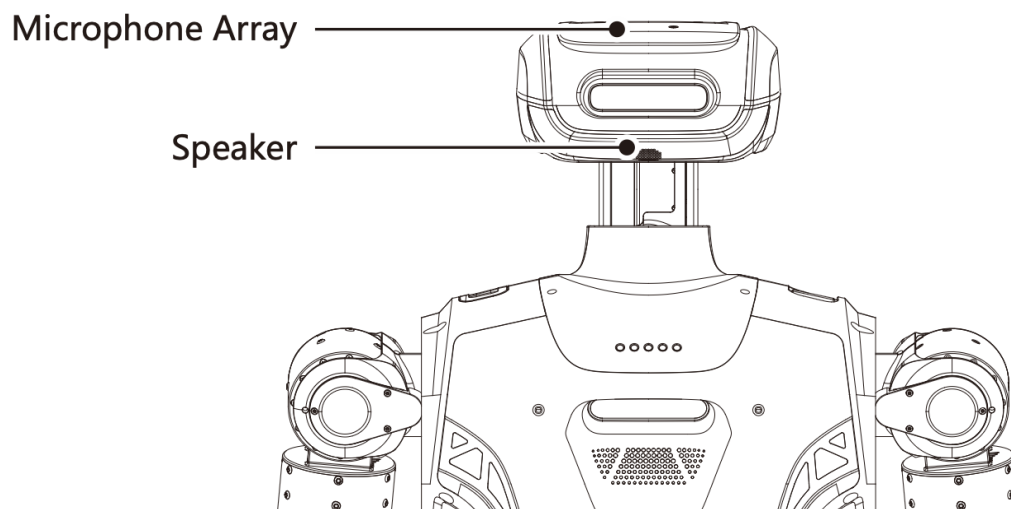
ment Control	Forward/Backwards	or backwards via the remote controller.	Joystick ↑↓	
	Lateral Movement	Moves the robot left or right via the remote controller.	Left Joystick ↔	Humanoid Walking
	In-place Turning	Rotates the robot on the spot via remote controller.	Right Joystick ↔ □	Humanoid Walking
	In-place Marching	Performs marching motion in a standing position.	Press & Hold R1	Standing Mode
	Humanoid walking	Enables natural bipedal walking control, ready to receive control commands.	L1 + X	Standing Mode
Interactive Actions/ Functions	Victory Dance	Performs a built-in victory dance via the remote controller.	L1 + Δ	Standing Mode
	Motion Library Access	Access the motion library menu via the remote controller.	Press OPTIONS	Standing Mode
	Motion Switching	Selects and switches between actions in the motion library.	Press UP / DOWN	Standing Mode
	Motion Select & Execute	Executes a customized motion via the remote controller.	Press SHARE	Standing Mode
	Custom Motion Sequence	Allows creation of custom motion sequences (e.g., dance routines).	L2 + R1	Standing Mode
	Voice	Forces activation of voice	Press	—

	Wake-Up	interaction.	& Hold R2	
	Teleoperation Initialization	Switches to teleoperation and begins initialization via the remote controller.	L2 + R2 + △	Standing Mode

Prerequisite States: Indicates that switching to a new action or state is only permitted when the robot is in the required preceding state.

2.4 Voice Interaction Instructions

Once connected to the Internet, the voice interaction feature enables conversational control and action execution. If the network connection fails, offline dialogue mode remains available.



2.4.1 Mode Description

- **Online Mode:** Supports voice recognition, action control, and advanced conversational interaction.
- **Offline Mode:** Supports basic command execution (e.g., posture adjustment, simple Q&A).
- **Mode Switching:**
 - Automatically switches to Offline Mode when the network is disconnected or the detected network speed drops below 1 Mbps.

- Automatically switches to Online Mode once the network connection is restored, accompanied by the voice prompt: "Network connected. Switched to Online Mode."

2.4.2 Wake-Up Methods

Oli currently supports wake-up dialogue mode only.

- **Remote Controller Wake-up:** Press and hold the R2 button
- **Voice Wake-Up:** "Hi, <Robot Name>" (e.g., "Hi, Oli").
 - The robot's name can be customized, limited to 2-4 Chinese characters or 2-4 English words.
- **Wake-Up Feedback:** Oli will automatically turn toward the speaker and respond, "I'm here, please go ahead." The chest LED changes to a green breathing light.

LED Status after Wake-Up:

Voice Operation Status	Chest LED Colour
Standby	White
Voice Recognition	Blue
Voice Response	Green

2.4.3 Dialogue Mode

- After a successful wake-up, Oli supports continuous, multi-turn conversations. The dialogue will automatically end if no voice input is detected for **15 seconds**, requiring a re-wake-up.
- If the response is interrupted by other sounds, the current reply will be terminated, and a new dialogue session will begin.

2.4.4 Dialogue Roles

- **Supports switching between predefined dialogue roles:**
 - **Retail Store:** Sales, Promotion, and Customer Guidance
 - **Exhibition Hall:** Commentary, Interaction, and Emergency Announcement

- **Switching Method:** Roles can be switched with a single click in the Studio interface, completing in 1 minute.

2.4.5 Dialogue Models

- Supports **multiple dialogue models**:
 - General Conversation Model
 - Professional Commentary Model (adapted for retail and exhibition halls)
- Supports **custom knowledge base**:
 - Upload knowledge base files via Studio (supported formats: `.doc`, `.txt`, `.pdf`)
 - Batch upload supported — up to 5 files per session, with a total size ≤ 20 MB
 - View the list of uploaded knowledge bases.
 - Uploaded knowledge bases take effect immediately and are prioritised in dialogue responses.

2.4.6 Multilingual Switching

- Supports recognition and responses in **Chinese** and **English**.
- Language Switching: Automatically switches based on detected language or via voice command (e.g., “Switch to English”).

Please visit the **LimX Dynamics Support Center – [Download Center]** to download and install the **Studio software**.

2.4.7 Interaction Deactivation

The program will enter sleep mode and stop recording. Once deactivated, no privacy risks will occur.

Notes:

1. In noisy environments, if the chest LED does not display a blue breathing light while speaking, it indicates a failed voice reception. In this case, please move closer to the robot and speak again.
2. When the battery level drops below 20%, non-core response functions will be automatically disabled.

2.5 Motion Library Interaction Instructions

2.5.1 Motion Library Overview

1. Motion Library List

Category	Motion ID	Motion Name
Basic Interactions	1	This way, please
	2	Bow
	3	Wave
	4	Nod
	5	Shake one's Head
	6	Take a bow
Gestures	7	Blow kisses
	8	Left-Hand Heart
	9	Right-Hand Heart
	10	Hand Heart
	11	High-Five Someone
	12	Clap
Dance	13	Warm-Up Dance
	14	Swag Dance
	15	Idol Dance 1
	16	Idol Dance 2

	17	Power-up Dance
--	----	----------------

The motion library will be continuously updated, and the latest content can be obtained directly via **OTA updates**.

2. Motion Library Display

- All motions can be viewed and operated through **Studio software**.

2.5.2 Studio Connection and Usage

1. Connecting the Robot to the Studio

The robot can be connected to Studio via Wi-Fi or a data cable.

2. Local Operations

All operations – including motion editing, generation, preview, simulation safety verification, and saving – can be performed locally within Studio, without requiring a connection to the robot.

3. Motion Sequence Editing

Studio allows atomic motions to be combined into continuous motion sequences. The sequence order is fully customizable.

4. Motion Sequence Transmission

To transfer verified motion sequences to the robot, ensure that Studio is connected to the robot. The transmission process is as follows:

- Connect the Robot:** Connected to the robot via Wi-Fi or a data cable.
- Verify Connection Information:** Once connected, Studio will display robot information, including model, remaining storage, connection status, and operational status.
- Transfer Motion Sequence:** Upload the verified motion sequence to the robot.
- Disconnect:** The connection can be safely disconnected once the motion sequence transmission is complete.

2.5.3 Motion Execution

All robot motions need to be executed using the designated **remote controller**. For detailed operation instructions, please refer to the *"Humanoid Robot Remote Controller User Manual"*.

3. SDK Development Guide @Eleven

3.1 Large Model API Interface

3.1.1 Built-in Models

Model	Description
qwen2.5:3b	Developed by Alibaba Cloud, the Qwen 2.5 series model features 3 billion parameters, offering robust language understanding and generation capabilities, suitable for text generation and dialogue interaction.
qwen2.5:1.5b	A lighter 1.5 billion-parameter model from the Qwen 2.5 series, providing solid performance in resource-efficient scenarios with moderate computational demands.
qwen2.5:0.5b	A lightweight 0.5 billion-parameter model optimized for terminal devices or environments with limited computational resources.
llama3.2:3b	A 3 billion-parameter model from Meta’s LLaMA 3.2 series. It excels across a wide range of natural language processing tasks. Its open-source nature allows developers to perform secondary development and customization.
llama3.2:1b	A 1 billion-parameter model from the LLaMA 3.2 series, offering faster training and inference speed due to its smaller model size.
deepseek-r1:1.5b	Developed by DeepSeek, this 1.5 billion-parameter model delivers competitive performance across multiple language processing tasks.

3.1.2 Model invocation

Oli has pre-deployed the above large models locally via **Ollama**. To invoke them, connect your device to the same network as the robot and follow the instructions below.

1. Invoke via Curl

```
c++
curl http://10.192.1.3:11434/api/generate \
  -H "Content-Type: application/json" \
  -d '{
    "model": "qwen2.5:3b",
    "prompt": "Please write a quatrain describing spring?",
    "temperature": 0.7,
    "max_tokens": 200,
    "stream": false
  }'
```

2. Invoke via Python

```
python
import requests

url = "http://10.192.1.3:11434/api/generate"
data = {
    "model": "qwen2.5:3b",
    "prompt": "Please write a quatrain describing spring?",
    "temperature": 0.7,
    "max_tokens": 200,
    "stream": False
}

response = requests.post(url, json=data)
if response.status_code == 200:
    print(str(response.json()['response']))
else:
    print(f"Request failed with status code:
{response.status_code}, Error: {response.text}")
```

3. Invoke via C++

Using **Ubuntu 20.04** or later as an example:

- **Install Dependencies**

```
python
sudo apt-get install libcurl4-openssl-dev nlohmann-json3-dev
```

- **Implement Code** (llm_demo.cpp)

```
c++
#include <iostream>
#include <string>
#include <curl/curl.h>      // HTTP client library
#include <nlohmann/json.hpp> // JSON parsing

using json = nlohmann::json;

// Callback to handle HTTP response data
static size_t WriteCallback(void* data, size_t size, size_t nmemb,
std::string* buf) {
    buf->append((char*)data, size * nmemb);
    return size * nmemb;
}

int main() {
    CURL* curl = curl_easy_init();
    if (!curl) {
        std::cerr << "CURL init failed" << std::endl;
        return 1;
    }

    // 1. Configure API endpoint
    const std::string url =
"http://10.192.1.3:11434/api/generate";

    // 2. Prepare JSON payload
    json req = {
        {"model", "qwen2.5:3b"},
        {"prompt", "Please write a quatrain describing spring?"},
        {"temperature", 0.7},
        {"max_tokens", 200},
        {"stream", false}
    };
    std::string payload = req.dump();
```

```

// 3. Set CURL options
curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
curl_easy_setopt(curl, CURLOPT_POSTFIELDS, payload.c_str());
curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, payload.size());

// 4. Add HTTP headers
struct curl_slist* headers = nullptr;
headers = curl_slist_append(headers, "Content-Type:
application/json");
curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

// 5. Capture response
std::string response;
curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response);

// 6. Execute request
CURLcode res = curl_easy_perform(curl);

// 7. Process result
if (res == CURLE_OK) {
    try {
        json resp_json = json::parse(response);
        std::cout << "Result: " << resp_json["response"] <<
std::endl;
    } catch (const json::exception& e) {
        std::cerr << "JSON error: " << e.what() << std::endl;
    }
} else {
    std::cerr << "HTTP error: " << curl_easy_strerror(res) <<
std::endl;
}

// 8. Cleanup
curl_slist_free_all(headers);
curl_easy_cleanup(curl);
return 0;
}

```

- **Compile and Run**

```

C++
# Compile with C++11 support
g++ -std=c++11 -o llm_demo llm_demo.cpp -lcurl

```

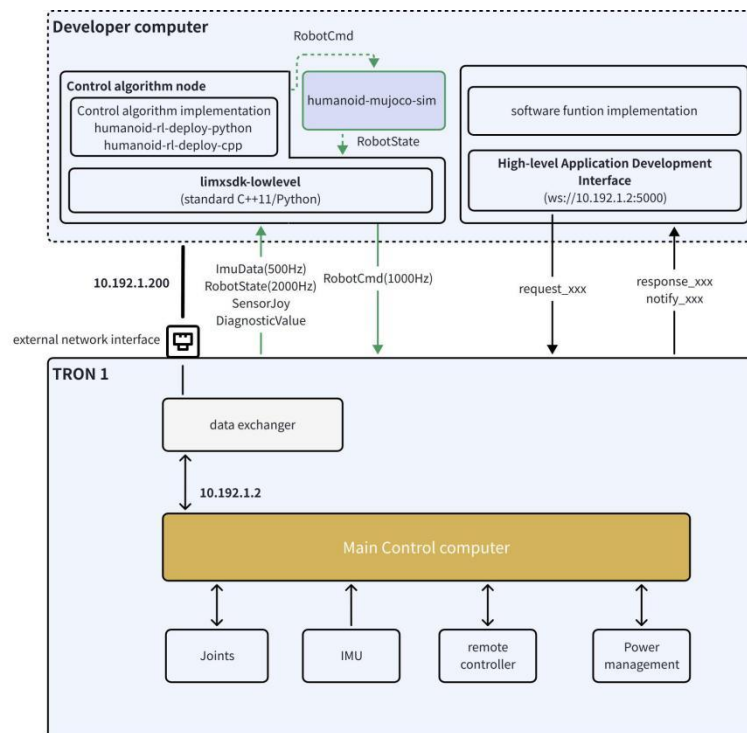
```
# Execute
./llm_demo
```

3.2 Communication Architecture

The diagram below illustrates the system composition and interaction between the developer's computer and the robot.

The development computer includes the motion control algorithm node and software business logic implementation module, which control the robot's motion via the high-level application protocol interface and `limxsdk-lowlevel` data communication interface.

The robot consists of a data switch, a main control computer, and various hardware components. The main control computer is responsible for coordinating the operation of all components and ensuring synchronized performance..



3.3 MCP Server Interface

3.3.1 Overview

MCP (Model Context Protocol) is a unified protocol introduced by Anthropic (the company behind *Claude*) in 2024. It enables large language models (LLMs) to interact securely and consistently with external tools, data sources, and services.

Oli MCP Server is a robot control service built on this standard protocol. Through the MCP interface, AI assistants such as Claude, Cursor, and others can directly control Oli to perform a wide range of actions and tasks.

3.3.2 Connecting to the MCP Service

1. Service Address

The MCP service is provided via an HTTP interface. The default service address is:

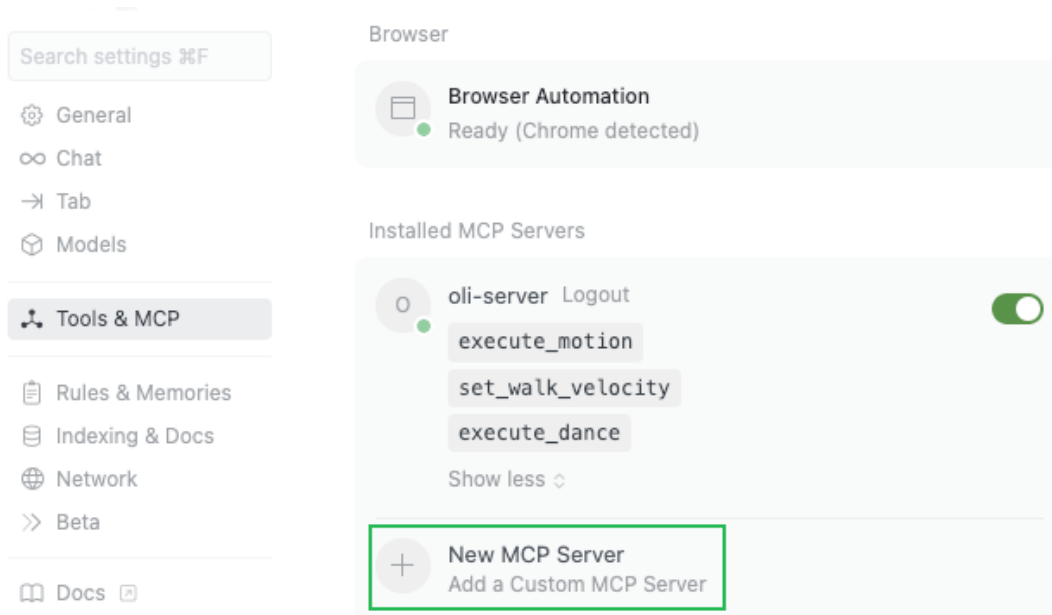
```
http://10.192.1.2:18080/mcp
```

2. Configure the MCP Client

In an MCP-supported tool, modify the MCP configuration file as follows:

```
JSON
MCP Service Configuration
{
  "mcpServers": {
    "oli-mcp-server": {
      "url": "http://10.192.1.2:18080/mcp",
      "disabled": false,
      "autoApprove": [],
      "headers": {
        "Accept": "application/json, text/event-stream",
        "Content-Type": "application/json"
      }
    }
  }
}
```

- Example (in Cursor):



3.3.3 MCP Tool Interface Description

Tool Name	Function	Main Parameters
execute_motion	Executes predefined robot actions. The robot automatically enters and exits Motion Engine Mode.	• motion_name (Required): ◦ stand ◦ this_way_please ◦ bow ◦ wave_greet_bye ◦ nod ◦ shake_head ◦ curtain_bow ◦ blow_kisses_multi ◦ left_hand_side_heart ◦ right_hand_side_heart ◦ raised_hand_heart ◦ high_five ◦ clap ◦ warm_up_dance

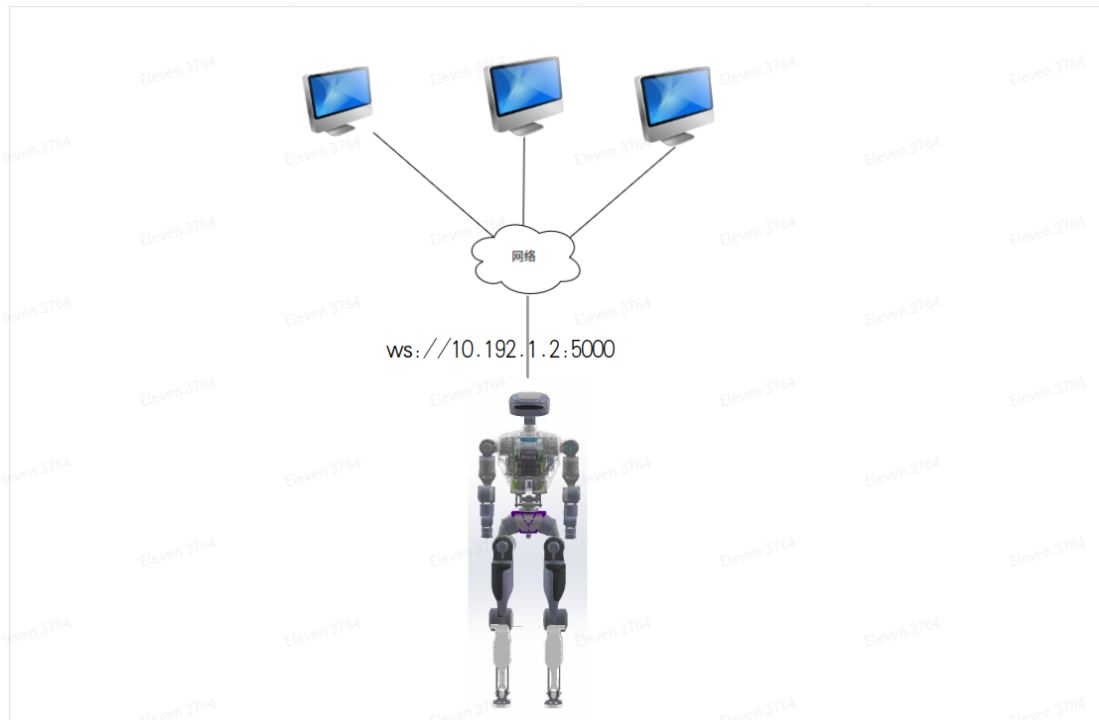
		◦ swag_dance ◦ idol_dance_1 ◦ idol_dance_2 ◦ power_up_dance
set_walk_velocity	Controls the robot's walking velocity and automatically switches to Walking Mode.	• x (Required, [-1,1]): Forward/backward velocity (positive = forward, negative = backward) • y (Required, [-1,1]): Lateral velocity (positive = right, negative = left) • yaw (Required, [-1,1]): Rotational velocity (positive = counterclockwise, negative = clockwise)
execute_dance	Executes a predefined robot dance action.	• dance_name (Required): ◦ victory

3.4 High-Level Application Protocol Interface

The robot communicates with the user terminal through a WebSocket connection on port 5000 to receive user commands such as stand, sit, walk, and other motion instructions.

