Dex3-1 Dexterous Hand

User Manual V1.0

Unitree Robotics

Unitree

This product is a civilian robot. We kindly request that all users refrain from making any dangerous modifications or using the robot in a hazardous manner.

Please visit Unitree Robotics Website for more related terms and policies, and comply with local laws and regulations.

Catalogue

Unitree Robotics

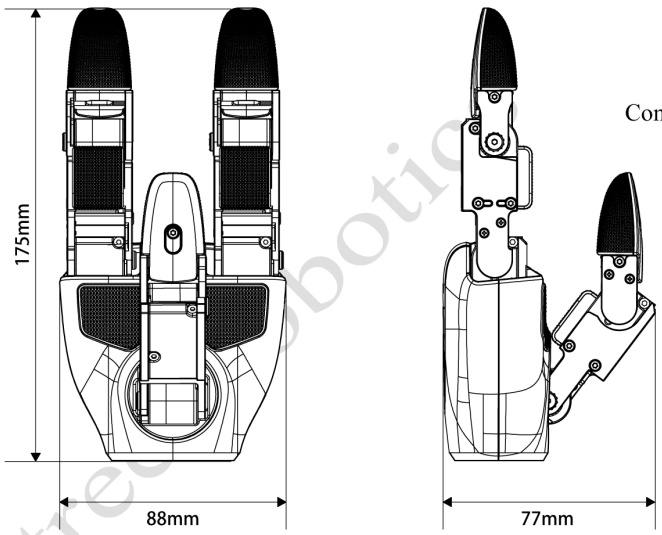
Dex3-1 Dexterous Hand

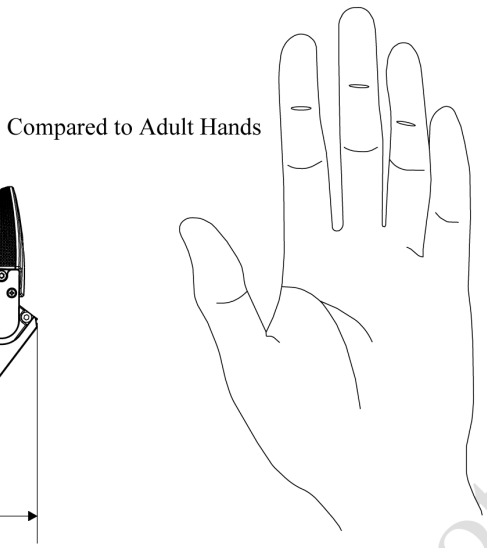
Introduction

The Dex3-1 three-fingered dexterous hand is a high-performance robotic hand actuator independently developed by Unitree Robotics. The weight of the dexterous hand is 710g, and its dimensions are 175mm × 88mm × 77mm. It has 7 degrees of freedom, including 3 for the thumb, 2 for the index finger, and 2 for the middle finger, providing rich possibilities for hand movements. It adopts 6 micro brushless force-controlled joints with direct drive and 1 micro brushless force-controlled joint with gear transmission, ensuring precise motion control and efficient energy transmission.

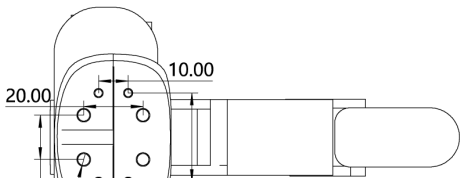
The Dex3-1 dexterous hand combines force-position hybrid control technology. It can be optionally equipped with a multi-point tactile array to achieve multi-point touch and simulate the fine touch of the human body. The dexterous hand is equipped with 33 tactile sensors, with an array resolution of 2×2 (palm) × 3, 2×2 (single fingertip) × 3, and 3×1 (single fingertip) × 3. It can cooperate with the G1 humanoid robot to perform various complex actions and is suitable for fields such as scientific research and industry.

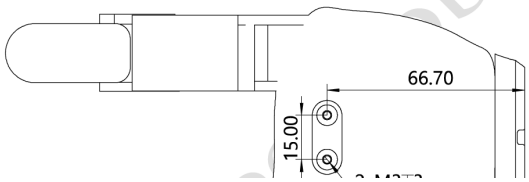
Dimensions and Structure

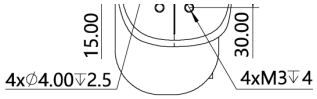


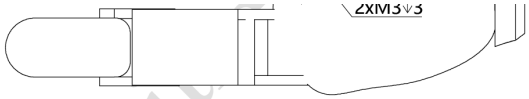


Mounting Holes:





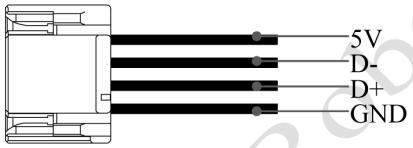
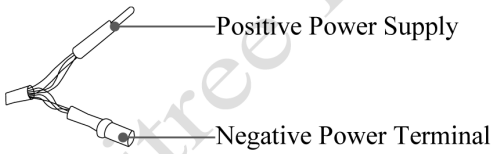




