

ATOM Series Humanoid Robot SDK Application Development Guide

Original Instructions

Issue: V2.1

Date: 2026-08-26

Copyright © SHENZHEN DOBOT CORP LTD 2025-2026. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without the prior written consent of SHENZHEN DOBOT CORP LTD (hereinafter referred to as "DOBOT").

Disclaimer

To the maximum extent permitted by applicable law, the products described (including its hardware, software and firmware, etc.) in this document are provided **AS IS**, which may have flaws, errors or faults. DOBOT makes no warranties of any kind, express or implied, including but not limited to, merchantability, satisfaction of quality, fitness for a particular purpose and non-infringement of third party rights. In no event will DOBOT be liable for any special, incidental, consequential or indirect damages resulting from the use of our products and documents.

Before using our product, please thoroughly read and understand the contents of this document and related technical documents that are published online, to ensure that the robot is used on the premise of fully understanding the robot and related knowledge. Please use this document with technical guidance from professionals. Even if follow this document or any other related instructions, damages or losses may happen in the using process DOBOT shall not be considered as a guarantee regarding all security information contained in this document.

The user has the responsibility to make sure following the relevant practical laws and regulations of the country, in order that there is no significant danger in the use of the robot.

SHENZHEN DOBOT CORP LTD

Address: Room 1003, Building 2, Chongwen Garden, Nanshan iPark, Liuxian Blvd, Nanshan District, Shenzhen, Guangdong Province, China

Website: www.dobot-robots.com

Preface

Purpose

This document introduces the product appearance, electrical interfaces, basic operations, SDK configuration, service interfaces, and reference examples of the DOBOT ATOM series robot, making it easier for you to perform secondary development based on actual application scenarios.

Intended audience

This document is intended for:

- Customer
- Sales Engineer
- Installation and Commissioning Engineer
- Technical Support Engineer

Revision history

Date	Version	Revision history
2026/08/26	V2.1	Modify the position limit parameters.
2026/07/21	V2.0	Added robot offline trajectory recording and playback, TCP communication between robots and external devices, ATOM Learning, and SLAM navigation. Deleted the Operation Instructions section, and merged the Product Introduction and Debugging instructions sections.
2025/09/26	V1.0	The first release

Symbol conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
 DANGER	Indicates a hazard with a high level of risk which, if not avoided, could result in death or serious injury.
 WARNING	Indicates a hazard with a medium level or low level of risk which, if not avoided, could result in minor or moderate injury, robot damage.
 NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in robot damage, data loss, or unanticipated result.
 NOTE	Provides additional information to emphasize or supplement important points in the main text.

Contents

Preface	ii
1. Product Introduction	1
1.1 Camera and radar field of view	1
1.2 Joint sequence names and joint limits	3
1.3 Coordinate system	8
1.4 Debugging instructions	9
2. Application Development	11
2.1 SDK overview	11
2.2 Architecture overview	11
2.3 Development languages	12
2.4 Get SDK	12
2.5 Configuring the development environment	12
2.5.1 Environment dependencies	12
2.5.2 Installing and compiling the example programs	13
3. Software Service Interface	15
3.1 DDS communication interface	15
3.1.1 Introduction	15
3.1.2 Shared sub-IDL file	17
3.1.3 Switching to upper body control	20
3.1.4 Upper body status	21
3.1.5 Upper body control	22
3.1.6 Lower body status	22
3.1.7 Lower body control	23
3.1.8 Joint and CAN board status	23
3.1.9 Clear error	24
3.1.10 Dexterous hand status	24
3.1.11 Dexterous hand control	25
3.1.12 Wheeled humanoid chassis status	25
3.1.13 Wheeled humanoid chassis control	27
3.1.14 Emergency stop status	28
3.1.15 State machine status	28
3.1.16 AMR LiDAR data	29
3.1.17 AMR odometry data	29
3.1.18 DH-Robotics gripper command	30
3.1.19 DH-Robotics gripper status	30
3.1.20 Position control joint position control command	31
3.1.21 Position control status	32
3.2 Low-level service interface	33
3.2.1 Low-level control command interface	33
3.2.2 Low-level control example	34
3.3 High-level motion service interface	36
3.3.1 High-level whole-body control RPC interface	37
3.3.2 Error code (RpcErrorCode)	38
3.3.3 Upper-body motor control DDS interface	39
3.4 State machine ID (FsmId)	39
3.5 Audio service interface	40
4. Reference Examples	46
4.1 DDS communication examples	46
4.2 ROS2 communication examples	48
4.3 RPC examples	49
4.4 LiDAR examples	50
4.4.1 LiDAR introduction	50
4.4.2 Set parameter	51
4.4.3 Get data	52

4.5	Depth camera demo	56
4.5.1	Depth camera introduction	56
4.5.2	Get data	58
4.6	Stereo camera example	60
5.	Recording and Playback of Offline Trajectories	63
5.1	Preface	63
5.2	Compressed SDK package	63
5.3	Configuration and compilation	63
5.4	Recording offline trajectories	64
5.5	Offline trajectories playback	66
6.	TCP Communication	67
6.1	Preface	67
6.2	Connect to ATOM	67
6.3	Remote login to PC2	68
6.4	Implementing TCP communication	69
7.	ATOM Learning	76
7.1	Features	76
7.2	Project structure	76
7.3	Environment and dependencies	77
7.4	Installation process	77
7.5	Training and evaluation	78
7.5.1	Training	78
7.5.2	Evaluate and export	78
7.6	Sim2Sim workflow	78
7.7	Add new environment	79
8.	SLAM Navigation Guide	80
8.1	Introduction	80
8.2	Function interface	82
8.3	Mapping	82
8.4	Localization	83
8.4.1	Map loading	83
8.4.2	Initialize pose	83
8.4.3	Stop localization	83
8.4.4	Get robot pose	84
8.4.5	Get robot 2D pose	84
8.5	Navigation	84
8.5.1	Single-point navigation	84
8.5.2	Navigation status	84
8.5.3	Control navigation status	85
8.6	Example	85
8.7	Code example	90
9.	Common Errors and Description	92
9.1	Power supply error	92
9.2	Voltage error	92
9.3	Battery error	92
9.4	File parsing error	94
9.5	File modification error	95
9.6	Other errors	95

1. Product Introduction

• Bipedal humanoid robot

The DOBOT ATOM series is the world's first "Dexterous Manipulation + Straight-Knee Walking" embodied intelligent humanoid robot, independently developed by DOBOT. The robot features a full joint hollow routing and a 1:1 full-scale anthropomorphic arm design. It is equipped with an industrial-grade 7-DOF bionic collaborative arm with a repeat positioning accuracy of ± 0.05 mm, along with an articulated humanoid head and an optional five-finger dexterous hand. Powered by embodied intelligence manipulation model and AI edge computing, it enables "collection, training, and inference" for specific application scenarios.

ATOM is suitable for continuous and repetitive work scenarios with unfixed equipment positions, multi-specification products, high operational similarity, and the need to pass through narrow spaces and make flexible turns. Typical applications include assembly and material handling in car factories, beverage preparation in coffee shops, and nighttime medicine retrieval in chain pharmacies.

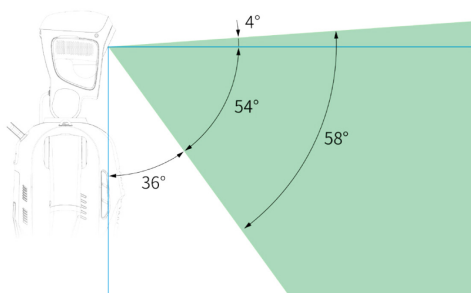
• Wheeled humanoid robot

The DOBOT ATOM-W, developed by DOBOT, is an embodied wheeled humanoid robot and a true powerhouse for industrial applications. It features a 7-degree-of-freedom (DOF) industrial-grade bionic arm (± 0.05 mm precision), a human-like articulated head, and an optional five-fingered dexterous hand. Its mobile chassis achieves a repetitive positioning accuracy of ± 10 mm, capable of autonomous stable movement and high-dynamic motion in complex scenarios. Powered by embodied intelligence manipulation model and AI edge computing, it enables "collection, training, and inference" for specific application scenarios.

1.1 Camera and radar field of view

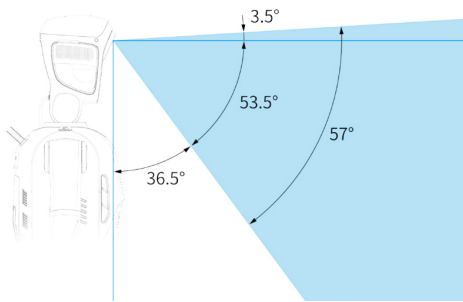
a) Head-mounted depth camera field of view

The ATOM - Max / ATOM - D / ATOM - W are equipped with a head-mounted depth camera, enabling the robot to perceive and understand its surroundings with greater accuracy. The head-mounted depth camera has a field of view as shown in the figure below:



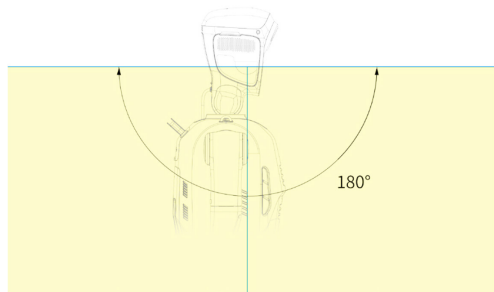
b) The field of view of the head high-definition stereo camera

The ATOM - Max / ATOM - D / ATOM - W are equipped with a head high-definition stereo camera, combined with a depth camera, enhancing the robot's spatial perception and obstacle detection capabilities. The head high-definition stereo camera has a field of view as shown in the figure below:



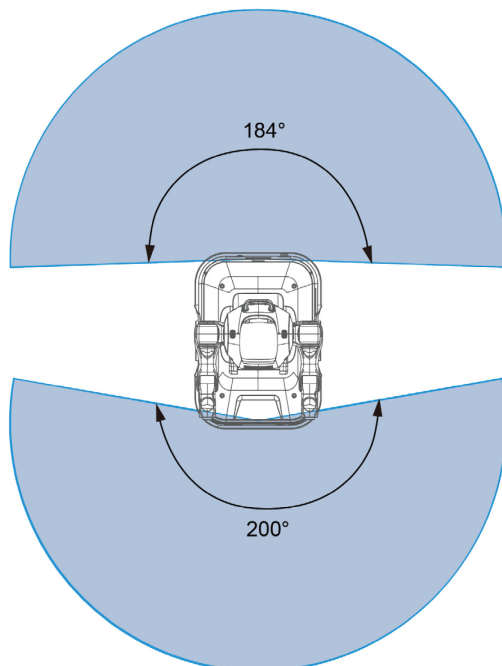
c) Field of view of the head-mounted LiDAR

The ATOM - Max is equipped with head-mounted LiDAR, which captures precise data about the environment in real time and provides the robot with environmental awareness. The head-mounted LiDAR has a field of view as shown in the figure below:

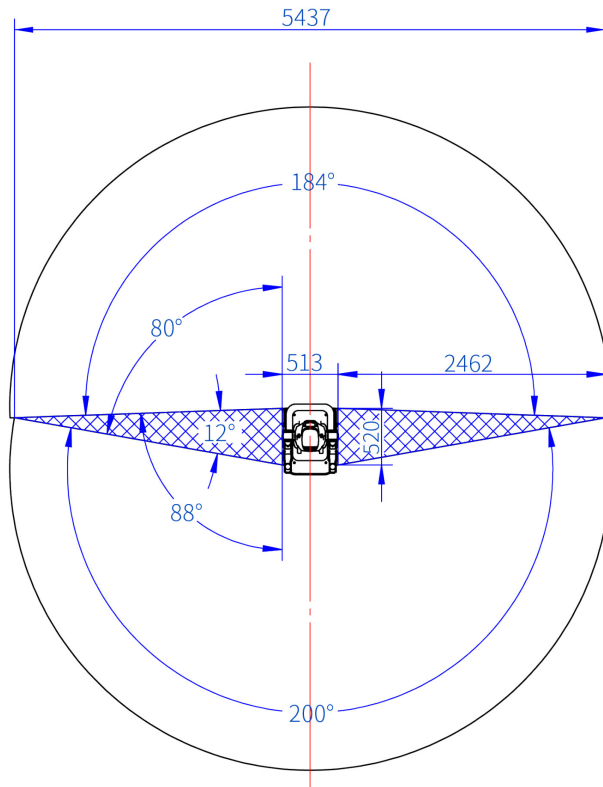


d) Field of view of the head-mounted LiDAR

The ATOM-W is equipped with a chassis-mounted LiDAR, which captures accurate environmental data in real time, and provides the robot with environmental awareness. The chassis-mounted LiDAR has a field of view as shown in the figure below:



The blind spot areas are shown in the following figure:



1.2 Joint sequence names and joint limits

Position limit (ATOM_W)

Joint index	Part	Joint	min (rad)	max (rad)
0	Lift and lower	J0	-2.5656	3.0194
1	Lift and lower	J1	-0.71	2.338
2	Lift and lower	J2	-2.338	0.9075
3-11	-	No corresponding joint	-	-
12	Waist	torso	-2.966	2.966
13	Left arm	shoulder_pitch	-2.97	2.97
14	Left arm	shoulder_roll	-0.43	3.14
15	Left arm	shoulder_yaw	-2.97	2.97
16	Left arm	elbow_pitch	-0.87	3.14
17	Left arm	elbow_roll	-2.97	2.97

Joint index	Part	Joint	min (rad)	max (rad)
18	Left arm	wrist_pitch	-1.57	1.57
19	Left arm	wrist_roll	-1.57	1.57
20	Right arm	shoulder_pitch	-2.97	2.97
21	Right arm	shoulder_roll	-3.14	0.43
22	Right arm	shoulder_yaw	-2.97	2.97
23	Right arm	elbow_pitch	-0.87	3.14
24	Right arm	elbow_roll	-2.97	2.97
25	Right arm	wrist_pitch	-1.57	1.57
26	Right arm	wrist_roll	-1.57	1.57
27	Head	head_yaw	-1.57	1.57
28	Head	head_pitch	-0.87	0.61

Position limit (ATOM_MAX, ATOM-Standard, ATOM-D)

NOTE

The ATOM-D doesn't have legs, so it has no 0-11 leg joints.

Global index	Part	Joint	min (rad)	max (rad)
0	Left leg	hip_pitch	-2.5656	3.0194
1	Left leg	hip_roll	-0.2618	1.6581
2	Left leg	hip_yaw	-2.7925	2.7925
3	Left leg	knee	0	2.4434
4	Left leg	ankle_pitch	-0.8727	0.5236
5	Left leg	ankle_roll	-0.48	0.48
6	Right leg	hip_pitch	-2.5656	3.0194
7	Right leg	hip_roll	-1.6581	0.2618
8	Right leg	hip_yaw	-2.7925	2.7925
9	Right leg	knee	-0.174	2.53
10	Right leg	ankle_pitch	-0.8727	0.5236
11	Right leg	ankle_roll	-0.48	0.48
12	Waist	torso	-2.97	2.97
13	Left arm	shoulder_pitch	-2.97	2.97
14	Left arm	shoulder_roll	-0.43	3.14
15	Left arm	shoulder_yaw	-2.97	2.97
16	Left arm	elbow_pitch	-0.87	3.14
17	Left arm	elbow_roll	-2.97	2.97
18	Left arm	wrist_pitch	-1.57	1.57
19	Left arm	wrist_roll	-1.57	1.57
20	Right arm	shoulder_pitch	-2.97	2.97
21	Right arm	shoulder_roll	-3.14	0.43
22	Right arm	shoulder_yaw	-2.97	2.97

Global index	Part	Joint	min (rad)	max (rad)
23	Right arm	elbow_pitch	-0.87	3.14
24	Right arm	elbow_roll	-2.97	2.97
25	Right arm	wrist_pitch	-1.57	1.57
26	Right arm	wrist_roll	-1.57	1.57
27	Head	head_yaw	-1.57	1.57
28	Head	head_pitch	-0.87	0.61

Speed / Torque limits (same for all models)

Rated speed = motor rated speed ÷ gear ratio × $2\pi/60$ (head gear ratio: 51; arm gear ratio: 100). Maximum speed = motor maximum speed ÷ gear ratio × $2\pi/60$ (for the lower limbs, directly use the joint maximum speed values in rad/s from the specification sheet). Rated/maximum torque correspond to the joint rated torque and joint peak torque values (N·m) in the specification sheet.

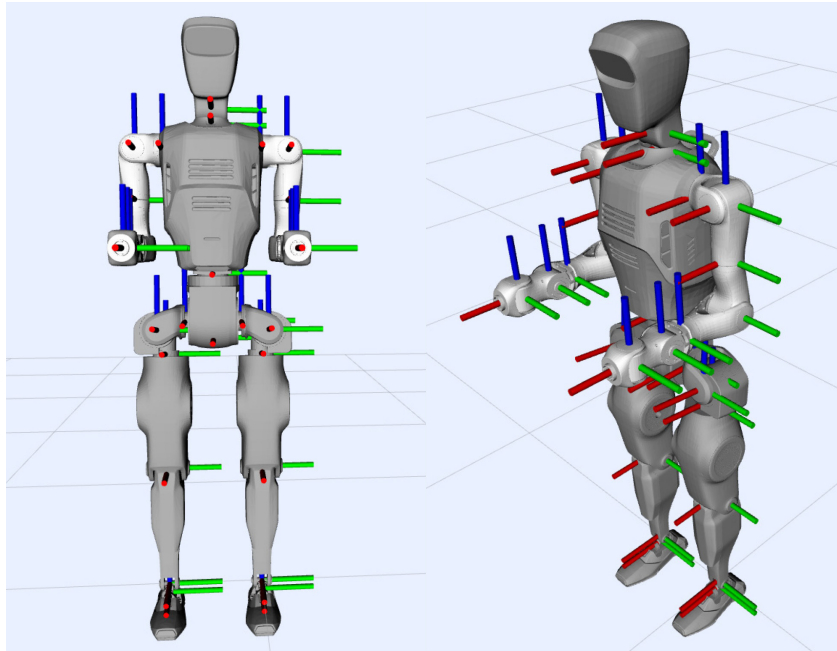
Global index	Part	Joint	Rated speed (rad/s)	Maximum speed (rad/s)	Rated torque (N·m)	Maximum torque (N·m)
0	Left leg	hip_pitch	11.9	13.8	56.1	207.76
1	Left leg	hip_roll	8.47	11.9	69.19	241.42
2	Left leg	hip_yaw	9.05	11.36	22.44	104.16
3	Left leg	knee	11.87	16.67	83.3	213.8
4	Left leg	ankle_pitch	11.13	15.71	17.68	89.9
5	Left leg	ankle_roll	11.13	15.71	17.68	89.9
6	Right leg	hip_pitch	11.9	13.8	56.1	207.76
7	Right leg	hip_roll	8.47	11.9	69.19	241.42
8	Right leg	hip_yaw	9.05	11.36	22.44	104.16
9	Right leg	knee	11.87	16.67	83.3	213.8
10	Right leg	ankle_pitch	11.13	15.71	17.68	89.9
11	Right leg	ankle_roll	11.13	15.71	17.68	89.9
12	Waist	torso	5.17	6.49	39.27	182.28
13	Left arm	shoulder_pitch	3.14	4.61	33	93
14	Left arm	shoulder_roll	3.14	4.5	22.5	70
15	Left arm	shoulder_yaw	3.14	4.5	22.5	70

Global index	Part	Joint	Rated speed (rad/s)	Maximum speed (rad/s)	Rated torque (N·m)	Maximum torque (N·m)
16	Left arm	elbow_pitch	3.14	4.5	22.5	70
17	Left arm	elbow_roll	3.14	5.13	7.5	34
18	Left arm	wrist_pitch	3.14	5.13	7.5	34
19	Left arm	wrist_roll	3.14	5.13	7.5	34
20	Right arm	shoulder_pitch	3.14	4.61	33	93
21	Right arm	shoulder_roll	3.14	4.5	22.5	70
22	Right arm	shoulder_yaw	3.14	4.5	22.5	70
23	Right arm	elbow_pitch	3.14	4.5	22.5	70
24	Right arm	elbow_roll	3.14	5.13	7.5	34
25	Right arm	wrist_pitch	3.14	5.13	7.5	34
26	Right arm	wrist_roll	3.14	5.13	7.5	34
27	Head	head_yaw	6.16	11.71	4	12
28	Head	head_pitch	6.16	11.71	4	12

1.3 Coordinate system

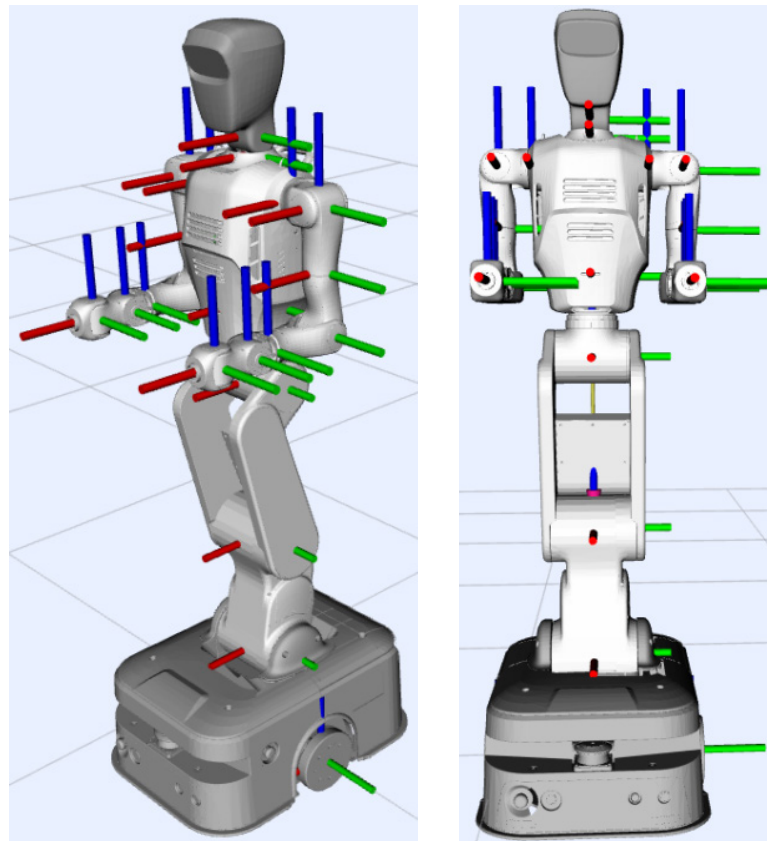
ATOM_MAX, ATOM-Standard, ATOM-D

When all joints are at zero degrees, the coordinate systems of the joints are shown in the figure below:



ATOM_W

When all joints are at zero degrees, the coordinate systems of the joints are shown in the figure below:



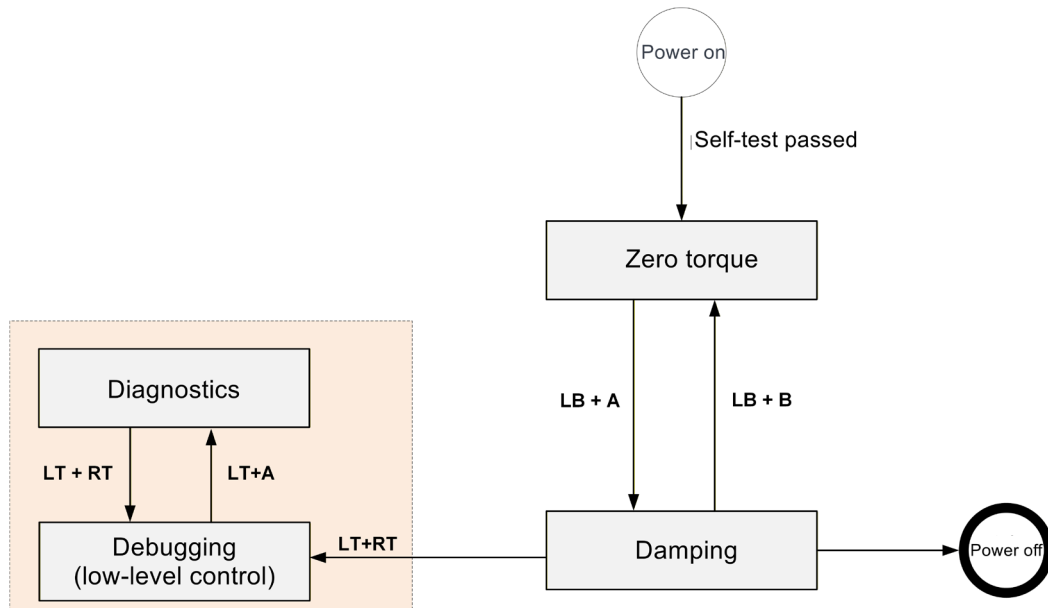
1.4 Debugging instructions

Follow the steps below to switch the robot to debugging mode for secondary development using the low-level control SDK. You can then implement custom teleoperation, deploy and control reinforcement learning models, and access real-time operating data including battery, motor, and IMU.