WebSocket is a real-time communication protocol that establishes a persistent connection between the robot and the user terminal, enabling fast and efficient transmission of control commands and data.

The communication structure is illustrated in the diagram below:



3.4.1 Coordinate System Description

- Unless otherwise specified, the position and orientation of both arm end-effectors are defined with respect to the robot's base coordinate system.
- The base coordinate system is defined for humanoid robots with serial numbers starting with "HU".
- The origin of the base coordinate system corresponds to the `base_link` defined in the URDF file. The coordinate system follows the right-hand rule ([REP-103](#) standard).

3.4.2 Communication Protocol Format

When the robot receives commands from the client via WebSocket, data is transmitted using the JSON communication protocol.

1. Request Data

Fields in the request data format	Description
<code>accid</code>	The robot's unique serial number identifies its identity.

<code>title</code>	The command name, prefixed with "request_".
<code>timestamp</code>	The timestamp when the command is sent, in milliseconds.
<code>guid</code>	A unique command identifier. For synchronous interfaces, the "response_xxx" message includes the same <code>guid</code> , allowing the client to confirm command completion by matching values
<code>data</code>	Contains the content of the request. Depending on the command, it may include multiple sub-fields, such as parameters for motion execution, text content for messaging, or other required data.

Request Data Example:

Plain Text

```
{
  "accid": "HU_D02_001", # Robot's unique serial number
  "title": "request_xxx", # Command name, prefixed with
  "request_"
  "timestamp": 1672373633989, # Timestamp (in milliseconds) when
  the response was generated.
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Unique identifier
  for the request. Used to match the response for synchronous
  operations
  "data": {} # Contains command-specific parameters
}
```

2. Respond Data

Fields in the response data format	Description
<code>accid</code>	The robot's unique serial number identifies its identity.
<code>title</code>	The command name, prefixed with "response_".
<code>timestamp</code>	The timestamp when the command is issued, in milliseconds.

<code>guid</code>	Matches the <code>guid</code> value from the corresponding request command.
<code>data</code>	The response data must include at least a "result" subfield to store the request's execution result. Additional subfields, such as error code or error message, may be included as needed to provide details about the operation outcome.

Response Data Example:

```
JSON
{
  "accid": "HU_D02_001",    # Robot's unique serial number
  "title": "response_xxx",  # Command name, prefixed with
  "response_"
  "timestamp": 1672373633989, # Timestamp (in milliseconds) when
  the response was generated.
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Must match the
  guid value from the corresponding request command.
  "data": { # Used to store the specific data content of the
  response command.
    "result": "success" # "result" Indicates whether the request
    was processed successfully, the value can be: "success or
    fail_xxx"
  }
}
```

3. Message Push

The message push process refers to the robot actively sending information to the client. These messages may include the robot's serial number, current operational status, executed actions, and other relevant data.

By providing real-time updates, the robot enables the client to understand its working status better and make more effective use of its services.

Fields in the message push format	Description
<code>accid</code>	The robot's unique serial number identifies its identity.
<code>title</code>	The command name, prefixed with "notify_".

timestamp	The message timestamp, in milliseconds.
guid	The unique identifier of the message.
data	Contains the message data. May include multiple subfields depending on the specific requirements of the notification.

Message Push Example:

```
JSON
{
  "accid": "HU_D02_001",    # Robot's unique serial number
  "title": "notify_xxx",    # notification command name, prefixed
                           # with "notify_"
  "timestamp": 1672373633989, # The time the message was sent, in
                           # milliseconds.
  "guid": "746d937cd8094f6a98c9577aaf213d98", # The unique
                           # identifier for this notification message.
  "data": { } # Contains detailed status data
}
```

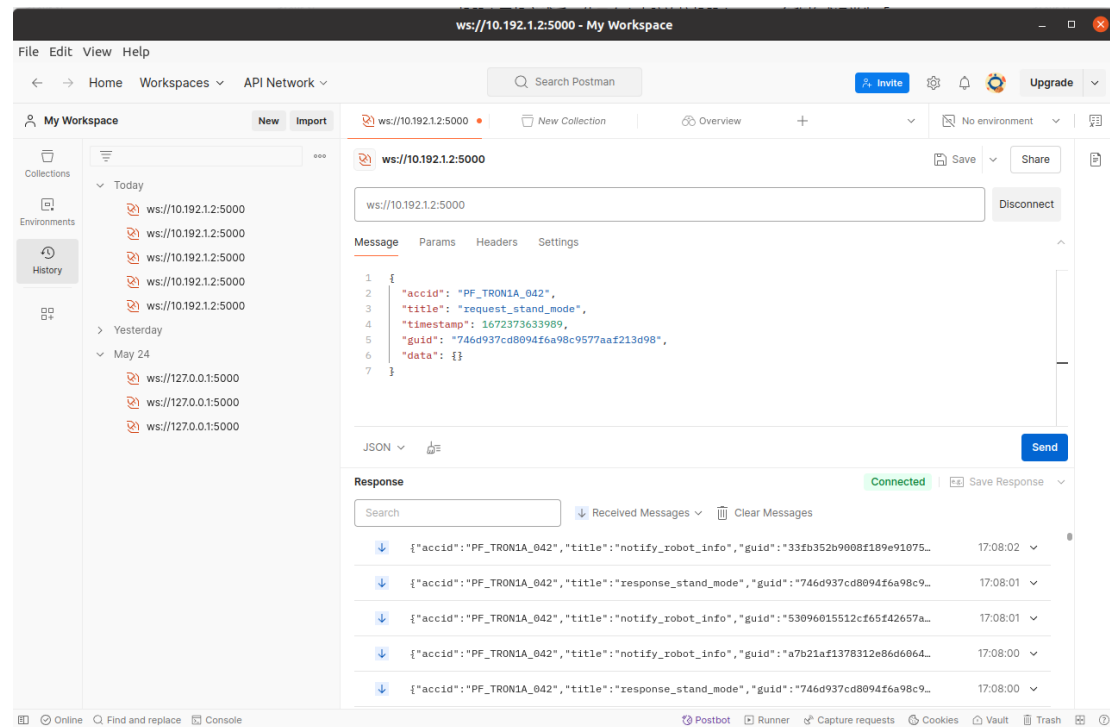
3.4.3 Communication Testing Method

Postman is a popular API development environment that can be used to test WebSocket interfaces.

Steps to Test the WebSocket Interface Using Postman:

1. **Install Postman:** Download address: <https://www.postman.com/downloads/>
2. **Create a WebSocket Request:** Launch Postman and create a new WebSocket request.
3. **Connect to the Robot's Wi-Fi Network:**
 - a. After the robot powers on successfully, connect your computer to the robot's Wi-Fi network. The network name typically follows the format: 「HU_D02_xxx」
 - b. Enter the Wi-Fi password: 12345678
4. **Enter the WebSocket URL:** In the request URL field, input the robot's WebSocket address, e.g.:
"ws://10.192.1.2:5000"
5. **Input the Command Request:** In the “**Message**” field, enter the JSON-formatted command request.
6. **Send the Command:** Click “**Send**” to transmit the request to the robot.

7. View the Response: After sending the command, the robot will return a response message. Review the response in Postman's output window to verify whether the result matches the expected outcome.



3.4.4 Basic Function Protocol Interfaces

1. Enter Ready Status

Controls the robot to enter the "Standing Mode", enabling it to receive velocity commands for walking control.

1.1 Request: request_prepare

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_prepare",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": { }
}
```

1.2 Response: response_prepare

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_prepare",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

1.3 Message Push: none

2. Enter Walking Mode

Sets the robot to Walking Mode, enabling it to receive velocity commands.

2.1 Request: request_set_walk_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

2.2 Response: response_set_walk_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_walk_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor: Motor Error
  }
}
```

```
}  
}
```

2.3 Message Push: none

3. Control Robot Walking

In Motion Operation Mode, use this protocol to control the robot's walking.

Note: This interface is invalid in Full-Body Operation Mode.

3.1 Request: request_set_walk_vel

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "request_set_walk_vel",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
    "x": 0.0,    # Forward/Backwards Speed Ratio, range[-1, 1]  
    "y": 0.0,    # Lateral Speed Ratio, range[-1, 1]  
    "yaw": 0.0   # Rotational Angular Velocity Ratio, range[-1, 1]  
  }  
}
```

3.2 Response: response_set_walk_vel

Returned only if the command execution fails; no response on success.

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "response_set_walk_vel",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
    "result": "fail_motor" # fail_imu: IMU Error, fail_motor:
```

```
Motor Error
  }
}
```

3.3 Message Push: none

4. Enter Damping Mode

All motors stop active control and exhibit resistance when moved.

4.1 Request: request_damping

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_damping",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

4.2 Response: response_damping

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "response_damping",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor: Motor Error
  }
}
```

4.3 Message Push: none

5. Enter Zero-Torque Mode

All motors stop active control and move freely without resistance.

5.1 Request: request_zero_torque

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_zero_torque",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

5.2 Response: response_zero_torque

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_zero_torque",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor: Motor Error
  }
}
```

5.3 Message Push: none

6. Zero-Point Calibration Command

6.1 Request: request_calibrate

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_calibrate",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
}
```

```
"data": {}  
}
```

6.2 Response: response_calibrate

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "response_calibrate",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
    "result": "success" # fail_motor: Motor Error  
  }  
}
```

6.3 Message Push: notify_calibrate

Pushed after calibration is completed.

Python

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "notify_calibrate",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
    "result": "success"  
  }  
}
```

7. Robot Dance

7.1 Request: request_dance

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",
```

```
"title": "request_dance",
"timestamp": 1672373633989,
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
  "name": "victory" # dance name
}
}
```

7.2 Response: response_dance

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_motion_engine",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

7.3 Message Push: notify_dance

Pushed when the dance completes or execution fails.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_dance",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

8. Robot Motion Library

8.1 Switching the Robot to Motion Library Mode

8.1.1 Request: request_set_motion_engine

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_motion_engine",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0: Exit Motion Library Mode
    # 1: Enter Motion Library Mode
    "mode": 0
  }
}
```

8.1.2 Response: response_set_motion_engine

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_motion_engine",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

8.1.3 Message Push: none

8.2 Executing Motion Library Actions

Actions can be executed once the robot is in Motion Library Mode. Supported atomic actions include:

Motion ID	Motion Name	Description
0	stand_still	Stand
1	this_way_please	This way, please

2	bow	Bow
3	wave_greet_bye	Wave(Greet/Bye)
4	nod	Nod
5	shake_head	Shake one's head
6	curtain_bow	Take a bow
7	blow_kisses_multi	Blow kisses
8	left_hand_side_heart	Left-hand heart
9	right_hand_side_heart	Right-hand heart
10	hand_heart	Hand Heart
11	high_five	High-five Someone
12	clap	Clap
13	warm_up_dance	Warm-up Dance
14	swag_dance	Swag Dance
15	idol_dance_1	Idol Dance1
16	idol_dance_2	Idol Dance2
17	power_up_dance	Power-up Dance

8.2.1 Request: request_execute_atomic_motion

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Motion Name
```

```
    "motion_name": "wave_greet_bye"
  }
}
```

8.2.2 Response: response_execute_atomic_motion

JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

8.2.3 Message Push: notify_execute_atomic_motion

Pushed when an action is completed or execution fails.

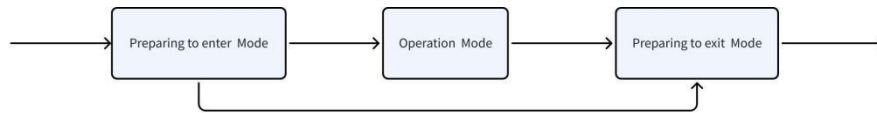
JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_execute_atomic_motion",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # fail_motor
  }
}
```

9. Motion Control

9.1 Switching to Motion Operation Mode

In Motion Operation Mode, walking can be controlled, while height and waist movements remain disabled.



9.1.1 Request: request_set_ub_manip_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_ub_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "mode": 0 # 0: Preparing to enter Mode 1: Operation Mode,
start tracking end-effector position 2: Ready to Exit Mode
  }
}
```

9.1.2 Response: response_set_ub_manip_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_ub_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

9.1.3 Message Push: none

9.2 Motion Operation Control

Activated only via the request_set_ub_manip_mode interface.

9.2.1 Request: request_set_ub_manip_ee_pose

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Coordinate System Definition Reference:
    # Origin: base_link coordinate frame
    # Axes: X-axis aligned with the robot's forward direction,
    Y-axis pointing to the robot's left, Z-axis pointing upward
    #
    # Parameter Definition:
    # Head orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "head_quat": [0.0, 0.0, 0.0, 1.0],

    # Left-hand position relative to the reference frame, in
    meters
    "left_hand_pos": [0.0, 0.0, 0.0],

    # Left-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

    # Right-hand position relative to the reference frame, in
    meters
    "right_hand_pos": [0.0, 0.0, 0.0],

    # Right-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}
```

9.2.2 Response: response_set_ub_manip_ee_pose

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```
"data": {
  "result": "success" # success, fail_motor, fail_invalid_cmd:
Invalid Command
}
}
```

9.2.3 Message Push: none

9.3 Get Motion Operation Pose Information

9.3.1 Request: request_get_ub_manip_ee_pose

JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

9.3.2 Response: response_get_ub_manip_ee_pose

JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_ub_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Coordinate System Definition Reference:
    # Origin: base_link coordinate frame
    # Axes: X-axis aligned with the robot's forward direction, Y-
axis pointing to the robot's left, Z-axis pointing upward
    #
    # Parameter Definition:
    # Head position relative to the reference frame, in meters
    "head_pos": [0.0, 0.0, 0.0],
```

```

    # Head orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "head_quat": [0.0, 0.0, 0.0, 1.0],

    # Left hand position relative to the reference frame, in
    meters.
    "left_hand_pos": [0.0, 0.0, 0.0],

    # Left-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "left_hand_quat": [0.0, 0.0, 0.0,1.0],

    # Right-hand position relative to the reference frame, in
    meters.
    "right_hand_pos": [0.0, 0.0, 0.0],

    # Right-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "right_hand_quat": [0.0, 0.0, 0.0,1.0],
    "result": "success" # success, fail_motor, fail_invalid_cmd
  }
}

```

10. In-Place Operation

10.1 Enter In-Place Operation Mode

In this mode, you can control the robot's body movements while walking control is disabled.

10.1.1 Request: request_set_wb_manip_mode

JSON
代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "request_set_wb_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "mode": 0 # 0: Preparing to enter Mode 1: Operation Mode,
    start tracking end-effector position 2: Ready to Exit Mode
  }
}

```

```
}
```

10.1.2 Response: response_set_wb_manip_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_wb_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

10.1.3 Message Push: none

10.2 In-Place Operation Control

The function takes effect only when In-Place Operation Mode is activated via the protocol interface `request_set_wb_manip_mode` with mode set to 1.

