Oli EDU SDK Development Guide

生成时间：2026-08-27

Version	Date	Description of Changes	Compatible Software （If incompatible, update to the latest version via the Official Download Center）
V1.0	2026.02.27	Initial Release	V2.1.21 or later
V1.1	2026.07.17	1. Removed the MCP server 2. Added an option to disable camera driver auto-start	V2.2.11 or later
V1.2	2026.07.30	1. Updated camera data acquisition instructions	V2.2.11 or later

1 Large Model API Interface

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.

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.

1.2.1 Invoke via Curl

▼

复制代码

```
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
      }'
```

1.2.2 Invoke via Python

▼

复制代码

```
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.tex
t}")
```

1.2.3 Invoke via C++

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

- **Install Dependencies**

▼

复制代码

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

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

▶

复制代码

- **Compile and Run**

▼

复制代码

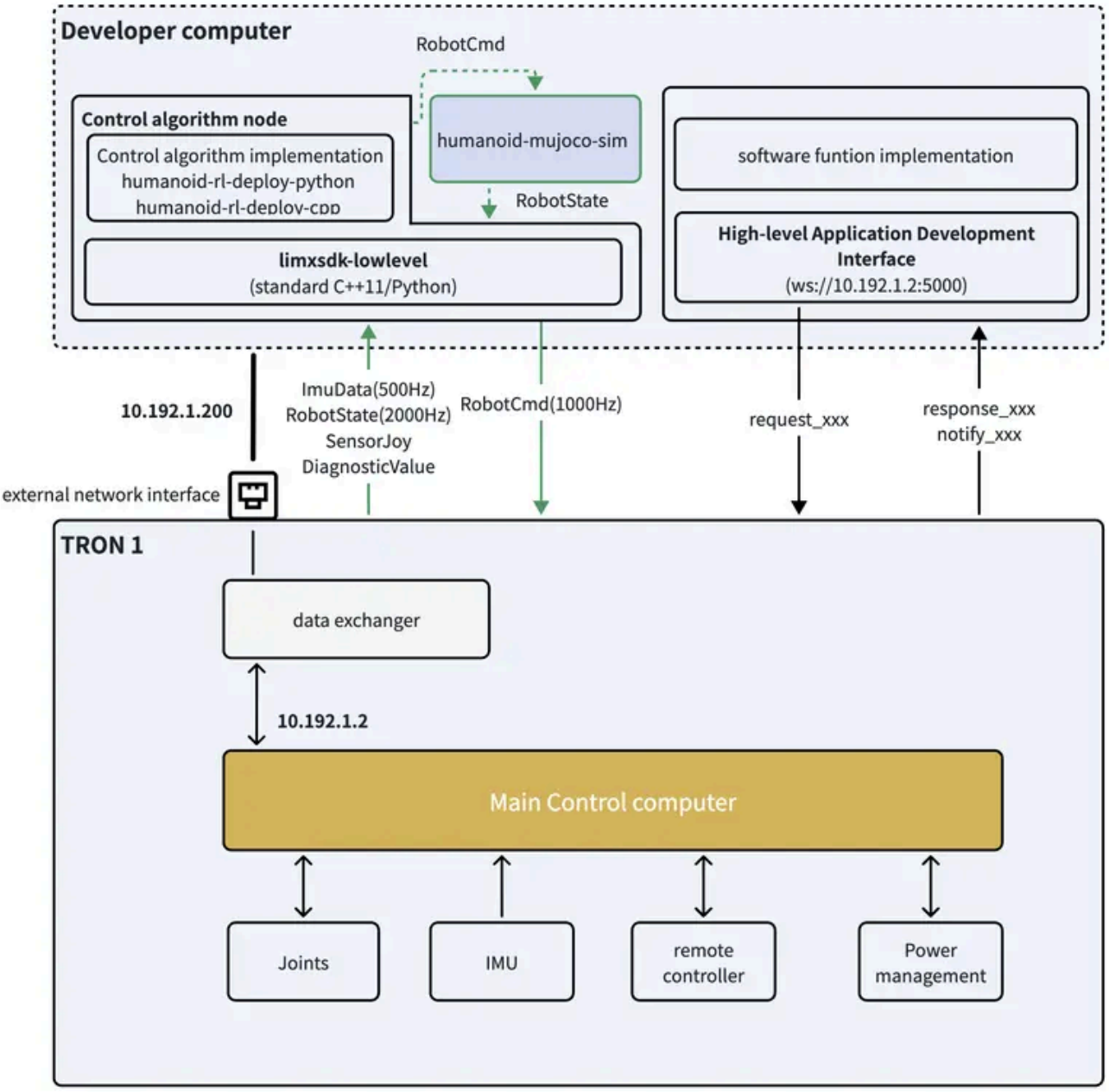
```
# Compile with C++11 support
g++ -std=c++11 -o llm_demo llm_demo.cpp -lcurl

# Execute
./llm_demo
```

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..

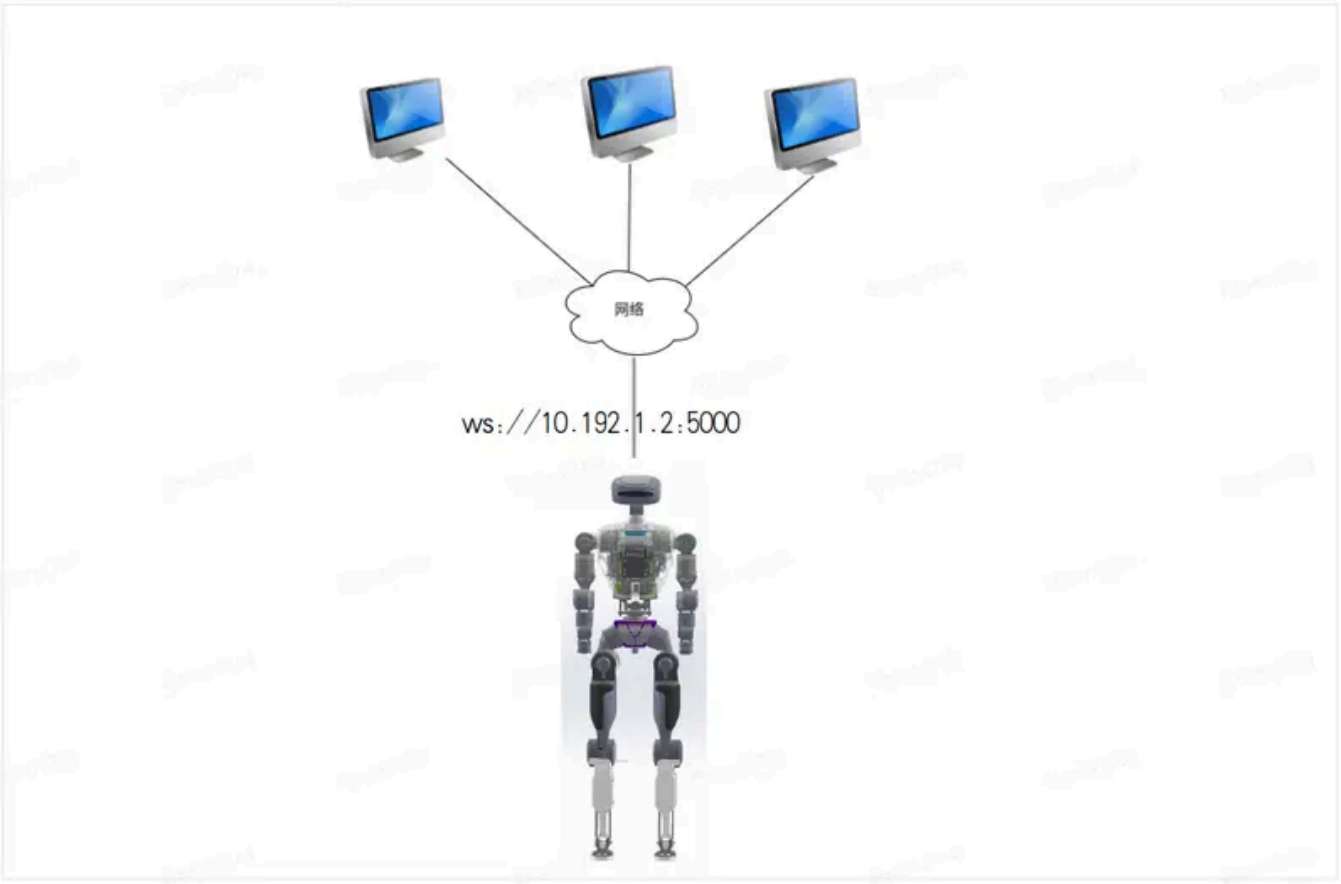


3 High-Level Application Protocol Interface

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:



3.1 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).

3.2 Communication Protocol Format

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

3.2.1 Request Data

Fields in the request data format	Description
`accid`	The robot's unique serial number identifies its identity.
`title`	The command name, prefixed with "`request_`".

Fields in the request data format	Description
`timestamp`	The timestamp when the command is sent, in milliseconds.
`guid`	A unique command identifier. For synchronous interfaces, the `"response_xxx"` message includes the same `guid`, allowing the client to confirm command completion by matching values
`data`	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:

▼

复制代码

```
{
  "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 generat
ed.
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Unique identifier for the request. Used t
o match the response for synchronous operations
  "data": {}  # Contains command-specific parameters
}
```

3.2.2 Response Data

Fields in the response data format	Description
`accid`	The robot's unique serial number identifies its identity.
`title`	The command name, prefixed with "`response_`".
`timestamp`	The timestamp when the command is issued, in milliseconds.
`guid`	Matches the `guid` value from the corresponding request command.
`data`	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 复制代码

```
{
  "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"
  }
}
```

3.2.3 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
`accid`	The robot's unique serial number identifies its identity.
`title`	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 复制代码

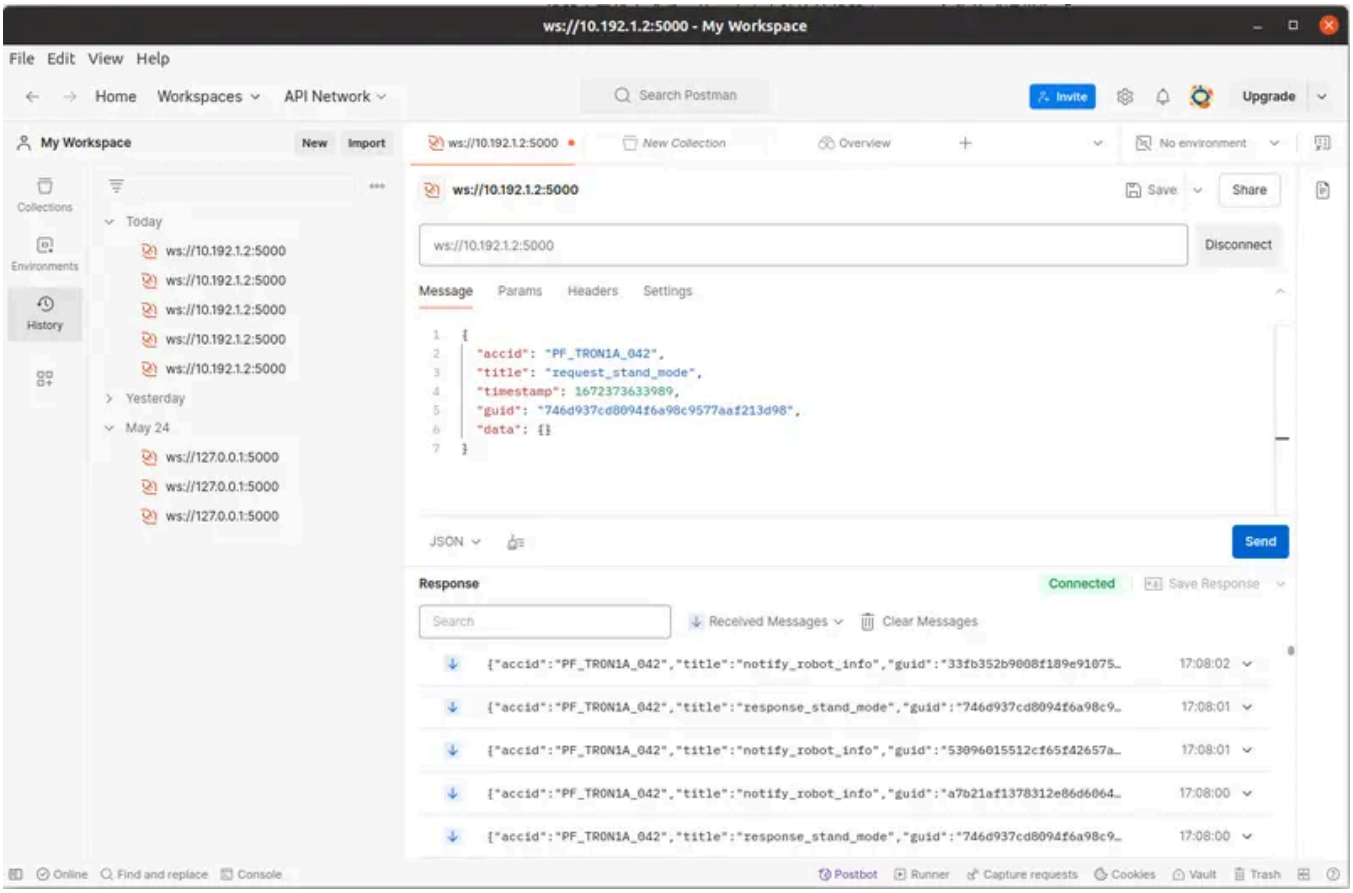
```
{
  "accid": "HU_D02_001",   # Robot’s unique serial number
  "title": "notify_xxx",  # notification command name, prefixed with "notify_"
  "timestamp": 1672373633989, # The time the message was sent, in milliseconds.
  "guid": "746d937cd8094f6a98c9577aaf213d98", # The unique identifier for this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

3.3 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:** 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」
4. Enter the Wi-Fi password: `12345678`
5. **Enter the WebSocket URL:** In the request URL field, input the robot’s WebSocket address, e.g.: "ws://10.192.1.2:5000"
6. **Input the Command Request:** In the **“Message”** field, enter the JSON-formatted command request.
7. **Send the Command:** Click **“Send”** to transmit the request to the robot.
8. **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 Basic Function Protocol Interfaces

3.4.1 Connect to Wi-Fi Hotspot

> This protocol is used to send a request to the robot router to connect to a specified Wi-Fi SSID and return the connection result. Supported robot versions: **v2.1.3 and later**.

3.4.1.1 Request: request_connect_wifi

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_connect_wifi",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "wifi_band": "0",  # WiFi band：0=5GHz，1=2.4GHz
      "wifi_ssid": "Limx-Guests",  # Target Wi-Fi SSID (network name) — case-sensitive and must match the actual hotspot name
      "wifi_password": "LimX2024",  # Target Wi-Fi password — password for a WPA2-PSK–encrypted network
```

```
      "router_admin_password": "12345678"  # Robot router administrator password
  }
}
```

