# TRON 2 SDK Development Guide（20260225）

| Version | Revision Date | Reviser | Change Log |
|---|---|---|---|
| V0.1 | 20250826 | @Eleven | First draft |
| v0.2 | 20250909 | @Damon | Outline adjustment |
| V0.3 | 20251011 | @Charlie | Update the acquisition software package, and add the joint target position and gripper target position in the section "5.5.5 HDF5 Data Storage Format Statistical Table" |
| v0.4 | 20251119 | @Damon | Add 3.5.3 ServoJ control command, 3.5.4 Obtain the end pose of the dual arms, and 3.5.5 Obtain the robot joint state |
| v0.5 | 20260113 | @Damon | Firmware version updated to the latest version 20260113 |

# 1. TRON2 SDK Overview

- **Low-level motion control development interface**: In developer mode, users can use this interface to develop their own motion control algorithms, complete algorithm simulation, and seamlessly deploy the algorithms on real machines.
- **High-level application development interface**: Under the developer mode, users can develop their own software business functions, such as dual-arm operation and robot management, etc., using the High-level application development interface on the basis of the pre-installed Low-level motion control algorithms of the robot.

## 1.1 Query/Configure Robot Model

When compiling and running control algorithms and simulator programs, selecting the correct robot model is crucial. You can ensure accurate identification and application of the corresponding robot model in different tasks by checking the robot model and setting it in the environment variable `ROBOT_TYPE`. The following are the steps to check and configure the robot model.

### 1.1.1 View robot model

1. Please select and connect to your robot's Wi-Fi hotspot, named`XXXX_TRON2A_XXX_5G`or`XXXX_TRON2A_XXX_2.4G`, with the password:`12345678`

2. Enter`http://10.192.1.2:8080`in the browser to access the "Robot Information Page" and view robot information. As shown in the figure below, the SN (Serial Number) displayed on the page isDACH_TRON2A_025, whereDACH_TRON2Ais therobot model.

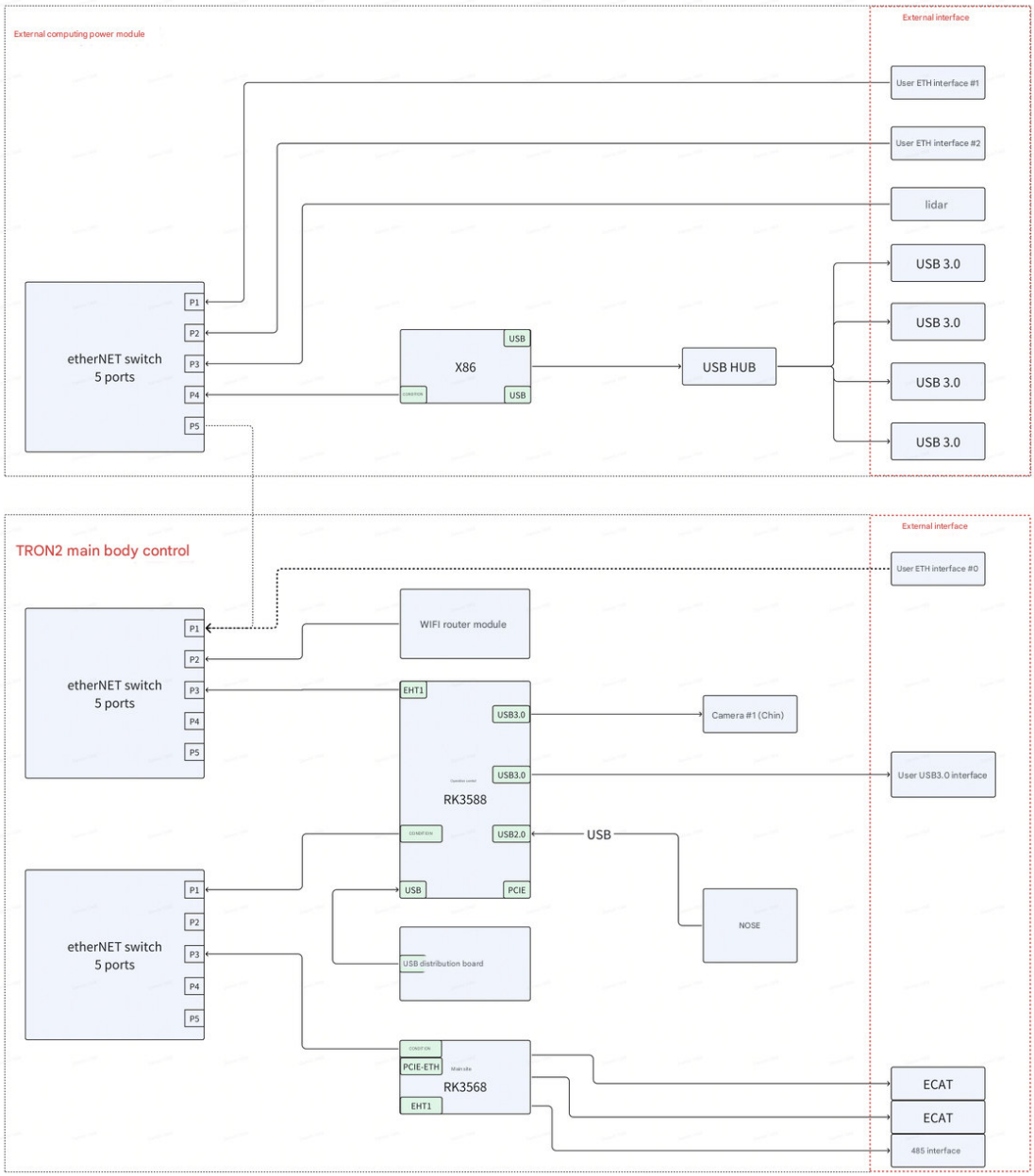
### 1.1.2 Set the robot model

Open a Bash terminal and enter the following Shell command to set the robot model. This way, when compiling and running the motion control algorithm and the simulator program, you can obtain the correct robot model information.

```
Bash
echo 'export ROBOT_TYPE=DACH_TRON2A' >> ~/.bashrc && source
~/.bashrc
```

## 1.2 Communication Architecture Diagram

The following figure shows the system composition and interactive relationship between the developer's computer and the Robot hardware / chassis. The developer's computer section includes motion control algorithm nodes and software business logic implementation modules, which control the motion of the Robot hardware / chassis through data communication via `High-level application development interfaces` and `lower-level motion control development interfaces` . The Robot hardware / chassis consists of a data switch, a main control computer, and various hardware components, with the main control computer responsible for coordinating the operation of each component.




## 1.3 Computing Power Module Computer

The computing power module computer is mainly used for developing robot-related algorithms and applications. You can connect to the Robot hardware / chassis system via WiFi or a wired network to log in to the developer computer. The specific steps are as follows:

- Please select and connect to your robot's Wi-Fi hotspot, password:`12345678`
- Log in to the developer's computer system via SSH
  - Login address is:`10.192.1.4`
  - Login password is:`123456`
  - Enter the following command in the terminal (confirm the command for the first connection):

```
Python
ssh guest@10.192.1.4
```

- Developer computer system configuration:
  - Operating System: Ubuntu 20.04.6 LTS
  - ROS2: The system is installed with the ROS2 Foxy version of the robot system by default
  - ROS1: The system will default to installing the ROS1 Noetic version of the robotic system

# 2. Low-level motion control Development Interface

## 2.1 C++ Motion Control Development Interface

### 2.1.1 Overview

The Low-level motion control development interface provides unified C++/Python APIs, is compatible with ROS1, ROS2, and non-ROS systems, and enables the rapid transplantation and deployment of motion control algorithms. Through the hardware abstraction layer and standardized communication protocol, developers can seamlessly switch between simulation and real hardware environments, significantly reducing the cost of multi-platform adaptation.