10.2.1 Request: request_set_wb_manip_ee_pose

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_wb_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Reference Frame Definition:
    # Origin: The projection on the ground (z=0) of the midpoint
    between the left and right feet.
    # Axes: X-axis aligned with the robot's forward direction,
    Y-axis pointing to the robot's left, Z-axis pointing upward
    #
    # Parameter Definition:
    # Left hand position relative to the reference frame, in
    meters.
  }
}
```

```

    "left_hand_pos": [0.0, 0.3, 0.8],

    # Left-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

    # Right-hand position relative to the reference frame, in
    meters.
    "right_hand_pos": [0.0, -0.3, 0.8],

    # Right-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}

```

10.2.2 Response: response_set_wb_manip_ee_pose

JSON

代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "response_set_wb_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor, fail_invalid_cmd
  }
}

```

10.2.3 Message Push: none

10.3 Get In-Place Operation Pose Information

10.3.1 Request: request_get_wb_manip_ee_pose

JSON

代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "request_get_wb_manip_ee_pose",
  "timestamp": 1672373633989,

```

```
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
}
}
```

10.3.2 Response: response_get_wb_manip_ee_pose

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_wb_manip_ee_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Reference Frame Definition:
    # Origin: The projection on the ground (z=0) of the midpoint
    between the left and right feet.
    # Axes: X-axis aligned with the robot's forward direction, Y-
    axis pointing to the robot's left, Z-axis pointing upward
    #
    # Parameter Definition:
    # Left-hand position relative to the reference frame, in
    meters.
    "left_hand_pos": [0.0, 0.0, 0.0],

    # Left-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "left_hand_quat": [0.0, 0.0, 0.0, 1.0],

    # Right-hand position relative to the reference frame, in
    meters.
    "right_hand_pos": [0.0, 0.0, 0.0],

    # Right-hand orientation relative to the reference frame,
    represented as a quaternion[x,y,z,w]
    "right_hand_quat": [0.0, 0.0, 0.0, 1.0],
    "result": "success" # success, fail_motor, fail_invalid_cmd
  }
}
```

10.3.3 Message Push: none

11. Dual-Arm Coordinated Move Control

11.1 Switch to Move Control Mode

11.1.1 Request: request_set_move_mode

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_set_move_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0: Exit Move Control
    # 1: Mobile Move Mode
    # 2: In-place Move Mode
    "mode": 0
  }
}
```

11.1.2 Response: response_set_move_mode

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "response_set_move_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

11.1.3 Message Push: none

11.2 MoveJ Control Command

11.2.1 Request: request_moveJ

```
Python
{
```

```

"accid": "HU_D04_01_001",
"title": "request_moveJ",
"timestamp": 1672373633989,
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
    # Joint position ranges are defined by the corresponding
robot model's URDF file
    # Model File Download Address:
https://github.com/limxdynamics/humanoid-description

    # If the following data are provided simultaneously, both
arms will be controlled (target positions in radians)
    # Left Arm Joint Order:
    # - "left_shoulder_pitch_joint"
    # - "left_shoulder_roll_joint"
    # - "left_shoulder_yaw_joint"
    # - "left_elbow_joint"
    # - "left_wrist_yaw_joint"
    # - "left_wrist_pitch_joint"
    # - "left_wrist_roll_joint"
    "left": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

    # Right Arm Joint Order:
    # - "right_shoulder_pitch_joint"
    # - "right_shoulder_roll_joint"
    # - "right_shoulder_yaw_joint"
    # - "right_elbow_joint"
    # - "right_wrist_yaw_joint"
    # - "right_wrist_pitch_joint"
    # - "right_wrist_roll_joint"
    "right": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

    # In In-place MoveJ Mode only, providing the following data
will control torso posture
    "torso_height": 0, # Body height ratio: range[-1, 1]
    "torso_pitch": 0, # Pitch motion ratio: range[-1, 1]
    "torso_roll": 0, # Roll motion ratio: range[-1, 1]
    "torso_yaw": 0, # Yaw motion ratio: range[-1, 1]

    # If the following data are provided, head motion will be
controlled
    "head_pitch": 0.0, # pitch joint target position (radians)
    "head_yaw": 0.0, # yaw joint target position (radians)

```

```
    "speed": 0.2 # Motion speed: range [0, 0.5] rad/s,
controlling the motion speed of both arms
  }
}
```

11.2.2 Response: response_moveJ

```
Python
{
  "accid": "HU_D04_01_001",
  "title": "response_moveJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

11.2.3 Message Push: notify_moveJ

Execution completion or failure will trigger an active message push.

```
Python
代码块
{
  "accid": "HU_D04_01_001",
  "title": "notify_moveJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor,
fail_invalid_speed: invalid speed value
  }
}
```

11.3 MoveP Control Command

11.3.1 Request: request_moveP

```
Python
{
```

```

"accid": "HU_D04_01_001",
"title": "request_moveP",
"timestamp": 1672373633989,
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
    # If the following data are provided simultaneously, both
    arms will be controlled
    "left_position": [0.0, 0.0, 0.0], # Target position of the
    left arm end-effector in meters, specified in the order x, y, z
    "left_quat": [0.0, 0.0, 0.0, 1.0], # Target orientation of
    the left arm end-effector, represented as a quaternion (x, y, z,
    w)
    "right_position": [0.0, 0.0, 0.0], # Target position of the
    right arm end-effector in meters, specified in the order x, y, z
    "right_quat": [0.0, 0.0, 0.0, 1.0], # Target orientation of
    the right arm end-effector, represented as a quaternion (x, y, z,
    w)

    # In In-place MoveP Mode only, providing the following data
    will control torso posture
    "torso_height": 0, # Body height ratio: range[-1, 1]
    "torso_pitch": 0, # Pitch motion ratio: range[-1, 1]
    "torso_roll": 0, # Roll motion ratio: range[-1, 1]
    "torso_yaw": 0, # Yaw motion ratio: range[-1, 1]

    # If the following data are provided simultaneously, the
    head will be controlled
    "head_pitch": 0.0, # pitch joint target position (radians)
    "head_yaw": 0.0, # yaw joint target position (radians)

    "speed": 0.2 # Motion speed: range [0, 0.5] rad/s,
    controlling the motion speed of both arms
}
}

```

11.3.2 Response: response_moveP

```

Python
{
  "accid": "HU_D04_01_001",
  "title": "response_moveP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {

```

```
    "result": "success" # success, fail_motor
  }
}
```

11.3.3 Message Push: notify_moveP

Execution completion or failure will trigger an active message push.

Python

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_moveP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor,
    fail_invalid_speed: invalid speed value
  }
}
```

11.4 Get Dual-Arm End-Effector Pose

11.4.1 Request: request_get_move_pose

This interface is used to obtain the end-effector pose information of both robot arms.

JSON

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_move_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

11.4.2 Response: response_get_move_pose

Upon receiving the request, the robot returns the current pose data of both arms.

```

json
{
  "accid": "HU_D04_01_001",
  "title": "response_get_move_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "timestamp": 1672373633989, # Represents the data timestamp,
in milliseconds
    "left_position": [0.0, 0.0, 0.0], # The position of the left
arm end-effector, in meters, ordered as x, y, z
    "left_quat": [0.0, 0.0, 0.0, 1.0], # The orientation of the
left arm end-effector, as a quaternion x, y, z, w
    "right_position": [0.0, 0.0, 0.0], # The position of the
right arm end-effector, in meters, ordered as x, y, z
    "right_quat": [0.0, 0.0, 0.0, 1.0], # The orientation of the
right arm end-effector, as a quaternion x, y, z, w
    "result": "success" # fail_not_data
  }
}

```

12. Dual-Arm Coordinated Servo Control

12.1 Switch to Servo Control Mode

12.1.1 Request: request_set_servo_mode

```

JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_set_servo_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0: Exit Servo Control
    # 1: Mobile Servo Mode
    # 2: In-place Servo Mode
    "mode": 0
  }
}

```

12.1.2 Response: response_set_servo_mode

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_servo_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

12.1.3 Message Push: none

12.2 ServoJ Control Command

12.2.1 Request: request_servoJ

Recommend controlling the robotic arm at the control frequency of ≥ 500 Hz in real-time systems to ensure control performance and stability.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_servoJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Joint position ranges are defined by the corresponding
    robot model's URDF file
    # Model File Download Address:
    https://github.com/limxdynamics/humanoid-description

    # If the following data are provided simultaneously, both
    arms will be controlled (target positions in radians)
    # Left Arm Joint Order:
    # - "left_shoulder_pitch_joint"
    # - "left_shoulder_roll_joint"
    # - "left_shoulder_yaw_joint"
```

```

# - "left_elbow_joint"
# - "left_wrist_yaw_joint"
# - "left_wrist_pitch_joint"
# - "left_wrist_roll_joint"
"left": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

# Right Arm Joint Order:
# - "right_shoulder_pitch_joint"
# - "right_shoulder_roll_joint"
# - "right_shoulder_yaw_joint"
# - "right_elbow_joint"
# - "right_wrist_yaw_joint"
# - "right_wrist_pitch_joint"
# - "right_wrist_roll_joint"
"right": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

# In In-place ServoJ Mode only, providing the following data
will control torso posture
"torso_height": 0, # Body height ratio: range[-1, 1]
"torso_pitch": 0, # Pitch motion ratio: range[-1, 1]
"torso_roll": 0, # Roll motion ratio: range[-1, 1]
"torso_yaw": 0, # Yaw motion ratio: range[-1, 1]

# If the following data are provided, head motion will be
controlled
"head_yaw": 0.0, # yaw joint target position (radians)
"head_pitch": 0.0 # pitch joint target position (radians)
}
}

```

12.2.2 Response: none

12.2.3 Message Push: notify_servoJ

The server will send this message to indicate the failure reason when a ServoJ control operation fails.

```

JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "notify_servoJ",
  "timestamp": 1672373633989,

```

```

"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
  "result": "fail_invalid_cmd" # fail_invalid_cmd: fail_motor
}
}

```

12.3 ServoP Control Command

12.3.1 Request: request_servoP

Recommend controlling the robotic arm at the control frequency of **≥500 Hz** in real-time systems to ensure control performance and stability.

```

JSON
{
  "accid": "HU_D04_01_001",
  "title": "request_servoP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # If the following data are provided simultaneously, both
    arms will be controlled
    "left_position": [0.0, 0.0, 0.0], # Target position of the
    left arm end-effector in meters, specified in the order x, y, z
    "left_quat": [0.0, 0.0, 0.0, 1.0], # Target orientation of
    the left arm end-effector, represented as a quaternion (x, y, z,
    w)
    "right_position": [0.0, 0.0, 0.0], # Target position of the
    right arm end-effector in meters, specified in the order x, y, z
    "right_quat": [0.0, 0.0, 0.0, 1.0] # Target orientation of
    the right arm end-effector, represented as a quaternion (x, y, z,
    w)

    # In In-place ServoP Mode only, providing the following data
    will control torso posture
    "torso_height": 0, # Body height ratio: range[-1, 1]
    "torso_pitch": 0, # Pitch motion ratio: range[-1, 1]
    "torso_roll": 0, # Roll motion ratio: range[-1, 1]
    "torso_yaw": 0, # Yaw motion ratio: range[-1, 1]

    # If the following data are provided simultaneously, the
    head will be controlled

```

```

        "head_yaw": 0.0,    # yaw joint target position (radians)
        "head_pitch": 0.0  # pitch joint target position (radians)
    }
}

```

12.3.2 Response: none

12.3.3 Message Push: notify_servoP

The server will send this message to indicate the failure reason when a ServoP control operation fails.

JSON

代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "notify_servoP",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_invalid_cmd" # fail_invalid_cmd, fail_motor
  }
}

```

12.4 Get Dual-Arm End Pose

12.4.1 Request: request_get_servo_pose

Retrieve the end-effector pose information of both robot arms via this interface.

JSON

```

{
  "accid": "HU_D04_01_001",
  "title": "request_get_servo_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}

```

12.4.2 Response: response_get_servo_pose

Upon receiving the request, the system returns the current pose data for both arms.

```
json
{
  "accid": "HU_D04_01_001",
  "title": "response_get_servo_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "timestamp": 1672373633989, # Represents the data timestamp,
    in milliseconds
    "left_position": [0.0, 0.0, 0.0], # The position of the left
    arm end-effector, in meters, ordered as x, y, z
    "left_quat": [0.0, 0.0, 0.0, 1.0], # The orientation of the
    left arm end-effector, as a quaternion x, y, z, w
    "right_position": [0.0, 0.0, 0.0], # The position of the
    right arm end-effector, in meters, ordered as x, y, z
    "right_quat": [0.0, 0.0, 0.0, 1.0], # The orientation of the
    right arm end-effector, as a quaternion x, y, z, w
    "result": "success" # fail_not_data
  }
}
```

12.4.3 Message Push: none

13. Robot Joints Status

13.1 Request: request_get_joint_state

This request is used to retrieve the status of all robot joints.

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_get_joint_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

```
}
```

13.2 Response: response_get_joint_state

Upon receiving the request, the system returns the current state of all joints.

```
json
代码块
{
  "accid": "HU_D04_01_001",
  "title": "response_get_joint_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "names": [], # Names of each joint
    "q": [],     # Positions of each joint
    "dq": [],    # Velocities of each joint
    "tau": [],   # Torques of each joint
    "result": "success" # fail_not_data
  }
}
```

13.3 Message Push: none

14. Get IMU Data

14.1 Request: request_get_imu_data

```
JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "request_get_imu_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

14.2 Response: response_get_imu_data

```
JSON
```

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_imu_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success", # fail_no_data
    "euler": [0.0, 0.0, 0.0], // Euler Angles [roll,
pitch, yaw] in degrees
    "acc": [0.0, 0.0, 0.0], // Acceleration [x, y, z]
in m/s²
    "gyro": [0.0, 0.0, 0.0], // Gyroscope Angular
Velocity [x, y, z] in rad/s
    "quat": [0.0, 0.0, 0.0, 0.0] // Quaternion [w, x, y, z]
  }
}
```

14.3 Message Push: none

3.4.5 Dexterous Hand and Gripper Protocol Interface

1. LimX 2-Finger Gripper

1.1 Gripper Control Command

1.1.1 Request: request_set_limx_2fclaw_cmd

This protocol controls the gripper's grasping actions.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # If the following data are provided simultaneously, the
left gripper will be controlled
    "left_opening": 50, # Gripper Opening, 0-100, dimensionless
  }
}
```

```

(0 represents fully closed, 100 represents fully open)
    "left_speed": 50,    # Gripper Speed, 0~100 dimensionless
    (higher values indicate faster motion)
    "left_force": 50,    # Gripper Force, 0~100 dimensionless
    (higher values represent stronger force)

    # If the following data are provided simultaneously, the
    right gripper will be controlled
    "right_opening": 50,  # Gripper Opening, 0-100,
    dimensionless (0 represents fully close, 100 represents fully open
    大)
    "right_speed": 50,    # Gripper Spee, 0~100 dimensionless
    (higher values indicate faster motion)
    "right_force": 50,    #Gripper Forc, 0~100 dimensionless
    (higher values represent stronger force)
}
}

```

1.1.2 Response: response_set_limx_2fclaw_cmd

```

JSON
代码块
{
  "accid": "HU_D04_01_001",
  "title": "response_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}

```

1.1.3 Message Push: none

1.2 Get Gripper Status Information

1.2.1 Request: request_get_limx_2fclaw_state

```

JSON
代码块

```

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_limx_2fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

1.2.2 Response: response_get_limx_2fclaw_state

Return Gripper Status Information.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_limx_2fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "timestamp": 1672373633989, # Represents the data timestamp,
    in milliseconds

    "left_opening": 50, # Gripper Opening, 0-100, dimensionless
    (0 represents fully close, 100 represents fully open)
    "left_speed": 50, # Gripper Speed, 0~100 dimensionless
    (higher values indicate faster motion)
    "left_force": 50, # Gripper Force, 0~100 dimensionless
    (higher values represent stronger force)

    "right_opening": 50, # Gripper Opening, 0-100,
    dimensionless (0 represents fully close, 100 represents fully
    open)
    "right_speed": 50, # Gripper Speed, 0~100 dimensionless
    (higher vahigher values represent stronger force)

    "result": "success" # success, fail_motor
  }
}
```

1.2.3 Message Push: none

2. Limx 3-Finger Gripper

2.1 Gripper Control Command

2.1.1 Request: request_set_limx_3fclaw_cmd

This protocol controls the gripper's grasping actions.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_limx_3fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # If the following data are provided simultaneously, the
    left gripper will be controlled
    "left_left": 0,    # Left Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "left_middle": 0,  # Middle Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "left_right": 0,   # Right Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "left_lr_rot": 0,  # Finger Rotation Angle, 0~180 degrees,
    (0 = fingers adjacent, 180 = fingers opposed, default: 180)

    # If the following data are provided simultaneously, the
    right gripper will be controlled
    "right_left": 0,   # Left Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "right_middle": 0, # Middle Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "right_right": 0,  # Right Finger Bend Angle, 0-100,
    dimensionless (0 = fully closed, 100 = fully open, default: 42)
    "right_lr_rot": 1  # Finger Rotation Angl, 0~180 degrees, (0
    = fingers adjacent, 180 = fingers opposed, default: 180)
  }
}
```

2.1.2 Response: response_set_limx_3fclaw_cmd

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_limx_3fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}
```

2.1.3 Message Push: none

2.2 Get Gripper Status Information

2.2.1 Request: request_get_limx_3fclaw_state

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_limx_3fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

2.2.2 Response: esponse_get_limx_3fclaw_state

Return Gripper Status Information.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_limx_3fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
```

```

"data": {
  "timestamp": 1672373633989, # Represents the data timestamp,
in milliseconds
  "left_left": 0,    # Left Gripper Left Finger Bend Angle, 0-
100, dimensionless (0 = fully closed, 100 = fully open, default:
42)
  "left_middle": 0,  # Left Gripper Middle Finger Bend Angle,
0-100, dimensionless (0 = fully closed, 100 = fully open, default:
42)
  "left_right": 0,   # Left Gripper Right Finger Bend Angle,
0-100, dimensionless (0 = fully closed, 100 = fully open, default:
42)
  "left_lr_rot": 0,  # Left Gripper Finger Rotation Angle,
0~180 degrees, (0 = fingers adjacent 贴, 180 = fingers opposed,
default: 180)
  "right_left": 0,   # Right Gripper Left Finger Bend Angle,
0-100, dimensionless (0 = fully closed, 100 = fully open, default:
42)
  "right_middle": 0, # Right Gripper Middle Finger Bend
Angle, 0-100, dimensionless (0 = fully closed, 100= fully open,
default: 42)
  "right_right": 0,  # Right Gripper Right Finger Bend Angle,
0-100, dimensionless (0 = fully closed, 100= fully open, default:
42)
  "right_lr_rot": 1  # Right Gripper Finger Rotation Angle,
0~180 degrees, (0 = fingers adjacent, 180 = fingers opposed,
default: 180)

  "result": "success" # success, fail_motor
}
}

```

2.2.3 Message Push: none

3. Inspire 2-Finger Gripper

3.1 Gripper Control Command

3.1.1 Request: request_set_claw_cmd

This protocol controls the gripper's grasping actions.