3.4.1.2 Response: response_connect_wifi

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_connect_wifi",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success
                           # fail_no_wifi_band
                           # fail_no_wifi_ssid
                           # fail_no_wifi_password
                           # fail_no_router_admin_password
  }
}
```

3.4.1.3 Message Push: none

3.4.2 Query Wi-Fi Connection Status

> This protocol allows the client to query the Wi-Fi connection status of the robot router. The system returns the SSID, signal strength, and connection status for real-time monitoring.

3.4.2.1 Request: request_wifi_connection_status

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_wifi_connection_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "router_admin_password": "12345678"  # Robot router administrator password
  }
}
```

3.4.2.2 Response: response_wifi_connection_status

> The router returns the current Wi-Fi status for client-side parsing and real-time display.

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_wifi_connection_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "ssid": "Limx-Guests",
      "signal": -56,        # Signal strength (dBm)
      "result": "success"   # success
                            # fail_disconnected
  }
}
```

3.4.2.3 Message Push: none

3.4.3 Enter Ready State

> The robot slowly moves into a ready position.

3.4.3.1 Request: request_prepare

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

▼

json 复制代码

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

3.4.3.2 Response: response_prepare

▼

json 复制代码

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

3.4.3.3 Message Push: none

3.4.4 Control Robot Walking

3.4.4.1 Enter Walking Mode

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

3.4.4.1.1 Request: request_set_walk_mode

▼

json 复制代码

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

3.4.4.1.2 Response: response_set_walk_mode

▼

json 复制代码

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

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

3.4.4.1.3 Message Push: none

3.4.4.2 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.

3.4.4.2.1 Request: request_set_walk_vel

▼

json 复制代码

```
{
  "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]
  }
}
```

3.4.4.2.2 Response: response_set_walk_vel

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

▼

json 复制代码

```
{
  "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
  }
}
```

3.4.4.2.3 Message Push: none

3.4.5 Enter Damping State

> All motors stop active control and exhibit resistance when moved.

3.4.5.1 Request: request_damping

▼

json 复制代码

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

3.4.5.2 Response: response_damping

▼

json 复制代码

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

3.4.5.3 Message Push: none

3.4.6 Enter Zero-Torque State

> All motors stop active control and move freely without resistance.

3.4.6.1 Request: request_zero_torque

▼

json 复制代码

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

3.4.6.2 Response: response_zero_torque

▼

json 复制代码

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

3.4.6.3 Message Push: none

3.4.7 Enter Sitting Command

3.4.7.1 Request: request_from_stand_to_sit

▼

json 复制代码

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

3.4.7.2 Response: response_from_stand_to_sit

▼

json 复制代码

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

3.4.7.3 Message Push: none

3.4.8 Enter Standing Command

> **Notes：**
> Interface Function: Starts robot operation. After the robot is powered on, calling this interface transitions the robot into the standing state.
> Parameter mode: lying — robot is lying on the ground, hanging — robot is suspended, sit — robot is sitting
> Return: Returns after the robot has successfully stood up.

3.4.8.1 Request: request_standup

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_standup",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "mode": "lying" // "lying"/"sitting"：The robot is currently lying or sitting.  or
                      // "hanging"：The robot is currently hanging.
                      // If the "mode" field is not provided, the robot state defaults to
"sitting".
}
```

3.4.8.2 Response: response_standup

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_standup",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success" # fail_motor: Motor error
                          # fail_invalid_cmd: Invalid parameter
                          # fail_invalid_mode: Invalid robot state
                          # fail_timeout: Execution timeout error
  }
}
```

3.4.8.3 Message Push: none

3.4.9 Enter Lying Command

3.4.9.1 Request: request_lie_down

> **This interface is available when the robot is in the Walk state.**

▼

json 复制代码

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

3.4.9.2 Response: response_lie_down

▼

json 复制代码

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

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

3.4.9.3 Message Push: none

3.4.10 Zero-Point Calibration Command

3.4.10.1 Request: request_calibrate

▼

json 复制代码

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

3.4.10.2 Response: response_calibrate

▼

json 复制代码

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

3.4.10.3 Message Push: notify_calibrate

> Pushed after calibration is completed.

▼

python 复制代码

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

3.4.11 Robot Dance

3.4.11.1 Switch to Dance Mode

3.4.11.1.1 Request: request_enter_dance_mode

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_enter_dance_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    # 0：exit dance mode
    # 1：enter dance mode
    "mode": 0
  }
}
```

3.4.11.1.2 Response: response_enter_dance_mode

▼

json 复制代码

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

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

3.4.11.1.3 Message Push: none

3.4.11.2 Get Dance List

3.4.11.2.1 Request: request_get_dance_list

▼

json 复制代码

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

3.4.11.2.2 Response: response_get_dance_list

▼

json 复制代码

```
{
    "accid": "HU_D04_01_001",
    "title": "response_get_dance_list",
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "timestamp": 1672373633989,
    "data": {
        "result": "success",
        "code": 0,
        "dances": [
            {
                "id": "DAN-14",
                "index": 0,
                "name": "\u70ed\u70c8",
                "english_name": "One and Only Dance",
                "rc_mapping": "one_and_only_dance"
            },
            {
                "id": "DAN-08",
                "index": 1,
                "name": "\u4f4e\u4fd7\u5c0f\u8bf4",
```

```
                "english_name": "Pulp Fiction Dance",
                "rc_mapping": "pulp_fiction_dance"
            }
        ]
    }
}
```

3.4.11.3 Robot Dancing

3.4.11.3.1 Request: request_dance

> **Notes：**
>
> 1. **Precondition:** The robot must be in Action Library Mode.
> 2. **Supported on:** Main controller firmware v2.1.21 and later.

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_dance",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "name": "one_and_only_dance"  # rc_mapping Dance name
  }
}
```

3.4.11.3.2 Response: response_dance

▼

json 复制代码

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

3.4.11.3.3 Message Push: notify_dance

> Pushed when the dance completes or execution fails.

▼

json 复制代码

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

3.4.12 Robot Marching in Place

3.4.12.1 Enable March-in-Place

3.4.12.1.1 Request: request_start_walktoggle

▼

json 复制代码

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

3.4.12.1.2 Response: response_start_walktoggle

▼

json 复制代码

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

```
  "data": {
      "result": "success" # fail_motor
  }
}
```

3.4.12.1.3 Message Push: notify_dance

3.4.12.2 Stop March-in-Place

3.4.12.2.1 Request: request_stop_walktoggle

▼

json 复制代码

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

3.4.12.2.2 Response: response_stop_walktoggle

▼

json 复制代码

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

3.4.13 Robot Action Library

3.4.13.1 Action Interruption

3.4.13.1.1 Request: request_interrupt_action_joystick

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_interrupt_action_joystick",
  "timestamp": 1779355330784,
  "guid": "32cef03a-5563-4b21-9bbb-3e65a8c9ae9e",
  "data": {}
}
```

3.4.13.1.2 Response: response_interrupt_action_joystick

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "response_interrupt_action_joystick",
  "guid": "32cef03a-5563-4b21-9bbb-3e65a8c9ae9e",
  "timestamp": 1779355330784,
  "data": {
    "result": "success"
  }
}
```

3.4.13.2 Get Action Library Status

> Interface Description:
>
> 1. When the robot has entered the Action Library, and is in Action Library Mode, Atomic Action Execution, or Dance, the following field is returned: "action_library_mode": "action_library"
> 2. When the robot is executing an atomic action or a dance routine, the following field is returned: "action_library_state": "running"

3.4.13.2.1 Request: request_get_action_library_status

▼

json 复制代码

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

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

3.4.13.2.2 Response: response_get_action_library_status

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "get_action_library_status",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "action_library_mode": "action_library" //"remote_control"
      "action_library_state": "running"       //"idle"
      "result": "success" # fail_motor
  }
}
```

3.4.13.3 Execute Action Library

> Interface Description:
>
> 1. If not already in Menu mode, it will automatically enter Menu mode. After the action finishes, the system will remain in Menu mode.
> 2. Must be used together with the `action_library_state` field from `request_get_action_library_status`.

3.4.13.3.1 Request: request_action_sync_stay_menu

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_action_sync_stay_menu",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "name": "one_and_only_dance"
  }
}
```

3.4.13.3.2 Response: response_action_sync_stay_menu

▼

json 复制代码

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

3.4.13.4 Execute Action Library (Synchronous Interface)

> **Notes：**
> Interface Description:
>
> 1. If the robot is in a state where the Action Library cannot be executed, a failure response is returned within 100ms.
> 2. If the robot is in a state where the Action Library can be executed (walk/motion library), the response is returned after the action library execution is completed.
> 3. Synchronous interface, automatically transitions back to the Walk state. Completion is indicated when the system returns to the Walk state.

3.4.13.4.1 Request: request_action_sync

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "request_action_sync",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "name": "one_and_only_dance,this_way_please"
                                    # name:Multiple dances, multiple actions, or a mix of
dances and actions, separated by commas (,).
  }
}
```

3.4.13.4.2 Response: response_action_sync

▼

json 复制代码

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

3.4.13.5 Switching the Robot to Action Library Mode

3.4.13.5.1 Request: request_set_motion_engine

▼

json 复制代码

```
{
  "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
  }
}
```

3.4.13.5.2 Response: response_set_motion_engine

▼

json 复制代码

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

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

3.4.13.5.3 Message Push: none

3.4.13.6 Get Action Library List

3.4.13.6.1 Request: request_get_atomic_motion_list

▼

json 复制代码

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

3.4.13.6.2 Response: response_get_atomic_motion_list

▼

json 复制代码

```
{
    "accid": "HU_D04_01_001",
    "title": "response_get_atomic_motion_list",
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "timestamp": 287883835,
    "data": {
        "result": "success",
        "motion_list": [
            {
                "motion_index": 0,
                "motion_name_en": "stand"
            },
            {
                "motion_index": 1,
                "motion_name_en": "this_way_please"
            }
            ......
        ],
        "count": 2
```

```
    }
}
```

3.4.13.7 Executing Actions

Actions can be executed once the robot is in Action Library Mode.

3.4.13.7.1 Request: request_execute_atomic_motion

▼

json 复制代码

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

3.4.13.7.2 Response: response_execute_atomic_motion

▼

json 复制代码

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

3.4.13.7.3 Message Push: notify_execute_atomic_motion

> Pushed when an action is completed, or execution fails.