### 2.1.2 Install the Motion Control Development Library

- Linux x86_64 Environment

```
Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/amd64/limxsdk-*-py3-none-any.whl
```

- Linux aarch64 environment

```
Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/aarch64/limxsdk-*-py3-none-any.whl
```

- Windows Environment

```
Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/win/limxsdk-*-py3-none-any.whl
```

### 2.1.3 getInstance Interface Introduction

Functio **getInstance**

| | |
|---|---|
| n Name | |
| Function Prototype | static Tron2* getInstance(); |
| Function Overview | Retrieve the singleton pointer the singleton instance of the Tron2 robot class |
| parameter | None |
| Return Value | Tron2*, a pointer to the Tron2 instance |
| Remarks | uses the singleton pattern to ensure that only one instance of the Tron2 class exists in the program |
| Code Example | C++<br>代码块<br>`#include <thread>`<br><br>`// Include header file`<br>`#include "limxsdk/tron2.h"`<br><br>`// Use the limxsdk namespace`<br>`using namespace limxsdk;`<br><br>`int main(int argc, char *argv[]){`<br>`  // Get the singleton instance`<br>`  Tron2* robot = Tron2::getInstance();`<br><br>`  // Infinite loop to keep the program running`<br>`  while (true)`<br>`  {`<br>`  // Sleep for 1000 milliseconds`<br><br>`std::this_thread::sleep_for(std::chrono::milliseconds(1000));`<br>`  }`<br><br>`  return 0;`<br>`}` |

### 2.1.4 init Interface Introduction

| | |
|---|---|
| Function Name | **init** |
| Function Prototype | bool init(const std::string& robot_ip_address = "127.0.0.1"); |
| Function Overview | **Initialize** the communication operating environment of the motion control algorithm program, usually called before calling other interfaces in the main function to complete the initialization work. |
| parameter | robot_ip_address: The IP address of the robot. For simulation, it is usually set to "127.0.0.1", while for a real robot, it is set to "10.192.1.2". |
| Return Value | If the initialization is successful, return true; otherwise, return false. |
| Remarks | None |

Code Example

C++
代码块

```
#include <thread>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }
```

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

  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 seconds millisecond

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.5 getMotorNumber Interface Introduction

| | |
|---|---|
| Function Name | **getMotorNumber** |
| Function Prototype | uint32_t getMotorNumber(); |
| Function Overview | Get the number of **motors in the robot**. |
| parameter | None |
| Return Value | Returns an unsigned integer representing the total number of motors in the robot. |
| Remarks | Under normal circumstances, the number of motors in thedual-arm configurationof the robot is16 (including the head motor). |
| Code Example | C++<br>代码块<br>`#include <thread>` |

```
// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

  // Get the number of motors in the robot
  uint32_t motor_num = robot->getMotorNumber();

  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.6 Introduction to the subscribeImuData Interface

| Function | subscribeImuData |
| --- | --- |

**Name**

| | |
|---|---|
| Function Prototype | void subscribeImuData(std::function<void(const ImuDataConstPtr&)> cb); |
| Function Overview | Subscribe to the **IMU data** of the robot, and call the specified callback function when new IMU data is received. |
| parameter | cb: Callback function for processing new IMU data. |
| Return Value | None |
| Remarks | The prototype of the ImuData data structure is as follows: |

```C++
/**
 * @struct ImuData
 *
 * @brief represents the structure for robot IMU data based on sensor feedback.
 *
 * This structure encapsulates IMU data, including accelerometer, gyroscope, and quaternion data.
 */
struct ImuData {
  uint64_t stamp; // Timestamp, in nanoseconds, typically representing the time when this data was recorded or generated.
  float acc[3];   // Used to store IMU accelerometer data to track linear acceleration along the three axes (X, Y, Z).
  float gyro[3];  // Used to store IMU gyroscope data to track angular velocity or rotational velocity along the three axes (X, Y, Z).
  float quat[4];  // Used to store IMU quaternion data, representing directions (w, x, y, z) in three-dimensional space. };
};

// Smart pointer type alias
typedef std::shared_ptr<ImuData> ImuDataPtr;
```

```
typedef std::shared_ptr<ImuData const> ImuDataConstPtr;
```

Code Example

C++
代码块

```
#include <thread>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

  // Subscribe to robot status updates and specify
callback functions
  robot->subscribeImuData([&](const ImuDataConstPtr&
msg) {
    // Process the received ImuData data here
    // Note: The callback function will be called when
ImuData is received
  });

  // Infinite loop to keep the program running
  while (true)
```

```
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(10
00));
  }
  return 0;
}
```

### 2.1.7 subscribeRobotState Interface Introduction

| Function Name | **subscribeRobotState** |
| --- | --- |
| Function Prototype | void subscribeRobotState(std::function<void(const RobotStateConstPtr&)> cb); |
| Function Overview | Subscribe to receive updates about **robot status**. |
| parameter | cb: Callback function, which will be called when a robot state update is received. The callback function parameter is a constant pointer to a RobotState object. |
| Return Value | None |
| Remarks | • The prototype of the RobotState data structure is as follows: |

```C++
/**
 * @struct RobotState
 *
 * @brief represents the structure for the robot's
state based on sensor feedback.
 *
 * This structure encapsulates various data points that
can be used to monitor and control the robot, including
IMU data (accelerometer, gyroscope, quaternion), output
torque, current angle, and velocity, etc.
 */
struct RobotState {
  // Default constructor
```

```
  RobotState() { }
  // Parameterized constructor, used to initialize
vectors tau, q, and dq of size motor_num, all
initialized to 0.0
  RobotState(int motor_num)
  : tau(motor_num, 0.0)
  , q(motor_num, 0.0)
  , dq(motor_num, 0.0) { }
  uint64_t stamp;              // Timestamp, typically
representing the time when this data was recorded or
generated, in nanoseconds
  std::vector<float> tau;      // Vector used to store
the currently estimated output torque (in Newton-
meters)
  std::vector<float> q;        // Vector used to store
the current angle (in radians)
  std::vector<float> dq;       // Vector used to store
the current velocity (in radians per second)
};

// Smart pointer type aliases:
typedef std::shared_ptr<RobotState> RobotStatePtr;
typedef std::shared_ptr<RobotState const>
RobotStateConstPtr;
```

Code Example

C++
代码块

```
#include <thread>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
```

```
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

  // Subscribe to robot status updates and specify
callback functions
  robot->subscribeRobotState([&](const
RobotStateConstPtr& msg) {
    // Process the received RobotState data here
    // Note: The callback function will be called when
a robot state update is received
  });

  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.8 publishRobotCmd Interface Introduction

| **Function Name** | **publishRobotCmd** |
| --- | --- |
| Function Prototype | bool publishRobotCmd(const RobotCmd& cmd); |
| Function Overvie | Send control commands **the robot's actions**. |

| | |
|---|---|
| w | |
| parameter | cmd: A RobotCmd object representing the required robot command. |
| Return Value | None |
| Remarks | |
| Code Example | C++<br>代码块 |

```
#include <thread>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

  // Get the number of motors in the robot
  uint32_t motor_num = robot->getMotorNumber();

  // Creates a RobotCmd object containing the number of
```

```
robot motors
  RobotCmd cmd(motor_num);

  // Issues control commands
  robot->publishRobotCmd(cmd);

  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.9 Introduction to the subscribeSensorJoy Interface

| **Function Name** | **subscribeSensorJoy** |
| --- | --- |
| Function Prototype | void subscribeSensorJoy(std::function<void(const SensorJoyConstPtr&)> cb); |
| Function Overview | During real device deployment, this method is used to subscribe to data from the robot's **remote controller**. When the robot receives data from the remote controller, it will call the specified callback function and pass a constant pointer to the SensorJoy structure containing the remote controller data to the callback function for processing. |
| parameter | cb: Represents a callback function, used to receive data from the robot remote controller. The parameter type of the callback function is SensorJoyConstPtr, which is a shared pointer to a constant SensorJoy structure. |
| Return Value | None |
| Remarks | |
| Code Example | C++<br>代码块<br>#include <thread> |

```
// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

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

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

```
  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.10 Introduction to the subscribeDiagnosticValue Interface

| | |
|---|---|
| Function Name | **subscribeDiagnosticValue** |
| Function Prototype | void subscribeDiagnosticValue(std::function<void(const DiagnosticValueConstPtr&)> cb); |
| Function Overview | In real device deployment, this method is used to **subscribe to the diagnostic values and status information of the robot** . When the robot sends out diagnostic values, the system will call the specified callback function and pass a constant pointer to the `DiagnosticValue` structure containing the diagnostic values to the callback function for processing. This can help monitor the health status of the robot in real time and respond promptly to handle potential issues. |
| parameter | cb: A callback function used to receive robot diagnostic values, whose parameter type is DiagnosticValueConstPtr, which is a shared pointer to a constant DiagnosticValue structure. The DiagnosticValue structure contains information about the robot diagnostic values, including timestamp, level, name, code, and message fields. |
| Return Value | None |
| Remarks | |
| Code Example | C++<br>代码块<br>`#include <thread>` |