NOTE

The ATOM-D does not require manually switching to debugging mode. It enters debugging state automatically once connected to the debug interface.

- **Switch to debugging state**



1. For ATOM-MAX and ATOM-Standard only: Hang the robot back onto the protective frame and secure it with ropes so that both feet are suspended off the ground. Ensure that the robot is in its initial position after successful startup (arms idle, standing still).
2. Press the **LB+A** buttons on the remote controller to put the robot into damping status.
3. Press the **LT+RT** button on the remote controller to put the robot into debugging status. After successful switching, the robot will make a voice prompt (NOTE: the voice prompt feature must be enabled in advance on the App's voice assistant page), and the robot mode on the "Device > Joint Setting" page of the App will show "Low-Level Control Mode".

4. Press the **LT+A** button, and ATOM enters diagnostic state and assumes a specific diagnostic posture. Press the **LT+RT** buttons again, the robot returns to the debugging mode.

 NOTE

This process is used to confirm whether the robot has successfully entered the debugging state or for hardware troubleshooting.

- **Debugging interface position**

The debugging interface is located beneath the fixing ring on the robot's back. You can remove the silicone cover to access it.

2. Application Development

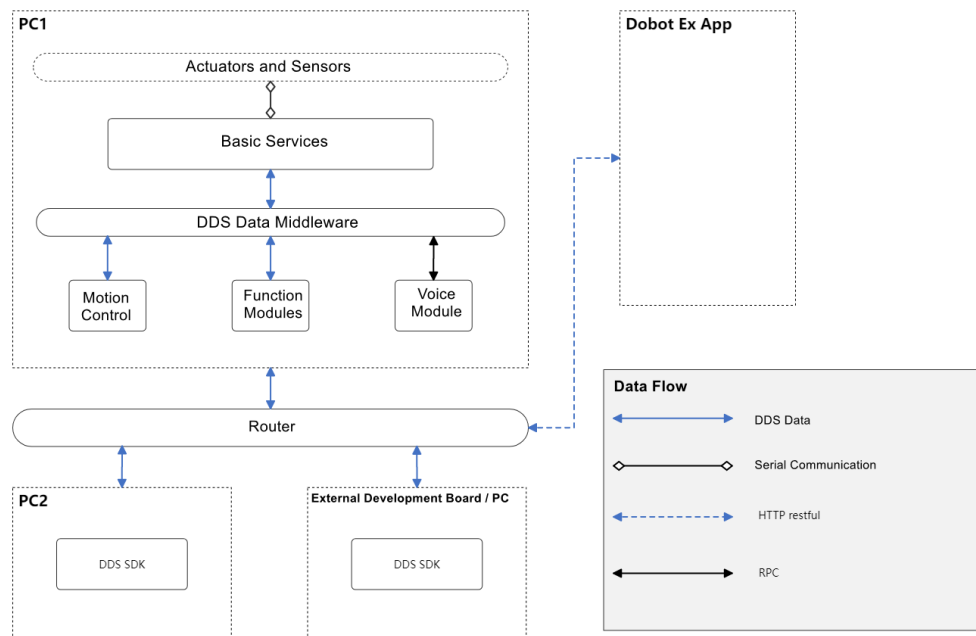
2.1 SDK overview

The DOBOT ATOM SDK is an integrated development kit that supports developers in secondary development on the ATOM robot.

The SDK uses DDS as its message middleware, which is the same communication mechanism used by ROS2, enabling direct communication between the two. As a result, developers can carry out secondary development on the ATOM robot using either ROS2 or the SDK.

- **Subscribe/Publish:** The receiver subscribes to a specific message, and the sender delivers the message based on the subscription list. This mode is mainly used for medium-to-high frequency or continuous data exchange.
- **Request/Response:** A question-and-answer pattern that retrieves data or performs operations via requests. This mode is used for low-frequency data exchange or when switching functions.

2.2 Architecture overview



ATOM robot

- Communication between the various functional modules is primarily carried out through DDS.
- Data from motors, LiDAR, and other sensors is collected via serial ports and then forwarded to the DDS middleware layer.
- The ATOM-Max robot comes with two built-in computing units: PC1 and PC2. PC1 is dedicated to the ATOM robot's motion control program and is not open to external access. Developers are only permitted to use PC2 for secondary development. The IP address for PC2 is 192.168.8.13, port 3390. For the initial username and password, please contact technical support.

Dobot Ex App	• DOBOT Ex App user accounts are created on the DOBOT management console and are provided to client individually. • The DOBOT Ex App connects to the ATOM robot via WiFi, and you can use it to view robot status or modify certain configurations.
--------------	--

2.3 Development languages

C++ and Python are supported for development.

2.4 Get SDK

SDK download link	https://github.com/Dobot-Arm/dobot_atom_sdk
URDF download link	https://github.com/Dobot-Arm/dobot_atom_urdf
ROS2 download link	https://github.com/Dobot-Arm/dobot_atom_ros2

2.5 Configuring the development environment

2.5.1 Environment dependencies

System environment	Development is recommended on Ubuntu 20.04; Mac and Windows systems are currently not supported. PC1 runs official services only and does not support development; development can only be performed on PC2. • OS: Ubuntu 20.04 LTS • Compiler: GCC 9.4.0
Network environment	Connect the user's computer and the ATOM switch to the same network. If you are a new user, we recommend connecting your computer to the ATOM switch using an Ethernet cable and adapter, and setting the IP address of the network interface for robot communication to 192.168.8.x. If you have experience, you may set up the network environment on your own.

2.5.2 Installing and compiling the example programs

Installing the SDK

```
mkdir build
cd build
cmake ..
make
```

Compiling the built-in example programs

NOTICE

When running these examples, the robot will move. Please make sure your environment is safe before operating.

- **High-level RPC example**

```
cd build
./rpc_test
```

- **Low-level DDS example**

NOTICE

When running these examples, the robot will move.

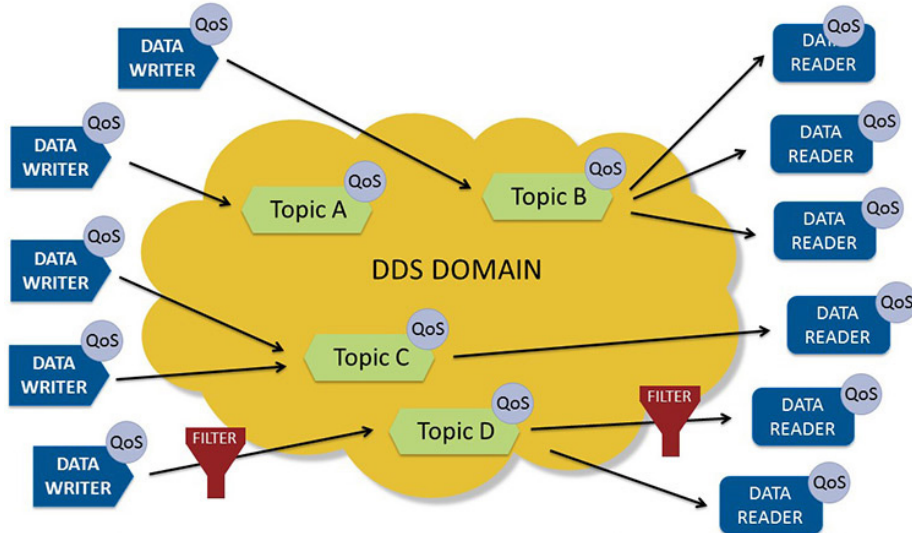
```
cd build
./bridge_test
```

3. Software Service Interface

3.1 DDS communication interface

3.1.1 Introduction

DDS (Data Distribution Service) is a middleware standard for distributed communication published by the OMG. It uses a publish/subscribe model and provides a variety of QoS (Quality of Service) policies to ensure real-time, efficient, and flexible data distribution, meeting the needs of various distributed real-time communication applications.



This document is an interface specification for DOBOT ATOM interactions using the DDS (Data Distribution Service) method. Since the DDS interfaces are defined through IDL files and generated via serialization, all interface documentation in this document is defined by IDL files.

All topics that require real-time publishing are prefixed with **rt/**.

Due to the diverse product configurations of DOBOT ATOM, the robot uses a split control approach for the upper and lower body, meaning that different topics are used to control the upper body and lower body separately.

Upper body control: waist, left arm, right arm, head

Lower body control: left leg, right leg

In the following sections, "low-level" refers to the program that manages the joints, BMS, IMU, and controller. If you are familiar with reinforcement learning or model-based related theories and are able to program using these techniques, you can communicate with the low-level program through the interfaces described in this document to achieve higher-level control over the DOBOT ATOM.

The algorithm state machine is a program built into PC1. Under certain conditions, it also controls the upper and lower body. For details on the state machine, refer to [State machine ID](#) section.

Dexterous hand array order: Right hand (thumb bend, index finger, middle finger, ring finger, little finger, thumb rotation), Left hand (thumb bend, index finger, middle finger, ring finger, little finger, thumb rotation).

Topic frequency description

The Host real-time main loop is aligned with the CODESYS EtherCAT task, with a cycle of approximately 1200 μ s (833 Hz). The table below lists the typical frequencies for each topic in the project. The recommended publishing period for upper and lower body control topics is 833 Hz. The Host takes the latest command frame within the real-time loop and sends it to the servos.

Topic	Direction	Typical frequency	Description
rt/upper/state	Status	$\approx$ 833 Hz	Cycle 1200 μ s
rt/lower/state	Status	$\approx$ 833 Hz	Cycle 1200 μ s
rt/upper/cmd	Control	$\approx$ 833 Hz	Cycle 1200 μ s
rt/lower/cmd	Control	$\approx$ 833 Hz	Cycle 1200 μ s
rt/posctl/state	Status	$\approx$ 833 Hz	Cycle 1200 μ s
rt/posctl/cmd	Control	Determined by the publisher	Event-driven; CART_JOG/RUN_TO recommends a keep-alive rate of approximately 10 Hz
rt/main/nodes/state	Status	5 Hz	Cycle 200 ms
rt/hands/state	Status	$\approx$ 125 Hz	Cycle 8000 μ s
rt/hands/cmd	Control	Determined by the publisher	Event-driven
rt/power/state	Status	5 Hz	Cycle 200 ms
rt/bms/info	Status	10 Hz	Cycle 100 ms
rt/amr/state	Status	40 Hz	Cycle 25 ms
Rt/amr/laserscan, rt/amr/odom	Status	After subscription, approximately 100 Hz	Requires SUBSCRIBE_LASER first

3.1.2 Shared sub-IDL file

bms information: bms_state.idl

```
cat bms_state.idl
struct BmsState_
module dobot_atom{
module msg{
module dds_{
@extensibility(FINAL)
struct BmsState_
{
uint16 bms_state; /*BMS status*/
uint16 afe_state; /*AFE chip status*/
uint32 bms_alarms; /*BMS error codes*/
uint16 battery_level; /*Battery percentage*/
uint16 battery_health; /*Battery health*/
uint16 pcb_board_temp; /*PCB board temperature*/
uint16 afe_chip_temp; /*AFE chip temperature*/
uint16 battery_now_current; /*Battery pack current*/
uint16 cells_voltage[16]; /*16 cell voltages*/
uint16 battery_pack_io_voltage; /*Battery pack discharge/charge interface voltage*/
uint32 bms_work_time; /*BMS runtime*/
uint16 bms_hardware_version; /*BMS hardware version*/
uint16 bms_software_version; /*BMS software version*/
uint16 heartbeat; /*Heartbeat*/
};
};
};
};
```

Joint information: motor_state.idl

```
cat motor_state.idl
module dobot_atom{
module msg{
module dds_{
@extensibility(FINAL)
struct MotorState_ {
uint8 mode; // Mode
float q; // Angular position
float dq; // Angular velocity
float ddq; // Angular acceleration
float tau_est; // Estimated torque
float q_raw; // Raw angular position
float dq_raw; // Raw angular velocity
float ddq_raw; // Raw angular acceleration
uint8 mcu_temp; // Servo control board temperature
uint8 mos_temp; // MOSFET temperature
uint8 motor_temp; // Motor temperature
uint8 bus_voltage; // Bus voltage
};
};
```

```
};  
};
```

Joint command: motor_cmd.idl

```
cat motor_cmd.idl  
module dobot_atom{  
  module msg{  
    module dds_{  
      @extensibility(FINAL)  
      struct MotorCmd_ {  
        uint8 mode;           // Mode  
        float q;              // Position  
        float dq;             // Velocity  
        float tau;            // Torque  
        float kp;             // Proportional gain  
        float kd;             // Derivative gain  
      };  
    };  
  };  
};
```

imu information: imu_state.idl

```
cat imu_state.idl  
module dobot_atom{  
  module msg{  
    module dds_{  
      @extensibility(FINAL)  
      struct IMUState_ {  
        float quaternion[4]; // Quaternion (4D)  
        float gyroscope[3];  // Gyroscope data (3D), in deg/s  
        float accelerometer[3]; // Accelerometer data (3D), in m/s2  
        float rpy[3];        // Euler angles (roll, pitch, yaw), in deg  
        uint8 temperature;   // Temperature sensor data, in °C  
      };  
    };  
  };  
};
```

wireless_remote description

When a corresponding key is pressed, a value is sent; when released, it returns to 0.

wireless_remote[2], from high bit to low bit, corresponds to the keys: " ", 'LT', 'RT', 'SELECT', 'START', 'LB', 'RB'.

wireless_remote[3], from high bit to low bit, corresponds to the keys: 'LEFT', 'DOWN', 'RIGHT', 'UP', 'Y', 'X', 'B', 'A'.

wireless_remote[4->7] store the float value of the LX axis, mapped to the range [-1, 0]. wireless_remote[4] is the low byte of the value.

wireless_remote[8->11] store the float value of the RX axis, mapped to the range [0, 1]. wireless_remote[8] is the low byte of the value.

wireless_remote[12->15] store the float value of the RY axis, mapped to the range [0, 1]. wireless_remote[12] is the low byte of the value.

wireless_remote[16->19] store the float value of the LY axis, mapped to the range [0, 1]. wireless_remote[16] is the low byte of the value.

Chassis status: amr_state.idl

```
module dobot_atom{
  module msg{
    module dds_{
      enum NavigationStatus_ {
        UNKNOWN,           // Unknown
        QUEUING,           // Queuing
        RUNNING,           // Running
        COMPLETED,        // Completed
        FAILED,            // Failed
        PAUSED,            // Paused
        CANCELED,          // Canceled
        WAITING_CONFIRM,    // Waiting for confirmation
        IDLE,              // Idle
        STOPPED            // Stop status
      }; // Enumeration for navigation_status value reference only
      enum DeviceStatus_ {
        DEVUNKNOWN,        // Unknown
        DEVIDLE,           // Robot idle
        TASKING,           // Task in progress
        ERROR,             // Error
        OFFLINE,           // Offline
        INIT,              // Initializing
        CHARGING,          // Charging
        UPGRADE            // Upgrading
      }; // Enumeration for device_status value reference only
      @extensibility(FINAL)
      struct AMREventStatus_{
        boolean emergency_stop_pressed; // Emergency stop button pressed
        boolean enable_pressed;        // Enable button pressed
        boolean path_blocked;          // Path blocked
        boolean low_battery;           // Low battery
        boolean obstacle_detected;     // Blocked by obstacle
      };

      @extensibility(FINAL)
    }
  }
}
```

```

struct AMRBasicStatus_{
    float battery_level;           // Battery percentage
    float battery_voltage;        // Battery voltage
    float battery_current;        // Battery current
    uint16 heartbeat;             // Heartbeat
};
@extensibility(FINAL)
struct AMRState_{
    DeviceStatus_device_status;   // Device status
    NavigationStatus_navigation_status; // Navigation status refer to the above
enum, used to check if the task is complete
    AMRBasicStatus_basic_status; // Basic status
    float position[3];             // Current position {x, y, yaw}
    AMREventStatus_amr_event;      // Chassis-related events
    uint32 error_code[32];         // Error codes
    uint32 task_id;                // Task ID
    uint32 work_mode;              // Robot work mode, with enum values
    listed on the right: 0: Task mode || 3: Remote control mode || 4: Maintenance mode
};
};
};
};
};

```

3.1.3 Switching to upper body control

Function description	To prevent both external sources (user or teleoperation) and the algorithm state machine from controlling the upper body at the same time, a switch interface for upper body control is added. If an external source (including PC2) needs to control the upper body, turn on this switch.
Topic name	rt/switch/upper/control

IDL file content

```

cat switch_upper_control.idl
module dobot_atom{
    module msg{
        module dds_{
            @extensibility(FINAL)
            struct SwitchUpperControl_{
                boolean flag; /* true: upper body control enabled, false: upper body control disabled */
            };
        };
    };
};

```

3.1.4 Upper body status

Function description	Low-level information of the upper body
Topic name	rt/upper/state

IDL file content

```
cat upper_state.idl
#include "imu_state.idl"
#include "motor_state.idl"
#include "bms_state.idl"
module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct UpperState_ {
        char robot_type[16];          /* Product type as defined in the product definition */
        boolean is_upper_control;     /* true: upper body control enabled, false: upper body control
disabled */
        uint16 fsm_id;               /*Corresponding algorithm state machine*/
        MotorState_ motor_state[17];
        BmsState_ bms_state;
        octet wireless_remote[40];   /*Controller key values*/
        uint32 reserve;              /*Reserved for DOBOT*/
      };
    };
  };
};
```

NOTE

is_upper_control indicates whether teleoperation is enabled. true means teleoperation is turned on.