▼

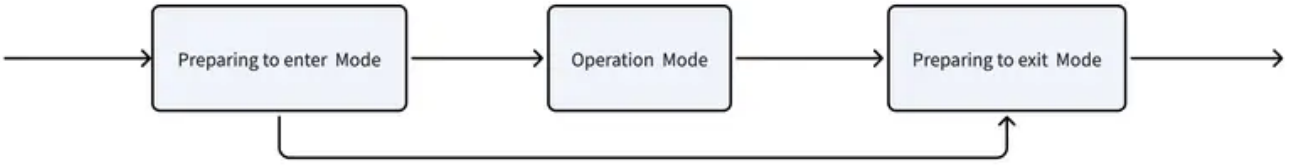
json 复制代码

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

3.4.14 Motion Control

3.4.14.1 Switching to Motion Operation Mode

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



3.4.14.1.1 Request: request_set_ub_manip_mode

▼

json 复制代码

```
{
  "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-effec
tor position 2: Ready to Exit Mode
  }
}
```

3.4.14.1.2 Response: response_set_ub_manip_mode

▼

json 复制代码

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

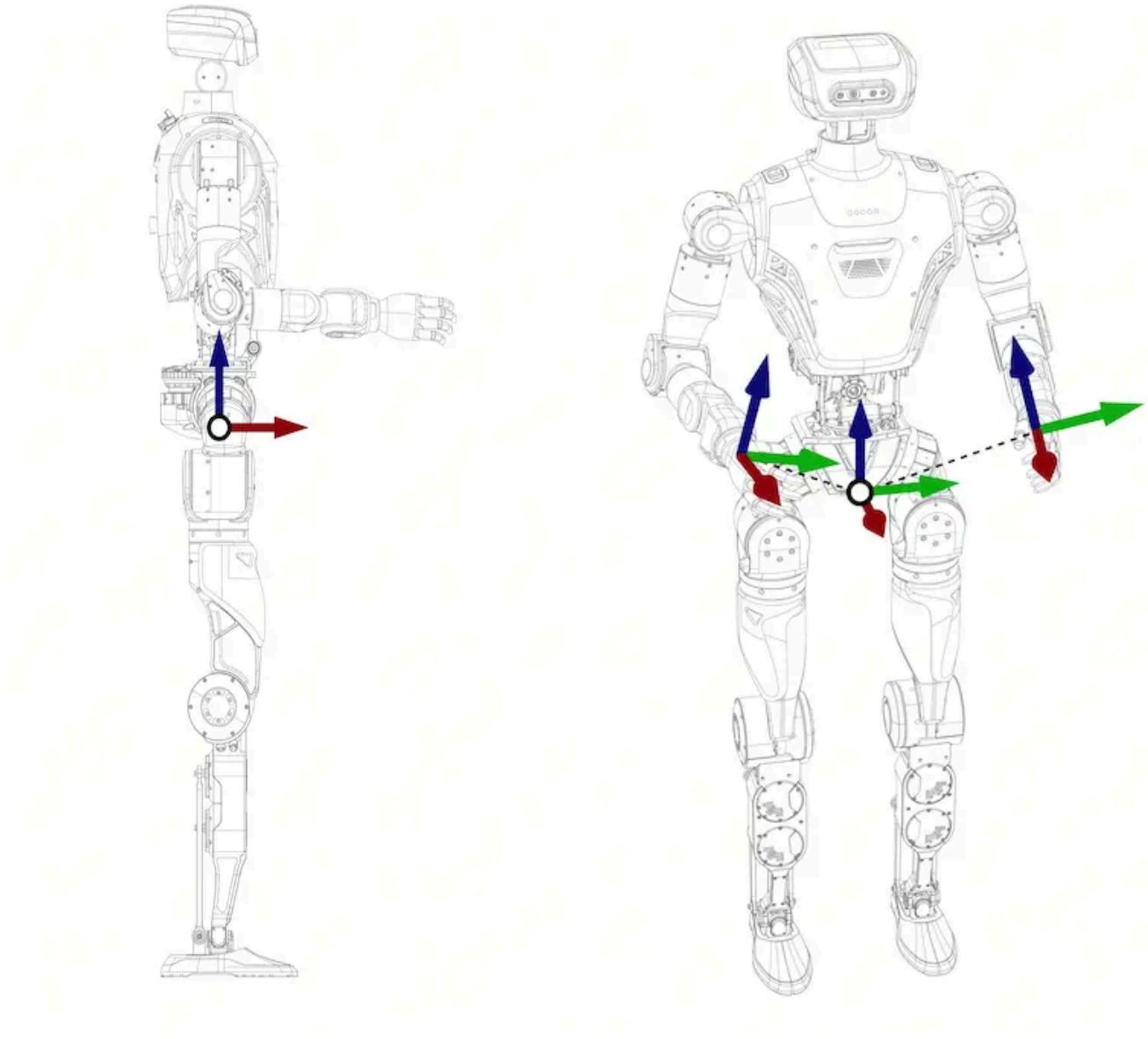
```
  "title": "response_set_ub_manip_mode",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": "success"  # success, fail_motor
  }
}
```

3.4.14.1.3 Message Push: none

3.4.14.2 Motion Operation Control

> Activated only via the `request_set_ub_manip_mode` interface.

- Reference Coordinate Frame Diagram:
  - Position: The origin of base_link (located at the center beneath the hip).
  - Axis Definitions:
    - Red (X-axis): Forward direction of the robot
    - Green (Y-axis): Positive direction to the left
    - Blue (Z-axis): Vertical upward direction



3.4.14.2.1 Request: request_set_ub_manip_ee_pose

▼

json 复制代码

```
{
  "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 r
obot’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 quaternio
n[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 quaterni
on[x,y,z,w]
      "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}
```

3.4.14.2.2 Response: response_set_ub_manip_ee_pose

▼

json 复制代码

```
{
  "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
  }
}
```

3.4.14.2.3 Message Push: none

3.4.14.3 Get Motion Operation Pose Information

3.4.14.3.1 Request: request_get_ub_manip_ee_pose

▼

json 复制代码

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

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

3.4.14.3.2 Response: response_get_ub_manip_ee_pose

▶

json 复制代码

3.4.15 In-Place Operation

3.4.15.1 Enter In-Place Operation Mode

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

3.4.15.1.1 Request: request_set_wb_manip_mode

▼

json 复制代码

```
{
  "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-effec
tor position 2: Ready to Exit Mode
  }
}
```

3.4.15.1.2 Response: response_set_wb_manip_mode

▼

json 复制代码

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

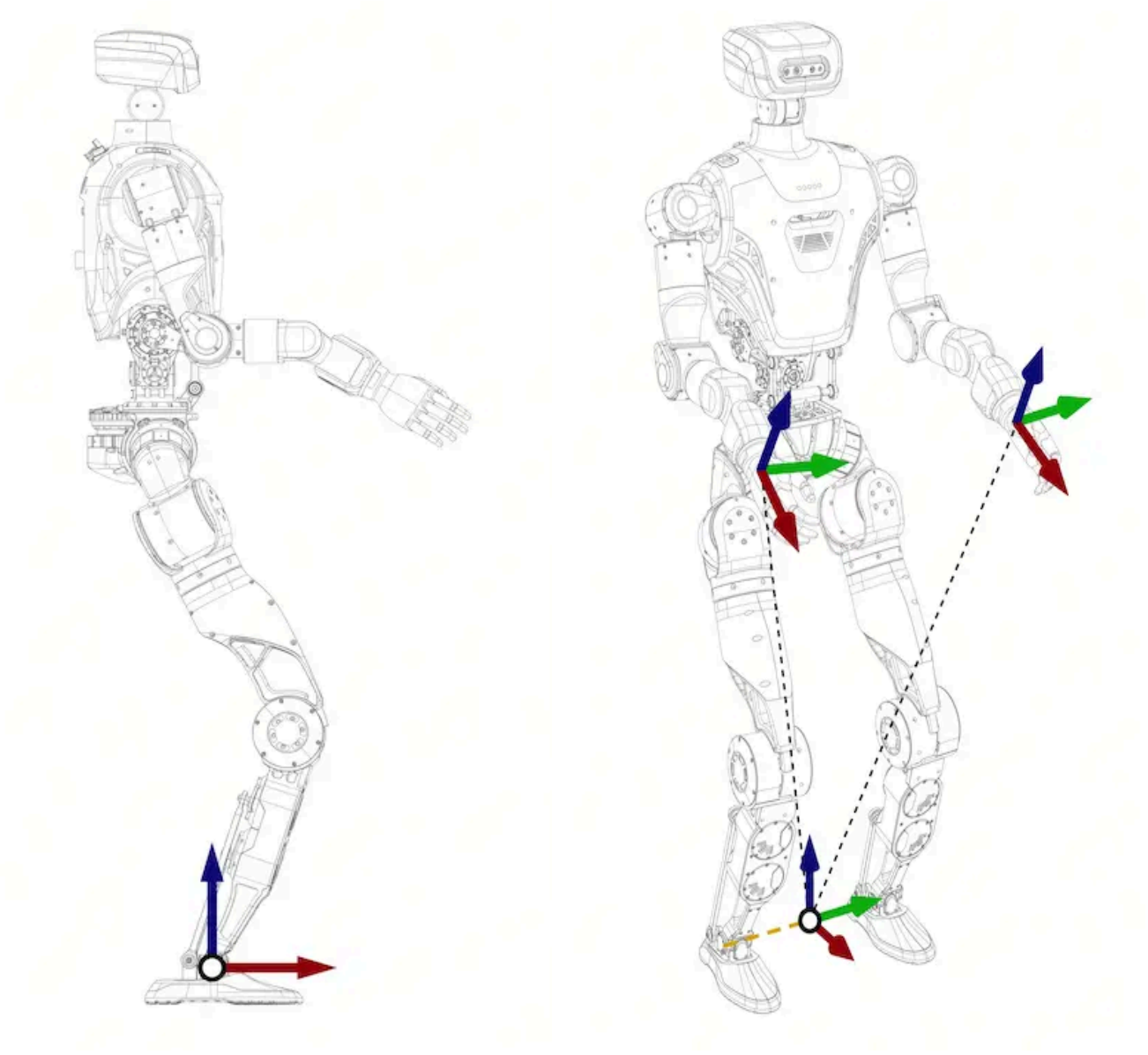
```
  }
}
```

3.4.15.1.3 Message Push: none

3.4.15.2 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.

- Reference Coordinate Frame Diagram:
  - Position: Midpoint of the line connecting the origins of left_ankle_roll_link and right_ankle_roll_link
  - Orientation: The yaw angle is the midpoint of the yaw orientations of left_ankle_roll_link and right_ankle_roll_link
  - Axis Definitions:
    - Red (X-axis): Determined by the forward-facing direction of the robot’s feet
    - Green (Y-axis): Determined according to the right-hand coordinate system rule
    - Blue (Z-axis): Vertical upward direction



3.4.15.2.1 Request: request_set_wb_manip_ee_pose

▼

json 复制代码

```
{
  "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 r
ight feet.
      # Axes: X-axis aligned with the robot’s forward direction, Y-axis pointing to the r
obot’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 quaternio
n[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 quaterni
on[x,y,z,w]
      "right_hand_quat": [0.0, 0.0, 0.0, 1.0]
  }
}
```

3.4.15.2.2 Response：response_set_wb_manip_ee_pose

▼

json 复制代码

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

3.4.15.2.3 Message Push: none

3.4.15.3 Get In-Place Operation Pose Information

3.4.15.3.1 Request: request_get_wb_manip_ee_pose

▼

json 复制代码

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

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

3.4.15.3.2 Response: response_get_wb_manip_ee_pose

▼

json 复制代码

```
{
  "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 rig
ht feet.
    # Axes: X-axis aligned with the robot’s forward direction, Y-axis pointing to the rob
ot’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
  }
}
```

3.4.15.3.3 Message Push: none

3.4.16 Dual-Arm Coordinated Move Control

3.4.16.1 Switch to Move Control Mode

3.4.16.1.1 Request: request_set_move_mode

▼

json 复制代码

```
{
  "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
  }
}
```

3.4.16.1.2 Response: response_set_move_mode

▼

json 复制代码

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

3.4.16.1.3 Message Push: none

3.4.16.2 MoveJ Control Command

3.4.16.2.1 Request: request_moveJ

▶

python 复制代码

3.4.16.2.2 Response: response_moveJ

▼

python 复制代码

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

3.4.16.2.3 Message Push: notify_moveJ

> Execution completion or failure will trigger an active message push.