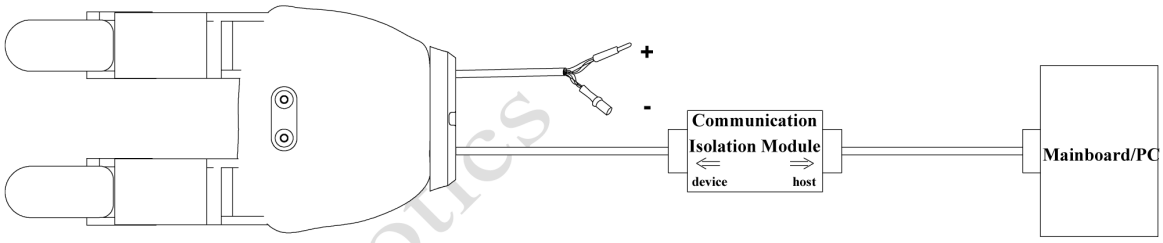
© 2025 All rights reserved, Unitree Robotics

Electrical Connections

The electrical interface of the Dex3-1 three-fingered dexterous hand uses lead wire connections for output, including communication external wiring and power supply wires. The definitions of the electrical connection interfaces are as follows.

Interface Type	Cable Length	Definition
USB Communication Interface	1150mm (Including Internal)	
Power Interface	90mm (Including Internal)	

Connection Schematic:



Specifications

Body Parameters

Weight	710g
Size [1]	175mm x 88mm x 77mm
Degrees of Freedom	Thumb x 3 Index Finger x 2 Middle Finger x 2

Transmission Mechanism	6 Motors Drive Directly 1 Motor Drive with Gears

© 2025 All rights reserved, Unitree Robotics

Angle of Joint	Thumb Joint 0: -60°~60° Thumb Joint 1: -35°~60° The Thumb Joint 2: 0 to 100°	Index Finger Joint 0: 0 °~90° Index Finger Joint 1: 0 °~100° Middle Finger Joint 0: 0 °~90° Middle Finger Joint 1: 0 °~100°
Working Voltage	24~58V	
Static Current	58V @ 0.2A	
Maximum Current	10A	
Communication Interface	USB2.0	
Working Temperature Range	-20℃ ~ 60℃	
Load Conditions (Palm Facing Down At Room Temperature, Grasping A 5cm Round Hard Object)	The Maximum Weight Is: 500g	The Maximum Power Is: 400W@3s
Load Condition (Palm Facing Left At Room Temperature, Grasping A 5cm Round Hard Object)	The Maximum Weight Is: 500g	The Maximum Power Is: 400W@3s

Sensing parameters

Number of Pressure Sensors	9 (Total of 33 Pressure Sensors)
Array resolution	2 x 2 (Palm) x 3 2 x 2 (Single Finger Pulp) x 3 3 x 1 (Single Fingertip) x 3
Range of Perception	10g-2500g

Maximum Acceptance (Unamaged) [2]	20kg

Software functions

Communication rate	1000Hz

© 2025 All rights reserved, Unitree Robotics

The number of bytes in a full packet	Sender: 1234 bytes Receiver: 1270 bytes
Perceptual feedback	Joint Mode, Joint Position, Joint Velocity, Joint Torque, Joint Temperature, Voltage and Current, Sensor Pressure Value, Sensor Temperature Value, imu Data
Control feedback	Joint Mode, Joint Position, Joint Velocity, Joint Torque, Joint Stiffness Coefficient, Joint Damping Coefficient

Miniature Brushless Force Control Joint Parameters

Model	F-1515-108 Miniature Brushless Force Control Joint	F-1515-214 Miniature Brushless Force Control Joint
Weight	45g	
Size	34.8mm x 23.1mm x 23.3mm	
Reduction Ratio	1：108	1：214
Maximum Torque (Ideal)	0.76Nm	1.498Nm
Maximum Torque (Actual)	The Torque Is In The Same Direction As The Speed: 0.49Nm The Toque Is In The Opposite Direction of The Rotational Speed: 1.37Nm	The Torque Is In The Same Direction As The Speed: 0.86Nm The Toque Is In The Opposite Direction of The Rotational Speed: 3.1 Nm
Maximum Rotational Speed	23rad/s	11rad/s
Maximum Line Current	4.76A	
Communication Control		

Communication Control Frequency	1000Hz`
Method of Communication	High Speed Unibus
Encoder	Rotor Absolute Value Encoder
Working Temperature Range	-20℃ ~ 60℃

© 2025 All rights reserved, Unitree Robotics

IP Rating	/
Working Voltage	12-24V
Motor Preceived Feedback	Torque, Angle, Angular Velocity, Temperature
Motor Control Instruction	Torque, Angle, Angular Velocity, Stiffness, Damping

[1] It is the size of the dexterous hand in flat state.

