# Robomaster Doc Documentation

dji

Oct 09, 2020

# Get started


ROBOMASTER EP

CHAPTER 1

# Get Started Quickly with RoboMaster SDK

## 1.1 Introduction

As an education robot, RoboMaster EP features strong scalability and programmability. In terms of programmability, it provides Scratch programming, Python programming, and SDK to facilitate users in conducting secondary development on RoboMaster EP and expanding more functions.

Next, we will complete the **Control the blaster emission** function as an example and use the **Wi-Fi direct connection** mode (for other connection modes, please refer to *Connection*) to introduce the use of the plain-text protocol in the SDK.

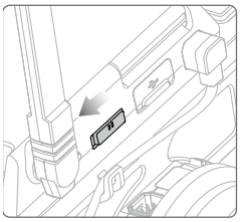
## 1.2 Pre-development preparation

1. Prepare a PC with Wi-Fi.
2. Establish a Python 3.x environment on the PC. For installation methods, please refer to Python Getting Started

## 1.3 Establish connection

1. Power on

   Power on the robot and toggle the connection mode switch of the smart console to **direct connection mode**

2. Establish a Wi-Fi connection

   Open the computer's wireless network access list, select the corresponding Wi-Fi name that is displayed on the robot's sticker, enter the 8-digit password, and select Connect

3. Prepare a connection script

   After completing the Wi-Fi connection, we also need to program a TPC/IP connection with the robot, and transmit the specific **plain-text protocol** on the corresponding port, so as to implement corresponding control. For more information on **plain-text protocol**, please refer to *Protocol Content*.

   Here we take Python programming language as an example and compose a script to complete the process of *establishing control connection, receiving instructions from the user, and transmitting plain-text protocol* for the purpose of controlling the robot.

   The reference code is as follows

```python
# Test environment: Python version 3.6

import socket
import sys

# In direct connection mode, the robot's default IP address is 192.168.2.1, and the control command port number is 40923
host = "192.168.2.1"
port = 40923

def main():

        address = (host, int(port))

        # Establish a TCP connection with the robot's control command port
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:

                # Wait for the user to input a control command
                msg = input(">>> please input SDK cmd: ")

                # Exit the current program when the user enters Q or q
                if msg.upper() == 'Q':
                        break

                # Add a ';' terminator to the end
                msg += ';'

                # Transmit the control command to the robot
                s.send(msg.encode('utf-8'))

                try:
                        # Wait for the robot to return the execution result
                        buf = s.recv(1024)

```

(continues on next page)

(continued from previous page)

```
                        print(buf.decode('utf-8'))
                except socket.error as e:
                        print("Error receiving :", e)
                        sys.exit(1)
                if not len(buf):
                        break

        # Disable the port connection
        s.shutdown(socket.SHUT_WR)
        s.close()

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

4. Save the above code as rm_sdk.py
5. Run the script

   Run the rm_sdk.py file (For a Windows system, you can double-click the *.py file to run it directly after the Python environment is installed. If it can't run, press `Win+R` and enter `cmd`. Press Enter to open the command line, and type `python rm_sdk.py` to run it; for Linux system, press `Ctrl+Alt+T` to open the command line and type `python rm_sdk.py` to run it)

6. Establish a TCP/IP control connection

   When the Run window displays `Connecting...`, it is trying to establish a connection with the robot. When the Run window displays `Connected!`, it means that the control connection has been successfully established.

## 1.4 Enable SDK mode

To implement SDK control, we need to control the robot to enter SDK mode. Enter *command* in the Python Run window above, press Enter, and then the program will send the command to the robot. If it returns *OK*, it means the robot has successfully entered SDK mode:

```
>>> please input SDK cmd: command
ok
```

After entering SDK mode, we can input control commands to control the robot.

## 1.5 Transmit control commands

Proceed to input *blaster fire*, and it should return *OK*. At the same time, the blaster fires once:

```
>>> please input SDK cmd: blaster fire
ok
```

Then, you can input other control commands to control the robot. For more control commands, please refer to *Protocol*

## 1.6 Exit SDK mode

After completing all our control commands, we need to exit from SDK mode so that other functions of our robot can be used normally.

Enter *quit* to exit SDK mode. After exiting SDK mode, you cannot continue to use the SDK functions. To use them, please re-enter *command* to enter SDK mode:

```
>>> please input SDK cmd: quit
quit sdk mode successfully
```

## 1.7 Summary

In the foregoing, we implemented relevant robot control functions via SDK through several steps, including establishing a physical connection and then TCP/IP control connection with the robot, controlling the robot to enter SDK mode, transmitting control commands, and exiting SDK mode. You can implement more complex logic and more interesting functions by adding to the content of the *Transmitting control commands* section.

In the Python programming control section, if you are more familiar with other languages, you can also use other languages to complete the whole control process.

If your device doesn't support Wi-Fi and can't use **Wi-Fi direct connection**, please refer to *Connection* to use other connection modes.

This concludes how to get started with SDK. For more details, see *SDK documentation*.

CHAPTER 2

# Connection

## 2.1 Connection Modes

The robot supports multiple connection modes, and can access and use SDK functions through any connection mode

- **Direct connection**:
  1. *Wi-Fi direct connection*: Access SDK functions by setting the robot to direct connection mode and connecting the robot's Wi-Fi hotspot
  2. *USB connection*: Access SDK functions through the USB port on the robot's smart console (RNDIS function support required)
  3. *UART connection*: Access SDK functions through the UART interface on the robot's motion controller
- **Networking connection**:
  1. *Wi-Fi networking*: The network connection is achieved by setting the robot to networking mode and adding the computing device and the robot onto the same LAN

## 2.2 Connection Parameters

1. For Wi-Fi direct connection, Wi-Fi networking, and USB connection, please refer to the following parameter configuration:

- **IP address description**:
  - In Wi-Fi direct connection mode, the robot's default IP is 192.168.2.1
  - In Wi-Fi networking mode, the robot's IP is dynamically assigned by the router, and connection is made by listening to the *IP broadcast* data port to obtain the robot's IP address in the current LAN
  - In USB connection mode, the computing device needs to support the RNDIS function. The robot's default IP is 192.168.42.2
- **Port and connection mode description**:

| Data | Port No. | Connection mode | Description |
|---|---|---|---|
| Video streaming | 40921 | TCP | To output data, you need to execute the command to enable video streaming push |
| Audio streaming | 40922 | TCP | To output data, you need to execute the command to enable audio streaming push |
| Control command | 40923 | TCP | SDK mode can be enabled through the current channel. See **SDK Mode Control** |
| Message push | 40924 | UDP | To output data, you need to execute the command to enable message push |
| Event reporting | 40925 | TCP | To output data, you need to execute the command to enable event reporting |
| IP broadcasting | 40926 | UDP | Data will be output when the robot is not connected to any device |

2. Please refer to the following UART parameter configuration for the UART connection mode

| Baud rate | Data bits | Stop bits | Parity bits |
|---|---|---|---|
| 115200 | 8 | 1 | None |

**Warning:** Description of the data under the UART connection mode:

Under the UART connection mode, only *control command, message push, and event reporting* data is provided. If *video streaming or audio streaming* data is required, please use the *Wi-Fi/USB* connection mode

## 2.3 Connection Examples

Next, we will introduce examples of using various connection modes based on the Python programming language. In all of the following examples, a Python 3.x environment is required to be integrated on the default PC (please refer to 'Python Getting Started <https://www.python.org/about/gettingstarted/>' for the installation method). No further details on this will be provided here.

### 2.3.1 Wi-Fi direct connection

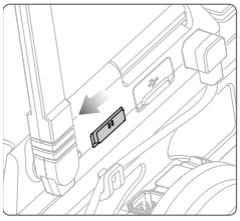
- **Environmental preparation**

1. Prepare a PC with Wi-Fi.

- **Establish connection**

1. Power on

   Power on the robot and toggle the connection mode switch of the smart console to **direct connection mode**



2. Establish a Wi-Fi connection

   Open the computer's wireless network access list, select the corresponding Wi-Fi name that is displayed on the robot's sticker, enter the 8-digit password, and select Connect

3. Prepare a connection script

   After establishing the Wi-Fi connection, we also need to program a TPC/IP connection. The robot offers multiple connection ports for connection. First, we should connect via the **control command port** (in direct connection mode, the IP address of the robot is `192.168.2.1`, and the control command port number is `40923`), so as to enable the robot's SDK mode.

   Here we compose a script to *establish control connection and enable SDK mode* by using the Python programming language as an example

   The reference code is as follows

```python
# Test environment: Python version 3.6

import socket
import sys

# In direct connection mode, the default IP address of the robot is 192.168.2.1, and the control command port number is 40923
host = "192.168.2.1"
port = 40923

def main():

        address = (host, int(port))

        # Establish a TCP connection with the robot's control command port
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:

                # Wait for the user to input a control command
                msg = input(">>> please input SDK cmd: ")

                # Exit the current program when the user enters Q or q
                if msg.upper() == 'Q':
                        break

                # Add a ';' terminator to the end
                msg += ';'

                # Transmit the control command to the robot
                s.send(msg.encode('utf-8'))

                try:
                        # Wait for the robot to return the execution result
                        buf = s.recv(1024)
```

(continues on next page)

(continued from previous page)

```
                print(buf.decode('utf-8'))
            except socket.error as e:
                print("Error receiving :", e)
                sys.exit(1)
            if not len(buf):
                break

    # Disable the port connection
    s.shutdown(socket.SHUT_WR)
    s.close()

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

4. Save the above code as rm_direct_connection_sdk.py
5. Run the script

   **Windows system** After installing the Python environment, you can double-click the *.py file to run it. If it does not run, press `win+r` and enter `cmd`. Press Enter to open and run the command, and then type and run `python rm_direct_connection_sdk.py`;

   **Linux system** Please press `Ctrl+Alt+T` to open the command line, and type and run `python rm_direct_connection_sdk.py`
6. Establish a TCP/IP control connection

   When the run window displays `Connecting...`, it is trying to establish a connection with the robot. When the run window displays `Connected!`, it indicates that the control connection has been successfully established.

- **Validation**

After a successful control connection is established, enter `command` in the command line. If the robot returns `OK`, the connection has been completed and the robot has successfully entered SDK mode. Then you can enter any control command to control the robot.

### 2.3.2 Wi-Fi/Wired network connection

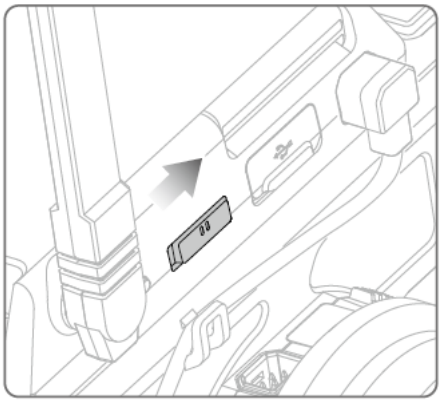
- **Environmental preparation**

1. Prepare a PC with a network function (either Wi-Fi or wired network is accepted)
2. Prepare a home router

- **Establish connection**

1. Power on

   Power on the robot and toggle the connection mode switch of the smart console to **networking mode**



2. Establish a network connection

   Wi-Fi

   If you use Wi-Fi connection, connect your PC to the router via Wi-Fi

   Wired network:

   If you use a wired network connection, connect your PC to the LAN port of the router via a network cable

   After your PC is connected to the router, open the RoboMaster program, go to the Networking Connection page, and press the Scan Code to Connect button on the robot's smart console to scan the QR code to connect to the network.


3. Obtain the IP address of the robot in the LAN

   After completing the networking connection, the PC should be in the same LAN as the robot. Next, we need to program a TPC/IP connection with the robot and connect to the **control command port** to enable SDK mode.

   If you are using a router with DHCP service enabled, the IP address of the robot is dynamically assigned by the router. You need to further obtain the IP address of the robot in the LAN. There are two ways to obtain the IP address:

   1. If you have connected through the RoboMaster program, go to the *Settings - > Connection* page of the RoboMaster program. The IP address of the robot in the LAN is displayed here.
   2. If you have established network connection via other means, you need to obtain the robot's IP address in the LAN by *listening to the address broadcast of the robot*. For more details, please refer to the **Broadcast** section.

