

AGIQUAD D1 Max Quadruped Robot

SDK Development Guide



Version: V0.1.0

Date: July 2026

Contents

1. Product Introduction.....	5
1.1 Product Overview.....	5
1.2 Product Checklist.....	6
1.3 Product Specifications.....	6
1.4 Rear-Mount Payload Installation.....	12
1.4.1 Payload Mounting Hole Layout.....	12
1.4.2 Center-of-Gravity Constraints.....	13
1.5 Electrical Expansion Interfaces.....	13
1.5.1 Power Port - 48V (10A).....	15
1.5.2 Power Port - 24V (20A).....	17
1.5.3 Ethernet Port (24V).....	18
1.5.4 Ethernet Port (12V).....	21
1.5.5 Serial Port.....	23
1.5.6 USB Port.....	24
1.6 D1 Max Hardware Architecture Diagram.....	25
2. SDK Development Guide.....	26
2.1 SDK Introduction.....	26
2.2 SDK Software Service Interface List.....	26
2.3 Safety Protection Mechanisms.....	27
2.3.1 Hard Emergency Stop.....	27
2.3.2 Soft Emergency Stop.....	28
2.3.3 Internal Exception Protection (Automatic Trigger).....	28
2.4 Environment Dependencies.....	28
2.5 Device Login.....	28
2.6 Obtaining the SDK.....	30
2.7 Demo Execution.....	30
2.8 SDK Control Instructions.....	31
2.9 Joint Naming Convention.....	33
2.10 Sensor Device Coordinate Definitions.....	34
3. SDK Function Introduction.....	36
3.1 Introduction to HighLevel API Functions.....	36
3.1.1 Error Code Mechanism.....	36
3.1.2 Type Definitions.....	39
3.1.3 Constructor & Destructor.....	39
3.1.4 Connection Management Interfaces.....	39
3.1.5 Callback Configuration Functions.....	40
3.1.6 Control Interfaces.....	45
3.1.7 Motion & Posture Control Interfaces.....	53
3.1.8 Data Configuration Interfaces.....	55
3.1.9 Version Information.....	58
4. Sensor Data Retrieval.....	60
4.1 Video Stream Data.....	60
4.2 LiDAR Data.....	61
4.2.1 ROS2 Interface Definition.....	61
4.2.2 Retrieving LiDAR Data.....	62
4.3 Ultrasonic Radar Data.....	62
4.3.1 ROS2 Interface Definition.....	62
4.3.2 Retrieving Ultrasonic Radar Data.....	63
4.4 IMU Data.....	63
4.5 Camera Data.....	63

5. Debugging Instructions	64
5.1 Control Authority Switching Instructions.....	64
5.2 Autonomous Docking & Recharging Instructions.....	64
5.3 ROS2 Multi-Machine Communication Configuration.....	66
5.4 Log Viewing.....	67
5.5 Device Process Utility Usage.....	70
5.6 Connecting to an External Wi-Fi Network.....	71
5.7 Accessing the Wired Ethernet Port While Connected to Wi-Fi.....	71

Document Revision History

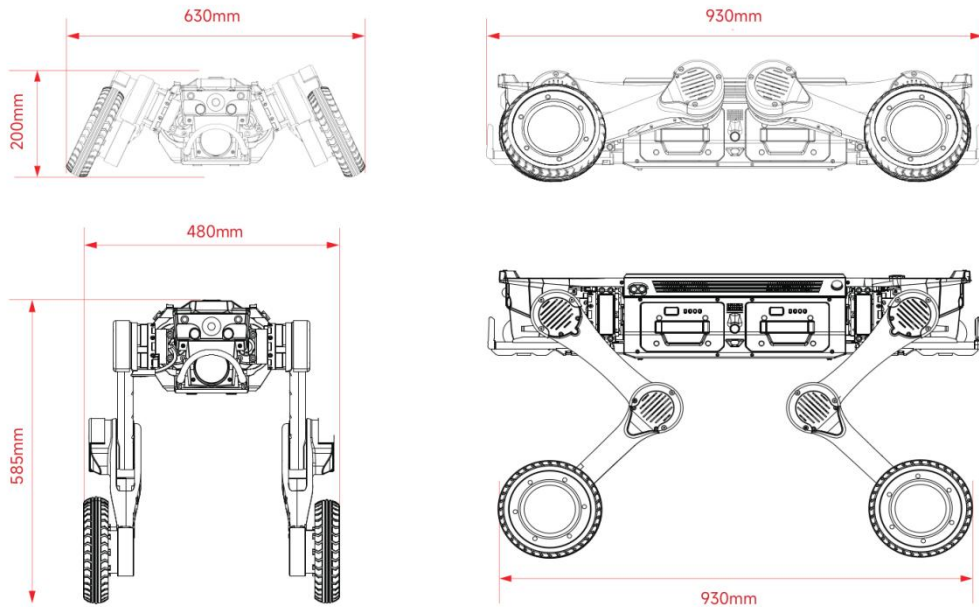
Revision Date	Revision Description	Version
Sep 2025	● Initial pre-test version	V0.0.1
Oct 2025	● Provided basic robot control interfaces ● Supported robot status data callbacks ● Supported fault information callbacks	v0.0.2
Nov 2025	● Added stair-climbing mode interface ● Added high-platform climbing interface in locomotion mode ● Added mode status retrieval interface	v0.0.3
Dec 27, 2025	● Added system version retrieval interface ● Added UDP communication support ● Optimized error messages on connection failure ● Internally adapted to new communication protocol	v0.0.4
Jan 17, 2026	● Added Section 2.10: Robot Body Coordinate System Orientation ● Added Section 4.4: Robot Body Sensor Coordinates	v0.0.41
Jan 15, 2026	● Added gait posture interface (Gait(...)) ● Added slim posture interface (Slim(...)) ● Added configuration interface for automatic reconnection after disconnection ● Added interface to retrieve current connection status ● Added examples for both automatic and manual reconnection ● Added send-result callback parameter in asynchronous interfaces	v0.0.5
Feb 08, 2026	● Added control authority switching functionality	v0.0.6

	● Added "4.5 RTK Usage Instructions" ● Fixed IMU acc/gyro sign inversion issue	
Mar 17, 2026	● Added autonomous docking & recharging functionality ● Modified the NX SSH login username ● Updated front and rear LiDAR IP addresses and ROS topics ● Added "1.6 D1 Max Hardware Architecture Diagram"	v0.0.8
Mar 31, 2026	● Added a dedicated velocity-information reporting interface, with a configurable reporting frequency ● Adapted to joint temperature reporting for point-foot mode ● Improved autonomous docking & recharging success rate ● Added a mapless docking & recharging API interface	v0.0.9
Apr 30, 2026	● Added a mapless autonomous docking & recharging SDK interface ● Added a debugging guide	v0.1.0
Jul 2026	● Errata/corrections	v0.1.0

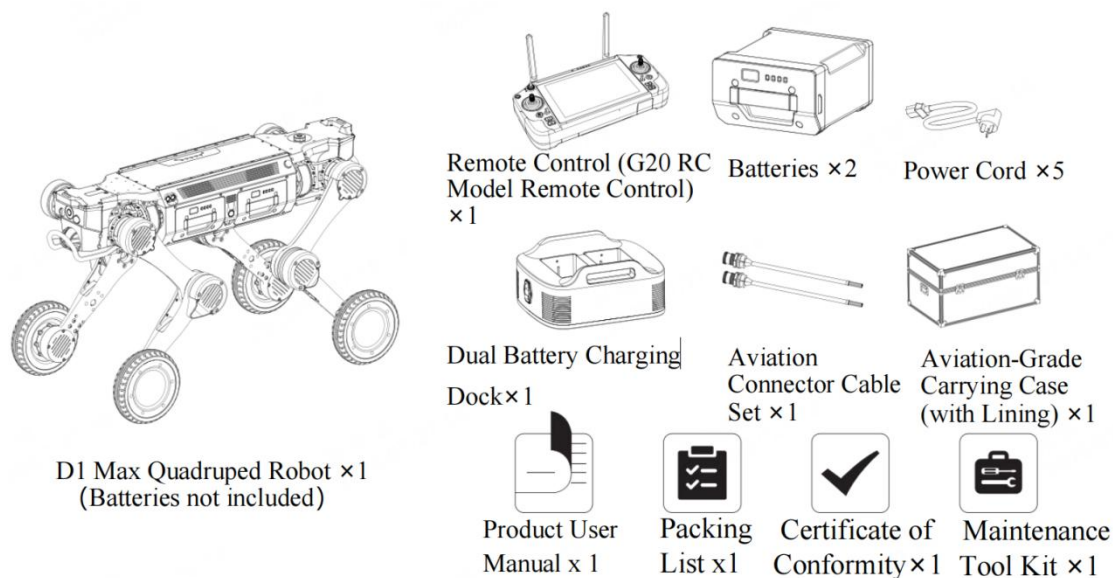
1. Product Introduction

1.1 Product Overview

The AGIBOT D1 Max Quadruped Robot is a wheel-legged bionic quadruped robot featuring low self-weight, high payload capacity, long endurance, robust protection, and agile, stable locomotion. Each leg is equipped with 3 joint motors and 1 hub motor. The robot integrates optical cameras, LiDAR, ultrasonic sensors, an IMU, an RTK module, and other sensing devices. It is powered by a high-performance onboard computing platform for motion control, autonomous navigation and positioning, environmental perception, and other functions. It provides extensive power and communication interfaces, supporting a wide range of task-payload expansions.



1.2 Product Checklist



1.3 Product Specifications

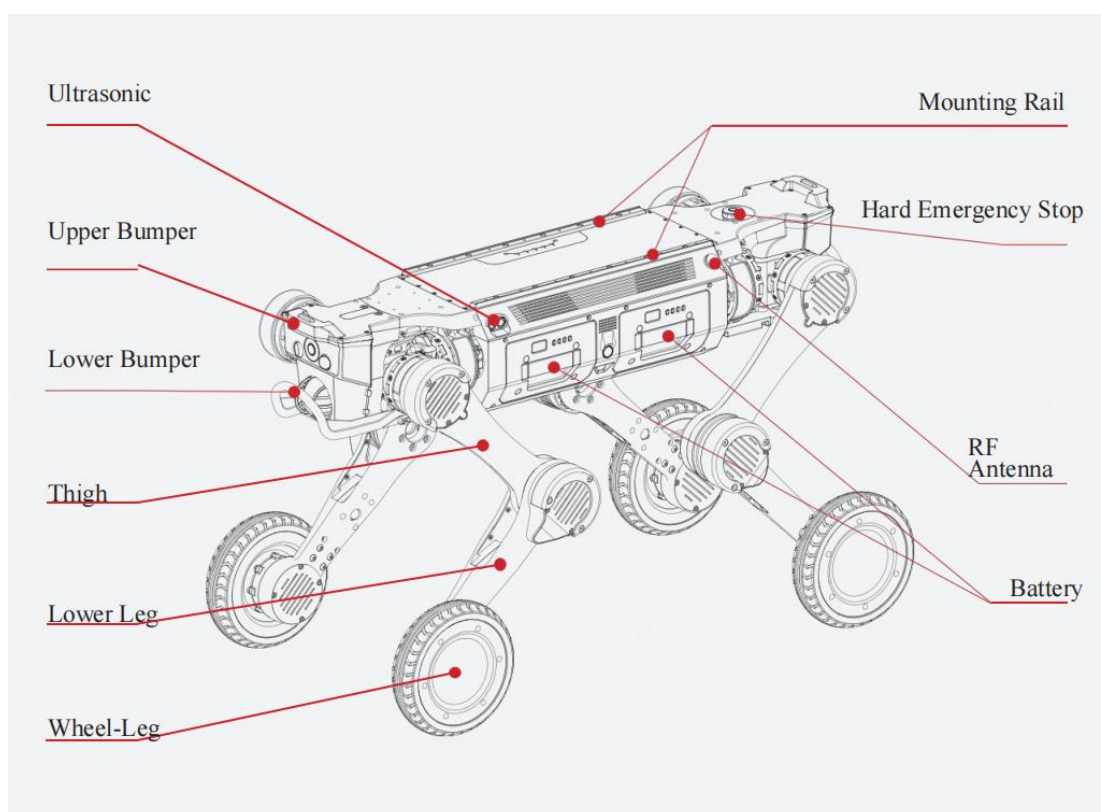
Category	Specification	Description
Basic Information	Standing Dimensions	930mm * 480mm * 585mm
	Prone Dimensions	930mm * 630mm * 200mm
	Total Weight (including battery)	41kg
	Operating Temperature	-20° ~55°
	Ingress Protection Rating	IP67
	Battery Module	504.9Wh * 2, dual batteries, supports quick-release/quick-swap and hot-swapping
	Charging Time	< 1.5h
	Battery Life	Unloaded: 5 ± 0.5h; Full load: 3.5 ± 0.5h

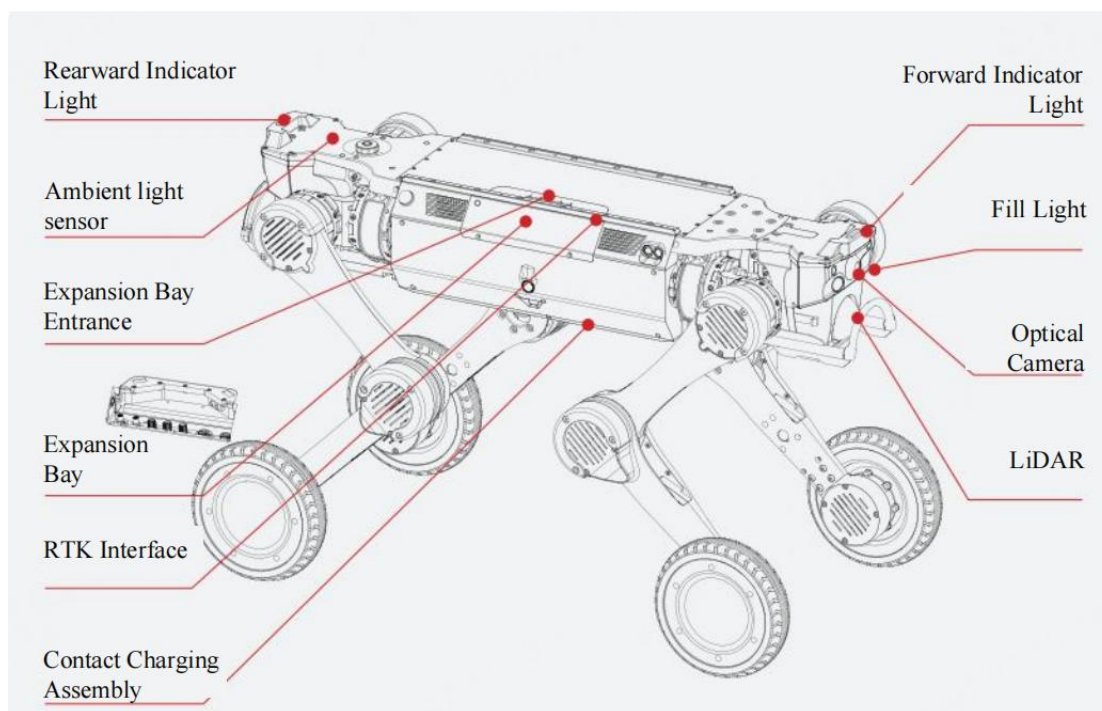
Performance Parameters	Maximum Speed	6m/s
	Rated Stable Payload	30kg
	Continuous Step-Climbing Height	25cm
	Maximum Obstacle Clearance Height	80cm
	Maximum Gap Crossing Width	80cm
	Payload Crossing Height	50cm
	Minimum Passage Width	50cm
	Maximum Climbing Angle	45°
	Minimum Turning Width	Supports in-place turning and forward/backward direction switching
	Main Control Compute	157 TOPS
Sensing Units	LiDAR	One 96-channel LiDAR (Airy 96-channel) each front and rear, 360° × 90° coverage, 120m detection range
	Wide-Angle Camera	FPV Camera Configuration: Front x1, Rear x1, industrial-grade 8 MP, Field of view: DFOV 122° , HFOV 111° , VFOV 70°
	Fill Light	Fill light * 4, 2 on each side, paired with the FPV camera. Illumination distance $\geq 3\text{m}$
	IMU	Automotive-grade IMU * 1