```
#include <iostream>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
    robot_ip = argv[1];
  }

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

  // Subscribe to robot diagnostic data
  robot->subscribeDiagnosticValue([&](const
DiagnosticValueConstPtr& msg) {
    // Process robot diagnostic values here
    // For example, take appropriate actions based on
the diagnostic value's level and message
    std::cout << "Diagnostic Value: " << msg->name <<
std::endl;
    std::cout << "Level: " << msg->level << std::endl;
    std::cout << "Code: " << msg->code << std::endl;
    std::cout << "Message: " << msg->message <<
std::endl;
  });

  // Infinite loop to keep the program running
```

```
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

### 2.1.11 Introduction to the setRobotLightEffect Interface

| | |
|---|---|
| Function Name | **setRobotLightEffect** |
| Function Prototype | bool setRobotLightEffect(int effect); |
| Function Overview | In real device deployment, this method is used to **set the lighting effects of the robot** . |
| parameter | effect: An integer representing the desired robot lighting effect, with specific definitions found in the `Tron2::LightEffect` enumeration. |
| Return Value | bool: Indicates whether the robot lighting effect was successfully set. |
| Remarks | Definition of Tron2::LightEffect Enumeration: |

```
C++
enum LightEffect : int {
    STATIC_RED = 0,     // Static red light
    STATIC_GREEN,       // Static green light
    STATIC_BLUE,        // Static blue light
    STATIC_CYAN,        // Static cyan light
    STATIC_PURPLE,      // Static purple light
    STATIC_YELLOW,      // Static yellow light
    STATIC_WHITE,       // Static white light
    LOW_FLASH_RED,      // Red light flashing (slow
flash)
    LOW_FLASH_GREEN,    // Green light flashing (slow
flash)
    LOW_FLASH_BLUE,     // Blue light flashing (slow
```

```
flash)
    LOW_FLASH_CYAN,     // Cyan light flashing (slow
flash)
    LOW_FLASH_PURPLE,   // Purple light flashing (slow
flash)
    LOW_FLASH_YELLOW,   // Yellow light flashing (slow
flash)
    LOW_FLASH_WHITE,    // White light flashing (slow
flash)
    FAST_FLASH_RED,     // Red flashing (fast flashing)
    FAST_FLASH_GREEN,   // Green flashing (fast
flashing)
    FAST_FLASH_BLUE,    // Blue flashing (fast
flashing)
    FAST_FLASH_CYAN,    // Cyan flashing (fast
flashing)
    FAST_FLASH_PURPLE,  // Purple flashing (fast
flashing)
    FAST_FLASH_YELLOW,  // Yellow flashing (fast
flashing)
    FAST_FLASH_WHITE    // White flashing (fast
flashing)
};
```

Code Example

C++
代码块

```
#include <thread>

// Include header file
#include "limxsdk/tron2.h"

// Use limxsdk namespace
using namespace limxsdk;

int main(int argc, char *argv[]){
  // Get singleton instance
  Tron2* robot = Tron2::getInstance();

  // Default robot IP address
  std::string robot_ip = "127.0.0.1";
  if (argc > 1)
  {
    // If command-line arguments are provided, use the
command-line arguments as the robot IP address
```

```
    robot_ip = argv[1];
  }

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

  // Set the robot lighting effect to static red light
  robot-
>setRobotLightEffect(limxsdk::Tron2::STATIC_RED);

  // Infinite loop to keep the program running
  while (true)
  {
    // Sleep for 1000 milliseconds

std::this_thread::sleep_for(std::chrono::milliseconds(1
000));
  }
  return 0;
}
```

## 2.2 Python Motion Control Development Interface

### 2.2.1 Overview

Provides a Python motion control algorithm development interface with the same functionality as C++ , enabling developers unfamiliar with the C++ programming language to use Python for motion control algorithm development.

Python is easy to learn, with concise and clear syntax and a rich set of third-party libraries, enabling developers to quickly get started and implement algorithms rapidly. Through the Python interface, developers can leverage Python's dynamic features for rapid prototyping and experimental verification, accelerating the iteration and optimization process of algorithms. Meanwhile, Python's cross-platform capabilities and strong ecosystem support allow motion algorithms to be more widely applied to different platforms and environments.

In addition, the rapid deployment of algorithm models to simulation and real-world environments also benefits from Python's flexibility. Developers can easily integrate algorithm models into various simulation platforms and real hardware using

Python, achieving rapid iteration and verification of algorithm performance.

The following examples have all been verified in the Python 3.10.4 environment

### 2.2.2 Install the Motion Control Development Library

- Linux x86_64 Environment

```Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/amd64/limxsdk-*-py3-none-any.whl
```

- Linux aarch64 environment

```Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/aarch64/limxsdk-*-py3-none-any.whl
```

- Windows Environment

```Bash
git clone https://github.com/limxdynamics/limxsdk-lowlevel.git
pip install limxsdk-lowlevel/python3/win/limxsdk-*-py3-none-any.whl
```

### 2.2.3 Introduction to the __init__ Interface

| **Function Name** | **__init__** |
| --- | --- |
| Function Prototype | def __init__(self, robot_type: robot.RobotType) |
| Function Overview | During initialization, specify the type of the robot and create a local robot instance of the corresponding type. |
| parameter | robot_type: An enumeration value representing the robot type, with the type being RobotType.Tron2. |
| Return Value | None |
| Remarks | None |

| | |
|---|---|
| Code Example | Python<br>代码块<br>`import sys`<br>`import limxsdk.robot.Robot as Robot`<br>`import limxsdk.robot.RobotType as RobotType`<br><br>`if __name__ == '__main__':`<br>`    # Create a Robot instance of type Tron2`<br>`    robot = Robot(RobotType.Tron2)` |

### 2.2.4 Introduction to the init Interface

| **Function Name** | **init** |
|---|---|
| Function Prototype | def init(self, robot_ip: str = "127.0.0.1") |
| Function Overview | Initialize the communication operating environment of the motion control algorithm program, usually called before calling other interfaces in the main function to complete the initialization work. |
| parameter | robot_ip: The IP address of the robot. For simulation, it is usually set to "127.0.0.1", while for a real robot, it is set to "10.192.1.2". |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | None |
| Code Example | Python<br>代码块<br>`import sys`<br>`import limxsdk.robot.Robot as Robot`<br>`import limxsdk.robot.RobotType as RobotType`<br><br>`if __name__ == '__main__':`<br>`    # Create a Robot instance`<br>`    robot = Robot(RobotType.Tron2)`<br><br>`    robot_ip = "10.192.1.2"`<br>`    # Check if command-line arguments are provided as the robot's IP` |

```
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

    # Initialize the robot's communication runtime
environment using the IP address
    if not robot.init(robot_ip):
        sys.exit()
```

### 2.2.5 Introduction to the getMotorNumber Interface

| | |
|---|---|
| Function Name | **getMotorNumber** |
| Function Prototype | def getJointLimit(self, timeout: float = -1.0) |
| Function Overview | Get the number of motors in the robot. |
| parameter | None |
| Return Value | Returns an unsigned integer representing the total number of motors in the robot. |
| Remarks | Normally, the number of motors in thedual-arm configurationof the robot is16 (including the head motor) |

Code Example

```
Python
代码块
import sys
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if command-line arguments for the robot IP
are provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]
```

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

    # Get the number of motors in the robot
    motor_number = robot.getMotorNumber()
```

### 2.2.6 Introduction to the subscribeImuData Interface

| | |
|---|---|
| Function Name | **subscribeImuData** |
| Function Prototype | def subscribeImuData(self, callback: Callable[[datatypes.ImuData], Any]) |
| Function Overview | Subscribe to the IMU data of the robot and call the specified callback function when new IMU data is received. |
| parameter | callback: A callback function used to process new IMU data. |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | The prototype of the datatypes.ImuData data structure is as follows: |