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_set_claw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # If the following data are provided simultaneously, the
    left gripper will be controlled
    "left_opening": 100, # Left Gripper Opening, range[0, 1000],
    dimensionless
    "left_speed": 500,   # Left Gripper Speed, range[0, 1000],
    dimensionless
    "left_force": 500,   # Left Gripper Force, range[0, 1000],
    dimensionless
    "left_mode": 1,      # Left Gripper Control Mode (1: Grip;
    2: Release; 3: Position Control)

    # 如 If the following data are provided simultaneously, the
    right gripper will be controlled
    "right_opening": 100, # Right Gripper Opening, range[0,
    1000], dimensionless
    "right_speed": 500,   # Right Gripper Speed, range[0, 1000],
    dimensionless
    "right_force": 500,   # Right Gripper Force, range[0, 1000],
    dimensionless
    "right_mode": 1       # Right Gripper Control Mode (1: Grip;
    2: Release; 3: Position Control)
  }
}
```

3.1.2 Response: response_set_claw_cmd

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_set_claw_cmd",
  "timestamp": 1672373633989,
```

```
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
  "result": "success" # success, fail_motor
}
}
```

3.1.3 Message Push: none

3.2 Get Gripper Status Information

3.2.1 Request: request_get_claw_state

JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_claw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

3.2.2 Response: response_get_claw_state

Return Gripper Status Information.

JSON
代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_claw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "timestamp": 1672373633989, # Represents the data timestamp,
    in milliseconds
    "left_opening": 100, # Left Gripper Opening, range[0, 1000],
    dimensionless
    "left_speed": 500, # Left Gripper Speed, range[0, 1000],
    dimensionless
  }
}
```

```

    "left_force": 500,    # Left Gripper Force, range[0, 1000],
dimensionless

    "right_opening": 100, # Right Gripper Opening, range[0,
1000], dimensionless
    "right_speed": 500,   # Right Gripper Speed, range[0, 1000],
dimensionless
    "right_force": 500,   # Right Gripper Force, range[0, 1000],
dimensionless

    "result": "success"  # success, fail_motor
}
}

```

3.2.3 Message Push: none

4. BrainCo's Revo 1 Dexterous Hand

4.1 Dexterous Hand Control Command

4.1.1 Request: request_set_brainco_hand_cmd

JSON
代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "request_set_brainco_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # If the following data are provided, the left hand will
    be controlled
    "left_thumb": 50,          # Left Thumb Flexion Angle, 0-
100, dimensionless
    "left_thumb_aux": 50,     # Left Thumb Adduction Angle, 0-
100, dimensionless
    "left_index": 50,         # Left Index Finger Flexion
Angle, 0-100, dimensionless
    "left_middle": 50,        # Left Middle Finger Flexion
Angle, 0-100, dimensionless
    "left_ring": 50,          # Left Ring Finger Flexion
Angle, 0-100, dimensionless

```

```

        "left_pinky": 50,          # Left Little Finger Flexion
Angle, 0-100, dimensionless
        "left_mode": 3,           # Force Level 1: low 2: medium
3: high, default: 2

        # If the following data are provided, the right hand
will be controlled
        "right_thumb": 50,        # Right Thumb Flexion Angle, 0-
100, dimensionless
        "right_thumb_aux": 50,    # Right Thumb Adduction Angle,
0-100, dimensionless
        "right_index": 50,        # Right Index Finger Flexion
Angle, 0-100, dimensionless
        "right_middle": 50,       # Right Middle Finger Flexion
Angle, 0-100, dimensionless
        "right_ring": 50,         # Right Ring Finger Flexion
Angle, 0-100, dimensionless
        "right_pinky": 50,        # Right Little Finger Flexion
Angle, 0-100, dimensionless
        "right_mode": 3           # Force Level 1: low 2: medium
3: high, default: 2
    }
}

```

4.1.2 Response: response_set_brainco_hand_cmd

JSON
代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "response_set_brainco_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success" # success, fail_motor
  }
}

```

4.1.3 Message Push: none

4.2 Get Dexterous Hand Status

4.2.1 Request: request_get_brainco_hand_state

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "request_get_brainco_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
}
```

4.2.2 Response: response_get_brainco_hand_state

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "response_get_brainco_hand_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "timestamp": 1672373633989, # Represents the data
    timestamp, in milliseconds
    "left_thumb": 50,          # Left Thumb Flexion Angle, 0-
    100, dimensionless
    "left_thumb_aux": 50,      # Left Thumb Adduction Angle, 0-
    100, dimensionless
    "left_index": 50,          # Left Index Finger Flexion
    Angle, 0-100, dimensionless
    "left_middle": 50,         # Left Middle Finger Flexion
    Angle, 0-100, dimensionless
    "left_ring": 50,           # Left Ring Finger Flexion
    Angle, 0-100, dimensionless
    "left_pinky": 50,          # Left Little Finger Flexion
    Angle, 0-100, dimensionless

    "right_thumb": 50,         # Right Thumb Flexion Angle, 0-
    100, dimensionless
    "right_thumb_aux": 50,     # Right Thumb Adduction Angle,
    0-100, dimensionless
    "right_index": 50,         # Right Index Finger Flexion
```

```

Angle, 0-100, dimensionless
    "right_middle": 50,      # Right Middle Finger Flexion
Angle, 0-100, dimensionless
    "right_ring": 50,       # Right Ring Finger Flexion
Angle, 0-100, dimensionless
    "right_pinky": 50       # Right Little Finger Flexion
Angle, 0-100, dimensionless
    "result": "success"    # success, fail_motor
  }
}

```

4.2.3 Message Push: none

5. BrainCo's Revo 2 Dexterous Hand

5.1 Dexterous Hand Control Command

5.1.1 Request: request_set_brainco2_hand_cmd

JSON
代码块

```

{
  "accid": "HU_D04_01_001",
  "title": "request_set_brainco2_hand_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # Left Hand:
    # Finger Index Mapping (0-5): 0: Thumb Tip, 1: Thumb
    Base, 2: Index Finger, 3: Middle Finger, 4: Ring Finger, 5:
    Little Finger
    # left_mode: Control mode
    #           0: Exit control mode
    #           1: Position-Time Mode, requires specifying
    left_pos and left_time
    #           2: Position-Velocity Mode, requires
    specifying left_pos and left_vel
    #           3: Force Control Mode, requires specifying
    left_current
    # left_pos: Target position of each finger (unit: rad)
    #           Respective Ranges: 0-1.0297、0-1.5707、0-
    1.4137、0-1.4137、0-1.4137、0-1.4137
  }
}

```

```

    # left_vel: Target velocity of each finger (unit: rad/s)
    #           Respective Ranges: 0-2.5367、0-2.6180、0-
2.2689、0-2.2689、0-2.2689、0-2.2689
    # left_current: Target current of each finger (unit:
mA), range: ±1000mA
    # left_time: Control time for each finger (unit: ms),
range: 1-2000ms

    "left_mode": 1,
    "left_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    "left_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    "left_current": [500, 500, 500, 500, 500, 500],
    "left_time": [1000, 1000, 1000, 1000, 1000, 1000],

    # Right Hand:
    # Finger Index Mapping (0-5): 0: Thumb Tip, 1: Thumb
Base, 2: Index Finger, 3: Middle Finger, 4: Ring Finger, 5:
Little Finger
    # right_mode: Control mod
    #           0: Exit control mode
    #           1: Position-Time Mod, requires specifying
right_pos and right_time
    #           2: Position-Velocity Mode, requires
specifying right_pos and right_vel
    #           3: Force Control Mode, requires specifying
right_current
    # right_pos: Target position of each finger (unit: rad)
    #           Respective Ranges: 0-1.0297、0-1.5707、0-
1.4137、0-1.4137、0-1.4137、0-1.4137
    # right_vel: Target velocity of each finger (unit:
rad/s)
    #           Respective Ranges: 0-2.5367、0-2.6180、0-
2.2689、0-2.2689、0-2.2689、0-2.2689
    # right_current: Target current of each finger (unit:
mA), range: ±1000mA
    # right_time: Control time for each finger (unit: ms),
range: 1-2000ms

    "right_mode": 1,
    "right_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    "right_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    "right_current": [500, 500, 500, 500, 500, 500],
    "right_time": [1000, 1000, 1000, 1000, 1000, 1000]

```

```
}  
}
```

5.1.2 Response: response_set_brainco2_hand_cmd

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "response_set_brainco2_hand_cmd",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
    "result": "success" # success, fail_motor: motor error,  
    fail_invalid_cmd: invalid command  
  }  
}
```

5.1.3 Message Push: none

5.2 Get Dexterous Hand Status

5.2.1 Request: request_get_brainco2_hand_state

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "request_get_brainco2_hand_state",  
  "timestamp": 1672373633989,  
  "guid": "746d937cd8094f6a98c9577aaf213d98",  
  "data": {  
  }  
}
```

5.2.2 Response: response_get_brainco2_hand_state

JSON

代码块

```
{  
  "accid": "HU_D04_01_001",  
  "title": "response_get_brainco2_hand_state",
```

```

"timestamp": 1672373633989,
"guid": "746d937cd8094f6a98c9577aaf213d98",
"data": {
  "timestamp": 1672373633989,
  "left_mode": 1,
  "left_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
  "left_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
  "left_current": [500, 500, 500, 500, 500, 500],
  "left_time": [1000, 1000, 1000, 1000, 1000, 1000],
  "right_mode": 1,
  "right_pos": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
  "right_vel": [0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
  "right_current": [500, 500, 500, 500, 500, 500],
  "right_time": [1000, 1000, 1000, 1000, 1000, 1000]
}
}

```

5.2.3 Message Push: none

3.4.6 Global Message Protocol Interface

1. Robot Status Information

This protocol periodically reports the robot's status information.

Status Infomation	Description
accid	The robot's unique serial number.
title	notify_robot_info
timestamp	The timestamp when the command is sent, in milliseconds.
guid	The unique identifier of the message.
data	Contains the message data.

Example:

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": []
  }
}
```

2. The Remote Controller Data

This protocol reports the robot's remote controller data.

Data Information	Description
accid	The robot's unique serial number.
title	notify_joy_data
timestamp	The timestamp when the command is sent, in milliseconds.
guid	The unique identifier of the message.
data	Contains the message data.

Example:

JSON

代码块

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_joy_data",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "axes": [],      # joystick data
    "buttons": []    # button data
  }
}
```

```
}
```

3.4.7 Protocol Interface Usage Example

1. Python Example

- **Environment Setup:** using Ubuntu 20.04 as an example, install the required dependencies

```
Python
sudo apt install python3-dev python3-pip
sudo pip install websocket-client==1.8.0
```

- **Execute the Script**

```
Python
python humanoid.py
```

- **humanoid.py Implementation**

- ACCID: Replace with the actual software serial number (SN).
- ROBOT_IP: Usually, use 127.0.0.1 for simulation and 10.192.1.2 for real hardware.

```
Python
代码块
import json
import uuid
import threading
import time
import websocket
from datetime import datetime

# Replace this ACCID value with your robot's actual serial
number (SN)
ACCID = None

# Replace it with the real IP address of the robot.
# Usually, for simulation, it is: 127.0.0.1
# for a real machine, it is: 10.192.1.2
ROBOT_IP = "10.192.1.2"
```

```

# Atomic flag for graceful exit
should_exit = False

# WebSocket client instance
ws_client = None

# Generate dynamic GUID
def generate_guid():
    return str(uuid.uuid4())

# Send WebSocket request with title and data
def send_request(title, data=None):
    global ACCID
    if data is None:
        data = {}

    # Create message structure with necessary fields
    message = {
        "accid": ACCID,
        "title": title,
        "timestamp": int(time.time() * 1000), # Current
timestamp in milliseconds
        "guid": generate_guid(),
        "data": data
    }

    message_str = json.dumps(message)

    # Send the message through WebSocket if client is
connected
    if ws_client:
        ws_client.send(message_str)

# Handle user commands
def handle_commands():
    global should_exit
    while not should_exit:
        command = input("Enter command ('prepare', 'servo',
'movej', 'move1', 'movep', 'head', 'waist', 'state',
'claw_cmd', 'claw_state', 'damping', 'zero') or 'exit' to
quit:\n")

        if command == "exit":
            should_exit = True # Set exit flag to stop the

```

```

loop
    break
elif command == "prepare":
    send_request("request_prepare") # request_prepare
elif command == "servo":
    # Servo control mode flag from user
    mode_input = input("Enable Servo (1/0):").strip()
    mode_value = 1 if mode_input == '1' else 0
    send_request("request_set_control_mode", {"mode":
mode_value})
    elif command == "movej":
        send_request("request_moveJ", { # request_moveJ
            "left": [-1.44532, 0.0987686, 0.179059, -
1.64716, -0.0537614, 0.200834, -0.236136],
            "right": [0.10103, -0.0987769, -0.179462, -
1.64705, 0.0527488, 0.198867, 0.235933],
            "speed": 0.2
        })
    elif command == "movel":
        send_request("request_moveL", { # request_moveL
            "left_position": [0.282554, 0.282495, 0.183549],
            "left_quat": [0.0568807, -0.821029, -
0.0970607, 0.559692],
            "right_position": [0.240977, -0.460845, 0.296067],
            "right_quat": [-0.27937, -
0.677035, 0.393395, 0.555713],
            "speed": 0.1
        })
    elif command == "movep":
        send_request("request_moveP", { # request_moveP
            "left_position": [0.089644, 0.428712, 0.0519788],
            "left_quat": [0.269296, -0.119683, -
0.489868, 0.820478],
            "right_position": [0.0835307, -0.531453, 0.13568],
            "right_quat": [-0.436152, -
0.285065, 0.265969, 0.81103],
            "speed": 0.1
        })
    elif command == "head":
        send_request("request_moveJ", { # request_moveJ
            "head_pitch": 0.5854,
            "head_yaw": 0.5854,
            "speed": 0.1
        })

```

```

        elif command == "waist":
            send_request("request_set_waist_and_height", { #
request_set_waist_and_height
                "height": 0.0,
                "pitch": 0.0,
                "roll": 0.0,
                "yaw": 0.0
            })
        elif command == "claw_cmd":
            send_request("request_set_claw_cmd", { #
request_set_claw_cmd
                "left_opening": 100,
                "left_speed": 500,
                "left_force": 500,
                "left_mode": 1,
                "right_opening": 100,
                "right_speed": 500,
                "right_force": 500,
                "right_mode": 1
            })
        elif command == "claw_state":
            send_request("request_get_claw_state")
        elif command == "state":
            send_request("request_get_pose") #
request_get_pose
        elif command == "damping":
            send_request("request_damping") # request_damping
        elif command == "zero":
            send_request("request_zero_torque") #
request_zero_torque

# WebSocket on_open callback
def on_open(ws):
    print("Connected!")
    # Start handling commands in a separate thread
    threading.Thread(target=handle_commands,
daemon=True).start()

# WebSocket on_message callback
def on_message(ws, message):
    global ACCID
    root = json.loads(message)
    title = root.get("title", "")
    ACCID = root.get("accid", None)

```

```

        if title != "notify_robot_info":
            print(f"Received message: {message}") # Print the
received message

# WebSocket on_close callback
def on_close(ws, close_status_code, close_msg):
    print("Connection closed.")

# Close WebSocket connection
def close_connection(ws):
    ws.close()

def main():
    global ws_client

    # Create WebSocket client instance
    ws_client = websocket.WebSocketApp(
        f"ws://{ROBOT_IP}:5000", # WebSocket server URI
        on_open=on_open,
        on_message=on_message,
        on_close=on_close
    )

    # Configure socket send and receive buffer sizes
    # Increase send buffer size to 2MB (default is typically
much smaller)
    # This helps prevent data loss when sending large messages
or high-frequency data
    ws_client.sock_opt = [("socket", "SO_SNDBUF", 2 * 1024 *
1024)]

    # Increase receive buffer size to 2MB
    # This allows handling larger incoming messages without
truncation
    ws_client.sock_opt.append(("socket", "SO_RCVBUF", 2 * 1024
* 1024))

    # Run WebSocket client loop
    print("Press Ctrl+C to exit.")
    ws_client.run_forever()

if __name__ == "__main__":
    main()

```

2. Linux C++ Example

- **Environment Setup:** using Ubuntu 20.04 as an example, install websocketpp, nlohmann/json and boost dependencies:

```
Bash
sudo apt-get install libboost-all-dev libwebsocketpp-dev
nlohmann-json3-dev
```

- **Compile the Code**

```
Bash
g++ -std=c++11 -o humanoid humanoid.cpp -lssl -lcrypto -
lboost_system -lpthread
```

- **Execute the Program**

```
Bash
./humanoid
```

- **humanoid.cpp Implementation**

```
C++
代码块
#include <iostream>
#include <atomic>
#include <string>
#include <thread>
#include <chrono>
#include <websocketpp/client.hpp>
#include <websocketpp/config/asio.hpp>
#include <nlohmann/json.hpp>
#include <boost/uuid/uuid.hpp>
#include <boost/uuid/uuid_generators.hpp>
#include <boost/uuid/uuid_io.hpp>

using json = nlohmann::json;
using websocketpp::client;
using websocketpp::connection_hdl;

// Replace this value with the actual serial number (SN) of the
```

```

robot.
static std::string ACCID = "";

// Replace it with the real IP address of the robot.
// Usually, for simulation, it is: 127.0.0.1
// for a real machine, it is: 10.192.1.2
const std::string ROBOT_IP = "10.192.1.2";

// WebSocket client instance
static client<websocketpp::config::asio> ws_client;

// Atomic flag for graceful exit
static std::atomic<bool> should_exit(false);

// Connection handle for sending messages
static connection_hdl current_hdl;

// Generate dynamic GUID
static std::string generate_guid()
{
    boost::uuids::random_generator gen;
    boost::uuids::uuid u = gen();
    return boost::uuids::to_string(u);
}

// Send WebSocket request with title and data
static void send_request(const std::string &title, const json
&data = json::object())
{
    json message;

    // Adding necessary fields to the message
    message["accid"] = ACCID;
    message["title"] = title;
    message["timestamp"] =
std::chrono::duration_cast<std::chrono::milliseconds>(
std::chrono::system_clock::now().time_since_epoch())
        .count();
    message["guid"] = generate_guid();
    message["data"] = data;

    std::string message_str = message.dump();

```

```

        // Send the message through WebSocket
        ws_client.send(current_hdl, message_str,
websocketpp::frame::opcode::text);
    }

// Handle user commands
void handle_commands()
{
    std::cout << "Enter command ('prepare', 'loco_manip', 'movej',
'movel', 'movep', 'head', 'state', 'claw_cmd', 'claw_state',
'damping', 'zero') or 'exit' to quit:\n";
    while (!should_exit)
    {
        std::string command;
        std::cin >> command;

        if (command == "exit")
        {
            should_exit = true;
            return;
        }
        else if (command == "prepare")
        {
            send_request("request_prepare");
        }
        else if (command == "loco_manip")
        {
            int mode_value;
            std::cout << "Mode move or servo (0/1): ";
            std::cin >> mode_value;
            bool valid_input = (mode_value == 0 || mode_value ==
1);
            if (!valid_input)
            {
                std::cerr << "Error: Invalid input. Please enter 0
or 1." << std::endl;
                continue;
            }

            nlohmann::json data = {{"mode", mode_value}};
            send_request("request_set_loco_manip_mode", data);
        }
        else if (command == "movej")
        {

```