	Ultrasonic Sensor	One ultrasonic sensor each on the left and right sides; supports obstacle detection within 5m
	RTK Module	GNSS RTK * 1
	Microphone & Speaker	Supported on select versions
	Ambient Light Sensor	Supported on select versions
Function List	Locomotion Performance	Supports stepping, gliding, and crawling gaits; capable of straight-line movement forward/backward/left/right, in-place rotation, etc; supports knee-joint posture transitions. Operates on various terrains such as gravel, asphalt, grass, sandy soil, forest floor, and rubble;
	Wheel/Foot Quick-Swap	Supports quick-swapping between wheeled-leg and point-foot modes
	Real-Time Image Transmission	Supported
	OTA Upgrades	Supported
	Secondary Development	Supported; provides SDK development packages; robot model includes 3D CAD and URDF files for simulation
	Power Output Interfaces	5V, 12V, 24V, 48V; maximum output power 480W
	Communication Interface	Gigabit Ethernet port * 2, USB 3.0 * 2, Serial port * 1
Communication	Wi-Fi	Supported
	Bluetooth	Supported
	RF Communication	Supported

Accessories	Adapter Charging Dock	Standard
	Remote Controller with Integrated Display	Standard; control range 700-1000m
	Autonomous Charging Dock	Optional
	Aviation-Grade Shipping Case	Standard
Other	Warranty Period	1 year

Note: The above parameters are laboratory test data. Actual performance may vary depending on factors such as the operating environment and usage methods; please refer to the actual product performance.





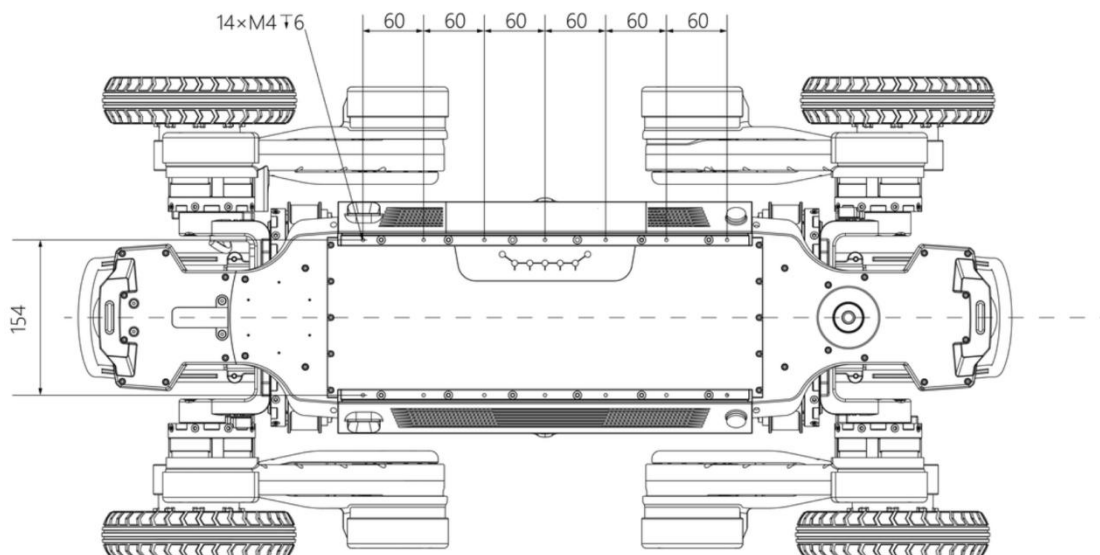
● Joint Parameters

Part	Parameters	Value
Hip Roll	Max Torque	150N • m
	Max Speed	190RPM
	Weight	0.93kg
	Joint Limits	-39.9° ~30°
Hip Pitch	Max Torque	150N • m
	Max Speed	190RPM
	Weight	0.93kg
	Joint Limits	-140° ~140°
Knee Pitch	Max Torque	150N • m
	Max Speed	190RPM
	Weight	0.93kg
	Joint Limits	-160° ~160°
Foot	Max Torque	33N • m
	Max Speed	1200RPM
	Weight	1.275kg
	Radius	9cm
	Joint Limits	-180° ~180°

1.4 Rear-Mount Payload Installation

1.4.1 Payload Mounting Hole Layout

The robot's back provides a flat, level surface with a payload capacity of 25 - 30kg. Two guide rails are deployed on the back, each fitted with 14 M4-threaded mounting holes to allow convenient installation of top-mounted payloads. The specific dimensions of the dual guide rails are as follows:

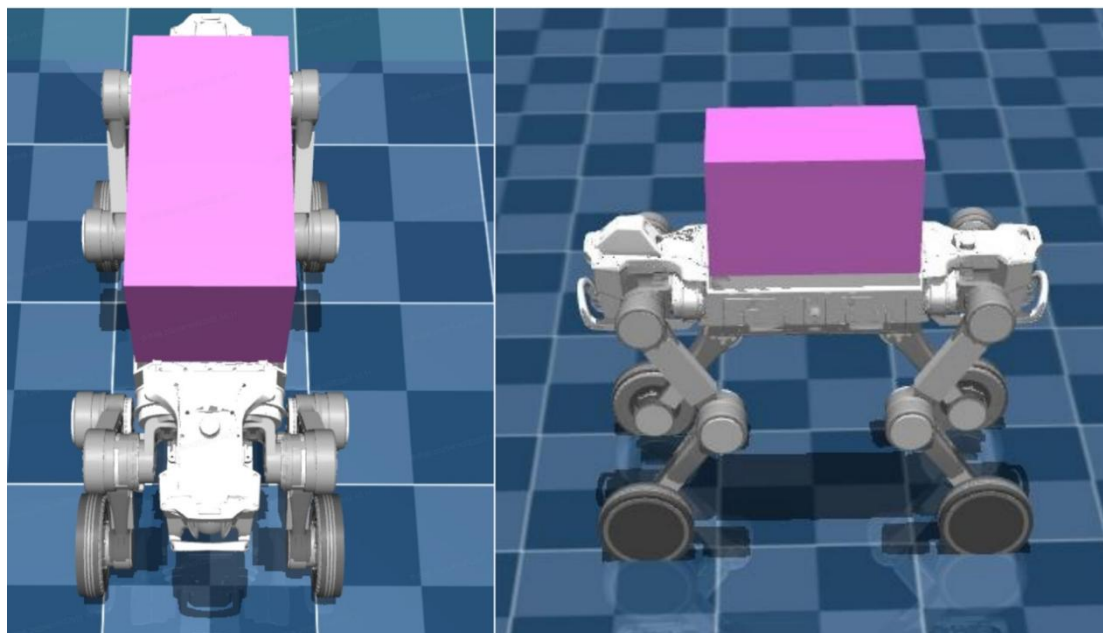


Once a payload is installed on the robot, only basic locomotion mode is supported. Do not use the high-platform mode once a payload is mounted

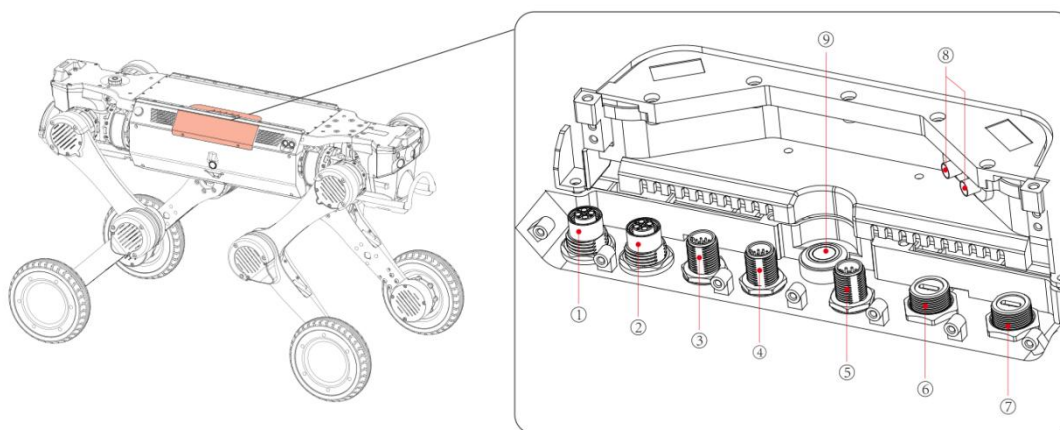
Mounting a payload on the robot's back may affect its locomotion performance. Please consult after-sales support personnel before installing any payload

1.4.2 Center-of-Gravity Constraints

To prevent collision or interference between the legs and a top-mounted payload while traversing various terrains, the D1 Max provides the following recommended payload mounting envelope after payload installation: 40 cm (L) $\times$ 22 cm (W) $\times$ 24 cm (H). The mounting space is centered on the robot's back, as shown in the figure below:



1.5 Electrical Expansion Interfaces



No.	Interface Name
1	48V (10A)
2	24V (20A)
3	24V (3A) + Ethernet Port
4	12V (5A) + Ethernet Port
5	Serial Port (RS232 * 1, RS485 * 1, SBUS * 2, PPS * 1)
6, 7	USB 3.0 (5V, 1A)
8	RTK Antenna Interface
9	Expansion Bay Button

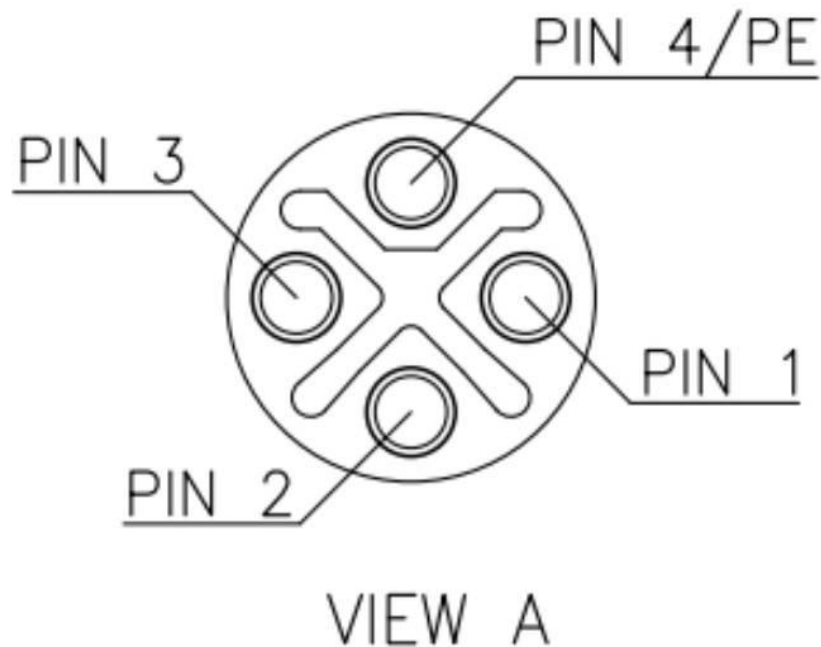
- The combined power output of all power interfaces shall not exceed 480W;
- All aviation connectors are equipped with protective caps to achieve IP67 rating. Ensure that protective caps are properly fitted on any unused aviation connectors;
- The expansion bay button controls power delivery to the expansion interfaces. Before connecting any payload wiring, be sure to turn off the expansion bay power supply.

Aviation Connector Expansion Function Interface Description:

Device Name	Device Path
RS232	/dev/ttyCH9344USB4
RS485	/dev/ttyCH9344USB5
SBUS1	/dev/ttyCH9344USB2
SBUS2	/dev/ttyCH9344USB3

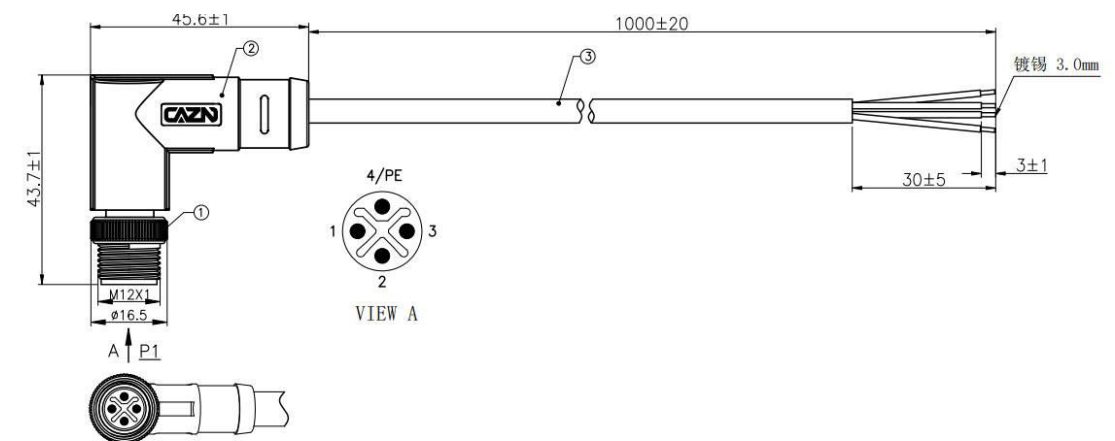
1.5.1 Power Port - 48V (10A)

- Female Aviation Connector



Connector Name		48V 10A
Board-End Connector Model		M12-S4S-GPFM16-135-CS24044
Pin Assignment	1 (Brown)	GND
	2 (Red)	48V
	3 (Blue)	48V
	4 (Yellow-Green)	GND