3.1.5 Upper body control

Function description	Upper body control command
Topic name	rt/upper/cmd

IDL file content

```
cat upper_cmd.idl
#include "motor_cmd.idl"
module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct UpperCmd_ {
        MotorCmd_ motor_cmd[17];          // Motor commands, containing 17 MotorCmd_
structures
      };
    };
  };
};
```

3.1.6 Lower body status

Function description	Lower body status information
Topic name	rt/lower/state

IDL file content

```
cat lower_state.idl
#include "imu_state.idl"
#include "motor_state.idl"
#include "bms_state.idl"

module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct LowerState_ {
        uint16 fsm_id;          /*Corresponding algorithm state machine ID*/
        IMUState_ imu_state;
        MotorState_ motor_state[12];
        BmsState_ bms_state;
        octet wireless_remote[40];
        uint32 reserve;          /*Reserved for DOBOT*/
      };
    };
  };
};
```

3.1.7 Lower body control

Function description	Lower body control command
Topic name	rt/lower/cmd

IDL file content

```
cat lower_cmd.idl
#include "motor_cmd.idl"

struct LowerCmd_ {
    MotorCmd_ motor_cmd[12];           // Motor commands, containing 12 MotorCmd_ structures
};
```

3.1.8 Joint and CAN board status

Function description	Provided to users for auxiliary debugging purposes
Topic name	rt/main/nodes/state

IDL file content

```
module dobot_atom{
    module msg{
        module dds_{
            @extensibility(FINAL)
            struct AxisStateInfo_
            {
                uint8  servo_state;
                uint16 error_code; // Motor alarm error code
                uint16 warn_code; // Motor warning error code
                int32  pos_err_code;
                int32  vel_err_code;
                int32  torque_err_code;
                uint8  node_state;
                uint8  display_op_mode;
                boolean is_virtual; /* default: false */
                uint8  mcu_temp;
                uint8  mos_temp;
                uint8  motor_temp;
                uint8  bus_voltage;
                uint16 software_version;
            };

            @extensibility(FINAL)
            struct EcatSlaveInfo_
            {
                boolean is_virtual;
                uint8  slave_state;
```

```

uint16 error_code;
uint16 software_version;
};

@extensibility(FINAL)
struct MainNodesState_ {
    AxisStateInfo_ left_leg[6];
    AxisStateInfo_ right_leg[6];
    AxisStateInfo_ waist;
    AxisStateInfo_ left_arm[7];
    AxisStateInfo_ right_arm[7];
    AxisStateInfo_ head[2];
    EcatSlaveInfo_ ecat2can[2];
    /* AxisStateInfo left_hand[6]; */
    /* AxisStateInfo right_hand[6]; */
    /* uint16 power_state[8]; */
};
};
};
};
};

```

3.1.9 Clear error

Function description	Clears only low-level errors
Topic name	rt/clear/errors

IDL file content

```

cat clear_errors.idl
struct ClearErrors_ {
    int32 msg_id;
};

```

3.1.10 Dexterous hand status

Function description	Angle information of each finger of the dexterous hand only
Topic name	rt/hands/state

IDL file content

```

cat hands_state.idl
#include "motor_state.idl"

module dobot_atom{
    module msg{
        module dds_{
            @extensibility(FINAL)
            struct HandsState_

```

```
{
    MotorState_hands[12];
};
};
};
};
```

3.1.11 Dexterous hand control

Function description	Angle control of each finger of the dexterous hand only
Topic name	rt/hands/cmd

IDL file content

```
cat hands_cmd.idl
#include "motor_cmd.idl"
module dobot_atom{
    module msg{
        module dds_{
            @extensibility(FINAL)
            struct HandsCmd_
            {
                MotorCmd_hands[12];
            };
        };
    };
};
```

3.1.12 Wheeled humanoid chassis status

Function description	Contains various status information of the chassis
Topic name	rt/amr/state

IDL file content

```
module dobot_atom{
    module msg{
        module dds_{
            enum NavigationStatus_ {
                UNKNOWN,           // Unknown
                QUEUING,           // Queuing
                RUNNING,           // Running
                COMPLETED,        // Completed
                FAILED,            // Failed
                PAUSED,            // Paused
                CANCELED,          // Canceled
                WAITING_CONFIRM,    // Waiting for confirmation
                IDLE,              // Idle
                STOPPED,           // Stop status
            };
        };
    };
};
```

```

}; // Enumeration for navigation_status value reference only

enum DeviceStatus_ {
DEVUNKNOWN,          // Unknown
DEVIDLE,             // Robot idle
TASKING,             // Task in progress
ERROR,              // Error
OFFLINE,            // Offline
INIT,              // Initializing
CHARGING,          // Charging
UPGRADE            // Upgrading
}; // Enumeration for device_status value reference only
@extensibility(FINAL)
struct AMREventStatus_{
boolean emergency_stop_pressed;          // Emergency stop button pressed
boolean enable_pressed;                // Enable button pressed
boolean path_blocked;                 // Path blocked
boolean low_battery;                  // Low battery
boolean obstacle_detected;            // Blocked by obstacle
};

@extensibility(FINAL)
struct AMRBasicStatus_{
float battery_level;          // Battery percentage
float battery_voltage;        // Battery voltage
float battery_current;        // Battery current
uint16 heartbeat;            // Heartbeat
};

@extensibility(FINAL)
struct AMRState_ {
DeviceStatus_ device_status;          // Device status
NavigationStatus_ navigation_status;  // Navigation status refer to the above
enum, used to check if the task is complete
AMRBasicStatus_ basic_status;          // Basic status
float position[3];                    // Current position {x, y, yaw}
AMREventStatus_ amr_event;            // Chassis-related events
uint32 error_code[32];                // Error codes
uint32 task_id;                       // Task ID
uint32 work_mode;                     // Robot work mode, with enum values
listed on the right: 0: Task mode || 3: Remote control mode || 4: Maintenance mode
};
};
};
};
};

```

3.1.13 Wheeled humanoid chassis control

Function description	Used to send commands to the chassis of the wheeled humanoid robot
Topic name	rt/amr/cmd

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      enum AMRCommandType {
        CANCEL_TASK,      // Cancel task
        PAUSE_TASK,       // Pause task
        RESUME_TASK,      // Resume task
        MOVE_TO_TAG,      // Move to tag point (target_id required, theta optional, theta is absolute
angle; if 0 or omitted, no angle is specified)
        REMOTE_CONTROL,   // Remote control
        ROTATE,           // Rotate in place (theta required — relative angle from current chassis
orientation; not available on iRAYPLE)
        START_REMOTE,     // Start remote control
        STOP_REMOTE,      // Stop remote control
        SUBSCRIBE_LASER,   // Subscribe to chassis LiDAR data
        UNSUBSCRIBE_LASER, // Unsubscribe from chassis LiDAR data
        START_MAPPING,     // Start mapping with the chassis
        SAVE_MAP,          // Save the current map locally
        STOP_MAPPING,      // Stop mapping
        SET_VEL,           // Set velocity (both linear and angular velocity for no-load and loaded
conditions)
        SET_ACCEL,         // Set acceleration (both linear and angular acceleration for no-load and
loaded conditions)
        SET_DECEL          // Set deceleration (both linear and angular deceleration for no-load and
loaded conditions)
      };

      @extensibility(FINAL)
      struct AMRCommand_ {
        AMRCommandType command_type; // Command type (see AMRCommandType above)
        uint32 target_id;             // Target ID (used for MOVE_TO_TAG)
        float linear_vel;             // Linear velocity, in m/s
        float angular_vel;            // Angular velocity, in rad/s
        uint32 command_id;            // Command ID (for tracking)
        uint64 timestamp;             // Timestamp
        float theta;                  // Input angle
      };
    };
  };
};

```

3.1.14 Emergency stop status

Function description	Emergency stop information in the system
Topic name	rt/emergency/state

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct EmergencyState_ {
        boolean soft_emergency_triggered; // Soft emergency stop (triggered by App)
        boolean hard_emergency_triggered; // Hard emergency stop (triggered by user board)
        boolean amr_emergency_triggered; // AMR emergency stop (triggered by wheeled chassis)
      };
    };
  };
};

```

3.1.15 State machine status

Function description	Information of the state machine published by the algorithm
Topic name	rt/set/fsm/id

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct SetFsmId_ {
        uint16 id; // For algorithm state machine
        char current_action; // Current action being executed by the algorithm, empty if none
      };
    };
  };
};

```

3.1.16 AMR LiDAR data

Function description	After sending a command to subscribe to LiDAR data via <code>amr_cmd</code> , the thread begins publishing LiDAR data and odometry data from the wheeled humanoid chassis
Topic name	<code>rt/amr/laserscan</code>

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct LaserScan_ {
        uint64 timestamp;           // Timestamp (ms)
        int32 lidar[1024];         // LiDAR data (up to 1024 points, padded with 0 if less), in mm
      };
    };
  };
};

```

3.1.17 AMR odometry data

Function description	After sending a command to subscribe to LiDAR data via <code>amr_cmd</code> , the thread begins publishing LiDAR data and odometry data from the wheeled humanoid chassis
Topic name	<code>rt/amr/odom</code>

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      @extensibility(FINAL)
      struct Odom_ {
        uint64 timestamp;           // Timestamp (ms)
        int32 x;                    // Odometry X coordinate, in mm
        int32 y;                    // Odometry Y coordinate, in mm
        int32 angle;               // Odometry angle, in 0.001 degrees
      };
    };
  };
};

```

3.1.18 DH-Robotics gripper command

Function description	Sends initialization, target opening position, and gripping force commands to the left and right DH-Robotics grippers. After the Host subscribes, it writes to the gripper registers via Modbus
Topic name	rt/hands/cmd

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      enum DHGripperInitCmd {
        DH_INIT_NONE,          // No initialization triggered
        DH_INIT_HOMING         // Return to home (find the reference position)
      };

      @extensibility(FINAL)
      struct DHGripperSingleCmd_ {
        DHGripperInitCmd init_cmd;
        float position; // 0 to 1, where 1.0 = fully open (internally mapped to register value 0–1000)
        float force;
      };

      @extensibility(FINAL)
      struct DHGripperCmd_ {
        DHGripperSingleCmd_ right;
        DHGripperSingleCmd_ left;
      };
    };
  };
};

```

3.1.19 DH-Robotics gripper status

Function description	The Host periodically publishes the initialization status, gripping state, real-time opening position, and force feedback of the left and right grippers
Topic name	rt/hands/cmd rt/hands/state

IDL file content

```

module dobot_atom{
  module msg{
    module dds_{
      enum DHGripperInitResultState {
        DH_INIT_NOT_DONE,
        DH_INIT_SUCCESS,
        DH_INIT_EXECUTING
      };
    };
  };
};

```

```
enum DHGripperGripState {
    DH_GRIP_MOVING,
    DH_GRIP_IN_PLACE,
    DH_GRIP_HOLDING_OBJECT,
    DH_GRIP_OBJECT_DROPPED
};

@extensibility(FINAL)
struct DHGripperSingleState_ {
    DHGripperInitResultState init_result_state;
    DHGripperGripState grip_state;
    float realtime_position; // 0~1
    float get_force_value;
};

@extensibility(FINAL)
struct DHGripperState_ {
    DHGripperSingleState_ right;
    DHGripperSingleState_ left;
};
};
};
};
};
```

3.1.20 Position control joint position control command

Function description	Switches to MIT/position control application mode, or sends position control motion commands to the dual arms and lift (ATOM-W). Once the Host subscribes, it calls the position control algorithm and writes to the PLC shared memory
Topic name	rt/posctl/cmd

IDL file content

```
module dobot_atom{
    module msg{
        module dds_{
            enum PosControlCommandType {
                SWITCH_MODE, // Switch MIT <-> position control, fill in target_mode
                MOVE_L, // Cartesian linear motion
                MOVE_J, // Joint space motion
                RUN_TO, // Cartesian tracking, stop=0 start, stop=1 stop
                CART_JOG, // Cartesian jogging, stop same as above
                STOP_MOTION // Stop position control motion
            };

            @extensibility(FINAL)
            struct PosControlCmd_ {
                PosControlCommandType command_type;
                sequence<double> left_q; // MOVE_J: 1..7 prefix, deg; empty = left arm not participating
                sequence<double> right_q;
            };
        };
    };
};
```

```
sequence<double> hips_q; // 1..3, lift
sequence<double> left_pose; // MOVE_L/RUN_TO: 0 or 6, [x,y,z]mm + [rx,ry,rz]deg
sequence<double> right_pose;
sequence<double> hips_height; // 0 or 1, mm
sequence<int32> left_cart_jog; // CART_JOG: 0 or 6, -1/0/1
sequence<int32> right_cart_jog;
sequence<int32> hips_cart_jog; // 0 or 1, -1/0/1
int32 stop;
double speed;
int32 target_mode; // SWITCH_MODE: 0=MIT, 1=position control
uint32 command_id;
uint64 timestamp;
};
};
};
};
```

3.1.21 Position control status

Function description	The Host periodically publishes the position control application mode, dual-arm/lift joint angles, Cartesian poses, algorithm error codes, and planning status
Topic name	rt/posctl/state

IDL file content

```
module dobot_atom{
  module msg{
    module dds_{
      struct CartTargetPose {
        double left_pose[6]; // mm, deg
        double right_pose[6];
        double hips_height; // mm
      };

      struct PosControlJointQ {
        double left_q[7]; // deg
        double right_q[7];
        double hips_q[3];
      };

      struct PosControlState_ {
        CartTargetPose pose;
        int32 error_code[3]; // [0] left arm, [1] right arm, [2] lift
        int32 status_code[3]; // Same axis order
        int32 control_mode; // 0 = MIT, 1 = Position control
        PosControlJointQ joint_q;
      };
    };
  };
};
```

3.2 Low-level service interface

The low-level communication provides data exchange between the user-side PC (PC2 / external PC) and the robot. The low-level communication uses the DDS protocol.

The interface takes effect after the state machine is switched to debugging state (fsm id: 2).

WARNING

When using the low-level service interface, please set the robot to debugging mode first. Failure to do so may result in unexpected behavior or risk.

3.2.1 Low-level control command interface

The Bridge class encapsulates DDS communication and provides interfaces for "sending control commands" and "getting the latest status".

Bridge.h

```
namespace Atom
{
    class Bridge
    {
    public:
        Bridge();
        ~Bridge();

        /* Command */
        void SetNewestLegCommand(const LegCommand &motor_command);
        void SetNewestHandCommand(const HandCommand &hand_command);
        void SetNewestArmCommand(const ArmCommand &arm_command);
        void SetNewestFsmCommand(const int &fsm_id);

        /* State */
        const std::shared_ptr<const BaseState> GetNewestBaseStatePtr() { return
base_state_buffer_.GetData(); }
        const std::shared_ptr<const JointState> GetNewestJointStatePtr() { return
joint_state_buffer_.GetData(); }
        const std::shared_ptr<const ArmJointState> GetNewestArmStatePtr() { return
arm_state_buffer_.GetData(); }
        const std::shared_ptr<const DexterousHandState> GetNewestHandStatePtr() { return
hand_state_buffer_.GetData(); }
        const std::shared_ptr<const RemoteCommand> GetNewestRemoteCommandPtr() { return
remote_command_buffer_.GetData(); }

    private:
        .....
    };
} // namespace Atom

#endif // BRIDGE_H
```

motor_command.h

```

.....

namespace Atom
{
    struct LegCommand {
        Eigen::Matrix<float, Atom::kLegDofs, 1> q_ref = Eigen::Matrix<float, Atom::kLegDofs, 1>::Zero();
// angle
        Eigen::Matrix<float, Atom::kLegDofs, 1> dq_ref = Eigen::Matrix<float, Atom::kLegDofs, 1>::Zero();
// velocity
        Eigen::Matrix<float, Atom::kLegDofs, 1> kp = Eigen::Matrix<float, Atom::kLegDofs, 1>::Zero();
// position stiffness coefficients
        Eigen::Matrix<float, Atom::kLegDofs, 1> kd = Eigen::Matrix<float, Atom::kLegDofs, 1>::Zero();
// velocity stiffness coefficients
        Eigen::Matrix<float, Atom::kLegDofs, 1> tau_forward = Eigen::Matrix<float, Atom::kLegDofs,
1>::Zero(); // forward torque
    };

    .....
}

```

robot_state.h

```

.....

namespace Atom
{
    struct JointState {
        Eigen::Matrix<float, Atom::kLegDofs, 1> q = Eigen::Matrix<float, Atom::kLegDofs,
1>::Zero(); // angle
        Eigen::Matrix<float, Atom::kLegDofs, 1> dq = Eigen::Matrix<float, Atom::kLegDofs,
1>::Zero(); // velocity
        Eigen::Matrix<float, Atom::kLegDofs, 1> tau = Eigen::Matrix<float, Atom::kLegDofs,
1>::Zero(); // torque
    };

    .....
}

```

3.2.2 Low-level control example

```

#include "common/bridge.h"
.....

int main(int argc, char *argv[])
{
    .....

    // Initialize the bridge for communication and the command structures for legs, hands, and arms.
    Atom::Bridge bridge;
    Atom::LegCommand leg_command;
    Atom::HandCommand hand_command;
    Atom::ArmCommand arm_command;
}

```

```

// Define timing parameters for the control loop.
// The control loop will run at a frequency of 1/control_dt.
const float init_duration = 5.0; // time unit: s
const float control_dt = 0.01; // time unit: s
const int s2us = 1e6;
float time = 0.0;
Timer timer;

// Set the initial position for the robot's arms and legs.
// The robot will move to this position at the beginning of the program.
Eigen::Vector<float, Atom::kArmDofs> q_arm_init;
Eigen::Vector<float, Atom::kLegDofs> q_leg_init;
q_leg_init.setZero();
q_arm_init.setZero();
for (int i = 0; i < Atom::kLegDofs; ++i) { q_leg_init[i] = q_val[0][i]; }
for (int i = 0; i < Atom::kArmDofs; ++i) { q_arm_init[i] = q_val[0][i + Atom::kLegDofs]; }
Eigen::Vector<float, Atom::kLegDofs> q_leg_ref;
Eigen::Vector<float, Atom::kArmDofs> q_arm_ref;

int row = 0;

// Main control loop.
while (1) {
    // Get the current joint states from the robot.
    const std::shared_ptr<const Atom::JointState> js_tmp_ptr =
bridge.GetNewestJointStatePtr(); // leg
    const std::shared_ptr<const Atom::ArmJointState> ajs_tmp_ptr =
bridge.GetNewestArmStatePtr(); // arm
    timer.Tic();

    if (js_tmp_ptr && ajs_tmp_ptr) {
        time += control_dt;
        if (time < init_duration) {
            // Interpolate to the initial position over a specified duration.
            // This ensures a smooth start for the robot's motion.
            q_leg_ref = (q_leg_init - js_tmp_ptr->q) * time / init_duration + js_tmp_ptr->q;
            q_arm_ref = (q_arm_init - ajs_tmp_ptr->q) * time / init_duration + ajs_tmp_ptr->q;
        } else {
            // Follow the predefined trajectory from the file.
            for (int i = 0; i < Atom::kLegDofs; ++i) { q_leg_ref[i] = q_val[row][i]; }
            for (int i = 0; i < Atom::kArmDofs; ++i) { q_arm_ref[i] = q_val[row][i +
Atom::kLegDofs]; }

            row++;
            if (row == q_val.size() - 1) { row = 0; } // Loop back to the beginning of the
trajectory.
        }

        // Set the PID gains for the legs from the configuration file.
        for (int j = 0; j < Atom::kLegDofs; j++) {
            leg_command.kp(j) = controlParams[j]["kp"].get<double>();
            leg_command.kd(j) = controlParams[j]["kd"].get<double>();
        }

        // Set the PID gains for the arms from the configuration file.
        for (int j = 0; j < Atom::kArmDofs; j++) {
            arm_command.kp(j) = controlParams[j + Atom::kLegDofs]["kp"].get<double>();

```

```

        arm_command.kd(j) = controlParams[j + Atom::kLegDofs]["kd"].get<double>();
    }

    // Set and send the commands to the robot.
    leg_command.q_ref = q_leg_ref;
    leg_command.dq_ref.setZero();
    leg_command.tau_forward.setZero();
    bridge.SetNewestLegCommand(leg_command);

    hand_command.q_ref.setZero();
    hand_command.dq_ref.setZero();
    bridge.SetNewestHandCommand(hand_command);

    arm_command.q_ref = q_arm_ref;
    arm_command.dq_ref.setZero();
    arm_command.tau_forward.setZero();
    bridge.SetNewestArmCommand(arm_command);
}

// Wait for the next control cycle.
timer.Toc();
timer.usDelay(control_dt * s2us - timer.usDuration());
}

return 0;
}

```

3.3 High-level motion service interface

The high-level motion service interface mainly provides whole-body control and upper body control for the robot.

- Whole-body control: Controls the robot's mode switching and movement, providing RPC interfaces.
- Upper-body control: Allows independent control of the upper-body motors to perform operational tasks while the robot's built-in motion controller is running, providing DDS interfaces.

3.3.1 High-level whole-body control RPC interface

Interface name	GetFsmId
Interface definition	<code>RpcErrorCode GetFsmId(int32_t &fsm_id);</code>
Overview	Get the current state machine ID.
Parameter	fsm_id: The retrieved current state machine ID.
Return	Returns 0 on success; otherwise, returns the corresponding error code. See Error code for details.
Remark	For the state machine ID, see State Machine ID for details.

Interface name	SetFsmId
Interface definition	<code>RpcErrorCode SetFsmId(int32_t fsm_id);</code>
Overview	Set the state machine ID.
Parameter	fsm_id: Target state machine ID.
Return	Returns 0 on success; otherwise, returns the corresponding error code. See Error code for details.
Remark	For the state machine ID, see State Machine ID for details.

Interface name	SwitchUpperLimbControl
Interface definition	<code>RpcErrorCode SwitchUpperLimbControl(bool is_on);</code>
Overview	Switching to upper body control.
Parameter	is_on: true switches upper body control to interface control; false disables the switch and returns control to the main motion control for whole-body control.
Return	Returns 0 on success; otherwise, returns the corresponding error code. See Error code for details.
Remark	Switching is only allowed in position control states such as "Ready" and walking states. The switch will be automatically turned off when the robot enters a state where switching is not allowed.

Interface name	SetVel
Interface definition	RpcErrorCode SetVel(float vx, float vy, float vyaw, float duration = 1.0);
Overview	Set velocity command.
Parameter	vx: Forward/backward movement velocity, forward is positive, unit: m/s; vy: Left/right movement velocity, leftward is positive, unit: m/s; vyaw: Rotation velocity, counterclockwise is positive, unit: rad/s; duration: Duration of the speed command. Motion stops after timeout. Unit: seconds (s).
Return	Returns 0 on success; otherwise, returns the corresponding error code. See Error code for details.
Remark	The low-level layer automatically limits the set parameters to a reasonable range. The remote control command has higher priority. To stop the whole-body motion, set the velocity command to zero: SetVel(0, 0, 0).

3.3.2 Error code (RpcErrorCode)

ID	Description
0	No error
1	socket creation failed
2	Communication connection failure
3	Failed to read response
4	Failed to parse response data
5	Response data format error
6	Server internal error
100	Illegal state transition request. Please call only when in the allowed state
101	Illegal upper-body control authority switching request. Please call only when in the allowed state
102	Illegal velocity command setting request. Please call

ID	Description
	only when in the allowed state

3.3.3 Upper-body motor control DDS interface

Refer to [Low-level service interface](#).