```

        nlohmann::json data = {
            {"left", {-1.44532, 0.0987686, 0.179059, -1.64716,
-0.0537614, 0.200834, -0.236136}},
            {"right", {0.10103, -0.0987769, -0.179462, -
1.64705, 0.0527488, 0.198867, 0.235933}},
            {"speed", 0.2}};
        send_request("request_moveJ", data);
    }
    else if (command == "moveI")
    {
        nlohmann::json data = {
            {"left_position", {0.282554, 0.282495, 0.183549}},
            {"left_pose", {0.0568807, -0.821029, -0.0970607,
0.559692}},
            {"right_position", {0.240977, -0.460845,
0.296067}},
            {"right_pose", {-0.27937, -0.677035, 0.393395,
0.555713}},
            {"speed", 0.1}};
        send_request("request_moveL", data);
    }
    else if (command == "moveP")
    {
        nlohmann::json data = {
            {"left_position", {0.089644, 0.428712,
0.0519788}},
            {"left_pose", {0.269296, -0.119683, -0.489868,
0.820478}},
            {"right_position", {0.0835307, -0.531453,
0.13568}},
            {"right_pose", {-0.436152, -0.285065, 0.265969,
0.81103}},
            {"speed", 0.1}};
        send_request("request_moveP", data);
    }
    else if (command == "head")
    {
        nlohmann::json data = {
            {"head_yaw", 0.5854},
            {"head_pitch", 0.5854},
            {"speed", 0.1}};
        send_request("request_moveJ", data);
    }
    else if (command == "claw_cmd")

```

```

    {
        nlohmann::json data = {
            {"left_opening", 100},
            {"left_speed", 500},
            {"left_force", 500},
            {"left_mode", 1},
            {"right_opening", 100},
            {"right_speed", 500},
            {"right_force", 500},
            {"right_mode", 1}};
        send_request("request_set_claw_cmd", data);
    }
    else if (command == "claw_state")
    {
        send_request("request_get_claw_state");
    }
    else if (command == "state")
    {
        send_request("request_get_pose");
    }
    else if (command == "damping")
    {
        send_request("request_damping");
    }
    else if (command == "zero")
    {
        send_request("request_zero_torque");
    }

    sleep(1);

    std::cout << "\nEnter command ('prepare', 'loco_manip',
'movej', 'move1', 'movep', 'head', 'state', 'claw_cmd',
'claw_state', 'damping', 'zero') or 'exit' to quit:\n";
}
}

// WebSocket open callback
static void on_open(connection_hdl hdl)
{
    std::cout << "Connected!" << std::endl;

    // Save connection handle for sending messages later
    current_hdl = hdl;
}

```

```

        // Start handling commands in a separate thread
        std::thread(handle_commands).detach();
    }

    // WebSocket TCP initialization handler
    static void on_tcp_init(connection_hdl hdl)
    {
        auto con = ws_client.get_con_from_hdl(hdl);

        // Obtain the underlying TCP socket
        auto &socket = con->get_socket().lowest_layer();

        // Configure socket options
        try
        {
            boost::system::error_code ec;

            // Set send buffer size (e.g., 2MB)
            const size_t sendBufferSize = 2 * 1024 * 1024;

            socket.set_option(websocketpp::lib::asio::socket_base::send_buffer_
                _size(sendBufferSize), ec);

            if (ec)
            {
                printf("Failed to set send buffer size: %s",
                    ec.message().c_str());
            }

            // Set receive buffer size (e.g., 2MB)
            const size_t recvBufferSize = 2 * 1024 * 1024;

            socket.set_option(websocketpp::lib::asio::socket_base::receive_buf
                fer_size(recvBufferSize), ec);

            if (ec)
            {
                printf("Failed to set receive buffer size: %s",
                    ec.message().c_str());
            }

            // Disable Nagle's algorithm to reduce latency

```

```

socket.set_option(websocketpp::lib::asio::ip::tcp::no_delay(true),
ec);

    if (ec)
    {
        printf("Failed to disable Nagle's algorithm: %s",
ec.message().c_str());
    }
}
catch (const std::exception &e)
{
    printf("Socket configuration exception: %s", e.what());
}
}

// WebSocket message callback
static void on_message(connection_hdl hdl,
client<websocketpp::config::asio>::message_ptr msg)
{
    // Parse JSON data from message payload
    json data = json::parse(msg->get_payload());

    // Extract 'accid' field if present
    if (data.contains("accid") && data["accid"].is_string() &&
ACCID.empty())
    {
        ACCID = data["accid"].get<std::string>();
    }

    if (msg->get_payload().find("notify_robot_info") ==
std::string::npos)
    {
        std::cout << "Received message: " << msg->get_payload() <<
std::endl;
    }
}

// WebSocket close callback
static void on_close(connection_hdl hdl)
{
    std::cout << "Connection closed." << std::endl;
}

// Close WebSocket connection

```

```

static void close_connection(connection_hdl hdl)
{
    ws_client.close(hdl, websocketpp::close::status::normal,
        "Normal closure"); // Close connection normally
}

int main()
{
    ws_client.init_asio(); // Initialize ASIO for WebSocket client

    ws_client.set_access_channels(websocketpp::log::alevel::none);

    // Set WebSocket event handlers
    ws_client.set_open_handler(&on_open);           // Set open
handler
    ws_client.set_message_handler(&on_message);     // Set message
handler
    ws_client.set_close_handler(&on_close);         // Set close
handler
    ws_client.set_tcp_init_handler(&on_tcp_init);   // Set tcp init
handler

    std::string server_uri = "ws://" + ROBOT_IP + ":5000"; //
WebSocket server URI

    websocketpp::lib::error_code ec;
    client<websocketpp::config::asio>::connection_ptr con =
ws_client.get_connection(server_uri, ec); // Get connection
pointer

    if (ec)
    {
        std::cout << "Error: " << ec.message() << std::endl;
        return 1; // Exit if connection error occurs
    }

    connection_hdl hdl = con->get_handle(); // Get connection
handle
    ws_client.connect(con);                  // Connect to server
    std::cout << "Press Ctrl+C to exit." << std::endl;

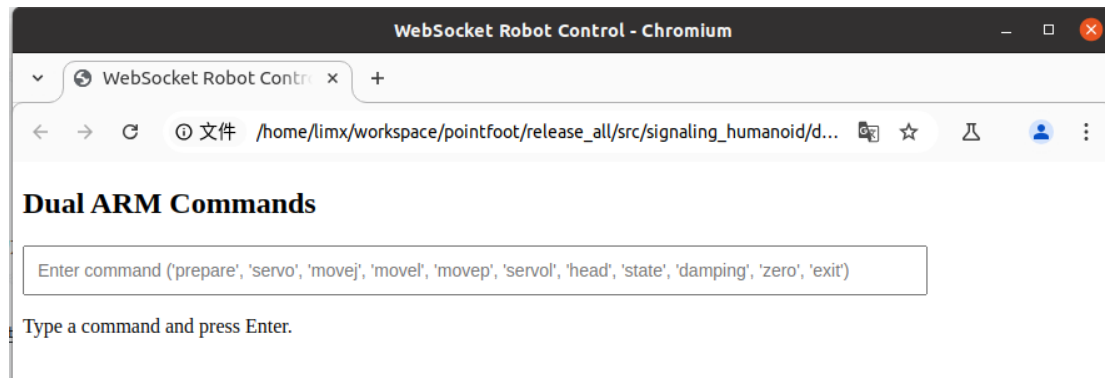
    // Run the WebSocket client loop
    ws_client.run();

```

```
    return 0;
}
```

3. JavaScript Example

- **Execute humanoid.html:** Save the `humanoid.html` file to your computer and open it in a browser to run the demo.



- **humanoid.html Implementation**

```
xml
代码块
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>WebSocket Robot Control</title>
  <style>
    #commandInput {
      width: 700px;
      padding: 10px;
      font-size: 14px;
    }
  </style>
</head>

<body>
  <h2>Humanoid Commands</h2>
  <input type="text" id="commandInput"
    placeholder="Enter command ('prepare', 'loco_manip',
```

```

'movej', 'movel', 'movep', 'head', 'claw_cmd', 'claw_state',
'state', 'damping', 'zero', 'exit')">
    <p>Type a command and press Enter.</p>

    <script>
        // Replace this ACCID value with your robot's actual
        serial number (SN)
        let ACCID = "";

        // WebSocket client instance
        let wsClient = null;

        // Generate dynamic GUID
        function generateGuid() {
            return 'xxxxxxxx-xxxx-4xxx-yxxx-
xxxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
                const r = Math.random() * 16 | 0,
                v = c === 'x' ? r : (r & 0x3 | 0x8);
                return v.toString(16);
            });
        }

        // Send WebSocket request with title and data
        function sendRequest(title, data = {}) {
            const message = {
                accid: ACCID,
                title: title,
                timestamp: Date.now(),
                guid: generateGuid(),
                data: data
            };

            if (wsClient && wsClient.readyState ===
WebSocket.OPEN) {
                wsClient.send(JSON.stringify(message));
            }
        }

        // Handle user commands
        function handleCommands() {
            const commandInput =
document.getElementById('commandInput');
            commandInput.addEventListener('keydown', function
(event) {

```

```

        if (event.key === 'Enter') {
            const command = commandInput.value.trim();
            commandInput.value = '';

            switch (command) {
                case 'prepare':
                    sendRequest('request_prepare');
                    break;
                case 'loco_manip':
                    const modeInput = prompt("Mode move or
servo (0/1):");

                    const modeValue = (modeInput ===
'1') ? 1 : 0;

                    sendRequest('request_set_loco_manip_mode', { mode: modeValue });
                    break;
                case 'movej':
                    sendRequest('request_moveJ', {
                        left: [-1.44532, 0.0987686,
0.179059, -1.64716, -0.0537614, 0.200834, -0.236136],
                        right: [0.10103, -0.0987769, -
0.179462, -1.64705, 0.0527488, 0.198867, 0.235933],
                        speed: 0.2
                    });
                    break;
                case 'movel':
                    sendRequest('request_moveL', {
                        left_position: [0.282554,
0.282495, 0.183549],
                        left_pose: [0.0568807, -0.821029,
-0.0970607, 0.559692],
                        right_position: [0.240977, -
0.460845, 0.296067],
                        right_pose: [-0.27937, -0.677035,
0.393395, 0.555713],
                        speed: 0.1
                    });
                    break;
                case 'movep':
                    sendRequest('request_moveP', {
                        left_position: [0.089644,
0.428712, 0.0519788],
                        left_pose: [0.269296, -0.119683, -
0.489868, 0.820478],

```

```

        right_position: [0.0835307, -
0.531453, 0.13568],
        right_pose: [-0.436152, -0.285065,
0.265969, 0.81103],
        speed: 0.1
    });
    break;
case 'head':
    sendRequest('request_moveJ', {
        head_yaw: 0.5854,
        head_pitch: 0.5854,
        speed: 0.1
    });
    break;
case 'claw_cmd':
    sendRequest('request_set_claw_cmd', {
        left_opening: 100,
        left_speed: 500,
        left_force: 500,
        left_mode: 1,
        right_opening: 100,
        right_speed: 500,
        right_force: 500,
        right_mode: 1
    });
    break;
case 'claw_state':
    sendRequest('request_get_claw_state');
    break;
case 'state':
    sendRequest('request_get_pose');
    break;
case 'damping':
    sendRequest('request_damping');
    break;
case 'zero':
    sendRequest('request_zero_torque');
    break;
case 'exit':
    wsClient.close();
    break;
default:
    alert("Invalid command. Try again.");
}

```

```

        }
    });
}

// WebSocket onOpen callback
function onOpen() {
    console.log("Connected!");
    handleCommands();
}

// WebSocket onMessage callback
function onMessage(event) {
    try {
        const message = JSON.parse(event.data);

        // Dynamically set ACCID from message if not
already set
        if (!ACCID && message.accid) {
            ACCID = message.accid;
            console.log(`ACCID set to: ${ACCID}`);
        }
    } catch (error) {
        console.log("Failed to parse message:", error);
    }

    if (event.data.includes('notify_robot_info')) return;
    console.log("Received message:", event.data);
}

// WebSocket onClose callback
function onClose(event) {
    console.log("Connection closed.");
}

// Initialize WebSocket client
function initWebSocket() {
    // Replace it with the real IP address of the robot.
    // Usually, for simulation, it is: 127.0.0.1
    // for a real machine, it is: 10.192.1.2
    wsClient = new WebSocket('ws://10.192.1.2:5000');
    wsClient.onopen = onOpen;
    wsClient.onmessage = onMessage;
    wsClient.onclose = onClose;
    console.log("Press Ctrl+C to exit.");
}

```

```

    }

    // Start WebSocket connection when the page loads
    window.onload = initWebSocket;
</script>
</body>

</html>

```

3.5 Low-Level Motion Control Development Interface

The **cross-platform low-level motion control API** provides a unified C++/Python interface, compatible with ROS1, ROS2, and non-ROS systems, enabling rapid migration and deployment of motion control algorithms. Through a hardware abstraction layer and standardized communication protocols, developers can seamlessly switch between simulation and real hardware environments, significantly reducing multi-platform adaptation costs.

Notes:

1. To use the low-level control API, press R1 + START to switch to **Developer Mode**. In this mode, high-level control interfaces are disabled, and the robot only responds to power-on/power-off and zeroing remote controller commands.
2. Example code for the low-level interface can be found in the RL deployment and training section.
3. When switched to Developer Mode, the robot will retain the power state. To exit Developer Mode, press L2 + ○.

3.5.1 C++ Motion Control Development Interface

1. getInstance Interface

Function Name	getInstance
Function Prototype	static Humanoid* getInstance();
Description	Retrieves the singleton instance pointer of the Humanoid robot class.

Parameters	None
Return Value	Humanoid*, Pointer to the Humanoid instance
Remarks	Implements the singleton pattern to ensure only one instance of the Humanoid class exists in the program.
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ //obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // Enter an infinite loop to keep the program running while (true) { // sleep for 1000 milliseconds std::this_thread::sleep_for(std::chrono::milliseconds (1000)); } return 0; } </pre>

2. init Interface

Function	init
----------	-------------

Name	
Function Prototype	<code>bool init(const std::string& robot_ip_address = "127.0.0.1");</code>
Description	Initializes the communication and runtime environment for the motion control algorithm, typically called before other interfaces in the main function.
Parameters	<code>robot_ip_address</code> : The IP address of the robot. Use <code>"127.0.0.1"</code> for simulation and <code>"10.192.1.2"</code> for real robots.
Return Value	<code>true</code> if initialization succeeds; otherwise <code>false</code> .
Remarks	None
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "127.0.0.1"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication environment for the motion control algorithm program </pre>

	<pre> if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } // Enter an infinite loop to keep the program running while (true) { // sleep for 1000 milliseconds std::this_thread::sleep_for(std::chrono::milliseco nds(1000)); } return 0; } </pre>
--	---

3. getMotorNumber Interface

Function Name	getMotorNumber
Function Prototype	uint32_t getMotorNumber();
Description	Retrieves the motor number in the robot.
Parameters	None
Return Value	Returns an unsigned integer representing the total number of motors in the robot.
Remarks	None
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class </pre>

```

#include "limxsdk/humanoid.h"

// use the limxsdk namespace to simplify references
// to the Humanoid class
using namespace limxsdk;

int main(int argc, char *argv[]){
    // obtain the singleton instance of the Humanoid
    class
    Humanoid* robot = Humanoid::getInstance();

    // default IP address of the robot
    std::string robot_ip = "127.0.0.1";
    if (argc > 1)
    {
        // If a command-line argument is provided, use
        it as the robot's IP address
        robot_ip = argv[1];
    }

    // Initialize the communication environment for
    the motion control algorithm program
    if (!robot->init(robot_ip))
    {
        // If initialization fails, terminate the
        program
        exit(1);
    }

    // Obtain the motor number of the robot
    uint32_t motor_num = robot->getMotorNumber();

    // Enter an infinite loop to keep the program
    running
    while (true)
    {
        // sleep for 1000 milliseconds

        std::this_thread::sleep_for(std::chrono::millisecon
        ds(1000));
    }
    return 0;
}

```

4. subscribeImuData Interface

Function Name	subscribeImuData
Function Prototype	<code>void subscribeImuData(std::function<void(const ImuDataConstPtr&)> cb);</code>
Description	Subscribes to the robot's IMU data and triggers the specified callback function whenever new IMU data is received.
Parameters	cb: Callback function to process the incoming IMU data.
Return Value	None
Remarks	IMU Data structure prototype defined as follows: <pre>C++ /** * @struct ImuData * * @brief Represents a data structure for robot IMU information based on sensor feedback. * * This structure encapsulates IMU data, including the accelerometer, gyroscope, and quaternion information. */ struct ImuData { uint64_t stamp; // Timestamp: Recorded in nanoseconds, indicating the time at which the data was recorded or generated. float acc[3]; // Stores IMU accelerometer data to track linear acceleration along the X, Y, and Z axes. float gyro[3]; // Stores IMU gyroscope data to track angular velocity (rotational speed) along the X, Y, and Z axes. float quat[4]; // Stores IMU quaternion values representing the robot's orientation in 3D space (w, x, y, z).</pre>

	<pre>}; // Smart Pointer Type Alias typedef std::shared_ptr<ImuData> ImuDataPtr; typedef std::shared_ptr<ImuData const> ImuDataConstPtr;</pre>
Code Example	<pre>C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "127.0.0.1"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication environment for the motion control algorithm program if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } // Subscribe to robot state updates and specify a callback function robot->subscribeImuData([&](const ImuDataConstPtr&</pre>

```

msg) {
    // Handle the received ImuData within this
    callback
    // Note: The callback function is invoked when
    new ImuData is received
    });

