Robomaster Doc Documentation

dji

2020 年 03 月 15 日

快速开始


ROBOMASTER EP

CHAPTER 1

快速上手 RoboMaster SDK 的使用

1.1 介绍

RoboMaster EP 作为一款教育机器人，具有强大的扩展性和可编程性，在编程方面提供了 Scratch 编程，Python 编程以及 SDK，方便用户对 RoboMaster EP 进行二次开发，扩展更加丰富的功能。

下面将使用 **Wi-Fi 直接连接**方式（其他连接模式请参考连接），以完成 **控制发射器发射**功能为例，介绍 SDK 中明文协议的使用。

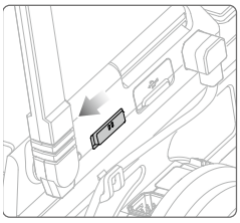
1.2 开发前的准备

1. 准备一台 PC 电脑，需具备 Wi-Fi 功能。
2. PC 上搭建 Python 3.x 环境，安装方式请参考 Python Getting Started

1.3 建立连接

1. 开启电源

   开启机器人电源，切换智能中控的连接模式开关至 **直连模式**



2. 建立 Wi-Fi 连接

   打开电脑的无线网络访问列表，选择位于机身贴纸上对应的 Wi-Fi 名称，输入 8 位密码，选择连接

3. 准备连接脚本

   在完成 Wi-Fi 后，我们还需要编程与机器人建立 TPC/IP 连接，并在对应的端口上传输特定的 **明文协议**，就可以实现相应的控制，更多 **明文协议**请参考协议内容。

   这里我们以 Python 编程语言为例，编写脚本来完成 建立控制连接，接收用户指令，传输明文协议的过程，达到控制机器人的目的。

   参考代码如下

```python
# 测试环境: Python 3.6 版本

import socket
import sys

# 直连模式下，机器人默认 IP 地址为 192.168.2.1，控制命令端口号为 40923
host = "192.168.2.1"
port = 40923

def main():

        address = (host, int(port))

        # 与机器人控制命令端口建立 TCP 连接
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:
```

(下页继续)

(续上页)

```python

                # 等待用户输入控制指令
                msg = input(">>> please input SDK cmd: ")

                # 当用户输入 Q 或 q 时，退出当前程序
                if msg.upper() == 'Q':
                        break

                # 发送控制命令给机器人
                s.send(msg.encode('utf-8'))

                try:
                        # 等待机器人返回执行结果
                        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

        # 关闭端口连接
        s.shutdown(socket.SHUT_WR)
        s.close()

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

4. 将上述代码保存为 rm_sdk.py
5. 运行脚本

   运行 rm_sdk.py 文件 (Windows 系统在安装完成 Python 环境后可直接双击 *.py 文件运行，若无法运行，请按键 `win+r` 并输入 `cmd`，按回车后打开命令运行，键入 `python rm_sdk.py` 运行；Linux 系统请按键 `ctrl+alt+t` 打开命令行键入 `python rm_sdk.py`)

6. 建立 TCP/IP 控制连接

   当运行窗口输出 `Connecting...` 时，代表正在尝试与机器人建立连接，当运行窗口输出 `Connected!` 时，表示已经成功建立控制连接。

1.4 使能 SDK 模式

要进行 SDK 控制，我们需要控制机器人进入 SDK 模式。在上述 Python 运行窗口输入 *command* 命令，按回车键，程序将会发送该命令至机器人，返回 *ok* 即机器人成功进入 SDK 模式:

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

成功进入 SDK 模式后，我们就可以输入控制命令来进行机器人的控制了。

1.5 发送控制命令

续输入 *blaster fire* ，返回 *ok* ，同时，发射器会发射一次:

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

此时，您可以输入其他控制指令来进行机器人控制，更多控制指令请参考协议

1.6 退出 SDK 模式

在完成我们的所有控制指令之后，我们需要退出 SDK 模式，这样我们机器人的其他功能才可以正常使用。

输入 *quit*, 退出 SDK 模式，退出 SDK 模式后无法继续使用 SDK 功能，若要使用，请重新输入 *command* 进入 SDK 模式:

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

1.7 小结

上面我们通过与机器人建立物理连接，与机器人建立 TCP/IP 控制连接，控制机器人进入 SDK 模式，发送控制指令，退出 SDK 模式等几个步骤，实现了通过 SDK 对机器人进行相关的控制功能。您可以通过增加其中 发送控制指令部分的内容，来实现更为复杂的逻辑，完成更为有趣的功能。

其中 Python 编程控制部分，如果您更熟悉其他语言的使用，也可以使用其他语言完成整个控制流程。

如果您手边的设备不支持 Wi-Fi 无法使用 **Wi-Fi 直接连接**，可以参考连接 使用其他连接模式。

以上就是 SDK 快速入门内容，更多使用细节请参见*SDK 文档*，更多示例代码请参见 RoboMaster Sample Code

CHAPTER 2

连接

2.1 连接方式

机器人支持多种连接方式，可通过任意一种连接方式接入使用 SDK 功能

- **直接连接**：
  1. *Wi-Fi 直连*：通过将机器人设置为直连模式，并连接机器人的 Wi-Fi 热点进行接入
  2. *USB 连接*：通过机器人的智能中控上的 USB 端口接入（需支持 RNDIS 功能）
  3. *UART 连接*：通过机器人的运动控制器上的 UART 接口接入
- **组网连接**：
  1. *Wi-Fi 组网*：通过将机器人设置为组网模式，并将计算设备与机器人加入到同一个局域网内，实现组网连接

2.2 连接参数

1. Wi-Fi 直连/Wi-Fi 组网/USB 连接方式请参考以下参数配置：

- **IP 地址说明**：
  - Wi-Fi 直连模式下，机器人默认 IP 为 192.168.2.1
  - Wi-Fi 组网模式下，机器人 IP 由路由器动态分配，可通过监听 *IP 广播数据端口*来获取当前局域网内机器人 IP 地址来进行连接

  - USB 连接模式下，需要计算设备支持 RNDIS 功能，机器人默认 IP 为 192.168.42.2

- **端口及连接方式说明**：

数据	端口号	连接方式	说明
视频流	40921	TCP	需执行开启视频流推送命令，才有数据输出
音频流	40922	TCP	需执行开启音频流推送命令，才有数据输出
控制命令	40923	TCP	可通过当前通道使能 SDK 模式，参见 **SDK 模式控制**
消息推送	40924	UDP	需执行开启消息推送命令，才有数据输出
事件上报	40925	TCP	需执行开启事件上报命令，才有数据输出
IP 广播	40926	UDP	当机器人未与任何设备建立连接时，会有数据输出

2. UART 连接方式请参考以下 UART 参数配置

波特率	数据位	停止位	校验位
115200	8	1	None

> **警告：** UART 连接方式下的数据说明：
>
> UART 连接方式下，仅提供 *控制命令/消息推送/事件上报数据*，如需 *视频流/音频流数据*，请使用 *Wi-Fi/USB* 连接模式

2.3 连接示例

下面我们将以 Python 编程语言为基础，介绍多种连接方式的使用范例。以下所有示例中，默认 PC 上需要集成 Python 3.x 环境（安装方式请参考 Python Getting Started），后面不再赘述。

2.3.1 Wi-Fi 直连

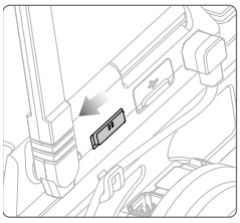
- **环境准备**

1. 准备一台 PC 电脑，需具备 Wi-Fi 功能。

- **建立连接**

1. 开启电源

   开启机器人电源，切换智能中控的连接模式开关至 **直连模式**



2. 建立 Wi-Fi 连接

   打开电脑的无线网络访问列表，选择位于机身贴纸上对应的 Wi-Fi 名称，输入 8 位密码，选择连接

3. 准备连接脚本

   建立 Wi-Fi 连接后，我们还需要编程与机器人建立 TPC/IP 连接机器人开放多个连接端口可供连接，我们首先应完成 **控制命令端口**的连接（直连模式下机器人 IP 地址为 `192.168.2.1`，控制命令端口号: `40923`），以使能机器人 SDK 模式。

   这里我们以 Python 编程语言为例，编写脚本来完成 建立控制连接，使能 *SDK* 模式功能

   参考代码如下

```python
# 测试环境: Python 3.6 版本

import socket
import sys

# 直连模式下，机器人默认 IP 地址为 192.168.2.1，控制命令端口号为 40923
host = "192.168.2.1"
port = 40923

def main():

        address = (host, int(port))

        # 与机器人控制命令端口建立 TCP 连接
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:
```

（下页继续）

(续上页)

```

                # 等待用户输入控制指令
                msg = input(">>> please input SDK cmd: ")

                # 当用户输入 Q 或 q 时，退出当前程序
                if msg.upper() == 'Q':
                        break

                # 发送控制命令给机器人
                s.send(msg.encode('utf-8'))

                try:
                        # 等待机器人返回执行结果
                        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