```
Python
import sys

class ImuData(object):
    __slots__ = ['stamp','acc','gyro','quat']
    def __init__(self):
        self.stamp = 0 # Timestamp, usually
representing the time when this data was recorded or
generated, in nanoseconds
        self.acc = [0. for x in range(0, 3)]  # Used to
store IMU (Inertial Measurement Unit) accelerometer
data, used to track linear acceleration on three axes
        self.gyro = [0. for x in range(0, 3)] # Used to
store IMU gyroscope data, used to track angular
velocity or rotational velocity
```

```
        self.quat = [0. for x in range(0, 4)] # Used to
store IMU quaternion data, representing orientation in
three-dimensional space
```

Code Example

Python
代码块

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

class RobotReceiver:
    # Subscribe to the robot's IMU data
    def imuDataCallback(self, imu: datatypes.ImuData):
        print("\n------\nrobot_state:" + \
              "\n  stamp: " + str(imu.stamp) + \
              "\n  acc: " + str(imu.acc) + \
              "\n  gyro: " + str(imu.gyro) + \
              "\n  quat: " + str(imu.quat))

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if command-line arguments for the robot IP
are provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

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

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

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

    # Subscribe to robot IMU data
```

```
    robot.subscribeImuData(imuDataCallback)

    # Sleep for 1 second to prevent the program from
exiting
    import time
    while True:
        time.sleep(1)
```

### 2.2.7 Introduction to the subscribeRobotState Interface

| | |
|---|---|
| Function Name | **subscribeRobotState** |
| Function Prototype | def subscribeRobotState(self, callback: Callable[[datatypes.RobotState], Any]) |
| Function Overview | Subscribe to receive updates on the robot's status. |
| parameter | callback: A callback function that will be called when a robot state update is received. The callback function parameter points to a datatypes.RobotState object. |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | |
| Code Example | Python<br>代码块<br>`import sys`<br>`from functools import partial`<br>`import limxsdk.robot.Robot as Robot`<br>`import limxsdk.robot.RobotType as RobotType`<br>`import limxsdk.datatypes as datatypes`<br><br>`class RobotReceiver:`<br>`  # Callback function for receiving robot state`<br>`  def robotStateCallback(self, robot_state: datatypes.RobotState):`<br>`  print("\n------\nrobot_state:" + \`<br>`  "\n  stamp: " + str(robot_state.stamp)` |

```
+ \
                "\n  tau: " + str(robot_state.tau) + \
                "\n  q: " + str(robot_state.q) + \
                "\n  dq: " + str(robot_state.dq))

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if command-line arguments for the robot IP
are provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

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

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

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

    # Subscribe to the robot state
    robot.subscribeRobotState(robotStateCallback)

    # Sleep for 1 second to prevent the program from
exiting
    import time
    while True:
        time.sleep(1)
```

### 2.2.8 Introduction to the publishRobotCmd Interface

| | |
|---|---|
| Function Name | **publishRobotCmd** |
| Function Prototy | def publishRobotCmd(self, cmd: datatypes.RobotCmd) |

| | |
|---|---|
| pe | |
| Function Overview | Send / Publish control commands the robot's actions. |
| parameter | cmd: A datatypes.RobotCmd object representing the required robot command |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | • The following example demonstrates controlling the robotic arm to move directly to the zero position. Please first control the robotic arm to move to an initial position or a posture close to the zero position to prevent damage to the robotic arm after running the code.<br>• The kpkd value here is only an example of interface call. When controlling the robotic arm in actual business, please replace it with the kpkd value you need. |
| Code Example | Python<br>代码块 |

```python
import sys
import time
import limxsdk.robot.Rate as Rate
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType
import limxsdk.datatypes as datatypes

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    # For simulation, it is usually set to "127.0.0.1",
while for real robots it is set to "10.192.1.2"
    robot_ip = "10.192.1.2"
    # Check if the command line argument for the robot
IP is provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

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

```
    # Get the number of motors
    motor_number = robot.getMotorNumber()

    # Main loop continuously issues robot commands
    rate = Rate(300) # 300Hz
    cmd_msg = datatypes.RobotCmd()
    while True:
        # Set default values for timestamp, control
mode, joint position, speed, torque, Kp, and Kd
        # motor_names corresponds to the names of the
joints you want to control
        # Note that the following is a format example;
specific parameters should be filled in during actual
use
        cmd_msg.stamp = time.time_ns()
        cmd_msg.mode = [0.0 for _ in range(16)] # No
need to change in actual use
        cmd_msg.q = [0.0 for _ in range(16)]    # Fill
in the angle you plan, control frequency 300hz
        cmd_msg.dq = [0.0 for _ in range(16)]   # No
need to change in actual use
        cmd_msg.tau = [0.0 for _ in range(16)]  # No
need to change in actual use
        cmd_msg.Kp =
[21,21,15,15,10,10,10,21,21,15,15,10,10,10,10,10]
        cmd_msg.Kd =
[0.6,0.6,0.75,0.75,0.5,0.5,0.5,0.6,0.6,0.75,0.75,0.5,0.
5,0.5,0.5,0.5]
        cmd_msg.motor_names = ["" for _ in range(16)]
# No need to change in actual use
        robot.publishRobotCmd(cmd_msg)  # Publish robot
commands
        rate.sleep()  # Control the loop frequency
```

### 2.2.9 Introduction to the subscribeSensorJoy Interface

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

| | |
|---|---|
| Function Overview | In real device deployment, this method is used to subscribe to data from the robot remote controller. When the robot receives data from the remote controller, it will call the specified callback function and pass a constant pointer to the datatypes.SensorJoy structure containing the remote controller data to the callback function for processing. |
| parameter | callback: Represents a callback function used to receive data from the robot remote control. The parameter type of the callback function is datatypes.SensorJoy |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | |

Code Example

```
Python
代码块
import sys
import time
from functools import partial
import limxsdk.robot.Robot as Robot
import limxsdk.robot.RobotType as RobotType
import limxsdk.datatypes as datatypes

class RobotReceiver:
    # Callback function for receiving remote control
data
    def sensorJoyCallback(self, sensor_joy:
datatypes.SensorJoy):
        print("\n------\nsensor_joy:" + \
              "\n  stamp: " + str(sensor_joy.stamp) + \
              "\n  axes: " + str(sensor_joy.axes) + \
              "\n  buttons: " +
str(sensor_joy.buttons))

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if robot IP is provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]
```

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

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

    # partial functions for creating callback functions
    sensorJoyCallback =
partial(receiver.sensorJoyCallback)

    # Subscribe to robot remote control data
    robot.subscribeSensorJoy(sensorJoyCallback)

    # Sleep for 1 second to prevent the program from
exiting.
    import time
    while True:
        time.sleep(1)
```

### 2.2.10 Introduction to the subscribeDiagnosticValue Interface

| | |
|---|---|
| Function Name | **subscribeDiagnosticValue** |
| Function Prototype | def subscribeDiagnosticValue(self, callback: Callable[[datatypes.DiagnosticValue], Any]) |
| Function Overview | In real device deployment, this method is used to subscribe to the diagnostic values and status information of the robot. When the robot emits diagnostic values, the system will call the specified callback function and pass a datatypes.DiagnosticValue structure object containing the diagnostic values to the callback function for processing. This can help monitor the health status of the robot in real time and respond promptly to handle potential issues. |
| parameter | callback: A callback function used to receive the robot's diagnostic value, whose parameter type is datatypes.DiagnosticValue. The datatypes.DiagnosticValue structure contains information about the robot's diagnostic value, including timestamp, level, name, code, and message fields. |

| | |
|---|---|
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | |
| Code Example | |

Python
代码块

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

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

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if arguments for the robot IP are provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

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

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

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

    # Subscribe to robot diagnostic information

robot.subscribeDiagnosticValue(diagnosticValueCallback)