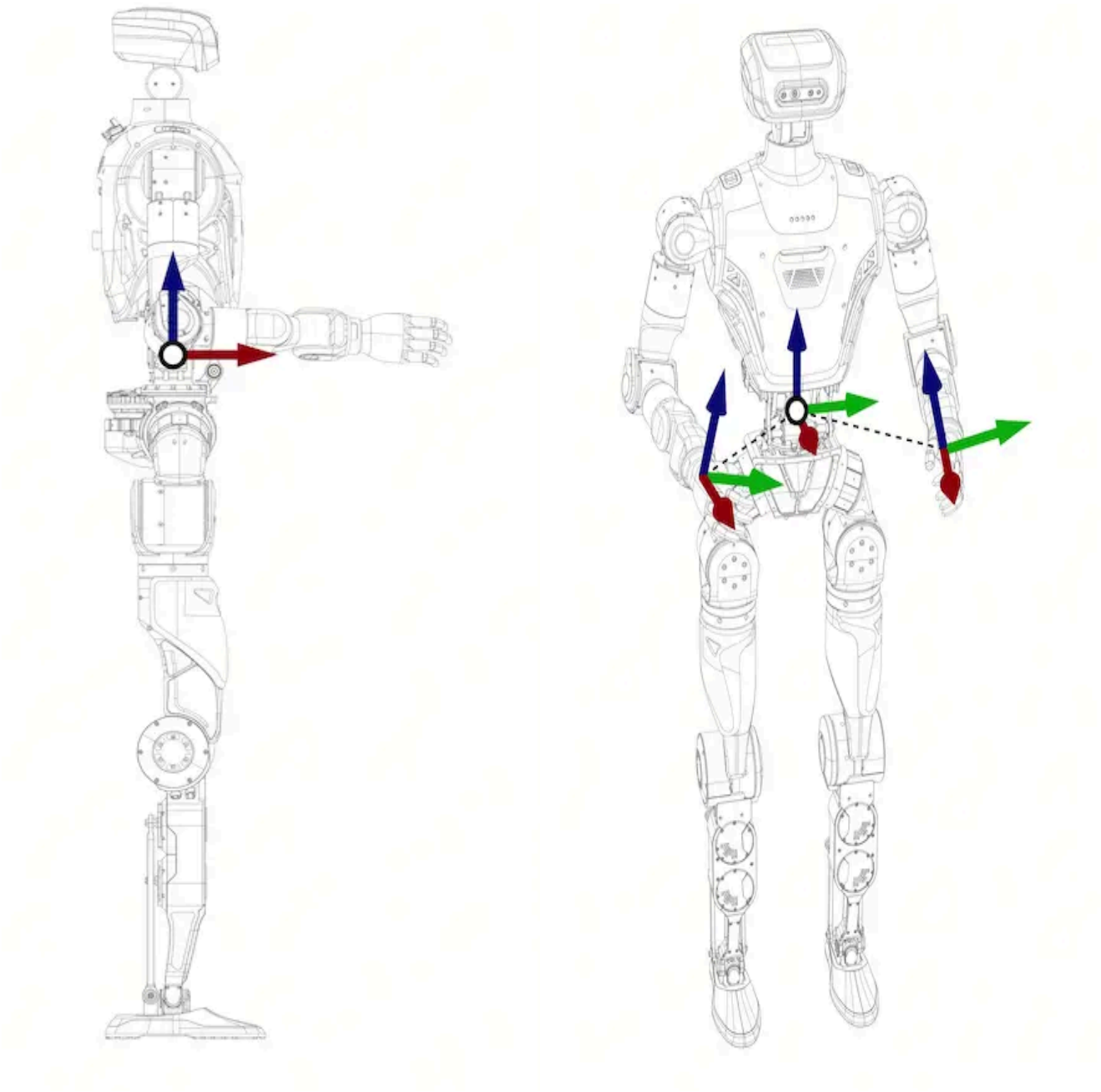
▼

python 复制代码

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

3.4.16.3 MoveP Control Command

- Reference Coordinate Frame Diagram:
  - Position: Origin of the waist_pitch_link.
  - Axis Definitions:
    - Red (X-axis): Forward direction of the robot
    - Green (Y-axis): Positive direction to the left
    - Blue (Z-axis): Vertical upward direction



3.4.16.3.1 Request: request_moveP

▼

python 复制代码

```
{
  "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-effecto
r, 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-effec
tor, represented as a quaternion (x, y, z, w)

      # In In-place MoveP Mode only, providing the following data will control torso post
ure
      "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
  }
}
```

3.4.16.3.2 Response: response_moveP

▼

python 复制代码

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

3.4.16.3.3 Message Push: notify_moveP

> Execution completion or failure will trigger an active message push.

▼

python 复制代码

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

3.4.16.4 Get Dual-Arm End-Effector Pose

3.4.16.4.1 Request: request_get_move_pose

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

▼

json 复制代码

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

3.4.16.4.2 Response: response_get_move_pose

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

▼

复制代码

```
{
  "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 m
eters, 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-effecto
r, as a quaternion x, y, z, w
      "result": "success"  # fail_not_data
  }
}
```

3.4.17 Dual-Arm Coordinated Servo Control

3.4.17.1 Switch to Servo Control Mode

3.4.17.1.1 Request: request_set_servo_mode

▼

json 复制代码

```
{
  "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
  }
}
```

3.4.17.1.2 Response: response_set_servo_mode

▼

json 复制代码

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

```
  }
}
```

3.4.17.1.3 Message Push: none

3.4.17.2 ServoJ Control Command

3.4.17.2.1 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 复制代码

3.4.17.2.2 Response: none

3.4.17.2.3 Message Push: notify_servoJ

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

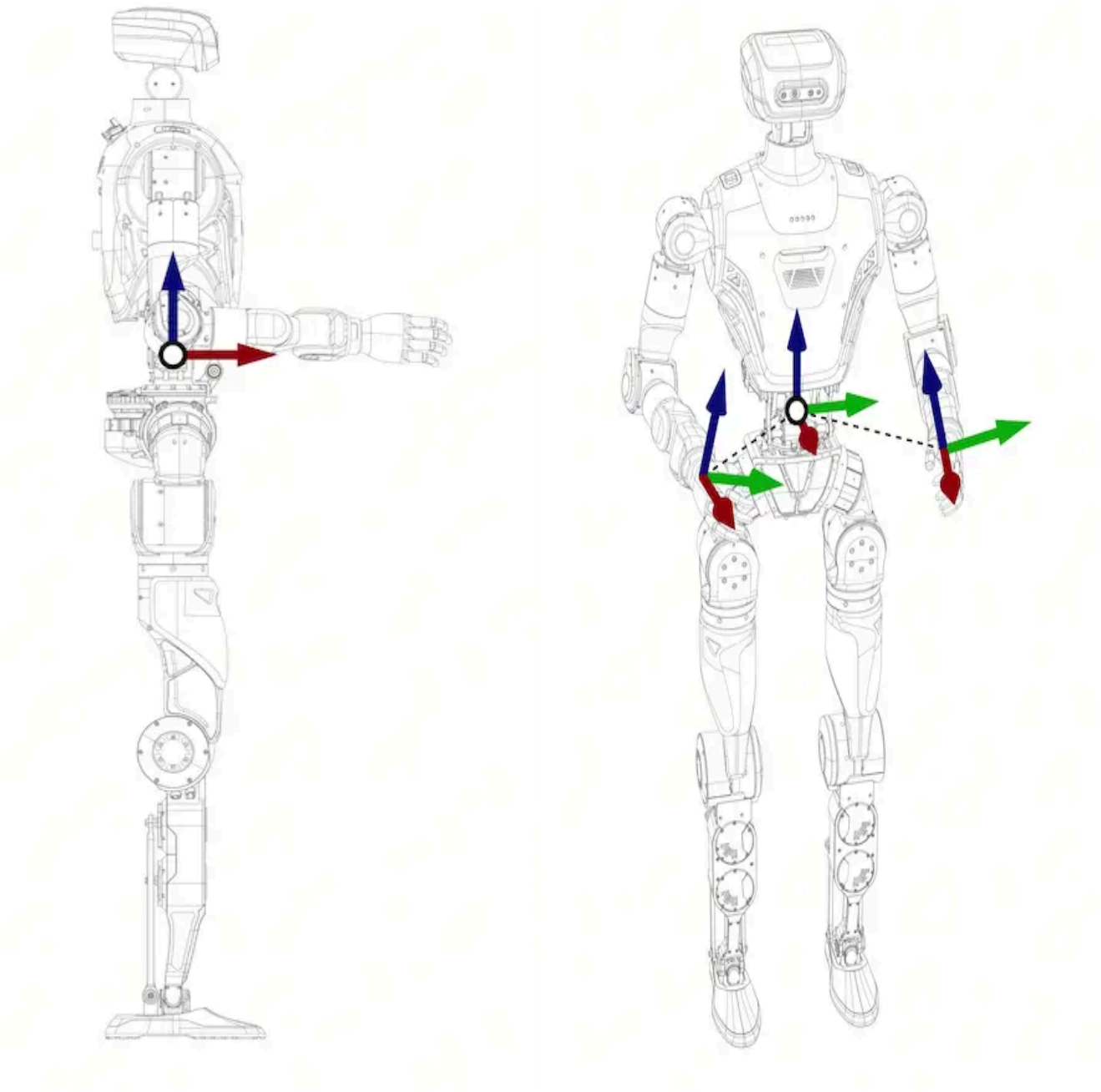
▼

json 复制代码

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

3.4.17.3 ServoP Control Command

- Reference Coordinate Frame Diagram:
  - Position: Origin of the waist_pitch_link.
  - Axis Definitions:
    - Red (X-axis): Forward direction of the robot
    - Green (Y-axis): Positive direction to the left
    - Blue (Z-axis): Vertical upward direction



3.4.17.3.1 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 复制代码

```
{
  "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  # picth joint target position (radians)
  }
}
```

3.4.17.3.2 Response: none

3.4.17.3.3 Message Push: notify_servoP

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

▼

json 复制代码

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

3.4.17.4 Get Dual-Arm End Pose

3.4.17.4.1 Request: request_get_servo_pose

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

▼

json 复制代码

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

3.4.17.4.2 Response: response_get_servo_pose

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

▼

复制代码

```
{
  "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 m
eters, 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-effecto
r, as a quaternion x, y, z, w
      "result": "success"  # fail_not_data
  }
}
```

3.4.17.4.3 Message Push: none

3.4.18 Robot Joints State

3.4.18.1 Request: request_get_joint_state

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

▼

json 复制代码

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

3.4.18.2 Response: response_get_joint_state

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

▼

复制代码

```
{
  "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
  }
}
```

3.4.18.3 Message Push: none

3.4.19 Get IMU Data

3.4.19.1 Request: request_get_imu_data

▼

json 复制代码

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

3.4.19.2 Response: response_get_imu_data

▼

json 复制代码

```
{
  "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]
  }
}
```

3.4.19.3 Message Push: none

3.5 Dexterous Hand and Gripper Protocol Interface

3.5.1 LimX 2-Finger Gripper

3.5.1.1 Gripper Control Command

3.5.1.1.1 Request: request_set_limx_2fclaw_cmd

> This protocol controls the gripper's grasping actions.

▼

json 复制代码

```
{
  "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 contr
olled
      "left_opening": 50,  # Gripper Opening，0-100，dimensionless（0 represents fully clo
sed，100 represents fully open）
      "left_speed": 50,    # Gripper Speed，0~100 dimensionless（higher values indicate fa
ster motion）
      "left_force": 50,   # Gripper Force，0~100 dimensionless（higher values represent st
ronger force）

      # If the following data are provided simultaneously, the right gripper will be cont
rolled
      "right_opening": 50,  # Gripper Opening，0-100，dimensionless（0 represents fully cl
ose，100 represents fully open大）
      "right_speed": 50,    # Gripper Spee，0~100 dimensionless（higher values indicate fa
ster motion）
      "right_force": 50,   #Gripper Forc，0~100 dimensionless（higher values represent str
onger force）
  }
}
```

3.5.1.1.2 Response: response_set_limx_2fclaw_cmd

▼

json 复制代码

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

3.5.1.1.3 Message Push: none

3.5.1.2 Get Gripper Status Information

3.5.1.2.1 Request: request_get_limx_2fclaw_state

▼

json 复制代码

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

3.5.1.2.2 Response: response_get_limx_2fclaw_state

> Return Gripper Status Information.