- Mating Cable Harness

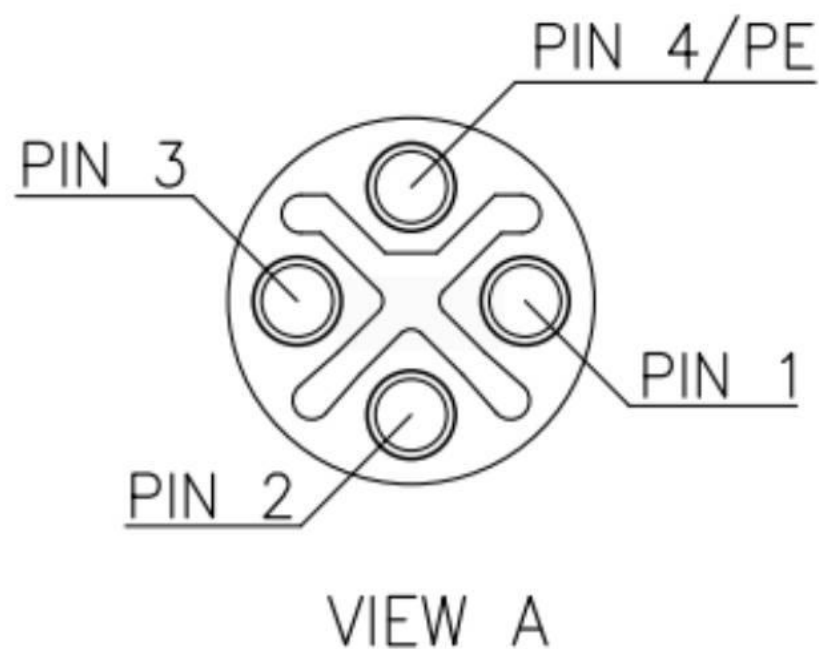
[illegible]

Wire Diagram 接线定义		
P1	Wire Color 线的颜色	Open 开放
1	Brown 棕	
2	Red 红	
3	Blue 蓝	
4	Yellow Green 黄绿	
Shell 外壳	NA	

3	YY07. 04. 0150 4C*1.5mm ² (15AWG)+无纺布, 内芯PVC, OD: 2.8外被黑 PVC OD: 9.0 (1. 标: 2. 红; 3. 黄绿; 4. 蓝) 柔性电源电缆 (4C1.5-002)			
2	M12包胶成型 黑色PVC			
1	M12螺纹成型式, 长体胶芯针型插头半成品, 4芯 S 编码, 短屏蔽管 1 铜大体积螺母 (3+PE款): CS24-M12-P4S-SF01C			
No.	Specification			
GENERAL TOLERANCE (UNLESS SPECIFIED)		绘图 DWN	李平 2025. 10. 10	STA·思
X. ±0.50 X.X ±0.30 X.XX ±0.20 X.XXX ±0.15	确认 CHK	核准	品名	

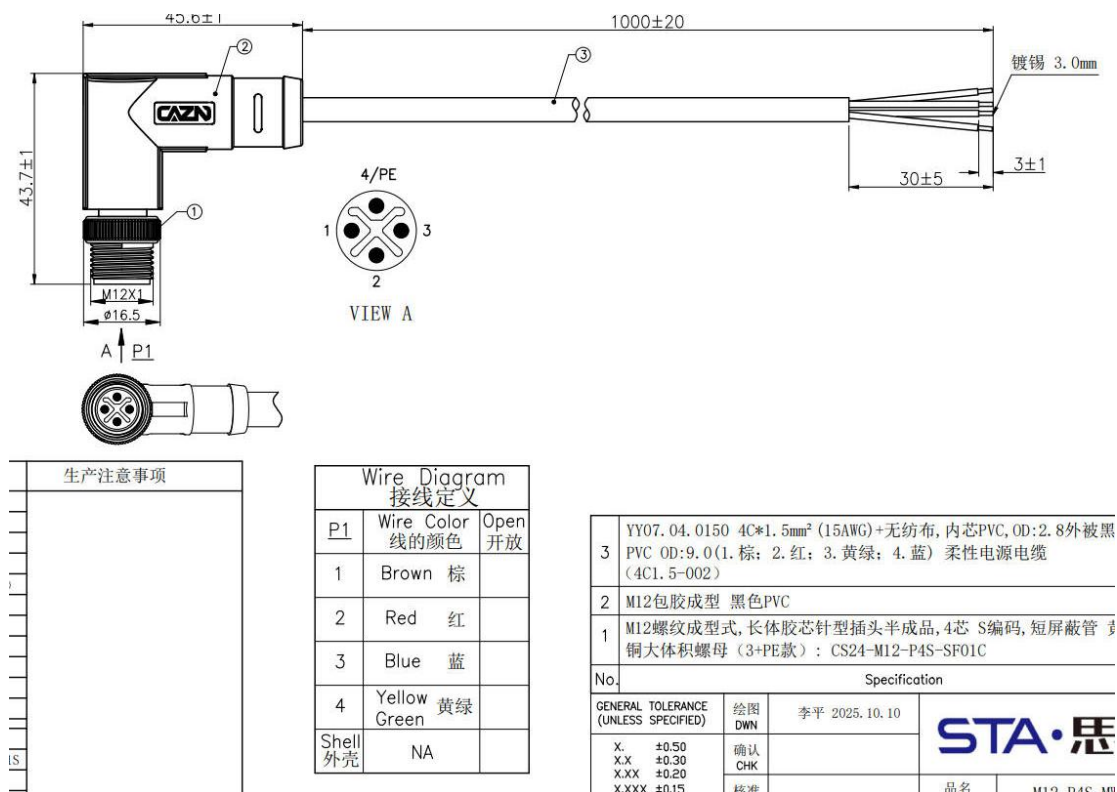
1.5.2 Power Port - 24V (20A)

- Female Aviation Connector



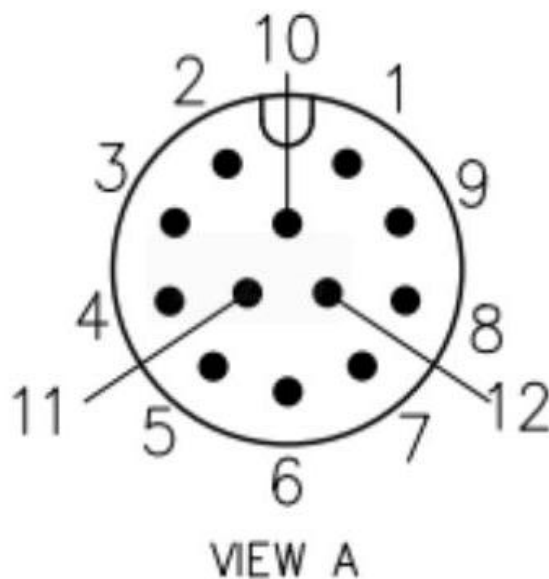
Connector Name		24V 20A
Board-End Connector Model		M12-S4S-GPFM16-135-CS24044
Pin Assignment	1 (Brown)	GND
	2 (Red)	24V
	3 (Blue)	24V
	4 (Yellow-Green)	GND

- Mating Cable Harness



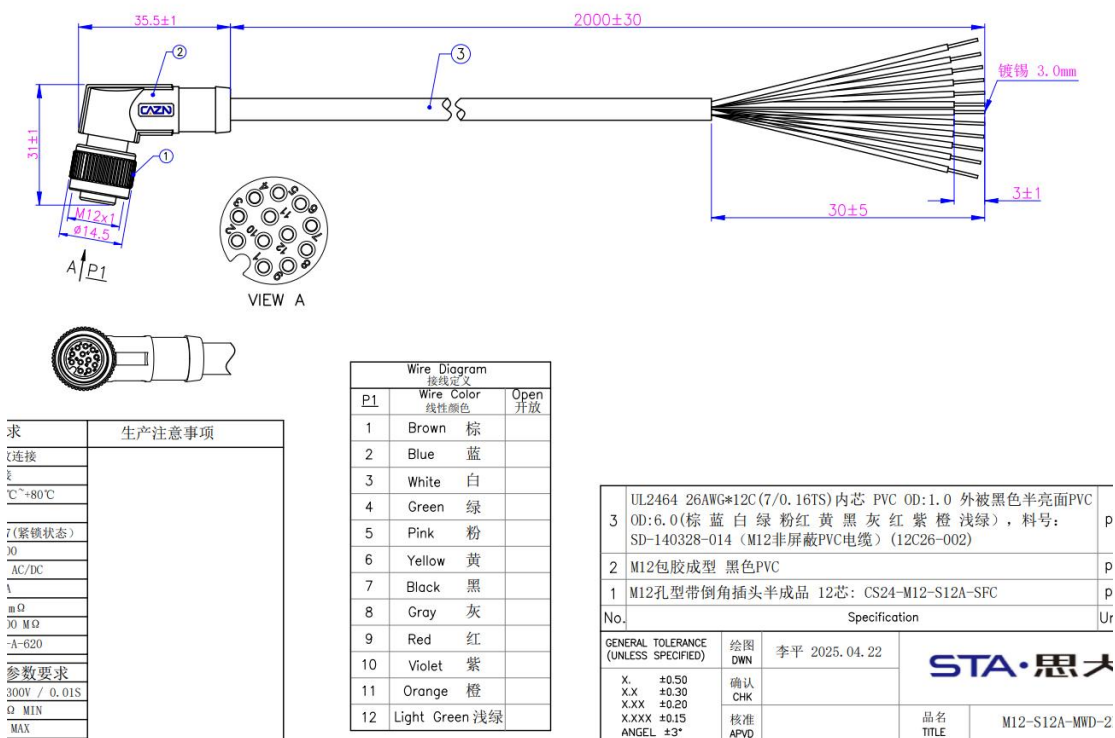
1.5.3 Ethernet Port (24V)

- Female Aviation Connector



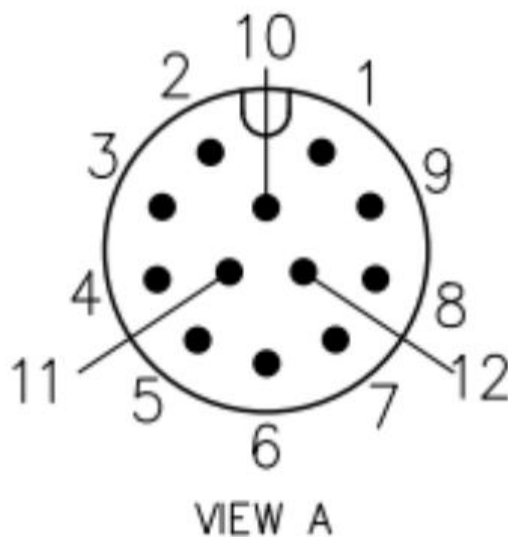
Connector Name		Ethernet + 24V
Board-End Connector Model		M12-P12A-GPFM12
Pin Assignment	1 (Brown)	GND
	2 (Blue)	24V
	3 (White)	24V
	4 (Green)	Ethernet A+
	5 (Pink)	Ethernet A-
	6 (Yellow)	Ethernet B+
	7 (Black)	Ethernet B-
	8 (Gray)	Ethernet C+
	9 (Red)	Ethernet C-
	10 (Purple)	GND
	11 (Orange)	Ethernet D+
	12 (Light Green)	Ethernet D-

- Mating Cable Harness



1.5.4 Ethernet Port (12V)

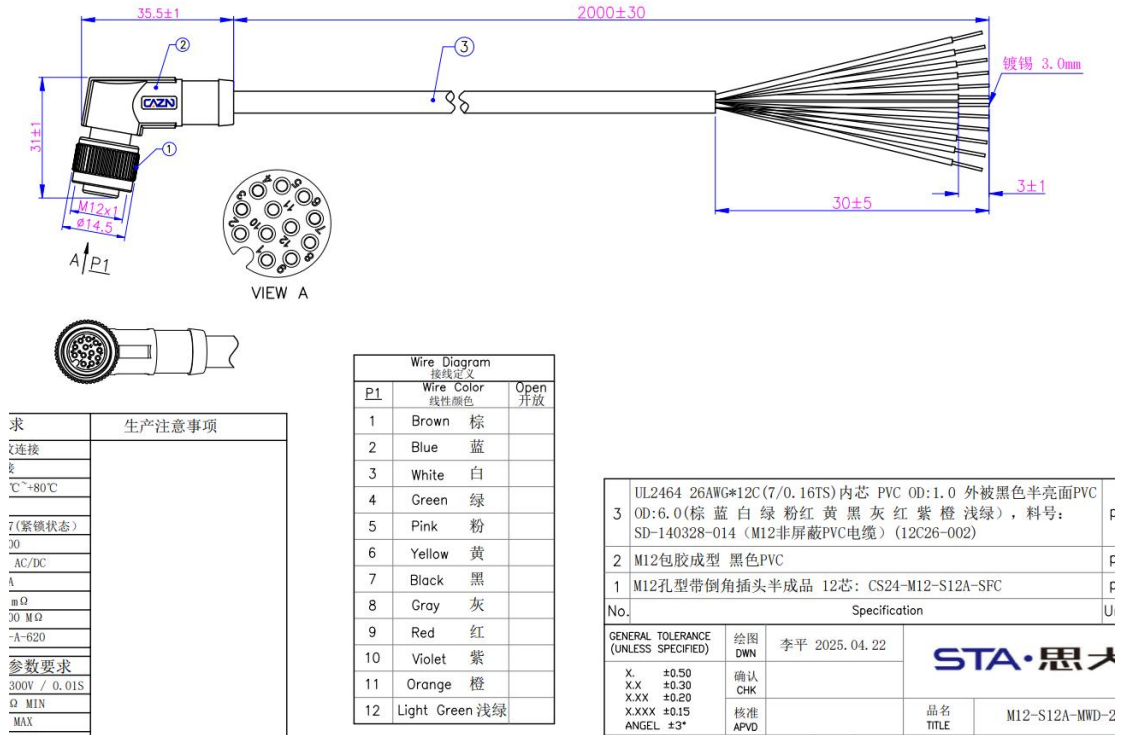
- Female Aviation Connector



Connector Name		Ethernet +12V
Board-End Connector Model		M12-P12A-GPFM12
Pin Assignment	1 (Brown)	GND
	2 (Blue)	12V
	3 (White)	12V
	4 (Green)	Ethernet A+
	5 (Pink)	Ethernet A-
	6 (Yellow)	Ethernet B+
	7 (Black)	Ethernet B-
	8 (Gray)	Ethernet C+
	9 (Red)	Ethernet C-
	10 (Purple)	GND