[2] The data refers to the pressure exerted by the dexterous hand when it is pressed by a vertical downward cylinder with a diameter of 1cm, which varies in different application scenarios. Please refer to the actual situation.

Communication Protocols

The communication protocol of the dexterous hand is currently divided into the bottom layer and the upper layer. The bottom layer protocol is used when the dexterous hand is used alone with USB. The upper layer communication protocol is used to control the dexterous hand through the DDS protocol after the dexterous hand is connected to the Unitree product via the encapsulated SDK. This system aims to achieve precise control and status monitoring of the robot hand and ensure the accuracy and real-time performance of hand movements through an efficient data transmission and processing mechanism.

USB Serial Bus Communication Method

USB Communication Method

When the device is inserted into the computer, the following information can be seen:

```
usb 1-2: new high-speed USB device number 8 using tegra-xusb
usb 1-2: New USB device found, idVendor=34b7, idProduct=1010, bcdDevice= 1.00
```

```
usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-2: Product: Dexthree LeftHand
usb 1-2: Manufacturer: UnitreeTC
usb 1-2: SerialNumber: 2024061800
```

It is recommended to use the lsusb command to check the device status. If there is no response when the device is inserted, please check the device power supply or contact the after-sales service. When using the USB communication method, it is recommended to use the libusb library for communication.

```
#include <libusb-1.0/libusb.h>
```

© 2025 All rights reserved, Unitree Robotics

USB Communication Method Code Reference:

```
class LinuxSendMcu
{
public:
    LinuxSendMcu() : handle(nullptr) {}
    ~LinuxSendMcu() { close(); }

    int init(int LorR)
    {
        int ret = libusb_init(NULL);
        if (ret < 0) {
            LOGE() << "Failed to initialize libusb: " << libusb_error_name(ret);
            return -1;
        }

        uint16_t device_id = LorR ? RIGHT_HAND_VENDOR : LEFT_HAND_VENDOR;
        uint16_t product_id = LorR ? RIGHT_HAND_PRODUCT : LEFT_HAND_PRODUCT;
        handle = libusb_open_device_with_vid_pid(NULL, device_id, product_id);
        if (!handle) {
            LOGE() << "Failed to open device";
            return -1;
        }

        libusb_detach_kernel_driver(handle, 0);

        ret = libusb_claim_interface(handle, 0);
        if (ret < 0) {
            LOGE() << "Failed to claim interface: " << libusb_error_name(ret) ;
            return -1;
        }

        LOGE() << "Device initialized successfully" ;
        return 0;
    }
```

```
    }

    int sendCommand(const LinuxToMcuCmd_t& cmd)
    {
        int actual_sendlen;
        return sendRevData((uint8_t*)&cmd, sizeof(LinuxToMcuCmd_t), &actual_sendlen, nullptr, 0, nullptr, 0);
    }

    int receiveState(McuToLinuxState_t& state)
    {
        //const size_t BUFFER_SIZE = 4096;  // Use a large enough buffer
        uint8_t buffer[sizeof(McuToLinuxState_t)];
        int actual_revlen;
        int ret = sendRevData(nullptr, 0, nullptr, buffer, sizeof(McuToLinuxState_t),
        &actual_revlen, 1000);
```

© 2025 All rights reserved, Unitree Robotics


```

        if (ret == 0) {
            if (actual_revlen >= sizeof(McuToLinuxState_t)) {
                // Copy only the data we need
                memcpy(&state, buffer, sizeof(McuToLinuxState_t));

                if (actual_revlen > sizeof(McuToLinuxState_t)) {
                    LOGE() << "Received " << actual_revlen << " bytes, but only used "
                           << sizeof(McuToLinuxState_t) << " bytes" ;
                }
            } else {
                LOGE() << "Received data size too small. Expected at least " << sizeof(McuToLinuxState_t)
                       << " bytes, but received " << actual_revlen << " bytes" ;
                return -1;
            }
        }

        return ret;
    }

    void close()
    {
        if (handle) {
            libusb_release_interface(handle, 0);
            libusb_close(handle);
            handle = nullptr;
        }
        libusb_exit(NULL);
    }

```

```
77  private:
78      int sendRevData(uint8_t* sendData, int target_sendlen, int* actual_sendlen,
79                      uint8_t* revData, int target_revlen, int* actual_revlen,
80                      uint32_t timeout)
81      {
82          int ret = 0;
83
84          if (sendData && target_sendlen > 0) {
85              ret = libusb_bulk_transfer(handle, OUT_EP, sendData, target_sendlen, actual_sendlen,
86  0);
                if (ret != 0) {
87                  LOGE() << "Error sending data: " << libusb_error_name(ret);
88                  return ret;
89              }
90          }
91
92          if (revData && target_revlen > 0) {
```

© 2025 All rights reserved, Unitree Robotics

```
93              ret = libusb_bulk_transfer(handle, IN_EP, revData, target_revlen, actual_revlen,
94  timeout);
95              if (ret != 0) {
96                  LOGE() << "Error receiving data: " << libusb_error_name(ret);
97                  return ret;
98              }
99          }
100
101         return ret;
102     }
103
104     libusb_device_handle* handle;
105     unsigned char w_buf[BUF_SIZE];
106     unsigned char r_buf[BUF_SIZE];
107 };
```

When the device uses the USB communication method, it adopts a one-send-one-receive mode. When sending control commands to the dexterous hand, the bottom MCU will return the current status to the upper control terminal. The send and receive structures will be further explained in the following code.