▼

json 复制代码

```
{
  "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 clo
```

```
se，100 represents fully open）
      "left_speed": 50,    # Gripper Speed，0~100 dimensionless（higher values indicate fa
ster motion）
      "left_force": 50,   # Gripper Force，0~100 dimensionless（higher values represent st
ronger force）

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

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

3.5.1.2.3 Message Push: none

3.5.2 Limx 3-Finger Gripper

3.5.2.1 Gripper Control Command

3.5.2.1.1 Request: request_set_limx_3fclaw_cmd

> This protocol controls the gripper’s grasping actions.

▼

json 复制代码

```
{
  "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 contr
olled
      "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 close
d，100 = fully open，default: 42）
      "left_right": 0,   # Right Finger Bend Angle，0-100，dimensionless（0 = fully close
d，100 = fully open，default: 42）
      "left_lr_rot": 0,  # Finger Rotation Angle，0~180 degrees，（0 = fingers adjacent，18
0 = fingers opposed, default: 180）

      # If the following data are provided simultaneously, the right gripper will be cont
```

```
rolled
      "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 close
d，100 = fully open，default: 42）
      "right_right": 0,  # Right Finger Bend Angle，0-100，dimensionless（0 = fully close
d，100 = fully open，default: 42）
      "right_lr_rot": 1  # Finger Rotation Angl，0~180 degrees，（0 = fingers adjacent，180
= fingers opposed，default: 180）
  }
}
```

3.5.2.1.2 Response: response_set_limx_3fclaw_cmd

▼

json 复制代码

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

3.5.2.1.3 Message Push: none

3.5.2.2 Get Gripper Status Information

3.5.2.2.1 Request: request_get_limx_3fclaw_state

▼

json 复制代码

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

3.5.2.2.2 Response: esponse_get_limx_3fclaw_state

> Return Gripper Status Information.

▼

json 复制代码

```
{
  "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
  }
}
```

3.5.2.2.3 Message Push: none

3.5.3 Inspire 2-Finger Gripper

3.5.3.1 Gripper Control Command

3.5.3.1.1 Request: request_set_claw_cmd

> This protocol controls the gripper’s grasping actions.

▼

json 复制代码

```
{
  "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 contr
olled
      "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 Co
ntrol）

      # 如If the following data are provided simultaneously, the right gripper will be con
trolled
      "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）
  }
}
```

3.5.3.1.2 Response: response_set_claw_cmd

▼

json 复制代码

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

3.5.3.1.3 Message Push: none

3.5.3.2 Get Gripper Status Information

3.5.3.2.1 Request: request_get_claw_state

▼

json 复制代码

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

3.5.3.2.2 Response: response_get_claw_state

> Return Gripper Status Information.

▼

json 复制代码

```
{
  "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
  }
}
```

3.5.3.2.3 Message Push: none

3.5.4 BrainCo's Revo 1 Dexterous Hand

3.5.4.1 Dexterous Hand Control Command

3.5.4.1.1 Request: request_set_brainco_hand_cmd

▼

json 复制代码

```
{
  "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
  }
}
```

3.5.4.1.2 Response: response_set_brainco_hand_cmd

▼

json 复制代码

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

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

3.5.4.1.3 Message Push: none

3.5.4.2 Get Dexterous Hand Status

3.5.4.2.1 Request: request_get_brainco_hand_state

▼

json 复制代码

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

3.5.4.2.2 Response: response_get_brainco_hand_state

▼

json 复制代码

```
{
  "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
  }
}
```

3.5.4.2.3 Message Push: none

3.5.5 BrainCo's Revo 2 Dexterous Hand

3.5.5.1 Dexterous Hand Control Command

3.5.5.1.1 Request: request_set_brainco2_hand_cmd

▶

json 复制代码

3.5.5.1.2 Response: response_set_brainco2_hand_cmd

▼

json 复制代码

```
{
  "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 co
mmand
  }
}
```

3.5.5.1.3 Message Push: none

3.5.5.2 Get Dexterous Hand Status

3.5.5.2.1 Request: request_get_brainco2_hand_state

▼

json 复制代码

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

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

3.5.5.2.2 Response: response_get_brainco2_hand_state

▼

json 复制代码

```
{
  "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]
  }
}
```

3.5.5.2.3 Message Push: none

3.6 Global Message Protocol Interface

3.6.1 Robot Status Information

> This protocol periodically reports the robot's status information.

Status Infomation	Description
`accid`	The robot's unique serial number.

Status Infomation	Description
`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 复制代码

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

3.6.1.1 Battery Data

▶

json 复制代码

Field	Description
bmsconn	Battery connection status: **OFF** = not connected, **ON** = connected
bat_chg	Battery charger status: **OFF** = not connected, **ON** = connected
bat_off	Battery pre-shutdown status: OFF = power will be cut off after 1 s, ON = normal
bat_prt	Battery fault code: 0 = normal, non-zero = fault
bat_vol	Real-time battery voltage (unit: mV)
bat_cur	Real-time battery current (unit: mA)
battery	Battery level percentage (0–100)

Field	Description
bat_temp0	Battery temperature sensor 0 (range: 0–100, unit: ×10 °C)
bat_temp2	Battery temperature sensor 2 (range: 0–100, unit: ×10 °C)
bat_temp4	Battery temperature sensor 4 (range: 0–100, unit: ×10 °C)

3.6.1.2 System Information

▶

json 复制代码

Field	Description
version	Main controller firmware version
ecm_version	Master station version
pms_version	Power distribution board version
motor_version	Motor firmware version
sn	Robot serial number
robot_status	Current robot status
ability_running	Currently active controller

3.6.1.3 Motor Status Information

▼

json 复制代码

```
{
  "accid": "HU_D04_01_001",
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "result": [
          {
                "level": 1,
                "name": "ethercatCommunicationExp",
                "message": "WARN",
                "hardware_id": "ethercat",
```

```
                "values": [
                    {
                        "key": "ethercatCommunicationExp",
                        "value": "motor 17 MOTOR_LOST triggered HALF_STAND"
                    },
                    {
                        "key": "ethercatResetNormal",
                        "value": "ok!"
                    }
                ]
            }
        ]
    }
}
```


Field	Description
level	Fault severity level: 0 = OK, 1 = Warning, 2 = Error
name	Fault type
message	Severity description string
hardware_id	Hardware ID
values	Collection of all fault entries for the specified hardware

3.6.2 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 复制代码

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

3.7 Protocol Interface Usage Example

3.7.1 Python Example

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

▼

bash 复制代码

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

- Execute the Script

▼

bash 复制代码

```
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 复制代码

3.7.2 Linux C++ Example

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

▼

bash 复制代码

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

- **Compile the Code**

▼

bash 复制代码

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

- **Execute the Program**

▼

bash 复制代码

```
./humanoid
```

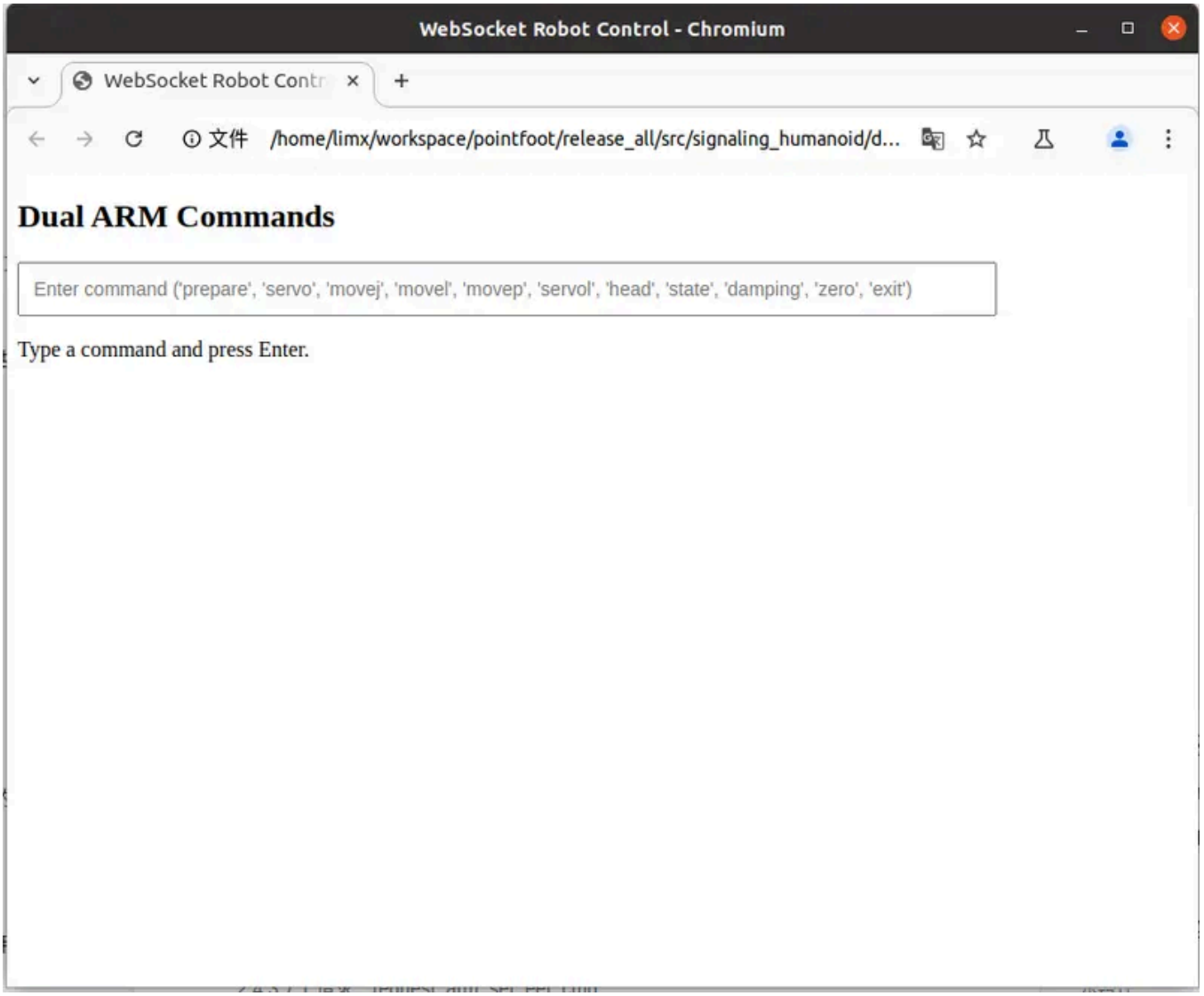
- **`humanoid.cpp` Implementation**

▶

cpp 复制代码

3.7.3 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**

▶

html 复制代码

4 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 + ○`.

4.1 C++ Motion Control Development Interface

4.1.1 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：

▼

cpp 复制代码

```cpp
#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;
}
```

4.1.2 init Interface

Function Name	init
Function Prototype	bool init(const std::string& robot_ip_address = "127.0.0.1");
Description	Initializes the communication and runtime environment for the motion control algorithm, typically called before other interfaces in the main function.
Parameters	robot_ip_address: The IP address of the robot. Use "127.0.0.1" for simulation and "10.192.1.2" for real robots.
Return Value	true if initialization succeeds; otherwise false.
Remarks	None

Code Example：

▶

cpp 复制代码

4.1.3 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：

▶

cpp 复制代码

4.1.4 subscribeImuData Interface

Function Name	subscribeImuData
Function Prototype	void subscribeImuData(std::function<void(const ImuDataConstPtr&)> cb);
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:

▼

cpp 复制代码

```cpp
/**
 * @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).
};

// Smart Pointer Type Alias
typedef std::shared_ptr<ImuData> ImuDataPtr;
typedef std::shared_ptr<ImuData const> ImuDataConstPtr;
```

Code Example：

▶

cpp 复制代码

4.1.5 subscribeRobotState Interface

Function Name	subscribeRobotState
Function Prototype	void subscribeRobotState(std::function<void(const RobotStateConstPtr&)> cb);
Description	Subscribes to receive updates on robots' status. **Notes:** This interface reports joint states based on the robot’s equivalent serial URDF. This robot adopts a series-parallel hybrid configuration design, with parallel drive structures used at some joints. Unlike traditional serial joints (where a joint is directly driven by a single actuator), parallel joints have the following characteristics: - Multiple actuators collaboratively driving a single joint DOF - Higher load capacity, stiffness, and dynamic performance - More compact mechanical design with higher torque density - Improved fault tolerance through actuator redundancy While parallel actuation offers significant mechanical advantages, it also introduces challenges for control and application development: - More complex kinematic and dynamic modeling - Incompatibility with conventional serial robot control algorithms - Higher learning curve due to parallel mechanism theory - Limited compatibility with existing robotics ecosystems (e.g., ROS, MoveIt!) To eliminate these complexities, the robot controller transparently maps parallel joints to an equivalent serial joint model, allowing upper-layer applications to interact with the robot as if it were a standard serial manipulator. **State Feedback:** Parallel actuator states (position, velocity, current/torque) → Equivalent serial joint computation → Equivalent serial joint states (position, velocity, torque)
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:

▼

cpp 复制代码

```cpp
/**
 * @struct RobotState
 *
 * @brief Represents a data structure for robot state based on sensor feedback.
 *
 * 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;
```

Code Example：

▶

cpp 复制代码

4.1.6 publishRobotCmd Interface

Function Name	publishRobotCmd
Function Prototype	bool publishRobotCmd(const RobotCmd& cmd);

Function Name	publishRobotCmd
Description	Publishes a command to control the robot's actions. **Notes:** This interface accepts joint commands for the robot's equivalent serial URDF model. The robot adopts a serial–parallel hybrid architecture, with parallel drive structures used at some joints. Unlike conventional serial joints, where each joint is driven by a single actuator, parallel joints have the following characteristics: - Multiple actuators collaboratively drive a single joint DOF. - Higher load capacity, stiffness, and dynamic performance. - More compact mechanical design with higher joint torque output. - Improved fault tolerance through actuator redundancy. Although parallel joints offer significant mechanical advantages, they also introduce challenges for control and application development: - More complex kinematic and dynamic models. - Conventional serial robot control algorithms and toolchains cannot be applied directly. - Higher learning curve due to the complexity of parallel mechanisms. - Limited compatibility with existing robotics ecosystems (e.g., ROS and MoveIt!). To address these challenges, the robot controller performs a transparent conversion from parallel joints to an equivalent serial joint model, completely hiding the complexity of the parallel mechanisms from upper-layer applications. **Command Flow:** Equivalent serial joint commands (position, velocity, torque) → Parallel actuator command computation → Actuator command execution **Note:** Since Kp and Kd gains cannot be efficiently converted to parallel actuator commands, users requiring force control are recommended to command joint torque (tau) directly instead of achieving force control indirectly through PD control. This is particularly relevant for reinforcement learning policies that output position or velocity actions, where the controller converts PD commands into torque commands internally.
Parameters	cmd: a RobotCmd object specifying the desired robot command
Return Value	None
Remarks	- RobotCmd data structure prototype defined as follows:

▶

cpp 复制代码

Code Example：

▶

cpp 复制代码

4.1.7 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 SensorJoyConstPtr, a shared pointer to a constant SensorJoy structure.
Return Value	None
Remarks	- SensorJoy data structure prototype defined as follows:

▼

cpp 复制代码

```
/**
 * @struct SensorJoy
 *
 * @brief The structure of the robot remote controller data
 *
 * This structure contains timestamp information related to the controller, as well as jo
ystick and button values.
 */
struct SensorJoy {
    uint64_t stamp;                    // Timestamp corresponding to the sensor input ti
me, in nanoseconds
    std::vector<float> axes;           //  Values representing controller joystick manip
ulation
    std::vector<int32_t> buttons;      // Values representing controller button operatio
n states
};
```

```
// SensorJoy Smart Pointer Type Alias
typedef std::shared_ptr<SensorJoy> SensorJoyPtr;
typedef std::shared_ptr<const SensorJoy> SensorJoyConstPtr;
```

Code Example：

▶

cpp 复制代码

4.1.8 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 DiagnosticValue 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 DiagnosticValueConstPtr, a shared pointer to a constant DiagnosticValue structure containing fields such as timestamp, level, name, code, and message.
Return Value	None
Remarks	- DiagnosticValue data structure prototype defined as follows:

▼

cpp 复制代码

```
/**
 * @struct DiagnosticValue
 *
 * @brief Structure representing diagnostic values
 *
 * This structure contains information about the diagnostic level, name, code, and messag
e.
 */
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> DiagnosticValuePtr;
typedef std::shared_ptr<DiagnosticValue const> DiagnosticValueConstPtr;
```

Code Example：

▶

cpp 复制代码

4.1.9 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 High-Level Application Protocol Interface.
Parameters	JSON_payload: A JSON string conforming to the High-Level Application Protocol specification. Example: {"accid": "xxx", "title": "request_xxx", "timestamp": xxx, "guid": "xxx", "data": {}}
Return Value	None
Remarks	Valid only in high-level development mode.

Code Example：

▶

cpp 复制代码

4.1.10 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 High-Level Application Protocol Interface. The callback is triggered in the following cases: 1. When the robot returns a response to a previously sent JSON command (publishJsonMessage). 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：

▶

cpp 复制代码

4.1.11 Reference Example

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

4.2 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.

4.2.1 Install Motion Control Development Library

- Linux x86_64 Environment

▼

bash 复制代码

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

- Linux aarch64 Environment

▼

bash 复制代码

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

- Windows Environment

▼

bash 复制代码

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

4.2.2 init Interface

Function Name	**init**
Function Prototype	def **init**(self, robot_type: robot.RobotType)
Description	Specifies the robot type during initialization and creates a local robot instance of the corresponding type.
Parameters	robot_type: an enumeration value specifying the robot type, where RobotType.Humanoid represents a bipedal humanoid robot.
Return Value	None
Remarks	None

Code Example：

▼

python 复制代码

```
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)
```

4.2.3 init Interface

Function Name	init
Function Prototype	def init(self, robot_ip: str = "127.0.0.1")
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 复制代码

```
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()
```

4.2.4 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：

▼

python 复制代码

```
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()
```

4.2.5 subscribeImuData Interface

Function Name	subscribeImuData
Function Prototype	def subscribeImuData(self, callback: Callable[[datatypes.ImuData], Any])

Function Name	subscribeImuData
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 True Failure: return False
Remarks	- datatypes.ImuData structure prototype defined as follows:

▼

python 复制代码

```
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 re
corded or generated, in nanoseconds
        self.acc = [0. for x in range(0, 3)]  # Stores IMU (Inertial Measurement Unit) ac
celerometer 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
```

Code Example：

▶

python 复制代码

4.2.6 subscribeRobotState Interface

Function Name	subscribeRobotState
Function Prototype	def subscribeRobotState(self, callback: Callable[[datatypes.RobotState], Any])
Description	Subscribes to receive updates on robots' status

Function Name	subscribeRobotState
Parameters	Callback: callback function, invoked upon receiving a state update. Its parameter points to a datatypes.RobotState object. - datatypes.RobotState 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. **Notes:** This interface reports joint states based on the robot's equivalent serial URDF. This robot adopts a series-parallel hybrid configuration design, with parallel drive structures used at some joints. Unlike traditional serial joints (where a joint is directly driven by a single actuator), parallel joints have the following characteristics: - Multiple actuators collaboratively driving a single joint DOF - Higher load capacity, stiffness, and dynamic performance - More compact mechanical design with higher torque density - Improved fault tolerance through actuator redundancy While parallel actuation offers significant mechanical advantages, it also introduces challenges for control and application development: - More complex kinematic and dynamic modeling - Incompatibility with conventional serial robot control algorithms - Higher learning curve due to parallel mechanism theory - Limited compatibility with existing robotics ecosystems (e.g., ROS, MoveIt!) To eliminate these complexities, the robot controller transparently maps parallel joints to an equivalent serial joint model, allowing upper-layer applications to interact with the robot as if it were a standard serial manipulator. **State Feedback:** Parallel actuator states (position, velocity, current/torque) → Equivalent serial joint computation → Equivalent serial joint states (position, velocity, torque)
Return Value	Success: return True Failure: return False
Remarks	- datatypes.RobotState structure prototype defined as follows:

▼

python 复制代码

```
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 re
corded 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
```

Code Example：

▶

python 复制代码

4.2.7 publishRobotCmd Interface

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

Function Name	publishRobotCmd
Parameters	cmd: A datatypes.RobotCmd 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. **Notes:** This interface accepts joint commands for the robot's equivalent serial URDF model. The robot adopts a serial–parallel hybrid architecture, with parallel drive structures used at some joints. Unlike conventional serial joints, where each joint is driven by a single actuator, parallel joints have the following characteristics: - Multiple actuators collaboratively drive a single joint DOF. - Higher load capacity, stiffness, and dynamic performance. - More compact mechanical design with higher joint torque output. - Improved fault tolerance through actuator redundancy. Although parallel joints offer significant mechanical advantages, they also introduce challenges for control and application development: - More complex kinematic and dynamic models. - Conventional serial robot control algorithms and toolchains cannot be applied directly. - Higher learning curve due to the complexity of parallel mechanisms. - Limited compatibility with existing robotics ecosystems (e.g., ROS and MoveIt!). To address these challenges, the robot controller performs a transparent conversion from parallel joints to an equivalent serial joint model, completely hiding the complexity of the parallel mechanisms from upper-layer applications. **Command Flow:** Equivalent serial joint commands (position, velocity, torque) → Parallel actuator command computation → Actuator command execution **Note:** Since Kp and Kd gains cannot be efficiently converted to parallel actuator commands, users requiring force control are recommended to command joint torque (tau) directly instead of achieving force control indirectly through PD control. This is particularly relevant for

Function Name	publishRobotCmd
	reinforcement learning policies that output position or velocity actions, where the controller converts PD commands into torque commands internally.
Return Value	Success: return True Failure: return False
Remarks	- datatypes.RobotCmd structure prototype defined as follows:

▼

python 复制代码

```
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 whi
ch 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 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
```

Code Example：

▶

python 复制代码

4.2.8 subscribeSensorJoy Interface

Function Name	subscribeSensorJoy
Function Prototype	def subscribeSensorJoy(self, callback: Callable[[datatypes.SensorJoy], Any])

Function Name	subscribeSensorJoy
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 datatypes.SensorJoy structure for processing.
Parameters	callback: Callback Function for receiving remote controller data with parameter type SensorJoyConstPtr.
Return Value	Success: return True Failure: return False
Remarks	- datatypes.SensorJoy structure prototype defined as follows:

▼

python 复制代码

```
import sys

class SensorJoy(object):
    __slots__ = ['stamp','axes','buttons']
    def __init__(self):
        self.stamp = 0     # Timestamp corresponding to the sensor input time, in nanosec
onds
        self.axes = []     # Values representing controller joystick manipulation
        self.buttons = []  # Values representing controller button operation states
```

Code Example：

▶

python 复制代码

4.2.9 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 datatypes.DiagnosticValue object, allowing continuous health monitoring and prompt response to detected issues.

Function Name	subscribeDiagnosticValue
Parameters	Callback function for receiving diagnostic data. The parameter type is datatypes.DiagnosticValue, which contains fields such as timestamp, level, name, code, and message.
Return Value	Success: return True Failure: return False
Remarks	- datatypes.DiagnosticValue structure prototype defined as follows:

▼

python 复制代码

```
import sys

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 复制代码

4.2.10 publishJsonMessage Interface

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

Function Name	publishJsonMessage
Return Value	None
Remarks	Valid only in high-level development mode.

Code Example：

▶

python 复制代码

4.2.11 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 High-Level Application Protocol Interface. The callback is triggered in the following cases: 1. When the robot returns a response to a previously sent JSON command (publishJsonMessage). 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：

▶

python 复制代码

4.2.12 Reference Example

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

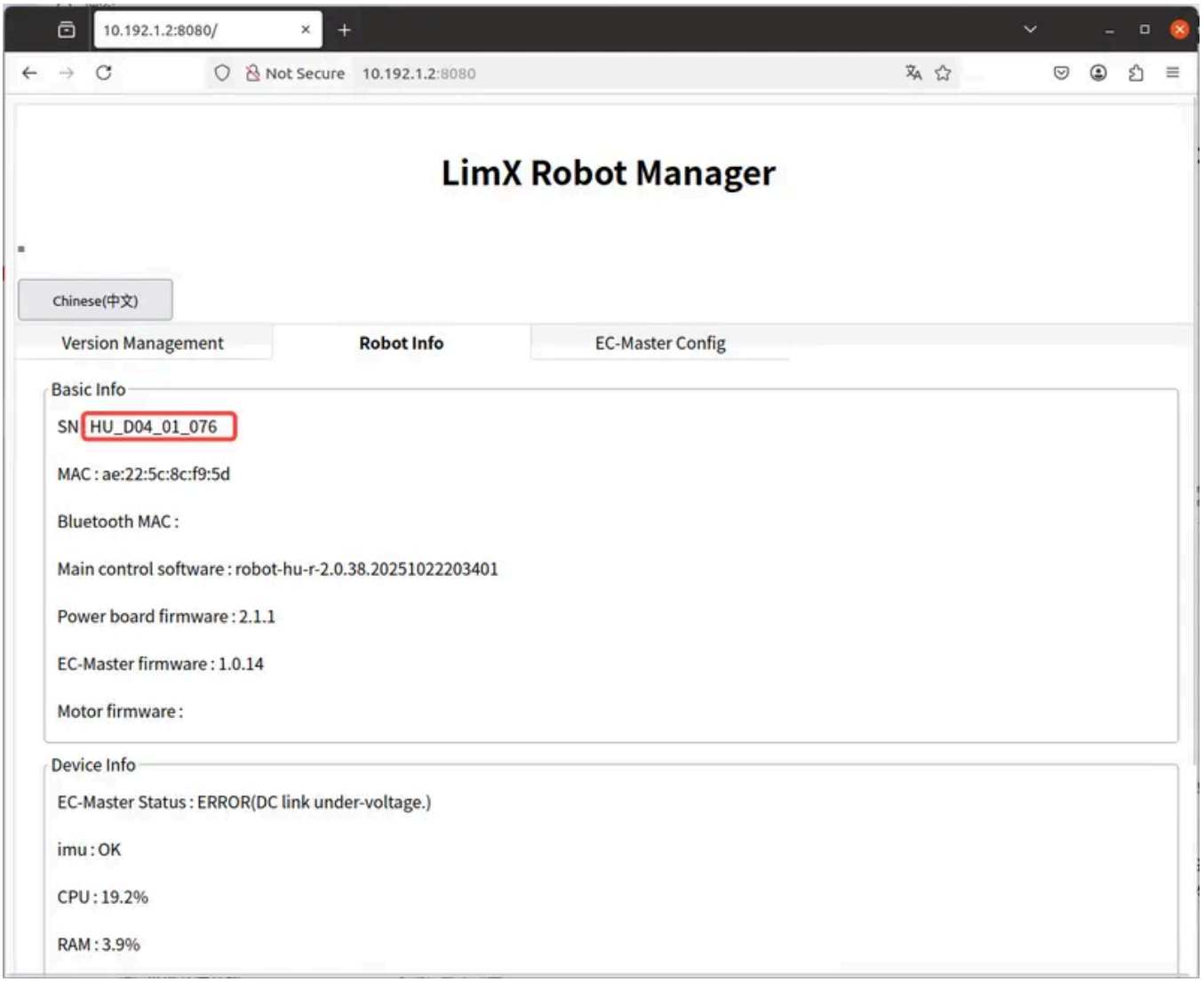
5 Check and Set the Robot Model

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 复制代码

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

6 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.

6.1 Running the Simulator

1. Environment Requirements: Python 3.8 or higher is recommended

2. Open a Bash Terminal
3. Download the MuJoCo Simulator Code:

▼

bash 复制代码

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

4. Install the Motion Control Development Library:

- Linux x86_64 Environment

▼

bash 复制代码

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

- Linux aarch64 Environment

▼

bash 复制代码