   The reference code is as follows

```
import socket

ip_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the IP broadcast port
ip_sock.bind(('0.0.0.0', 40926))

# Wait to receive data
ip_str = ip_sock.recvfrom(1024)

# Output data
print(ip_str)
```

Save the above code as rm_get_robot_ip.py, and run it. The command line shall output:

robot ip 192.168.0.115

We can see that the IP address of the robot in the LAN, as obtained by *listening to the address broadcast of the robot*, is `192.168.0.115`

3. Prepare a connection script

Now we have obtained the robot's IP address, we shall compose a script to *establish control connection and enable SDK mode* by using the Python programming language as an example

The reference code is as follows

```
# Test environment: Python version 3.6

import socket
import sys

# In networking mode, the current IP address of the robot is 192.168.0.
↪115, and the control command port number is 40923
# The robot's IP address is modified according to the actual IP address
host = "192.168.0.115"
port = 40923

def main():

        address = (host, int(port))

        # Establish a TCP connection with the robot's control command port
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:

                # Wait for the user to input a control command
                msg = input(">>> please input SDK cmd: ")

                # Exit the current program when the user enters Q or q
                if msg.upper() == 'Q':
                        break
```

(continues on next page)

(continued from previous page)

```
                # Add a ';' terminator to the end
                msg += ';'

                # Transmit the control command to the robot
                s.send(msg.encode('utf-8'))

                try:
                        # Wait for the robot to return the execution result
                        buf = s.recv(1024)

                        print(buf.decode('utf-8'))
                except socket.error as e:
                        print("Error receiving :", e)
                        sys.exit(1)
                if not len(buf):
                        break

        # Disable the port connection
        s.shutdown(socket.SHUT_WR)
        s.close()

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

4. Save the above code as rm_networking_connection_sdk.py
5. Run the script

   **Windows system** After installing the Python environment, you can double-click the *.py file to run it. If it does not run, press `win+r` and enter `cmd`. Press Enter to open and run the command, and then type and run `python rm_networking_connection_sdk.py`;

   **Linux system** Please press `Ctrl+Alt+T` to open the command line, and type and run `python rm_networking_connection_sdk.py`

6. Establish a TCP/IP control connection

   When the run window displays `Connecting...`, it is trying to establish a connection with the robot. When the run window displays `Connected!`, it indicates that the control connection has been successfully established.

- **Validation**

After a successful control connection is established, enter `command` in the command line. If the robot returns `OK`, the connection has been completed and the robot has successfully entered SDK mode. Then you can enter any control command to control the robot.

### 2.3.3 USB Connection

USB connection mode essentially uses the RNDIS protocol to virtualize the USB device on the robot as a network card device, and initiate a TCP/IP connection via USB. For more information about RNDIS, see XXXXX

- **Environmental preparation**

1. Prepare a PC with the RNDIS function (please check that the RNDIS function is configured on the PC)
2. Prepare a micro-USB cable

- **Establish connection**

1. Power on

   Power on the robot. The position of the connection mode switch is not important

2. Establish a USB connection

   Connect the USB cable to the USB port on the smart console of the robot, and connect the other end of the cable to the computer

3. Test the connection

   Open a command line window and run:

```
ping 192.168.42.2
```

   If the command line outputs **Reply from 192.168.42.2...**, it indicates that the link works. You can proceed to the next step, such as:

```
PING 192.168.42.2 (192.168.42.2) 56(84) bytes of data.
64 bytes from 192.168.42.2: icmp_seq=1 ttl=64 time=0.618 ms
64 bytes from 192.168.42.2: icmp_seq=2 ttl=64 time=1.21 ms
64 bytes from 192.168.42.2: icmp_seq=3 ttl=64 time=1.09 ms
64 bytes from 192.168.42.2: icmp_seq=4 ttl=64 time=0.348 ms
64 bytes from 192.168.42.2: icmp_seq=5 ttl=64 time=0.342 ms

--- 192.168.42.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4037ms
rtt min/avg/max/mdev = 0.342/0.723/1.216/0.368 ms
```

   If the command line outputs ** Cannot access... ** or the display times out, you need to check whether the RNDIS service on the PC is configured properly and restart the robot to try again, such as:

```
PING 192.168.42.2 (192.168.42.2) 56(84) bytes of data.

--- 192.168.42.2 ping statistics ---
```

4. Prepare for connection

   The connection process is similar to *Wi-Fi Direct Connection* - > **Prepare a Connection Script**. You need to replace the robot's IP address with the IP address in USB mode, and the rest of the codes and steps remain unchanged, which will not be repeated here

   The reference code is changed as follows

```
# Test environment: Python version 3.6

import socket
import sys

# In USB mode, the robot's default IP address is 192.168.42.2, and the control command port number is 40923
host = "192.168.42.2"
port = 40923

# other code
```

- **Validation**

After a successful control connection is established, enter `command` in the command line. If the robot returns `OK`, the connection has been completed and the robot has successfully entered SDK mode. Then you can enter any control command to control the robot.

### 2.3.4 UART Connection

- **Environmental preparation**

1. Prepare a PC and confirm that the USB to serial port module driver is installed
2. Prepare a USB to serial port module
3. Prepare three DuPont cables

- **Establish connection**

1. Power on

   Power on the robot. The position of the connection mode switch is not important

2. Connecting the UART

   Plug the DuPont cables into the UART interface on the main controller of the robot chassis, that is into the GND, RX, and TX pins, respectively, and the other ends into the corresponding GND, TX, and RX pins of the USB serial port module

3. Configure the UART and establish a communication connection

   Here, we still use Python programming as an example to configure the UART for a Windows system.

   1. Confirm that the PC has recognized the USB to serial port module, and confirm the corresponding serial port number from the **Port** in the **Computer Device Manager**, such as COM3.
   2. Install the serial module:

      pip install pyserial

   3. Write the code for UART control. The reference code is as follows

```
# Test environment: Python version 3.6
import serial

ser = serial.Serial()

# Configure the serial port: baud rate: 115200; data bits: 8; stop bits: 1; parity bits: 0; timeout: 0.2s
ser.port = 'COM3'
ser.baudrate = 115200
ser.bytesize = serial.EIGHTBITS
ser.stopbits = serial.STOPBITS_ONE
ser.parity = serial.PARITY_NONE
ser.timeout = 0.2

# Open the serial port
ser.open()

while True:

        # Wait for the user to input a control command
        msg = input(">>> please input SDK cmd: ")

```

(continues on next page)

(continued from previous page)

```
        # Exit the current program when the user enters Q or q
        if msg.upper() == 'Q':
                break

        # Add a ';' terminator to the end
        msg += ';'

        ser.write(msg.encode('utf-8'))

        recv = ser.readall()

        print(recv.decode('utf-8'))