        # 关闭端口连接
        s.shutdown(socket.SHUT_WR)
        s.close()

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

4. 将上述代码保存为 rm_direct_connection_sdk.py
5. 运行脚本

   Windows 系统在安装完成 Python 环境后可直接双击 *.py 文件运行，若无法运行，请按 `win+r` 并输入 `cmd`，按回车后打开命令运行，键入 `python rm_direct_connection_sdk.py` 运行；

   Linux 系统请按 `ctrl+alt+t` 打开命令行键入 `python rm_direct_connection_sdk.py` 运行

6. 建立 TCP/IP 控制连接

   当运行窗口输出 `Connecting...` 时，代表正在尝试与机器人建立连接，当运行窗口输出 `Connected!` 时，表示已经成功建立控制连接。

- **验证**

在成功建立控制连接后，在命令行里输入 `command`, 机器人返回 `ok`，则表示已经完成连接，并且机器人进入 SDK 模式成功，之后你就可以输入任意控制指令进行机器人控制了。

2.3.2 Wi-Fi/有线网络组网连接

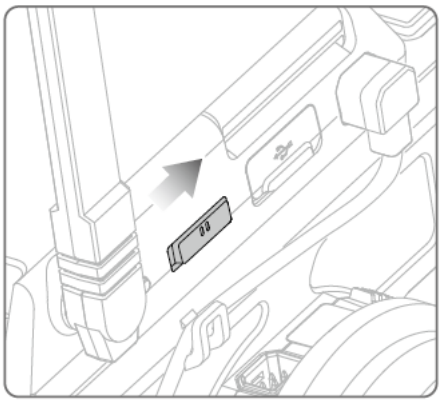
- **环境准备**

1. 准备一台 PC 电脑，具备网络功能（Wi-Fi 或者有线网络皆可）
2. 准备一台家用路由器

- **建立连接**

1. 开启电源

   开启机器人电源，切换智能中控的连接模式开关至 **组网模式**



2. 建立组网连接

   Wi-Fi:

   若使用 Wi-Fi 连接，请将 PC 电脑通过 Wi-Fi 连接至路由器上

   有线网络:

   若使用有线网络连接，请将 PC 电脑通过网线连接至路由器的 LAN 口

   确保 PC 已经接入路由器后，打开 RoboMaster 程序，进入组网连接页面，按下机器人智能中控上的扫码连接按键，扫描二维码进行组网连接，直到连接成功。

3. 获取机器人在局域网内的 IP 地址

   在完成组网连接后，我们的 PC 机已经和机器人处于同一个局域网内，接下来需要编程与机器人建立 TPC/IP 连接，并连接到 **控制命令端口**端口，以使能机器人 SDK 模式。

   若您使用的路由器开启了 DHCP 服务，则机器人的 IP 地址为路由器动态分配，我们需要进一步获取机器人在局域网内的 IP 地址。这里提供两种办法获取：

   1. 若您通过 RoboMaster 程序进行的组网连接，则进入 RoboMaster 程序的 设置-> 连接页面，机器人在局域网内的 IP 地址会在此处显示。
   2. 若您通过其他方式进行的组网连接，则需要通过 监听机器人地址广播来获取机器人在局域网内的 IP 地址，更多细节请参考 **广播**部分。

   参考代码如下

```python
import socket

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

# 绑定 IP 广播端口
ip_sock.bind(('0.0.0.0', 40926))

# 等待接收数据
ip_str = ip_sock.recvfrom(1024)

# 输出数据
print(ip_str)
```

   将上述代码保存为 rm_get_robot_ip.py, 运行上述代码，命令行输出:

```
robot ip 192.168.0.115
```

   我们可以看到，通过 监听机器人地址广播可以获取到机器人在局域网内的 IP 地址为 192.168.0.115

3. 准备连接脚本

我们已经获取到机器人的 IP 地址，这里我们仍以 Python 编程语言为例，编写脚本来完成建立控制连接，使能 *SDK* 模式功能

参考代码如下

```
# 测试环境: Python 3.6 版本

import socket
import sys

# 组网模式下，机器人当前 IP 地址为 192.168.0.115，控制命令端口号为 40923
# 机器人 IP 地址根据实际 IP 进行修改
host = "192.168.0.115"
port = 40923

def main():

        address = (host, int(port))

        # 与机器人控制命令端口建立 TCP 连接
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        print("Connecting...")

        s.connect(address)

        print("Connected!")

        while True:

                # 等待用户输入控制指令
                msg = input(">>> please input SDK cmd: ")

                # 当用户输入 Q 或 q 时，退出当前程序
                if msg.upper() == 'Q':
                        break

                # 发送控制命令给机器人
                s.send(msg.encode('utf-8'))

                try:
                        # 等待机器人返回执行结果
```

(下页继续)

(续上页)

```
                        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

        # 关闭端口连接
        s.shutdown(socket.SHUT_WR)
        s.close()

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

4. 将上述代码保存为 rm_networking_connection_sdk.py
5. 运行脚本

   Windows 系统：在安装完成 Python 环境后可直接双击 *.py 文件运行，若无法运行，请按 `win+r` 并输入 `cmd`，按回车后打开命令运行，键入 `python rm_networking_connection_sdk.py` 运行；

   Linux 系统请按：`ctrl+alt+t` 打开命令行键入 `python rm_networking_connection_sdk.py` 运行

6. 建立 TCP/IP 控制连接

   当运行窗口输出 `Connecting...` 时，代表正在尝试与机器人建立连接，当运行窗口输出 `Connected!` 时，表示已经成功建立控制连接。

- **验证**

在成功建立控制连接后，在命令行里输入 `command`，机器人返回 `ok`，则表示已经完成连接，并且机器人进入 SDK 模式成功，之后你就可以输入任意控制指令进行机器人控制了。

2.3.3 USB 连接

USB 连接模式，实质上是使用 RNDIS 协议，将机器人上的 USB 设备虚拟为一张网卡设备，通过 USB 发起 TCP/IP 连接。更多 RNDIS 内容请参见 RNDIS Wikipedia

- **环境准备**

1. 准备一台具备 RNDIS 功能的 PC 电脑（请确认 PC 电脑上已经配置好 RNDIS 功能）

2. 准备一根 Micro-USB 数据线

- **建立连接**

1. 开启电源

   开启机器人电源，无需关心连接模式开关位置

2. 建立 USB 连接

   将 USB 数据线接入到机器人智能中控上的 USB 口，另一端与电脑相连

3. 测试连接

   打开命令行窗口，运行:

```
ping 192.168.42.2
```

   若命令行输出通信成功，则表示链路正常，可以进行下一步，如:

```
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
```

   若命令行输出 **无法访问…**或者显示超时，则需要检查 PC 上 RNDIS 服务是否配置正常，并重启小车重试，如:

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

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

4 packets transmitted, 0 received, 100% packet loss, time 3071ms

4. 准备连接

   连接过程与*Wi-Fi 直连*-> **准备连接脚本**类似，需要将机器人 IP 地址替换为 USB 模式下的 IP 地址，其余代码与步骤保持不变即可，这里不再赘述

   参考代码变更如下

```python
# 测试环境: Python 3.6 版本

import socket
import sys

# USB 模式下，机器人默认 IP 地址为 192.168.42.2，控制命令端口号为 40923
host = "192.168.42.2"
port = 40923

# other code
```

- **验证**

在成功建立控制连接后，在命令行里输入 `command`, 机器人返回 `ok`，则表示已经完成连接，并且机器人进入 SDK 模式成功，之后你就可以输入任意控制指令进行机器人控制了。

2.3.4 UART 连接

- **环境准备**

1. 一台 PC 电脑，并确定已安装 USB 转串口模块驱动
2. USB 转串口模块
3. 三根杜邦线

- **建立连接**

1. 开启电源

   开启机器人电源，无需关心连接模式开关位置

2. 连接 UART

   将杜邦线插在机器人底盘主控上的 UART 接口上，分别插在 GND, RX, TX 引脚上，另一端对应插在 USB 转串口模块的 GND, TX, RX 引脚

3. 配置 UART，建立通信连接

   这里，我们仍以 Python 编程为例，进行 Windows 系统下 UART 相关配置。

   1. 确认 PC 已识别 USB 转串口模块，并在 **电脑设备管理器**中的 **端口**里确认对应的串口号，如 COM3。
   2. 安装 serial 模块:

      ```
      pip install pyserial
      ```

   3. 编写代码进行 UART 控制，参考代码如下

```python
# 测试环境: Python 3.6 版本
import serial

ser = serial.Serial()

# 配置串口 波特率 115200, 数据位 8 位, 1 个停止位, 无校验位, 超时时间 0.2 秒
ser.port = 'COM3'
ser.baudrate = 115200
ser.bytesize = serial.EIGHTBITS
ser.stopbits = serial.STOPBITS_ONE
ser.parity = serial.PARITY_NONE
ser.timeout = 0.2

# 打开串口
ser.open()

while True:

        # 等待用户输入控制指令
        msg = input(">>> please input SDK cmd: ")

        # 当用户输入 Q 或 q 时, 退出当前程序
        if msg.upper() == 'Q':
                break

        ser.write(msg)

        recv = ser.read()

        print(recv)