3.4 State machine ID (FsmId)

ID	Status	Description
0	Zero-torque status	The robot's motors stop active movements, and its joints swing freely without any damping resistance.
1	Damping status	All robot motors will stop active movement, and there is a noticeable damping sensation during swinging.
2	Debugging status	Development and debugging are conducted based on the underlying control SDK. Acquire real-time operation data (battery, motors, IMU, etc.) and directly drive the motors.
3	Diagnostic status	Controls joint positions to switch the robot to a specific diagnostic posture.
100	Prepare for locked standing status	The robot slowly assumes the prepared posture before the motion mode within 5 seconds (without balance control)
101	Squatting status	The robot slowly assumes a squatting posture within 5 seconds (without balance control)
102	Sitting status	The robot slowly assumes a sitting posture within 5 seconds (without balance control)
301	Bent-knee walking status	The robot will slowly move to the starting position of the entry model within 5 seconds (without balance control), and then enter the model via the controller. After entering the model, it will maintain balance and serve as the preparation posture before motion states and skill states.

3.5 Audio service interface

The following are the RPC interface description provided by the server. All interfaces return type `RpcErrorCode`, and the result or data is returned via reference parameters.

Interface name	GetDevState
Interface definition	<code>RpcErrorCode GetDevState(int32_t& devstate)</code>
Overview	Get the device status.
Output parameter	devstate: Device status value
Example	<pre>enum SpeakAudioStateEnum { eSpeakAudioInit = 1, // Not initialized eSpeakAudioPlaying = 2, // Playing eSpeakAudioPause = 3, // Paused eSpeakAudioStop = 4 // Stopped };</pre>

Interface name	SetVolume
Interface definition	<code>RpcErrorCode SetVolume(int32_t volume, bool& result)</code>
Overview	Set the device volume.
Input parameter	volume: Target volume value volume:0-128
Output parameter	result: Whether the setting was successful

Interface name	GetVolume
Interface definition	<code>RpcErrorCode GetVolume(int32_t& volume)</code>
Overview	Obtain the current device volume.
Output parameter	volume: Current volume value

Interface name	PlayAudio
Interface definition	RpcErrorCode PlayAudio(bool& result)
Overview	Play audio.
Output parameter	result: Whether the playback was successful

Interface name	StopAudio
Interface definition	RpcErrorCode StopAudio(bool& result)
Overview	Stop audio playback.
Output parameter	result: Whether the stop was successful

Interface name	PauseAudio
Interface definition	RpcErrorCode PauseAudio(bool& result)
Overview	Pause audio playback.
Output parameter	result: Whether the pause was successful

Interface name	ContinueAudio
Interface definition	RpcErrorCode ContinueAudio(bool& result)
Overview	Resume audio playback.
Output parameter	result: Whether the resume was successful

Interface name	Pc2SwitchTTS
Interface definition	RpcErrorCode Pc2SwitchTTS(bool, bool& result)
Function overview	Switch the TTS function on or off.
Input parameter	flag: Enable or disable TTS
Output parameter	result: The operation result

Interface name	Pc2PlayTTS
Interface definition	RpcErrorCode Pc2PlayTTS(const std::string text, bool& result)
Overview	Play TTS audio for the specified text.
Input parameter	text: Text content
Output parameter	result: Whether the playback was successful

Interface name	Pc2SwitchASR
Interface definition	RpcErrorCode Pc2SwitchASR(bool, bool& result)
Overview	Switch the ASR function on or off.
Input parameter	flag: Enable or disable ASR
Output parameter	result: The operation result

Interface name	Pc2PlayASR
Interface definition	RpcErrorCode Pc2PlayASR(const std::string language, uint32_t timeout, std::string& result)
Overview	Start ASR recognition.
Input parameter	- language: Language type - Timeout: timeout duration
Output parameter	result: Recognized text - Timeout function timeout - http interface error Http fail - Invalid response content Invalid json - udp multicast connection failed

	Socket fail
	Failed to load audio device
	Device load fail

Interface name	Pc2PlayDify
Interface definition	RpcErrorCode Pc2PlayDify(const std::string text, std::string& result)
Overview	Generate speech using the Dify model and play it.
Input parameter	Text: Question text
Output parameter	result: Returned answer text - http interface error Http fail - Invalid response content Invalid json - Failed to load audio device Device load fail

Interface name	Pc1MicTTS
Interface definition	RpcErrorCode Pc1MicTTS(const std::string wavPath, const std::string text, bool& result)
Overview	Use the microphone to play TTS for the specified text and generate an audio file.
Input parameter	- wavPath: Save path - text: Text content
Output parameter	result: Whether the playback was successful

Interface name	Pc1MicServer
Interface definition	RpcErrorCode Pc1MicServer(bool, bool& result)
Overview	Start or stop the microphone multicast service.
Input parameter	flag: true=start, false=stop.
Output parameter	result: Whether the operation was successful

Interface name	Pc1MicVW
Interface definition	RpcErrorCode Pc1MicVW(const uint32_t timeout, std::string& result)
Overview	Start wake-up recognition using the microphone.
Input parameter	timeout: timeout duration (s)
Output parameter	result: Wake up keyword • • Timeout: function timeout • udp multicast connection failed Socket fail • aikit interface error Function fail • Failed to load audio device Device load fail

Interface name	Pc1MicESR
Interface definition	RpcErrorCode Pc1MicESR(const uint32_t timeout, std::string& result)
Overview	Start command word recognition using the microphone.
Input parameter	timeout: timeout duration (s)
Output parameter	result: - Recognition result - Timeout: function timeout - udp multicast connection failed Socket fail - ait interface error Function fail - Failed to load audio device Device load fail

4. Reference Examples

4.1 DDS communication examples

```
#include "common/bridge.h"
.....

int main(int argc, char *argv[])
{
    .....

    // Initialize the bridge for communication and the command structures for legs, hands, and
arms.
    Atom::Bridge bridge;
    Atom::LegCommand leg_command;
    Atom::HandCommand hand_command;
    Atom::ArmCommand arm_command;

    // Define timing parameters for the control loop.
    // The control loop will run at a frequency of 1/control_dt.
    const float init_duration = 5.0; // time unit: s
    const float control_dt = 0.01; // time unit: s
    const int s2us = 1e6;
    float time = 0.0;
    Timer timer;

    // Set the initial position for the robot's arms and legs.
    // The robot will move to this position at the beginning of the program.
    Eigen::Vector<float, Atom::kArmDofs> q_arm_init;
    Eigen::Vector<float, Atom::kLegDofs> q_leg_init;
    q_leg_init.setZero();
    q_arm_init.setZero();
    for (int i = 0; i < Atom::kLegDofs; ++i) { q_leg_init[i] = q_val[0][i]; }
    for (int i = 0; i < Atom::kArmDofs; ++i) { q_arm_init[i] = q_val[0][i + Atom::kLegDofs]; }
    Eigen::Vector<float, Atom::kLegDofs> q_leg_ref;
    Eigen::Vector<float, Atom::kArmDofs> q_arm_ref;

    int row = 0;

    // Main control loop.
    while (1) {
        // Get the current joint states from the robot.
        const std::shared_ptr<const Atom::JointState> js_tmp_ptr =
bridge.GetNewestJointStatePtr(); // leg
        const std::shared_ptr<const Atom::ArmJointState> ajs_tmp_ptr =
bridge.GetNewestArmStatePtr(); // arm
        timer.Tic();

        if (js_tmp_ptr && ajs_tmp_ptr) {
            time += control_dt;
            if (time < init_duration) {
                // Interpolate to the initial position over a specified duration.
                // This ensures a smooth start for the robot's motion.
                q_leg_ref = (q_leg_init - js_tmp_ptr->q) * time / init_duration + js_tmp_ptr->q;
```

```

        q_arm_ref = (q_arm_init - ajs_tmp_ptr->q) * time / init_duration + ajs_tmp_ptr->q;
    } else {
        // Follow the predefined trajectory from the file.
        for (int i = 0; i < Atom::kLegDofs; ++i) { q_leg_ref[i] = q_val[row][i]; }
        for (int i = 0; i < Atom::kArmDofs; ++i) { q_arm_ref[i] = q_val[row][i +
Atom::kLegDofs]; }

        row++;
        if (row == q_val.size() - 1) { row = 0; } // Loop back to the beginning of the
trajectory.
    }

    // Set the PID gains for the legs from the configuration file.
    for (int j = 0; j < Atom::kLegDofs; j++) {
        leg_command.kp(j) = controlParams[j]["kp"].get<double>();
        leg_command.kd(j) = controlParams[j]["kd"].get<double>();
    }

    // Set the PID gains for the arms from the configuration file.
    for (int j = 0; j < Atom::kArmDofs; j++) {
        arm_command.kp(j) = controlParams[j + Atom::kLegDofs]["kp"].get<double>();
        arm_command.kd(j) = controlParams[j + Atom::kLegDofs]["kd"].get<double>();
    }

    // Set and send the commands to the robot.
    leg_command.q_ref = q_leg_ref;
    leg_command.dq_ref.setZero();
    leg_command.tau_forward.setZero();
    bridge.SetNewestLegCommand(leg_command);

    hand_command.q_ref.setZero();
    hand_command.dq_ref.setZero();
    bridge.SetNewestHandCommand(hand_command);

    arm_command.q_ref = q_arm_ref;
    arm_command.dq_ref.setZero();
    arm_command.tau_forward.setZero();
    bridge.SetNewestArmCommand(arm_command);
}

// Wait for the next control cycle.
timer.Toc();
timer.usDelay(control_dt * s2us - timer.usDuration());
}

return 0;
}

```

4.2 ROS2 communication examples

```
#include "rclcpp/rclcpp.hpp"
#include "dobot_atom/msg/upper_state.hpp"

class SimpleUpperStateReader : public rclcpp::Node
{
public:
    SimpleUpperStateReader() : Node("simple_upper_state_reader")
    {
        // Subscribe to /upper/state topic
        upper_state_sub_ = this->create_subscription<dobot_atom::msg::UpperState>(
            "/upper/state", 10,
            std::bind(&SimpleUpperStateReader::upperStateCallback, this,
std::placeholders::_1));

        RCLCPP_INFO(this->get_logger(), "Simple upper state reader started, waiting for data...");
    }

private:
    void upperStateCallback(const dobot_atom::msg::UpperState::SharedPtr msg)
    {
        RCLCPP_INFO(this->get_logger(), "Received upper state data:");
        RCLCPP_INFO(this->get_logger(), "  Timestamp: %ld.%09ld", msg->stamp.sec,
msg->stamp.nanosec);
        RCLCPP_INFO(this->get_logger(), "  Joint count: %zu", msg->q.size());

        // Print first 3 joint positions (if available)
        for (size_t i = 0; i < std::min(msg->q.size(), size_t(3)); ++i) {
            RCLCPP_INFO(this->get_logger(), "  Joint %zu position: %.4f", i, msg->q[i]);
        }
    }

    rclcpp::Subscription<dobot_atom::msg::UpperState>::SharedPtr upper_state_sub_;
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<SimpleUpperStateReader>());
    rclcpp::shutdown();
    return 0;
}
```

4.3 RPC examples

```
#include "rpc/algs_rpc_client.h"
#include <iostream>
#include <thread>

int main()
{
    // Initialize the RPC client for communication with the robot's algorithm server.
    Atom::AlgsRpcClient rpc;
    int sw = 0;

    while (true) {
        // Main loop to interact with the user and send commands to the robot.
        if (sw == 0) {
            // Prompt the user to select a command.
            std::cout << "Enter: 1 = FSM, 2 = SetVel." << std::endl;
            std::cin >> sw;
            if (std::cin.fail()) {
                std::cin.clear(); // Clear
error flags
invalid input
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Discard

                std::cout << "Invalid input. Please enter a number." << std::endl;
                continue;
            }
            std::cout << "Test Start..." << std::endl;
        }
        switch (sw) {
            case 1: { // Finite State Machine (FSM) control
                int32_t fsm_id = 0;
                rpc.GetFsmId(fsm_id);
                std::cout << "Enter number for fsm_id: ";
                std::cin >> fsm_id;
                // Set the FSM ID to change the robot's state.
                Atom::RpcErrorCode errorCode = rpc.SetFsmId(fsm_id);
                if (errorCode != Atom::RpcErrorCode::SUCCESS) { std::cout << "SetFsmId
errorCode: " << errorCode << std::endl; }
                std::this_thread::sleep_for(std::chrono::seconds(2));
                break;
            }
            case 2: { // Velocity control
                int32_t fsm_id = 0;
                rpc.GetFsmId(fsm_id);
                // Check if the robot is in a state that allows velocity control.
                if (301 != fsm_id && 302 != fsm_id) {
                    std::cout << "SetVel failed. Please enter fsm 301 or 302." << std::endl;
                    sw = 0;
                    break;
                }
                const float kVel = 0.3; // Desired velocity.
                const float duration = 2.0; // Duration of the velocity command.
                // Send the velocity command to the robot.
                Atom::RpcErrorCode errorCode = rpc.SetVel(kVel, 0.0, 0.0, duration);
            }
        }
    }
}
```

```

        if (errorCode != Atom::RpcErrorCode::SUCCESS) { std::cout << "\033[31mSetVel
failed.\033[0m" << std::endl; }
        sw = 0;
        break;
    }
    default:
        std::cout << "not support" << std::endl;
        sw = 0;
        break;
    }
}
return 0;
}

```

4.4 LiDAR examples

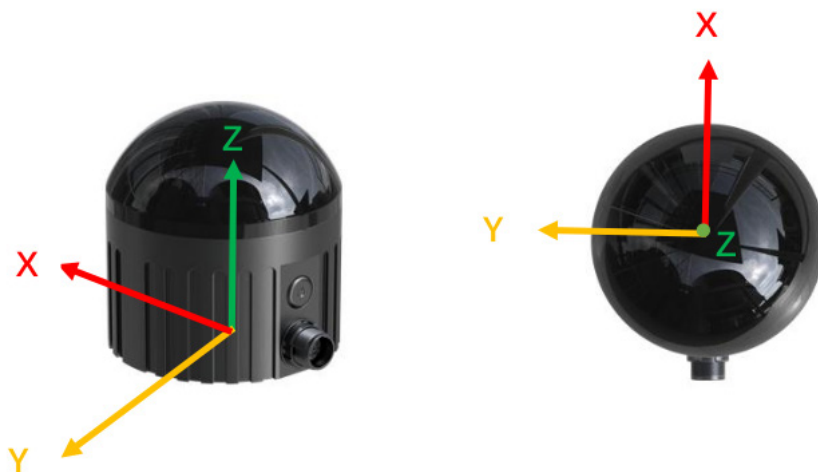
4.4.1 LiDAR introduction

The LiDAR of ATOM is located on the head, and the model is RoboSense Airy.

Airy is a new short-range 3D LiDAR specifically designed by RoboSense for blind-spot elimination. It is mainly used in fields such as robot environment perception, autonomous driving vehicle environment perception, UAV mapping, smart cities, and more.

LiDAR details page	https://www.robosense.ai/IncrementalComponents/Airy
LiDAR resources page	https://www.robosense.ai/resources-161
LiDAR official github address	https://github.com/RoboSense-LiDAR/rslidar_sdk

The coordinate system of Airy is shown in the figure below:



4.4.2 Set parameter

Communication connection

The radar's factory-default IP address is set to: **192.168.8.200**.

Connect the computer to the robot via an Ethernet cable, and set the computer's IP address to be on the 192.168.8.x subnet, with Netmask set to 255.255.255.0 and Gateway set to 192.168.8.1. You can verify communication with the LiDAR by pinging 192.168.8.200.

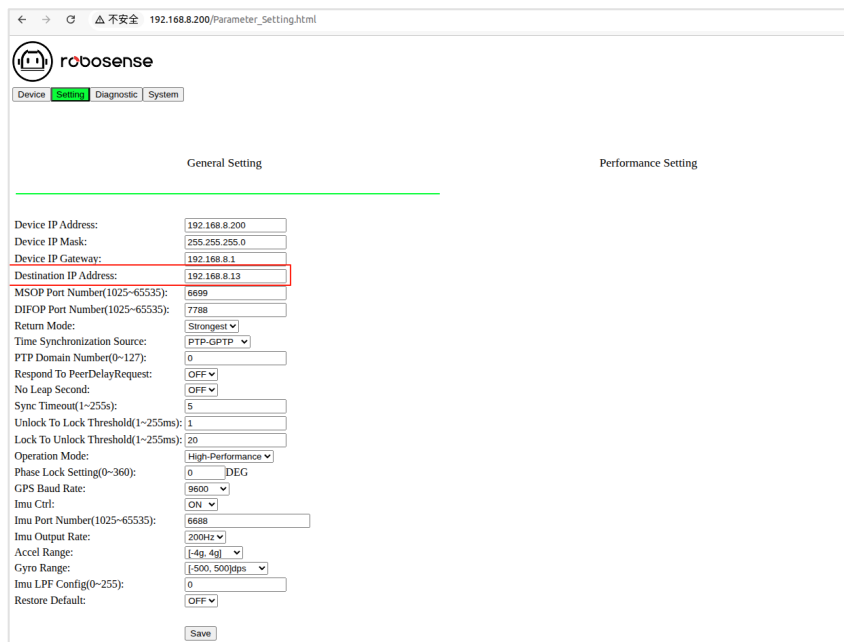
Set target IP

Airy supports parameter configuration, runtime information/status viewing, and firmware upgrades through a web interface.

1. Use a web browser on the computer connected to the LiDAR to access the product's IP address: 192.168.8.200.

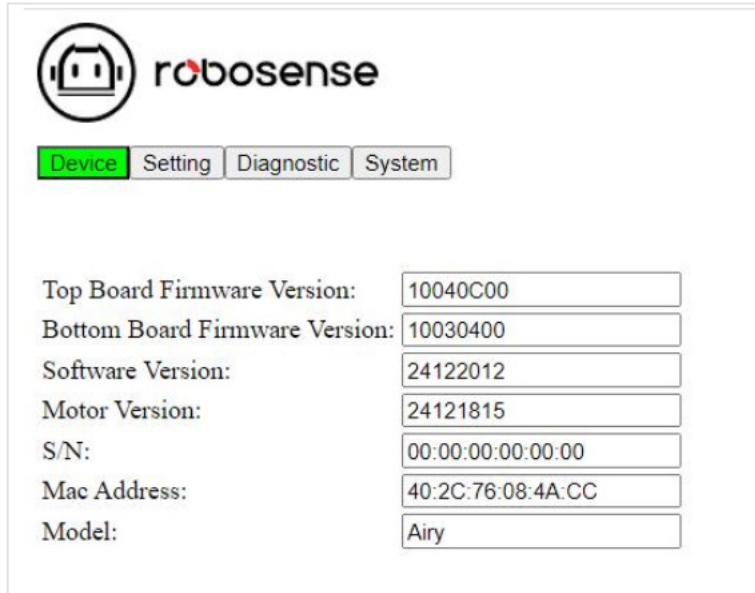


2. Click **Setting**, and change the **Destination IP Address** to the target machine's IP. The default setting is the IP address of ATOM PC2.



Set other parameters

For more parameter settings, please refer to the official [Product Resources](#).



The image shows a web interface for a Robosense device. At the top left is the Robosense logo. Below it are four tabs: 'Device' (highlighted in green), 'Setting', 'Diagnostic', and 'System'. The 'Device' tab is active, displaying a list of device parameters and their values in input fields:

Top Board Firmware Version:	10040C00
Bottom Board Firmware Version:	10030400
Software Version:	24122012
Motor Version:	24121815
S/N:	00:00:00:00:00:00
Mac Address:	40:2C:76:08:4A:CC
Model:	Airy

4.4.3 Get data

NOTICE

During data acquisition, make sure the IP address of the computer you are using matches the LiDAR's target IP.

Get data from the host

Download the host software

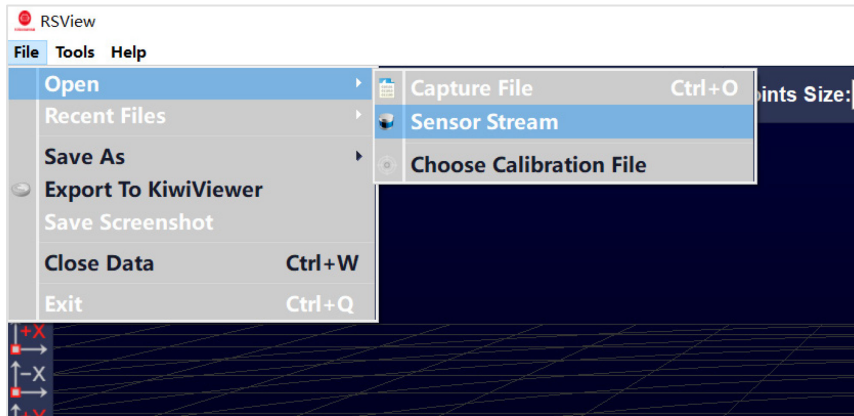
Download the host software RSView from [LiDAR Resources](#) page.

- **Windows:** After downloading, extract the package and go to the bin directory, then click RsView.exe to open the host software.
- **Ubuntu:** After downloading, extract the package to the home directory.