# Close the serial port
ser.close()
```

4. Save the above program as rm_uart.py and run it

- **Validation**

After a successful control connection is established, enter `command` in the command line. If the robot returns `OK`, the connection has been completed and the robot has successfully entered SDK mode. Then you can enter any control command to control the robot.

CHAPTER 3

# Protocol

## 3.1 Protocol Format

### 3.1.1 Control command

IN**<obj> <command> <params> [<seq>]**

- **Description**
  - Control command protocol format. It is generally used to interact with the robot in control
- **Parameters**
  - *obj* (str): Control object string
  - *command* (str): Control command string
  - *params* (str): Command parameter string, generally in the form of **<key> <value>**
  - *seq* (str): Command sequence number string, usually in the form of **seq <seq_value>**. An optional parameter

OUT**<result> [<seq>]**

- **Description**
  - The protocol format of the control command response result. It is generally used to confirm the execution result of the control command
  - Unless specifically instructed, all control commands have response results
- **Parameters**
  - *result* (exec_result_enum): Execution result string
  - *seq* (str): Execution result sequence number string, usually in the form of **seq <seq_value>**

---

**Note:** <seq>

<seq> can be used to identify the uniqueness of the current message. When the control command has the *<seq>* parameter, the response result of the corresponding command contains the corresponding sequence number

### 3.1.2 Message push

OUT: **<obj> push <attr> <value>**

- **Description**
  - Message push protocol format. Messages can be received after the message push is enabled via a control command
  - The message push will push messages at a fixed frequency, which depends on the frequency setting while enabling the current message push
- **Parameters**
  - *obj* (str): Push object
  - *attr* (str): Push data properties
  - *value* (str): Push data value

### 3.1.3 Event reporting

OUT: **<obj> event <attr> <value>**

- **Description**
  - Event reporting protocol format. Reports can be received after an event reporting is switched on through a control command
- **Parameters**
  - *obj* (str): The object of the event
  - *attr* (str): Event data properties
  - *value (str)* Event data value

**Note:** Trigger mechanism

When the *event reporting* function is enabled successfully, an event will be reported if one occurs

### 3.1.4 IP broadcasting

OUT: **robot ip <addr>**

- **Parameters**
  - *addr* (str): The IPv4 address of the robot in the current connection mode

**Note:** Broadcast life cycle

While in *Wi-Fi networking* mode, the robot will continuously broadcast its IPv4 address to corresponding ports. You can connect to the robot through this IP address. When the connection is successful, the broadcast will stop

### 3.1.5 Video streaming

OUT: H.264 encoded real-time video streaming data. Decoding the video streaming data correctly is required in order to display video images in real time.

### 3.1.6 Audio streaming

OUT: Opus encoded real-time audio streaming data. Decoding the audio streaming data correctly is required in order to play the audio in real time.

---

**Note:** IN/OUT

In this document, the prefix **IN** or **OUT** has no practical significance in control commands. It is only to identify the direction of data flow of the current command when the robot is the main body:

INIdentifies that the current data is sent from an external device to the robot

OUTIdentifies that the current data is sent from the robot to an external device

In actual practice, please ignore the IN and OUT identifiers. Sending and receiving the actual control commands is enough

---

## 3.2 Protocol Content

### 3.2.1 SDK mode control

#### Enter SDK mode

IN**command**

- **Description**
  - Control the robot to enter SDK mode
  - Only after the robot successfully enters SDK mode can it respond to other control commands

#### Exit SDK mode

IN: **quit**

**-Description**

- Control the robot to exit SDK mode and reset all settings
- In Wi-Fi/USB connection mode, when the connection is broken, the robot exits SDK mode automatically

### 3.2.2 Robot control

#### Robot motion mode control

IN**robot mode <mode>**

- **Description**

  - Set the robot motion mode

- **Parameters**
  - *mode* (*mode_enum*): Robot motion mode
- **Example**
  - *robot mode chassis_lead* : Set the robot motion mode to "Chassis lead mode"

---

**Note:** Robot motion mode

The robot motion mode describes the interaction and interplay between the platform and the chassis, and each robot mode corresponds to a specific relationship.

There are three modes of robot motion:

1. Chassis lead mode: In this mode, the yaw axis of the gimbal goes into a state in which it continuously follows the movement of the yaw axis of the chassis. The gimbal does not respond to the yaw axis control parts in any control commands. The affected commands include *gimbal motion speed control*, *gimbal relative position control*, and *gimbal absolute position control*
2. Gimbal lead mode: In this mode, the yaw axis of the chassis goes into a state in which it continuously follows the movement of the yaw axis of the gimbal. The chassis does not respond to the yaw axis control parts in any control commands. The affected commands include *chassis motion speed control*, *chassis wheel speed control*, and *chassis relative position control*
3. Free mode: In this mode, the yaw axis of the gimbal and the yaw axis of the chassis do not affect each other's movement.

---

#### Obtaining the robot's motion mode

IN: **robot mode ?**

- **Description**
  - Query the robot's current motion mode
- **Return value**
  - *mode* (*mode_enum*): Robot motion mode
- **Example**
  - IN*robot mode ?*: Query the robot's current motion mode
  - OUT: *chassis_lead*: The current motion mode returned by the robot is *chassis lead mode*

**Warning:** Obtain the *?* from commands

Note: There is a space between *?* in the query command and the foregoing command section

### 3.2.3 Chassis control

#### Chassis motion speed control

IN: **chassis speed x <speed_x> y <speed_y> z <speed_z>**

- **Description**
  - Control the chassis motion speed
- **Parameters**
  - *speed_x* (float:[-3.5,3.5]):x-axial velocity in m/s
  - *speed_y* (float:[-3.5,3.5]):y-axial velocity in m/s
  - *speed_z* (float:[-600,600]): z-axial rotation velocity in °/s
- **Example**
  - *chassis speed x 0.1 y 0.1 z 1* : The chassis's x-axial velocity is 0.1 m/s, the y-axial velocity is 0.1 m/s, and the z-axial velocity is 1°/s

### Chassis wheel speed control

IN: **chassis wheel w1 <speed_w1> w2 <speed_w2> w3 <speed_w3> w4 <speed_w4>**

- **Description**
  - Control the speed of the four wheels
- **Parameters**
  - *speed_w1* (int:[-1000, 1000]): Right front Mecanum wheel speed in rpm
  - *speed_w2* (int:[-1000, 1000]): Left front Mecanum wheel speed in rpm
  - *speed_w3* (int:[-1000, 1000]): Right rear Mecanum wheel speed in rpm
  - *speed_w4* (int:[-1000, 1000]): Left rear Mecanum wheel speed in rpm
- **Example**
  - *chassis wheel w2 100 w1 12 w3 20 w4 11* : The speed of the left front Mecanum wheel of the chassis is 100 rpm, the speed of the right front Mecanum wheel is 12 rpm, the speed of the right rear Mecanum wheel is 20 rpm, and the speed of the left rear Mecanum wheel is 11 rpm

### Chassis relative position control

IN: **chassis move { [x <distance_x>] | [y <distance_y>] | [z <degree_z>] } [vxy <speed_xy>] [vz <speed_z>]**

- **Description**
  - Control the chassis to move to a specified position. The origin of the coordinate axis is the current position
- **Parameters**
  - *distance_x* (int:[-5, 5]): x-axial distance in m
  - *distance_y* (int:[-5, 5]): y-axial distance in m
  - *degree_z* (int:[-1800, 1800]):z-axial distance in °
  - *speed_xy* (int:(0, 3.5]): xy-axial distance in m/s
  - *speed_z* (int:(0, 600]): z-axial distance in m/s
- **Example**
  - *chassiss move x 0.1 y 0.2* Using the current position as the origin of coordinates, move 0.1 m towards the x axis and 0.2 m towards the y axis

### Obtaining the chassis speed

IN: **chassis speed ?**

- **Description**
  - Obtain the chassis speed information
- **Return value**
  - *<x> <y> <z> <w1> <w2> <w3> <w4>* x axial velocity (m/s), y axial velocity (m/s), z axial rotation velocity (°/s), w1 right front Mecanum wheel speed (rpm), w2 left front Mecanum wheel speed (rpm), w3 right rear Mecanum wheel speed (rpm), w4 left rear Mecanum wheel speed (rpm)
- **Example**
  - IN: *chassis speed ?* : Obtain the motion speed information of the chassis
  - OUT: *1 2 30 100 150 200 250* : The current x-axial velocity of the chassis is 1 m/s, y-axial velocity is 2 m/s, z-axial rotation velocity is 20°/s, the speed of wheel 1 is 100 rpm, the speed of wheel 2 is 100 rpm, the speed of wheel 3 is 100 rpm, and the speed of wheel 4 is 100 rpm

### Obtaining the chassis position

IN: **chassis position ?**

- **Description**
  - Obtain the chassis position information
- **Return value**
  - *<x> <y> <z>* x-axis position (m), y-axis position (m), yaw angle (°)
- **Example**
  - IN: *chassis position ?* Obtain the chassis position information
  - OUT: *1 1.5 20* The current position of the chassis is 1 m along the x-axis, 1.5 m along the y-axis, and 20° from the position at the time of powering up

### Obtaining the chassis attitude

IN: **chasssis attitude ?**

- **Description**
  - Obtain chassis attitude information
- **Return value**
  - *<pitch> <roll> <yaw>* pitch axis angle (°), roll axis angle (°), yaw axis angle (°)
- **Example**
  - *chassis attitude ?* Query chassis attitude information

### Obtaining the chassis state

IN: **chassis status ?**

- **Description**
  - Obtain chassis state information
- **Return value**
  - ***<static> <uphill> <downhill> <on_slope> <pick_up> <slip> <impact_x> <impact_y> <impact_z> <roll_over> <hill_***
    - *static*Whether it is still
    - *uphill*Whether it is moving uphill
    - *downhill*Whether it is moving downhill
    - *on_slope*Whether it is on a slope
    - *pick_up*Whether it is picked up
    - *slip*Whether it is gliding
    - *impact_x*Whether the x-axis senses impact
    - *impact_y*Whether the y-axis senses impact
    - *impact_z*Whether the z-axis senses impact
    - *roll_over*Whether it has rolled over
    - *hill_static*Whether is standing still on a slope
- **Example**
  - IN: *chassis status ?* Query the status of the chassis
  - OUT: *0 1 0 0 0 0 0 0 0 0 0* : Chassis is currently in moving uphill

### Chassis information push control

IN**chassis push {[position <switch> pfreq <freq>][attitude <switch> afreq <freq>] | [status <switch> sfreq <switch>] [afreq <freq_all>]}**

- **Description**
  - Enable/disable the information push of corresponding attributes in the chassis
  - **Frequency setting**
    - **Each individual function supports a separate frequency setting, such as:**
      - *chassis push position on pfreq 1 attitude on* : Enable the position and attitude push. The position push frequency is 1 Hz, and the default setting of 5 Hz is used as the attitude push frequency
    - **Unified frequency setting is supported for all functions of the current module, such as:**
      - chassis push freq 10 #The push frequency is unified to 10 Hz for the chassis
      - chassis push position pfreq 1 freq 5 #If there is a freq parameter, pfreq is ignored
    - Supported frequencies: 1, 5, 10, 20, 30, and 50
  - For push data formats, refer to *Chassis Push Information Data*

- **Parameters**
  - *switch* (*switch_enum*) When *on* is used in the parameter here, the push of corresponding attributes is enabled; when *off* is used here, the push of corresponding attributes is disabled
  - *freq* (int:(1,5,10,20,30,50)) Push frequency of corresponding attributes
  - *freq_all* (int:(1,5,10,20,30,50)) : Push frequency of all relevant push information of the whole chassis
- **Example**
  - *chassis push attitude on* : Enable the push of chassis attitude information
  - *chassis push attitude on status on* Enable the push of chassis attitude and status information
  - *chassis push attitude on afreq 1 status on sfreq 5* Enable the push of chassis attitude information, the frequency of which is once per second, and, at the same time, enable the push of chassis status information, the frequency of which is five times per second
  - *chassis push freq 10* The push frequency of all chassis information is ten times per second

#### Chassis push information data

OUT: **chassis push <attr> <data>**

- **Description**
  - After the user enables chassis information push, the robot pushes the corresponding information to the user at the set frequency
- **Parameters**
  - *attr* (*chassis_push_attr_enum*) : The name of the subscribed attribute
  - ***data*** [The data of the subscribed attribute]
    - When *attr* is the **position**, the content of the *data* is <*x*> <*y*>
    - When *attr* is the **attitude**, the content of the *data* is <*pitch*> <*roll*> <*yaw*>
    - When *attr* is the **status**, the content of the *data* is <*static*> <*uphill*> <*downhill*> <*on_slope*> <*pick_up*> <*slip*> <*impact_x*> <*impact_y*> <*impact_z*> <*roll_over*> <*hill_static*>
- **Example**
  - *chassis push attitude 0.1 1 3* The pitch, roll, and yaw attitude information of the current chassis are 0.1, 1, and 3, respectively

### 3.2.4 Gimbal control

#### Gimbal motion speed control

IN: **gimbal speed p <speed> y <speed>**

- **Description**
  - Control the gimbal motion speed
- **Parameters**
  - *p* (float:[-450, 450]) pitch axis velocity in °/s
  - *y* (float:[-450, 450]) yaw axis velocity in °/s

- **Example**
  - *gimbal speed p 1 y 1* The pitch axis velocity of the gimbal is 1°/s, and the yaw axis velocity is 1°/s

### Gimbal relative position control

IN: **gimbal move { [p <degree>] [y <degree>] } [vp <speed>] [vy <speed>]**

- **Description**
  - Control the gimbal to move to a specified position. The origin of the coordinate axis is the current position
- **Parameters**
  - *p* (float:[-55, 55]) pitch axis angle in °
  - *y* (float:[-55, 55]) yaw axis angle in °
  - *vp* (float:[0, 540]) pitch axis velocity in °/s
  - *vy* (float:[0, 540]) yaw axis velocity in °/s
- **Example**
  - *gimbal move p 10* With the current position as the coordinate reference, control the gimbal to move to where the pitch axis angle is 10°

### Gimbal absolute position control

IN: **gimbal moveto { [p <degree>] [y <degree>] } [vp <speed>] [vy <speed>]**

- **Description**
  - Control the gimbal to move to a specified position. The origin of the coordinate axis is power-up position
- **Parameters**
  - *p* (int:[-25, 30]) pitch axis angle (°)
  - *y* (int:[-250, 250]) yaw axis angle (°)
  - *vp* (int:[0, 540]) pitch axis velocity (°/s)
  - *vy* (int:[0, 540]) yaw axis velocity (°/s)
- **Example**
  - *giimbal moveto p 10 y -20 vp 0.1* Taking the power-up position of the robot as the coordinate reference, control the gimbal to move to where the pitch axis angle is 10° and the yaw axis angle is -20°. As it moves, specify the pitch axis velocity as 0.1°/s

### Gimbal sleep control

IN: **gimbal suspend**

- **Description**
  - Control the gimbal to sleep
- **Example**

  - *gimbal suspend* Put the gimbal into sleep state

### Gimbal recovery control

IN: **gimbal resume**

- **Description**
  - Control the gimbal to recover from sleep state
- **Parameters**
  - *None*
- **Example**
  - *gimbal resume* Take the gimbal out of sleep state

> **Warning:** Sleep state When the gimbal goes into sleep state, the two-axis motor of the gimbal releases the control force, and the gimbal does not respond to any control command as a whole.
>
> To release the gimbal from sleep state, see *Gimbal recovery control*

### Gimbal recenter control

IN: **gimbal recenter**

- **Description**
  - Recenter the gimbal
- **Example**
  - *gimbal recenter* Control the gimbal to return to the center

### Obtaining gimbal attitude

IN: **gimbal attitude ?**

- **Description**
  - Obtain gimbal attitude information
- **Return values**
  - <*pitch*> <*yaw*> Pitch axis angle (°), yaw axis angle (°)
- **Example**
  - IN*gimbal attitude ?* Query gimbal angle information
  - OUT: *-10 20* The current pitch axis angle of the gimbal is -10°, and the yaw axis angle is 20°

### Gimbal information push control

IN: **gimbal push <attr> <switch> [afreq <freq_all>]**

- **Description**
  - Enable/disable the information push of corresponding attributes in the gimbal

  - For push data formats, refer to *Gimbal push information data*

- **Parameters**
  - *attr* (`gimbal_push_attr_enum`) : The name of the subscribed attribute
  - *switch* (`switch_enum`) When *on* is used in the parameter here, the push of corresponding attributes is enabled; when *off* is used here, the push of corresponding attributes is disabled
  - *freq_all* : Push frequency of all relevant push information of the gimbal
- **Example**
  - *gimbal push attitude on* Enable the push of gimbal information

#### Gimbal push information data

OUT: **gimabal push <attr> <data>**

- **Description**
  - After the user enables gimbal information push, the robot pushes the corresponding information to the user at the set frequency
- **Parameters**
  - *attr* (`gimbal_push_attr_enum`) : The name of the subscribed attribute
  - ***data*: The data of the subscribed attribute**
    - When *attr* is the **attitude**, the content of the *data* is <*pitch*> <*yaw*>
- **Example**
  - *gimbal push attitude 20 10* The pitch angle of the current gimbal is 20°, and the yaw angle is 10°

### 3.2.5 Blaster control

#### Blaster single emittance control

IN**blaster bead <num>**

- **Description**
  - Set the blaster single emittance
- **Parameters**
  - *num* (int:[1,5]) Emittance
- **Example**