# 关闭串口
ser.close()
```

4. 将上述程序保存为 rm_uart.py, 并运行

- **验证**

在成功建立控制连接后，在命令行里输入 `command`, 机器人返回 `ok`，则表示已经完成连接，并且机器人进入 SDK 模式成功，之后你就可以输入任意控制指令进行机器人控制了。

小技巧: 示例代码

更多连接相关示例代码请参考 RoboMaster Sample Code

CHAPTER 3

协议

3.1 协议格式

3.1.1 控制命令

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

- **描述**
  - 控制命令协议格式，一般用来与机器人做控制上的交互
- **参数**
  - *obj* (str): 控制对象字符串
  - *command* (str): 控制命令字符串
  - *params* (str): 命令参数字符串，一般为 **<key> <value>** 形式
  - *seq* (str): 命令序号字符串，一般为 **seq <seq_value>** 形式，可选参数

OUT：**<result>** [**<seq>**]

- **描述**
  - 控制命令响应结果协议格式，一般用来确认控制命令执行结果
  - 不做特殊说明的情况下，所有控制命令均有响应结果
- **参数**

- *result* (exec_result_enum): 执行结果字符串
- *seq* (str): 执行结果序号字符串，一般为 **seq <seq_value>** 形式

注解: <seq>

<seq> 可用来标识当前消息的唯一性，当控制命令中带有 *<seq>* 参数时，对应命令的响应结果结果中即包含对应的序号

3.1.2 消息推送

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

- **描述**
  - 消息推送协议格式，通过控制命令打开某消息推送后即可接收到
  - 消息推送将会以固定的频率进行的推送，推送频率取决于使能当前消息推送时的频率设置
- **参数**
  - *obj* (str): 推送对象
  - *attr* (str): 推送数据属性
  - *value* (str): 推送数据数值

3.1.3 事件上报

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

- **描述**
  - 事件上报协议格式，通过控制命令打开某事件上报开关后即可接收到
- **参数**
  - *obj* (str): 发生事件的对象
  - *attr* (str): 事件数据属性
  - *value (str)* 事件数据数值

注解: 触发机制

当成功使能对应 事件上报功能后，若发生事件，则会进行一次事件上报

3.1.4 IP 广播

OUT: **robot ip <addr>**

- **参数**
  - *addr* (str): 当前连接模式下的机器人的 IPv4 地址

注解: 广播生命周期

当处于 *Wi-Fi* 组网模式下，机器人会不断在对应端口上广播自己的 IPv4 地址，您可通过该 IP 地址连接到机器人，当成功连接后，广播停止

3.1.5 视频流

OUT: H.264 编码实时视频流数据，需要对视频流数据进行正确的解码操作才能实时显示视频画面。

3.1.6 音频流

OUT: Opus 编码实时音频流数据，需要对音频流数据进行正确的解码操作才能实时播放音频。

注解: IN/OUT

该文档中，控制指令中出现的前缀 **IN** 或者 **OUT** 在控制指令中无实际意义，仅为标识以机器人为主体的情况下，当前指令的数据流向：

IN：标识当前数据为从外部设备发送至机器人

OUT：标识当前数据为从机器人发送至外部设备

实际使用中，请忽略该标识，仅发送和接收实际的控制命令即可

3.2 协议内容

3.2.1 SDK 模式控制

进入 SDK 模式

IN：**command**

- **描述**
  - 控制机器人进入 SDK 模式

  - 当机器人成功进入 SDK 模式后，才可以响应其余控制命令

退出 SDK 模式

IN: **quit**

- **描述**
  - 控制机器人退出 SDK 模式，重置所有设置项
  - Wi-Fi/USB 连接模式下，当连接断开时，机器人会自动退出 SDK 模式

3.2.2 机器人控制

机器人运动模式控制

IN：**robot mode <mode>**

- **描述**
  - 设置机器人运动模式
- **参数**
  - *mode* (`mode_enum`): 机器人运动模式
- **示例**
  - *robot mode chassis_lead*：将机器人的运动模式设置为“云台跟随底盘模式”

注解： 机器人运动模式

机器人运动模式描述了云台与底盘之前相互作用与相互运动的关系，每种机器人模式都对应了特定的作用关系。

机器人运动模式分为三种模式：

1. 云台跟随底盘模式：该模式下，云台的 YAW 轴会进入持续跟随底盘 YAW 轴的运动的状态，云台将不响应所有控制指令中 YAW 轴控制的部分，影响指令有云台运动速度控制 云台相对位置控制 云台绝对位置控制
2. 底盘跟随云台模式：该模式下，底盘的 YAW 轴会进入持续跟随云台 YAW 轴的运动的状态，底盘将不响应所有控制指令中 YAW 轴控制的部分，影响指令有底盘运动速度控制 底盘轮子速度控制 底盘相对位置控制
3. 自由模式：该模式下，云台的 YAW 轴与底盘的 YAW 轴运动互不影响

机器人运动模式获取

IN: **robot mode ?**

- **描述**
  - 查询当前机器人运动模式
- **返回值**
  - *mode* (`mode_enum`): 机器人运动模式
- **示例**
  - IN：*robot mode ?*: 查询当前的机器人运动模式
  - OUT: *chassis_lead*: 机器人返回当前的运动模式为 云台跟随底盘模式

> **警告:** 获取指令中的 *?*
>
> 注意：查询指令中的 *?* 与前面命令部分中间存在一个空格

3.2.3 底盘控制

底盘运动速度控制

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

- **描述**
  - 控制底盘运动速度
- **参数**
  - *speed_x* (float:[-3.5,3.5]): x 轴向运动速度，单位 m/s
  - *speed_y* (float:[-3.5,3.5]): y 轴向运动速度，单位 m/s
  - *speed_z* (float:[-600,600]): z 轴向旋转速度，单位 °/s
- **示例**
  - *chassis speed x 0.1 y 0.1 z 1*：底盘 x 轴速度为 0.1 m/s，y 轴速度为 0.1 m/s，z 轴旋转速度为 1°/s

底盘轮子速度控制

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

- **描述**

  - 控制四个轮子的速度

- **参数**
  - *speed_w1* (int:[-1000, 1000]): 右前麦轮速度，单位 rpm
  - *speed_w2* (int:[-1000, 1000]): 左前麦轮速度，单位 rpm
  - *speed_w3* (int:[-1000, 1000]): 右后麦轮速度，单位 rpm
  - *speed_w4* (int:[-1000, 1000]): 左后麦轮速度，单位 rpm

- **示例**
  - *chassis wheel w2 100 w1 12 w3 20 w4 11*：底盘左前麦轮的速度为 100 rpm，右前麦轮速度为 12 rpm，右后麦轮速度为 20 rpm，左后麦轮速度为 11 rpm

底盘相对位置控制

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

- **描述**
  - 控制底盘运动当指定位置，坐标轴原点为当前位置

- **参数**
  - *distance_x* (int:[-5, 5]): x 轴向运动距离，单位 m
  - *distance_y* (int:[-5, 5]): y 轴向运动距离，单位 m
  - *degree_z* (int:[-1800, 1800]): z 轴向旋转角度，单位 °
  - *speed_xy* (int:(0, 3.5]): xy 轴向运动速度，单位 m/s
  - *speed_z* (int:(0, 600]): z 轴向旋转速度，单位 m/s

- **示例**
  - *chassis move x 0.1 y 0.2*：以当前位置为坐标原点，向 x 轴运动 0.1 m，向 y 轴运动 0.2 m

底盘速度获取

IN: **chassis speed ?**

- **描述**
  - 获取底盘速度信息

- **返回值**

  - *<x> <y> <z> <w1> <w2> <w3> <w4>*：x 轴向运动速度 (m/s)，y 轴向运动速度 (m/s)，z 轴向旋转速度 (°/s)，w1 右前麦轮速度 (rpm)，w2 左前麦轮速速 (rpm)，w3 右后麦轮速度 (rpm)，w4 左后麦轮速度 (rpm)

- **示例**
  - IN: *chassis speed ?*：获取底盘的运动速度信息
  - OUT: *1 2 30 100 150 200 250*：底盘当前的 x 轴向运动速度为 1 m/s，y 轴向运动速度 2 m/s，z 轴向旋转速度为 20°/s，1 号轮子转速为 100 rpm，2 号轮子转速为 100 rpm，3 号轮子转速为 100 rpm，4 号轮子转速为 100 rpm

底盘位置获取

IN: **chassis position ?**

- **描述**
  - 获取底盘位置信息
- **返回值**
  - *<x> <y> <z>*：x 轴位置 (m)，y 轴位置 (m)，偏航角度 (°)
- **示例**
  - IN: *chassis position ?*：获取底盘的位置信息
  - OUT: *1 1.5 20*：底盘当前的位置距离上电时刻位置，沿 x 轴运动了 1 m，沿 y 轴运动了 1.5 m，旋转了 20°

底盘姿态获取

IN: **chassis attitude ?**

- **描述**
  - 获取底盘姿态信息
- **返回值**
  - *<pitch> <roll> <yaw>*：pitch 轴角度 (°)，roll 轴角度 (°)，yaw 轴角度 (°)
- **示例**
  - *chassis attitude ?*：查询底盘的姿态信息

底盘状态获取

IN: **chassis status ?**

- **描述**

  - 获取底盘状态信息

- **返回值**
  - ***<static> <uphill> <downhill> <on_slope> <pick_up> <slip> <impact_x> <impact_y> <***