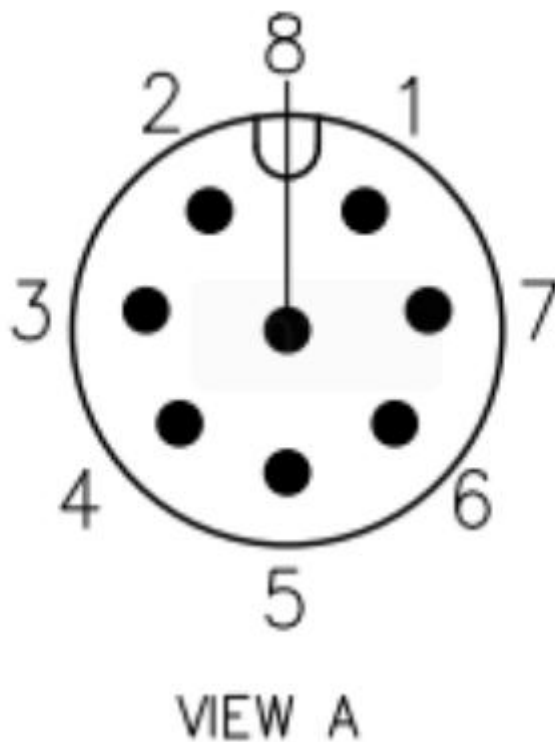
	11 (Orange)	Ethernet D+
	12 (Light Green)	Ethernet D-

● Mating Cable Harness



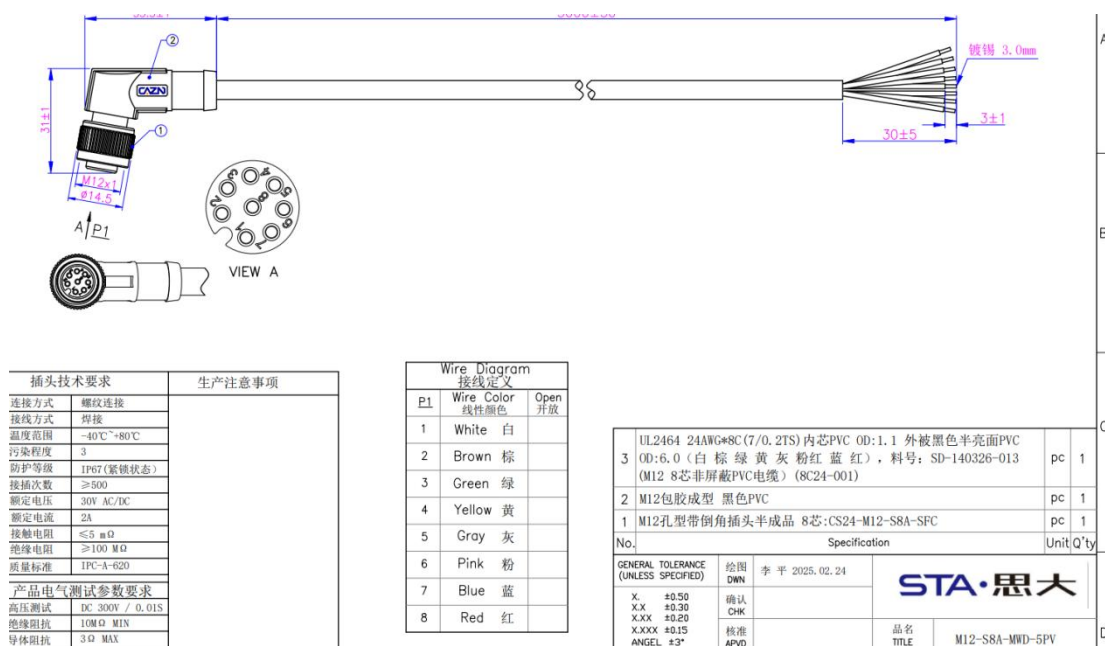
1.5.5 Serial Port

- Female Aviation Connector



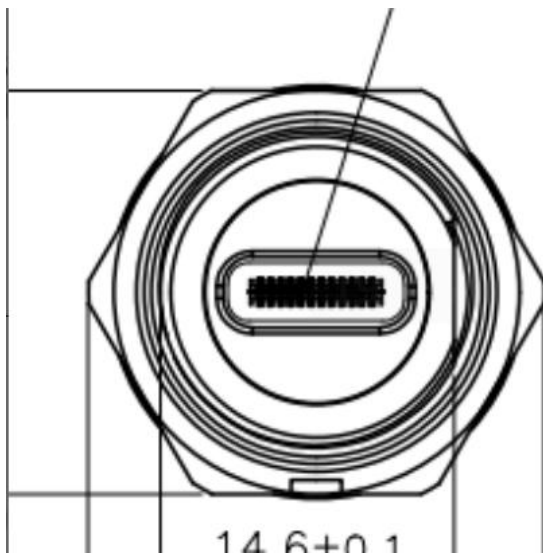
Connector Name		Serial Port (RS232, RS485, SBUS * 2, PPS)
Board-End Connector Model		M12-P8A-GPFM12
Pin Assignment	1 (White)	PPS
	2 (Brown)	SBUS1
	3 (Green)	SBUS2
	4 (Yellow)	RS485A
	5 (Gray)	RS485B
	6 (Pink)	RS232_TX
	7 (Blue)	RS232_RX
	8 (Red)	GND

● Mating Cable Harness



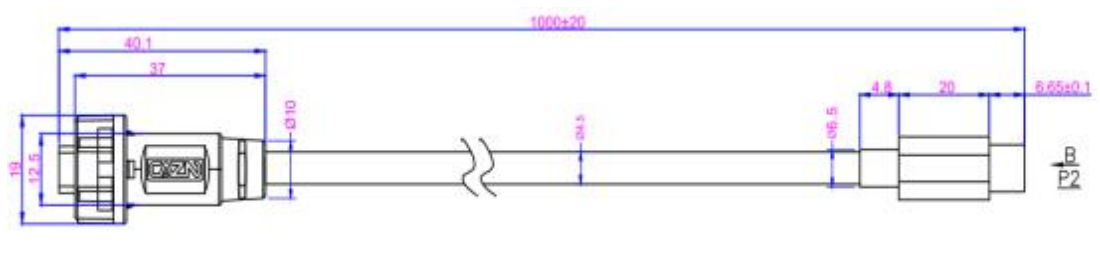
1.5.6 USB Port

● Female Aviation Connector

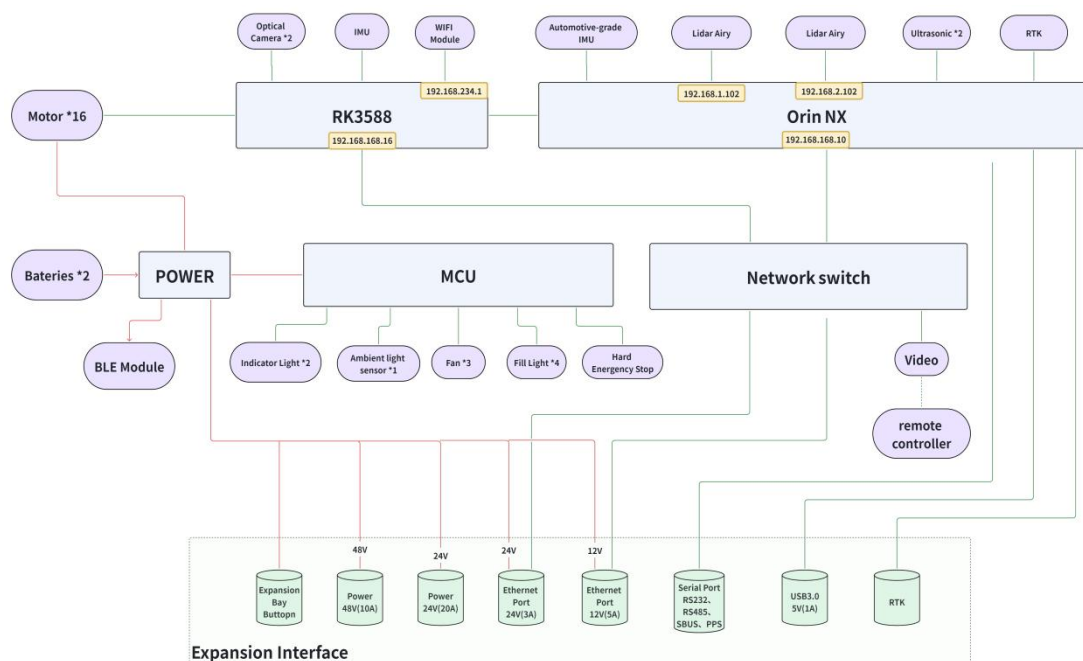


Connector Name	USB 3.0 + 5V (* 2)
Board-End Connector Model	E10B- FT3-PPF
Pin Assignment	None

● Mating Cable Harness



1.6 D1 Max Hardware Architecture Diagram



Note: The main control board is an RK3588, dedicated to quadruped motion control. Development or deployment on this board is not supported.

The compute board is an Orin NX, dedicated to perception and control of the quadruped robot. Development and deployment may be performed on this compute board. Ensure proper backup management, as flashing or Over-The-Air (OTA) upgrades may overwrite the system.

2. SDK Development Guide

2.1 SDK Introduction

- This document describes the Software Development Kit (SDK) for the D1 Max, providing detailed information on its core service interfaces, including high-level command set interfaces, Inertial Measurement Unit (IMU) inertial navigation data interfaces, and motor status data interfaces.
- Each chapter describes the invocation methods for different service interfaces, the standard format definitions for data input/output, and provides functional development examples.
- Interface definitions are subject to change based on production development status. Please refer to the latest version of the interface definition document.
- The SDK communicates with the quadruped robot body program via a specific protocol. Therefore, different SDK versions may have compatibility differences with the robot body software. Please select the SDK version that corresponds to the device version. If conditions permit, it is recommended to update the quadruped robot firmware to the latest version for better compatibility.
- When controlled via the SDK interfaces, all states are designed and specified by the secondary developer. The robot body motion controller does not perform additional intervention or state transitions.
- When running the SDK, please ensure that the system has sufficient resources; otherwise, the motion control module may fail.

2.2 SDK Software Service Interface List

No.	Interface Name	Function	SDK Version
1	High-Level Command Set Interface	Implements high-level command set of the robot, including forward/backward and turning motion, prone posture, rotation, and standing on all fours	V0.0.2
2	Video Stream Push Data Interface	Streams the robot body camera's video feed to the client	V0.0.2

3	Device Status Service Interface	Queries and retrieves current device status and system resource usage information	V0.0.3
4	Inertial Navigation Status Data Interface	Retrieves raw IMU data	V0.0.4
5	Ultrasonic Radar Interface	Implements autonomous obstacle-avoidance functionality	v0.0.5
6	Remote Controller & SDK Control Authority Switching Function and SDK Interface	Implements control authority switching functionality	v0.0.6
7	RTK Module Interface	Implements autonomous positioning functionality	v0.0.6
8	Autonomous Docking & Recharging Function and API Interface (Optional)	Implements autonomous charging functionality	v0.0.9
9	SLAM Mapping & Navigation Function and API Interface (Optional)	Implements autonomous navigation functionality	v0.1.0

2.3 Safety Protection Mechanisms

2.3.1 Hard Emergency Stop

- Trigger Method: Press the emergency stop button on the robot body
- Effect: The robot is slowly lowered to the ground, and the red indicator light illuminates
- Indicator Light Status
 - When pressed: Red light on
 - After release: Green light on
- Recovery: Once the system status is reset, the system automatically recovers (provided no other faults exist)

2.3.2 Soft Emergency Stop

- **Trigger Method:** Via the remote controller or via the SDK interface
- **Effect:** The robot stops immediately and no longer responds to any control commands
- **Recovery:** Normal operation can resume after releasing the soft emergency stop

Starting from SDK version V0.0.6, the system allows users to switch between remote controller mode and SDK mode. Before triggering the soft emergency stop via the remote controller during SDK control, you must open the App and enable control authority. Once acquired, you can trigger a soft emergency stop at any time using the red emergency stop button at the upper right of the remote controller. After the soft emergency stop is triggered, the robot displays a red light, maintains its current posture, and does not respond to any control commands. Press the emergency stop button on the remote controller again to resume operation.

2.3.3 Internal Exception Protection (Automatic Trigger)

The following conditions will trigger the internal protection mechanism, causing the robot to automatically stop and lower itself to the ground slowly:

- Remote control signal disconnection (including network congestion); reconnection via the APP will restore operation
- Battery present but charge level < 10%
- Joint fault
- IMU communication interruption

2.4 Environment Dependencies

- **Operating System:** Ubuntu 22.04
- **Build Tools:** CMake 3.8+, GCC 11+
- **Dependencies:** Boost 1.74+
- **Programming Language:** C++

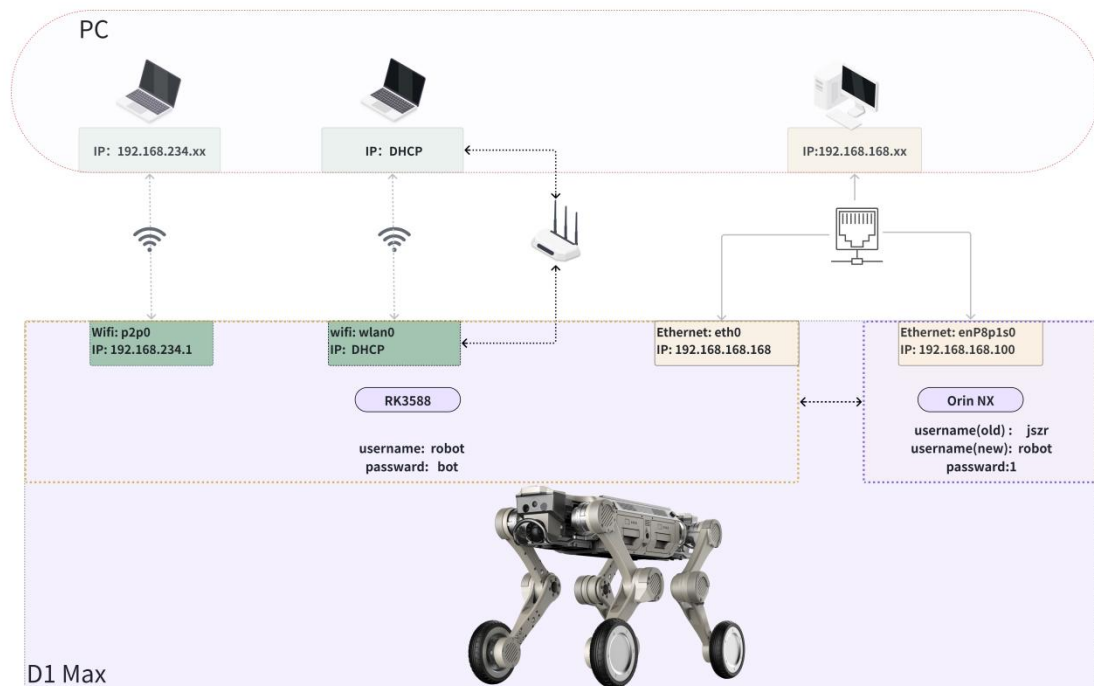
2.5 Device Login

Network Connection

- **Wi-Fi Connection**
 - Wi-Fi Name: XG2WIFI_XXXXXX
 - Wi-Fi Password: 12345678

● Wired Connection

- Use an Ethernet cable to connect the robot's expansion Ethernet port to the control PC
- Set the control PC's Ethernet IP to 192.168.168.x (cannot be set to 192.168.168.168 or 192.168.168.100)