USB Device ID

Left Hand

```
1  LEFT_HAND_VENDOR 0x34B7
2  LEFT_HAND_PRODUCT 0x1010
```

Right Hand

```
RIGHT_HAND_VENDOR 0x34B7
RIGHT_HAND_PRODUCT 0x1011
```

Define the necessary number of finger motors and sensors and the send and receive buffers:

```
#define SUPPORTED_FINGER_NUM 5
#define SUPPORTED_SENSOR_NUM 12
#define SUPPORTED_SEND_BUFFER_SIZE 140 //20*7 3+4
#define SUPPORTED_RECV_BUFFER_SIZE 240 //44*4+3*20 = 236约 240
#define BUF_SIZE 4096
```

Introduction to Send and Receive Structures

All structures are aligned by one byte.

Motor Control Structure

© 2025 All rights reserved, Unitree Robotics


```
typedef struct {          // Motor control command data packet 20 bytes
    uint8_t   head[2];    // Packet header 0xFE 0xEE Participate in CRC
    RIS_Mode_t mode;      // Motor control mode 1Byte
    uint8_t   res;       // reserved
    int16_t   tor_des;    // Expected joint output torque 256 represents 1mNM
    int16_t   spd_des;    // Expected joint output velocity 256/2π means 100 rad/s
    int32_t   pos_des;    // Expected joint output position 32768/2π represents 1rad
    int16_t   k_pos;      // Expected joint stiffness coefficient 1280 means 1mN.m/rad
    int16_t   k_spd;      // Expected joint damping coefficient 1280 means 1mN.m/100rad/s
    uint32_t  CRC32;      // CRC32
} MotorCmd_t;
```

The above structure explains the header, control content, and CRC checksum in the motor. The CRC checksum will be given at the end.

```
typedef struct {
    uint8_t id     : 4;     // Motor ID: 0,1...,13,14 15 means broadcasting data to all motors (no return at this time)
    uint8_t status : 3;     // Working mode: 0.Lock 1.FOC 6
    uint8_t timeout: 1;     // Master->Motor: 0. Disable timeout protection 1. Enable (default 1s timeout)
                        // Motor->Master: 0. No timeout 1. Timeout protection is triggered (need to control the bit to send 0 to clear)
} RIS_Mode_t;             // Control mode 1Byte
```

The above structure explains the RIS_Mode_t structure, which represents the motor ID, motor mode, and

timeout configuration.

Motor Status Receive Structure

```
typedef struct {          // Motor return data 26 bytes
    uint8_t   head[2];    // Packet header 0xFC 0xEE Not involved in CRC
    RIS_Mode_t mode;      // Motor control mode 1Byte mode.states updated in the current loop
    int8_t    temp;       // Case temperature: -128~127°C
    uint8_t   sensor;     // Winding temperature: 0-255°C
    uint8_t   vol;        // Motor terminal voltage 0-127.5V 255 means 127.5V
    int16_t   torque;     // Current rotor torque 256 means 1mNM
    int16_t   speed;      // Current rotor speed 256/2π means 100rad/s
    int32_t   pos;        // Current rotor position 32768/2π represents 1rad
    uint32_t  MError;     // Motor identification: 0x0. Normal bit0 overcurrent,...
    uint16_t  ExSensor1;   // Extended sensor data, such as output torque sensor
    uint16_t  ExSensor2;   // Extended sensor data, such as output torque sensor
    uint32_t  CRC32;      // CRC32
} MotorState_t;
```

The above structure explains some content of the motor return status and the CRC checksum. The CRC checksum will be given at the end.

© 2025 All rights reserved, Unitree Robotics

Sensor Status Structure

```
typedef struct
{
    uint8_t head[2];//{0x7E, 0xE7}; Participate in CRC check
    uint8_t id     : 4;     // ID: 0,1...,13,14 15 means broadcasting data to all sensors (no return at this time)
    uint8_t status : 4;     // Working mode: 0
    uint8_t reverse;        // Reserved bits
    uint16_t pressure[12];   // The sensor returns data of 12*2=24 bytes. The original pressure value is reduced by 8 times, that is, each 1 is 8Pa.
    int8_t temperature[12]; // The sensor returns 12*1=12 bytes of data, each ranging from -127°C to +128°C.
    uint32_t CRC32;         // CRC32 44 bytes
}SensorState_t;
```

This structure represents the returned sensor status data. For specific parameters, refer to the comments.