    # Sleep for 1 second to prevent the program from
exiting
    import time
    while True:
        time.sleep(1)
```

### 2.2.11 setRobotLightEffect Interface Introduction

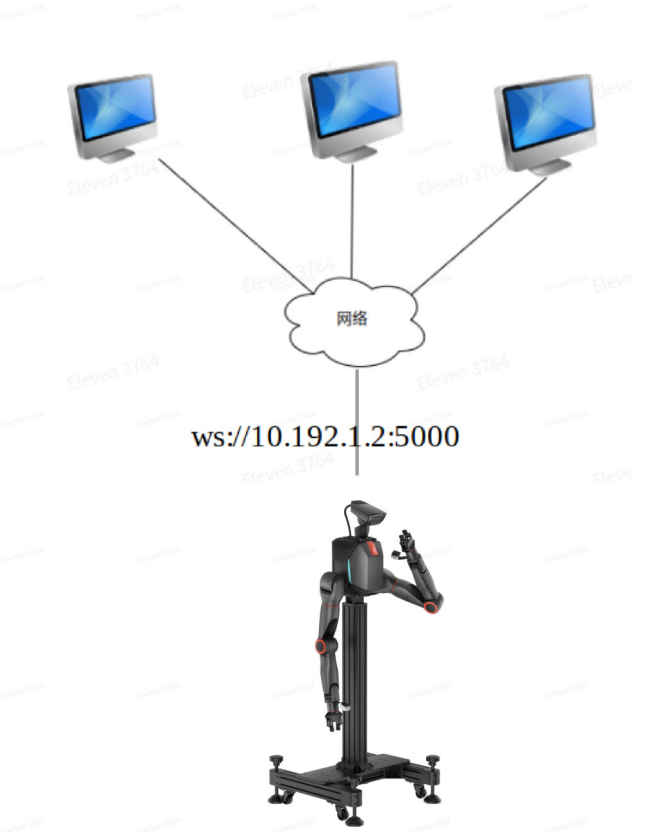
| | |
|---|---|
| Function Name | **setRobotLightEffect** |
| Function Prototype | def setRobotLightEffect(self, effect: datatypes.LightEffect) |
| Function Overview | In real device deployment, this method is used to set the lighting effects of the robot. |
| parameter | effect: An integer representing the desired robot lighting effect, with specific definitions found in the `Tron2::LightEffect` enumeration. |
| Return Value | Success: returns True<br>Failure: Return False |
| Remarks | |
| Code Example | Python<br>代码块<br>`import sys`<br>`from functools import partial`<br>`import limxsdk.robot.Robot as Robot`<br>`import limxsdk.robot.RobotType as RobotType`<br>`import limxsdk.datatypes as datatypes` |

```
class RobotReceiver:

if __name__ == '__main__':
    # Create a Robot instance
    robot = Robot(RobotType.Tron2)

    robot_ip = "10.192.1.2"
    # Check if the robot IP argument is provided
    if len(sys.argv) > 1:
        robot_ip = sys.argv[1]

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

    # Set the robot light effect to static red light

robot.setRobotLightEffect(datatypes.LightEffect.STATIC_
RED)
```

# 3. Upper Application Development Interface

## 3.1 Overview

Under `Developer Mode`, the robot receives user request instructions through the WebSocket communication port `5000`. WebSocket is a real-time communication protocol that establishes a persistent connection between the robot and the user end to quickly and effectively transmit control information and data. As shown in the following figure:




## 3.2 Communication Protocol Format

When the robot receives client instructions via WebSocket, it uses the JSON data protocol for information transfer. This approach has significant advantages: WebSocket is a full-duplex communication protocol that can establish a real-time, low-latency connection between the client and the server, making it particularly suitable for application scenarios with frequent interactions. The JSON data protocol, with its concise and highly readable structure, ensures that data transmission is intuitive and clear, and has cross-platform and cross-language compatibility. The combination of WebSocket and JSON is not only independent of programming languages, suitable for various devices and systems, but also enhances the flexibility of development and the convenience of maintenance.

- The requested data format includes the following fields:
  - `accid`: Robot serial number, please note that it is the serial number of your robot;
  - `title`: Instruction name, prefixed with `request_`;
  - `timestamp`: Timestamp when the instruction was issued, with the unit in milliseconds;
  - `guid`: The Unique Device Identifier of the instruction, used to distinguish different request instructions. If it is a synchronous interface, the value needs to be returned to the Client through the guid field in the "response_xxx" response

message. After the Client receives the response message, it can determine whether the instruction has been executed by comparing whether the value of the guid field is the same as that in the request instruction;

- `data`: Stores the data content of the request instruction. It can contain multiple subfields according to specific requirements to store the data content required by the request instruction, such as parameters for executing actions, text content of sent messages, etc.;

- Examples are as follows:

```
Plain Text
代码块
{
  "accid": "DA_TRON2A_001", # Robot serial number, note that
this is your robot's serial number
  "title": "request_xxx",   # Command name, prefixed with
"request_"
  "timestamp": 1672373633989, # Command issuance timestamp, in
milliseconds
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Unique
identifier for the command, used to distinguish different
request commands
  "data": {}  # Stores the data content of the request command
}
```

- The response data format includes the following fields:

- `accid`: Robot serial number. Please note that it is the serial number of your robot;

- `title`: Instruction name, prefixed with `response_`;

- `timestamp`: Timestamp when the instruction was issued, with the unit in milliseconds;

- `guid`: The same as the `guid` value of the corresponding request instruction;

- `data`: It should at least contain a "result" subfield to store the execution result data of the request instruction. If necessary, it can also contain other subfields, such as error codes, error messages, etc., which are used to describe information about the operation result;

- Examples are as follows:

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",   # Robot serial number, note that
```

```
this should be your robot's serial number
  "title": "response_xxx",  # Command name, prefixed with
"response_"
  "timestamp": 1672373633989, # # Command issuance timestamp,
in milliseconds
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Same as the
GUID value of the corresponding request command
  "data": { # Stores the specific data content of the response
command
    "result": "success"  # "result" is used to store whether
the request command was processed successfully; its value is:
"success" or "fail_xxx"
  }
}
```

- Message Push: It is the process by which a robot actively sends information to the Client. This information can include data such as the robot's serial number, current operating status, and executed operations. By promptly sending this information to the Client, the robot can help the Client better understand its working status, thereby making better use of the services it provides. Its data format includes the following fields:
  - `accid`: Robot serial number. Please note that it is the serial number of your robot;
  - `title`: Instruction name, prefixed with `notify_`;
  - `timestamp`: Timestamp when the message was sent, in milliseconds;
  - `guid`: The guid value of the message, uniquely identifying this message;
  - `data`: Stores the content of message data. It can contain multiple subfields according to specific requirements to store the data content required by the request instruction;
- Examples are as follows:

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",   # Robot serial number, note that
this is your robot's serial number
  "title": "notify_xxx",  # # Message name, prefixed with
"notify_"
  "timestamp": 1672373633989, # Message sending timestamp, in
milliseconds
  "guid": "746d937cd8094f6a98c9577aaf213d98", # Message GUID
value, uniquely identifying this message
```

```
  "data": { } # Stores message data content
}
```

## 3.3 View Software Serial Number (ACCID)

- Connect to the robot's wireless network
  - After the robot is powered on, use a personal computer to connect to the robot's Wi-Fi, whose name format is usually 「DA_TRON2A_xxx」
  - Enter Wi-Fi Password:12345678
- Enter`http://10.192.1.2:8080`in the browser to access the "Robot Information Page" and view robot information. As shown in the figure below, the SN (Serial Number) displayed on the page isDA_TRON2A_001, whereDA_TRON2A_001is the software serial number of this robot.

## 3.4 Communication Testing Method

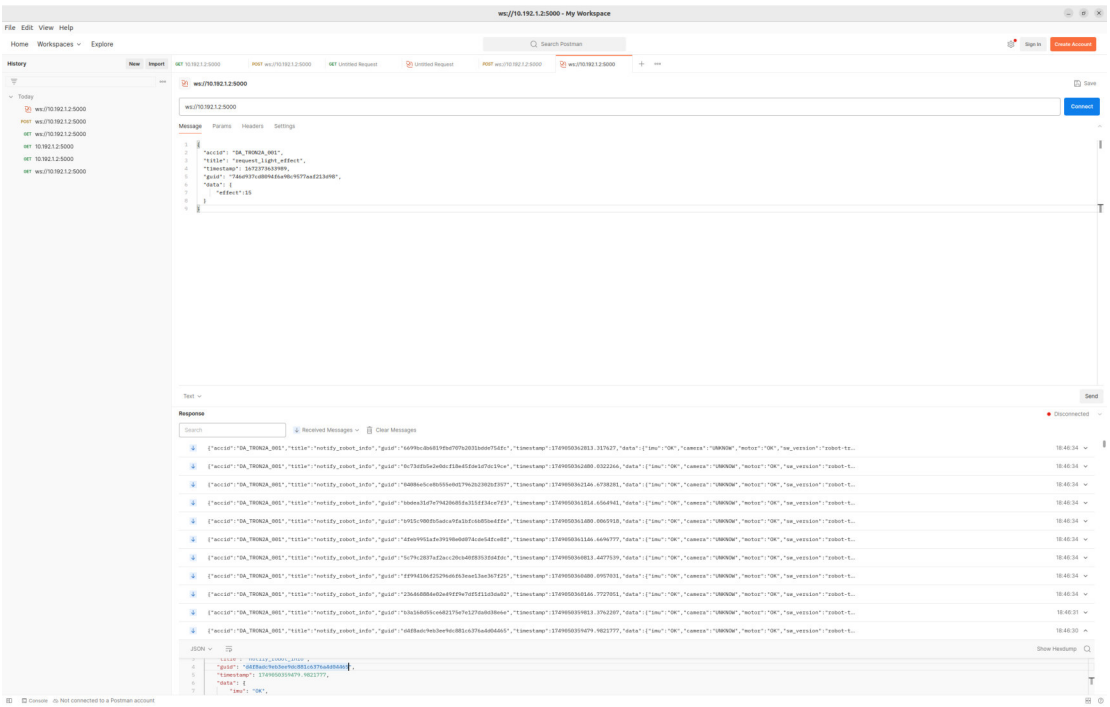
Postman is a popular API Development Environment that can be used to test WebSocket interfaces. To test WebSocket interfaces using Postman, please follow these steps:

1. Install the PostmanClient, download address:https://www.postman.com/downloads/?utm_source=postman-home;
   - Under the Windows environment, you can directly click the exe file to install.
   - Under the Ubuntu environment, the following commands can be used to install and uninstall via snap