SSH Login

Bash

#Connect via Wi-Fi

ssh robot@192.168.234.1 #RK3588, password: bot

#Connect via Ethernet cable

ssh robot@192.168.168.168 #RK3588, password: bot

ssh robot@192.168.168.100 #Orin NX, password: 1

2.6 Obtaining the SDK

Method 1: SSH into the robot body, query the current firmware version information, and send a screenshot of the quadruped robot's firmware version number to the after-sales technical support team to obtain the corresponding SDK package and example code. The query method is as follows:

```
bash

ssh robot@192.168.234.1      #password: bot
cat /opt/release/version.yaml #View RK-corresponding firmware version

ssh robot@192.168.168.100    #password: 1
cat /opt/release/version.yaml #View NX-corresponding firmware version
```

Method 2: Obtain via the following GitHub repository

```
bash

git clone https://github.com/AgibotTech/Agibot_D1_Max.git
```

2.7 Demo Execution

On the local PC, navigate into the SDK directory

```
Bash

# Enter the C++ demo directory
cd example

# Create a build directory
mkdir build && cd build

# Configure and build the project
cmake ..
make -j6

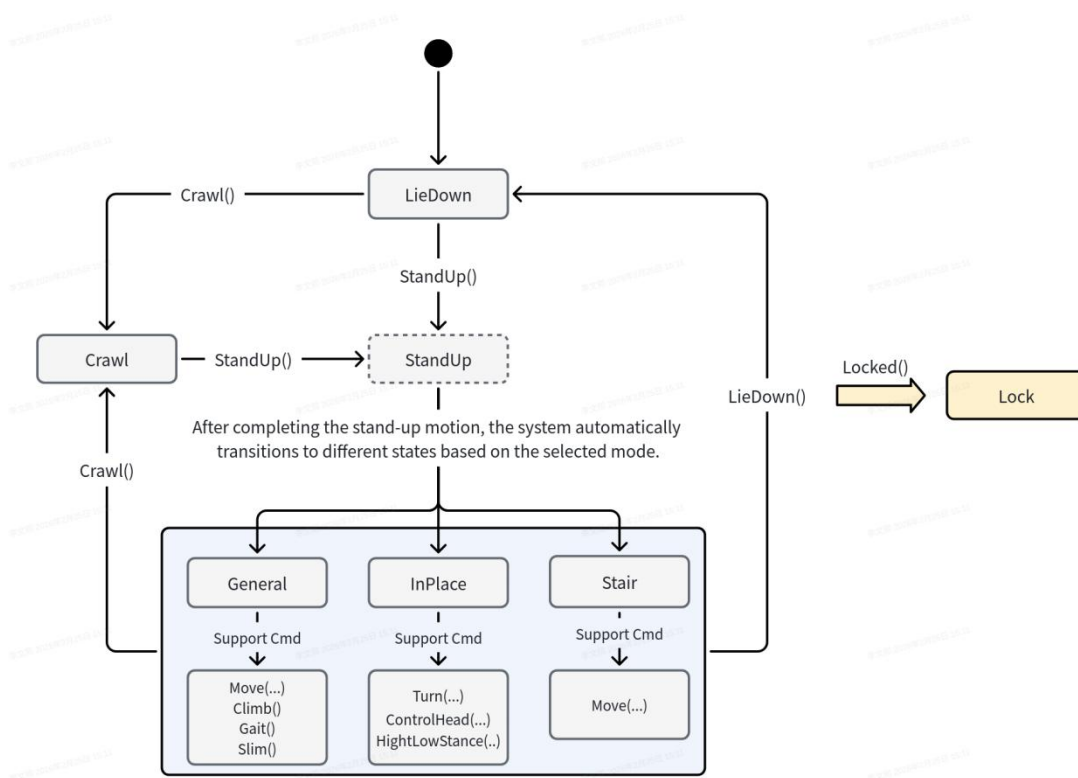
# Run the demo executable
./example_data 192.168.168.168 8081 # Wired connection
./example_data 192.168.234.1 8081  # Wireless connection
```

Before running the demo executable, ensure that the remote controller app

has been closed in the background

2.8 SDK Control Instructions

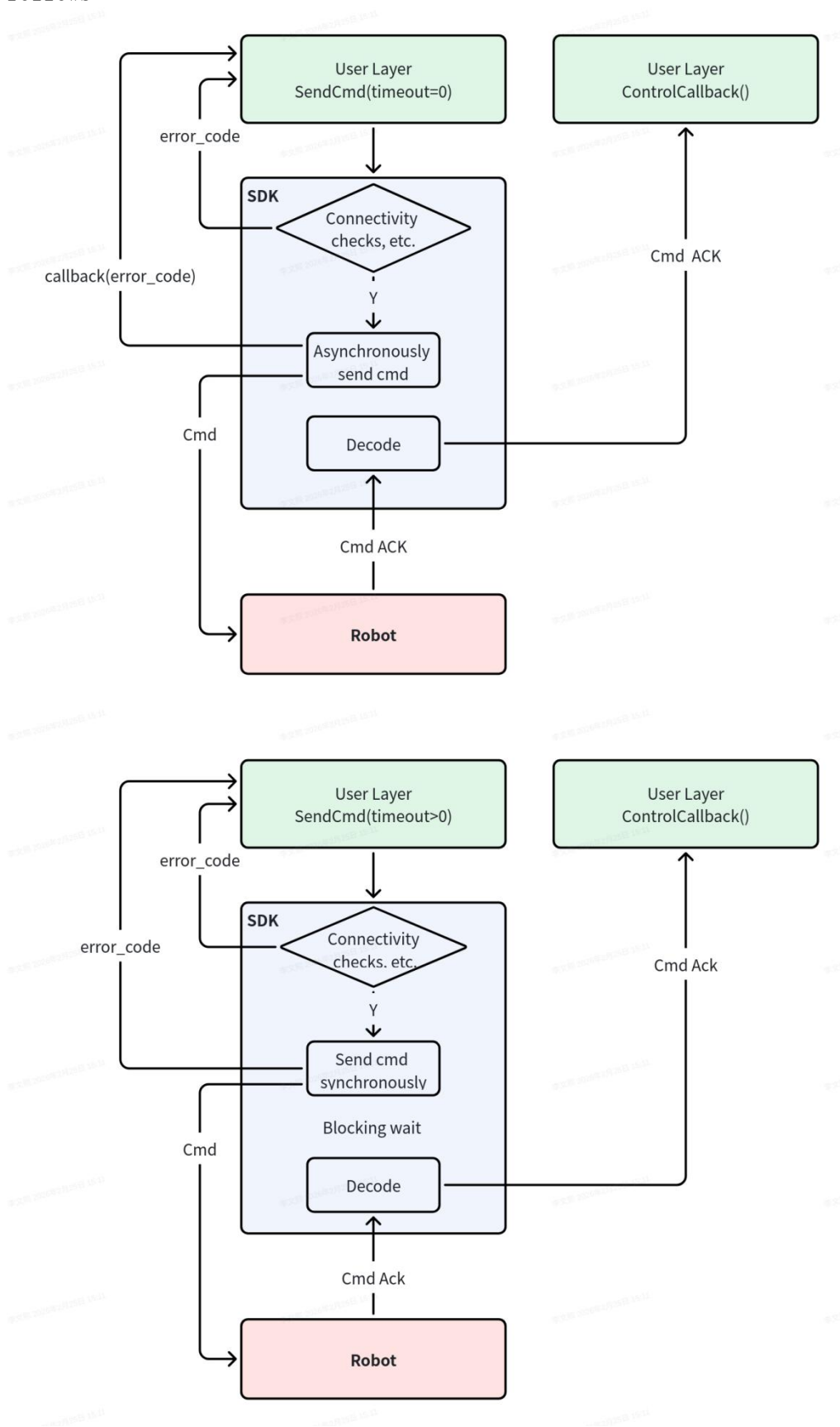
- SDK commands must be issued following the state transition logic illustrated below. Failure to do so may result in abnormal conditions such as the robot falling over, malfunctioning, or becoming unresponsive.



Notes:

1. When a "Stand" command is issued, upon completion of the standing motion, the robot will automatically transition to the general state or the in-place state depending on the current mode
2. Switching from the locked state to any other state will release the lock

- The SDK command issuance workflow is as follows



2.9 Joint Naming Convention

For ease of identification and debugging, this device adopts the following naming conventions:

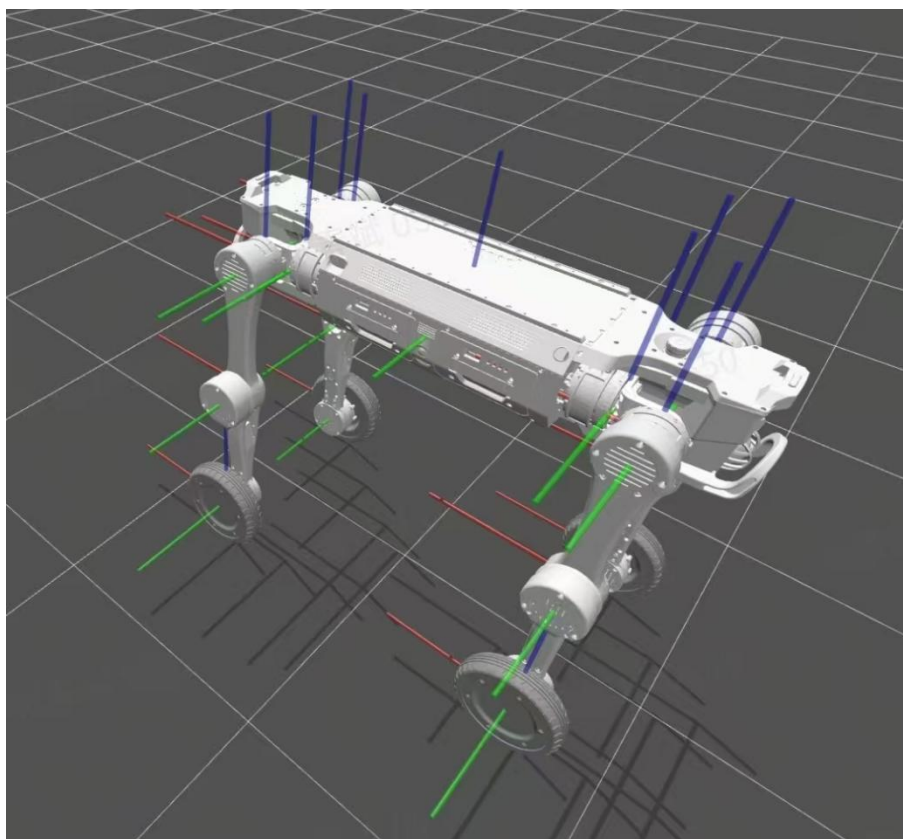
- Direction Abbreviations
 - f = front
 - b = back
 - l = left
 - r = right
- Number Meanings
 - 1 = Hip Roll
 - 2 = Hip Pitch
 - 3 = Knee Pitch
 - 4 = Foot
- Example Names
 - fr1_hip_roll: Right front leg, joint No. 1 (hip roll)
 - bl3_knee_pitch: Left back leg, joint No. 3 (knee pitch)

front_left_leg	#Left front leg
front_right_leg	#Right front leg
back_left_leg	#Left back leg
back_right_leg	#Right back leg
fl1_hip_roll	#Left front, No. 1
fl2_hip_pitch	#Left front, No. 2
fl3_knee_pitch	#Left front, No. 3
fl4_foot	#Left front No. 4
fr1_hip_roll	#Right front, No. 1
fr2_hip_pitch	#Right front, No. 2
fr3_knee_pitch	#Right front, No. 3
fr4_foot	#Right front, No. 4
bl1_hip_roll	#Left back, No. 1
bl2_hip_pitch	#Left back, No. 2
bl3_knee_pitch	#Left back, No. 3
bl4_foot	#Left back, No. 4
br1_hip_roll	#Right back, No. 1
br2_hip_pitch	#Right back, No. 2
br3_knee_pitch	#Right back, No. 3
br4_foot	#Right back, No. 4

2.10 Sensor Device Coordinate Definitions

Sensor Name	Sensor Coordinates (relative to body BASE origin) (mm)			Remarks
	x	y	z	
IMU_ICM42688 (for motion control)	0	0	36.2	
IMU_LUA300C (automotive-grade)	0	0	56.9	
Front LiDAR AIRY	404.3	0	-37.7	See datasheet for details
Front Camera IMX415	412.3	0	37.8	See datasheet for details
Front-Right Fill Light	420.3	-30.9	28.3	120° illumination range
Front-Left Fill Light	420.3	30.9	28.3	120° illumination range
Rear LiDAR AIRY	-404.3	0	-37.7	See datasheet for details
Rear Camera IMX415	-412.3	0	37.8	See datasheet for details
Rear-Right Fill Light	-420.3	-30.9	28.3	120° illumination range
Rear-Left Fill Light	-420.3	30.9	28.3	120° illumination range
Front-Right Ultrasonic Radar RADAR A22	179.2	-100.2	50	See datasheet for details
Front-Left Ultrasonic Radar RADAR A22	179.2	100.2	50	See datasheet for details

When all joints are at zero degrees, the coordinate systems are as shown in the figure below. Red represents the X-axis, green represents the Y-axis, and blue represents the Z-axis. Please refer to the figure below for the joint rotation axes and positive rotation directions.



3. SDK Function Introduction

- `SDKClient` is the core communication class of the Agibot D1 Max robot SDK, used to establish and terminate connections with the robot, send control commands, and configure data reporting. Developers can use this class to implement various functions including motion control, mode switching, emergency stops, and safety locking.
- This class supports both **blocking** and **non-blocking** invocation modes.
 - **Blocking Mode:** The function waits for the operation to complete or timeout before returning a result.
 - **Non-blocking Mode:** The function returns immediately and notifies the result via a callback function.
- SDK Directory Structure

SDK directory structure

```

├── example/          # Demo application (C++)
├── docs/             # Detailed documentation files
├── include/          # SDK C++ header files
└── lib/              # Precompiled libraries (.so) for different
architectures
  
```

- The current version provides the HighLevel control interfaces. Additional SDK interfaces will be made available in future releases.

3.1 Introduction to HighLevel API Functions

3.1.1 Error Code Mechanism

- Error Codes
 - `error_code`
 - ◆ **Success:** `ec.value() == 0`
 - ◆ **Failure:** `ec.value() != 0`
 - `error_msg`: Detailed error description can be obtained via `ec.message()`.
 - Example:

C++

```

std::error_code ec = sdk.Connect("192.168.234.1", 8081);
if (ec) {
    std::cerr << "Connection failed: " << ec.message() << std::endl;
}
  
```

```
} else {  
    std::cout << "Connection successful" << std::endl;  
}
```

- Error Code Definitions

All error codes follow the C++ standard definitions.