Communication Structure from Control End to MCU

```
typedef struct {
    uint8_t head[2];
    uint8_t mode[4];
    uint32_t cmd_size[SUPPORTED_FINGER_NUM];
    uint8_t cmd[SUPPORTED_FINGER_NUM][SUPPORTED_SEND_BUFFER_SIZE];
    uint32_t reserve;
    uint32_t crc;
} LinuxToMcuCmd_t;
```

The cmd_size array stores the total data size of each finger bus, and the cmd stores the corresponding data to be sent on the corresponding finger bus. The CRC calculation method here is:

```
cmd.crc = crc32((uint32_t *)&cmd.mode[0], sizeof(LinuxToMcuCmd_t) / 4 - 1);
```

The CRC check will be given at the end.

MCU Receiving Motor Status Structure

© 2025 All rights reserved, Unitree Robotics


```
// MCU to Linux status data structures
typedef struct {
    uint8_t head[2];//The frame header is not included in the CRC calculation 0xFC 0xFD
    uint8_t mode[2];//mode[1] is used to control the timeout protection mode[0] indicates the model of the hand
    uint8_t fans[2];
    uint32_t state_size[SUPPORTED_FINGER_NUM];
   uint8_t state[SUPPORTED_FINGER_NUM][SUPPORTED_RECV_BUFFER_SIZE];
     uint16_t pin_v; // Convert to voltage mcu_to_linux_.power_v / 65536.0 * 3.3 * 21.0
    uint16_t vin_v; // Convert to current mcu_to_linux_.power_a / 65536.0 * 3.3 * 10.0
    uint16_t vm_v; // Convert to voltage mcu_to_linux_.power_v / 65536.0 * 3.3 * 21.0
    uint16_t pin_a; //  Convert to current mcu_to_linux_.power_a / 65536.0 * 3.3 * 10.0
    int16_t imu_data[6]; // 0-5 are ax ay az gx gy gz respectively. Conversion method: raw_data = imu_data/100
    uint32_t error;
    uint32_t reserve[4];
    uint32_t crc;
} McuToLinuxState_t;
```

This structure is the dexterous hand status data transmitted from the internal MCU of the dexterous hand and is parsed using this structure.

CRC32 Function Calculation Method

```
1
2  // Generate CRC32 Lookup Table
3  constexpr std::array<uint32_t, 256> generate_crc32_table() {
4      constexpr uint32_t POLYNOMIAL = 0x04C11DB7;
5      std::array<uint32_t, 256> table{};
6      for (uint32_t i = 0; i < 256; i++) {
7          uint32_t c = i << 24;
8          for (int j = 0; j < 8; j++) {
9              c = (c & 0x80000000) ? ((c << 1) ^ POLYNOMIAL) : (c << 1);
10         }
11         table[i] = c;
12     }
13     return table;
14 }
15
16 constexpr auto CRC32_TABLE = generate_crc32_table();
17
18 uint32_t crc32(uint32_t* ptr, uint32_t len) {
19     uint32_t crc = 0xFFFFFFFF;
20     for (uint32_t i = 0; i < len; i++) {
21         uint32_t data = ptr[i];
22         for (int j = 0; j < 4; j++) {
23             crc = (crc << 8) ^ CRC32_TABLE[((crc >> 24) ^ (data >> (24 - j * 8))) & 0xFF];
24         }
25     }
```

© 2025 All rights reserved, Unitree Robotics

```
26     return crc;
27 }
```

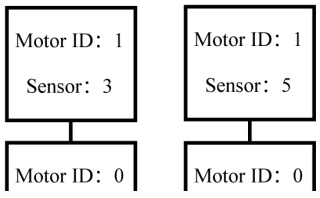
© 2025 All rights reserved, Unitree Robotics

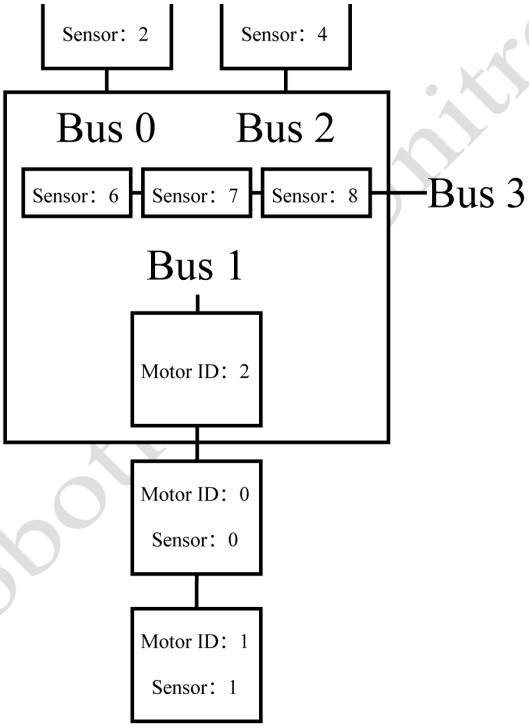
Communication Process

Data Sending

In the above protocol, the LinuxToMcuCmd_t structure protocol is used to pass in the corresponding MotorCmd_t data of each bus. The motor data on each bus is placed in the corresponding cmd two-dimensional array of the LinuxToMcuCmd_t structure in sequence. After the motor data is sent, the data returned by the USB is parsed, and the McuToLinuxState_t structure is used for the corresponding parsing.