  - *blaster bead 2* : Control the blaster to emit two at a time

#### Blaster emission control

IN: **blaster fire**

- **Description**
  - Control the water gun to fire once
- **Example**

  - *blaster fire* Control the water gun to fire once

#### Obtaining blaster single emittance

IN: **blaster bead ?**

- **Description**
  - Obtain the number of water bombs fired by the water gun at a single time
- **Return values**
  - <*num*> Number of water bombs fired by the water gun at a single time
- **Example**
  - IN: *blaster bead ?* Query the number of water bombs fired by the water gun at a single time
  - OUT: *3* At present, the number of water bombs fired by the water gun at a single time is 3

### 3.2.6 Armor plate control

#### Armor plate sensitivity control

IN: **armor sensitivity <value>**

- **Description**
  - Set the strike detection sensitivity of the armor plate
- **Parameters**
  - *value* (int:[1,10]) Armor plate sensitivity. The greater the value, the easier a strike is detected. The default sensitivity value is 5
- **Example**
  - *armor sensitivity 1* Set the strike detection sensitivity of the armor plate to 1

#### Obtaining armor plate sensitivity

IN: **armor sensitivity ?**

- **Description**
  - Obtain the strike detection sensitivity of the armor plate
- **Parameters**
  - <*value*> Armor plate sensitivity
- **Example**
  - IN: *armor sensitivity ?* Query the strike detection sensitivity of the armor plate
  - OUT: *5* Query the strike detection sensitivity of the armor plate

#### Armor plate event reporting control

IN: **armor event <attr> <switch>**

- **Description**
  - Control the armor plate detection event report
  - For event data formats, please refer to *Armor plate event reporting data*
- **Parameters**
  - *attr* (`armor_event_attr_enum`) : Event attribute name
  - *switch* (`switch_enum`) : Event attribute control switch
- **Example**
  - *armor event hit on* Enable the armor plate detection event push

#### Armor plate event reporting data

OUT: **armor event hit <index> <type>**

- **Description**
  - This message can be received from the event push port when an armor plate hit event occurs
- **Parameters**
  - ***index* (int:[1, 6]) Armor plate ID of the current hit event**
    - `1`
    - `2`
    - `3`
    - `4`
    - `5`
    - `6`
  - ***type* (int:[0, 2]) Types of current hit events**
    - `0` water bomb attack
    - `1` impact
    - `2` hand knock
- **Example**
  - *armor event hit 1 0* Water gun attack detected on armor plate 1

### 3.2.7 Sound recognition control

#### Sound recognition event reporting control

IN: **sound event <attr> <switch>**

- **Description**
  - Sound recognition time reporting control. Once enabled, related events will be reported

  - For event reporting data formats, refer to *Sound recognition event reporting data*

- **Parameters**
  - *attr* (`sound_event_attr_enum`) : Event attribute name
  - *switch* (`switch_enum`) : Event attribute control switch
- **Example**
  - *sound event applause on* Enable sound (applause) recognition

#### Sound recognition event reporting data

OUT: **sound event <attr> <data>**

- **Description**
  - When a specific sound event occurs, this data can be received from the event push port
  - To enable the event, please refer to *Sound recognition event reporting control*
- **Parameters**
  - *attr* (`sound_event_attr_enum`): Event attribute name
  - ***data* Event attribute data**
    - When *attr* is `applause`, the *data* is <*count*>, which indicates the number of applauses in a short time
- **Example**
  - *sound event applause 2* Recognize 2 claps in a short time

### 3.2.8 PWM control

#### PWM output duty cycle control

IN: **pwm value <port_mask> <value>**

- **Description**
  - PWM output duty cycle setting
- **Parameters**
  - *port_mask* (hex:0-0xffff) PWM expansion port mask combination. The corresponding mask of output port X is **1 << (X-1)**
  - *value* (float:0-100) PWM output duty cycle. The default output is 12.5
- **Example**
  - *pwm value 1 50* : Control the duty cycle of PWM port 1 to 50%

#### PWM output frequency control

IN: **pwm freq <port_mask> <value>**

- **Description**
  - PWM output frequency control

- **Parameters**
  - *port_mask* (hex:0-0xffff) PWM expansion port mask combination. The corresponding mask of output port X is **1 << (X-1)**
  - *value* (int:XXX) PWM output frequency value
- **Example**
  - *pwm freq 1 1000* : Control the frequency of PWM port 1 to 1,000 Hz

### 3.2.9 Sensor adaptor board control

#### Obtaining the ADC value of the sensor adaptor board

IN: **sensor_adapter adc id <adapter_id> port <port_num> ?**

- **Description**
  - Obtain the ADC value of the sensor adaptor board
- **Parameters**
  - *adapter_id* (int:[1, 6]) Adaptor board ID
  - *port_num* (int:[1, 2]) Port No.
- **Return values**
  - *adc_value* Measure the voltage value of the specified port on the corresponding adaptor board. The voltage has a value range of [0V, 3, 3V]
- **Example**
  - IN: *sensor_adapter adc id 1 port 1 ?* : Query the ADC value of port 1 on adaptor board 1
  - OUT: *1.1* The ADC value of the port currently queried is 1.1

#### Obtaining the IO value of the sensor adaptor board

IN: **sensor_adapter io_level id <adapter_id> port <port_num> ?**

- **Description**
  - Obtain the logic level of the IO port of the sensor adaptor board
- **Parameters**
  - *adapter_id* (int:[1, 6]) Adaptor board ID
  - *port_num* (int:[1, 2]) Port No.
- **Return values**
  - *io_level_value* Measure the logic level value of the specified port on the corresponding adaptor board. The value is 0 or 1
- **Example**
  - IN: *sensor_adapter io_level id 1 port 1 ?* Query the IO logic level of port 1 on adaptor board 1
  - OUT: *1* The IO value of the currently queried port is 1

### Obtaining the level jump time value of the IO pin of the sensor adaptor board

IN: **sensor_adapter pulse_period id <adapter_id> port <port_num>**

- **Description**
  - Obtain the level jump duration of the IO port of the sensor adaptor board
- **Parameters**
  - *adapter_id* (int:[1, 6])Adaptor board ID
  - *port_num* (int:[1, 2])Port No.
- **Return values**
  - *pulse_period_value*: The value of the level jump duration of the specified port on the corresponding adaptor board, in ms
- **Example**
  - *sensor_adapter pulse_period id 1 port 1* Query the level jump duration of port 1 on adaptor board 1

### Sensor adaptor board event reporting control

IN: **sensor_adapter event io_level <switch>**

- **Description**
  - Enable/disable the level transition event push of the sensor adaptor board. Once enabled, a message will be pushed when the level transition occurs on the IO. See [Level Transition Event Push of the Sensor Adaptor Board] (#sensor adaptor board level transition push) in the next chapter
- **Parameters**
  - *switch* (`switch_enum`)Control switch for level transition event reporting
- **Example**
  - *sensor_adapter event io_level on* Enable the level transition event push for the sensor adaptor board

### Sensor adaptor board event reporting data

OUT: *sensor_adapter event io_level (<id>, <port_num>, <io_level>)*

- **Description**
  - Push a message when the level of the sensor adaptor board changes. You can receive this message from the event push port
  - Enabling the level transition push of the sensor adaptor board is required. See *Sensor adaptor board event reporting control*
- **Parameters**
  - *id*Sensor adaptor board ID
  - *port_num*IO ID
  - *io_level*Current logic level value
- **Example**
  - *sensor_adapter event io_level (1, 1, 0)* At present, the logic level of IO 1 of adaptor board 1 jumps to 0

### 3.2.10 TOF control

#### TOF switch control

IN: **ir_distance_sensor measure <switch>**

- **Description**
  - Turn all infrared sensor switches on/off
- **Parameters**
  - *switch* (`switch_enum`)Infrared sensor switch
- **Example**
  - *ir_distance_sensor meaure on* Turn on all TOFs

#### Obtaining the TOF distance

IN: **ir_distance_sensor distance <id> ?**

- **Description**
  - Obtain the distance measured by the TOF with the specified ID
- **Parameters**
  - *id* (int:[1, 4])Infrared sensor ID
- **Return values**
  - *distance_value*Distance value measured by the TOF with the specified ID, in mm
- **Example**
  - IN: *ir_distance_sensor distance 1* Query the distance value measured by TOF 1
  - OUT: *1000* The distance value of the currently queried TOF is 1,000 mm

### 3.2.11 Servo control

#### Servo angle control

IN: **servo angle id <servo_id> angle <angle_value>**

- **Description**
  - Set the servo angle
- **Parameters**
  - *servo_id* (int:[1, 3])Servo ID
  - *angle_value* (float:[-180, 180])Specified angle in °
- **Example**
  - *servo angle id 1 angle 20* Control the angle of servo 1 to 20°

#### Servo speed control

IN: **servo speed id <servo_id> speed <speed_value>**

- **Description**
  - Set the speed of the specified servo
- **Parameters**
  - *servo_id* (int:[1, 3])Servo ID
  - *speed_value* (float:[-1800, 1800])Set speed value in °/s
- **Example**
  - *servo speed id 1 speed 20* The set speed of servo 1 is 10°/s

#### Servo stop control

IN: **servo stop**

- **Description**
  - Stop the servo
- **Example**
  - *servo stop* Control the servo to stop moving

#### Servo angle query

IN: **servo angle id <servo_id> ?**

- **Description**
  - Obtain the angle of the specified servo
- **Parameters**
  - *servo_id* (int:[1, 3])Servo ID
- **Return values**
  - *angle_value* : Specify the angle value of the servo
- **Example**
  - IN: *servo angle id 1 ?* Obtain the angle value of servo 1
  - OUT: *30* The angle value of the currently queried servo is 30°

### 3.2.12 Robotic arm control

#### Robotic arm relative position motion control

IN: **robotic_arm move x <x_dist> y <y_dist>**

- **Description**
  - Control the robotic arm to move a certain distance. The current position is the origin of coordinates
- **Parameters**

  - *x_dist* (float:[])x-axis movement distance in cm
  - *y_dist* (float:[]) y-axis movement distance in cm

- **Example**
  - *robotic_arm move x 5 y 5* Control the robotic arm to move 5 cm along the x-axis and 5 cm along the y-axis

### Robotic arm absolute position motion control

IN: **robotic_arm moveto x <x_pos> y <y_pos>**

- **Description**
  - Control the robotic arm to move to a certain position. The robot power-up position is the origin of coordinates
- **Parameters**
  - *x_pos* (float:[])x-axis move-to coordinate in cm
  - *y_pos* (float:[])y-axis move-to coordinate in cm
- **Example**
  - *robotic_arm moveto x 5 y 5* Control the x-axis of the robotic arm to move to the coordinate position of 5 cm, and the y-axis to move to the coordinate position of 5 cm

### Robotic arm recenter control

IN: **robotic_arm recenter**

- **Description**
  - Control the robotic arm to go back to the center
- **Parameters**
  - *None*
- **Example**
  - *robotic_arm recenter* Control the robotic arm to go back to the center

### Robotic arm movement stop control

IN: **robotic_arm stop**

- **Description**
  - Stop robotic arm movement
- **Parameters**
  - *None*
- **Example**
  - *robotic_arm stop* Stop robotic arm movement

#### Robotic arm absolute position query

IN: **robotic_arm position ?**

- **Description**
  - Obtain the position of the robotic arm
- **Parameters**
  - *None*
- **Return values**
  - **<*x_pos*> <*y_pos*>: The position coordinates of the robotic arm**
    - *x_pos*x-axis coordinate in cm
    - *y_pos*y-axis coordinate in cm
- **Example**
  - IN: *robotic_arm position ?* Query the position of the robotic arm
  - OUT*50 60* The distance between the position of the currently queried robotic arm and the calibration point is 50 cm in the x-axis direction and 60 cm in the y-axis direction

### 3.2.13 Gripper control

#### Gripper opening motion control

IN: **robotic_gripper open [leve <level_num>]**

- **Description**
  - Open the gripper
- **Parameters**
  - *level_num* (int:[1,4])The force of the gripper opening. The value range is [1,4]
- **Example**
  - *robotic_gripper open 1* Control the robotic arm to open with a force of 1

#### Gripper closing motion control

IN: **robotic_gripper close [leve <level_num>]**

- **Description**
  - Close the gripper
- **Parameters**
  - *level_num* (int:[1,4])The force of the gripper closing. The value range is [1,4]
- **Example**
  - *robotic_gripper close 1* Control the robotic arm to close with a force of 1

**Note:** Gripper control force

**The gripper control force** describes the movement speed of the gripper during the movement and the maximum clamping force in the locked rotor state

The greater the force, the faster the movement speed, and the greater the clamping force; vice versa.

#### Robotic arm relative position motion control

IN: **robotic_gripper status ?**

- **Description**
  - Obtain the opening and closing state of the gripper
- **Parameters**
  - *None*
- **Return values**
  - ***status*** [Current opening and closing state of the gripper] > `0` Gripper fully closed > `1` Gripper neither fully closed nor fully opened > `2` Gripper fully opened
- **Example**
  - IN: *robotic_gripper status ?* Obtain the opening and closing state of the gripper
  - OUT: *2* The currently queried gripper is open

### 3.2.14 Video streaming control

#### Video streaming enabling control

IN: **stream on**

- **Description**
  - Enable video streaming
  - Once enabled, the H.264 encoded bitstream data can be received from the video streaming port
- **Example**
  - *stream on* Enable video streaming

#### Video streaming disabling control

IN: **stream off**

- **Description**
  - Disable video streaming
  - Once video streaming is disabled, the H.264 encoded bitstream data stops being output
- **Example**
  - *stream off* Disable video streaming

### 3.2.15 Audio streaming control

#### Audio streaming enabling control

IN: **audio on**

- **Description**
  - Enable audio streaming
  - Once audio streaming is disabled, the Opus encoded audio streaming data can be received from the audio streaming port
- **Example**
  - *audio on* Enable audio streaming

#### Audio streaming disabling control

IN: **audio off**

- **Description**
  - Disable audio streaming
  - Once audio streaming is disabled, the Opus encoded audio stream data stops being output
- **Example**
  - *audio off* Disable audio streaming

### 3.2.16 IP broadcasting

OUT: **robot ip <ip_addr>**

- **Description**
  - When there is no connection with the robot, you can receive this message from the IP broadcast port. Once the connection is successful, the message stops broadcasting
  - The IP address of the current robot is provided. It is applicable to situations where the robot is in the same LAN with the robot, but the IP information of the robot is unknown
- **Parameters**
  - *ip_addr* : The robot's current IP address
- **Example**
  - *robot ip 192.168.1.102* : The robot's current IP address is 192.168.1.102

## 3.3 Data Description

**switch_enum**

- `on` : On
- `off` : Off

**mode_enum**

- `chassis_lead` : Chassis lead mode
- `gimbal_lead` : Gimbal lead mode
- `free` : Free mode

**chassis_push_attr_enum**

- `position` : Chassis position
- `attitude` : Chassis attitude
- `status` : Chassis status

**gimbal_push_attr_enum**

- `attitude` Gimbal attitude

**armor_event_attr_enum**

- `hit` : Armor hit

**sound_event_attr_enum**

- `applause` : Applause

# CHAPTER 4

# Multi-Machine Communication

multi_comm_ctrl.**set_group**(*send_group*, *recv_group_list*)

**Description** Set the group number of the machine to `send_group`. The machine can receive messages from the group numbers registered in `recv_group_list`. If the parameter `recv_group_list` is not used, messages from group number 0 are received by default

**Parameters**

- **send_group** (*int*) – The sending group number of the current machine. The default group number is 0
- **recv_group_list** (*list/tuple*) – The list of groups currently receiving messages. The type can be list or tuple

**Returns** None

**Example** `multi_comm_ctrl.set_group(1, (1,2,3))`

**Example description** Set the current sending group number to 1 and the receiving group numbers as 1, 2, and 3. If the receiving group includes the sending group, it will receive messages sent by itself

multi_comm_ctrl.**send_msg**(*msg*, *group*)

**Description** By sending a message through multi-machine communication, you can individually set the group to which the message is sent

**Parameters**

- **msg** (*int*) – Message to be sent
- **group** (*int*) – An optional parameter. It specifies the group to which the current message is sent. If not specified, the previously set group number will be used by default

**Returns** None

**Example** `multi_comm_ctrl.send('RoboMaster EP', 3)`

**Example description** Send the message `'RoboMaster EP'` to group number 3

multi_comm_ctrl.**recv_msg**(*timeout*)

**Description** Set a timeout when receiving messages (effective when *recv_callback* is not registered)

**Parameters** **timeout** (*int*) – Waiting time, i.e. the time that the receiving function has waited, with an accuracy of 1 second. The default setting is 72 seconds

**Returns** <msg_group>, <msg> The group number of the message sender and the message content

**Example** `group, recv_msg = multi_comm_ctrl.recv_msg(30)`

**Example description** When receiving messages, the waiting time is 30 seconds; group is the group number of the sender, and msg is the content of the received message

multi_comm_ctrl.**register_recv_callback**(*callback*)

**Description** Register the callback function used to receive messages. When the message is received, the callback function is executed automatically

**Parameters** **callback** (*function*) – The callback function to be registered. The prototype of the callback function is `def callback(msg)`, where the `msg` parameter type is the tuple `(msg_group, msg)`

**Returns** None

**Example**