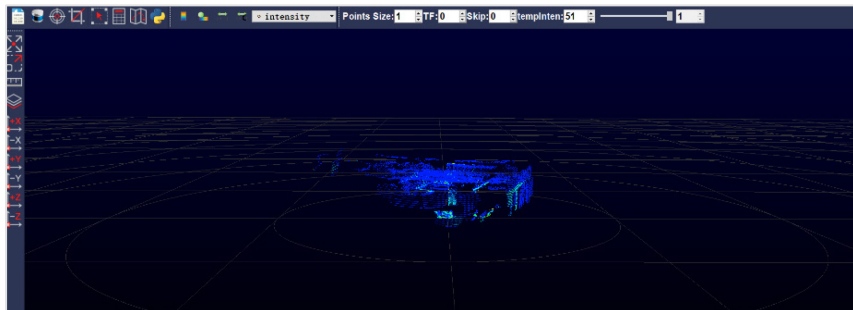
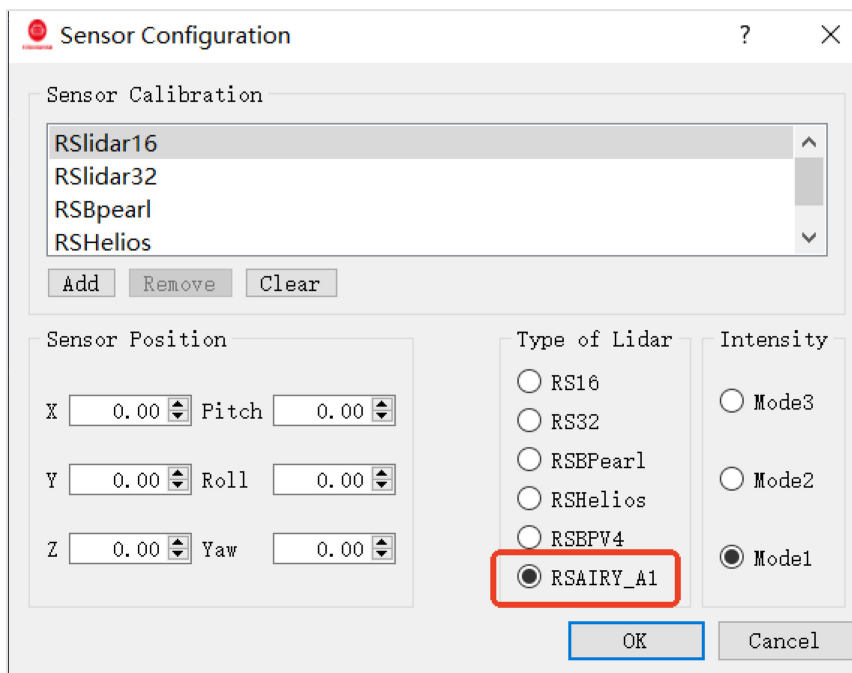
```
cd rsview
./run_rsview.sh # Enter password
```

Data acquisition process

1. Click **File > Open > Sensor Stream** in sequence.



2. In the **Type of LiDAR** field, check **RSAIRY_A1**, then click "OK" to view the point cloud.



Acquiring data via SDK

Function introduction

rslidar_sdk is RoboSense's LiDAR driver software package for the Ubuntu environment, which includes:

- LiDAR driver core [rs_driver](#)
- ROS extension features
- ROS2 extension features

Repository address: https://github.com/RoboSense-LiDAR/rslidar_sdk

Acquiring data via rs_driver

rs_driver is RoboSense's LiDAR driver library, which is included in **rslidar_sdk** (**rslidar_sdk/src/rs_driver**). It can also be downloaded separately. **rs_driver** provides a development library and sample code for acquiring LiDAR data.

- **Download and install rs_driver separately**

```
git clone https://github.com/RoboSense-LiDAR/rs\_driver.git
sudo apt-get install libpcap-dev libeigen3-dev libboost-dev libpcl-dev # Install dependencies
(optional)
rs_driver
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

- **Get data demo**

rs_driver/demo/demo_online.cpp provides an example routine for acquiring LiDAR data via **rs_driver**. To compile the demo, you need to enable the compilation option **COMPILE_DEMOS=ON** in Step 5 of the compilation process.

```
cmake .. -DCOMPILE_DEMOS=ON
make -j4
```

After compilation, run **demo/demo_online** in the build directory to see the following output in the terminal:

```
./demo/demo_online
```

```
num_blks_split: 1
-----
RoboSense Transform Parameters
x: 0
y: 0
z: 0
roll: 0
pitch: 0
yaw: 0
-----
RoboSense Lidar-Driver Linux online demo start.....
msg: 0 point cloud size: 31872
msg: 1 point cloud size: 86400
msg: 2 point cloud size: 86400
msg: 3 point cloud size: 86400
msg: 4 point cloud size: 86400
msg: 5 point cloud size: 86400
msg: 6 point cloud size: 86400
msg: 7 point cloud size: 86400
msg: 8 point cloud size: 86304
msg: 9 point cloud size: 86112
msg: 10 point cloud size: 86400
```

Acquiring data via the ROS driver

rslidar_sdk automatically detects whether the current environment is ROS1 or ROS2 and adjusts the compilation method accordingly. It is not recommended to compile rslidar_sdk on a system with both ROS1 and ROS2 installed.

Place the rslidar_sdk project into the src folder of the corresponding workspace, and navigate to the workspace directory.

- **ROS1**

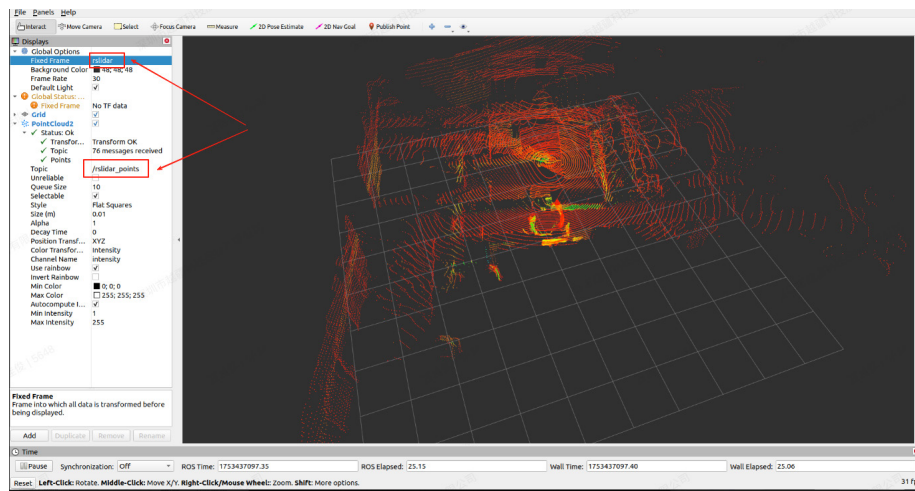
```
catkin_make
source devel/setup.bash
roslaunch rslidar_sdk start.launch
```

- **ROS2**

```
colcon build
source install/setup.bash
ros2 launch rslidar_sdk start.py
```

After running, the LiDAR node will publish the /rslidar_points and /rslidar_imu_data topics.

The point cloud data can be viewed in Rviz.



4.5 Depth camera demo

4.5.1 Depth camera introduction

ATOM has three depth cameras: the head is equipped with a realsense D455FC, the waist with a realsense D435CB, and the wrist with a realsense D405.

Official product page	https://realsenseai.com/stereo-depth-cameras/
Official document	https://dev.realsenseai.com/docs/docs-get-started
Official github address	https://github.com/IntelRealSense

Camera parameters are shown below:

Model	D435CB	D455FC	D405C
Applicable environments	Indoor/Outdoor	Indoor/Outdoor	Indoor/Outdoor
Image sensor technology	Global shutter	Global shutter	Global shutter
Stereo baseline	50 mm	95 mm	18 mm
Depth field of view (H × V)	87° × 58°	87° × 58°	87° × 58°
Max depth resolution @frame rate	1280×720@30fps	1280×720@30fps	1280×720@30fps
Max frame rate @depth resolution	90fps@848×480	90fps@848×480	90fps@848×480
Z accuracy error	<2% (at 2m)	<2% (at 4m)	±2% (at 0.5m)
RGB sensor technology	Rolling shutter	Global shutter	Global shutter
RGB resolution @frame rate	1920×1080@30fps	1280×800@30fps	1280×720@30fps
RGB field of view (H × V)	69° × 42°	90° × 65°	87° × 58°
Inertial measurement unit	Null	Available	Null
Minimum distance at max resolution	~28cm	~52cm	7cm @ 480p
Ideal operating range	0.3 - 3m	0.6 - 6m	7 - 50 cm
Main components	D430 Module, D4/D4V3 Board, RGB Sensor	D450 Module, D4V3 Board	D401 Module, D4/D4V3 Board
Size(mm)	90×25×25	124×29×26	42×42×23 (Peripheral) 36.5×19.4×10.5 (Module)
Interface	Type - C	Type - C	USB 2 USB 3.1
Remark	Null	With 750nm near-infrared filter	Null

4.5.2 Get data

Acquiring data via the host software

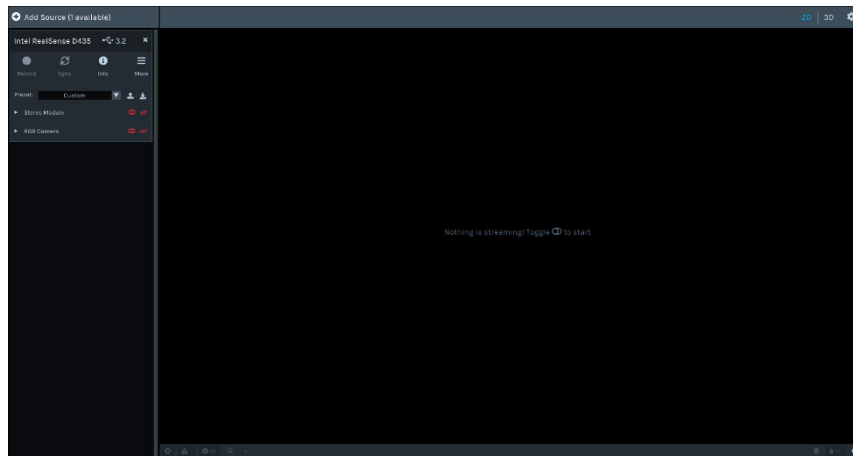
Download the host software

Windows: Download [Intel.RealSense.Viewer.exe](#)

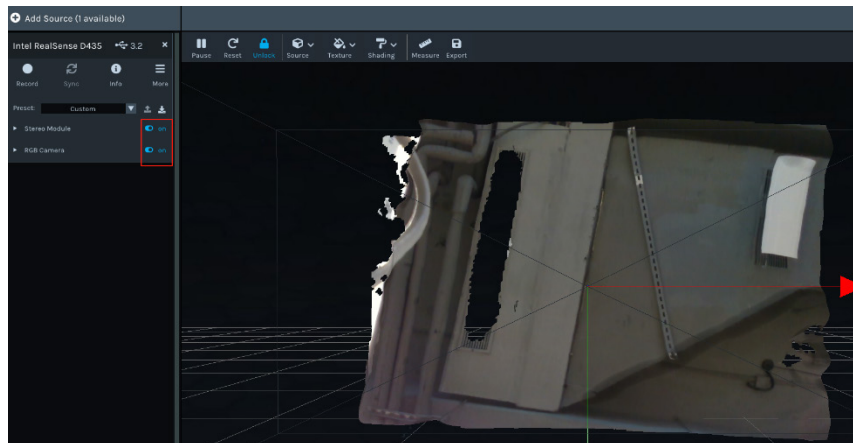
Linux: Refer to the [Official Tutorial](#)

Data acquisition process

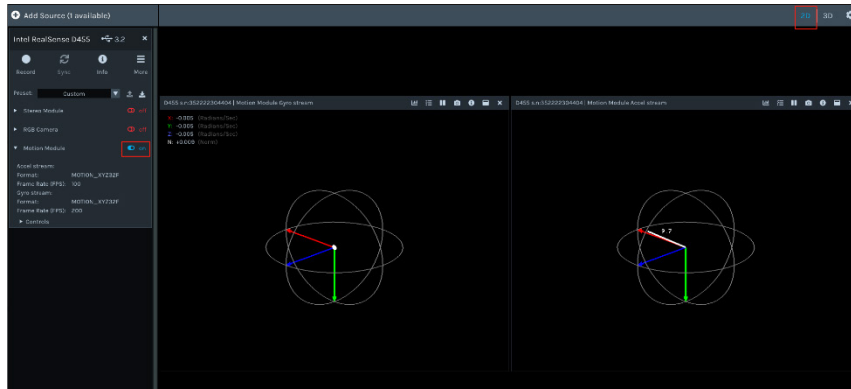
1. On Windows, double-click the .exe file to open the host software; on Linux, type `realsense-viewer` in the terminal to launch it.



2. After turning on the corresponding module switch, you can view the data.



3. Since the D455FC integrates an inertial sensor, it also provides inertial data.



Acquiring data via ROS

Official ROS driver repository

<https://github.com/IntelRealSense/realsense-ros>

For details, refer to the readme. In general, it is recommended to use the SDK for development.

Acquiring data via SDK

Official SDK repository

<https://github.com/IntelRealSense/librealsense>

Development example reference

<https://github.com/IntelRealSense/librealsense/tree/master/examples>

4.6 Stereo camera example

You can obtain RGB images from the stereo camera through the OpenCV general camera interface. Since the USB index of the camera may change after the robot is powered on again, you need to obtain the USB index of the stereo camera first. Python code example:

Get the USB index of the stereo camera

```
import subprocess

import cv2

import time

def get_camera_usb_info(cam_index):
    """Get USB information of the camera"""

    try:
        # Get USB information for the video device

        result = subprocess.run(
            f"udevadm info -q property -n /dev/video{cam_index} | grep -E 'ID_VENDOR|ID_MODEL|ID_VENDOR_ID|ID_MODEL_ID'",
            shell=True,
            capture_output=True,
            text=True
        )

        info = {}

        for line in result.stdout.strip().split("\n"):

            if '=' in line:
                key, value = line.split('=', 1)
                info[key] = value

        return info
    except:
        return {}

def is_intel_camera(cam_index):
    """Check if it is an Intel camera"""

    info = get_camera_usb_info(cam_index)

    if 'ID_VENDOR_ID' in info and info['ID_VENDOR_ID'] == '8086':
        return True

    if 'ID_VENDOR' in info and 'Intel' in info['ID_VENDOR']:
        return True

    if 'ID_MODEL' in info and 'RealSense' in info['ID_MODEL']:
        return True

    return False
```

```

def found_top_camera(cam_id, show):
    """Camera detection function"""

    if is_intel_camera(cam_id):
        print(f"Intel pass: {cam_id}")
        return False

    cap = cv2.VideoCapture(cam_id, cv2.CAP_V4L2)
    try:
        cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'))
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280 * 2)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
        cap.set(cv2.CAP_PROP_FPS, 60)

        time.sleep(0.1)
        ret, frame = cap.read()
        if not ret:
            return False

        if show:
            cam_id_str = f"camera_id:{cam_id}"
            cv2.putText(frame, cam_id_str, (400, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),
2)            cv2.imshow("cam", frame)
            cv2.waitKey(100)

        return frame.shape[0] == 720 and frame.shape[1] == 2560 and frame.shape[2] == 3
    finally:
        cap.release()
        if show:
            cv2.destroyAllWindows()
            time.sleep(0.1)

def search_top_cameraID():
    """
    Function: Automatically find the ID of the head stereo camera

    Parameters: None

    Returns: bool, Whether the search succeeded

            video_index, the found index
    """

    video_index = 0

    res = False

    while video_index < 35:
        print("test video_index:", video_index)
        try:
            res = found_top_camera(video_index, show=False)
            if res:
                break
        except:
            pass
        video_index += 1
    if res:
        print("[Successful]:Found the top camera index:", video_index)
        return True, video_index
    else:

```

```
print("[Fail]:Head binocular camera not found, please check if it is connected correctly")
return False,-1
```

Read camera image

```
def run_camera(cam_id):
    cap = cv2.VideoCapture(cam_id, cv2.CAP_V4L2)
    cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'))
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280*2)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    cap.set(cv2.CAP_PROP_FPS, 60)
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                print("[Error]:Head(top) binocular camera reading error, please check whether the
connection or ID setting is correct!")
                return
            time.sleep(0.001)

    except Exception as e:
        print(e)
        print(
            "[Error]:Head(top) binocular camera reading error, please check if the ID setting is correct!")

if __name__ == '__main__':
    res,id = search_top_cameraID()
    run_camera(id)
```

5. Recording and Playback of Offline Trajectories

5.1 Preface

This chapter introduces the offline trajectory recording and playback methods for the ATOM humanoid robot.

Environment and role description	- OS: Ubuntu - Host: Local computer used for running program logic and data logging - Slave: The built-in PC1 on the ATOM robot, used for motor control and sensor data acquisition
Connection and communication method	- A direct Ethernet connection is used between the local computer and the robot - Motion control and data exchange are implemented based on the DDS publish/subscribe mechanism

5.2 Compressed SDK package

Please contact DOBOT technical support to obtain the SDK package **trajectory.tar.gz**.

5.3 Configuration and compilation

1. Connect your computer directly to the ATOM robot using an Ethernet cable. Open a terminal and run the `ip addr list` command to check the network interface name connected to the robot (as shown in the figure below, the detected interface name is "enp88s0").

```

yu@yu: ~
yu@yu: ~ 116x26
yu@yu:~$ ip addr list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp88s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 9c:7d:11:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.255 scope global dynamic noprefixroute enp88s0
        valid_lft 42293sec preferred_lft 0
    inet6 fe80::9c7d:11ff:fe00:0000 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 8c:5e:0c:10:10:31 brd ff:ff:ff:ff:ff:ff
    inet 10.10.29.2 scope global dynamic noprefixroute wlp0s20f3
        valid_lft 27439sec preferred_lft 27439sec
    inet6 fe80::41c7:c7b9:b705:55cd scope link noprefixroute
        valid_lft forever preferred_lft forever
yu@yu:~$

```

2. Extract the SDK package **trajectory.tar.gz**, and open the configuration file `cyclonedds.xml` located in the `~/trajectory/install/sdk/config` directory. Replace "eth0" with the name of the network interface connected to the robot (as shown in the figure below, the interface name is "enp88s0").

```

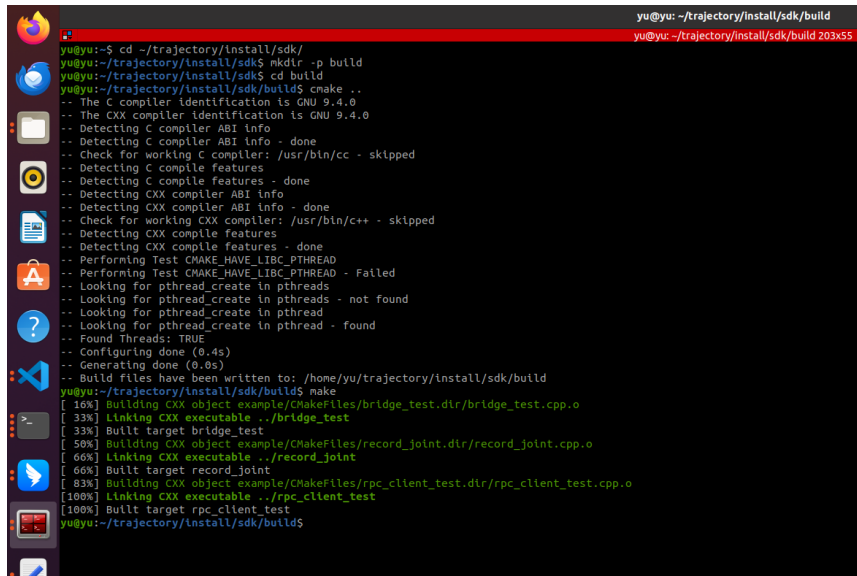
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <cyclonedds xmlns="https://dds.la/config"
3 xmlns:sstt="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="https://dds.la/config https://raw.githubusercontent.com/eclipse-cyclonedds/cyclonedds/master/etc/cyclonedds.xsd">
4 <domain id="any">
5 <general>
6 <interfaces>
7 <!-- 将 'eth0' 替换为连接机器人的网卡名称。例如在Linux上使用 'ifconfig' 命令查找正确的接口名称。 -->
8 <NetworkInterface name="enp88s0" priority="default" multicast="true" />
9 </NetworkInterface>
10 </interfaces>
11 <allowmulticast-default/>
12 </general>
13 </cyclonedds>
14
15 <!-- 将 'eth0' 替换为连接机器人的网卡名称。例 -->
16 <NetworkInterface name="enp88s0" priority="default"
17 multicast="true" />
18 </NetworkInterface>
19 </interfaces>
20 </cyclonedds>

```

3. Open a terminal and run the following commands in sequence.