```
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:

▼

bash 复制代码

```
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):

▼

bash 复制代码

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

6. Run the MuJoCo simulator：

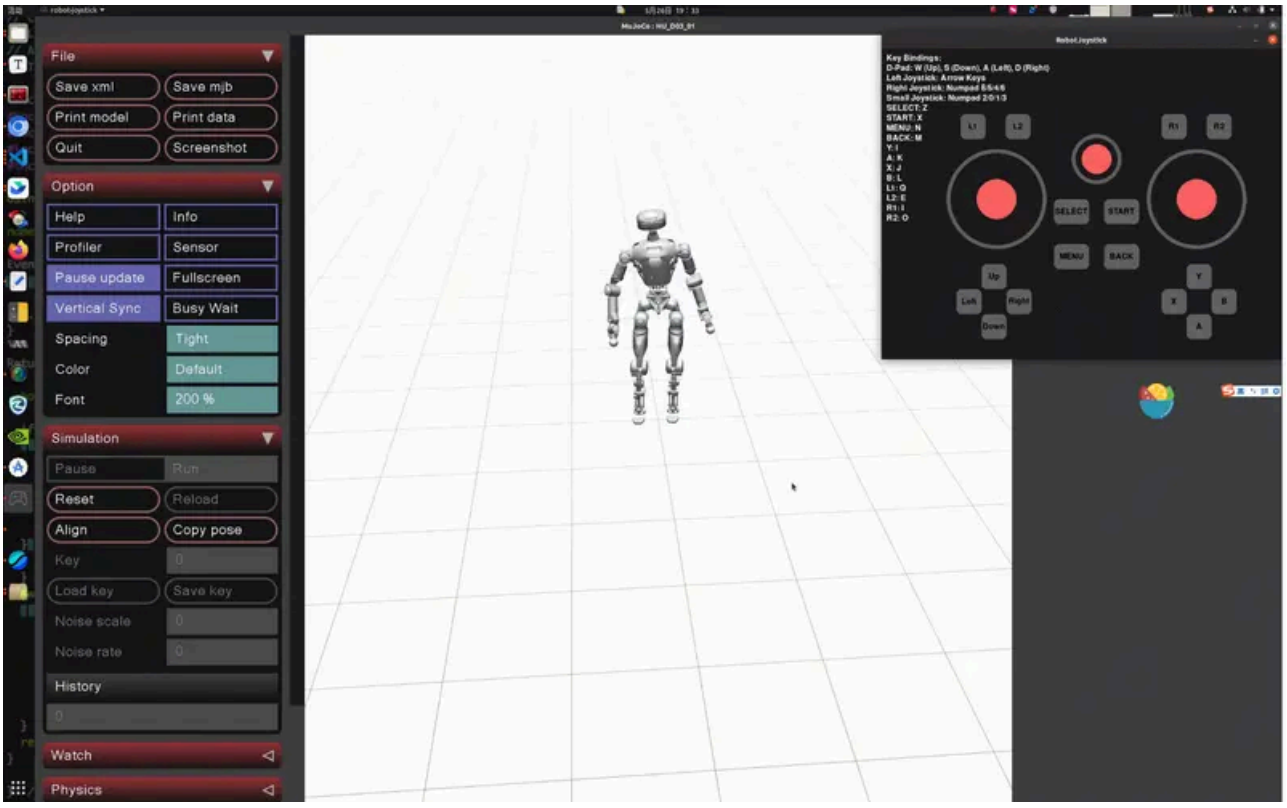
▼

bash 复制代码

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

6.2 Demonstration Results

> Actual performance may vary depending on your system configuration.



7 RL Algorithm Deployment

7.1 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.

7.2 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.

7.3 Deployment with ROS2

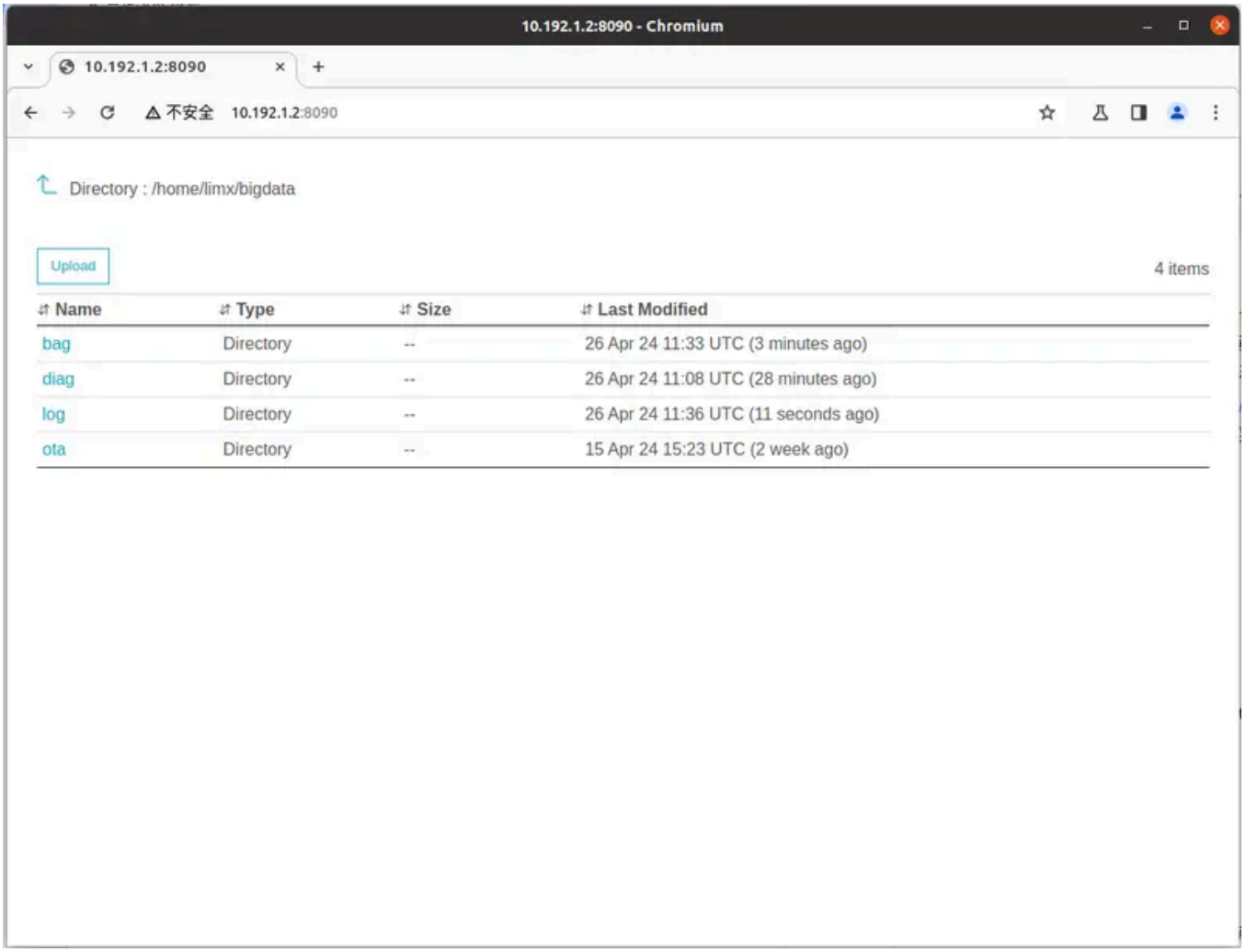
- **Github Repository:** https://github.com/limxdynamics/humanoid-rl-deploy-ros2
- A reinforcement learning deployment framework based on ROS2, allowing rapid deployment of trained models on Oli.

7.4 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.

8 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.



8.1 Data Package Visualization and Analysis

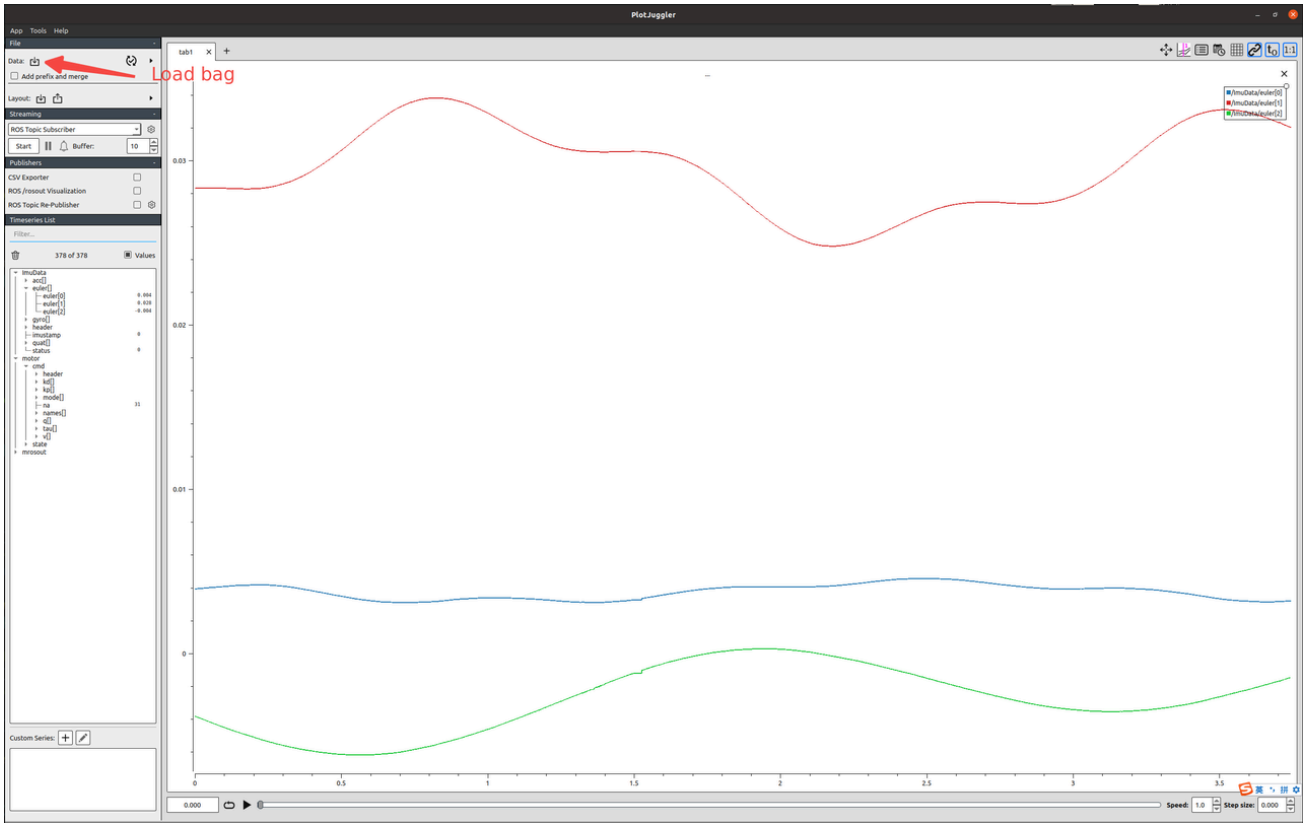
- **Downloading Data Packages:** After downloading a `.bag` file, you can use PlotJuggler to visualize and analyze the data. If the downloaded file is in `.bag.active` format, run the following shell command to reindex it and generate a new `.bag` file for PlotJuggler to load properly.

  ▼

  bash 复制代码

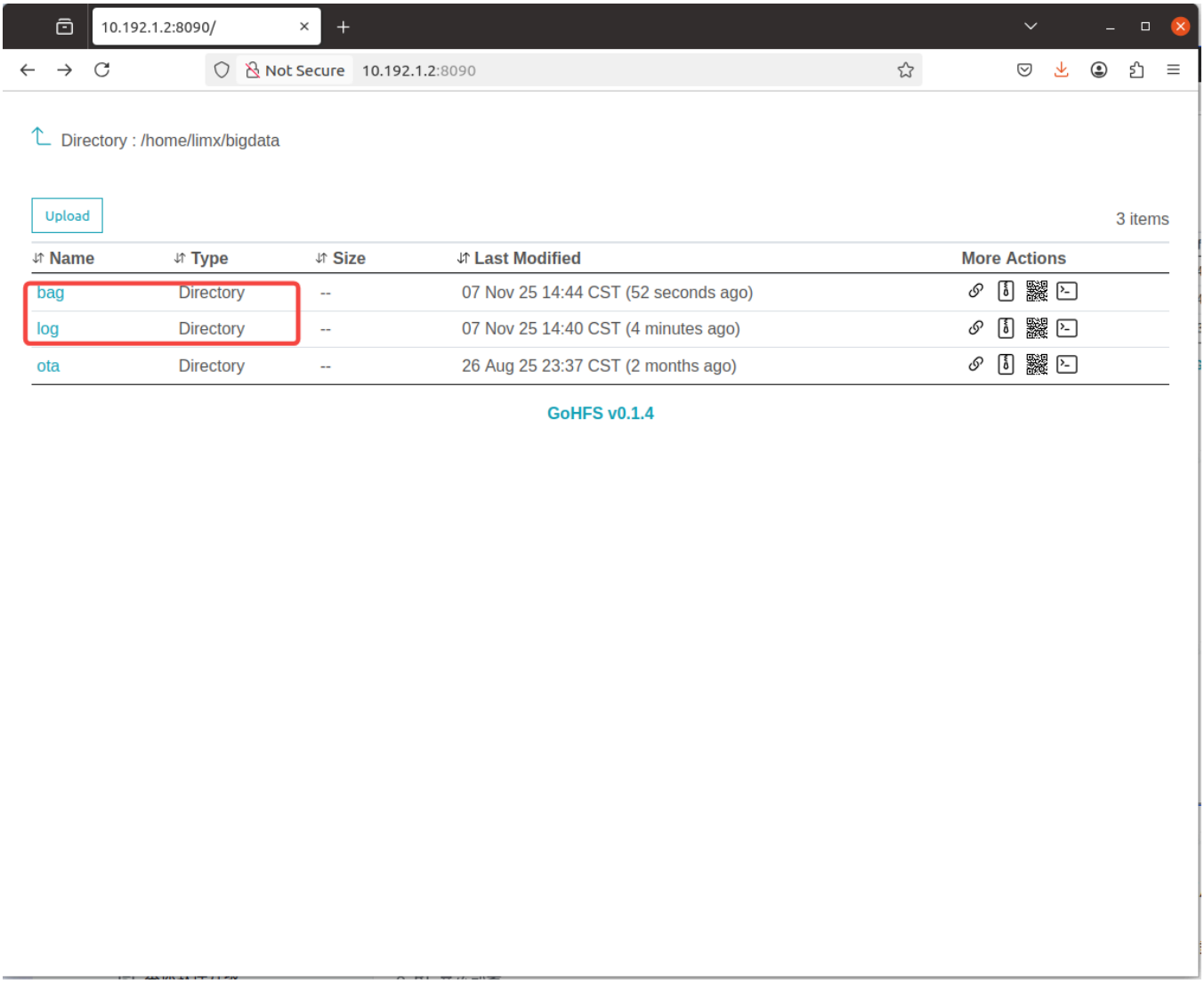
  ```
  rosbag reindex your_file.bag.active
  mv your_file.bag.active your_file.bag
  ```