    // Enter an infinite loop to keep the program
    running
    while (true)
    {
        // sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds
(1000));
    }
    return 0;
}

```

5. subscribeRobotState Interface

Function Name	subscribeRobotState
Function Prototype	void subscribeRobotState(std::function<void(const RobotStateConstPtr&)> cb);
Description	Subscribes to receive updates on robots' status
Parameters	cb: callback function, invoked upon receiving a state update, with a constant pointer to a RobotState object as its parameter.
Return Value	None
Remarks	RobotState data structure prototype defined as follows: <pre> C++ /** * @struct RobotState * * @brief Represents a data structure for robot state based on sensor feedback. </pre>

	<pre> * * This structure encapsulates various data points used for monitoring and controlling the robot, including IMU data (accelerometer, gyroscope, quaternion), output torques, current joint angles, velocities, and more. */ struct RobotState { // Default constructor RobotState() { } // Parameterized constructor, initializes vectors tau, q, and dq with size motor_num, all elements set to 0.0. RobotState(int motor_num) : tau(motor_num, 0.0) , q(motor_num, 0.0) , dq(motor_num, 0.0) { } , motor_names(motor_num, "") { } uint64_t stamp; // Timestamp (in nanoseconds), typically indicates the time at which the data was recorded or generated. std::vector<float> tau; // A vector for storing the current estimated output torques of the joints (unit: N·m). std::vector<float> q; // A vector for storing the current joint angles (unit: radians) std::vector<float> dq; // A vector for storing the current joint velocities (unit: radians per second) std::vector<std::string> motor_names; // A vector for storing the names of all joints. }; // Smart Pointer Type Alias typedef std::shared_ptr<RobotState> RobotStatePtr; typedef std::shared_ptr<RobotState const> RobotStateConstPtr; </pre>
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to </pre>

```

import the Humanoid class
#include "limxsdk/humanoid.h"

// use the limxsdk namespace to simplify references
to the Humanoid class
using namespace limxsdk;

int main(int argc, char *argv[]){
    // obtain the singleton instance of the Humanoid
class
    Humanoid* robot = Humanoid::getInstance();

    // default IP address of the robot
    std::string robot_ip = "127.0.0.1";
    if (argc > 1)
    {
        // If a command-line argument is provided, use
it as the robot's IP address
        robot_ip = argv[1];
    }

    // Initialize the communication environment for
the motion control algorithm program
    if (!robot->init(robot_ip))
    {
        // If initialization fails, terminate the
program
        exit(1);
    }

    // Subscribe to robot state updates and specify a
callback function
    robot->subscribeRobotState([&](const
RobotStateConstPtr& msg) {
        // Handle the received ImuData within this
callback
        // Note: The callback function is invoked when
new ImuData is received
    });

    // Enter an infinite loop to keep the program
running
    while (true)
    {

```

	<pre>// sleep for 1000 milliseconds std::this_thread::sleep_for(std::chrono::milliseconds(1000)); } return 0; }</pre>
--	--

6. publishRobotCmd Interface

Function Name	publishRobotCmd
Function Prototype	bool publishRobotCmd(const RobotCmd& cmd);
Description	Publishes a command to control the robot's actions .
Parameters	cmd: a RobotCmd object specifying the desired robot command
Return Value	无
Remarks	RobotCmd data structure prototype defined as follows: <pre>C++ /** * @struct RobotCmd * * @brief Represents a data structure for robot state based on sensor feedback. * * This structure contains various commands that can be used to control the robot, including the desired operating mode, target joint angles, target velocities, target output torques, desired position stiffness, and desired velocity stiffness. */ struct RobotCmd { RobotCmd() { } RobotCmd(int motor_num) : mode(motor_num, 0) , q(motor_num, 0.0)</pre>

	<pre> , dq(motor_num, 0.0) , tau(motor_num, 0.0) , Kp(motor_num, 0.0) , Kd(motor_num, 0.0) { } , motor_names(motor_num, "") { } uint64_t stamp; // Timestamp (in nanoseconds), typically indicates the time at which the data was recorded or generated. std::vector<uint8_t> mode; // 0: Torque control mode; 1: Velocity control mode; 2: Position control mode, default setting: 0 std::vector<float> q; // A vector for storing desired joint angles (unit: radians) std::vector<float> dq; // A vector for storing desired joint velocities (unit: radians per second) std::vector<float> tau; // A vector for storing desired output torques (unit: N·m) std::vector<float> Kp; // A vector for storing desired position stiffness values (unit: N·m per radian) std::vector<float> Kd; // A vector for storing desired velocity stiffness values (unit: N·m per radian per second) std::vector<std::string> motor_names; // Stores the names of the robot joints to be controlled }; // Smart Pointer Type Alias typedef std::shared_ptr<RobotCmd> RobotCmdPtr; typedef std::shared_ptr<RobotCmd const> RobotCmdConstPtr; </pre>
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references </pre>

```

to the Humanoid class
using namespace limxsdk;

int main(int argc, char *argv[]){
    // obtain the singleton instance of the Humanoid
class
    Humanoid* robot = Humanoid::getInstance();

    // default IP address of the robot
    std::string robot_ip = "127.0.0.1";
    if (argc > 1)
    {
        // If a command-line argument is provided, use
it as the robot's IP address
        robot_ip = argv[1];
    }

    // Initialize the communication environment for
the motion control algorithm program
    if (!robot->init(robot_ip))
    {
        // If initialization fails, terminate the
program
        exit(1);
    }

    // Obtain the motor number of the robot
    uint32_t motor_num = robot->getMotorNumber();

    // Create a RobotCmd object that includes the
number of robot motors
    RobotCmd cmd(motor_num);

    // Publish the control command
    robot->publishRobotCmd(cmd);

    // Enter an infinite loop to keep the program
running
    while (true)
    {
        // sleep for 1000 milliseconds

        std::this_thread::sleep_for(std::chrono::millisecon
ds(1000));
    }
}

```

	<pre> } return 0; } </pre>
--	------------------------------------

7. subscribeSensorJoy Interface

Function Name	subscribeSensorJoy
Function Prototype	void subscribeSensorJoy(std::function<void(const SensorJoyConstPtr&)> cb);
Description	Subscribes to the robot's remote controller data during real-machine deployment. When data is received, the specified callback function is invoked and provided with a constant pointer to a SensorJoy structure for processing.
Parameters	cb: Callback Function for receiving remote controller data. The parameter type is <code>SensorJoyConstPtr</code> , a shared pointer to a constant SensorJoy structure.
Return Value	None
Remarks	SensorJoy data structure prototype defined as follows: <pre> C++ /** * @struct SensorJoy * * @brief The structure of the robot remote controller data * * This structure contains timestamp information related to the controller, as well as joystick and button values. */ struct SensorJoy { uint64_t stamp; // Timestamp corresponding to the sensor input time, in nanoseconds std::vector<float> axes; // </pre>

	<pre> Values representing controller joystick manipulation std::vector<int32_t> buttons; // Values representing controller button operation states }; // SensorJoy Smart Pointer Type Alias typedef std::shared_ptr<SensorJoy> SensorJoyPtr; typedef std::shared_ptr<const SensorJoy> SensorJoyConstPtr; </pre>
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "127.0.0.1"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication environment for the motion control algorithm program if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } } </pre>

```

    // Subscribe to the remote controller data of the
robot
    robot->subscribeSensorJoy([&](const
limxsdk::SensorJoyConstPtr &joy) {
        // L1 & R1 press
        if (joy->buttons[4] == 1 && joy->buttons[7] == 1)
        {
            // Perform the corresponding operations here
        }

        // Process the joystick data
        double axes_left_horizontal = joy->axes[0];
        double axes_left_vertical = joy->axes[1];
        double axes_right_horizontal = joy->axes[2];
        double axes_right_vertical = joy->axes[3];
    });

    // Enter an infinite loop to keep the program
running
    while (true)
    {
        // sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds
(1000));
    }
    return 0;
}

```

8. subscribeDiagnosticValue Interface

Function Name	subscribeDiagnosticValue
Function Prototype	void subscribeDiagnosticValue(std::function<void(const DiagnosticValueConstPtr&)> cb);
Description	Subscribes to the robots' diagnostic value and status information. When a diagnostic message is issued, the specified callback function is triggered and receives a constant pointer to a <code>DiagnosticValue</code> structure, enabling real-time monitoring of the

	robot's health status and allowing timely handling of potential issues.
Parameters	cb: Callback function for receiving diagnostic data. The parameter type is <code>DiagnosticValueConstPtr</code> , a shared pointer to a constant <code>DiagnosticValue</code> structure containing fields such as timestamp, level, name, code, and message.
Return Value	None
Remarks	DiagnosticValue data structure prototype defined as follows: <pre> C++ /** * @struct DiagnosticValue * * @brief Structure representing diagnostic values * * This structure contains information about the * diagnostic level, name, code, and message. */ struct DiagnosticValue { enum { OK = 0 }; // Diagnostic level for normal status enum { WARN = 1 }; // Diagnostic level for warning status enum { ERROR = 2 }; // Diagnostic level for error status uint64_t stamp; // Timestamp, in nanoseconds int32_t level; // Diagnostic level associated with the diagnostic value std::string name; // Name identifying the diagnostic value int32_t code; // Code corresponding to the diagnostic value std::string message; // Detailed message related to the diagnostic value }; // DiagnosticValue Smart pointer type definition typedef std::shared_ptr<DiagnosticValue> </pre>

	<pre>DiagnosticValuePtr; typedef std::shared_ptr<DiagnosticValue const> DiagnosticValueConstPtr;</pre>
Code Example	<pre>C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "127.0.0.1"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication environment for the motion control algorithm program if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } // Subscribe to robot diagnostic data robot->subscribeDiagnosticValue([&](const DiagnosticValueConstPtr& msg) { // Handle the robot diagnostic value here // For example, actions can be taken based on</pre>

```

the diagnostic level and message
    std::cout << "Diagnostic Value: " << msg->name
<< std::endl;
    std::cout << "Level: " << msg->level <<
std::endl;
    std::cout << "Code: " << msg->code << std::endl;
    std::cout << "Message: " << msg->message <<
std::endl;
    });

    // Enter an infinite loop to keep the program
    running
    while (true)
    {
        // sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::millisecond
s(1000));
    }
    return 0;
}

```

9. publishJsonMessage Interface

Function Name	publishJsonMessage
Function Prototype	void publishJsonMessage(const std::string &json_payload);
Description	Sends a JSON-formatted message to the robot following the <i>High-Level Application Protocol Interface</i> .
Parameters	JSON_payload: A JSON string conforming to the <i>High-Level Application Protocol specification</i>. Example: {"accid": "xxx", "title": "request_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
Return Value	None

Remarks	Valid only in high-level development mode.
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "10.192.1.2"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication runtime environment if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } // The JSON protocol content to be sent std::string json_payload = R"({ "accid": "HU_D03_01", # Replace with the real robot serial number (SN) "title": "request_get_joint_state", "timestamp": 1672373633989, "guid": "746d937cd8094f6a98c9577aaf213d98", </pre>

	<pre> "data": {} }); // Publish control command robot->publishJsonMessage(json_payload); // Enter an infinite loop to keep the program running while (true) { // sleep for 1000 milliseconds std::this_thread::sleep_for(std::chrono::milliseconds(1000)); } return 0; } </pre>
--	---

10. subscribeJsonMessage Interface

Function Name	subscribeJsonMessage
Function Prototype	void subscribeJsonMessage(std::function<void(const std::string &)> cb);
Description	Registers a callback function to handle responses and notifications from the robot's <i>High-Level Application Protocol Interface</i>. The callback is triggered in the following cases: 1. When the robot returns a response to a previously sent JSON command (<code>publishJsonMessage</code>). 2. When the robot actively sends an unsolicited notification.
Parameters	cb: callback function, prototype as void(const std::string &json_payload) json_payload includes: • Command response: {"accid": "xxx", "title": "response_xxx", "timestamp": xxx, "guid": "xxx", "data": {}} • Notification: {"accid": "xxx", "title": "notify_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}

Return Value	None
Remarks	Valid only in high-level development mode
Code Example	<pre> C++ 代码块 #include <thread> // include the limxsdk: Humanoid header file to import the Humanoid class #include "limxsdk/humanoid.h" // use the limxsdk namespace to simplify references to the Humanoid class using namespace limxsdk; int main(int argc, char *argv[]){ // obtain the singleton instance of the Humanoid class Humanoid* robot = Humanoid::getInstance(); // default IP address of the robot std::string robot_ip = "10.192.1.2"; if (argc > 1) { // If a command-line argument is provided, use it as the robot's IP address robot_ip = argv[1]; } // Initialize the communication runtime environment if (!robot->init(robot_ip)) { // If initialization fails, terminate the program exit(1); } // Handle responses and notifications from the robot's "High-Level Application Protocol Interface" calls. robot->subscribeJsonMessage([&](const std::string </pre>

```

& json_payload) {
    std::cout << json_payload << std::endl;
});

// Enter an infinite loop to keep the program
running
while (true)
{
    // sleep for 1000 milliseconds

    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
}
return 0;
}

```

11. Reference Example

Github: <https://github.com/limxdynamics/humanoid-rl-deploy-ros>

3.5.2 Python Motion Control API

Provides a [Python motion-control API](#) with equivalent functionality to the C++ motion-control interface, enabling developers who are not familiar with C++ to develop motion-control algorithms in Python.

1. Install Motion Control Development Library

- **Linux x86_64 Environment**

```

Bash
pip install python3/amd64/limxsdk-*-py3-none-any.whl

```

- **Linux aarch64 Environment**

```

Bash
pip install python3/aarch64/limxsdk-*-py3-none-any.whl

```

- **Windows Environment**

```

Bash
pip install python3/win/limxsdk-*-py3-none-any.whl

```

2. `__init__` Interface

Function Name	<code>__init__</code>
Function Prototype	<code>def __init__(self, robot_type: robot.RobotType)</code>
Description	Specifies the robot type during initialization and creates a local robot instance of the corresponding type.
Parameters	<code>robot_type</code> : an enumeration value specifying the robot type, where <code>RobotType.Humanoid</code> represents a bipedal humanoid robot.
Return Value	None
Remarks	None
Code Example	<pre>Python 代码块 import sys import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid)</pre>

3. `init` Interface

Function Name	<code>init</code>
Function Prototype	<code>def init(self, robot_ip: str = "127.0.0.1")</code>
Description	Initializes the communication and runtime environment for the motion control algorithm, typically called before other interfaces in the main function.

Parameters	robot_ip: The IP address of the robot. Use "127.0.0.1" for simulation and "10.192.1.2" for real robots.
Return Value	Success: return True Failure: return False
Remarks	None
Code Example	Python 代码块 <pre> import sys import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid) robot_ip = "127.0.0.1" # Check whether a command-line argument is # provided as the robot's IP address if len(sys.argv) > 1: robot_ip = sys.argv[1] # Initialize the robot's communication runtime # environment using the IP address if not robot.init(robot_ip): sys.exit() </pre>

4. getMotorNumber Interface

Function Name	getMotorNumber
Function Prototype	def getMotorNumber(self)
Description	Retrieves the motor number in the robot.
Parameters	None

Return Value	Returns an unsigned integer representing the total number of motors in the robot.
Remarks	Typically, a bipedal humanoid robot is equipped with 6 motors.
Code Example	<pre> Python 代码块 import sys import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid) robot_ip = "127.0.0.1" # Check whether a command-line argument is provided as the robot's IP address if len(sys.argv) > 1: robot_ip = sys.argv[1] # Initialize the robot using robot_ip if not robot.init(robot_ip): sys.exit() # Obtain the number of motors in the robot motor_number = robot.getMotorNumber() </pre>

5. subscribelmuData Interface

Function Name	subscribelmuData
Function Prototype	<code>def subscribelmuData(self, callback: Callable[[datatypes.ImuData], Any])</code>
Description	Subscribes to the robot's IMU data and triggers the specified callback function whenever new IMU data is received.
Parameters	Callback: Callback function to process the new IMU data.

Return Value	Success: return <code>True</code> Failure: return <code>False</code>
Remarks	<code>datatypes.ImuData</code> structure prototype defined as follows: <pre> Python import sys class ImuData(object): __slots__ = ['stamp', 'acc', 'gyro', 'quat'] def __init__(self): self.stamp = 0 # Timestamp: Typically indicates the time at which the data was recorded or generated, in nanoseconds self.acc = [0. for x in range(0, 3)] # Stores IMU (Inertial Measurement Unit) accelerometer data, used to track linear acceleration along the three axes self.gyro = [0. for x in range(0, 3)] # Stores IMU gyroscope data, used to track angular velocity or rotational speed self.quat = [0. for x in range(0, 4)] # Stores IMU quaternion data, representing orientation in 3D space </pre>
Code Example	<pre> Python 代码块 import sys from functools import partial import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes class RobotReceiver: # Subscribe to the robot's IMU data def imuDataCallback(self, imu: datatypes.ImuData): print("\n-----\nrobot_state:" + \ "\n stamp: " + str(imu.stamp) + \ "\n acc: " + str(imu.acc) + \ "\n gyro: " + str(imu.gyro) + \ "\n quat: " + str(imu.quat)) </pre>

```

if __name__ == '__main__':
    # Create a Robot instance of type Humanoid
    robot = Robot(RobotType.Humanoid)

    robot_ip = "127.0.0.1"
    # Check whether a command-line argument is
    provided for the robot's IP address
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

    # Initialize the robot using robot_ip
    if not robot.init(robot_ip):
        sys.exit()

    # Create a RobotReceiver instance to handle
    callbacks
    receiver = RobotReceiver()

    # Create a partial function for the callback
    imuDataCallback =
    partial(receiver.imuDataCallback)

    # Subscribe to the robot's IMU data
    robot.subscribeImuData(imuDataCallback)

```

6. subscribeRobotState Interface

Function Name	subscribeRobotState
Function Prototype	def subscribeRobotState(self, callback: Callable[[<code>datatypes.RobotState</code>], Any])
Description	Subscribes to receive updates on robots' status
Parameters	Callback: callback function, invoked upon receiving a state update. Its parameter points to a <code>datatypes.RobotState</code> object. <code>datatypes.RobotState</code> structure fields: - stamp: Timestamp, indicates when the data was recorded or generated. - tau: Vector storing the estimated output torques (in

	newton-meters). ◦ q: Vector storing the current joint positions (in radians). ◦ dq: Vector storing the current joint velocities (in radians per second). ◦ motor_names: store the corresponding joint name.
Return Value	Success: return <code>True</code> Failure: return <code>False</code>
Remarks	datatypes.RobotState structure prototype defined as follows: <pre> Python import sys class RobotState(object): __slots__ = ['stamp', 'tau', 'q', 'dq'] def __init__(self): self.stamp = 0 # Timestamp: Typically indicates the time at which the data was recorded or generated, in nanoseconds self.tau = [] # Stores the vector of current estimated output torques (unit: N·m) self.q = [] # Stores the vector of current joint angles (unit: radians) self.dq = [] # Stores the vector of current joint velocities (unit: radians per second) self.motor_names = [] # Stores the corresponding joint names </pre>
Code Example	<pre> Python 代码块 import sys from functools import partial import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes class RobotReceiver: # Callback function for receiving the robot's state def robotStateCallback(self, robot_state: datatypes.RobotState): </pre>