```
cd ~/trajectory/install/sdk/  
mkdir build  
cd build  
cmake ..  
make
```

The running result is as follows:



```
yu@yu: ~/trajectory/install/sdk/build  
yu@yu:~/trajectory/install/sdk/build$ cd ..  
yu@yu:~/trajectory/install/sdk$ mkdir -p build  
yu@yu:~/trajectory/install/sdk$ cd build  
yu@yu:~/trajectory/install/sdk/build$ cmake ..  
-- The C compiler identification is GNU 9.4.0  
-- The CXX compiler identification is GNU 9.4.0  
-- Detecting C compiler ABI info  
-- Detecting C compiler ABI info - done  
-- Check for working C compiler: /usr/bin/cc - skipped  
-- Detecting C compile features  
-- Detecting C compile features - done  
-- Detecting CXX compiler ABI info  
-- Detecting CXX compiler ABI info - done  
-- Check for working CXX compiler: /usr/bin/c++ - skipped  
-- Detecting CXX compile features  
-- Detecting CXX compile features - done  
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD  
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Failed  
-- Looking for pthread_create in pthreads - not found  
-- Looking for pthread_create in pthread  
-- Looking for pthread_create in pthread - found  
-- Found Threads: TRUE  
-- Configuring done (0.4s)  
-- Generating done (0.0s)  
-- Build files have been written to: /home/yu/trajectory/install/sdk/build  
yu@yu:~/trajectory/install/sdk/build$ make  
[ 16%] Building CXX object example/CMakeFiles/bridge_test.dir/bridge_test.cpp.o  
[ 33%] Linking CXX executable ../bridge_test  
[ 33%] Built target bridge_test  
[ 50%] Building CXX object example/CMakeFiles/record_joint.dir/record_joint.cpp.o  
[ 66%] Linking CXX executable ../record_joint  
[ 66%] Built target record_joint  
[ 83%] Building CXX object example/CMakeFiles/rpc_client_test.dir/rpc_client_test.cpp.o  
[100%] Linking CXX executable ../rpc_client_test  
[100%] Built target rpc_client_test  
yu@yu:~/trajectory/install/sdk/build$
```

5.4 Recording offline trajectories

1. Switch the robot to debugging mode

First, press the LB + A keys on the remote controller to put the robot into damping mode, then press LT + RT to switch the robot to debugging mode.

2. Start offline trajectory recording

```
cd ~/trajectory/install/sdk/build/  
ls  
./record_joint
```

The running result is as follows:

```

yu@yu: ~/trajectory/install/sdk/build
yu@yu:~/trajectory/install/sdk/build$ ls
bridge_test      cmake_install.cmake  Makefile
CMakeCache.txt   example              record_joint
CMakeFiles       joint_angles.txt     rpc_client_test
yu@yu:~/trajectory/install/sdk/build$ ./record_joint
CYCLONEDDS_URI set to config file: ../config/cyclonedds.xml
请查看配置文件说明, 确保配置文件中的网络接口名称与连接机器人的网卡名称一致。
ListenOperation [0]: create reader successfully
ListenOperation [1]: create reader successfully
ListenOperation [2]: create reader successfully
ListenOperation is running
start time: 5, end time: 20, current time: 5.01
start time: 5, end time: 20, current time: 5.02
start time: 5, end time: 20, current time: 5.03
start time: 5, end time: 20, current time: 5.04
start time: 5, end time: 20, current time: 5.05
start time: 5, end time: 20, current time: 5.06
start time: 5, end time: 20, current time: 5.07
start time: 5, end time: 20, current time: 5.08
start time: 5, end time: 20, current time: 5.09
start time: 5, end time: 20, current time: 5.1
start time: 5, end time: 20, current time: 5.11
start time: 5, end time: 20, current time: 5.12
start time: 5, end time: 20, current time: 5.13
start time: 5, end time: 20, current time: 5.14
start time: 5, end time: 20, current time: 5.15
start time: 5, end time: 20, current time: 5.16
start time: 5, end time: 20, current time: 5.17

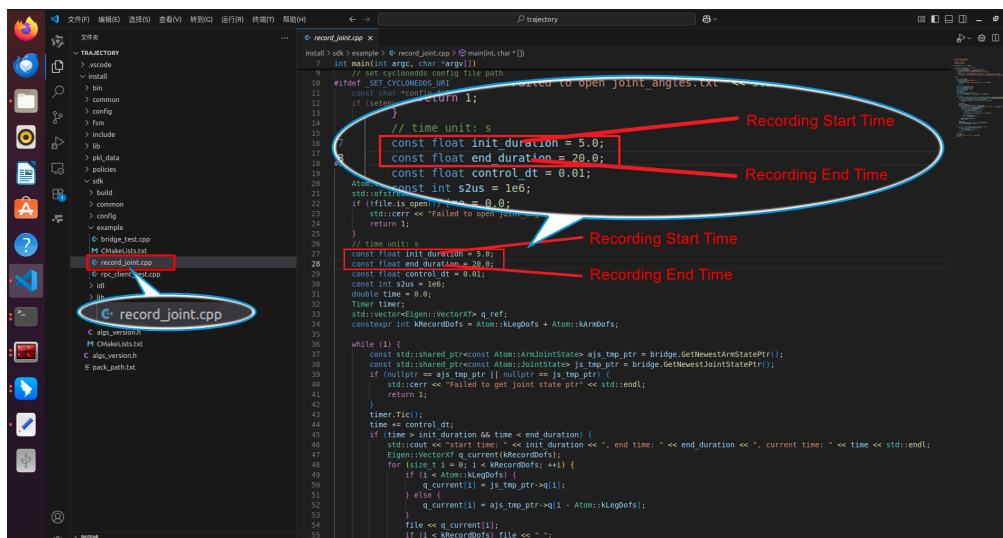
```

NOTE

After starting offline trajectory recording, you can drag the robot to record the desired motion trajectory.

3. Modify the recording time

- Extract the SDK package **trajectory.tar.gz**, and open the configuration file **cyclonedds.xml** located in the **~/trajectory/install/sdk/config** directory.
- Locate the corresponding position as shown in the figure below, and modify the recording time according to the action duration.



- After the modification is complete, run the following command to recompile.

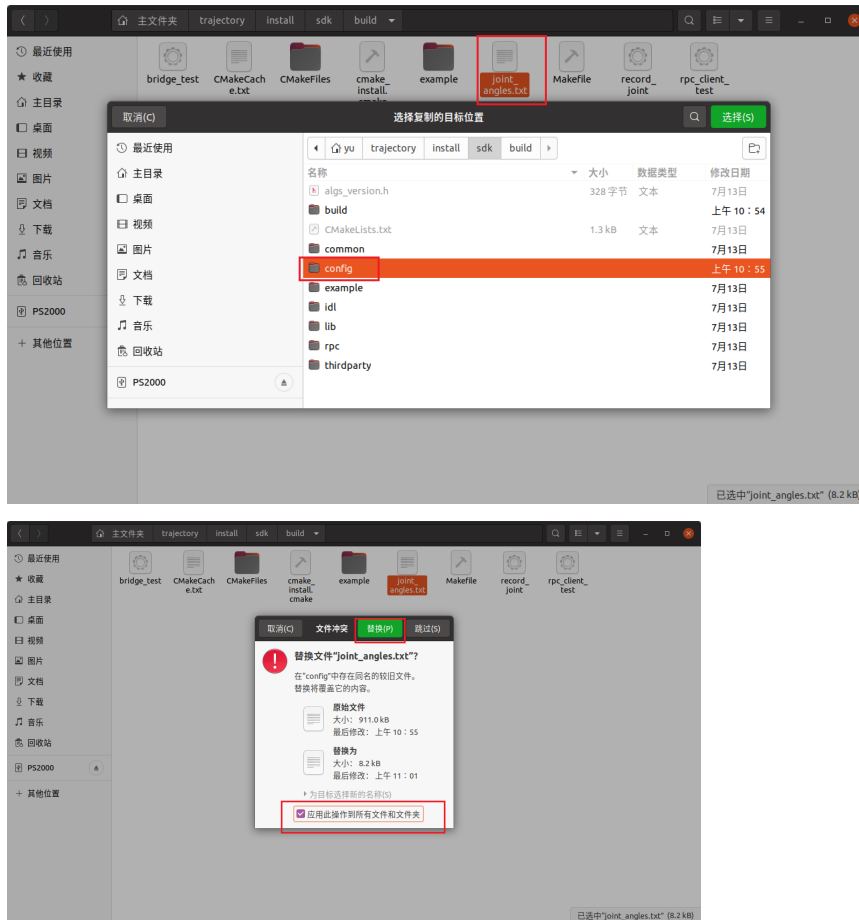
```

cd ~/trajectory/install/sdk/build/
cmake ..
make

```

5.5 Offline trajectories playback

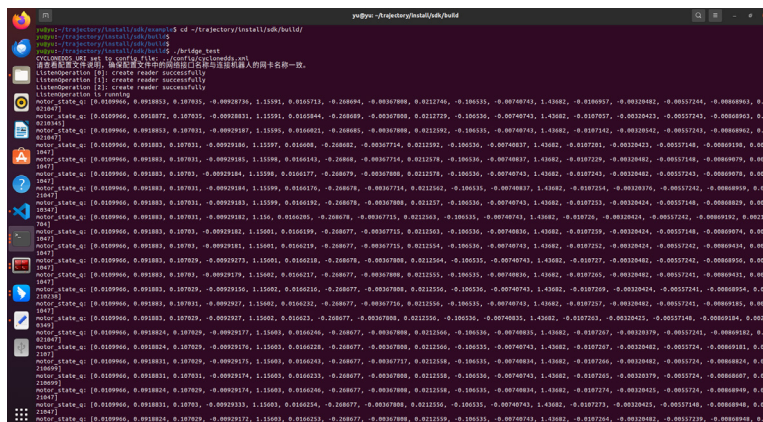
1. Copy `joint_angles.txt` from the build directory to the config directory to replace the offline trajectory.



2. Then run the following command to play back the offline trajectory.

```
cd ~/trajectory/install/sdk/build/
./bridge_test
```

The running result is as follows:



3. After offline trajectory recording is complete, please back up the `joint_angles.txt` file to prevent it from being overwritten.

6. TCP Communication

6.1 Preface

This chapter introduces how to establish TCP communication between the humanoid robot ATOM and external devices (such as a PLC).

Environment description	- OS: Windows - Access method: Connect to the ATOM's built-in PC2 (Ubuntu system) via Windows remote desktop, and run the relevant programs on PC2
Example description	Debugging tool: Using "TCP debug assistant" as an example to demonstrate the TCP communication process with PC2 Development method: Both Python and C++ are supported. The following example uses C++

NOTE

This chapter only covers the implementation method of TCP communication and does not include the specific implementation of ATOM motion control. For motion control, please refer to other sections in the *SDK Application Development Guide*.

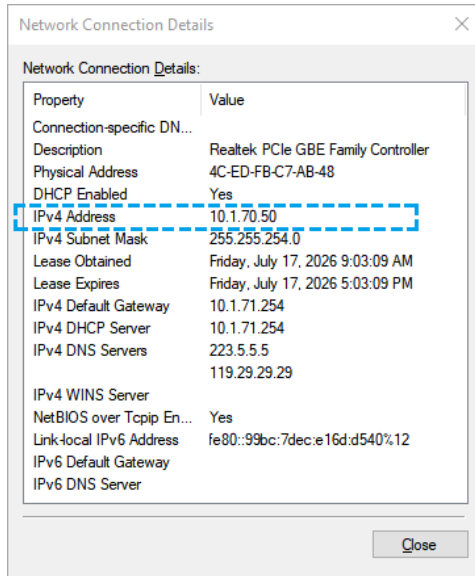
6.2 Connect to ATOM

Both computers and other devices can connect to the ATOM via WiFi or Ethernet cable. The following example uses a WiFi connection.

1. Connect to the ATOM WiFi.

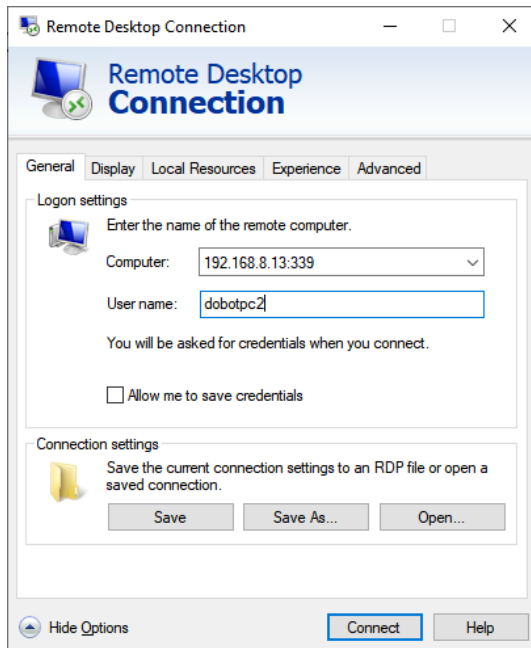
ATOM WiFi	WiFi name: ATOM-xxxx-xxxx (device serial number) WiFi password: 12345678
ATOM IP assignment	PC1: 192.168.8.234 PC2: 192.168.8.13 Router: 192.168.8.1

- After the computer is successfully connected to the ATOM WiFi, the router automatically assigns an IP address within the same subnet (192.168.8.xx) to the computer.

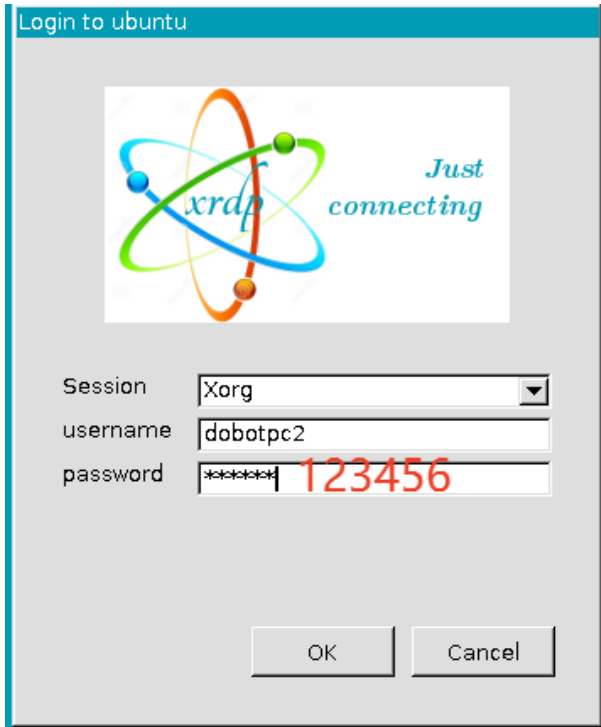


6.3 Remote login to PC2

- Open Windows "Remote Desktop Connection" as described below.
 - Press Win + R at the same time, type mstsc, and press Enter.
 - Search for "Remote Desktop" directly
- Enter the IP address and port of ATOM PC2 in the format: 192.168.8.13:3390.
- Enter the ATOM PC2 username and click "Connect".



4. Select **Xorg** for the **Session**, enter the password (123456), and click "OK". After successful login, you will enter the remote desktop of ATOM PC2.



6.4 Implementing TCP communication

1. Create a new C++ script in the home directory of ATOM PC2 to write the socket communication script, or add it to the SDK examples. This section uses the approach of adding ATOM to the SDK examples as the client (ATOM can also act as the server). You may refer to the following code or write your own.

```
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdexcept>

// TCP connection mode enumeration
enum class TcpMode {
    SERVER, // Server mode
    CLIENT  // Client mode
};

// TCP communication tool class (supports both server and client modes)
class TcpConnection {
private:
    TcpMode mode; // Connection mode
    int server_fd; // Server socket descriptor (server mode only)
    int socket_fd; // Communication socket descriptor (used uniformly)
```

```

struct sockaddr_in address;    // Address structure
int addrlen;
bool is_connected;           // Connection status flag

public:
    // Unified constructor: switch between client/server mode via the first parameter
    // is_client: //    true for client mode, false for server mode
    // ip: Target server IP in client mode; can be ignored in server mode (uses INADDR_ANY)
    // port: Port No.
    TcpConnection(bool is_client, const std::string& ip, int port)
        : mode(is_client ? TcpMode::CLIENT : TcpMode::SERVER),
          is_connected(false),
          server_fd(-1),
          socket_fd(-1) {

        if (is_client) {
            // Client mode
            // Create socket
            if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                throw std::runtime_error("Socket creation failed");
            }

            address.sin_family = AF_INET;
            address.sin_port = htons(port);

            // Convert IP address
            if (inet_pton(AF_INET, ip.c_str(), &address.sin_addr) <= 0) {
                throw std::runtime_error("Invalid IP address: " + ip);
            }

            std::cout << "[Client mode] Connecting to server " << ip << ":" << port << "..." <<
std::endl;

            // Auto-connect to server
            if (connect(socket_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
                throw std::runtime_error("Failed to connect to server");
            }
            is_connected = true;
            std::cout << "[Client mode] Successfully connected to server" << std::endl;
        } else {
            // Server mode
            // Create socket
            if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
                throw std::runtime_error("Socket creation failed");
            }

            // Set address reuse
            int opt = 1;
            if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
                throw std::runtime_error("setsockopt SO_REUSEADDR 失败");
            }
            if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEPORT, &opt, sizeof(opt))) {
                throw std::runtime_error("setsockopt SO_REUSEPORT 失败");
            }
        }
    }

```

```

    }

    address.sin_family = AF_INET;
    // Server mode: if IP is empty or "0.0.0.0", use INADDR_ANY to listen on all network
interfaces
    if (ip.empty() || ip == "0.0.0.0") {
        address.sin_addr.s_addr = INADDR_ANY;
    } else {
        // You can also bind to a specific IP address
        if (inet_pton(AF_INET, ip.c_str(), &address.sin_addr) <= 0) {
            throw std::runtime_error("Invalid IP address: " + ip);
        }
    }
    address.sin_port = htons(port);

    // Bind to port
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        throw std::runtime_error("Port binding failed");
    }

    // Listen for connections (max 1 client in queue)
    if (listen(server_fd, 1) < 0) {
        throw std::runtime_error("Listening failed");
    }

    addrlen = sizeof(address);
    std::cout << "[Server mode] TCP server started on port " << port << ", waiting for client
connection..." << std::endl;

    // Automatically wait for client connection
    if ((socket_fd = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen))
< 0) {
        throw std::runtime_error("Client connection failed");
    }
    is_connected = true;
    std::cout << "[Server mode] Client connected, IP: " << inet_ntoa(address.sin_addr) <<
std::endl;
}

// Server mode: wait for client connection (blocking)
void acceptClient() {
    if (mode != TcpMode::SERVER) {
        throw std::runtime_error("acceptClient() can only be used in server mode");
    }
    if ((socket_fd = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
        throw std::runtime_error("Client connection failed");
    }
    is_connected = true;
    std::cout << "[Server mode] Client connected, IP: " << inet_ntoa(address.sin_addr) <<
std::endl;
}

// Client mode: connect to server (blocking)
void connectToServer() {

```

```

        if (mode != TcpMode::CLIENT) {
            throw std::runtime_error("connectToServer() can only be used in client mode");
        }
        if (connect(socket_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
            throw std::runtime_error("Failed to connect to server");
        }
        is_connected = true;
        std::cout << "[Client mode] Successfully connected to server" << std::endl;
    }

    // Send message (common to both modes)
    void sendMessage(const std::string& msg) {
        if (!is_connected) {
            throw std::runtime_error("No connection established, cannot send message");
        }
        ssize_t sent = send(socket_fd, msg.c_str(), msg.length(), 0);
        if (sent < 0) {
            is_connected = false;
            throw std::runtime_error("Failed to send message");
        }
        std::cout << "[Send] " << msg << std::endl;
    }

    // Receive message (common to both modes, blocks indefinitely)
    std::string receiveMessage() {
        if (!is_connected) {
            throw std::runtime_error("No connection established, cannot receive message");
        }

        char buffer[1024] = {0};
        ssize_t valread = read(socket_fd, buffer, 1024);
        if (valread <= 0) {
            is_connected = false;
            if (valread == 0) {
                throw std::runtime_error("The other side has actively disconnected");
            } else {
                throw std::runtime_error("Receive failed (connection error)");
            }
        }

        std::string msg(buffer, valread);
        std::cout << "[Receive] " << msg << std::endl;
        return msg;
    }

    Please check the connection status
    bool isConnected() const {
        return is_connected;
    }

    // Get the current mode
    TcpMode getMode() const {
        return mode;
    }

```

```

// Close connection
~TcpConnection() {
    if (is_connected && socket_fd >= 0) {
        close(socket_fd);
    }
    if (server_fd >= 0)
        close(server_fd);
}
std::cout << "TCP connection closed" << std::endl;

};

////////////////////Main////////////////////////////////////

int main(int argc, char *argv[])
{
    // Parse command line arguments
    // Usage:
    //   Server mode: //      .TCP_test server <ip> <port>  (ip can be empty or 0.0.0.0 to listen
on all interfaces)
    // Client mode: .TCP_test client <server_ip> <port>

    TcpConnection* tcp = nullptr;
    std::string ip = "192.168.8.139";      // IP address
    int port = 8888;                      // Port number

    try {
        tcp = new TcpConnection(true, ip, port);
    } catch (const std::exception& e) {
        std::cerr << "TCP initialization failed: " << e.what() << std::endl;
        if (tcp) delete tcp;
        return 1;
    }

    // Main loop
    while (1) {
        try {
            // Receive message
            std::cout << "Waiting to receive message..." << std::endl;
            std::string received_msg = tcp->receiveMessage();
            std::cout << "Received message: " << received_msg << std::endl;

            // Send acknowledgment message
            tcp->sendMessage("ATOM sends a message");

        } catch (const std::exception& e) {
            std::cerr << "TCP communication error: " << e.what() << std::endl;

            // Reconnect based on mode when connection is lost

```

```

        if (!tcp->isConnected()) {
            if (tcp->getMode() == TcpMode::SERVER) {
                sstd::cout << "Waiting for client to reconnect..." << std::endl;
                try {
                    tcp->acceptClient();
                } catch (const std::exception& e2) {
                    std::cerr << "Reconnection failed: " << e2.what() << std::endl;
                    std::this_thread::sleep_for(std::chrono::seconds(1));
                }
            } else {
                // Client mode: recreate connection
                std::cout << "Attempting to reconnect to server " << ip << "." << port << "..."
                << std::endl;

                delete tcp;
                tcp = nullptr;
                std::this_thread::sleep_for(std::chrono::seconds(1));

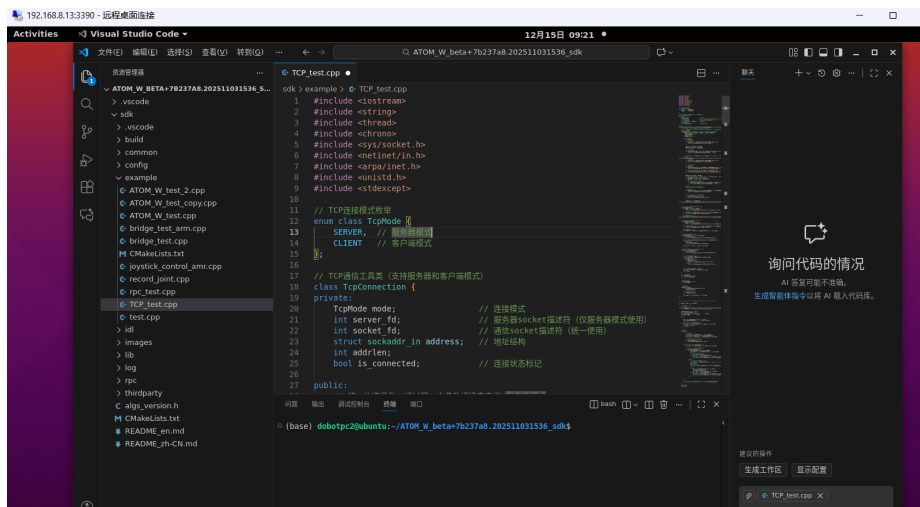
                try {
                    tcp = new TcpConnection(true, ip, port);
                    tcp->connectToServer();
                    std::cout << "Reconnection successful!" << std::endl;
                } catch (const std::exception& e2) {
                    std::cerr << "Reconnection failed: " << e2.what() << std::endl;
                    std::this_thread::sleep_for(std::chrono::seconds(2));
                }
            }