```Shell
#Installation command
sudo apt update
sudo snap install postman
#Uninstallation command
sudo snap remove postman
```

2. Open Postman and create a WebSocket request;
3. Connect to the robot's wireless network ;
   - After the robot has completed booting up, use a personal computer to connect to the robot's Wi-Fi
   - Enter Wi-Fi password:12345678
4. Enter the address of the WebSocket interface in the requested URL, for example, "ws://10.192.1.2:5000";
5. In "Message", enter the instruction request to be sent;
6. Click the "Send" button to send the request command;

7. After sending the command, you can receive a response message from the server. Use Postman's response window to view the data returned by the server and check if it matches the expected results.



## 3.5 Protocol Interface Definition

The robot interface design follows the same process and state transitions as remote control operation, ensuring that the call sequence, response timing, and state transitions are strictly aligned with the remote control logic. Users can obtain an intuitive experience similar to using a remote control through interface calls, while also supporting seamless switching between the remote control and the interface to achieve a unified and stable robot control effect.

### 3.5.1 Introduction to MoveJ Interface

#### 3.5.1.1 Request: request_movej

Note:

- When moveJ is called, it will check whether the joint angle of each joint has reached the limit. If it exceeds the limit, an error will be reported for over-limit, and this call will not be executed.

  moveJ Joint Limits (in the order from top to bottom of the left arm joints -> top to bottom of the right arm joints):

  Upper limit: [2.5994, 2.9671, 1.4835, 0.5236, 1.3963, 0.7854, 1.5708, 2.5994, 2.9671, 1.4835, 0.5236, 1.3963, 0.7854, 1.5708]

  Lower Limit: [-3.1416, -0.2618, -3.6652, -2.618, -1.7453, -0.7854, -1.5708, -3.1416, -0.2618, -3.6652, -2.618, -1.7453, -0.7854, -1.5708]

- Only 14 joint values need to be input

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this should be modified to your robot's serial number;
  "title": "request_movej",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "time": 5, // Move to the specified position in 5 seconds
    "joint": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  //14 joint
values, unit: rad
  }
}
```

Example:

```
JSON
{
    "accid": "DACH_TRON2A_026",
    "title": "request_movej",
    "timestamp": 1672373633989,
    "guid": "746d937cd8094f6a98c9577aaf213d98",
    "data": {
        "time": 5,
        "joint": [0,0,0,-1.6,0,0,0,0,0,0,-1.6,0,0,0]
    }
}
```

#### 3.5.1.2 Response: response_movej

```
JSON
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this should be modified to your robot's serial number;
  "title": "response_movej",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: success: success,
fail_invalid_cmd: incorrect command (missing field)
  }
}
```

### 3.5.2 Introduction to MoveP Interface

#### 3.5.2.1 Request: request_movep

Note:

- When moveP is called, it will check whether the end positions of the left and right hands are within the reachable range. If they exceed the limit, an out-of-limit error will be reported, and this call will not be executed.

  moveP Joint Limit (in accordance with x_min, x_max, y_min, y_max, z_min, z_max, unit: m)

  Left Arm: [[0.250, 0.732], [-0.213, 0.900], [-0.673, 0.5]]

  Right Arm: [[0.250, 0.732], [-0.900, 0.213], [-0.673, 0.5]]

- There is no restriction on the rotation attitude, but it is recommended that the rpy Euler angles be kept between [-90°, 90°]. If only position control is required, it is recommended that the attitude be kept as a 3*3 identity matrix.

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this should be modified to your robot's serial number;
  "title": "request_movep",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "time": 5, //Move to this position in 5 seconds
    "pos": []  //left_position(x,y,z) + spatial rotation
matrix matrix 3*3 + right_position(x,y,z) + spatial rotation
matrix matrix 3*3, unit: m, rad.
  }
}
```

Example:

```
JSON
{
  "accid": "DACH_TRON2A_026", // Robot serial number, note that
this should be modified to your robot's serial number;
  "title": "request_movep",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "time": 5, //Move to this position in 5 seconds
    "pos": [0.3,0.2,-0.3,1,0,0,0,1,0,0,0,1,0.3,-0.2,-
```

```
0.3,1,0,0,0,1,0,0,0,1]
  }
}
```

#### 3.5.2.2 Response: response_movep

```
JSON
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this should be modified to your robot's serial number;
  "title": "response_movep",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: success,
fail_invalid_cmd: incorrect command (missing field)
  }
}
```

### 3.5.3 ServoJ Control Command

#### 3.5.3.1 Request: request_servoj

- It is recommended to control the movement of the robotic arm in a real-time system according to the requirement of control frequency >= 500Hz to ensure control effectiveness and stability.

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this should be modified to your robot's serial number;
  "title": "request_servoj",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "q": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    "v": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    "kp": [100.0, 110.0, 120.0, 130.0, 140.0, 150.0, 160.0,
170.0, 180.0, 190.0, 200.0, 210.0, 220.0, 230.0],
    "kd": [10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0,
18.0, 19.0, 20.0, 21.0, 22.0, 23.0],
    "tau": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    "mode": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    "na": 0
  }
```

```
}

// Please note that you should replace the above parameters
with the actual
```

#### 3.5.3.2 Response: None

#### 3.5.3.3 Message Push: notify_servoJ

When the ServoJ control operation fails to execute, the server will proactively push this message to inform the Client of the reason for the failure.

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001",
  "title": "notify_servoJ",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "fail_invalid_cmd"  # fail_invalid_cmd: Illegal
command; fail_motor: Motor error.
  }
}
```

### 3.5.4 Get robot joint status

#### 3.5.4.1 Request：request_get_joint_state

This request is used to obtain the status of each joint of the robot.

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, please
note the serial number of the robot being repaired;
  "title": "request_get_joint_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

#### 3.5.4.2 Response：response_get_joint_state

After receiving the request, return the current state of each joint of the robot.

```
json
代码块
{
  "accid": "DACH_TRON2A_001",
  "title": "response_get_joint_state",
```

```
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "timestamp": 1672373633989,
  "data": {
      "result": "success"  # fail_not_data
      "timestamp": 0,
      "names": [], # Names of each joint
      "q": [],     # Positions of each joint
      "dq": [],    # Velocities of each joint
      "tau": [],   # Torque of each joint
  }
}
```

### 3.5.5 Obtain the pose of the end of both arms

#### 3.5.5.1 Request：request_get_move_pose

Obtain the end pose information of the robot's dual arms through this interface.