    - *static*：是否静止
    - *uphill*：是否上坡
    - *downhill*：是否下坡
    - *on_slope*：是否溜坡
    - *pick_up*：是否被拿起
    - *slip*：是否滑行
    - *impact_x*：x 轴是否感应到撞击
    - *impact_y*：y 轴是否感应到撞击
    - *impact_z*：z 轴是否感应到撞击
    - *roll_over*：是否翻车
    - *hill_static*：是否在坡上静止

- **示例**
  - IN: *chassis status ?* ：查询底盘的状态
  - OUT: *0 1 0 0 0 0 0 0 0 0 0*：底盘当前处于上坡状态

底盘信息推送控制

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

- **描述**
  - 打开/关闭底盘中相应属性的信息推送
  - **频率设置**
    - **各单独的功能支持单独的频率设置，如：**
      - *chassis push position on pfreq 1 attitude on*：打开位置和姿势推送，位置推送频率为 1 Hz，姿势推送频率使用默认设置 5 Hz
    - **支持当前模块所有功能频率统一设置，如：**
      - chassis push freq 10 #chassis 推送统一为 10 Hz
      - chassis push position pfreq 1 freq 5 # 此时有 freq 参数，将会忽略 pfreq

        * 支持的频率 1, 5, 10, 20, 30, 50

    - 推送数据格式参见底盘推送信息数据

- **参数**

    - *switch* (*switch_enum*)：当此处参数使用 *on* 时，表示打开对应属性的推送；当此处参数使用 *off* 时，表示关闭对应属性的推送
    - *freq* (int:(1,5,10,20,30,50))：对应的属性推送的推送频率
    - *freq_all* (int:(1,5,10,20,30,50))：整个底盘所有相关推送信息的推送频率

- **示例**

    - *chassis push attitude on*：打开底盘姿态信息推送
    - *chassis push attitude on status on*：打开底盘姿态、状态信息推送
    - *chassis push attitude on afreq 1 status on sfreq 5*：打开底盘的姿态信息推送，推送频率为每秒一次，同时打开底盘的状态信息推送，推送频率为每秒五次
    - *chassis push freq 10*：底盘所有信息推送的频率为每秒十次

底盘推送信息数据

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

- **描述**

    - 当用户使能底盘信息推送后，机器人会以设置的频率向用户推送相应信息

- **参数**

    - *attr* (*chassis_push_attr_enum*)：订阅的属性名称
    - ***data*** [订阅的属性数据]

        * 当 *attr* 为 **position** 时，*data* 内容为 *<x> <y>*
        * 当 *attr* 为 **attitude** 时，*data* 内容为 *<pitch> <roll> <yaw>*
        * 当 *attr* 为 **status** 时，*data* 内容为 *<static> <uphill> <downhill> <on_slope> <pick_up> <slip> <impact_x> <impact_y> <impact_z> <roll_over> <hill_static>*

- **示例**

    - *chassis push attitude 0.1 1 3*：当前底盘的 pitch、roll、yaw 姿态信息分别为 0.1、1、3

3.2.4 云台控制

云台运动速度控制

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

- **描述**
  - 控制云台运动速度
- **参数**
  - *p* (float:[-450, 450])：pitch 轴速度，单位 °/s
  - *y* (float:[-450, 450])：yaw 轴速度，单位 °/s
- **示例**
  - *gimbal speed p 1 y 1*：云台的 pitch 轴速度为 1°/s，yaw 轴速度为 1°/s

云台相对位置控制

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

- **描述**
  - 控制云台运动到指定位置，坐标轴原点为当前位置
- **参数**
  - *p* (float:[-55, 55])：pitch 轴角度，单位 °
  - *y* (float:[-55, 55])：yaw 轴角度，单位 °
  - *vp* (float:[0, 540])：pitch 轴运动速速，单位 °/s
  - *vy* (float:[0, 540])：yaw 轴运动速度，单位 °/s
- **示例**
  - *gimbal move p 10*：以当前位置为坐标基准，控制云台运动到 pitch 轴角度为 10° 的状态

云台绝对位置控制

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

- **描述**
  - 控制云台运动到指定位置，坐标轴原点为上电位置
- **参数**
  - *p* (int:[-25, 30])：pitch 轴角度 (°)
  - *y* (int:[-250, 250])：yaw 轴角度 (°)
  - *vp* (int:[0, 540])：pitch 轴运动速速 (°)

  - *vy* (int:[0, 540])： yaw 轴运动速度 (°)

- **示例**
  - *gimbal moveto p 10 y -20 vp 0.1*： 以机器人上电位置为坐标基准，控制云台运动到 pitch 轴角度为 10°，yaw 轴角度为 -20° 的状态，运动时指定 pitch 轴的运动速度为 0.1°/s

云台休眠控制

IN: **gimbal suspend**

- **描述**
  - 控制云台进入休眠状态
- **示例**
  - *gimbal suspend*： 使云台进入休眠状态

云台恢复控制

IN: **gimbal resume**

- **描述**
  - 控制云台从休眠状态中恢复
- **参数**
  - *None*
- **示例**
  - *gimbal resume*： 使云台退出休眠状态

> **警告：** 休眠状态当云台进入休眠状态时，云台两轴电机将会释放控制力，云台整体不响应任何控制指令。要解除云台休眠状态，请参见云台恢复控制

云台回中控制

IN: **gimbal recenter**

- **描述**
  - 云台回中
- **示例**
  - *gimbal recenter*： 控制云台回中

云台姿态获取

IN: **gimbal attitude ?**

- **描述**
  - 获取云台姿态信息
- **返回值**
  - *<pitch> <yaw>*： pitch 轴角度 (°)，yaw 轴角度 (°)
- **示例**
  - IN：*gimbal attitude ?*： 查询云台的角度信息
  - OUT: *-10 20*： 云台当前 pitch 轴角度 -10°，yaw 轴角度 20°

云台信息推送控制

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

- **描述**
  - 打开/关闭云台中相应属性的信息推送,
  - 推送数据格式参见云台推送信息数据
- **参数**
  - *attr* (*gimbal_push_attr_enum*)：订阅的属性名称
  - *switch* (*switch_enum*)： 当此处参数使用 *on* 时，表示打开对应属性的推送；当此处参数使用 *off* 时，表示关闭对应属性的推送
  - *freq_all*：云台所有相关推送信息的推送频率
- **示例**
  - *gimbal push attitude on*： 打开云台的信息推送

云台推送信息数据

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

- **描述**
  - 当用户使能云台信息推送后，机器人会以设置的频率向用户推送相应信息
- **参数**
  - *attr* (*gimbal_push_attr_enum*)：订阅的属性名称
  - ***data*: 订阅的属性数据**

* 当 *attr* 为 **attitude** 时，*data* 内容为 *<pitch> <yaw>*

- **示例**
  - *gimbal push attitude 20 10*： 当前云台的 pitch 角度为 20°，yaw 角度为 10°

3.2.5 发射器控制

发射器单次发射量控制

IN：**blaster bead <num>**

- **描述**
  - 设置发射器单次发射量
- **参数**
  - *num* (int:[1,5])： 发射量
- **示例**
  - *blaster bead 2*： 控制发射器单次发射两发

发射器发射控制

IN: **blaster fire**

- **描述**
  - 控制水弹枪发射一次
- **示例**
  - *blaster fire*： 控制水弹枪发射一次

发射器单次发射量获取

IN: **blaster bead ?**

- **描述**
  - 获取水弹枪单次发射的水弹数
- **返回值**
  - *<num>*： 水弹枪单次发射的水弹数
- **示例**
  - IN: *blaster bead ?*： 查询水弹枪单次发射的水弹数
  - OUT: *3*： 当前水弹枪单次发射水弹数量为 3

3.2.6 装甲板控制

装甲板灵敏度控制

IN: **armor sensitivity <value>**

- **描述**
  - 设置装甲板打击检测灵敏度
- **参数**
  - *value* (int:[1,10])：装甲板灵敏度，数值越大，越容易检测到打击。默认灵敏度值为 5
- **示例**
  - *armor sensitivity 1*：设置装甲板打击检测灵敏度为 1

装甲板灵敏度获取

IN: **armor sensitivity ?**

- **描述**
  - 获取装甲板打击检测灵敏度
- **参数**
  - *<value>*：装甲板灵敏度
- **示例**