            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }

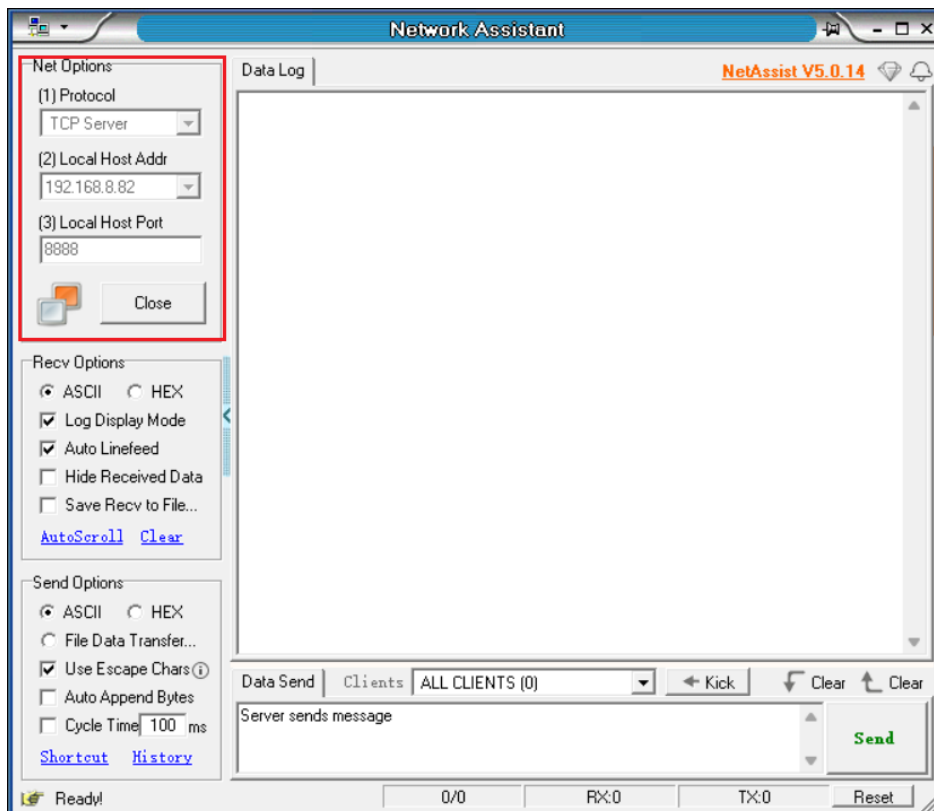
        delete tcp;
        return 0;
    }
}

```

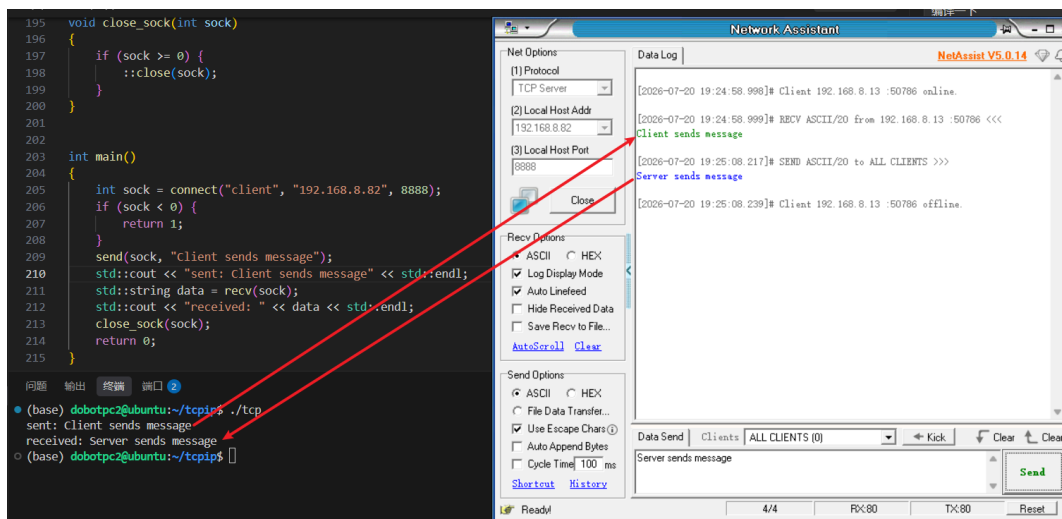
The running result is as follows:



- On the computer, open the TCP debug assistant and set the protocol type to **TCP Server**.



- Run the C++ script on ATOM PC2 to establish a communication connection with the TCP Debug Assistant, and TCP communication will then be achieved.



7. ATOM Learning

ATOM Learning is a comprehensive reinforcement learning framework designed for the ATOM robot. Built on NVIDIA Isaac Gym, it covers the complete workflow from environment setup and policy training to Sim2Sim validation. It also enhances policy robustness and deployment readiness through cross-engine evaluation (Isaac Gym → MuJoCo).

Github address for **ATOM Learning**: <https://github.com/embodied-dobot/atom-locomotion-training>

7.1 Features

- **End-to-end RL workflow**

Complete training framework: PPO implementation optimized for humanoid robot motion

Model export: Directly export models in PyTorch and ONNX formats for deployment

- **Sim2Sim validation**

Cross-engine testing: Transfer policies from Isaac Gym to MuJoCo

Performance benchmarking: Comparative analysis across physics engines

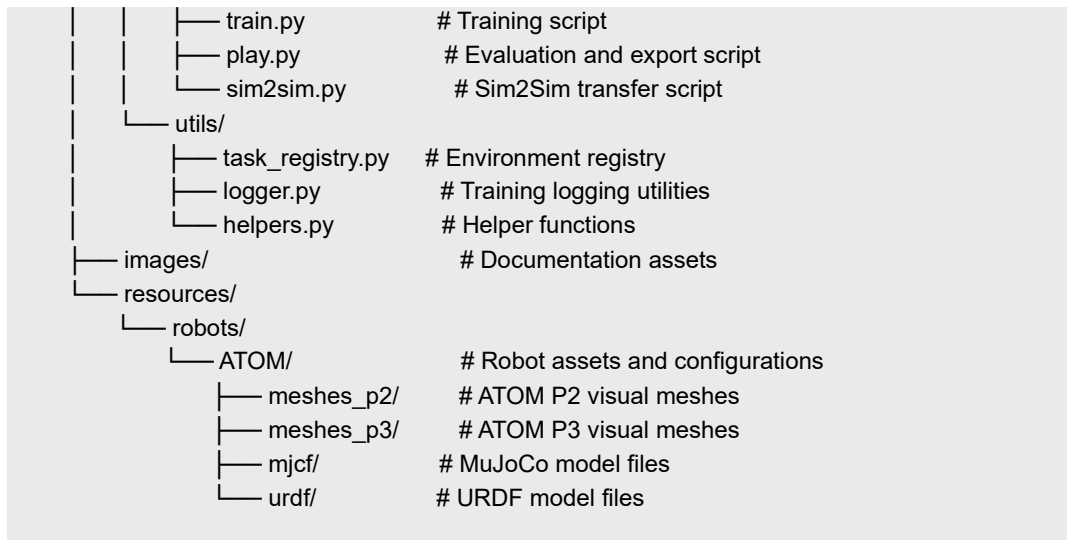
- **Specific ATOM optimization**

Custom reward design: Humanoid-specific rewards tailored for stable motion

Deployment ready: Seamless integration with the ATOM deployment workflow

7.2 Project structure

```
atom-locomotion-training/
├── README.md                # English documentation
├── README.zh.md             # Chinese documentation
├── environment.yml           # Conda environment configuration
├── setup.py                  # Python package setup
├── humanoid/
│   ├── __init__.py
│   └── algo/
│       ├── ppo/              # PPO algorithm implementation
│       │   ├── actor_critic.py # Policy network architecture
│       │   ├── ppo.py          # PPO training logic
│       │   └── rollout_storage.py # Experience buffer
│       └── vec_env.py          # Vectorized environment wrapper
│   └── envs/
│       ├── base/              # Base robot environment classes
│       │   ├── legged_robot.py # Core legged robot environment
│       │   ├── legged_robot_config.py # Base configuration
│       │   └── terrain.py       # Terrain generation utilities
│       └── custom/             # ATOM-specific environments
│           ├── atom_p2.py       # ATOM P2 environment
│           └── atom_p3.py       # ATOM P3 environment
└── scripts/
```



7.3 Environment and dependencies

- OS: Ubuntu 20.04/ 22.04
- Python ≥ 3.8
- NVIDIA GPU (driver version 515 or higher)
- NVIDIA Isaac Gym Preview 4
- Required Libraries: PyTorch, NumPy, YAML, etc. (see `environment.yml` for details)

7.4 Installation process

1. Clone this repository

```
git clone https://github.com/embody-dobot/atom-locomotion-training.git
cd atom-locomotion-training
```

2. Create the conda environment

```
conda env create -f environment.yml
conda activate atom-gym
```

3. Install the Isaac Gym

```
cd isaacgym/python && pip install -e . # Download Isaac Gym Preview 4 from NVIDIA
cd examples && python 1080_balls_of_solitude.py # Verify the installation
```

4. Install the ATOM-Gym

```
cd atom-locomotion-training && pip install -e .
```

7.5 Training and evaluation

7.5.1 Training

Train the reinforcement learning policy for the ATOM robot:

```
python humanoid/scripts/train.py \  
  --task ATOM_p2_ppo \  
  --run_name test \  
  --headless \  
  --num_envs 4096
```

Key parameter:

- `--task`: Select ATOM_p2_ppo or ATOM_p3_ppo depending on the robot model
- `--run_name`: Experiment name
- `--num_envs`: Number of parallel environments
- `--headless`: Headless mode (no GUI), used for server/cloud training

7.5.2 Evaluate and export

Evaluate the trained policy and export it for deployment:

```
# Evaluate and export the trained policy  
python humanoid/scripts/play.py \  
  --task ATOM_p2_ppo \  
  --run_name test \  
  --load_run /path/to/logs/ATOM_p2_ppo/{timestamp}_test
```

This command will:

- Evaluate policy performance in simulation
- Export to PyTorch (.pt) format for Python deployment
- Export to ONNX (.onnx) format for C++/deployment use

7.6 Sim2Sim workflow

Verify policy robustness by transferring the policy from Isaac Gym to the MuJoCo physics engine:

```
# Transfer policies from Isaac Gym to MuJoCo  
python humanoid/scripts/sim2sim.py \  
  --task ATOM_p2_ppo \  
  --load_model /path/to/logs/policies/policy.pt
```

This workflow supports comparative testing across physics engines, improving policy robustness and identifying deployment risks before real-world testing.

7.7 Add new environment

1. Create environment files
 - ① Add <your_env>_config.py and <your_env>_env.py to humanoid/envs/custom/
 - ② Inherit from LeggedRobotCfg and LeggedRobot respectively
2. Add robot resources
 - ① Place the URDF, mesh, and MJCF files in resources/robots/.
 - ② Set paths in the configuration file
3. Parameter Configuration
 - ① Set URDF path, PD gains, body names, and default joint angles
 - ② Define experiment parameters and reward functions
4. Register the environment

Add the new environment to humanoid/envs/__init__.py.

NOTE

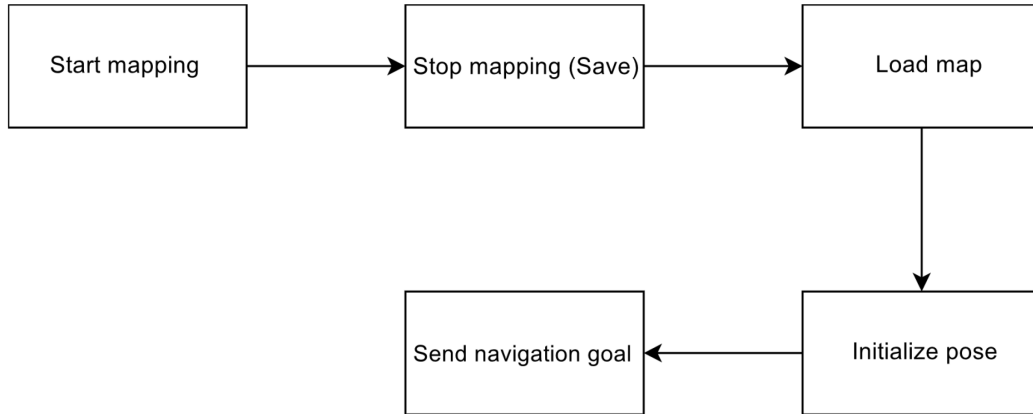
- For more in-depth customization, please refer to the existing ATOM configurations and base classes in humanoid/envs/base/.
- The implementation of Humanoid-Gym-Atom builds upon the [legged_gym](#) and [rsl_rl](#) projects (created by the Robotic Systems Lab) and the [humanoid-gym](#) project (open-sourced by RobotEra).

8. SLAM Navigation Guide

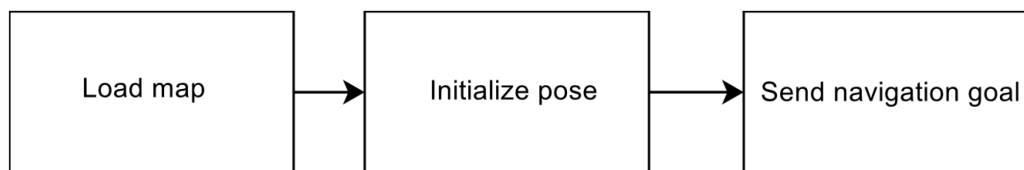
8.1 Introduction

The current navigation is developed based on ROS1, and all functional interfaces rely on the ROS communication mechanism.

- Workflow (without map):



- Workflow (with map):



NOTICE

- The current SDK is suitable for open scenarios with an area of less than 30m × 30m and with significant environmental features. Currently, it only supports the "obstacle stop" function. Please pay attention to safety during use.
- The robot comes with SLAM pre-installed from the factory, and the SLAM software is not open-source.

8.2 Function interface

Module	Function	Description
Mapping	Start mapping	The robot starts mapping
	Stop mapping	When the robot finishes mapping, you need to select whether to save the map and specify the map name in the parameters.
Localization	Load map	Map loading
	Initialize pose	It receives the robot's approximate position information and performs precise localization based on this information.
	Stop localization	Stop localization program
	Get robot pose (3D)	Get robot 3D pose
	Get robot pose (2D)	Get robot 2D pose
Navigation	Single-point navigation	Receives a single navigation goal, autonomously plans the path, and automatically navigates to it
	Navigation status	Get Navigation status
	Control navigation status	Control navigation status, including: Stop, Pause, and Resume. Stop: Ends the current navigation task Pause: Pause the current navigation; the robot stops moving Resume: Resume navigation

8.3 Mapping

Start mapping

Communication mode	Service
Name	/start_mapping
Message type	std_srvs/Trigger
Message definition	<pre> --- bool success #Whether successful string message #No content </pre>

Stop Mapping

Communication mode	Service
Name	/stop_mapping
Message type	type_define/FinishMapping
Message definition	<pre> bool save_map #Whether to save the map string map_name #Map name --- bool success #Whether successful </pre>

8.4 Localization

8.4.1 Map loading

Communication mode	Service
Name	/load_map
Message type	type_define/SetString
Message definition	<pre> string string_value #Map name --- bool success #Whether successful </pre>

8.4.2 Initialize pose

Communication mode	Topic
Name	/initialpose
Message type	geometry_msgs/PoseWithCovarianceStamped
Message definition	<pre> std_msgs/Header header #Message header (ignore) geometry_msgs/PoseWithCovariance pose #Robot pose, approximate pose is acceptable </pre>

8.4.3 Stop localization

Communication mode	Service
Name	/stop_localization
Message type	std_srvs/Trigger
Message definition	<pre> --- bool success #Whether successful string message #No content </pre>

8.4.4 Get robot pose

Communication mode	Topic
Name	/robot_pose
Message type	geometry_msgs/PoseWithCovarianceStamped
Message definition	std_msgs/Header header #Message header geometry_msgs/PoseWithCovariance pose #Robot pose and covariance

8.4.5 Get robot 2D pose

Communication mode	Topic
Name	/robot_pose_2d
Message type	geometry_msgs/Pose2D
Message definition	float64 x #x coordinate float64 y #y coordinate float64 theta # angle (in radians)

8.5 Navigation

8.5.1 Single-point navigation

Communication mode	Topic
Name	/navigation_goal
Message type	geometry_msgs/Pose2D
Message definition	float64 x #x coordinate float64 y #y coordinate float64 theta # target pose angle (in radians)

8.5.2 Navigation status

Communication mode	Topic
Name	/navigation_state
Message type	std_msgs/UInt8
Message definition	uint8 data # navigation status: 0 - idle; 1 - executing; 2 - completed; 3 - obstacle stop; 4 - error. The state transition for one task cycle is: 0 → 1 → 2 → 0

8.5.3 Control navigation status

Communication mode	Service
Name	/controll_nav
Message type	type_define/NavControl
Message definition	<pre>uint8 cmd # 1 - stop; 2 - pause; 3 - resume --- bool success string error_msg</pre>

8.6 Example

Open a terminal and enter the following command to configure the environment variables, so that ROS commands can autocomplete parameters with the Tab key later.

```
echo "source ~/Traverse/install/setup.bash" >> ~/.bashrc
```

The complete operation procedure is as follows:

1. Start the program

Extract the algorithm package to the home directory. Depending on whether auto-start has been configured, enter the corresponding command in the terminal:

- If auto-start has been enabled, the robot will automatically start the navigation service each time it powers on. You only need to launch rviz:

```
cd ~/Traverse
./script/start_rviz.sh
```

- If automatic startup is not configured, start the program manually:

```
cd ~/Traverse
./script/start_nav.sh
```

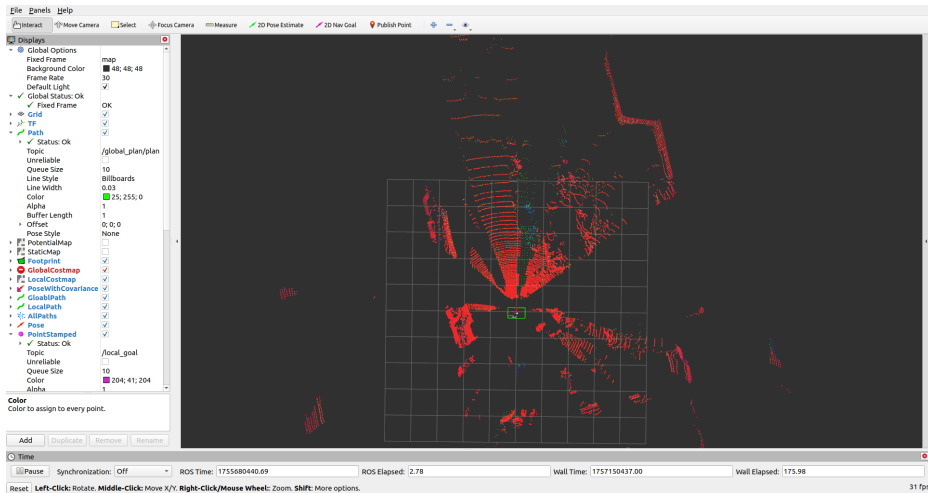
Wait for RViz to appear; startup is then complete.

2. Start mapping

- Control the robot to enter the Bent-Knee Walking mode or the Straight-Knee Walking mode. In this mode, the robot will raise its head and look straight ahead to ensure that the LiDAR remains parallel to the ground.
- Open the terminal and enter the following command to call the mapping interface, or call the mapping interface through your own code.

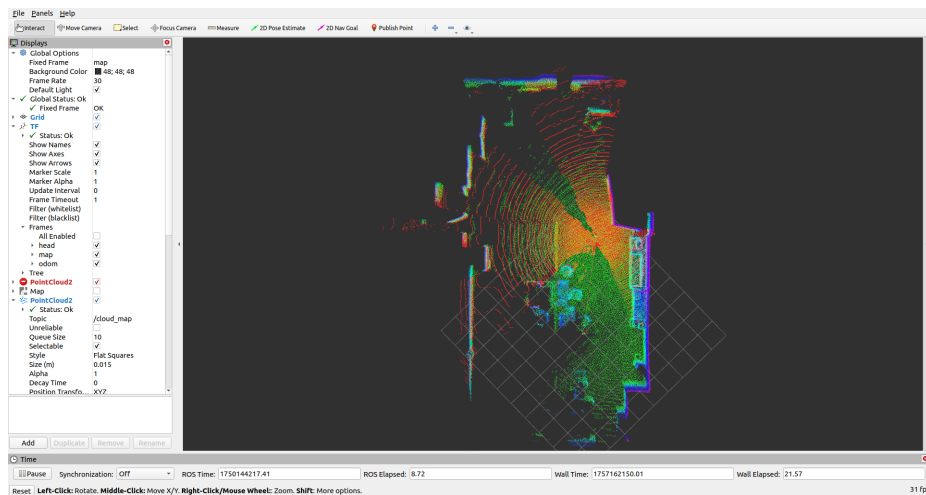
```
rosservice call /start_mapping "{}"
```

- ③ The running result is as follows: the point cloud and pose display can be seen in RViz. The red represents the real-time LiDAR point cloud, and the color represents the map point cloud.



3. Complete the map

Remotely drive the robot around the entire site to ensure that all areas are included in the map. Observe in RViz as the map gradually takes shape.



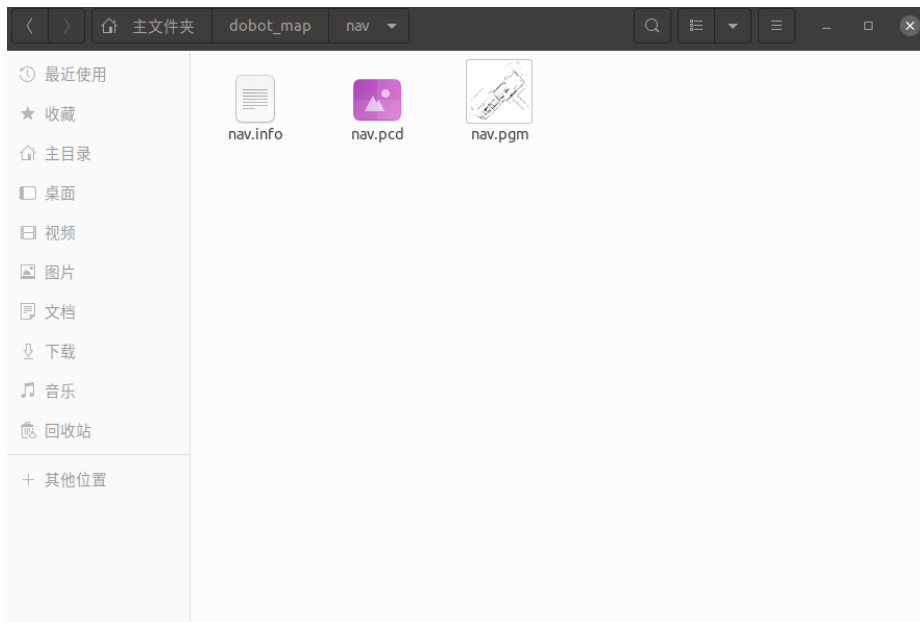
4. Stop mapping

Enter the following command in the terminal to call the stop-mapping interface, or call it through your own code.

If you need to save the map, set "save_map" to "true" and enter a custom map name for "map_name". The map saving process may take a few seconds to over ten seconds.