Commonly used error codes in the SDK are listed below

Enum Value	Integer Value	Meaning
success	0	Success
invalid argument	22	Invalid argument
already connected	106	Already connected
not connected	107	Not connected
timeout	110	Operation timeout
connection refused	111	Connection refused
connection already in progress	114	Connection already in progress
operation in progress	115	Operation in progress
operation canceled	125	Operation canceled

● Example:

C++

```
std::error_code ec = sdk.Connect("192.168.234.1", 8081);
if (ec) {
    std::cerr << "Connection failed: " << ec.message() << std::endl;
} else {
    std::cout << "Connection successful" << std::endl;
}
```

3.1.2 Type Definitions

Type Name	Definition	Description
ConnectHandler	<code>std::function<void(const std::error_code&)></code>	Callback function invoked upon connection completion
DisConnectHandler	<code>std::function<void(const std::error_code&)></code>	Callback function invoked upon disconnection completion
ErrorHandler	<code>std::function<void(const std::error_code&)></code>	Callback function for internal SDK errors

3.1.3 Constructor & Destructor

● Constructor

Function Name	<code>SDKClient(ErrorHandler error_callback = [] (const std::error_code&) {})</code>
Function Overview	Constructs an SDK client object
Parameters	error_callback: Callback for internal SDK communication exceptions
Return Value	
Remarks	

● Destructor

Function Name	<code>~SDKClient()</code>
Function Overview	Destructor, releases resources
Parameters	
Return Value	
Remarks	

3.1.4 Connection Management Interfaces

● Establish Connection

Function Name	<code>std::error_code Connect(std::string ip, std::string port, bool block = false, ConnectHandler handler = {})</code>
Function Overview	Establishes a communication connection with the robot
Parameters	ip: Target IP address port: Target port number block: Whether to block: true = blocking mode; false = non-blocking handler: Callback function upon connection completion in non-blocking mode
Return Value	Success: returns <code>std::error_code()</code> Failure: returns <code>std::error_code</code> with error information
Remarks	When communication setup fails, a failure log is output to the terminal

● Disconnect

Function Name	std::error_code Disconnect(bool block = false, DisconnectHandler handler = {})
Function Overview	Disconnects from the robot
Parameters	block: Whether to block: true = blocking mode; false = non-blocking handler: Callback function upon disconnection completion in non-blocking mode
Return Value	Success: returns std::error_code() Failure: returns std::error_code with error information
Remarks	When communication setup fails, a failure log is output to the terminal

● Get Connection Status

Function Name	bool IsConnected() const
Function Overview	Checks whether the robot is currently connected
Parameters	None
Return Value	true: Connected false: Not connected
Remarks	

3.1.5 Callback Configuration Functions

Set Control Response Callback

Function Name	void SetControlCallback(std::shared_ptr<IControlCallback>& control_callback)
Function Overview	Sets the control response callback function, used to receive command acknowledgments in non-blocking mode
Callback Function	See the detailed description of the IControlCallback class
Remarks	

IControlCallback Class

- **Description:** Control command callback interface. This interface is used to receive callbacks verifying that the robot has "received" the control command.

Only used in **non-blocking mode**

- **Definition**

```
C++
IControlCallback
class ROBOT_EXPORT_API IControlCallback {
public:
    virtual void OnSoftEmergencyStop(bool on) {}
    virtual void OnStandUp() {}
    virtual void OnLieDown() {}
    virtual void OnCrawl() {}
    virtual void OnClimb() {}
    virtual void OnSlim() {}
    virtual void OnGait() {}
    virtual void OnReverseHeadTail() {}
    virtual void OnMode(int mode) {}
    virtual void OnSpeed(int speed_level) {}
    virtual void OnLocked() {}
    virtual void OnFrontLight(bool on) {}
    virtual void OnBackLight(bool on) {}
    virtual void OnAutoModeLight(bool on) {}
    virtual void OnLuxConfig(bool on) {}
    virtual void OnImuConfig(int freq) {}
    virtual void OnMcConfig(bool on) {}
    virtual void OnTakeControlAck(const TakeControlAck& ack) {}
    virtual void OnReleaseControlAck(const ReleaseControlAck& ack) {}
    virtual ~IControlCallback() = default;
};
```

Callback List

Callback Function	Description	Parameters
<code>OnSoftEmergencyStop(bool on)</code>	Emergency stop command received	<code>on = true</code> Enable emergency stop <code>on = false</code> Disable emergency stop

<code>OnStandUp()</code>	Stand-up command received	—
<code>OnLieDown()</code>	Lie-down command received	—
<code>OnCrawl()</code>	Crawl command received	—
<code>OnClimb()</code>	Climb high-platform command received	
<code>OnSlim()</code>	Slim command received	
<code>OnGait()</code>	Gait command received	
<code>OnReverseHeadTail()</code>	Reverse head/tail command received	—
<code>OnMode(int mode)</code>	Mode switch command received	Mode number
<code>OnSpeed(int speed_level)</code>	Speed switch command received	Speed level
<code>OnLocked()</code>	Lock command received	—
<code>OnFrontLight(bool on)</code>	Front fill-light command received	<code>true</code> On, <code>false</code> Off
<code>OnBackLight(bool on)</code>	Rear fill-light command received	<code>true</code> On, <code>false</code> Off
<code>OnLuxConfig(bool on)</code>	Light intensity configuration command received	<code>true</code> On, <code>false</code> Off
<code>OnImuConfig(int freq)</code>	IMU configuration command received	Frequency value
<code>OnMcConfig(bool on)</code>	Motion control data configuration	<code>true</code> On, <code>false</code> Off

	command received	
OnTakeControlAck(const TakeControlAck& ack)	Take control command acknowledgment received	Acknowledgment information
OnReleaseControlAck(const ReleaseControlAck& ack)	Release control command acknowledgment received	Acknowledgment information

● Example:

```
C++

#include "robot_sdk/sdk_callback.hpp"

class MyDataCallback : public robot_sdk::IDataCallback {
public:
    void OnImuData(const ImuData& data) override {
        // Lightweight processing, e.g., caching
    }
};

class MyControlCallback : public robot_sdk::IControlCallback {
public:
    void OnStandUp() override {
        // Stand-up command acknowledgment received
    }
};
```

● Precautions

Do not execute time-consuming tasks inside callbacks.

If time-consuming processing is required, copy the data inside the callback and dispatch it to a separate thread.

Set Data Reception Callback Function

Function Name	void SetDataCallback(std::shared_ptr<IDataCallback>& data_callback)
Function Overview	Sets the data reception callback function, triggered when robot data is reported
Callback Function	See the detailed description of the IDataCallback class
Remarks	

IDataCallback Class

- Description: Data reporting callback interface. Users must implement this interface as needed to receive data from the robot system

Callback functions must be **lightweight** and must not perform time-consuming operations.

Only fast operations such as data copying and validation are recommended inside callbacks.

If time-consuming tasks are required (e.g., database writes, file I/O, complex computations, network transmission), **copy the data within the callback and dispatch it to a separate thread or thread pool** for processing.

- Definition

```
C++
IDataCallback
class ROBOT_EXPORT_API IDataCallback {
public:
    virtual void OnImuData(const ImuData& data) {}
    virtual void OnLuxData(const LuxData& data) {}
    virtual void OnMcData(const MotionData& data) {}
    virtual void OnRobotStateData(const RobotState& data) {}
    virtual void OnFaultData(const FaultData& data) {}
    virtual void OnControlLost(const ControlLostInfo& info) {}
    virtual void OnControlAvailable(const ControlAvailableInfo& info) {}

    virtual ~IDataCallback() = default;
};
```

- Callback List

Callback Function	Description	Parameters	Invocation Frequency
OnImuData	IMU data callback	<code>const ImuData& data:</code> IMU data structure	Reported at configured frequency
OnLuxData	Light intensity data callback	<code>const LuxData& data:</code> Light intensity data structure	Reported at fixed 1Hz frequency after configuration
OnRobotStateDa	Robot status	<code>const RobotState&</code>	Actively reported

ta	data callback	data: Robot status structure	at 1 Hz
OnFaultData	Fault information data callback	const FaultData& data: Fault data collection	Actively reported when a fault occurs
OnMcData	Robot motion data callback	const MotionData& data: Robot motion structure	Reported at fixed 100Hz frequency after configuration
OnControlLost	Control lost callback	const ControlLostInfo& info: Control lost information	
OnControlAvail able	Control available callback	const ControlAvailableInfo & info: Control available information	

3.1.6 Control Interfaces

3.1.6.1 Emergency Stop Control (SoftEmergencyStop)

Function Name	<code>std::error_code SoftEmergencyStop(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code& std::size_t) {})</code>
Function Overview	Emergency stop control. After a soft emergency stop, the system will not respond to other motion commands and velocity is set to 0
Parameters	on: Emergency stop state: true = enable; false = disable timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	- Asynchronous mode: Function return indicates command has been sent; result is notified via callback function - Synchronous mode: Function return indicates command send result Once the emergency stop is enabled, the system will not respond to other motion commands; velocity is 0

3.1.6.2 Stand Up (StandUp)

Function Name	<code>std::error_code StandUp(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function	Robot stand-up motion

Overview	
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	After calling StandUp, the robot reports the MOTION_STATUS_STAND_UP status during the stand-up process. Upon completion, the robot automatically transitions to MOTION_STATUS_GENERAL or MOTION_STATUS_IN_PLACE depending on the current mode.

3.1.6.3 Lie Down (LieDown)

Function Name	std::error_code LieDown(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})
Function Overview	Robot lie-down motion
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.6.4 Crawl (Crawl)

Function Name	<code>std::error_code Crawl(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Robot crawl motion
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	

3.1.6.5 Climb (Climb)

Function Name	<code>std::error_code Climb(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Robot high-platform climbing motion
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	Used only in "General Mode"

3.1.6.6 Slim (Slim)

Function Name	<code>std::error_code Slim(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Robot slim posture
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	Used only in "General Mode"

3.1.6.7 Gait (Gait)

Function Name	<code>std::error_code Gait(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Robot gait posture
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	Used only in "General Mode"

3.1.6.8 Barrier-Crossing Posture (DSB)

Function Name	<code>std::error_code DSB(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Barrier-Crossing posture
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	Used only in "General Mode"

3.1.6.9 Reverse Head/Tail Direction(ReverseHeadTail)

Function Name	<code>std::error_code ReverseHeadTail(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Reverses the robot's head and tail direction
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.6.10 Set Current Operating Mode (SetMode)

Function Name	<code>std::error_code SetMode(int mode, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Sets the robot's operating mode; General Mode by default.
Parameters	mode: 1 = General Mode; 2 = In-Place Mode; 3 = Stair-Climbing Mode timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.6.11 Set Motion Speed Level (SetSpeed)

Function Name	<code>std::error_code SetSpeed(int speed_level, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Sets the speed level, primarily affecting the speed of the Move command in General Mode. Low speed by default. Low Speed - Effect on Move Interface: forward_back: [-1.0, 1.0] → [-1.0m/s, +1.0m/s] left_right: [-1.0, 1.0] → [-0.5m/s, +0.5m/s] yaw: [-1.0, 1.0] → [-1.5rad/s, +1.5rad/s] ===== Medium Speed - Effect on Move Interface: forward_back: [-1.0, 1.0] → [-2.0m/s, +2.0m/s] left_right: When -1.0m/s < forward_back < 1.0m/s, [-1.0, 1.0] → [-0.5m/s, +0.5m/s] When forward_back > 1.0m/s or < -1.0m/s, limited to 0m/s yaw: When -1.0m/s < forward_back < 1.0m/s, [-1.0, 1.0] → [-1.5rad/s, +1.5rad/s] When forward_back > 1.0m/s or < -1.0m/s, [-1.0, 1.0] → [-1.0rad/s, +1.0rad/s] ===== High Speed - Effect on Move Interface: forward_back: [-1.0, 1.0] → [-3.0m/s, +3.0m/s] left_right: When -1.0m/s < forward_back < 1.0m/s, [-1.0, 1.0] → [-0.5m/s, +0.5m/s] When forward_back > 1.0m/s or < -1.0m/s, limited to 0m/s yaw: When -1.0m/s < forward_back < 1.0m/s, [-1.0, 1.0] → [-1.5rad/s, +1.5rad/s] When forward_back > 1.0m/s or < -1.0m/s, [-1.0, 1.0] → [-1.0rad/s, +1.0rad/s] When forward_back > 2.0m/s or < -2.0m/s, [-1.0, 1.0] → [-0.5rad/s, +0.5rad/s]
Parameters	speed_level: 1 = Low; 2 = Medium; 3 = High timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.6.12 Lock Robot (Locked)

Function Name	<code>std::error_code Locked(int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Locks the robot: the robot is held in a fixed position and all joints remain stationary.
Parameters	timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	Note: When a command to switch to another motion state (Stand, Crawl, Lie Down) is issued, the robot automatically unlocks and transitions to the corresponding motion state

3.1.6.13 Front Fill-Light Control (FrontLight)

Function Name	<code>std::error_code FrontLight(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Controls the front fill-lights. When set, auto mode is disabled.
Parameters	on: true = On; false = Off timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.6.14 Rear Fill-Light Control (BackLight)

Function Name	<code>std::error_code BackLight(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Controls the rear fill-lights
Parameters	on: true = On; false = Off timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	

3.1.6.15 Auto Fill-Light Mode (AutoModeLight)

Function Name	<code>std::error_code AutoModeLight(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Sets the auto fill-light mode
Parameters	on: true = On; false = Off timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	New in SDK version V0.0.6

3.1.7 Motion & Posture Control Interfaces

3.1.7.1 Move (Move)

Function Name	<code>std::error_code Move(float left_right, float forward_back, float yaw, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Controls the robot's forward/backward movement, lateral translation, and rotation. Used in General Mode for movement. Units are specified as percentages, and the speed is limited according to SpeedLevel. The latest Move command remains active for 1 second.
Parameters	left_right: Lateral translation speed, range [-1.0, 1.0]; positive = move left, negative = move right forward_back: Forward/backward speed, range [-1.0, 1.0]; positive = move forward, negative = move backward yaw: Rotation speed, range [-1.0, 1.0]; positive = turn left, negative = turn right timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled - std::errc::invalid_argument: Invalid parameter
Remarks	Asynchronous mode: Function return indicates command has been sent; result is notified via callback Synchronous mode: Function return indicates command send result It is recommended to issue the Move command at 50Hz during motion

3.1.7.2 In-Place Turning (Turn)

Function Name	<code>std::error_code Turn(int direction, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Controls the robot to perform in-place left or right turning motions
Parameters	direction: 0 = recover; 1 = left rolling; 2 = right rolling timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	Effective only in "In-Place Mode"

3.1.7.3 Head Control (ControlHead)

Function Name	<code>std::error_code ControlHead(float left_right, float up_down, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Controls the robot's "head swivel left/right" and "head tilt up/down" motions. Effective only in In-Place Mode.
Parameters	left_right: Range [-1.0, 1.0]; positive = swivel left; negative = swivel right; unit: rad/s up_down: Range [-1.0, 1.0]; positive = tilt up; negative = tilt down; unit: rad/s timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode

Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	Effective only in "In-Place Mode"

3.1.7.4 Set High/Low Stance (HighLowStance)

Function Name	std::error_code HighLowStance(int stance, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})
Function Overview	Controls the robot's high/low stance. Effective only in "In-Place Mode"
Parameters	stance: 0 = recover; 1 = high stance; 2 = low stance timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- std::errc::success: Operation successful - Common failure error codes: - std::errc::timed_out: Operation timeout - std::errc::not_connected: Not connected - std::errc::operation_canceled: Operation canceled
Remarks	

3.1.8 Data Configuration Interfaces

3.1.8.1 Configure IMU (SetImuConfig)

Function Name	<code>std::error_code SetImuConfig(int freq, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Configures the IMU reporting frequency. Disabled by default.
Parameters	freq: Range [0, 100]; 0 = disabled timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	

3.1.8.2 Configure Light Intensity (SetLuxConfig)

Function Name	<code>std::error_code SetLuxConfig(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Configures the light intensity reporting switch. Disabled by default.
Parameters	on: true = enable; false = disable timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	After configuration, light intensity data is reported at 1Hz

3.1.8.3 Configure Motion Data (SetMcConfig)

Function Name	<code>std::error_code SetMcConfig(bool on, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Configures the motion data reporting switch. Disabled by default.
Parameters	on: true = enable; false = disable timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	After configuration, motion data is reported at 50Hz.