```
#Define a function and register it as the callback function used to receive messages

def recv_callback(msg):
    pass

multi_comm_ctrl.register_recv_callback(recv_callback)
```

CHAPTER 5

# Custom UI System

## 5.1 Summary

The custom UI system is a way for users to expand the input and output of programs. Users compose their own programs to generate user-defined UI controls.

A very important part of our programming work is to process inputs and outputs. For our robot, the program output can be the action of the chassis, gimbal, water gun, and other modules, or the performance of light and sound effects. Input includes initial variables, the visual recognition of the robot, applause recognition, armor plate attack detection, and cell phone gyroscope, among others. We can now achieve the purpose of input through the interaction between the custom UI system and the generated UI control, or output the processing results of the program through UI controls.

We can compose a Python program in the RoboMaster app to generate UI controls and bind the event callback of the controls by calling the relevant interface of the custom UI system. After composing and debugging the program in the laboratory, we may assemble it into custom skills, which can be used in solo practice or multi-player competitions.

## 5.2 Interfaces

### 5.2.1 Common

The methods in this section are applicable to all custom UI controls except Stage, so a separate introduction is made here.

`common_object.`**`set_active`**`(`*`status`*`)`

**Description** Control whether the current control is displayed

**Parameters** **`status`** (*`bool`*) – The active state of the control. True means the current control is displayed, and False means the current control is hidden

**Returns** None

**Example** `my_Slider.set_active(False)`

**Example description** Set the my_Slider control to the hidden state

`common_object.`**`get_active`**`()`

**Description** Obtain the display status of the current control

**Parameters** **`void`** – None

**Returns** Bool type. It represents the display status of the control

**Example** `status = my_Slider.get_active()`

**Example description** Obtain the display status of the my_Slider control and assign it to the status variable

`common_object.`**`set_name`**`(`*name*`)`

**Description** Set the current control's name

**Parameters** **`name`** (*`string`*) – The name of the control

**Returns** None

**Example** `my_Dropdown.set_name('my_dropdown')`

**Example description** Set the my_Dropdown control name to "my_dropdown"

`common_object.`**`get_name`**`()`

**Description** Obtain the current control's name

**Parameters** **`void`** –

**Returns** String type. It represents the control's name

**Example** `name = my_Dropdown.get_name()`

**Example description** Obtain the my_Dropdown control's name and assign it to the name variable

`common_object.`**`set_position`**`(`*x*, *y*`)`

**Description** Set the position coordinates of the control, with the origin at the center of the screen

**Parameters**

- **`x`** (*`int`*) – The horizontal coordinate of the control. The value is the position of the actual pixel on the screen. Point 0 is in the horizontal center of the screen, and the right part is the positive direction
- **`y`** (*`int`*) – The vertical coordinate of the control. The value is the position of the actual pixel on the screen. Point 0 is in the vertical center of the screen, and the upward part is the positive direction

**Returns** None

**Example** `my_Text.set_position(-200, 500)`

**Example description** Set the coordinates of the my_Text control to (-200, 500)

`common_object.`**`get_position`**`()`

**Description** Obtain the position coordinates of the control

**param void** None

**return** [x,y]. It represents the position of the control

**Example** `pos = my_Text.get_position()`

**Example description** Obtain the position of the my_Text control and assign it to the pos variable, which is a list

common_object.**set_size**(*w*, *h*)

**Description** Set the size of the control

**Parameters**

- **w** (*int*) – The width of the control
- **h** (*int*) – The height of the control

**Returns** None

**Example** `my_Button.set_size(300, 200)`

**Example description** Set the width of the my_Button control to 300 and the height to 200

common_object.**get_size**()

**Description** Obtain the size of the control

**Parameters** **void** – None

**Returns** [w,h]. It represents the control size

**Example** `size = my_Button.get_size()`

**Example description** Obtain the my_Button control's size, and assign it to the size variable, which is a list

common_object.**set_rotation**(*degree*)

**Description** Set the rotation angle of the control

**Parameters** **degree** (*int*) – The rotation angle of the control. The range is [0, 360]. A positive value indicates clockwise rotation, and a negative value indicates counterclockwise rotation

**Returns** None

**Example** `my_Button.set_rotation(90)`

**Example description** Set the my_Button control to rotate 90 degrees clockwise

common_object.**get_rotation**()

**Description** Obtain the rotation angle of the control

**Parameters** **void** – None

**Returns** An integer. Indicates the rotation angle of the control. The range is [0, 360]. A positive value indicates clockwise rotation, and a negative value indicates counterclockwise rotation

**Example** `degree = my_Button.get_rotation()`

**Example description** Obtain the rotation angle of the my_Button control and assign it to the degree variable

common_object.**set_privot**(*x*, *y*)

**Description** Set the anchor coordinates of the control. The input parameter is normalized. The origin is located in the lower left corner of the control. The anchor of the control is defaulted to the control center, that is, (0.5,0.5). The anchor is used as the control point for the position and rotation of the control

**Parameters**

- **x** (*int*) – The x coordinate of the anchor point. The range is [0, 1], and the right part is the positive direction
- **y** (*int*) – The y coordinate of the anchor point. The range is [0, 1], and the upward part is the positive direction

**Returns** None

**Example** `my_Button.set_pivot(0, 1)`

**Example description** Set the anchor point of the control to the top left corner of the control

`common_object.`**`get_privot`**`()`

**Description** Obtain the anchor coordinates of the control

**Parameters** **`void`** – None

**Returns** [x,y]. Indicates the anchor coordinates of the control

**Example** `pivot = my_Button.get_pivot()`

**Example description** Obtain the anchor coordinate of the control, and assign it to the pivot variable, which is a list

`common_object.`**`set_order`**`(`*order*`)`

**Description** Set the display priority of the control. When multiple controls overlap, the control with a higher priority is at the top. The higher the number, the higher the priority

**Parameters** **`order`** (*int*) – The specified priority of the control. When several controls overlap, the control with a higher priority is displayed first

**Returns** None

**Example** `my_Button.set_order(8)`

**Example description** Set the display priority of the control to 8. When several controls overlap, the controls below this priority will be overwritten

`common_object.`**`get_order`**`()`

**Description** Obtain the display priority of the control

**Parameters** **`void`** – None

**Returns** An integer. Indicates the display priority of the control

**Example** `order = my_Button.get_order()`

**Example description** Obtain the display priority of the my_Button control and assign it to the order variable

`common_object.`**`callback_register`**`(`*event*, *callback*`)`

**Description** The callback function triggered by the control registration event. When the control detects the corresponding event, the registered callback function is executed

**param string event** Specifies the trigger event for the callback function

The events that can be registered for each control are as follows:

- **Button control:**
  - `on_click`: In a button press and release process, the event is triggered when the button is released
  - `on_press_down`: This event is triggered when the button is pressed

  - `on_press_up`: This event is triggered when the button is released

- **Toggle control:**
  - `on_value_changed`: This event is triggered when the value is changed. The args parameter in the callback function is a bool type, indicating the changed toggle control value

- **Dropdown control:**
  - `on_value_changed`: This event is triggered when the value is changed. The args parameter in the callback function is an integer, indicating the selected index after the Dropdown control value is changed

- **Text control:**
  - No trigger event

- **InputField control:**
  - `on_value_changed`: This event is triggered when the value is changed. The args parameter in the callback function is a string type, indicating the changed InputField control value

**param function callback** The callback function to be registered. The unified signature of the callback function is: `def callback(widget,*args,**kw):`, where widget is the control reference of the trigger event, and args is a parameter; TODO: Supplementary parameter description

**return** None

**Example 1**

```
# After the my_Button control is clicked, information is printed to the console, and the robot shoots once

def button_callback(widget,*args,**kw):
    print('the button is clicked and the button's name is '+ widget.get_name())
    gun_ctrl.fire_once()
my_Button.callback_register('on_click',button_callback)
```

**Example 2**

```
# After the my_Toggle control is clicked, the value changes, information is printed to the console, and the robot plays the sound

def toggle_callback(widget,*args,**kw):
    print('the toggle's value is changed and the toggle's name is '+ widget.get_name())
    print('the toggle's value now is '+ str(args))
    media_ctrl.play_sound(rm_define.media_sound_recognize_success)
my_Toggle.callback_register('on_value_changed',toggle_callback)
```

**Example 3**

```
# When you click the my_Dropdown control to change its selected value, the value changes, information is printed to the console, and the robot plays the sound

def dropdown_callback(widget,*args,**kw):
    print('the dropdown's value is changed and the dropdown's name is '+ widget.get_name())
```

(continues on next page)

(continued from previous page)

```
    print('the dropdown's value now is '+ str(args))
    media_ctrl.play_sound(rm_define.media_sound_solmization_1A)
my_Dropdown.callback_register('on_value_changed',dropdown_callback)
```

**Example 4**

```
# When you click my_InputField control to change its selected value, the value changes and information is printed to the console

def input_field_callback(widget,*args,**kw):
    print('the input_field's value is changed and the input_field's name is '+ widget.get_name())
    print('the input_field's value now is '+ str(args))
my_InputField.callback_register('on_value_changed',input_field_callback)
```

### 5.2.2 Stage

As the system initializes, it automatically creates a Stage class object stage, which can be used directly, and does not need to be created by the user.

`object.`**`add_widget`**(*widget_obj*)

**Description** Add controls from parameters to the UI

**Parameters** **`widget_obj`** (*object*) – The control object to be added to the UI

**Returns** None

**Example**