```
JSON
{
  "accid": "DACH_TRON2A_001", // Robot serial number, please
note the serial number of the robot being repaired;
  "title": "request_get_move_pose",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

#### 3.5.5.2 Response：response_get_move_pose

After receiving the request, return the relevant information about the current pose of the dual arms.

```
json
{
  "accid": "DACH_TRON2A_001",
  "title": "response_get_move_pose",
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "timestamp": 1672373633989,
  "data": {
      "left_position": [0.0, 0.0, 0.0], # Represents the
position of the left arm's end in meters, in the order of x,
y, z
      "left_quat": [0.0, 0.0, 0.0, 1.0], # Represents the pose
of the left arm's end, expressed as a quaternion, in the order
of w, x, y, z
      "right_position": [0.0, 0.0, 0.0], # Represents the
position of the right arm's end in meters, in the order of x,
```

```
y, z
      "right_quat": [0.0, 0.0, 0.0, 1.0], # Represents the
pose of the right arm's distal end, expressed as a quaternion
in the order w, x, y, z
      "timestamp": 1672373633989, # Represents the data
timestamp in milliseconds
      "result": "success"  # fail_not_data
  }
}
```

#### 3.5.5.3 Message Push: None

### 3.5.6 Emergency Stop

#### 3.5.6.1 Request: request_emgy_stop

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this is your robot's serial number;
  "title": "request_emgy_stop",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {}
}
```

#### 3.5.6.2 Response: response_emgy_stop

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001",
  "title": "response_emgy_stop",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  // success: success, fail_imu: IMU
error, fail_motor: motor error
  }
}
```

#### 3.5.6.3 Message Push: None

### 3.5.7 Set lighting effects

#### 3.5.7.1 Request: request_light_effect

This interface is used to set the lighting effects of the robot. Users can send specific lighting effect instructions to the robot through this interface, and the robot will adjust its lighting display according to the instructions.

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001", // Robot serial number, note
that this is your robot's serial number;
  "title": "request_light_effect",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "effect": 1
  }
}
```

Request Parameter Description:

| Parameter Name | Type | Description |
| --- | --- | --- |
| data.effect | Number | The number of the lighting effect corresponds to different lighting display modes, and the specific mapping relationships are as follows:<br>1: STATIC_RED (Static Red)<br>2: STATIC_GREEN (Static Green)<br>3: STATIC_BLUE (Static Blue)<br>4: STATIC_CYAN (Static Cyan)<br>5: STATIC_PURPLE (Static Purple)<br>6: STATIC_YELLOW (Static Yellow)<br>7: STATIC_WHITE (Static White)<br>8: LOW_FLASH_RED (Low Frequency Flashing Red)<br>9: LOW_FLASH_GREEN (Low Frequency Flashing Green)<br>10: LOW_FLASH_BLUE (Low Frequency Flashing Blue)<br>11: LOW_FLASH_CYAN (Low Frequency Flashing Cyan)<br>12: LOW_FLASH_PURPLE (Low Frequency Flashing Purple)<br>13: LOW_FLASH_YELLOW (Low Frequency Flashing Yellow)<br>14: LOW_FLASH_WHITE (Low Frequency Flashing White)<br>15: FAST_FLASH_RED (High-frequency flashing red)<br>16: FAST_FLASH_GREEN (High-frequency flashing green)<br>17: FAST_FLASH_BLUE (High-frequency flashing blue)<br>18: FAST_FLASH_CYAN (High-frequency flashing cyan) |

19: FAST_FLASH_PURPLE (High Frequency Flashing Purple)
20: FAST_FLASH_YELLOW (High Frequency Flashing Yellow)
21: FAST_FLASH_WHITE (High Frequency Flashing White)

#### 3.5.7.2 Response: response_light_effect

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001",
  "title": "response_light_effect",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "effect": 1,
    "result": "success"  // success: success,
fail_light_effect: failure
  }
}
```

#### 3.5.7.3 Message Push: None

### 3.5.8 Global Message

#### 3.5.8.1 Robot Basic Information

Robot basic information is reported once per second, including the following content:

- `accid`: Robot Serial Number
- `title`：notify_robot_info
- `timestamp`: Timestamp when the message was sent, in milliseconds
- `guid`: The guid value of the message, uniquely identifying this message
- `data`: Stores message content, example as follows:

```
JSON
代码块
{
  "accid": "DACH_TRON2A_001",   # Robot serial number
  "title": "notify_robot_info",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "imu": "OK",    # Robot IMU diagnostic information
```

```
    "camera": "OK", # Robot camera diagnostic information
    "motor": "OK",  # Robot motor diagnostic information
    "sw_version": "robot-tron2-2.0.10.20241111103012", #
Robot hardware / chassis software version information
    "battery": 95,  # Robot battery level
    "status": "UNKNOW" # Reserved field
  }
}
```

#### 3.5.8.2 Illegal instruction message

Send this message when the robot receives a request instruction in an illegal format, including the following content:

- `accid`: Robot Serial Number
- `title`：notify_invalid_request
- `timestamp`: Timestamp when the message was sent, in milliseconds
- `guid`: The guid value of the message, uniquely identifying this message
- `data`: Stores message content
- Examples are as follows:

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",
  "title": "notify_invalid_request",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": "Returns the original request content to facilitate
client troubleshooting"
}
```

## 3.6 Example of Protocol Interface Call

### 3.6.1 C++ Example Implementation

- Install dependencies: Taking the Ubuntu 20.04 system as an example, install the websocketpp, nlohmann/json, and boost dependencies:

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

- Compile code

```
Bash
```

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

- Run the program

```
Bash
./websocket_client
```

- Implementation of websocket_client.cpp

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

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

// Replace this value with the actual serial number (SN) of
the robot.
static std::string ACCID = "";

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

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

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

// Connection handle for sending messages
static connection_hdl current_hdl;

// Generate dynamic GUID
```

```
static std::string generate_guid()
{
  boost::uuids::random_generator gen;
  boost::uuids::uuid u = gen();
  return boost::uuids::to_string(u);
}

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

  // Adding necessary fields to the message
  message["accid"] = ACCID;
  message["title"] = title;
  message["timestamp"] =
std::chrono::duration_cast<std::chrono::milliseconds>(

std::chrono::system_clock::now().time_since_epoch())
                              .count();
  message["guid"] = generate_guid();
  message["data"] = data;

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

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

// Handle user commands
void handle_commands()
{
  std::cout << "Enter command ('movej', 'movep', 'light',
'stop') or 'exit' to quit:\n";
  while (!should_exit)
  {
    std::string command;
    std::cin >> command;

    if (command == "exit")
    {
      should_exit = true;
```

```
      return;
    }
    else if (command == "movej")
    {
      nlohmann::json data = {
          {"joint", {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
0}},
          {"time", 5}};
      send_request("request_movej", data);
    }
    else if (command == "movep")
    {
      nlohmann::json data = {
          {"pos", {0.3, 0.2, -0.3, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0.3, -0.2, -0.3, 1, 0, 0, 0, 1, 0, 0, 0, 1}},
          {"time", 5}};
      send_request("request_movep", data);
    }
    else if (command == "light")
    {
      send_request("request_light_effect", {{"effect", 1}});
    }
    else if (command == "stop")
    {
      send_request("request_emgy_stop"); // Send emergency
stop request
    }

    sleep(1);

    std::cout << "Enter command ('movej', 'movep', 'light',
'stop') or 'exit' to quit:\n";
  }
}

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

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

  // Start handling commands in a separate thread
```

```
  std::thread(handle_commands).detach();
}

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

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

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

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

socket.set_option(websocketpp::lib::asio::socket_base::send_bu
ffer_size(sendBufferSize), ec);

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

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

socket.set_option(websocketpp::lib::asio::socket_base::receive
_buffer_size(recvBufferSize), ec);

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

    // Disable Nagle's algorithm to reduce latency

socket.set_option(websocketpp::lib::asio::ip::tcp::no_delay(tr
ue), ec);
```

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

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

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

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

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

// Close WebSocket connection
static void close_connection(connection_hdl hdl)
{
```

```
  ws_client.close(hdl, websocketpp::close::status::normal,
"Normal closure"); // Close connection normally
}

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


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

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

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

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

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

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

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

```
  return 0;
}
```

### 3.6.2 Python Example Implementation

- Environment Preparation: Taking Ubuntu 22.04 system and python3.10.4 version as an example, install the following dependencies

```
Python
sudo apt install python3-dev python3-pip
sudo pip3 install websocket-client
```