3.1.8.4 Configure Speed Data (SetSpeed ReportConfig)

Function Name	<code>std::error_code SetSpeedReportConfig(bool on, uint32_t frequency, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {})</code>
Function Overview	Configures the speed data reporting switch. Disabled by default. Once configured, speed data will be reported.
Parameters	on: true = enable; false = disable frequency: Reporting frequency, range [1 Hz, 50 Hz] timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	After configuration, speed data is reported at the specified frequency.

3.1.8.5 Take Control (TakeControl)

Function Name	<code>std::error_code TakeControl(int timeout_ms = 0,</code>
Function Overview	Acquires control authority over the robot. If the APP is currently in control, control cannot be preempted.
Parameters	<code>timeout_ms</code> : 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) <code>handler</code> : Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	

3.1.8.6 Release Control (ReleaseControl)

Function Name	<code>std::error_code ReleaseControl(int timeout_ms = 0,</code> <code>WriteHandler handler = [] (const std::error_code&,</code> <code>std::size_t) {})</code>
Function Overview	Releases control authority over the robot
Parameters	<code>timeout_ms</code> : 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) <code>handler</code> : Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	New in SDK version v0.0.6

3.1.8.7 Update Camera Bitrate

Function Name	<code>std::error_code UpdateCameraBitrate(CameraBitrateCmd cmd, int timeout_ms = 0, WriteHandler handler = [] (const std::error_code&, std::size_t) {});</code>
Function Overview	Updates the camera bitrate
Parameters	cmd: Configuration for the specific camera's bitrate timeout_ms: 0 = asynchronous mode; >0 = synchronous mode with maximum wait time (ms) handler: Callback for command send result in asynchronous mode; not used in synchronous mode
Return Value	- <code>std::errc::success</code>: Operation successful - Common failure error codes: - <code>std::errc::timed_out</code>: Operation timeout - <code>std::errc::not_connected</code>: Not connected - <code>std::errc::operation_canceled</code>: Operation canceled
Remarks	New in SDK version v0.0.6

3.1.9 Version Information

3.1.9.1 Get SDK Version

Function Name	<code>const std::string& Version() const</code>
Function Overview	Retrieves the SDK version string
Parameters	None
Return Value	
Remarks	

3.1.9.2 Get Protocol Version

Function Name	<code>const std::string& ProtocolVersion() const</code>
Function Overview	Retrieves the protocol version string
Parameters	None
Return Value	
Remarks	

3.1.9.3 Get System Version

Function Name	<code>const std::string& SystemVersion() const</code>
Function Overview	Retrieves the system version string
Parameters	None
Return Value	
Remarks	

Example Code

```

C++

#include "robot_sdk/sdk_client.hpp"
using namespace robot_sdk;

int main() {
    SDKClient client([](const std::error_code& ec) {
        if (ec) std::cerr << "SDK Error: " << ec.message() << std::endl;
    });

    auto ec = client.Connect("192.168.234.1", "8081", true);
    if (ec) {
        std::cerr << "Connect failed: " << ec.message() << std::endl;
        return -1;
    }

    client.StandUp(2000);
    client.Move(0, 0.5, 0);
    client.Disconnect(true);
    return 0;
}

```

Note: For additional SDK data interfaces, please refer to the actual SDK files

4. Sensor Data Retrieval

4.1 Video Stream Data

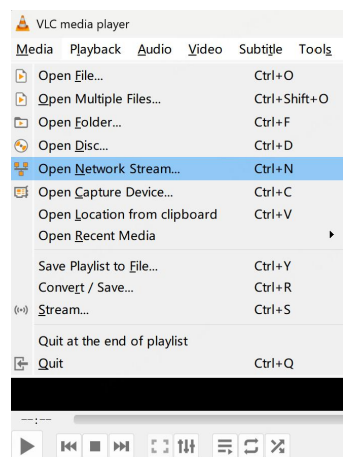
Video streams from the front and rear wide-angle cameras are pushed via GStreamer to the following URLs:

Front Camera: `rtsp://192.168.234.1:8554/front`
 Rear Camera: `rtsp://192.168.234.1:8554/back`

By default, the live video feed is pulled and displayed on the remote-controller. The streams can also be previewed and accessed using VLC or ffmpeg. Examples:

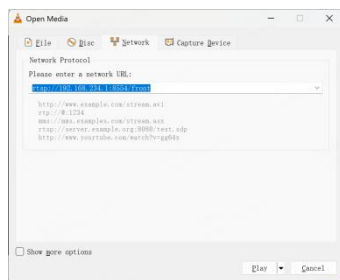
Method 1: On a Windows PC connected to the robot's Wi-Fi, use VLC to view the video stream:

Step 1: Open VLC, click "Media → Open Network Stream"

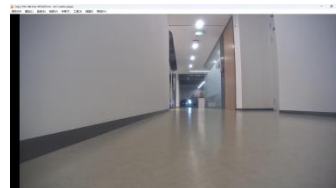


Step 2: Enter the stream URL, e.g.,

`rtsp://192.168.234.1:8554/front`



Step 3: Preview the video



Method 2: On an Ubuntu PC connected to the robot's Wi-Fi, use the following commands in the terminal to retrieve the video stream:

Bash

```
ffmpeg rtsp://192.168.234.1:8554/front
ffmpeg rtsp://192.168.234.1:8554/back
```

Note: For low-latency video stream retrieval, use the following method:

Bash

```
gst-launch-1.0 rtspsrc location=rtsp://192.168.234.1:8554/front
latency=0 ! lrtph264depay ! h264parse ! avdec_h264 ! videoconvert !
videoscale ! video/x-raw,width=1280,height=720 ! autovideosink
```

4.2 LiDAR Data

4.2.1 ROS2 Interface Definition

Topic	Msg	Field Type	Field	Field Description	Default	Range	Frequency
Front LiDAR point cloud: /front_lidar	sensor_msgs::msg::PointCloud2 (QoS: best_effort)	std_msgs/Header	header	ROS2 standard message header			10hz
Rear LiDAR point cloud: /rear_lidar		uint32	height	Vertical height of the point cloud (rows)			
		uint32	width	Horizontal width of the point cloud (columns)			
		PointField[]	fields	Field descriptors for each channel in the point cloud			
		bool	is_bigendian	Whether the data is in big-endian byte order			
		uint32	point_step	Number of bytes per point			
		uint32	row_step	Number of bytes per row of point data			
		uint8[]	data	Raw binary point cloud data as a 1D array			

		bool	is_dense	Whether the point cloud is dense (i.e., contains no invalid points)			
--	--	------	----------	---	--	--	--

Front LiDAR IP: 192.168.1.102

Rear LiDAR IP: 192.168.2.102

4.2.2 Retrieving LiDAR Data

Bash

```
ssh robot@192.168.168.100
ros2 topic echo /front_lidar --once
ros2 topic echo /rear_lidar --once
```

4.3 Ultrasonic Radar Data

4.3.1 ROS2 Interface Definition

Topic	Msg	Field Type	Field	Field Description	Default	Range	Frequency
Left ultrasonic radar: /uss_driver/uss_left/range Right ultrasonic radar: /uss_driver/uss_right/range	sensor_msgs::msg::Range	std_msgs/Header	header	ROS2 standard message header			10hz
		uint8_t	radiation_type	Sensor type. Enum values: 0 = ultrasonic, 1 = infrared	0	0-1	
		float	field_of_view	Sensor field of view (radians); typically the angle of the conical ranging area	0	Currently fixed at 0	
		float	min_range	Sensor minimum	0	Currently fixed at	

				measurable distance (meters)	0	
		float	max_range	Sensor maximum measurable distance (meters)	0	Currently fixed at 0
		float	range	Current actual distance measured (meters)	0	0 - 4m

4.3.2 Retrieving Ultrasonic Radar Data

Bash

```
ros2 topic echo /uss_driver/uss_left/range --once
ros2 topic echo /uss_driver/uss_right/range --once
```

4.4 IMU Data

The system currently has three IMUs (2 LiDARs + 1 automotive-grade). Data is published via ROS2 topics using the standard `sensor_msgs::msg::Imu` message type:

Front LiDAR IMU: `/front_lidar/imu`

Rear LiDAR IMU: `/rear_lidar/imu`

Automotive-grade IMU: `/imu_driver/imu_central`

Note: The automotive-grade IMU involves hardware modifications. If IMU data cannot be retrieved, please contact after-sales support

4.5 Camera Data

Camera data originates from the RK3588 and is published under the following topics:

- Front: `/front_camera/image_compressed`
- Rear: `/rear_camera/image_compressed`

5. Debugging Instructions

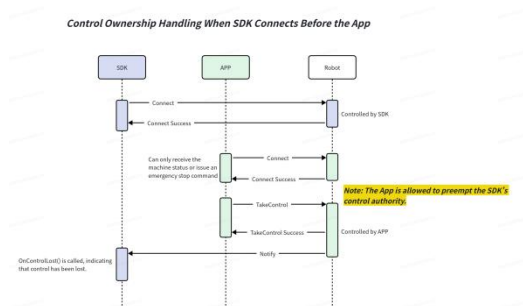
5.1 Control Authority Switching Instructions

Function Overview: The control authority switching function is primarily used during SDK control phases to handle emergency situations or scenarios requiring manual intervention.

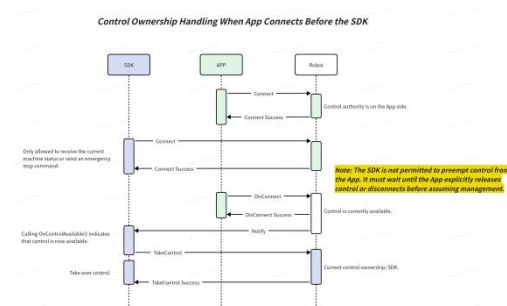
Control Principles:

- Simultaneous control by both the remote controller and the SDK is not supported.
- During the SDK control phase, the APP on the remote controller can preempt control authority and trigger a soft emergency stop from the remote controller.
- During the remote controller control phase, the SDK cannot acquire control authority, but it can still issue an emergency stop command via the SDK and retrieve device status.

SDK connects first,
APP connects later



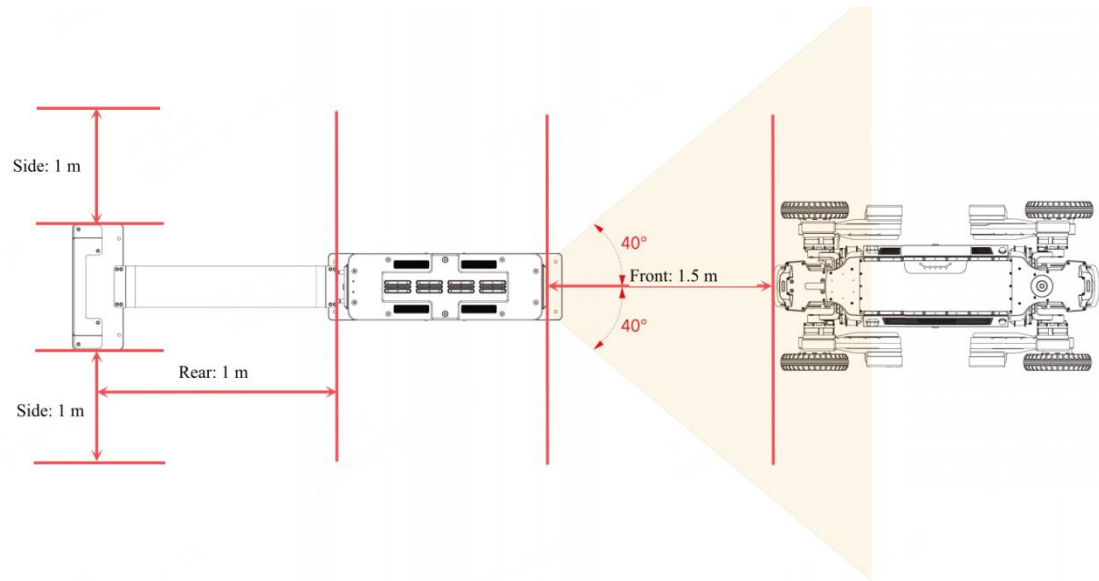
APP connects first,
SDK connects later



5.2 Autonomous Docking & Recharging Instructions

Version Requirements: RK3588 version: 0.2.4+; Orin NX firmware version: 0.4.5+

Usage Requirements: A charging station is required. For installation instructions, refer to the "AGIBOT D1 Max Autonomous Charging Product Manual". Currently, the system supports mapless docking & recharging, which requires positioning the D1 Max robot directly facing the charging station with the robot's head approximately 1.5m away from the station, ensuring that the QR code on the charging station is fully visible within the camera's field of view.



Warning Reminders

- Ensure no other QR codes are present near the charging dock to avoid interference.
- Do not obstruct the QR code after installation.
- Avoid placing the QR code in overexposed lighting conditions.