```

        print("\n-----\nrobot_state:" + \
              "\n  stamp: " + \
str(robot_state.stamp) + \
              "\n  tau: " + str(robot_state.tau) + \
              "\n  q: " + str(robot_state.q) + \
              "\n  dq: " + str(robot_state.dq))

if __name__ == '__main__':
    # Create a Robot instance of type Humanoid
    robot = Robot(RobotType.Humanoid)

    robot_ip = "127.0.0.1"
    # Check whether a command-line argument is
    provided for the robot's IP address
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

    # Initialize the robot using robot_ip
    if not robot.init(robot_ip):
        sys.exit()

    # Create a RobotReceiver instance to handle
    callbacks
    receiver = RobotReceiver()

    # Create a partial function for the callback
    robotStateCallback =
    partial(receiver.robotStateCallback)

    # Subscribe to the robot's IMU data
    robot.subscribeRobotState(robotStateCallback)

```

7. publishRobotCmd Interface

Function Name	publishRobotCmd
Function Prototype	def publishRobotCmd (self, cmd: datatypes.RobotCmd)
Description	Publishes a command to control the robot's actions .

Parameters	cmd: A <code>datatypes.RobotCmd</code> object representing the desired robot command, containing the following fields: - stamp: Timestamp in nanoseconds indicating when the data was recorded or generated. - q: Vector storing the desired joint positions (in radians). - dq: Vector storing the desired joint velocities (in radians per second). - tau: Vector storing the desired output torques (in newton-meters). - Kp: Vector storing the desired position stiffness (in newton-meters per radian). - Kd: Vector storing the desired velocity stiffness (in newton-meters per radian per second). - motor_names: store the joint names that need to be controlled.
Return Value	Success: return <code>True</code> Failure: return <code>False</code>
Remarks	<code>datatypes.RobotCmd</code> structure prototype defined as follows: <pre> Python import sys class RobotCmd(object): __slots__ = ['stamp', 'mode', 'q', 'dq', 'tau', 'Kp', 'Kd'] def __init__(self): self.stamp = 0 # Timestamp (in nanoseconds) , typically indicates the time at which the data was recorded or generated. self.mode = [] # The robot's desired working mode self.q = [] # A vector for storing desired joint angles (unit: radians) self.dq = [] # A vector for storing desired joint velocities (unit: radians per second) self.tau = [] # A vector for storing desired output torques (unit: N·m) self.Kp = [] # A vector for storing </pre>

	<pre> desired position stiffness values (unit: N·m per radian) self.Kd = [] # A vector for storing desired velocity stiffness values (unit: N·m per radian per second) self.motor_names = [] # Stores the names of the robot joints to be controlled </pre>
Code Example	<pre> Python 代码块 import sys import time import limxsdk.robot.Rate as Rate import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid) robot_ip = "127.0.0.1" # Check whether a command-line argument is provided as the robot's IP address if len(sys.argv) > 1: robot_ip = sys.argv[1] # Initialize the robot using robot_ip if not robot.init(robot_ip): sys.exit() # Obtain information on joint offsets, joint limits, and the number of motors joint_offset = robot.getJointOffset() joint_limit = robot.getJointLimit() motor_number = robot.getMotorNumber() # Main loop to continuously publish robot commands rate = Rate(500) # 1500 Hz cmd_msg = datatypes.RobotCmd() while True: # Set default values for timestamp, control mode, joint positions, velocities, </pre>

	<pre> torques, Kp, and Kd cmd_msg.stamp = time.time_ns() cmd_msg.mode = [1.0 for _ in range(motor_number)] cmd_msg.q = [1.0 for _ in range(motor_number)] cmd_msg.dq = [1.0 for _ in range(motor_number)] cmd_msg.tau = [1.0 for _ in range(motor_number)] cmd_msg.Kp = [1.0 for _ in range(motor_number)] cmd_msg.Kd = [1.0 for _ in range(motor_number)] robot.publishRobotCmd(cmd_msg) # Publish the robot commands rate.sleep() # Control the loop frequency </pre>
--	---

8. subscribeSensorJoy Interface

Function Name	subscribeSensorJoy
Function Prototype	def subscribeSensorJoy(self, callback: Callable[[datatypes.SensorJoy], Any])
Description	Subscribes to the robot's remote controller data during real-machine deployment. When data is received, the specified callback function is invoked and provided with a constant pointer to a <code>datatypes.SensorJoy</code> structure for processing.
Parameters	callback: Callback Function for receiving remote controller data with parameter type <code>SensorJoyConstPtr</code> .
Return Value	Success: return <code>True</code> Failure: return <code>False</code>
Remarks	<code>datatypes.SensorJoy</code> structure prototype defined as follows: <pre>Python</pre>

	<pre> import sys class SensorJoy(object): __slots__ = ['stamp', 'axes', 'buttons'] def __init__(self): self.stamp = 0 # Timestamp corresponding to the sensor input time, in nanoseconds self.axes = [] # Values representing controller joystick manipulation self.buttons = [] # Values representing controller button operation states </pre>
Code Example	<pre> Python 代码块 import sys import time from functools import partial import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes class RobotReceiver: # Callback function for receiving the remote controller data def sensorJoyCallback(self, sensor_joy: datatypes.SensorJoy): print("\n-----\nsensor_joy:" + \ "\n stamp: " + str(sensor_joy.stamp) + \ "\n axes: " + str(sensor_joy.axes) + \ "\n buttons: " + str(sensor_joy.buttons)) if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid) robot_ip = "127.0.0.1" # Check whether a command-line argument is provided for the robot's IP address if len(sys.argv) > 1: robot_ip = sys.argv[1] </pre>

	<pre> # Initialize the robot using robot_ip if not robot.init(robot_ip): sys.exit() # Create a RobotReceiver instance to handle callbacks receiver = RobotReceiver() # Create a partial function for the callback sensorJoyCallback = partial(receiver.sensorJoyCallback) # Subscribe to the robot's IMU data robot.subscribeSensorJoy(sensorJoyCallback) </pre>
--	---

9. subscribeDiagnosticValue Interface

Function Name	subscribeDiagnosticValue
Function Prototype	def subscribeDiagnosticValue(self, callback: Callable[[datatypes.DiagnosticValue], Any])
Description	Subscribes to the robot's diagnostic and status updates. Upon receiving a diagnostic message, the callback function is triggered with a <code>datatypes.DiagnosticValue</code> object, allowing continuous health monitoring and prompt response to detected issues.
Parameters	Callback function for receiving diagnostic data. The parameter type is <code>datatypes.DiagnosticValue</code> , which contains fields such as timestamp, level, name, code, and message.
Return Value	Success: return <code>True</code> Failure: return <code>False</code>
Remarks	<code>datatypes.DiagnosticValue</code> structure prototype defined as follows: <pre> Python import sys </pre>

```

class DiagnosticValue(object):
    __slots__ =
['stamp','level','name','code','message']
    def __init__(self):
        self.stamp = 0 # Timestamp, in
nanoseconds
        self.level = 0 # Diagnostic level
associated with the diagnostic value - 0: OK,
1: WARN, 2: ERROR
        self.name = '' # Name identifying the
diagnostic value
        self.code = 0 # Code corresponding to
the diagnostic value
        self.message = '' # Detailed message
related to the diagnostic value

```

Code
Example

Python
代码块

```

import sys
from functools import partial
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType
import limxsdk.datatypes as datatypes

class RobotReceiver:
    # Callback function for receiving diagnostic
value
    def diagnosticValueCallback(self,
diagnostic_value: datatypes.DiagnosticValue):
        print("\n-----\ndiagnostic_value:" + \
            "\n stamp: " +
str(diagnostic_value.stamp) + \
            "\n name: " + diagnostic_value.name
+ \
            "\n level: " +
str(diagnostic_value.level) + \
            "\n code: " +
str(diagnostic_value.code) + \
            "\n message: " +
diagnostic_value.message)

if __name__ == '__main__':
    # Create a Robot instance of type Humanoid

```

```

robot = Robot(RobotType.Humanoid)

robot_ip = "127.0.0.1"
# Check whether a command-line argument is
provided for the robot's IP address
if len(sys.argv) > 1:
    robot_ip = sys.argv[1]

# Initialize the robot using robot_ip
if not robot.init(robot_ip):
    sys.exit()

# Create a RobotReceiver instance to handle
callbacks
receiver = RobotReceiver()

# Create a partial function for the callback
diagnosticValueCallback =
partial(receiver.diagnosticValueCallback)

# Subscribe to the robot's IMU data

robot.subscribeDiagnosticValue(diagnosticValueCallb
ack)

```

10. publishJsonMessage Interface

Function Name	publishJsonMessage
Function Prototype	def publishJsonMessage(self, json_payload: str)
Description	Sends a JSON-formatted message to the robot following the <i>High-Level Application Protocol Interface</i> .
Parameters	json_payload: A JSON string conforming to the <i>High-Level Application Protocol specification</i>. Example: {"accid": "xxx", "title": "request_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}

Return Value	None
Remarks	Valid only in high-level development mode.
Code Example	<pre> Python 代码块 import sys import time import limxsdk.robot.Rate as Rate import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes if __name__ == '__main__': # Create a Robot instance of type Humanoid robot = Robot(RobotType.Humanoid) robot_ip = "10.192.1.2" # Check whether a command-line argument is provided for the robot's IP address if len(sys.argv) > 1: robot_ip = sys.argv[1] # Initialize the robot using robot_ip if not robot.init(robot_ip): sys.exit() # Set the protocol content to be sent json_payload = '''{ "accid": "HU_D03_01", # Replace with the real robot serial number (SN) "title": "request_get_joint_state", "timestamp": 1672373633989, "guid": "746d937cd8094f6a98c9577aaf213d98", "data": {} }''' # Send the JSON protocol. robot.publishJsonMessage(json_payload) # Keep the program running try: while True: </pre>

	<pre> time.sleep(1) except KeyboardInterrupt: print("Program interrupted by the user") </pre>
--	---

11. subscribeJsonMessage Interface

Function Name	subscribeJsonMessage
Function Prototype	def subscribeJsonMessage(self, callback: Callable[[str], Any])
Description	Registers a callback function to handle responses and notifications from the robot's <i>High-Level Application Protocol Interface</i>. The callback is triggered in the following cases: 1. When the robot returns a response to a previously sent JSON command (<code>publishJsonMessage</code>). 2. When the robot actively sends an unsolicited notification.
Parameters	callback: callback function, json_payload includes: • Command response: {"accid": "xxx", "title": "response_xxx", "timestamp": xxx, "guid": "xxx", "data": {}} • Notification: {"accid": "xxx", "title": "notify_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
Return Value	None
Remarks	Valid only in high-level development mode
Code Example	<pre> Python 代码块 import sys import time from functools import partial import limxsdk.robot.Robot as Robot import limxsdk.robot.RobotType as RobotType import limxsdk.datatypes as datatypes </pre>

```

class RobotReceiver:
    # Handle responses and notifications from the
    robot's "High-Level Application Protocol
    Interface" calls.
    def jsonMessageCallback(self, json_payload:
    str):
        print("\n-----\njson_payload:" +
        json_payload)

if __name__ == '__main__':
    # Create a Robot instance of type Humanoid
    robot = Robot(RobotType.Humanoid)

    robot_ip = "10.192.1.2"
    # Check whether a command-line argument is
    provided for the robot's IP address
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

    # Initialize the robot using robot_ip
    if not robot.init(robot_ip):
        sys.exit()

    # Create a RobotReceiver instance to handle
    callbacks
    receiver = RobotReceiver()

    # Create a partial function for the callback
    jsonMessageCallback =
    partial(receiver.jsonMessageCallback)

    # Subscribe to the robot's IMU data

    robot.subscribeJsonMessage(jsonMessageCallback)

```

12. Reference Example

Github: <https://github.com/limxdynamics/humanoid-rl-deploy-python>

3.6 Check and Set the Robot Model

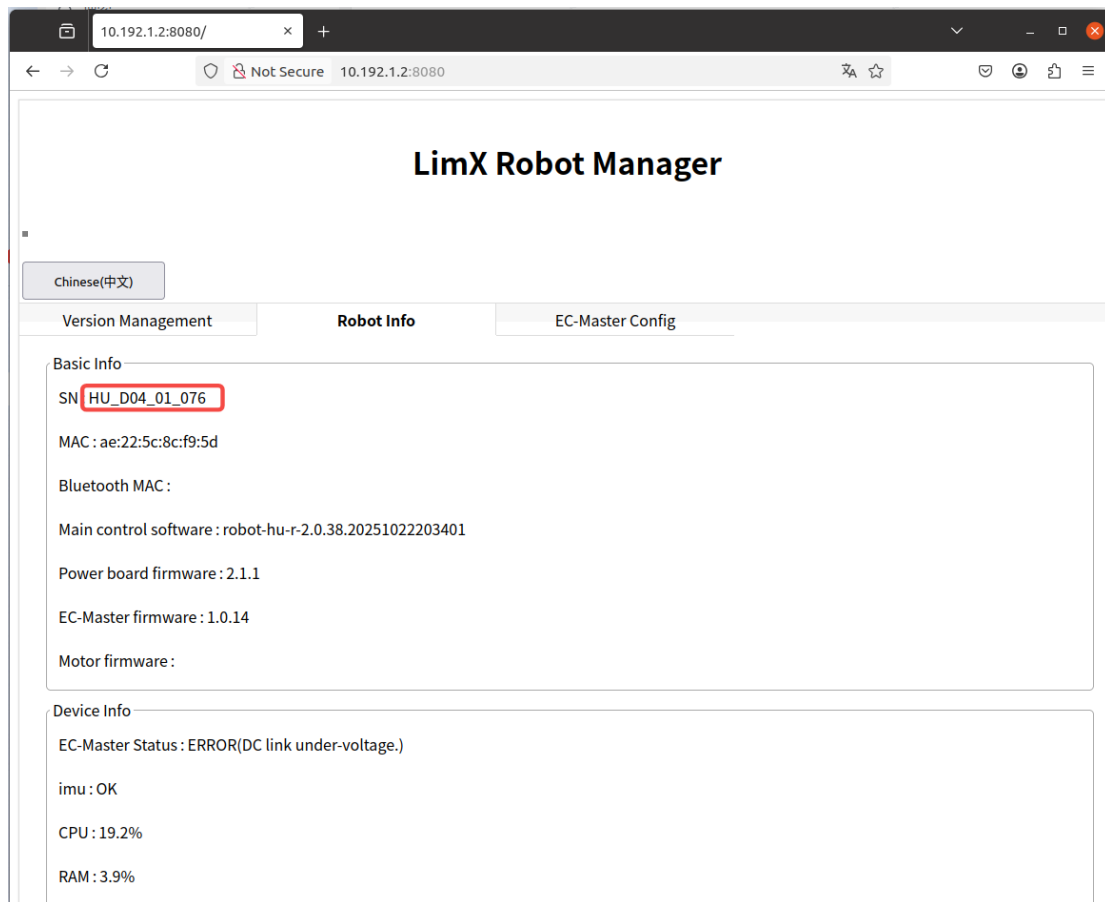
When compiling or running RL training, control algorithms and simulation programs, selecting the correct robot model is essential. Check the robot model and set it in the environment variable `ROBOT_TYPE` to ensure the correct model is identified and applied across different tasks.

Steps to View and Configure the Robot Model:

1. Connect to the robot's Wi-Fi hotspot and enter the password: `12345678`



2. Open a web browser and navigate to `http://10.192.1.2:8080`. The page displays the SN (serial number) — for example, `HU_D03_03_001` — where `HU_D03_03` represents the robot model, as shown below.



3. Set the robot model: Open a Bash terminal and run the following shell command to set the robot model. This ensures that the correct robot model information is recognized during secondary development.