```
#Create a Button object and add it to the UI

my_button = Button()
stage.add_widget(my_button)
```

`stage_object.`**`remove_widget`**(*widget_obj*)

**Description** Remove the control passed in by the parameter from the UI

**Parameters** **`widget_obj`** (*object*) – The control that needs to be removed from the UI

**Returns** N/A

**Example** `stage.remove_widget(my_button)`

**Example description** Remove the my_button control from the UI

### 5.2.3 Button

The Button control is used to initiate or confirm an action in response to a click from the user.

`button_object.`**`set_text`**(*content*[, *color_r*, *color_g*, *color_b*, *color_a* ], *align*, *size*)

**Description** Set text properties for button objects

**Parameters**

- **`content`** (*string*) – The string displayed on a button

- **[color_r, color_g, color_b, [color_a]]** (*list*) – Optional parameters. The color of the string to be displayed. The parameters are the display color's r value, b value, g value, and transparency. The value range is [0, 255]
- **align** (*enum*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *align*
- **size** (*int*) – The font size of the display text

**Returns** None

**Example** `my_Button.set_text(120, 120, 120, 255, text_anchor.upper_left, 12)`

**Example description** Set the RGB value of the text color to (120, 120, 120), the transparency to 255, the text alignment to top left, and the font size to 12

`button_object.`**`set_text_color`**(*r*, *g*, *b*[, *a*])

**Description** Set the color of the text

**Parameters**

- **r** (*int*) – The R value of the text color. The value range is [0, 255]
- **g** (*int*) – The G value of the text color. The value range is [0, 255]
- **b** (*int*) – The B value of the text color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. Transparency. The value range is [0, 255]

**Returns** None

**Example** `my_button.set_text_color(120, 120, 120, 200)`

**Example description** Set the RGB value of the text color to (120, 120, 120), and transparency to 200

`button_object.`**`set_text_align`**(*align*)

**Description** Set the text alignment

**Parameters** **align** (*enum*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *align*

**Returns** None

**Example** `my_button.set_text_align(text_anchor.upper_left)`

**Example description** Set the text alignment to top left

`button_object.`**`set_text_size`**(*size*)

**Description** Set the text's font size

**Parameters** **size** (*int*) – The font size value of the text

**Returns** None

**Example** `my_button.set_text_size(12)`

**Example description** Set the font size of the text to 12

`button_object.`**`set_background_color`**(*r*, *g*, *b*[, *a*])

**Description** Set the background color of the button

**Parameters**

- **r** (*int*) – The R value of the font color. The value range is [0, 255]
- **g** (*int*) – The G value of the font color. The value range is [0, 255]
- **b** (*int*) – The B value of the font color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the font color. The value range is [0, 255]

**Returns** None

**Example** `my_button.set_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value of the background color to (200, 200, 200), and the transparency to 230

### 5.2.4 Toggle

The Toggle control is used to draw a switch on the screen. It can perform some specific operations by controlling the opening and closing of the switch.

`toggle_object.`**`set_text`**(*string*[, *color_r*, *color_g*, *color_b*, *color_a* ], *align*, *size*)

**Description** Set the display text of the control

**Parameters**

- **string** (*string*) – String content to be displayed on the control
- **[color_r, color_g, color_b, color_a]** (*list*) – An optional parameter. The color of the string to be displayed. The parameters are the display color's r value, b value, g value, and transparency. The value range is [0, 255]
- **align** (*enum*) – An optional parameter. Enumeration type. The alignment of the text to be displayed. For details, see table *align*
- **size** (*int*) – The font size of the display text

**Returns** N/A

**Example**
```
my_Toggle.set_text(120, 120, 120, 200, text_anchor.
    upper_left, 12)
```

**Example description** Set the RGB value of the text to (120, 120, 120), the transparency to 200, the text alignment to top left, and the font size to 12

`toggle_object.`**`set_text_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the color of the text

**Parameters**

- **r** (*int*) – The R value of the text color. The value range is [0, 255]
- **g** (*int*) – The G value of the text color. The value range is [0, 255]
- **b** (*int*) – The B value of the text color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the text color. The value range is [0, 255]

**Returns** None

**Example** `my_Toggle.set_text_color(120, 120, 120, 200)`

**Example description** Set the RGB value of the font to (120, 120, 120), and transparency to 200

toggle_object.**set_text_align**(*align*)

**Description** Set the text alignment

**Parameters** **align** (*enum*) – An optional parameter. Enumeration type. The alignment of the text to be displayed. For details, see table *align*

**Returns** None

**Example** `my_Toggle.set_text_align(text_anchor.upper_left)`

**Example description** Set the text alignment to top left

toggle_object.**set_text_size**(*size*)

**Description** Set the text's font size

**Parameters** **size** (*int*) – The font size value of the text

**Returns** None

**Example** `my_Toggle.set_text_size(12)`

**Example description** Set the font size of the text to 12

toggle_object.**set_background_color**(*r*, *g*, *b*[, *a* ])

**Description** Set the background color for the control

**Parameters**

- **r** (*int*) – The R value of the background color. The value range is [0, 255]
- **g** (*int*) – The G value of the background color. The value range is [0, 255]
- **b** (*int*) – The B value of the background color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the background color. The value range is [0, 255]

**Returns** N/A

**Example** `my_Toggle.set_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value of the background color to (200, 200, 200), and the transparency to 230

toggle_object.**set_checkmark_color**(*r*, *g*, *b*[, *a* ])

**Description** Set the color for the selected icons of the control

**Parameters**

- **r** (*int*) – The R value of the icon color. The value range is [0, 255]
- **g** (*int*) – The G value of the icon color. The value range is [0, 255]
- **b** (*int*) – The B value of the icon color. The value range is [0, 255]
- **a** (*int*) – The transparency of the icon color. The value range is [0, 255]

**Returns** N/A

**Example** `my_Toggle.set_checkmark_color(200, 200, 200, 230)`

**Example description** Set the RGB value of the selected icon to (200, 200, 200), and the transparency to 230

toggle_object.**set_is_on**(*status*)

**Description** Set the state of the control

**Parameters** **status** (*bool*) – Set whether the control is open. True means open, and False means closed

**Returns** N/A

**Example** `my_Toggle.set_is_on(True)`

**Example description** Set the Toggle control to open

### 5.2.5 Text

The Text control is used to display texts

`text_object.`**`set_text`**(*string*[, *color_r*, *color_g*, *color_b*, *color_a* ], *align*, *size*)

**Description** Set the text properties of the control

**Parameters**

- **string** (*string*) – String content to be displayed
- **[color_r, color_g, color_b, color_a]** (*list*) – An optional parameter. The color of the string to be displayed. The parameters are the display color's r value, b value, g value, and transparency. The value range is [0, 255]
- **align** (*enum*) – An optional parameter. Enumeration type. The alignment of the text to be displayed. For details, see table *align*
- **size** (*int*) – The font size of the display text

**Returns** N/A

**Example**
```
my_Text.set_text(120, 120, 120, 200, text_anchor.
    upper_left, 12)
```

**Example description** Set the RGB value of the text color to (120, 120, 120), the transparency to 200, the text alignment to top left, and the font size to 12

`text_object.`**`set_text_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the text color of the control

**Parameters**

- **r** (*int*) – The R value of the text color. The value range is [0, 255]
- **g** (*int*) – The G value of the text color. The value range is [0, 255]
- **b** (*int*) – The B value of the text color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the text color. The value range is [0, 255]

**Returns** N/A

**Example** `my_Text.set_text_color(120, 120, 120, 200)`

**Example description** Set the RGB value of the text to (120, 120, 120), and transparency to 200

`text_object.`**`set_text_align`**(*align*)

**Description** Set the text alignment

**Parameters** **align** (*enum*) – An optional parameter. Enumeration type. The alignment of the text to be displayed. For details, see table *align*

**Returns** None

**Example** `my_Text.set_text_align(text_anchor.upper_left)`

**Example description** Set the text alignment to top left

`text_object.`**`set_text_size`**(*size*)

**Description** Set the text's font size

**Parameters** **`size`** (*int*) – The font size value of the text

**Returns** None

**Example** `my_Text.set_text_size(12)`

**Example description** Set the font size of the text to 12

`text_object.`**`set_border_active`**(*active*)

**Description** Display the text border or not

**Parameters** **`active`** (*bool*) – Whether the text border is displayed. True means to display the border, and False means not to display the border

**Returns** N/A

**Example** `my_Text.set_border_active(True)`

**Example description** Display the text border

`text_object.`**`set_background_color`**($r, g, b[, a]$)

**Description** Set the background color for the control

**Parameters**

- **`r`** (*int*) – The R value of the background color. The value range is [0, 255]
- **`g`** (*int*) – The G value of the background color. The value range is [0, 255]
- **`b`** (*int*) – The B value of the background color. The value range is [0, 255]
- **`a`** (*int*) – An optional parameter. The transparency of the background color. The value range is [0, 255]

**Returns** N/A

**Example** `my_Text.set_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value of the background color to (200, 200, 200), and the transparency to 230

`text_object.`**`append_text`**(*content*)

**Description** Add text to the Text control

**Parameters** **`content`** (*string*) – The text to be added to Text

**Returns** N/A

**Example** `my_Text.append_text('RoboMaster EP')`

**Example description** The text to be added to Text: `RoboMaster EP`

**`align`**

| | |
|---|---|
| text_anchor.upper_left | Top left aligned |
| text_anchor.upper_center | Top center aligned |
| text_anchor.upper_right | Top right aligned |
| text_anchor.middle_left | Middle left aligned |
| text_anchor.middle_center | Middle center aligned |
| text_anchor.middle_right | Middle right aligned |
| text_anchor.lower_left | Bottom left aligned |
| text_anchor.lower_center | Bottom center aligned |
| text_anchor.lower_right | Bottom right aligned |

### 5.2.6 InputField

The InputField control is used to receive textual information input by users

`inputfield_object.`**`set_text`**(*string*[, *color_r*, *color_g*, *color_b*, *color_a* ], *align*, *size*)

**Description** Set text properties for input field objects

**Parameters**

- **`string`** (*`string`*) – Strings to be displayed
- **`[color_r, color_g, color_b, color_a]`** (*`list`*) – Optional parameters. The color of the string to be displayed. The parameters are the display color's r value, b value, g value, and transparency. The value range is [0, 255]
- **`align`** (*`enum`*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *`align`*
- **`size`** (*`int`*) – The font size of the display text

**Returns** None

**Example**
```
my_InputField.set_text(120, 120, 120, 200, text_anchor.
    upper_left, 12)
```

**Example description** Set the RGB value of the font to (120, 120, 120), the transparency to 200, the text alignment to top left, and the font size to 12

`input_field_object.`**`set_text_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the color of the text

**Parameters**

- **`r`** (*`int`*) – The R value of the text color. The value range is [0, 255]
- **`g`** (*`int`*) – The G value of the text color. The value range is [0, 255]
- **`b`** (*`int`*) – The B value of the text color. The value range is [0, 255]
- **`a`** (*`int`*) – An optional parameter. The transparency of the text color. The value range is [0, 255]

**Returns** None

**Example**
```
my_button.set_text_color(120, 120, 120, 200)
```

**Example description** Set the RGB value of the font to (120, 120, 120), and transparency to 200

`input_field_object.`**`set_text_align`**(*align*)

**Description** Set the text alignment for the control

**Parameters** **align** (*enum*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *align*

**Returns** None

**Example** `my_Input_field.set_text_align(text_anchor.upper_left)`

**Example description** Set the text alignment to top left

`input_field_object.`**`set_text_size`**(*size*)

**Description** Set the text font size for the control

**Parameters** **size** (*int*) – The font size value of the text

**Returns** None

**Example** `my_Input_field.set_text_size(12)`

**Example description** Set the font size of the text to 12

`input_field_object.`**`set_background_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the background color for the control

**Parameters**

- **r** (*int*) – The R value of the background color. The value range is [0, 255]
- **g** (*int*) – The G value of the background color. The value range is [0, 255]
- **b** (*int*) – The B value of the background color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the background color. The value range is [0, 255]

**Returns** None

**Example** `my_Input_field.set_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value of the background color to (200, 200, 200), and the transparency to 230

`input_field_object.`**`set_hint_text`**(*string*[, *color_r*, *color_g*, *color_b*, *color_a* ], *align*, *size*)

**Description** Set properties for the hint text within the control

**Parameters**

- **string** (*string*) – Strings to be displayed
- **[color_r, color_g, color_b, color_a]** (*list*) – Optional parameters. The color of the string to be displayed. The parameters are the display color's r value, b value, g value, and transparency. The value range is [0, 255]
- **align** (*enum*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *align*
- **size** (*int*) – The font size of the display text

**Returns** None

**Example** `my_Input_field.set_hint_text(120, 120, 120, 200, text_anchor.upper_left, 12)`

**Example description** Set the RGB value of the hint text to (120, 120, 120), the transparency to 200, the text alignment to top left, and the font size to 12

`input_field_object.`**`set_hint_text_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the color of the control's hint text

**Parameters**

- **r** (*int*) – The R value of the text color. The value range is [0, 255]
- **g** (*int*) – The G value of the text color. The value range is [0, 255]
- **b** (*int*) – The B value of the text color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the text color. The value range is [0, 255]

**Returns** None

**Example** `my_Input_field.set_text_color(120, 120, 120, 200)`

**Example description** Set the RGB value of the hint text to (120, 120, 120), and transparency to 200

`input_field_object.`**`set_hint_text_align`**(*align*)

**Description** Set the alignment of the hint text

**Parameters** **align** (*enum*) – An optional parameter. Enumeration type. It represents the alignment of the text to be displayed. For details, please see table *align*

**Returns** None

**Example** `my_Input_field.set_text_align(text_anchor.upper_left)`

**Example description** Set the alignment of the hint text to top left

`input_field_object.`**`set_hint_text_size`**(*size*)

**Description** Set the font size of the hint text

**Parameters** **size** (*int*) – The font size value of the text

**Returns** None

**Example** `my_Input_field.set_text_size(12)`

**Example description** Set the font size of the text within hint objects to 12

### 5.2.7 Dropdown

The Dropdown control is usually used to select a specific value from multiple property options of an object

`dropdown_object.`**`set_option`**(**options*)

**Description** Set the content of the drop-down box. The input content is a string list, and the number of elements in the list is the number of options in the drop-down box

**Parameters** ***args** (*string*) – Options in the drop-down box

**Returns** None

**Example** `my_Dropdown.set_option('RoboMaser EP', 'People')`

**Example description** There are two options in the drop-down box: `RoboMaster EP` and `People`

`dropdown_object.`**`set_text_color`**($r, g, b[, a]$)

**Description** Set the color of the text

**Parameters**

- **r** (*int*) – The R value of the text color. The value range is [0, 255]
- **g** (*int*) – The G value of the text color. The value range is [0, 255]
- **b** (*int*) – The B value of the text color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the text color. The value range is [0, 255]

**Returns** None

**Example** `my_Dropdown.set_text_color(120, 120, 120, 200)`

**Example description** Set the RGB value of the text to (120, 120, 120), and transparency to 200

`dropdown_object.`**`set_background_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the background color of the selected item in the drop-down box

**Parameters**

- **r** (*int*) – The R value of the background color. The value range is [0, 255]
- **g** (*int*) – The G value of the background color. The value range is [0, 255]
- **b** (*int*) – The B value of the background color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the background color. The value range is [0, 255]

**Returns** None

**Example** `my_DropDown.set_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value for the background color of the selected item in the drop-down box to (200, 200, 200), and transparency to 230

`dropdown_object.`**`set_arrow_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the arrow color of the drop-down box

**Parameters**

- **r** (*int*) – The R value of the arrow color. The value range is [0, 255]
- **g** (*int*) – The G value of the arrow color. The value range is [0, 255]
- **b** (*int*) – The B value of the arrow color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the arrow color. The value range is [0, 255]

**Returns** None

**Example** `my_Dropdown.set_arrow_color(120, 120, 120, 200)`

**Example description** Set the RGB value for the color of the selected arrow in the drop-down box to (120, 120, 120), and transparency to 200

`dropdown_object.`**`set_item_color`**(*r*, *g*, *b*[, *a* ])

**Description** Set the font color of the unselected items in the drop-down box

**Parameters**

- **r** (*int*) – The R value of the font color. The value range is [0, 255]
- **g** (*int*) – The G value of the font color. The value range is [0, 255]
- **b** (*int*) – The B value of the font color. The value range is [0, 255]

- **a** (*int*) – An optional parameter. The transparency of the font color. The value range is [0, 255]

**Returns** None

**Example** `my_Dropdown.set_item_color(120, 120, 120, 200)`

**Example description** The RGB value for the font color of unselected items in the drop-down box is (120, 120, 120), and the transparency is 200

`dropdown_object.`**`set_item_background_color`**$(r, g, b[, a])$

**Description** Set the background color of unselected items in the drop-down box

**Parameters**

- **r** (*int*) – The R value of the background color. The value range is [0, 255]
- **g** (*int*) – The G value of the background color. The value range is [0, 255]
- **b** (*int*) – The B value of the background color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the background color. The value range is [0, 255]

**Returns** None

**Example** `my_DropDown.set_item_background_color(200, 200, 200, 230)`

**Example description** Set the RGB value for the background color of unselected items in the drop-down box to (200, 200, 200), and transparency to 230

`dropdown_object.`**`set_item_checkmark_color`**$(r, g, b[, a])$

**Description** Set the color for the selected icon in the drop-down box

**Parameters**

- **r** (*int*) – checkmarkThe R value of the check mark color. The value range is [0, 255]
- **g** (*int*) – checkmarkThe G value of the check mark color. The value range is [0, 255]
- **b** (*int*) – checkmarkThe B value of the check mark color. The value range is [0, 255]
- **a** (*int*) – An optional parameter. The transparency of the check mark color. The value range is [0, 255]

**Returns** None

**Example** `my_DropDown.set_item_checkmark_color(200, 200, 200, 230)`

**Example description** Set the RGB value for the color of the check mark in the drop-down box to (200, 200, 200), and transparency to 230

CHAPTER 6

# Blaster

ir_blaster_ctrl.**set_fire_count**(*count*)

**Description** Set the emission frequency of infrared beams, i.e. the number of infrared beams emitted per second

**Parameters** **color_enum** (*int*) – Emission frequency, i.e. the number of infrared beams emitted per second. The range is [1:8]

**Returns** None

**Example** `ir_blaster_ctrl.set_fire_count(4)`

**Example description** Set the emission frequency of infrared beams to 4

ir_blaster_ctrl.**fire_once**()

**Description** Control the blaster to emit infrared beams once only

**Parameters** **void** – None

**Returns** None

**Example** `ir_blaster_ctrl.fire_once()`

**Example description** Control the blaster to emit infrared beams once only

ir_blaster_ctrl.**fire_continuous**()

**Description** Control the blaster to emit infrared beams continuously

**Parameters** **void** – None

**Returns** None

**Example** `ir_blaster_ctrl.fire_continuous()`

**Example description** Control the blaster to emit infrared beams continuously

ir_blaster_ctrl.**stop**()

**Description** Stop emitting infrared beams

**Parameters** **void** – None

**Returns** None

**Example** `ir_blaster_ctrl.stop()`

**Example description** Stop emitting infrared beams

CHAPTER 7

# Extension Modules

## 7.1 Gripper

`gripper_ctrl.`**`open`**`()`

**Description** Control the gripper to open

**Parameters** **`void`** – None

**Returns** None

**Example** `gripper_ctrl.open()`

**Example description** Control the gripper to open

`gripper_ctrl.`**`close`**`()`

**Description** Control the gripper to close

**Parameters** **`void`** – None

**Returns** None

**Example** `gripper_ctrl.close()`

**Example description** Control the gripper to close

`gripper_ctrl.`**`stop`**`()`

**Description** Control the gripper to stop moving

**Parameters** **`void`** – None

**Returns** None

**Example** `gripper_ctrl.stop()`

**Example description** Control the gripper to stop moving

`gripper_ctrl.`**`update_power_level`**`(`*level*`)`

**Description** Set the force of the gripper

**Parameters** **level** (*int*) – The force of the gripper. The range is [1:4], and the default is 1

**Returns** None

**Example** `gripper_ctrl.update_power_level(1)`

**Example description** Set the force of the gripper to 1

`gripper_ctrl.`**`is_closed`**`()`

**Description** Obtain the clamping state of the gripper

**Parameters** **void** – None

**Returns** The clamping state of the gripper. If the gripper is clamped, it returns true; otherwise, it returns false

**Return type** bool

**Example** `ret = gripper_ctrl.is_closed()`

**Example description** Obtain the clamping state of the gripper

`gripper_ctrl.`**`is_open`**`()`

**Description** Obtain the opening state of the gripper

**Parameters** **void** – None

**Returns** The opening state of the gripper. If the gripper is fully open, it returns true; otherwise, it returns false

**Return type** bool

**Example** `ret = gripper_ctrl.is_open()`

**Example description** Obtain the opening state of the gripper

## 7.2 Robotic Arm

`robotic_arm_ctrl.`**`move`**(*x*, *y*, *wait_for_complete=True*)

**Description** Set the relative position of the robotic arm movement

**Parameters**

- **x** (*int32*) – Set the distance of the horizontal movement of the robotic arm. A positive number is forward movement, and a negative number is backward movement. The accuracy is 1 mm
- **y** (*int32*) – Set the distance of the vertical movement of the robotic arm. A positive number is upward movement, and a negative number is downward movement. The accuracy is 1 mm
- **wait_for_complete** (*bool*) – Whether to wait for execution to complete. The default setting is True

**Returns** N/A

**Example** `robotic_arm_ctrl.move(40, 50, True)`

**Example description** Set the robotic arm to move forward by 40 mm and upward by 50 mm, and wait for the execution to complete

`robotic_arm_ctrl.`**`moveto`**(*x*, *y*, *wait_for_complete=True*)

**Description** Set the movement of the robotic arm to an absolute coordinate (the absolute coordinate system and range are not given?)

**Parameters**

- **x** (*int32*) – Set the coordinate value of the horizontal movement of the robotic arm, with an accuracy of 1 mm
- **y** (*int32*) – Set the coordinate value of the vertical movement of the robotic arm, with an accuracy of 1 mm
- **wait_for_complete** (*bool*) – Whether to wait for execution to complete. The default setting is True

**Returns** N/A

**Example** `robotic_arm_ctrl.moveto(40, 50, True)`

**Example description** Set the robotic arm to move to the absolute coordinate (x = 40mm, y = 50mm), and wait for the execution to complete

`robotic_arm_ctrl.`**`get_position`**`()`

**Description** Obtain the position of the robotic arm

**Parameters** **void** – N/A

**Returns** The absolute coordinate of the robotic arm, with an accuracy of 1 mm

**Return type** List [x, y] x and y are int32 types

**Example** `[x, y] = robotic_arm_ctrl.get_position()`

**Example description** Obtain the absolute position of the robotic arm

`robotic_arm_ctrl.`**`recenter`**`()`

**Description** Set the robotic arm to go back to the center (move to an absolute coordinate?)

**Parameters** **void** – N/A

**Returns** N/A

**Example** `robotic_arm_ctrl.recenter()`

**Example description** Set the robotic arm to go back to the center

## 7.3 Servo

`servo_ctrl.`**`get_angle`**(*servo_id*)

**Description** Obtain the servo rotation angle

**Parameters** **servo_id** (*uint8*) – Servo number. The range is [1:4]

**Returns** Servo angle, with an accuracy of 0.1 degrees

**Return type** int32

**Example** `angle = servo_ctrl.get_angle(1)`

**Example description** Obtain the rotation angle of servo 1

`servo_ctrl.`**`set_angle`**(*servo_id*, *angle*, *wait_for_complete=True*)

**Description** Set the servo rotation angle

**Parameters**

- **servo_id** (*uint8*) – Servo number. The range is [1:4]
- **angle** (*int32*) – Rotation angle, with an accuracy of 0.1 degrees. A positive number indicates clockwise rotation, and a negative number indicates counterclockwise rotation
- **wait_for_complete** (*bool*) – Whether to wait for execution to complete. The default setting is True

**Returns** None

**Example** `servo_ctrl.set_angle(1, 900, True)`

**Example description** Set servo 1 to rotate 90° clockwise, and wait for the execution to complete

`servo_ctrl.`**recenter**(*servo_id*, *wait_for_complete=True*)

**Description** Set the servo to go back to center

**Parameters**

- **servo_id** (*uint8*) – Servo number. The range is [1:4]
- **wait_for_complete** (*bool*) – Whether to wait for execution to complete. The default setting is True

**Returns** N/A

**Example** `servo_ctrl.recenter(1, True)`

**Example description** Set servo 1 to go back to center, and wait for execution to complete

`servo_ctrl.`**set_speed**(*servo_id*, *speed*)

**Description** Set the servo rotation speed

**Parameters**

- **servo_id** (*uint8*) – Servo number. The range is [1:4]
- **speed** (*int32*) – Rotation speed, with an accuracy is 1 degree per second. A positive number indicates clockwise rotation, and a negative number indicates counterclockwise rotation

**Returns** None

**Example** `servo_ctrl.set_speed(1, 5)`

**Example description** Set servo 1 to rotate clockwise at 5 degrees per second

# CHAPTER 8

# Smart

vision_ctrl.**marker_detection_color_set**(*color_enum*)

**Description** Set the visual tag recognition color

**Parameters** **color_enum** – A tag color type. For details, see table *color_enum*

**Returns** None

**Example** `vision_ctrl.marker_detection_color_set(rm_define.marker_detection_color_red)`

**Example description** Set the visual tag recognition color to red

**color_enum**

| | |
|---|---|
| rm_define.marker_detection_color_red | Red |
| rm_define.marker_detection_color_green | Green |
| rm_define.marker_detection_color_blue | Blue |

# CHAPTER 9

# Armor plate

**def ir_hit_detection_event(msg):**

**Description** When the robot detects that it is being attacked by infrared beams, programs in the function are run

**Parameters** **msg** – Message parameters in the function

**Returns** None

**Example**