- Run Script

```
Python
python3 websocket_client.py
```

- Implementation of websocket_client.py

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

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

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

# Atomic flag for graceful exit
should_exit = False

# WebSocket client instance
ws_client = None

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

# Send WebSocket request with title and data
def send_request(title, data=None):
```

```
    global ACCID
    if data is None:
        data = {}

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

    message_str = json.dumps(message)

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

# Handle user commands
def handle_commands():
    global should_exit
    while not should_exit:
        command = input("Enter command ('movej', 'movep',
'light', 'stop') or 'exit' to quit:\n")

        if command == "exit":
            should_exit = True  # Set exit flag to stop the
loop
            break
        elif command == "movej":
            send_request("request_movej", { # request_movej
              "joint": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0 ,0],
              "time": 5
            })
        elif command == "movep":
            send_request("request_movep", { # request_movep
              "pos": [0.3, 0.2, -0.3, 1, 0, 0, 0, 1, 0, 0, 0,
1, 0.3, -0.2, -0.3, 1, 0, 0, 0, 1, 0, 0, 0, 1],
              "time": 5
            })
```

```
        elif command == "light":
            send_request("request_light_effect", { #
request_light_effect
              "effect": 1
            })
        elif command == "stop":
            send_request("request_emgy_stop", { #
request_emgy_stop
            })

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

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

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

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

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

def main():
    global ws_client

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

```
        on_close=on_close
    )

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

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

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

if __name__ == "__main__":
    main()
```

# 4. External component development interface

## 4.1 Two-Finger Gripper

### 4.1.1 Claw Control Instruction

#### 4.1.1.1 Request：request_set_limx_2fclaw_cmd

This protocol controls the grasping action of the gripper.

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",
  "title": "request_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      # Providing the following data will control the movement
of the left gripper
      "left_opening": 50,  # Opening degree, 0-100, normalized
value (0 corresponds to minimum closure, 100 corresponds to
maximum opening)
```

```
      "left_speed": 50,    # Gripper speed, 0~100, normalized
value (Higher values correspond to higher speeds the speed)
      "left_force": 50,   #Force, gripper clamping force,
0~100, unitless (the larger the value, the greater the force)

      # Providing the following data will control the movement
of the right gripper
      "right_opening": 50,  # Opening degree, 0-100,
normalized value (0 corresponds to minimum closure, 100
corresponds to maximum opening)
      "right_speed": 50,    # Gripper speed, 0~100, normalized
value (Higher values correspond to higher speeds the speed)
      "right_force": 50   #Force, gripper clamping force,
0~100, normalized value (the larger the value, the greater the
force)
  }
}
```

#### 4.1.1.2 Response：response_set_limx_2fclaw_cmd

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",
  "title": "response_set_limx_2fclaw_cmd",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
    "result": "success"  # success: success, fail_motor: motor
error
  }
}
```

#### 4.1.1.3 Message Push: None

### 4.1.2 Get gripper status information

#### 4.1.2.1 Request：request_get_limx_2fclaw_state

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",
  "title": "request_get_limx_2fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
  }
```

```
}
```

#### 4.1.2.2 Response：response_get_limx_2fclaw_state

Return gripper status information

```
JSON
代码块
{
  "accid": "DA_TRON2A_001",
  "title": "response_get_limx_2fclaw_state",
  "timestamp": 1672373633989,
  "guid": "746d937cd8094f6a98c9577aaf213d98",
  "data": {
      "timestamp": 1672373633989, # Represents the data
timestamp, in milliseconds

      "left_opening": 50,  # Opening degree, 0-100, normalized
value (0 corresponds to minimum closure, 100 corresponds to
maximum opening)
      "left_speed": 50,    # Gripper speed, 0~100, normalized
value (Higher values correspond to higher speeds the speed)
      "left_force": 50,   #Force, gripper force, 0~100,
normalized value (the larger the value, the greater the force)

      "right_opening": 50,  # Opening degree, 0-100,
normalized value (0 corresponds to minimum closure, 100
corresponds to maximum opening)
      "right_speed": 50,    # Gripper speed, 0~100, normalized
value (Higher values correspond to higher speeds the speed)
      "right_force": 50,   #Force, gripper force, 0~100,
unitless (the larger the value, the greater the force)

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

#### 4.1.2.3 Message Push: None

# 5. VRData AcquisitionUser Tutorial

Data Acquisition Software User Manual for Tron2

# 6. Software Upgrade

## 6.1 Robot Firmware Upgrade

We access the robot management page through the browser and select the robot software version downloaded locally in advance for upgrade. The specific steps are as follows:

1. Please select and connect to your robot's Wi-Fi hotspot, password:`12345678`
2. Use the Shell command `ping 10.192.1.2` to ensure the connection is normal
3. Access the Management Page:
    - Enter the following in the browser address bar:http://10.192.1.2:8080 to access the robot management page.
4. Select and Upgrade Software:
    - Select "Version Management -> Select File -> Upgrade" in sequence.
    - After the upgrade is completed, the robot's main control computer will automatically restart.

# Tron2 (v4) Firmware Package

**Software Category**

**Version Number**

**Release Date**

**Software Name**

**Software Package**

**Update Instructions**

**Master Software Version**

**r-1.2.25**

20260113

**robot-tron2-r-1.2.25.20260107211002.tar.gz**

**[robot-tron2-r-1.2.25.20260107211002.tar.gz]**

**Main Station Board Version**

**1.0.25**

20260113

**ecm_V1.0.25_20251230.tar.gz**

**[ecm_V1.0.25_20251230.tar.gz]**

**Driver Version**

**1.2.23**

20260113

**EcMotor_HU_V1.2.23_20251202_174735.bin**

**[EcMotor_HU_V1.2.23_20251202_174735.bin]**

**Power Distribution Board Version**

**1.0.4**

20260113

**PMSGen2_T2_V1.0.4_20251204.bin**

**[PMSGen2_T2_V1.0.4_20251204.bin]**

**Remote Control Version**

**1.0.2**

20251016

**RCGen2_T2_V1.0.2_20250919**

**[RCGen2_T2_V1.0.2_20250919.bin]**

**Studio Version**

**0.1.39**

20260113

**V0.1.39**

**[limx-studio_0.1.39_amd64.deb]**

**Note: studio0.1.39 must be used in conjunction with the master control software version 1.2.25!**

Studio Desktop Function Items:

1. Studio has added the TRON2 device management function, supporting wheeled-leg, foot, and dual-arm configurations.

2. Studio TRON2 adds a new real device management module, which displays basic information, device information, and battery information.

3. Studio TRON2 has added a new version management module, which displays version information, upgrade management, and supports software package upload and upgrade.

4. Studio TRON2 adds a new robot settings module, which displays camera angle settings, desktop height recognition, camera angle control and arc control, and head tracking VR functionality.

5. Requirement change: Function adjustment, hiding the desktop height recognition function.

6. Add a mobile camera, with arc echo.

7. Adjust the robot settings layout to increase compatibility.

## 6.2 VR Remote Operation Software Upgrade

1. Connect Quest 3 to the computer using a Type-C cable, click on the USB pop-up window with the controller, and allow the computer to access Quest 3's internal data

2. Download the remote control software to your computer

**[LimXTeleopTronV4_3.apk]**

3. Install the adb tool on your computer

```Shell
sudo apt install android-tools-adb
```

After the installation is complete, you can enter the command `adb devices` to check if it is connected to Quest 3

4. Install the remote operation software on the Quest 3 device

```Shell
adb install -r LimXTeleopTronV4_3.apk
```

# 7. Frequently Asked Questions

| **Serial N** | **Question** | Common Causes | Solution |
| --- | --- | --- | --- |

| number | | | |
|---|---|---|---|
| 1 | Accessing`http://10.192.1.2:8080`has no response | The computer is not in the same network segment as 10.192.1.2 | 1. First ping 10.192.1.2. If it is successful, clear the browser data, or switch to another browser and try to access again<br>2. If the ping fails, modify the local address, turn off DHCP, and set DNS to 223.5.5.5 or 8.8.8.8 |
| 2 | How to configure the camera SN number | | Camera SN Configuration Method |
| 3 | | | |
| 4 | | | |
| 5 | | | |

**LimX Dynamics Technology Co, Ltd**

**Address:**15th Floor, Building E, Nanshan Zhigu Industrial Park, No. 3157, Shahe West Road, Nanshan District, Shenzhen City

**Website:**https://limxdynamics.com/