  - IN: *armor sensitivity ?*：查询装甲板打击检测灵敏度
  - OUT: *5*：查询装甲板打击检测灵敏度

装甲板事件上报控制

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

- **描述**
  - 控制装甲板检测事件上报
  - 事件上报数据格式参见装甲板事件上报数据
- **参数**
  - *attr* (`armor_event_attr_enum`)：事件属性名称
  - *switch* (`switch_enum`)：事件属性控制开关
- **示例**

- *armor event hit on*：打开装甲板检测事件推送

装甲板事件上报数据

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

- **描述**
  - 当发生装甲板敲击事件时，可以从事件推送端口接收到此消息
- **参数**
  - ***index*** **(int:[1, 6])：当前发生敲击事件的装甲板 ID**
    - 1 前
    - 2 前
    - 3 前
    - 4 前
    - 5 前
    - 6 前
  - ***type*** **(int:[0, 2])：当前敲击事件的种类**
    - 0 水弹攻击
    - 1 撞击
    - 2 手敲击
- **示例**
  - *armor event hit 1 0*：1 号装甲板检测到水弹枪攻击

3.2.7 声音识别控制

声音识别事件上报控制

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

- **描述**
  - 声音识别时间上报控制，开启之后会有相关的事件上报
  - 事件上报数据格式详参见*声音识别事件上报数据*
- **参数**
  - *attr* (*sound_event_attr_enum*)：事件属性名称
  - *switch* (*switch_enum*)：事件属性控制开关

- **示例**
  - *sound event applause on*：打开声音（掌声）识别

声音识别事件上报数据

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

- **描述**
  - 当发生特定声音事件时，可以从事件推送端口接收到此数据
  - 使能该事件请参见声音识别事件上报控制
- **参数**
  - *attr* (*sound_event_attr_enum*): 事件属性名称
  - ***data*：事件属性数据**
    - 当 *attr* 为 `applause` 时，*data* 为 *<count>*，表示短时间内击掌的次数
- **示例**
  - *sound event applause 2*：识别到短时间内有 2 次拍掌

3.2.8 PWM 控制

PWM 输出占空比控制

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

- **描述**
  - PWM 输出占空比设置
- **参数**
  - *port_mask* (hex:0-0xfff)：PWM 拓展口掩码组合，编号为 X 的输出口对应掩码为 **1 << (X-1)**
  - *value* (float:0-100)：PWM 输出占空比，默认输出为 12.5
- **示例**
  - *pwm value 1 50*：控制 1 号 PWM 口的占空比为 50%

PWM 输出频率控制

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

- **描述**

  - PWM 输出频率设置

- **参数**
  - *port_mask* (hex:0-0xffff)：PWM 拓展口掩码组合，编号为 X 的输出口对应掩码为 **1 << (X-1)**
  - *value* (int:XXX)：PWM 输出频率值

- **示例**
  - *pwm freq 1 1000*：控制 1 号 PWM 口的频率为 1000 Hz

3.2.9 LED 控制

LED 灯效控制

IN：**led control comp <comp_str> r <r_value> g <g_value> b <value> effect <effect_str>**

- **描述**
  - 机器人 LED 灯效控制接口，可设置多种效果
  - 跑马灯效果仅可作用于云台两侧 LED

- **参数**
  - *comp_str* (`led_comp_enum`)：LED 编号
  - *r_value* (int:[0, 255])：RGB 红色分量值
  - *g_value* (int:[0, 255])：RGB 绿色分量值
  - *b_value* (int:[0, 255])：RGB 蓝色分量值
  - *effect_str* (`led_effect_enum`)：LED 灯效类型

- **示例**
  - led control comp all r 255 g 0 b 0 solid：机器人所有 LED 常亮为红色

3.2.10 传感器转接板控制

传感器转接板 ADC 值获取

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

- **描述**
  - 获取传感器转接板的 ADC 数值

- **参数**
  - *adapter_id* (int:[1, 6])：转接板的 ID 号

  - *port_num* (int:[1, 2])： port 的编号

- **返回值**
  - *adc_value*： 测量得到相应转接板上指定端口的电压值，电压取值范围 [0V, 3,3V]

- **示例**
  - IN: *sensor_adapter adc id 1 port 1 ?*：查询 1 号转接板上 1 号端口的 ADC 数值
  - OUT: *1.1*： 当前查询端口 ADC 值为 1.1

传感器转接板 IO 值获取

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

- **描述**
  - 获取传感器转接板 IO 口的逻辑电平

- **参数**
  - *adapter_id* (int:[1, 6])： 转接板的 ID 号
  - *port_num* (int:[1, 2])： port 的编号

- **返回值**
  - *io_level_value*： 测量得到相应转接板上指定端口的逻辑电平值，0 或 1

- **示例**
  - IN: *sensor_adapter io_level id 1 port 1 ?*： 查询 1 号转接板上 1 号端口的 IO 逻辑电平
  - OUT: *1*： 当前查询端口的 IO 值为 1

传感器转接板 IO 引脚电平跳变时间值获取

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

- **描述**
  - 获取传感器转接板 IO 口电平跳变持续时间

- **参数**
  - *adapter_id* (int:[1, 6])：转接板的 ID 号
  - *port_num* (int:[1, 2])：port 的编号

- **返回值**
  - *pulse_period_value*: 测量得到相应转接板上指定端口的电平跳变持续时间值，单位 ms

- **示例**

- *sensor_adapter pulse_period id 1 port 1*：查询 1 号转接板上 1 号端口的电平跳变持续时间

传感器转接板事件上报控制

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

- **描述**
  - 打开/关闭传感器转接板电平跳变事件推送，打开后当 IO 上电平跳变时推送消息，见下一章中 [传感器转接板电平跳变事件推送](# 传感器转接板电平跳变推送) 的介绍
- **参数**
  - *switch* (`switch_enum`)：电平跳变事件上报的控制开关
- **示例**
  - *sensor_adapter event io_level on*：打开传感器转接板的电平跳变事件推送、

传感器转接板事件上报控制

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

- **描述**
  - 当传感器转接板发生电平跳变时推送，可以从事件推送端口接收到此消息
  - 需要打开传感器转接板电平跳变推送，参见传感器转接板事件上报数据
- **参数**
  - *id*：传感器转接板的 ID
  - *port_num*：IO 的 ID
  - *io_level*：当前的逻辑电平值
- **示例**
  - *sensor_adapter event io_level (1, 1, 0)*：当前 1 号转接板的 1 号 IO 的逻辑电平跳变为 0

传感器转接板事件上报数据

3.2.11 红外深度传感器控制

红外深度传感器开关控制

IN: **ir_distance_sensor measure <switch>**

- **描述**
  - 打开/关闭所有红外传感器开关

- **参数**
  - *switch* (*switch_enum*)：红外传感器的开关
- **示例**
  - *ir_distance_sensor measure on*：打开所有红外深度传感器

红外深度传感器距离获取

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

- **描述**
  - 获取指定 ID 的红外深度传感器距离
- **参数**
  - *id* (int:[1, 4])：红外传感器的 ID
- **返回值**
  - *distance_value*：指定 ID 的红外传感器测得的距离值，单位 mm
- **示例**
  - IN: *ir_distance_sensor distance 1*：查询 1 号红外深度传感器测得的距离值
  - OUT: *1000*：当前查询红外深度传感器距离值为 1000 mm

3.2.12 舵机控制

舵机角度控制

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

- **描述**
  - 设置舵机角度
- **参数**
  - *servo_id* (int:[1, 3])：舵机的 ID
  - *angle_value* (float:[-180, 180])：指定的角度，单位 °
- **示例**
  - *servo angle id 1 angle 20*：控制 1 号舵机的角度为 20°

舵机速度控制

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

- **描述**
  - 设置指定舵机的速度
- **参数**