```
#When the robot detects that it is being attacked by infrared beams, programs in the function are run

def ir_hit_detection_event(msg):
    pass
```

armor_ctrl.**cond_wait**(*condition_enum*)

**Description** When the robot is attacked by infrared beams, the next command is executed

**Parameters** **condition_enum** – The rm_define.cond_ir_hit_detection event type indicates that the robot is being attacked by infrared beams

**Returns** None

**Example** armor_ctrl.cond_wait(rm_define.cond_ir_hit_detection)

**Example description** When the robot is attacked by infrared beams, the next command is executed

armor_ctrl.**check_condition**(*condition_enum*)

**Description** Judge whether the robot is being attacked by infrared beams

**Parameters** **condition_enum** – The rm_define.cond_ir_hit_detection event type indicates that the robot is being attacked by infrared beams

**Returns** Whether the robot is attacked by infrared beams or not. It returns true when attacked, otherwise it returns false.

**Return type** bool

**Example**
```
if armor_ctrl.check_condition(rm_define.
    cond_ir_hit_detection):
```

**Example description** If the robot is attacked by infrared beams, the next command is executed

# CHAPTER 10

# Sensor

ir_distance_sensor_ctrl.**enable_measure**(*port_id*)

**Description** Turn on the TOF ranging function

**Parameters** **port_id** (*int*) – TOF module number. The range is [1:4]

**Returns** None

**Example** `ir_distance_sensor_ctrl.enable_measure(1)`

**Example description** Turn on the ranging function of TOF 1

ir_distance_sensor_ctrl.**disable_measure**(*port_id*)

**Description** Turn off the TOF ranging function

**Parameters** **port_id** (*int*) – TOF module number. The range is [1:4]

**Returns** None

**Example** `ir_distance_sensor_ctrl.disable_measure(1)`

**Example description** Turn off the ranging function of TOF 1

ir_distance_sensor_ctrl.**get_distance_info**(*port_id*)

**Description** Obtain the ranging information from a TOF

**Parameters** **port_id** (*int*) – TOF module number. The range is [1:4]

**Returns** Distance of obstacle in front of the TOF, with an accuracy of 1 cm

**Return type** uint16

**Example** `ir_distance_sensor_ctrl.get_distance_info(1)`

**Example description** Obtain the ranging information from TOF 1

**def ir_distance_[port_id]_[compare_type]_[dist]_event(msg):**

**Description** When it is detected that the distance of the obstacle in front of the TOF module satisfies the condition, the program in the function is run

**Parameters**

- **port_id** (*int*) – TOF module number. The range is [1:4]
- **compare_type** – Comparison type. It can be eq, ge, gt, le, and lt (i.e. equal to, greater than or equal to, greater than, less than or equal to, or less than)
- **dist** – The distance used for comparison, with an accuracy of 1 cm, a range of 5-500 cm, and an error rate of 5%

**Returns** None

**Example**

```
#When the distance of the obstacle in front of TOF 1 is detected as less than 10 cm, the program in the function is run

def ir_distance_1_lt_10_event(msg):
    pass
```

ir_distance_sensor_ctrl.**cond_wait**(*'ir_distance_[port_id]_[compare_type]_[dist]'*)

**Description** When the distance of the obstacle in front of the TOF module satisfies the condition, the next command is executed

**Parameters**

- **'ir_distance_[port_id]_[compare_type]_[dist]'** – A string used for distance comparison. It contains the module number, comparison type, and distance
- **port_id** (*int*) – TOF module number. The range is [1:4]
- **compare_type** – Comparison type. It can be eq, ge, gt, le, and lt (i.e. equal to, greater than or equal to, greater than, less than or equal to, or less than)
- **dist** – The distance used for comparison, with an accuracy of 1 cm, a range of 5-500 cm, and an error rate of 5%

**Returns** None

**Example** `ir_distance_sensor_ctrl.cond_wait('ir_distance_1_gt_50')`

**Example description** When the distance of the obstacle in front of TOF 1 is greater than 50 cm, the next command is executed

ir_distance_sensor_ctrl.**check_condition**(*'ir_distance_[port_id]_[compare_type]_[dist]'*)

**Description** Judge whether the distance of the obstacle in front of the TOF module satisfies the condition

**Parameters**

- **'ir_distance_[port_id]_[compare_type]_[dist]'** – A string used for distance comparison. It contains the module number, comparison type, and distance
- **port_id** (*int*) – TOF module number. The range is [1:4]
- **compare_type** – Comparison type. It can be eq, ge, gt, le, and lt (i.e. equal to, greater than or equal to, greater than, less than or equal to, or less than)
- **dist** – The distance used for comparison, with an accuracy of 1 cm, a range of 5-500 cm, and an error rate of 5%

**Returns** Whether it satisfies the condition or not. When it does, it returns true; otherwise, it returns false.

**Return type** bool

**Example**