```
rosservice call /stop_mapping "save_map: false
map_name: 'your_map_name'"
```

After the map is saved, a folder with the corresponding map name will appear in the home/dobot_map directory. Enter the folder and you will see the following three files.

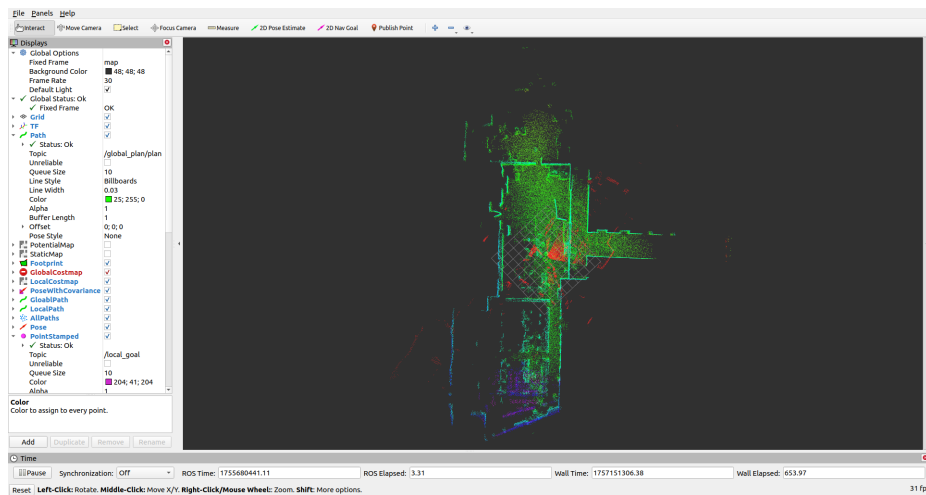


5. Map loading

Enter the following command in the terminal to call the load map interface, or call it through your own code. Enter the name of the map you want to load in map_name.

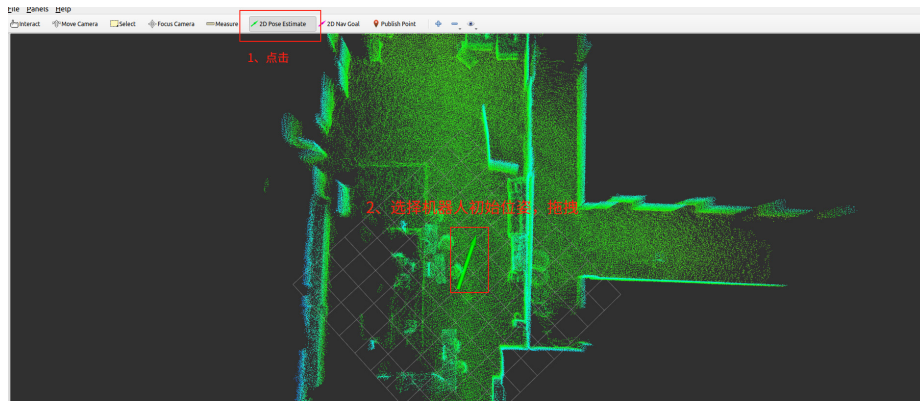
```
rosservice call /load_map "string_value:: 'your_map_name'"
```

At this point, the map will be displayed in Rviz.

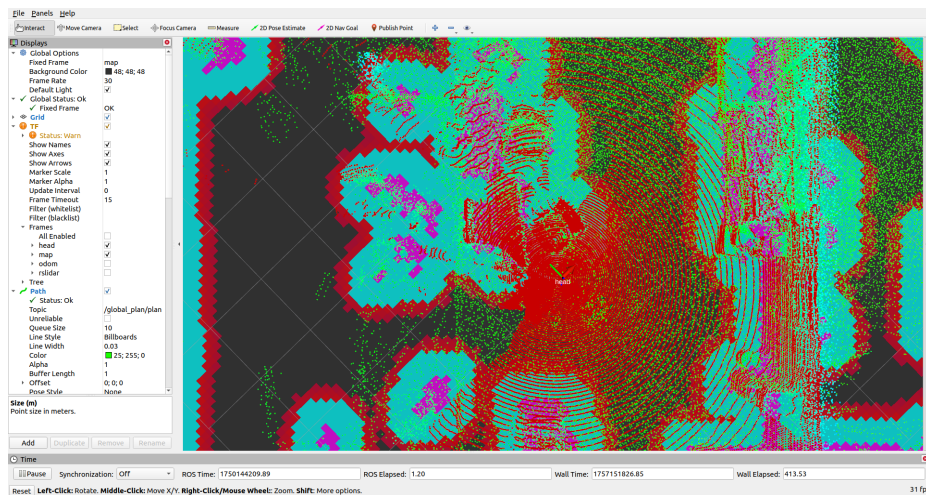


6. Initializing localization

Use Rviz controls or call the relocation interface to tell the robot its approximate current position.



After initialization is complete, you will see the real-time point cloud (red), the robot's pose (red arrow), and the costmap (composed of red, blue, purple, and transparent blocks).



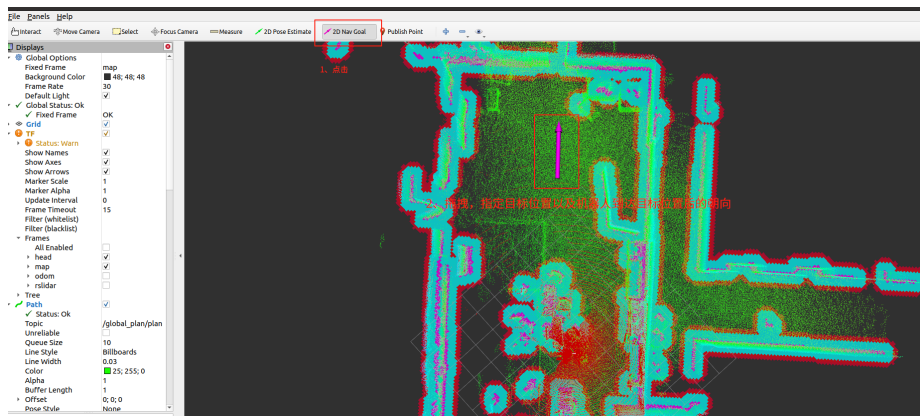
7. Publishing a navigation goal

Use Rviz controls or call the interface to publish a navigation goal.



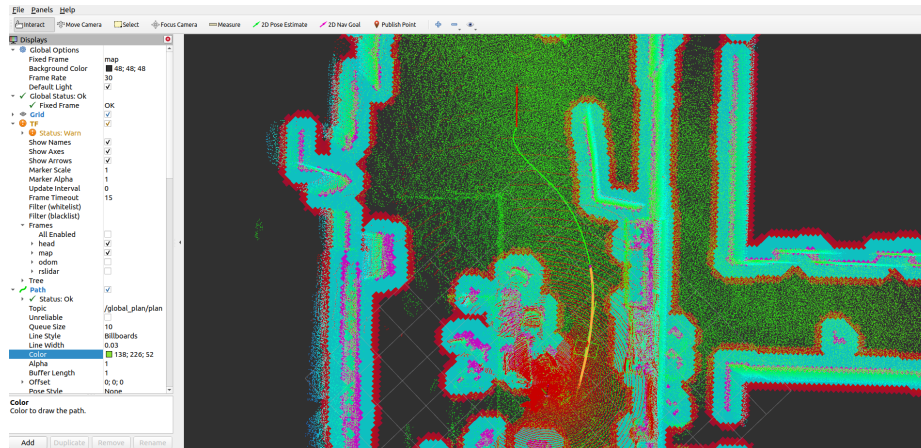
NOTICE

- The blue and purple areas in the costmap indicate positions with obstacles and cannot be set as navigation goals.
- It is recommended that the navigation point be at least 1.5 meters away from the robot.



8. Robot navigation

After publishing a navigation goal, you can see the path planned by the robot. The robot will roughly follow this path, stop when it reaches the goal, then rotate to the specified orientation and end the navigation task.



NOTICE

- Unless the environment has changed significantly, there is no need to rebuild the map every time.
- Each time the program is restarted, you need to reload the map and reinitialize localization.
- The robot currently cannot avoid obstacles with a height of less than 40 cm.

8.7 Code example

NOTICE

If you are currently in a Conda environment, exit it first:

```
conda deactivate
```

The following code demonstrates how to call the map loading interface:

```
import rospy
# Import custom service type
from type_define.srv import SetString, SetStringRequest

def call_custom_service():
    # Initialize the node
    rospy.init_node('my_python_client')

    # Wait for the service to become available (with a 1-second timeout)
    rospy.wait_for_service('/load_map', timeout=1)
    # Create a service client
    set_param_client = rospy.ServiceProxy('/load_map', SetString)
```

```
# Construct the request (assign values according to the fields defined in the .srv file)
req = SetStringRequest()
req.string_value = "map_name" #

# Call the service
resp = set_param_client(req)

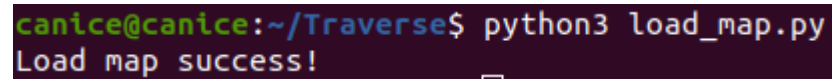
# Process the response (according to the response fields in the .srv file)
if resp.success: # Assuming the response has a success field
    print("Load map success!")
else:
    print("Load map fail!")

if __name__ == '__main__':
    call_custom_service()
```

Place load_map.py in the ~/Traverse/ directory, and run the following command:

```
python3 load_map.py
```

The running result is as follows:



```
canice@canice:~/Traverse$ python3 load_map.py
Load map success!
```

9. Common Errors and Description

9.1 Power supply error

ID	Level	Error description	Solution
256	1	Lower body power supply error	Check if the hardware is functioning properly, or contact technical support engineer
257	1	Upper body power supply error	
258	1	24V1 power supply error	
259	1	24V2 power supply error	
260	1	12V power supply error	
261	1	Peripheral power supply error	
262	1	Power supply error at the replenishment port	

9.2 Voltage error

ID	Level	Error description	Solution
263	1	Lower body undervoltage	Check if the hardware is functioning properly, or contact technical support engineer
264	1	Lower body overvoltage	
265	1	Upper body undervoltage	
266	1	Upper body overvoltage	

9.3 Battery error

ID	Level	Error description	Solution
512	1	Cell overvoltage	Single cell voltage exceeds 4.2V. Please verify hardware status or contact technical support for assistance
513	1	Cell undervoltage	Single cell voltage is below 2.5V. Please check if the hardware is functioning properly, or contact technical support for assistance
514	1	Battery failure	Single cell voltage is below 2V. Please check if the hardware is functioning properly, or contact technical support for assistance

ID	Level	Error description	Solution
515	1	Battery pack overvoltage	During charging, the battery pack voltage exceeds 70.5V. Please check if the hardware is functioning properly, or contact technical support for assistance
516	1	Battery pack undervoltage	The battery pack voltage is below 40V. Please check if the hardware is functioning properly, or contact technical support for assistance
517	1	Charging overcurrent	During charging, the current exceeds 12A for 20ms. Please check if the hardware is functioning properly, or contact technical support for assistance
518	1	Discharge overcurrent	During discharging, the current exceeds 100A for 20ms. Please check if the hardware is functioning properly, or contact technical support for assistance
519	1	Charging short circuit	During charging, the current exceeds 15A for 1.6ms. Please check if the hardware is functioning properly, or contact technical support for assistance
520	1	Discharge short circuit	During discharging, the current exceeds 120A for 200ms. Please check if the hardware is functioning properly, or contact technical support for assistance
521	1	Chip over-temperature	The BMS chip's internal temperature monitoring is abnormal. Please check if the hardware is functioning properly, or contact technical support for assistance
522	1	Battery over-temperature	During charging, the battery temperature exceeds 50°C; during discharging, the battery temperature exceeds 60°C. Please check if the hardware is functioning properly, or contact technical support for assistance
523	1	Battery low-temperature	During charging, the battery temperature is below 2°C; during discharging, the battery temperature is below -18°C. Please check if the hardware is functioning properly, or contact technical support for assistance
524	1	PCB Overtemperature	The NTC temperature on the PCB board exceeds 85°C. Please check if the hardware is functioning properly, or contact technical support for assistance
525	1	Open circuit alarm	Battery sampling wire disconnected, or open circuit detected in cell series

ID	Level	Error description	Solution
			connection. Check hardware or contact technical support
526	1	Cell mismatch	Maximum voltage difference between cells exceeds 0.234V. Check hardware or contact technical support
527	1	MOSFET drive error	MOSFET timeout (typically caused by soft-start error). Check hardware or contact technical support
528	1	MCU abnormal alarm	MCU and BMS chip communication error. Check if the hardware is functioning properly, or contact technical support
529	1	Sensor alarm	The BMS chip's internal temperature monitoring is abnormal. Please check if the hardware is functioning properly, or contact technical support for assistance

9.4 File parsing error

ID	Level	Error description	Solution
1280	1	Failed to parse the alarm_host.json file	The system reported a file error. Please contact technical support
1281	1	Failed to parse the alarm_servo.json file	The system reported a file error. Please contact technical support
1282	1	Failed to parse the Offset.json file	Joint offset file error. Please contact technical support
1283	1	Failed to parse the equipment_config.json file	Peripheral configuration file error. Please contact technical support
1284	1	Failed to parse the robotType.json file	Robot type file error. Please contact technical support
1285	1	Failed to parse the snCode.json file	SN code file error. Please contact technical support
1286	1	Failed to parse the is_virtual.json file	Virtual axis file error. Please contact technical support
1287	1	Failed to parse the offset.json.default file	Factory joint offset backup file error. Please contact technical support
1288	1	Failed to parse the Offset.json.bak file	User joint offset backup file error. Please contact technical support

9.5 File modification error

ID	Level	Error description	Solution
1289	1	Failed to modify the Offset.json file	Joint offset file error. Please contact technical support
1290	1	Failed to modify the offset.json.default file	Factory joint offset file error. Please contact technical support
1291	1	Failed to modify the offset.json.bak file	User joint offset backup file error. Please contact technical support
1292	1	Failed to modify the equipment_config.json file	Peripheral configuration file error. Please contact technical support
1293	1	Failed to modify the is_virtual.json file	Virtual axis file error. Please contact technical support

9.6 Other errors

ID	Level	Error description	Solution
1294	1	Enable failed	Enable failed. Please contact technical support
1295	1	CAN1 board offline	Joint offset file error. Please contact technical support.
1296	1	CAN2 board offline	Factory joint offset file error. Please contact technical support
1297	1	Position limit exceeded	Servo position limit exceeded. Please contact technical support
1298	1	Speed limit exceeded.	Servo speed limit exceeded. Please contact technical support
1299	1	Torque limit exceeded	Servo torque limit exceeded. Please contact technical support
1300	1	Zero calibration failed	Zero calibration failed. Please contact technical support
1301	1	Bus communication error	Bus communication error. Please contact technical support
1302	1	Failed to switch modes	Mode switch failed. Please contact technical support

ID	Level	Error description	Solution
1303	1	IMU zero calibration failed	IMU zero calibration failed. Please contact technical support
1304	1	Log file compression failed	Log file compression failed. Please contact technical support
1305	1	Joint offset modification failed	Joint offset modification failed. Please contact technical support
1306	1	Dexterous hand configuration file error	Dexterous hand configuration file error. Please contact technical support
1307	1	Position over-limit file error	Position limit file error. Please contact technical support
1308	1	Speed over-limit file error	Speed limit file error. Please contact technical support
1309	1	Torque over-limit file error	Torque limit file error. Please contact technical support
1310	1	Position over-limit file modification failed	Position over-limit file modification failed. Please contact technical support
1311	1	Speed over-limit file modification failed	Speed over-limit file modification failed. Please contact technical support
1312	1	Torque over-limit file modification failed	Torque over-limit file modification failed. Please contact technical support
1314	1	Failed to parse AtomOs.version file	AtomOs.version file parsing failed. Please contact technical support engineer
1315	1	Power heartbeat error	Power heartbeat error. Please contact technical support engineer
1316	1	Bms heartbeat error	Bms heartbeat error. Please contact technical support engineer
8752	1	Hardware overcurrent	The actual current value collected by the servo exceeds 90% of the hardware maximum sampling current. Check whether the hardware is functioning properly, or contact technical support engineer

ID	Level	Error description	Solution
8992	1	Software overcurrent	Power off and restart, or contact technical support engineer
9040	1	Module in overload status	Check if the hardware is functioning properly, or contact technical support engineer.
9088	1	The current offset of the zero point is too high	Reset the zero-point current offset
12592	1	Driver input phase lost	Check if the hardware is functioning properly, or contact technical support engineer
12816	1	Voltage between PN is too high	The servo detects that the bus voltage exceeds the set threshold. Check whether the hardware is functioning properly, or contact technical support engineer
12832	1	Voltage between PN is too low	The servo detects that the bus voltage is below the set threshold. Check whether the hardware is functioning properly, or contact technical support engineer
12928	1	Servo and motor power matching fault	Check if the hardware is functioning properly, or contact technical support engineer
13184	1	Driver output phase sequence error	Wire in the correct phase order
17168	1	Driver temperature is too high	Check if the hardware is functioning properly, or contact technical support engineer
17169	1	Damaged or uninstalled temperature measuring resistors	Check if the hardware is functioning properly, or contact technical support engineer
21008	1	Driver board recognition error protection	Check if the hardware is functioning properly, or contact technical support engineer
21120	1	FPGA configuration error	System error, please contact technical support engineer
21121	1	Internal error	System error, please contact technical support engineer
21122	1	STO safety wiring error	System error, please contact technical support engineer

ID	Level	Error description	Solution
21569	1	Driver board connection is abnormal	Check if the hardware is functioning properly, or contact technical support engineer
25378	1	Repeated assignment of DI functions	Check the DI group parameters to ensure the same function is not assigned to multiple DIs
25379	1	DO function assignment overrun	Check if the communication cables are securely connected
25382	1	Wrong setting for software position upper and lower limits	Set the correct values for 0x607D-01h and 0x607D-02h
25383	1	Wrong home position offset setting	Set the correct values for 0x607D-01h and 0x607D-02h, and ensure that the origin offset value (0x607C) falls between them
28961	1	Stalled motor overheat protection	Check whether the brake is properly released or whether the motor is stuck due to mechanical failure
29056	1	Motor is in overload status	Check whether the end load of the robot exceeds the robot's rated load, or contact the relevant technical personnel to replace the motor
29568	1	Encoder communication is abnormal	Check the encoder wiring, or replace/repair the encoder
29569	1	Encoder is abnormal	Check if the hardware is functioning properly, or contact technical support engineer.
29570	1	Encoder battery is abnormal	Please contact technical support engineer
29571	1	Encoder battery power supply error	System error, please contact technical support engineer
29572	1	Main encoder CRC check error	System error, please contact technical support engineer
29573	1	Auxiliary encoder is disconnected	System error, please contact technical support engineer
29574	1	Auxiliary encoder internal error	System error, please contact technical support engineer
29575	1	Auxiliary encoder CRC check error	System error, please contact technical support engineer
29576	1	PWM output buffer error	Replace the driver

ID	Level	Error description	Solution
29577	1	Brake power supply anomaly	Check whether the brake is connected or damaged
29584	1	Large deviation of joint position	1. Check production parameters 2. Check encoder installation 3. Check the transmission mechanism
30080	1	Servo bus communication error	System error, please contact technical support engineer
30081	1	Unburned XML configuration file	System error, please contact technical support engineer
30082	1	Synchronization cycle configuration error	System error, please contact technical support engineer
30083	1	Master station synchronization signal lost	System error, please contact technical support engineer
30084	1	Excessive synchronization cycle error	System error, please contact technical support engineer
33920	1	Exceeding maximum speed protection	Check whether the motor wire sequence connection is correct
33921	1	Difference between the set speed and current speed exceeds the range	Check if the hardware is functioning properly, restart the controller, or contact technical support engineer
33922	1	Motor stalled	Check whether the motor angle is normal or whether the motor is damaged
34321	1	Difference between the set position and current position exceeds the range	System error, please contact technical support engineer
34322	1	Position is out of range	Enter the correct parameters
34323	1	Excessive deviation from fully closed-loop position	Check if the hardware is functioning properly, or contact technical support engineer
65282	1	Failed to identify angles	Check if the hardware is functioning properly, or contact technical support engineer
65344	1	Parameter recognition failure	Check if the hardware is functioning properly, or contact technical support engineer