The bus corresponding to the dexterous hand is:





This bus is the same for both the left and right hands. The Motors in the corresponding bus are numbered starting from 0. The overall control data is distinguished using the bus. In high-level application development, the ID number of the motor can be defined according to one's own habits.

Data Receiving

Data receiving is carried out using the USB protocol. The data can be parsed using the McuToLinuxState_t structure, and the corresponding CRC can be calculated using the CRC32 function calculation method for CRC check. The USB communication class explains how to use receiveState to receive data.

© 2025 All rights reserved, Unitree Robotics

DDS-based Communication Method

Agile hand control method

Inside G1, a program that communicates with Dex3-1 and converts it into DDS messages will be provided as a resident service. This service will receive and process user requests. After obtaining unitree_sdk2, the system will load the message structure library. Users can control the corresponding dexterous hand by correctly using the message structure and communicating with the corresponding interface. The dexterous hand header files are located at:

```
/usr/local/include/unitree/hg_idl/HandState_.hpp
/usr/local/include/unitree/hg_idl/HandCmd_.hpp
```

Users can view the structure information in these files.

Message Structure Sorting

According to the message structure sorting, the thumb, index finger, and middle finger are defined in the human perception situation and sorted in order. The two message structures **unitree_hg::msg::dds_::HandCmd_.motor_cmd** and **unitree_hg::msg::dds_::HandState_.motor_state** contain all the information of the dexterous hand motors. The motor order is as follows:

Hand Joint Index in IDL	Hand Joint Name
0	thumb_0
1	thumb_1
2	thumb_2
3	middle_0
4	middle_1
5	index_0
6	index_1

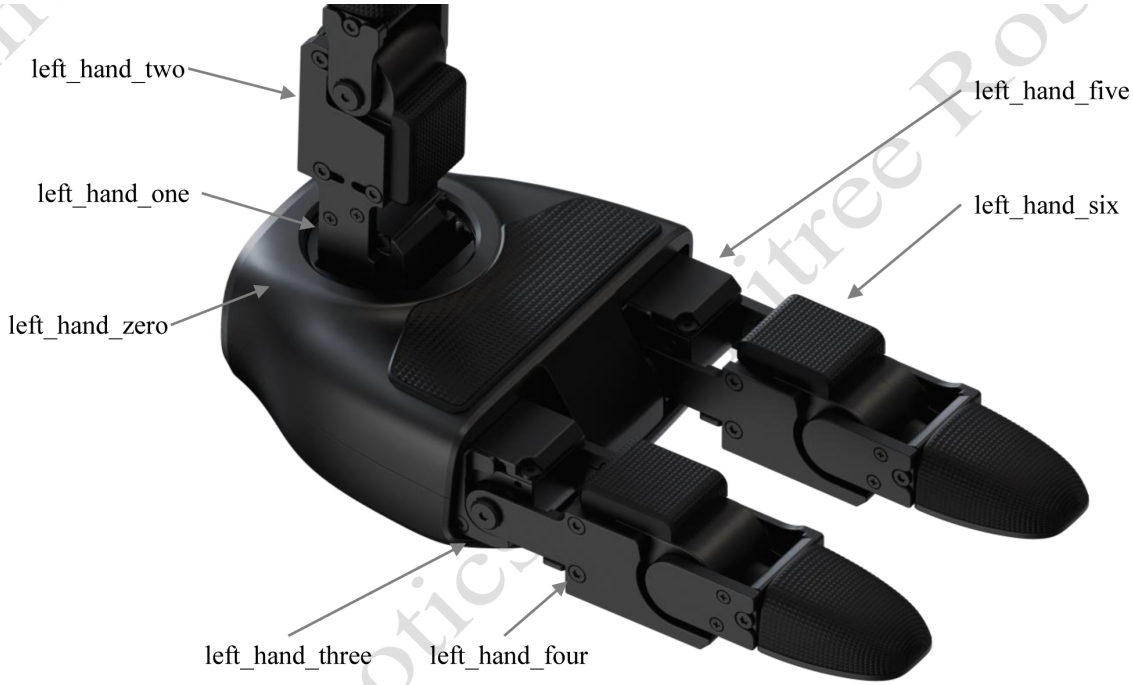
URDF Sorting

Since the URDF version of the dexterous hand generates position information in the same coordinate system, there are slight differences between the left and right hands. The URDF can be obtained from the open source platform:

© 2025 All rights reserved, Unitree Robotics

Left hand:

https://github.com/unitreerobotics/unitree_ros/blob/master/robots/g1_description/dex3_1_l.urdf

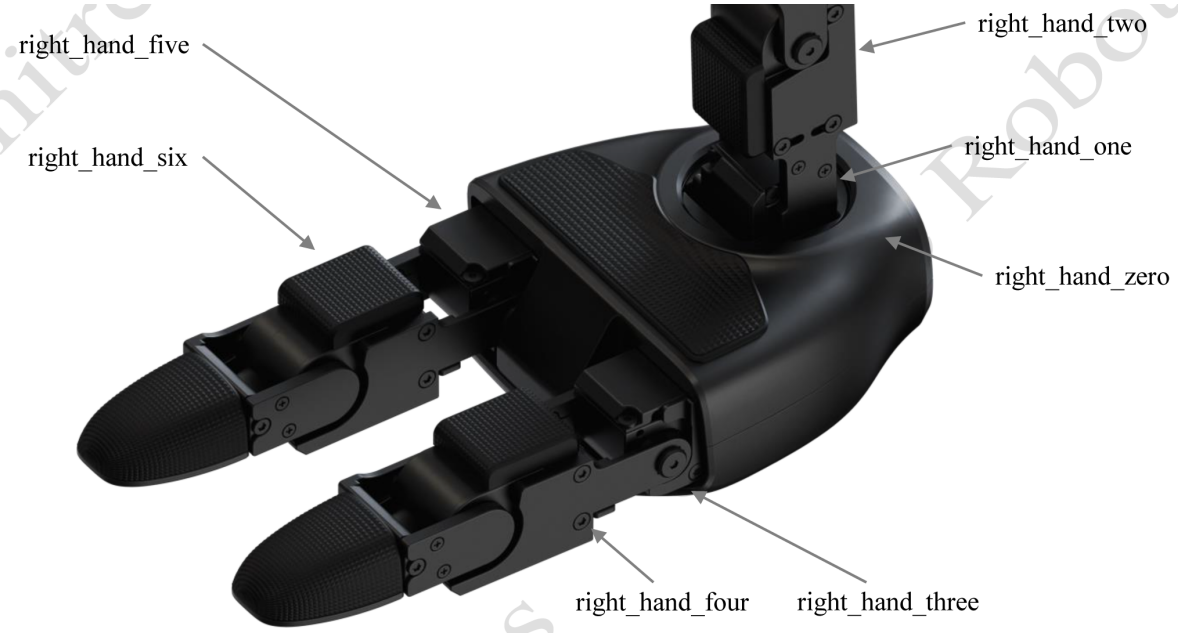


Hand Joint Index in URDF	Hand Joint Index in IDL	Hand Joint Name
left_hand_zero	0	thumb_0
left_hand_one	1	thumb_1
left_hand_two	2	thumb_2
left_hand_five	3	middle_0
left_hand_six	4	middle_1
left_hand_three	5	index_0
left_hand_four	6	index_1

© 2025 All rights reserved, Unitree Robotics

Right hand:

https://github.com/unitreerobotics/unitree_ros/blob/master/robots/g1_description/dex3_1_r.urdf



Hand Joint Index in URDF	Hand Joint Index in IDL	Hand Joint Name
right_hand_zero	0	thumb_0
right_hand_one	1	thumb_1
right_hand_two	2	thumb_2
right_hand_three	3	middle_0
right_hand_four	4	middle_1
right_hand_five	5	index_0
right_hand_six	6	index_1

© 2025 All rights reserved, Unitree Robotics

Interface Description

Users send **unitree_hg::msg::dds_::HandCmd_ messages** to the "**rt/dex3/(left or right)/cmd**" topic to control the dexterous hand. Receive **unitree_hg::msg::dds_::HandState_ messages** from the "**rt/dex3/(left or right)/state**" topic to obtain the dexterous hand status.

IDL Data Format

The following IDL represents the control structure and state structure of this communication

Control Structure HandCmd_:

```
struct HandCmd_ {
    sequence<unitree_hg::msg::dds_::MotorCmd_> motor_cmd;
    unsigned long reserve[4];
};//struct HandCmd_
```

MotorCmd_ structure is:

```
typedef struct { // Motor control command data packet 20 bytes
    uint8_t head[2]; // Packet header 0xFE 0xEE participates in CRC
    RIS_Mode_t mode; // Motor control mode 1Byte
    int16_t tor_des; // Desired joint output torque 256 represents 1mNM
    int16_t spd_des; // Desired joint output speed 256/2π represents 100rad/s
    int32_t pos_des; // Desired joint output position 32768/2π represents 1rad
    int16_t k_posi; // Desired joint stiffness coefficient 1280 represents 1mN.m/rad
    int16_t k_spd; // Desired joint damping coefficient 1280 represents 1mN.m/100rad/s
    uint32_t CRC32; // CRC32
    uint8_t res; // Reserved
} MotorCmd_t;
```