```
# When the distance of the obstacle in front of TOF 1 is detected as less than 10 cm, the program in the function is run

if ir_distance_sensor_ctrl.check_condition('ir_distance_1_gt_50'):
    pass
```

CHAPTER 11

# Adaptor

sensor_adapter_ctrl.**get_sensor_adapter_adc**(*board_id*, *port_num*)

**Description** Obtain the ADC value of the analog pin of the corresponding port of the sensor adaptor

**Parameters**

- **board_id** (*int*) – Sensor adaptor number. The range is [1:6]
- **port_num** (*uint8*) – The port number on the sensor adaptor. The range is [1:2]
- **wait_for_complete** (*bool*) – Whether to wait for execution to complete. The default setting is True

**Returns** The ADC value of analog pin of the corresponding port of the sensor adaptor. The range is [0:1023]

**Return type** uint16

**Example**
```
ret = sensor_adapter_ctrl.get_sensor_adapter_adc(1, 2)
```

**Example description** Obtain the ADC value of the analog pin of port 2 of sensor adaptor 1

sensor_adapter_ctrl.**get_sensor_adapter_pulse_period**(*board_id*, *port_num*)

**Description** Obtain the pulse duration of the corresponding port pin of the sensor adaptor

**Parameters**

- **board_id** (*int*) – Sensor adaptor number. The range is [1:6]
- **port_num** (*uint8*) – The port number on the sensor adaptor. The range is [1:2]

**Returns** The pulse duration of the corresponding port pin of the sensor adaptor, with an accuracy of 1 ms

**Return type** uint32

**Example**
```
ret = sensor_adapter_ctrl.get_sensor_pulse_period(1, 2)
```

**Example description** Obtain the pulse duration of port 2 pin of sensor adaptor 1

**def sensor_adapter[board_id]_port[port_id]_[judge_type]_event(msg):**

**Description** When it is detected that the corresponding port pin of the sensor adaptor jumps to high level/low level/bidirectional, the program in the function is run

**Parameters**

- **board_id** (*int*) – Sensor adaptor number. The range is [1:6]
- **port_num** (*uint8*) – The port number on the sensor adaptor. The range is [1:2]
- **judge_type** – Trigger condition, which can be high, low, and trigger, indicating high level, low level, or bidirectional jumping, respectively

**Returns** N/A

**Example**

```
#When it is detected that the pin of port 2 of sensor adaptor 1 jumps to a high level,
↪ the program in the function is run

def sensor_adapter1_port2_high_event(msg):
    pass
```

sensor_adapter_ctrl.**cond_wait**(*rm_define.cond_sensor_adapter[board_id]_port[port_id]_[judge_type]_event*)

**Description** When the pulse of the corresponding port pin of the sensor adaptor is high/low/jumping, the next command is executed

**Parameters**

- **board_id** (*int*) – Sensor adaptor number. The range is [1:6]
- **port_num** (*uint8*) – The port number on the sensor adaptor. The range is [1:2]
- **judge_type** – Trigger condition, which can be high, low, and trigger, indicating high level, low level, or bidirectional jumping, respectively

**Returns** N/A

**Example** `sensor_adapter_ctrl.cond_wait(rm_define.cond_sensor_adapter1_port2_high_event)`

**Example description** When the pin of port 2 of sensor adaptor 1 is at high level, the next command is executed

sensor_adapter_ctrl.**check_condition**(*rm_define.cond_sensor_adapter[board_id]_port[port_id]_[judge_type]_event*)

**Description** Judge whether the pulse of the corresponding port pin of the sensor adaptor is high/low/jumping

**Parameters**

- **board_id** (*int*) – Sensor adaptor number. The range is [1:6]
- **port_num** (*uint8*) – The port number on the sensor adaptor. The range is [1:2]
- **judge_type** – Trigger condition, which can be high, low, and trigger, indicating high level, low level, or bidirectional jumping, respectively

**Returns** Whether it satisfies the condition or not. When it does, it returns true; otherwise, it returns false.

**Return type** bool

**Example**

```
#If the port 2 pin of sensor adaptor 1 is jumping, the next command is executed

if sensor_adapter_ctrl.check_condition(rm_define.cond_sensor_adapter1_port2_trigger_
↪event):
    pass
```

# CHAPTER 12

# UART

serial_ctrl.**serial_config**(*baud_rate*, *data_bit*, *odd_even*, *stop_bit*)

**Description** Set the serial port's baud rate, data bit, parity bit, and stop bit properties

**Parameters**

- **baud_rate** – Set the baud rate. The optional baud rates are 9600, 19200, 38400, 57600, and 115200
- **data_bit** – Set the data bit. The optional data bits are cs7 and cs8
- **odd_even_crc** – Set the parity. For details, see table *odd_even_crc*
- **stop_bit** – Set the stop bit. The optional stop bits are 1 and 2

**Returns** None

**Example** `serial_ctrl.serial_config(9600, 'cs8', 'none', '1')`

**Example description** Set the baud rate of the serial port to 9600, the data bit to 8, do not use parity, and set the stop bit to 1

serial_ctrl.**write_line**(*msg_string*)

**Description** Transmit string information by adding line feed `'\n'` automatically

**Parameters** **msg_string** (*string*) – The string information to be transmitted. `'\n'` is added automatically to the end of the string while being transmitted

**Returns** None

**Example** `serial_ctrl.write_line('RoboMaster EP')`

**Example description** Write `'RoboMaster EP\n'` to the serial port. The last line feed will be added automatically. The user only needs to send `'RoboMaster EP'`

serial_ctrl.**write_string**(*msg_string*)

**Description** Transmit string information

**Parameters** **msg_string** (*string*) – String information to be transmitted

**Returns** None

**Example** `serial_ctrl.write_string('RoboMaster EP')`

**Example description** Write `'RoboMaster EP'` to the serial port

`serial_ctrl.`**`writ_numbers`**(*key*, *value*)

**Description** Form the parameters into strings in the form of key value pairs and transmit them through the serial port

**Parameters**

- **`key`** (*`string`*) – Keywords to be transmitted
- **`value`** (*`uint32`*) – Value to be transmitted

**Returns** None

**Example** `serial_ctrl.writ_numbers('x', 12)`

**Example description** Write the string `'x:12'` to the serial port

`serial_ctrl.`**`read_line`**()

**Description** Read strings ending with `'\n'` from the serial port

**Parameters** **`void`** – None

**Returns** The string read through the serial port

**Return type** string

**Example** `recv = serial_ctrl.read_line()`

**Example description** Read a line of strings ending with `'\n'` from the serial port

`serial_ctrl.`**`read_string`**()

**Description** Read strings from the serial port (It is OK if the strings do not end with `'\n'`)

**Parameters** **`void`** – None

**Returns** The string read through the serial port

**Return type** string

**Example** `recv = serial_ctrl.read_line()`

**Example description** Read a string from the serial port

`serial_ctrl.`**`read_until`**(*stop_sig*)

**Description** Read strings from the serial port until the specified end character `'stop_sig'` is matched

**Parameters** **`stop_sig`** – The specified end character. The parameter type is character. The range is `['\n'|'$'|'#'|'.'|':'|';']`

**Returns** The matched string read through the serial port

**Return type** string

**Example** `serial_ctrl.read_until('#')`

**Example description** Read strings from the serial port until `'#'` is matched, and then stop reading

**odd_even_crc**

| | |
|---|---|
| none | Do not use the parity check |
| odd | Use the odd check |
| even | Use the even check |

CHAPTER 13

# Instructions for Using Extension Modules

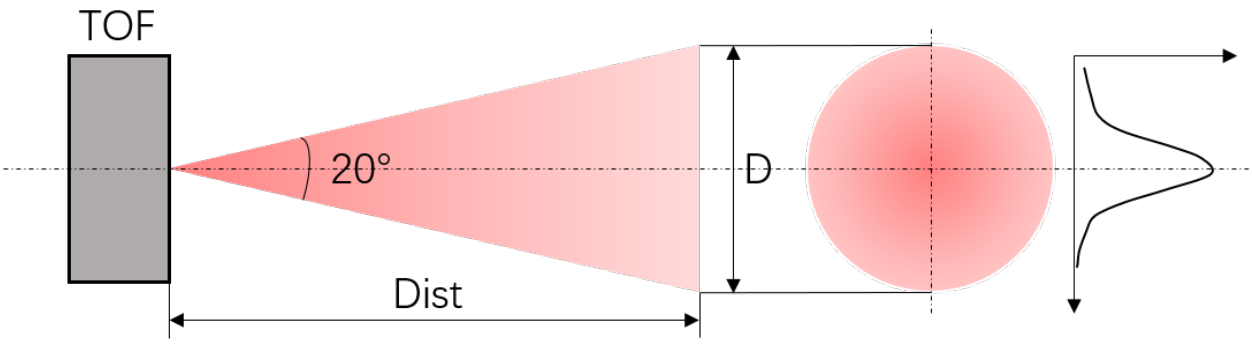
## 13.1 Infrared Distance Sensor (TOF)

1. Introduction

The infrared distance sensor is designed based on the principle of TOF (Time of Flight). In other words, the sensor emits modulated near-infrared light, which is reflected when it encounters an object. The sensor calculates the distance from the object by calculating the time difference or phase difference between the emission and reflection of the light.

2. Product characteristics

The detected area of the TOF is shown in the figure below.




It emits a conical light with an angle of 20°. The relationship between the light spot D and the distance Dist is:

**D=2×Dist×tan(10)**

In order to achieve the best test results, the size of the target should at least equate to the size of the TOF light spot.

**Tip:** If the target is smaller than the light spot size, the target should be as much towards the center of the light spot as possible. This is because the light intensity distribution in the light spot is not uniform, but rather a Gaussian-like

distribution, with strong light in the middle and weak light around. In order to ensure sufficient light energy is returned, the target should be at the center of the light spot as much as possible.

3. Pin description

| No. | Pin | Function | Corresponding connection item |
|---|---|---|---|
| 1 | GND | Power supply | GND |
| 2 | VCC | Power supply | VCC |
| 3 | RX | Receive | TX |
| 4 | TX | Transmit | RX |

4. Communication protocol and data formats

| COM interface | Baud rate | Data bits | Stop bits | Parity check |
|---|---|---|---|---|
| UART | 115200 | 8 | 1 | none |

Control command input:

**`ir_distance_sensor_measure_on`**

**Description** Enable data output of the TOF

**`ir_distance_sensor_measure_off`**

**Description** Disable data output of the TOF

Data output:

**`ir distance:xxx`**

**Description** Data format of the TOF

**Tip:** Command formats are input and output as strings

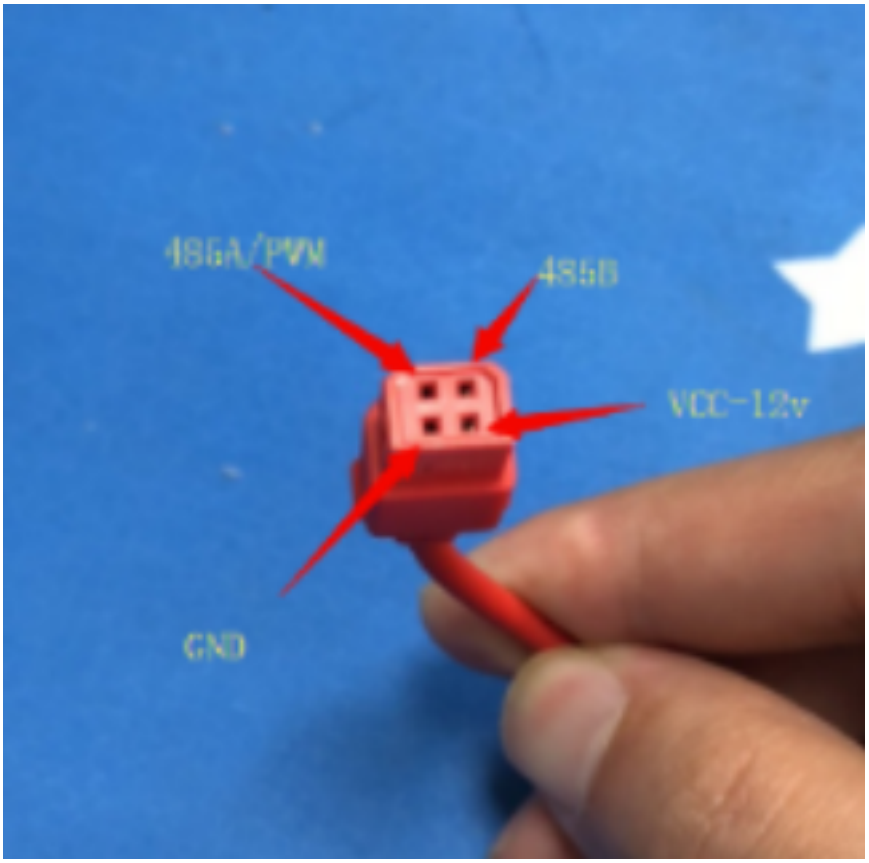
## 13.2 Servo

1. Introduction

In addition to supporting the 485 control, the throttle control mode of the servo can also carry out PWM control. PWM mode only supports angle control

2. Pin description

The 485 pin and PWM pin on the servo are multiplexed, as shown in the figure below




3. Control description

The corresponding input and output of the servo in PWM control mode

| Control mode | Pulse period | Throttle range | Servo output |
|---|---|---|---|
| Angle mode | 50Hz | 2.5%~12.5% | 0°~360° |

CHAPTER 14

# FAQ

CHAPTER 15

# Indices and tables

- genindex
- modindex
- search

# Index

## V