  - *servo_id* (int:[1, 3])：舵机的 ID
  - *speed_value* (float:[-1800, 1800])：设置的速度值，单位 °/s
- **示例**
  - *servo speed id 1 speed 20*：设置 1 号舵机的速度为 10°/s

舵机停止控制

IN: **servo stop**

- **描述**
  - 停止舵机运动
- **示例**
  - *servo stop*：控制舵机停止运动

舵机角度查询

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

- **描述**
  - 获取指定舵机的角度
- **参数**
  - *servo_id* (int:[1, 3])：舵机的 ID
- **返回值**
  - *angle_value*：指定舵机的角度值
- **示例**
  - IN: *servo angle id 1 ?*：获取 1 号舵机的角度值
  - OUT: *30*：当前查询舵机角度值为 30°

3.2.13 机械臂控制

机械臂相对位置运动控制

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

- **描述**
  - 控制机械臂运动一段距离，当前位置为坐标原点
- **参数**
  - *x_dist* (float:[])：x 轴运动距离，单位 cm
  - *y_dist* (float:[])：y 轴运动距离，单位 cm
- **示例**
  - *robotic_arm move x 5 y 5*：控制机械臂在 x 轴运动 5 cm，在 y 轴运动 5 cm

机械臂绝对位置运动控制

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

- **描述**
  - 控制机械臂运动到某位置，机器人上电位置为坐标原点
- **参数**
  - *x_pos* (float:[])：x 轴运动到的坐标，单位 cm
  - *y_pos* (float:[])：y 轴运动到的坐标，单位 cm
- **示例**
  - *robotic_arm moveto x 5 y 5*：控制机械臂 x 轴运动到 5 cm 的坐标位置，y 轴运动到 5 cm 的坐标位置

机械臂回中控制

IN: **robotic_arm recenter**

- **描述**
  - 控制机械臂回中
- **参数**
  - *None*
- **示例**
  - *robotic_arm recenter*：控制机械臂回中

机械臂停止运动控制

IN: **robotic_arm stop**

- **描述**
  - 停止机械臂运动
- **参数**
  - *None*
- **示例**
  - *robotic_arm stop*：停止机械臂运动

机械臂绝对位置查询

IN: **robotic_arm position ?**

- **描述**
  - 获取机械臂的位置
- **参数**
  - *None*
- **返回值**
  - ***<x_pos> <y_pos>*: 机械臂的位置坐标**
    - *x_pos*：x 轴的坐标，单位 cm
    - *y_pos*：y 轴的坐标，单位 cm
- **示例**
  - IN: *robotic_arm position ?*：查询机械臂的位置
  - OUT：*50 60*：当前查询机械臂的位置距离标定点 x 轴距离为 50 cm, y 轴距离为 60 cm

3.2.14 机械爪控制

机械爪张开运动控制

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

- **描述**
  - 张开机械爪
- **参数**

  - *level_num* (int:[1,4])：机械爪张开的力度等级，取值范围 [1,4]

- **示例**
  - *robotic_gripper open 1*：控制机械臂以力度 1 打开

机械爪关闭运动控制

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

- **描述**
  - 闭合机械爪
- **参数**
  - *level_num* (int:[1,4])：机械爪闭合的力度等级，取值范围 [1,4]
- **示例**
  - *robotic_gripper close 1*：控制机械臂以力度 1 关闭

注解： 机械爪控制力度

机械爪控制力度描述了机械爪在运动过程中的运动速度以及在堵转状态下最大夹取力度

力度越大，运动速度越快，夹取力越大；反之。

机械臂相对位置运动控制

IN: **robotic_gripper status ?**

- **描述**
  - 获取机械爪开合状态
- **参数**
  - *None*
- **返回值**
  - ***status*** [机械爪当前的开合状态] > 0 机械爪完全闭合 > 1 机械爪既没有完全闭合，也没有完全张开 > 2 机械爪完全张开
- **示例**
  - IN: *robotic_gripper status ?*：获取机械爪的开合状态
  - OUT: *2*：当前查询的机械爪状态为张开

3.2.15 视频流控制

视频流开启控制

IN: **stream on**

- **描述**
  - 打开视频流
  - 打开后，可从视频流端口接收到 H.264 编码的码流数据
- **示例**
  - *stream on*： 打开视频流

视频流关闭控制

IN: **stream off**

- **描述**
  - 关闭视频流
  - 关闭视频流后，H.264 编码的码流数据将会停止输出
- **示例**
  - *stream off*： 关闭视频流

3.2.16 音频流控制

音频流开启控制

IN: **audio on**

- **描述**
  - 打开音频流
  - 关闭音频流后，可以从音频流端口接收到 Opus 编码的音频流数据
- **示例**
  - *audio on*： 打开音频流

音频流关闭控制

IN: **audio off**

- **描述**

  - 关闭音频流
  - 关闭音频流后，Opus 编码的音频流数据将会停止输出

- **示例**
  - *audio off* ： 关闭音频流

3.2.17 IP 广播

OUT: **robot ip <ip_addr>**

- **描述**
  - 当未与机器人建立连接时，可以从 IP 广播端口接收到此消息，连接成功后，该消息停止广播
  - 描述当前机器人的 IP 地址，适用于与机器人在同一局域网内，但未知机器人 IP 信息的情况

- **参数**
  - *ip_addr*：机器人当前 IP 地址

- **示例**
  - *robot ip 192.168.1.102*：机器人当前的 IP 地址为 192.168.1.102

3.3 数据说明

`switch_enum`

- `on`：打开
- `off`：关闭

`mode_enum`

- `chassis_lead`：云台跟随底盘模式
- `gimbal_lead`：底盘跟随云台模式
- `free`：自由模式

`chassis_push_attr_enum`

- `position`：底盘位置
- `attitude`：底盘姿态
- `status`：底盘状态

`gimbal_push_attr_enum`

- `attitude` 云台姿态

`armor_event_attr_enum`

- `hit`：装甲被敲击

`sound_event_attr_enum`

- `applause`：掌声

`led_comp_enum`

- `all`：所有 LED 灯
- `top_all`：云台所有 LED 灯
- `top_right`：云台右侧 LED 灯
- `top_left`：云台左侧 LED 灯
- `bottom_all`：底盘所有 LED 灯
- `bottom_front`：底盘前侧 LED 灯
- `bottom_back`：所有后侧 LED 灯
- `bottom_left`：所有左侧 LED 灯
- `bottom_right`：所有右侧 LED 灯

`led_effect_enum`

- `solid`：常亮效果
- `off`：熄灭效果
- `pulse`：呼吸效果
- `blink`：闪烁效果
- `scrolling`：跑马灯

CHAPTER 4

多机通信

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

描述 设置机器的组号为 `send_group` ，机器可以接收来自 `recv_group_list` 中注册的组号的消息。如果不使用 `recv_group_list` 参数，默认接收组号 0 的消息

参数

- **`send_group`** (*int*) – 当前机器的发送组号，默认组号为 0
- **`recv_group_list`** (*list/tuple*) – 当前接收消息的组别列表，类型可以为列表或元组

返回 无

示例 `multi_comm_ctrl.set_group(1, (1,2,3))`

示例说明 设置当前发送组号为 1，接收组号 1,2,3 的消息，若接收组别包含发送组别，则会接收到自己发送的消息

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

描述 通过多机通信发送消息，可以单独设置该消息的发送组号

参数

- **`msg`** (*int*) – 需要发送的消息
- **`group`** (*int*) – 可选参数，指定当前消息发送组号，不指定则默认使用之前设置的组号

返回 无

示例 multi_comm_ctrl.send('RoboMaster EP', 3)

示例说明 向组号 3 发送消息 'RoboMaster EP'

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

描述 接收消息（当没有注册 'recv_callback'时生效），可设置超时时间

参数 **timeout** (*int*) – 等待时间，接收函数等待的时间，精确度为 1 秒，默认为 72 秒

返回 <msg_group>，<msg> 消息发送方的组号和消息内容

示例 group，recv_msg = multi_comm_ctrl.recv_msg(30)

示例说明 接收消息，等待时间为 30 秒，group 为信息发送方的组号，msg 为收到的消息内容

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

描述 注册接收消息的回调函数，当接收到信息后，自动执行回调函数

参数 **callback** (*function*) – 需要注册的回调函数，回调函数原型为 def callback(msg)，其中 msg 参数类型为元组（msg_group，msg）

返回 无

示例