RIS_Mode_t is:

```
typedef struct {
    uint8_t id : 4; // Motor ID: 0, 1..., 13, 14. 15 means broadcasting data to all motors
    uint8_t status : 3; // Working mode: 0. Locked 1. FOC 6
    uint8_t timeout: 1; // Master->Motor: 0. Disable timeout protection 1. Enable (default 1s timeout)
    Motor->Master: 0. No timeout 1. Trigger timeout protection (need to send 0 to the control bit to clear)
} RIS_Mode_t;
```

Motor and Hand State Instruction HandState_

```
struct HandState_ {
     sequence<unitree_hg::msg::dds_::MotorState_> motor_state;
     sequence<unitree_hg::msg::dds_::PressSensorState_> press_sensor_state;
     unitree_hg::msg::dds_::IMUState_ imu_state;
     float power_v;
     float power_a;
     float system_v;
     float device_v;
     unsigned long error[2];
```

© 2025 All rights reserved, Unitree Robotics

```
     unsigned long reserve[2];
};
```

In HandState_, IMUState_ can be understood in the [H1-2 Low-level Service Interface Chapter] in the

H1 SDK development guide for the data filling rules of structures such as MotorState_. The PressSensorState_ structure will be explained below.

power_v represents the total power supply voltage of the dexterous hand.

power_a represents the total power supply current of the dexterous hand. system_v represents the internal power supply voltage of the dexterous hand.

device_v represents the output voltage of the buck module in the dexterous hand.

The error field represents the error code output by the current dexterous hand.

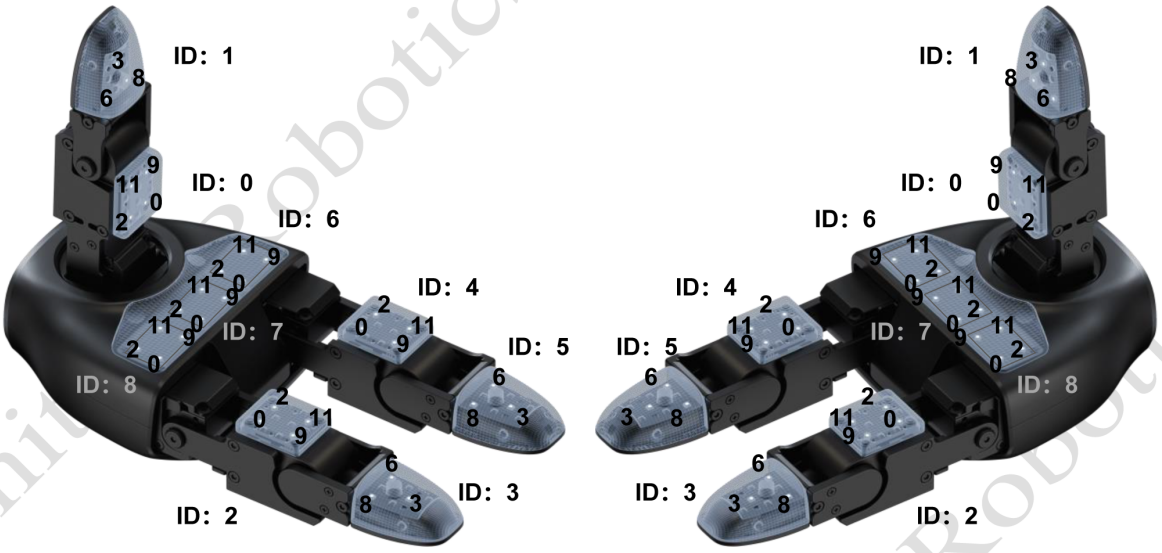
For the specific expression of the error code, please refer to [Motor Status Information Error] in the G1 SDK documentation (Unitree Technology Documentation Center).

The reserve field is reserved for the time being.

Hand Sensor Structure PressSensorState:

```
struct PressSensorState {
    float pressure[12];
    float temperature[12];
    unsigned long lost;
    unsigned long reserve;
};
```

For pressure, it is a single sensor.



Left Hand Numerical Correspondence Right Hand Numerical Correspondence

The ID represents a sensor data packet PressSensorState_. The sensor position corresponding to the pressure[12] data index in the packet has been marked in the figure.

© 2025 All rights reserved, Unitree Robotics

Sensor data with a value greater than or equal to 100000 is a valid value for transmission. A value of

30000 represents no value. Here, it is recommended to reduce 100000 to 10.0000 for display and calculation.

temperature represents the temperature value of the sensor.

© 2025 All rights reserved, Unitree Robotics

Unitree Support

https://www.unitree.com

If there is any update to the content, no separate notice will be given.

You can obtain more documents on the official Unitree website.

https://www.unitree.com/download

If you have any questions or suggestions about the manual, please contact us at the following

E-mail address: support@unitree.cc

© 2025 All rights reserved, Unitree Robotic

Scan the QR code with WeChat and follow the Unitree official account

Unitree Robotics