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



8.2 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.

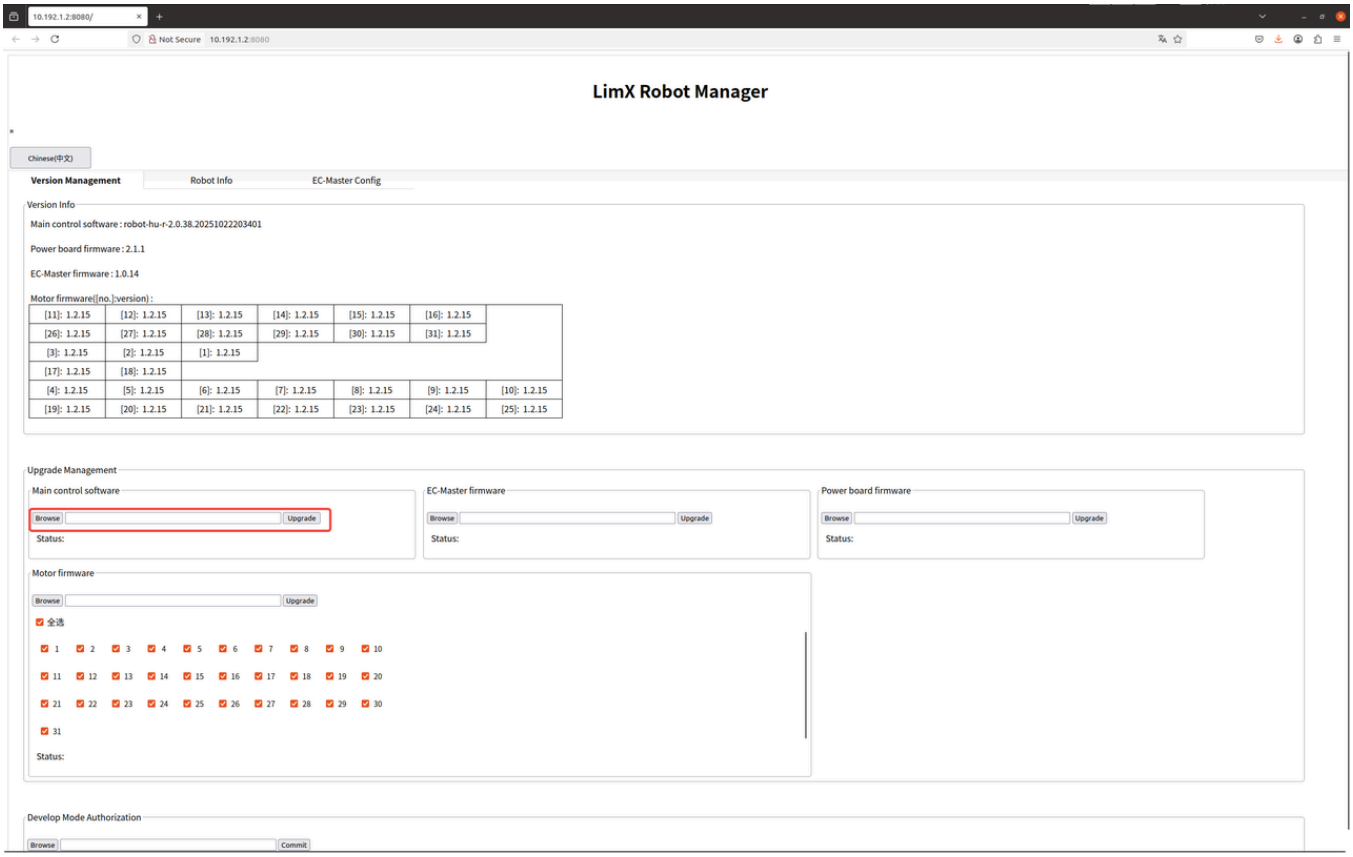


9 Robot Software Upgrade

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.



10 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 复制代码

     ```
     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 复制代码

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

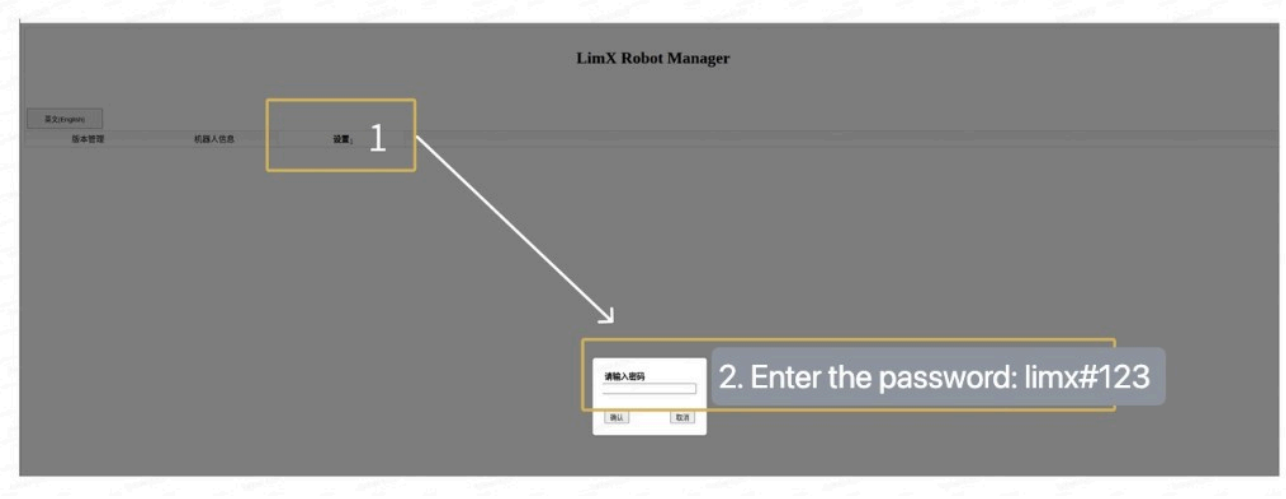
11 RealSense Camera Data Acquisition

Note:

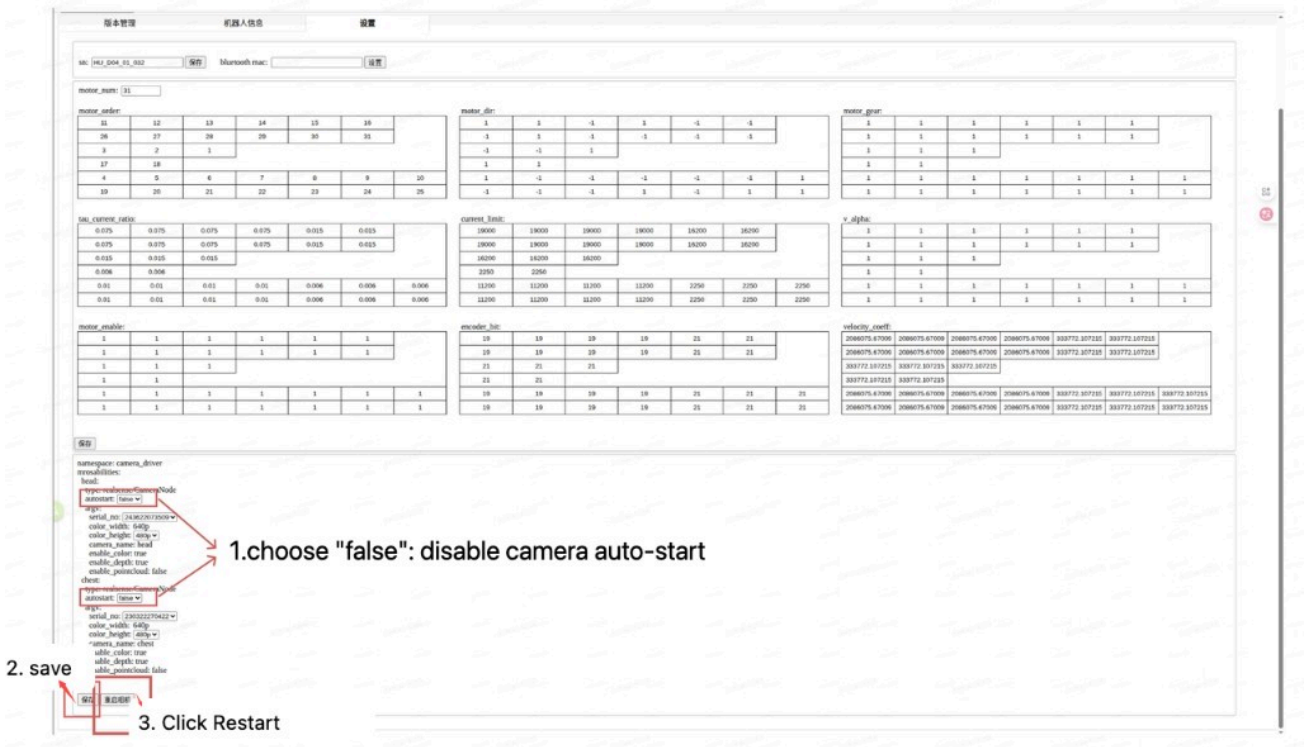
1. The camera driver starts automatically at power-on. Disable camera driver auto-start before acquiring camera data to prevent interference. (Supported in main controller software V2.0.33 or later.)
2. Disabling camera driver auto-start will prevent the provided data acquisition kit from functioning.

11.1 Disable Camera Auto-Start

1. Enter `10.192.1.2:8080` in a web browser to access the interface:



2. Set to disable and save:



11.2 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. Log in to the Developer PC via SSH.
2. Log in to the ROS 1 Environment.

    ▼

    shell 复制代码

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

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

    ▶

    bash 复制代码

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

    ▼

    shell 复制代码

    ```
    /bin/bash rs_camera.sh
    ```

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

   ▼

   shell 复制代码

   ```
   rostopic list
   ```