Demo Execution:

Bash

```
# Enter the C++ demo directory
cd example

# Create a build directory
mkdir build && cd build

# Configure and build the project
cmake ..
make -j6

# Run the demo executable
./control 192.168.168.168 8081 192.168.168.100 10010
```

5.3 ROS2 Multi-Machine Communication Configuration

The NX system provides a unified environment configuration file:
`/opt/runtime/env.bash`. This file is automatically sourced in `.bashrc`

```
rootfs > opt > runtime > env.bash
You, 3周前 | 1 author (You)
1  export PATH=$PATH:/opt/runtime/bin
2  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/runtime/lib
3
4  source /opt/ros/humble/setup.bash
5  source /usr/share/colcon_cd/function/colcon_cd.sh
6  source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
7  export ROS_DOMAIN_ID=24
8  export ROS_LOCALHOST_ONLY=0
9  export LC_NUMERIC="en_US.UTF-8"
10 export _colcon_cd_root=/opt/ros/humble/
11 export RMW_IMPLEMENTATION=rmw_zenoh_cpp #rmw_fastrtps_cpp
12 export ZENOH_ROUTER_CONFIG_URI=/opt/runtime/config/zenoh_router_config.json5
13 export RCUTILS_COLORIZED_OUTPUT=1
14 export RCUTILS_LOGGING_USE_STDOUT=1
15 export RCUTILS_LOGGING_BUFFERED_STREAM=1
16 export RCUTILS_CONSOLE_OUTPUT_FORMAT="[date] [{severity}] [{name}]: {message}"
17 export ROBOT_LOG_DIR=/home/jszr
18 # export ROBOT_API_HOST=0.0.0.0
```

As shown above, `ROS_DOMAIN_ID=24`. The robot currently uses the ROS2 Zenoh middleware to provide ROS2 topic services externally. Users are required to install the corresponding environment on their own laptops.

- IP Configuration

Set the user laptop IP to: 192.168.168.xxx (xxx cannot be 100, 168, or 255)

- User Laptop Environment Configuration

Ubuntu22.04

ROS2 Humble

Zenoh RMW

- Install Zenoh

Plain Text

```
sudo apt-get install ros-humble-rmw-zenoh-cpp
```

- Modify the Configuration File

Open the following file with an editor

```
/opt/ros/humble/share/rmw_zenoh_cpp/config/DEFAULT_RMW_ZENOH_ROUTER_CONFIG.json5
```

Locate the `connect/endpoints` field and add the robot NX's IP address:

```

rootfs > opt > runtime > config > zenoh_router_config.json5
4  {
29  /// For TCP/UDP links, it's possible to specify the DSCP field of the IP
30  /// E.g. tcp/192.168.0.1:7447#dscp=0x08
31  connect: {
32    /// timeout waiting for all endpoints connected (0: no retry, -1: inf)
33    /// Accepts a single value (e.g. timeout_ms: 0)
34    /// or different values for router, peer and client (e.g. timeout_ms:
35    timeout_ms: { router: -1, peer: -1, client: 0 },
36
37    /// The list of endpoints to connect to.
38    /// Accepts a single list (e.g. endpoints: ["tcp/10.10.10.10:7447", "t
39    /// or different lists for router, peer and client (e.g. endpoints: {
40    ///
41    /// See https://docs.rs/zenoh/latest/zenoh/config/struct.EndPoint.html
42    endpoints: [
43      /// "<proto>/<address>"
44      "tcp/192.168.168.100:7447",
45    ],
46
47    /// Global connect configuration,

```

Plain Text

```
"tcp/192.168.168.100:7447"
```

On the PC, open a new terminal 1 and set the middleware environment variables and ROS2 Domain ID

Bash

```

source /opt/ros/humble/setup.bash
export ROS_DOMAIN_ID=24
export RMW_IMPLEMENTATION=rmw_zenoh_cpp
ros2 run rmw_zenoh_cpp rmw_zenohd

```

Reset the ROS2 daemon

Bash

```

ros2 daemon stop
ros2 daemon start

```

With the above environment variables set, you can use RViz for visualization, record bag files, or launch your own ROS2 programs to interact with the robot's drivers.

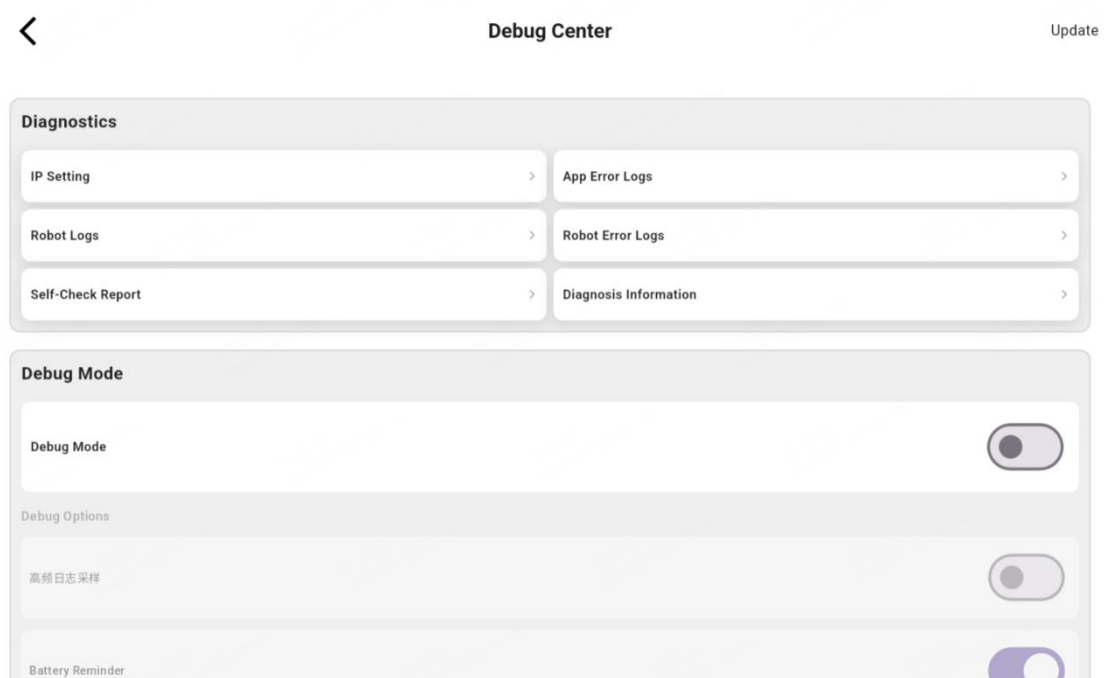
5.4 Log Viewing

- Viewing Remote Controller Error Messages

Continuously click the bottom-left corner of the homepage



Open the Settings Center to view the device exception logs



● Viewing Device Logs

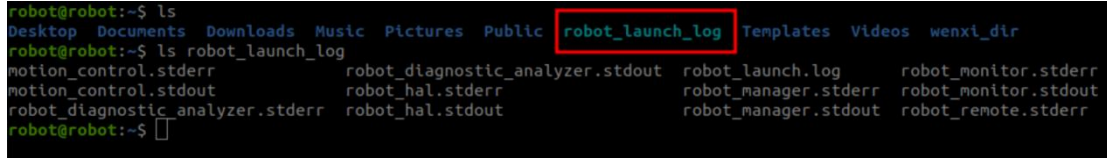
Log path for the most recent device startup (RK & NX):
/home/robot/robot_launch_log

Log path for the most recent 6 device startups (RK & NX): /userdata/log/

Log path for autonomous docking & recharging (NX): /tmp/log/alg_data

Bash

```
scp -r robot@192.168.234.1:/userdata/log. # Pull logs from the most  
recent 6 device startups  
scp -r robot@192.168.234.1:/home/robot/robot_launch_log . # Pull logs  
from the current device startup
```



```
robot@robot:~$ ls  
Desktop Documents Downloads Music Pictures Public robot_launch_log Templates Videos wenxi_dir  
robot@robot:~$ ls robot_launch_log  
motion_control.stderr      robot_diagnostic_analyzer.stdout  robot_launch.log      robot_monitor.stderr  
motion_control.stdout      robot_hal.stderr                  robot_manager.stderr  robot_monitor.stdout  
robot_diagnostic_analyzer.stderr  robot_hal.stdout                  robot_manager.stdout  robot_remote.stderr  
robot@robot:~$
```

5.5 Device Process Utility Usage

Usage Instructions: The robot-launch tool is used on the device to manage related processes, including viewing process status, starting and stopping processes, and viewing process logs.

Format Description: robot-launch [OPTIONS] [COMMAND]

```
jszr@orin-nx:~$ robot-launch help
机器人相关程序启动期管理器

Usage: robot-launch [OPTIONS] [COMMAND]

Commands:
  server      以服务器形式启动
  list        显示当前所有的 egg
  egg         显示指定 egg 的快照信息
  stop        停止多个正在运行中的 egg
  start       启动多个停止了的 egg
  restart     重启一个运行中的 egg
  remove      移除多个 egg
  collect     收集|启动一个 egg
  collects    收集|启动多个 egg
  add-service 收集|启动一个 service
  start-service 启动一个 service
  stop-service 关闭一个 service
  restart-service 重启一个 service
  service     显示一个 service 的详细信息
  remove-service 移除一个 service
  example-egg 展示 demo
  example-service 展示 service demo
  stdout-log  out log tail
  stderr-log  error log tail
  auto-complete 生成自动补全脚本
  help        Print this message or the help of the given subcommand(s)

Options:
  -i, --ip <IP>      Target IP address
  -p, --port <PORT>   Target port
  -h, --help          Print help
  -V, --version       Print version
```

Examples:

Stop a node: robot-launch stop <# id>

Start a node: robot-launch start <# id>

View node status: robot-launch egg <# id>

View help information: robot-launch help

...

5.6 Connecting to an External Wi-Fi Network

Bash

```
ssh robot@192.168.234.1    # Log in to the RK
wpa_passphrase <Wi-Fi_SSID> <Wi-Fi_password> > /tmp/wifi.conf
sudo wpa_supplicant -B -i wlan0 -c /tmp/wifi.conf
```

```
robot@robot:/tmp$ wpa_passphrase Agibot-guest Zyjqr0227 > /tmp/wifi.conf
robot@robot:/tmp$ ls
config-err-8MDr8c
hostapd.conf
launch_params_6v3090sa
launch_params_wtv4w32j
mediamtx.log
ssh-XXXXXXQCKKmM
systemd-private-82caab1629f24870a7f15bc38bfb782d-bluetooth.service-0k0QS8
systemd-private-82caab1629f24870a7f15bc38bfb782d-chrony.service-H9T3Kw
systemd-private-82caab1629f24870a7f15bc38bfb782d-systemd-logind.service-H5D6J2
systemd-private-82caab1629f24870a7f15bc38bfb782d-systemd-resolved.service-AnAny0
systemd-private-82caab1629f24870a7f15bc38bfb782d-upower.service-MDEhbU
test
wifi.conf
robot@robot:/tmp$ cat wifi.conf
network={
    ssid="Agibot-guest"
    #psk="Zyjqr0227"
    psk=f6d0acd84e1b5791737742f904a116fdeffa7b2d2cbe067c6840f35a982d9888
}
robot@robot:/tmp$ sudo wpa_supplicant -B -i wlan0 -c /tmp/wifi.conf
Successfully initialized wpa_supplicant
robot@robot:/tmp$ ping www.baidu.com
PING www.a.shifen.com (153.3.238.28) 56(84) bytes of data.
64 bytes from 153.3.238.28 (153.3.238.28): icmp_seq=1 ttl=53 time=9.31 ms
64 bytes from 153.3.238.28 (153.3.238.28): icmp_seq=2 ttl=53 time=10.0 ms
^C
--- www.a.shifen.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 9.313/9.667/10.021/0.354 ms
```

5.7 Accessing the Wired Ethernet Port While Connected to Wi-Fi

When connected to the device's AP hotspot (XG2WIFI_XXXXX), add the following route to enable access to the device's wired Ethernet port while connected to Wi-Fi:

Windows System:

Bash

```
route -p ADD 192.168.168.0 MASK 255.255.255.0 192.168.234.1
ping 192.168.168.168
ping 192.168.168.100
```

Linux System:

Bash

```
sudo ip route add 192.168.168.0/24 via 192.168.234.1  
ping 192.168.168.168  
ping 192.168.168.100
```

Glossary		
Abbreviation	Full Name	Description
API	Application Programming Interface	Application programming interface
Bash	Bourne Again SHell	The default command-line interpreter on Linux systems
CMake	Cross-platform Make	A cross-platform build tool used to compile C++ sample programs in the SDK
DFOV	Diagonal Field of View	The diagonal angular extent of a wide-angle camera's field of view
HFOV	Horizontal Field of View	The horizontal angular extent of a wide-angle camera's field of view
VFOV	Vertical Field of View	The vertical angular extent of a wide-angle camera's field of view
ffmpeg	FFmpeg player	Can be used to preview RTSP video streams
GCC	GNU Compiler Collection	SDK build dependency: GCC 11+ required
GNSS	Global Navigation Satellite System	Global navigation satellite system
RTK	Real-Time Kinematic	Real-time kinematic
GStreamer	GStreamer multimedia framework	Used to stream camera video via RTSP
RTSP	Real Time Streaming Protocol	Used for streaming video from the front and rear cameras (endpoints: rtsp://192.168.234.1:8554/front and rtsp://192.168.234.1:8554/back)
IMU	Inertial Measurement Unit	Inertial measurement unit
IP	Internet Protocol	The network address assigned to the device
IP67	Ingress Protection 67	A rating indicating complete dust-tightness and resistance to temporary immersion in water
NX	NVIDIA Jetson Orin NX	The onboard compute module for perception algorithm development and deployment (SSH password: 1)
RK	RK3588 Main controller chip	The robot motion control PCB (SSH password: bot); user code deployment is prohibited on this board
OTA	Over-The-Air	
PPS	Pulse Per Second	A timing signal used for high-precision time synchronization in GPS/RTK systems
PVC	Polyvinyl Chloride	Used as insulation material for cables
RMW	ROS Middleware Implementation	This system uses Zenoh as the RMW; developers must install rmw_zenoh_cpp and set the corresponding environment variables
ROS2	Robot Operating System 2	Robot operating system
rviz2	ROS Visualization	ROS2 3D visualization debugging tool
URDF	Unified Robot Description Format	Unified robot description format: An XML-based format used in ROS2 to describe the physical structure of a robot
Zenoh	Zenoh	A lightweight communication protocol/middleware
SBUS	Serial Bus	A serial bus protocol
SCP	Secure Copy	SSH-based secure file transfer command (for retrieving logs)
SDK	Software Development Kit	Software development kit; version compatibility with the firmware must be ensured
SMA	SubMiniature version A	A type of RF coaxial connector used for connecting an external RTK antenna
SSH	Secure Shell	A secure remote login protocol
TCP	Transmission	A connection-oriented transport protocol that ensures

	Control Protocol	reliable data delivery
Tops	Trillions of Operations Per Second	Computing power unit; tera operations per second
UDP	User Datagram Protocol	User datagram protocol; a connectionless transport protocol
URL	Uniform Resource Locator	Uniform resource locator; commonly referred to as a web address
UTC	Coordinated Universal Time	Coordinated universal time; the time reference for timestamps in RTK data
VLC	VideoLAN Client	A multimedia player for previewing RTSP video streams