```
# 定义一个函数，并将其注册为接收消息的回调函数

def recv_callback(msg):
    pass

multi_comm_ctrl.register_recv_callback(recv_callback)
```

CHAPTER 5

自定义 UI 系统

5.1 简介

自定义 UI 系统是用户通过自己编写的程序生成自定义的 UI 控件来拓展程序的输入和输出的一种方式。

我们编程时很重要的一部分工作是处理输入和输出，对我们的机器人来说，程序输出可以是底盘、云台、水弹枪等模块的动作，也可以是灯光、音效等的表现，输入的途径则有初始的变量，机器人的视觉识别、掌声识别、装甲板打击检测，手机陀螺仪等。现在我们可以通过自定义 UI 系统与生成的的 UI 控件进行交互达到输入的目的，也可以将程序的处理结果通过 UI 控件来进行信息的输出。

我们可以在 RoboMaster app 中编写 Python 程序，调用自定义 UI 系统的相关接口，来生成 UI 控件，绑定控件的事件回调。在实验室中完成程序的编写和调试后，可以将程序装配成自定义技能，在单机驾驶或者多人竞技中释放出来。

5.2 接口

5.2.1 Common

本部分方法适用于除 Stage 外所有自定义 UI 控件，因此单独拿出来介绍。

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

描述 控制当前控件是否显示

参数 `status` (*bool*) – 控件的活动状态，True 表示显示当前控件，False 表示隐藏当前控件

返回 无

示例 `my_Slider.set_active(Flase)`

示例说明 设置 my_Slider 控件为隐藏状态

`common_object.get_active()`

描述 获取当前控件的显示状态

参数 `void` – 无

返回 bool, 表示控件的显示状态

示例 `status = my_Slider.get_active()`

示例说明 获取 my_Slider 控件的显示状态，赋值给 status 变量

`common_object.set_name(name)`

描述 设置当前控件的名字

参数 `name` (*string*) – 控件的名字

返回 无

示例 `my_Dropdown.set_name('my_dropdown')`

示例说明 设置 my_Dropdown 控件的名字为『my_dropdown』

`common_object.get_name()`

描述 获取当前控件的名字

参数 `void` – 无

返回 string，表示控件的名字

示例 `name = my_Dropdown.get_name()`

示例说明 获取 my_Dropdown 控件的名字，赋值给 name 变量

`common_object.set_position(x, y)`

描述 设置控件的位置坐标，原点在屏幕的中心位置

参数

- `x` (*int*) – 控件的横坐标，取值为屏幕上实际像素的位置，0 点在屏幕水平中心位置，向右为正方向
- `y` (*int*) – 控件的纵坐标，取值为屏幕上实际像素的位置，0 点在屏幕垂直中心位置，向上为正方向

返回 无

示例 `my_Text.set_position(-200, 500)`

示例说明 设置 my_Text 控件的坐标为 (-200，500)

`common_object.get_position()`

描述 获取控件的位置坐标

参数 `void` – 无

返回 [x,y]，表示控件的位置

示例 `pos = my_Text.get_position()`

示例说明 获取 my_Text 控件的位置，赋值给变量 pos，pos 为一个列表

`common_object.set_size(w, h)`

描述 设置控件的大小

参数

- `w` (*int*) – 控件的宽度
- `h` (*int*) – 控件的高度

返回 无

示例 `my_Button.set_size(300, 200)`

示例说明 设置 my_Button 控件的宽度为 300，高度为 200

`common_object.get_size()`

描述 获取控件的大小

参数 `void` – 无

返回 [w,h], 表示控件的大小

示例 `size = my_Button.get_size()`

示例说明 获取 my_Button 控件的大小，赋值给变量 size，size 为一个列表

`common_object.set_rotation(degree)`

描述 设置控件的旋转角度

参数 `degree` (*int*) – 控件的旋转角度，范围为 [0, 360]，正值为顺时针旋转，负值为逆时针旋转

返回 无

示例 `my_Button.set_rotation(90)`

示例说明 设置 my_Button 控件顺时针旋转 90 度

`common_object.get_rotation()`

描述 获取控件的旋转角度

参数 `void` – 无

返回 int, 表示控件的旋转角度，范围为 [0, 360]，正值为顺时针旋转，负值为逆时针旋转

示例 `degree = my_Button.get_rotation()`

示例说明 获取 my_Button 控件的旋转角度，赋值给变量 degree

`common_object.set_privot(x, y)`

描述 设置控件的锚点坐标，输入参数是归一化参数，原点位于控件的左下角，控件的锚点默认为控件中心即 (0.5,0.5)，控件的位置和旋转均以锚点作为控制点

参数

- **x** (*int*) – 锚点的 x 坐标，范围为 [0, 1]，向右为正方向
- **y** (*int*) – 锚点的 y 坐标，范围为 [0, 1]，向上为正方向

返回 无

示例 `my_Button.set_privot(0, 1)`

示例说明 设置控件的锚点为控件的左上角

`common_object.get_privot()`

描述 获取控件的锚点坐标

参数 **void** – 无

返回 [x,y]，表示控件的锚点坐标

示例 `privot = my_Button.get_privot()`

示例说明 获取控件的锚点坐标，赋值给变量 privot，privot 为一个列表

`common_object.set_order(order)`

描述 设置控件的显示优先级，当多个控件重叠时，优先级高的控件在上层，数字越大优先级越高

参数 **order** (*int*) – 控件的指定优先级，控件重叠时优先级高的优先显示

返回 无

示例 `my_Button.set_order(8)`

示例说明 将控件的显示优先级设置为 8，当控件重叠时，低于此优先级的控件将被覆盖

`common_object.get_order()`

描述 获取控件的显示优先级

参数 **void** – 无

返回 int，表示控件的显示优先级

示例 `order = my_Button.get_order()`

示例说明 获取 my_Button 控件的显示优先级，赋值给变量 order

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

描述 注册控件事件触发的回调函数，当控件检测到相应的事件后，执行注册的回调函数

参数

- **event** (*string*) – 指定回调函数的触发事件

  各控件可注册的事件如下：

  - **Button 控件：**
    - on_click 一次按下松开按钮的过程，在松开按钮的时候触发该事件
    - on_press_down 按下按钮的时候触发该事件
    - on_press_up 松开按钮的时候触发该事件
  - **Toggle 控件：**
    - on_value_changed 值发生改变的时候触发该事件，回调函数中的 args 参数为 bool，表示该 Toggle 控件值发生改变后的值
  - **Dropdown 控件：**
    - on_value_changed 值发生改变的时候触发该事件，回调函数中的 args 参数为 int，表示该 Dropdown 控件值发生改变后的选中索引
  - **Text 控件：**
    - 无触发事件
  - **InputField 控件：**
    - on_value_changed 值发生改变的时候触发该事件，回调函数中的 args 参数为 string，表示该 InputField 控件值发生改变后的值

- **callback** (*function*) – 需要注册的回调函数，回调函数的统一签名为：`def callback(widget,*args,**kw):` ，其中 widget 为触发事件的控件引用，args，kw 为参数.

返回 无

示例 1

```
# 当 my_Button 控件被点击后，打印信息到控制台上，机器人会开枪射击一次

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)
```

示例 2

```
# 当 my_Toggle 控件被点击后，值会发生改变，打印信息到控制台上，机器人会播放声音

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)
```

示例 3

```
# 当点击 my_Dropdown 控件改变其选中的值，值会发生改变，打印信息到控制台上，机器人会播放声音

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

示例 4

```
# 当点击 my_InputField 控件改变其选中的值，值会发生改变，打印信息到控制台上

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

系统初始化时会自动创建一个 Stage 类的对象 stage ，直接使用即可，不需要用户自己创建。

`object.add_widget(widget_obj)`

描述 将参数中的控件添加到 UI 界面中

参数 `widget_obj` (*object*) – 需要添加进 UI 界面的控件对象

返回 无

示例

```
# 创建一个 Button 对象，并将其添加进 UI 界面

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

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

描述 从 UI 界面移除参数传入的控件

参数 **widget_obj** (*object*) – 需要从 UI 界面移除的控件

返回 无

示例 stage.remove_widget(my_button)

示例说明 从 UI 界面中移除控件 my_button

5.2.3 Button

Button 控件用于响应来自用户的点击来启动或确认操作。

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

描述 设置按钮对象的文字属性

参数

- **content** (*string*) – 按钮上显示的字符串内容
- **[color_r, color_g, color_b, color_a]** (*list*) – 可选参数，需要显示的字符串的颜色，参数分别为显示颜色 r 值、b 值，g 值，透明度，取值范围都为 [0, 255]
- **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*
- **size** (*int*) – 显示文字的字号大小

返回 无

示例 my_Button.set_text(120, 120, 120, 255, text_anchor.upper_left, 12)

示例说明 设置文字颜色的 rgb 值为（120, 120, 120），透明度为 255，文字对齐方式为顶端左对齐，字号大小为 12 号

button_object.**set_text_color**(*r*, *g*, *b*, *a*)

描述 设置文字的颜色

参数

- **r** (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 文字颜色的 b 值，范围为 [0, 255]

- **a** (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_button.set_text_color(120, 120, 120, 200)`

示例说明 设置文字颜色的 rgb 值为（120, 120, 120），透明度为 200

`button_object.set_text_align(align)`

描述 设置文字的对齐方式

参数 **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详见表格*align*

返回 无

示例 `my_button.set_text_align(text_anchor.upper_left)`

示例说明 设置文字的对齐方式为顶端左对齐

`button_object.set_text_size(size)`

描述 设置文字的字号大小

参数 **size** (*int*) – 文字的字号值

返回 无

示例 `my_button.set_text_size(12)`

示例说明 设置文字的字号为 12 号

`button_object.set_background_color(r, g, b, a)`

描述 设置按钮的背景色

参数

- **r** (*int*) – 字体颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 字体颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 字体颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 字体颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_button.set_background_color(200, 200, 200, 230)`

示例说明 设置背景色的 rgb 值为 (200, 200, 200)，透明度为 230

5.2.4 Toggle

Toggle 控件用于在屏幕上绘制一个开关，通过控制开关的开启与闭合来执行一些具体的操作。

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

描述 设置控件的显示文字

参数

- **string** (*string*) – 控件上显示的字符串内容
- **[color_r, color_g, color_b, color_a]** (*list*) – 可选参数，需要显示的字符串的颜色，参数分别为显示颜色 r 值、b 值、g 值、透明度，取值范围都为 [0, 255]
- **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*
- **size** (*int*) – 显示文字的字号大小

返回 无

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

示例说明 设置文字的 rgb 值为（120, 120, 120），透明度为 200，字体对齐方式为顶端左对齐，字号大小为 12 号

`toggle_object.set_text_color(r, g, b, a)`

描述 设置文字的颜色

参数

- **r** (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 文字颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Toggle.set_text_color(120, 120, 120, 200)`

示例说明 设置字体的 rgb 值为（120, 120, 120），透明度为 200

`toggle_object.set_text_align(align)`

描述 设置文字的对齐方式

参数 **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*

返回 无

示例 `my_Toggle.set_text_align(text_anchor.upper_left)`

示例说明 设置字体的对齐方式为顶端左对齐

`toggle_object.set_text_size(size)`

描述 设置文字的字号大小

参数 **size** (*int*) – 文字的字号值

返回 无

示例 `my_Toggle.set_text_size(12)`

示例说明 设置文字的字号为 12 号

`toggle_object.set_background_color(r, g, b, a)`

描述 设置控件的背景色

参数

- `r` (*int*) – 背景颜色的 r 值，范围为 [0, 255]
- `g` (*int*) – 背景颜色的 g 值，范围为 [0, 255]
- `b` (*int*) – 背景颜色的 b 值，范围为 [0, 255]
- `a` (*int*) – 背景颜色的透明度，[0, 255]