```
Bash
echo 'export ROBOT_TYPE=HU_D03_03' >> ~/.bashrc && source
~/.bashrc
```

3.7 Robot Simulator

MuJoCo is a lightweight, high-performance physics simulator designed for multi-joint robots and mechanical systems.

It features an efficient physics engine capable of accurately simulating contact and friction, and can operate independently without relying on ROS.

3.7.1 Running the Simulator

1. Environment Requirements: Python 3.8 or higher is recommended
2. Open a Bash Terminal
3. Download the MuJoCo Simulator Code:

Python

```
git clone --recurse git@github.com:limxdynamics/humanoid-  
mujoco-sim.git
```

4. Install the Motion Control Development Library:

- Linux x86_64 Environment

Python

```
pip install humanoid-mujoco-sim/limxsdk-  
lowlevel/python3/amd64/limxsdk-*-py3-none-any.whl
```

- Linux aarch64 Environment

Python

```
pip install humanoid-mujoco-sim/limxsdk-  
lowlevel/python3/aarch64/limxsdk-*-py3-none-any.whl
```

5. Set the Robot Model: Please refer to the “*check and set the Robot Model*” section to check your robot model. If not yet configured, please follow the steps below:

- List available robot types using the shell command `tree -L 3 -P "meshes" -I "urdf|world|xml|usd" humanoid-mujoco-sim/humanoid-description:`

Python

代码块

```
limx@limx:~$ tree -L 3 -P "meshes" -I "urdf|world|xml|usd"  
humanoid-mujoco-sim/humanoid-description  
humanoid-mujoco-sim/humanoid-description  
├── HU_D03_description  
│   └── meshes  
│       └── HU_D03_03  
└── HU_D04_description  
    └── meshes  
        └── HU_D04_01
```

- For example, to set the robot model type **HU_D04_01** (replace with your actual model):

Python

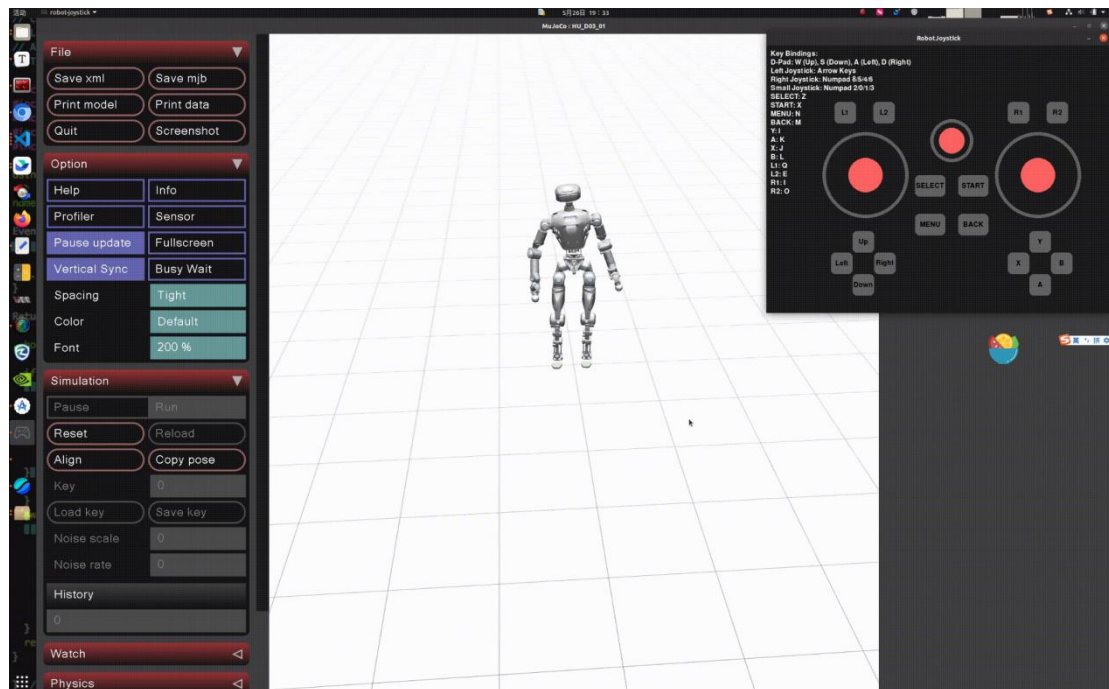
```
echo 'export ROBOT_TYPE=HU_D04_01' >> ~/.bashrc && source  
~/.bashrc
```

6. Run the MuJoCo simulator:

```
Python  
python humanoid-mujoco-sim/simulator.py
```

3.7.2 Demonstration Results

Actual performance may vary depending on your system configuration.



3.8 RL Algorithm Deployment

3.8.1 Deployment with Standard C++

- **Github Repository:** <https://github.com/limxdynamics/humanoid-rl-deploy-cpp>
- A lightweight algorithm framework implemented in standard C++, enabling fast deployment of trained models without requiring ROS1 or ROS2.

3.8.2 Deployment with Python

- **Github Repository:** <https://github.com/limxdynamics/humanoid-rl-deploy-python>
- A Python-based reinforcement learning deployment framework that simplifies the process of deploying trained models on Oli.

3.8.3 Deployment with ROS2

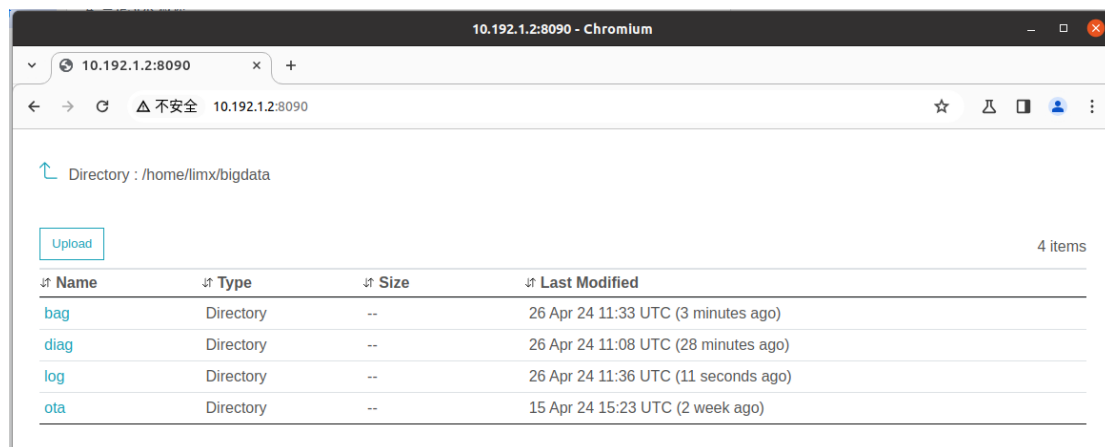
- **Github Repository:** <https://github.com/limxdynamics/humanoid-rl-deploy-ros2>
- A reinforcement learning deployment framework based on [ROS2](#), allowing rapid deployment of trained models on Oli.

3.8.4 Deployment with ROS1

- **Github Repository:** <https://github.com/limxdynamics/humanoid-rl-deploy-ros>
- A reinforcement learning deployment framework based on [ROS1](#), enabling efficient deployment of trained models on Oli.

3.9 Logs and Data Packages

- **Automatic Data Recording:** The robot system automatically records essential data, including IMU data (ImuData), state data (/joint/state), control data (/joint/cmd), and runtime logs. These records are critical for motion control analysis and performance evaluation.
- **Accessing and Downloading Data:** The robot stores runtime log data for troubleshooting and performance optimization. To access the data, connect your computer to the robot's Wi-Fi hotspot, then open a browser at <http://10.192.1.2:8090> to download the desired datasets.



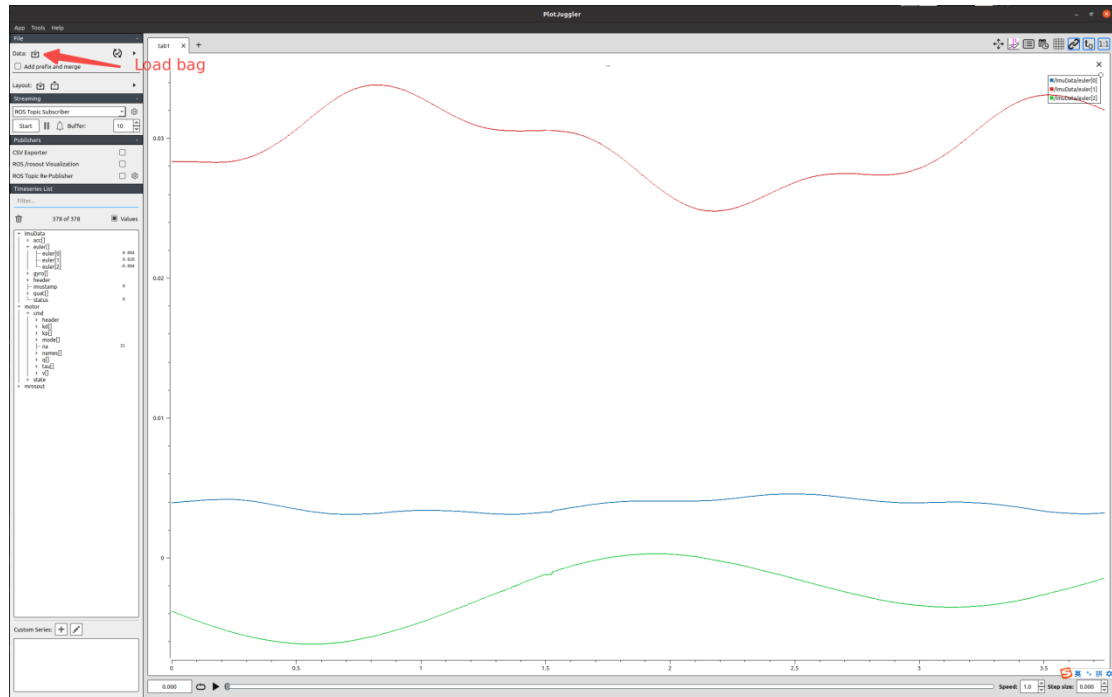
3.9.1 Data Package Visualization and Analysis

- **Downloading Data Packages:** After downloading a .bag file, you can use PlotJuggler to visualize and analyze the data. If the downloaded file is in .bag.active format, run the following shell command to reindex it and generate a new .bag file for PlotJuggler to load properly.

Bash

```
roscat reindex your_file.bag.active  
mv your_file.bag.active your_file.bag
```

- **Visualization:** Launch the PlotJuggler visualization tool using the shell command `roscat plotjuggler plotjuggler -n`, then load the data packages as shown below:



3.9.2 Logs and Diagnostic Trace Data

Structured log and diagnostic trace data are recorded for system monitoring, as shown below. These datasets can be used for troubleshooting and performance optimization when needed.

The screenshot shows a web browser window displaying a directory listing for `/home/limx/bigdata`. The directory contains three items: `bag`, `log`, and `ota`. The `bag` and `log` items are highlighted with a red box. The table below shows the details of these items.

Name	Type	Size	Last Modified	More Actions
bag	Directory	--	07 Nov 25 14:44 CST (52 seconds ago)	Link Icon QR Share
log	Directory	--	07 Nov 25 14:40 CST (4 minutes ago)	Link Icon QR Share
ota	Directory	--	26 Aug 25 23:37 CST (2 months ago)	Link Icon QR Share

GoHFS v0.1.4

3.10 Robot Software Upgrade

You can upgrade the robot software via the web management interface using a locally downloaded software package. The steps are as follows:

1. **Connect to Wi-Fi:**

- Connect to your robot's Wi-Fi hotspot, password: 12345678

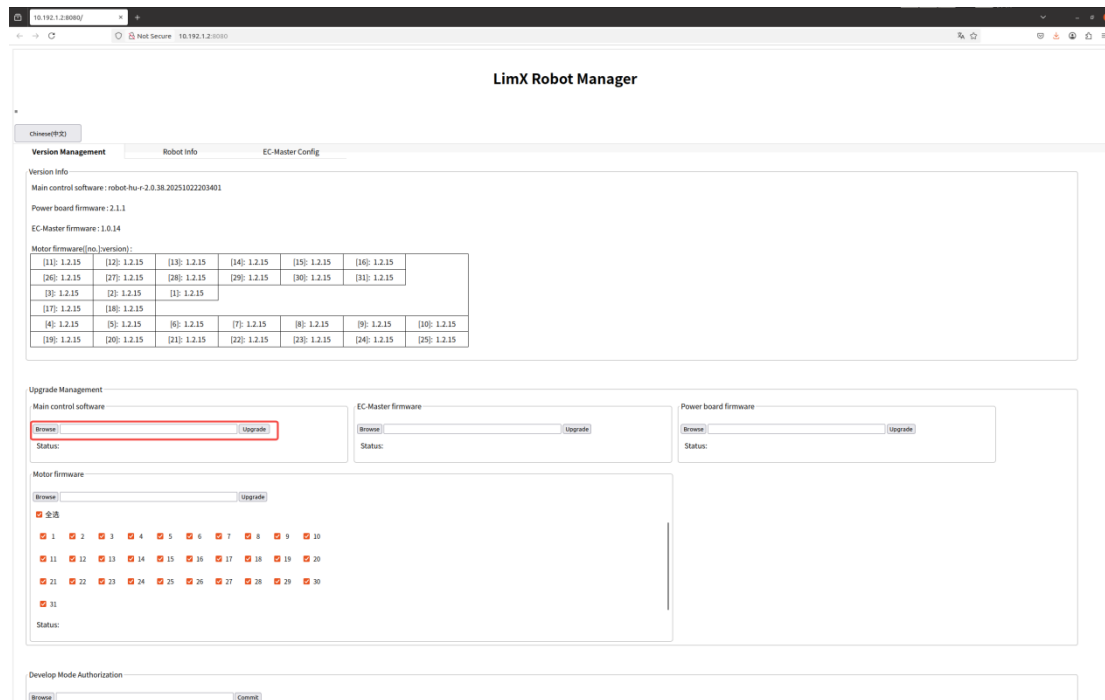


2. **Access The Management Page:**

- Open a browser and enter <http://10.192.1.2:8080> to access the robot management interface

3. **Select and Upgrade Software :**

- Navigate to "Version Management" -> "Browse" -> "Upgrade"
- After the upgrade is complete, the robot's main control computer will restart automatically.



3.11 Developer Computer

The developer computer is primarily used for developing robot-related algorithms and applications. It can be accessed via the robot's onboard Wi-Fi network. The steps are as follows:

1. Connect to Wi-Fi:

- Connect to your robot's Wi-Fi hotspot, password: 12345678



2. SSH Login:

- IP Address: 10.192.1.3
- Password: 123456
- Use the following command in the terminal (fingerprint confirmation needed on first login)

```
Python  
ssh guest@10.192.1.3
```

- System Configuration:
 - Operating System: Ubuntu 22.04 (Jetpack 6.2.1)
 - ROS2: ROS2 Humble (default installation)
 - ROS1: ROS1 Noetic (Docker-based environment), to enter the ROS1 Docker environment:

```
Python  
sudo docker exec -it ros_noetic /bin/bash
```

3.12 RealSense Camera Data Acquisition

1. **Log in to the Developer Computer:**
 - Please follow the procedures described in the “*Developer Computer*” section to log in to the computer
2. **Launch the RealSense ROS Node to Acquire Camera Data:**
 - launch instructions address: <https://github.com/IntelRealSense/realsense-ros>
 - The computer is preinstalled with the RealSense Camera SDK:
 - version: v2.56.3, Official download link: <https://github.com/IntelRealSense/librealsense/releases/tag/v2.56.3>. You may develop your own applications based on this official SDK to acquire camera data.
3. **Example Code Description:**
 - **Camera Naming Convention:**
 - **Default Naming:** The script automatically assigns topic name prefixes for multiple cameras using the format “*camera*” followed by an index (e.g., *camera0*, *camera1*).
 - **Custom Naming:** The script can be modified to assign topic prefixes based on the camera's Serial Number instead of using the default “*camera + counter*” naming scheme.

The following code example demonstrates how to acquire data from multiple cameras:

1. Save the script below as: `rs_camera.sh`

```
Bash
#!/bin/bash

source /opt/ros/noetic/setup.bash

# Function to detect connected RealSense cameras
detect_cameras() {
    # List all connected RealSense cameras, excluding ASIC
    # Serial Number
    serial_numbers=($(rs-enumerate-devices | grep "Serial
Number" | grep -v "Asic" | awk '{print $NF}'))
    echo "${serial_numbers[@]}"
}

# Loop to check for the launch flag file and connected cameras
while true; do
```

```

serial_numbers=$(detect_cameras))

# Check if any cameras were found
if [ ${#serial_numbers[@]} -gt 0 ]; then
    echo "Detected ${#serial_numbers[@]} cameras."
    break # Exit the loop if cameras are detected
else
    echo "No RealSense cameras detected. Retrying in 5
seconds..."
    sleep 5 # Wait for a while before retrying
fi
done

# Automatically start a ROS node for each detected camera
if [ ${#serial_numbers[@]} -gt 0 ]; then
    for i in "${!serial_numbers[@]}"; do
        serial=${serial_numbers[$i]}

        # Default camera naming using index (camera0,
camera1, ...)
        # Customize camera naming by modifying the code below
camera_topic="camera$i"

        # Example: Custom camera naming based on serial number
        # Uncomment and replace with your actual serial
numbers
        # if [[ "$serial" == "0123456789" ]]; then
        #     camera_topic="head" # Name specific camera as
"head"
        # elif [[ "$serial" == "9876543210" ]]; then
        #     camera_topic="chest" # Name another camera as
"chest"
        # fi

        echo "Starting ROS node for camera $serial (topic
prefix: $camera_topic)..."
        roslaunch realsense2_camera rs_camera.launch \
            serial_no:=$serial \
            camera:=$camera_topic \
            enable_pointcloud:=True \
            enable_accel:=True \
            enable_gyro:=True \
            enable_sync:=True \
            unite_imu_method:=linear_interpolation &

```

```

        sleep 10 # Optional: wait a bit before starting the
next camera
done

# Wait for all background processes to finish
wait
else
    echo "No cameras to start."
fi

```

2. Execute the script in the terminal to launch the camera node

```

Shell
/bin/bash rs_camera.sh

```

3. In another terminal, use `rostopic list` to verify the results

```

Shell
rostopic list

```

FAQ

Issues Category	Issue Description	Possible Causes	Troubleshooting & Solutions
Power Related	Failure to Power On	- Depleted battery - Emergency stop button pressed	- Check charge status and recharge - Ensure the emergency stop button is released
	Sudden Motor Shutdown	- Safety protection triggered - Loose connection - Power supply abnormality	- Check the emergency stop status. - Reconnect the power cable or check connectors - Restart the system.
	Failure to	Incorrect connection	Insert the aviation plug first,

	Charge	sequence • Adapter malfunction • Loose connection	then power on. • Check the indicator light (Red = Charging, Green = Full) • Tighten screw sleeve, check contact
Control & Execution	No Buzzer Prompt After Startup	• Self-check incomplete • Remote controller pairing failed	• Wait 1 minute for initialization. • Re-pair the remote controller
	Remote Controller Unresponsive	• Pairing unsuccessful • Remote controller low battery • Severe wireless interference	• Long-press the "Sync Button" for 3s to re-pair. • Replace the remote controller battery. • Move away from strong interference sources.
	Motor No Response	• No power applied • Pairing incomplete • Control signal lost	• Confirm the remote controller is paired. • Press the right stick to power on • Check the remote controller status
	Zero Calibration Failure	• Restricted space • Incorrect posture • Command not sent successfully	• Ensure no obstacles within 2m/6' 6.74" • Perform in a natural hanging position • Check the remote controller button operation
Communication & Network	Unable to connect to Wi-Fi	• Weak signal • Incorrect password • Module malfunction	• Move closer to the robot and reconnect. • Verify the password is <u>12345678</u> • Restart the robot

	Unable to access 10.192.1.2	• IP not on the same subnet • Network configuration error	• For wired connection: manually set the IP to 10.192.1.x subnet. • For wireless connection: automatic IP address assignment, no manual configuration needed.
--	------------------------------------	--	--

Common Errors and Descriptions

Fault Source	Error Code	Description
Device Overall Status	0x0001	IMU abnormality
	0x0002	Ethercat communication error
	0x0004	Low battery
	0x0008	Battery fault
	0x0010	24V power supply fault
	0x0020	15V power supply fault
	0x0040	12V power supply fault
	0x0080	Right-hand power fault
	0x0100	Left-hand power fault
	0x0200	Left leg power fault
	0x0400	Right leg power fault
	0x0800	Motor pre-charge fault
	0x8000000000	Other hardware abnormality

FCC Caution: Any changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate this equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

- (1) this device may not cause harmful interference, and
- (2) this device must accept any interference received, including interference that may cause undesired operation.

This device and its antenna(s) must not be co-located or operating in conjunction with any other antenna or transmitter.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

The device has been evaluated to meet general RF exposure requirement. The device can be used in portable exposure condition without restriction

FCC ID: **2BOD5-LIMXOLI**