返回 无

示例 `my_Toggle.set_background_color(200, 200, 200, 230)`

示例说明 设置背景色的 rgb 值为 (200, 200, 200)，透明度为 230

`toggle_object.set_checkmark_color(r, g, b, a)`

描述 设置控件选中图标的颜色

参数

- `r` (*int*) – 图标颜色的 r 值，范围为 [0, 255]
- `g` (*int*) – 图标颜色的 g 值，范围为 [0, 255]
- `b` (*int*) – 图标颜色的 b 值，范围为 [0, 255]
- `a` (*int*) – 图标颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Toggle.set_checkmark_color(200, 200, 200, 230)`

示例说明 设置选中图标的 rgb 值为 (200, 200, 200)，透明度为 230

`toggle_object.set_is_on(status)`

描述 设置控件的状态

参数 `status` (*bool*) – 设置控件是否为打开状态，True 表示打开，False 表示关闭

返回 无

示例 `my_Toggle.set_is_on(True)`

示例说明 设置 Toggle 控件为打开状态

5.2.5 Text

Text 控件用于显示文本

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

描述 设置控件的文字属性

参数

- **string** (*string*) – 需要显示的字符串内容
- **[color_r, color_g, color_b, color_a]** (*list*) – 可选参数，需要显示的字符串的颜色，参数分别为显示颜色 r 值、b 值、g 值，透明度，取值范围都为 [0, 255]
- **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*
- **size** (*int*) – 显示文字的字号大小

返回 无

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

示例说明 设置文字颜色的 rgb 值为（120, 120, 120），透明度为 200，字体对齐方式为顶端左对齐，字号大小为 12 号

`text_object.set_text_color(r, g, b, a)`

描述 设置控件的文字颜色

参数

- **r** (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 文字颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Text.set_text_color(120, 120, 120, 200)`

示例说明 设置文字的 rgb 值为（120, 120, 120），透明度为 200

`text_object.set_text_align(align)`

描述 设置文字的对齐方式

参数 **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*

返回 无

示例 `my_Text.set_text_align(text_anchor.upper_left)`

示例说明 设置文字的对齐方式为顶端左对齐

`text_object.set_text_size(size)`

描述 设置文字的字号大小

参数 `size` (*int*) – 文字的字号值

返回 无

示例 `my_Text.set_text_size(12)`

示例说明 设置文字的字号为 12 号

`text_object.set_border_active(active)`

描述 是否显示文字边框

参数 `active` (*bool*) – 是否显示文字边框，True 表示显示边框，False 表示不显示边框

返回 无

示例 `my_Text.set_border_active(True)`

示例说明 显示文字边框

`text_object.set_background_color(r, g, b, a)`

描述 设置控件的背景色

参数

- `r` (*int*) – 背景颜色的 r 值，范围为 [0, 255]
- `g` (*int*) – 背景颜色的 g 值，范围为 [0, 255]
- `b` (*int*) – 背景颜色的 b 值，范围为 [0, 255]
- `a` (*int*) – 背景颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Text.set_background_color(200, 200, 200, 230)`

示例说明 设置背景色的 rgb 值为 (200, 200, 200)，透明度为 230

`text_object.set_background_active(active)`

描述 是否显示文字背景

参数 `active` (*bool*) – 是否显示背景，True 表示显示背景，False 表示不显示背景

返回 无

示例 `my_Text.set_background_active(True)`

示例说明 显示文字背景

`text_object.append_text(content)`

描述 向 Text 控件中增加文本

参数 `content` (*string*) – 需要向 Text 中增加的文本

返回 无

示例 `my_Text.append_text('RoboMaster EP')`

示例说明 向 Text 中增加的文字 `RoboMaster EP`

`align`

text_anchor.upper_left	顶端左对齐
text_anchor.upper_center	顶端居中对齐
text_anchor.upper_right	顶端右对齐
text_anchor.middle_left	中间左对齐
text_anchor.middle_center	中间居中对齐
text_anchor.middle_right	中间右对齐
text_anchor.lower_left	底端左对齐
text_anchor.lower_center	底端居中对齐
text_anchor.lower_right	底端右对齐

5.2.6 InputField

InputField 控件用于接收用户输入的文本信息

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

描述 设置输入框对象中的的文字属性

参数

- `string` (*string*) – 需要显示的字符串内容
- `[color_r, color_g, color_b, color_a]` (*list*) – 可选参数，需要显示的字符串的颜色，参数分别为显示颜色 r 值、b 值，g 值，透明度，取值范围都为 [0, 255]
- `align` (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*
- `size` (*int*) – 显示文字的字号大小

返回 无

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

示例说明 设置字体的 rgb 值为（120, 120, 120），透明度为 200 ，字体对齐方式为顶端左对齐，字号大小为 12 号

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

描述 设置文字的颜色

参数

- **r** (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 文字颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_button.set_text_color(120, 120, 120, 200)`

示例说明 设置字体的 rgb 值为（120, 120, 120），透明度为 200

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

描述 设置控件中文字的对齐方式

参数 **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*

返回 无

示例 `my_Input_field.set_text_align(text_anchor.upper_left)`

示例说明 设置文字的对齐方式为顶端左对齐

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

描述 设置控件中文字的字号大小

参数 **size** (*int*) – 文字的字号值

返回 无

示例 `my_Input_field.set_text_size(12)`

示例说明 设置文字的字号为 12 号

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

描述 设置控件的背景色

参数

- **r** (*int*) – 背景颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 背景颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 背景颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 背景颜色的透明度，[0, 255]

返回 无

示例 `my_Input_field.set_background_color(200, 200, 200, 230)`

示例说明 设置背景色的 rgb 值为 (200, 200, 200) ，透明度为 230

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

描述 设置控件中的提示文字的属性

参数

- **string** (*string*) – 需要显示的字符串内容
- **[color_r, color_g, color_b, color_a]** (*list*) – 可选参数，需要显示的字符串的颜色，参数分别为显示颜色 r 值、b 值，g 值，透明度，取值范围都为 [0, 255]
- **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*
- **size** (*int*) – 显示文字的字号大小

返回 无

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

示例说明 设置提示文字的 rgb 值为（120, 120, 120），透明度为 200 ，字体对齐方式为顶端左对齐，字号大小为 12 号

`input_field_object.set_hint_text_color(r, g, b, a)`

描述 设置控件提示文字的颜色

参数

- **r** (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 文字颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Input_field.set_text_color(120, 120, 120, 200)`

示例说明 设置提示文字的 rgb 值为（120, 120, 120），透明度为 200

`input_field_object.set_hint_text_align(align)`

描述 设置提示文字的对齐方式

参数 **align** (*enum*) – 可选参数，枚举类型，需要显示文字的对齐方式，详细见表格*align*

返回 无

示例 `my_Input_field.set_text_align(text_anchor.upper_left)`

示例说明 设置提示文字的对齐方式为顶端左对齐

`input_field_object.set_hint_text_size(size)`

描述 设置提示文字的字号大小

参数 `size` (*int*) – 文字的字号值

返回 无

示例 `my_Input_field.set_text_size(12)`

示例说明 设置 hint 对象中文字的字号为 12 号

5.2.7 Dropdown

Dropdown 控件通常用于在某个对象的多个属性选项中，选中某个特定值

`dropdown_object.set_option(*options)`

描述 设置下拉框中的内容，输入为字符串列表，列表中元素个数为下拉框选项个数

参数 `*args` (*string*) – 下拉框中的选项内容

返回 无

示例 `my_Dropdown.set_option('RoboMaser EP', 'People')`

示例说明 下拉框中有两个选项，分别为 `RoboMaster EP` 与 `People`

`dropdown_object.set_text_color(r, g, b, a)`

描述 设置文字的颜色

参数

- `r` (*int*) – 文字颜色的 r 值，范围为 [0, 255]
- `g` (*int*) – 文字颜色的 g 值，范围为 [0, 255]
- `b` (*int*) – 文字颜色的 b 值，范围为 [0, 255]
- `a` (*int*) – 文字颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Dropdown.set_text_color(120, 120, 120, 200)`

示例说明 设置文字的 rgb 值为（120, 120, 120），透明度为 200

`dropdown_object.set_background_color(r, g, b, a)`

描述 设置下拉框中选中的条目的背景色

参数

- `r` (*int*) – 背景颜色的 r 值，范围为 [0, 255]
- `g` (*int*) – 背景颜色的 g 值，范围为 [0, 255]
- `b` (*int*) – 背景颜色的 b 值，范围为 [0, 255]
- `a` (*int*) – 背景颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_DropDown.set_background_color(200, 200, 200, 230)`

示例说明 设置下拉框中选中的条目的背景色的 rgb 值为 (200, 200, 200)，透明度为 230

`dropdown_object.set_arrow_color(r, g, b, a)`

描述 设置下拉框选箭头的颜色

参数

- **r** (*int*) – 箭头颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 箭头颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 箭头颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 箭头颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Dropdown.set_arrow_color(120, 120, 120, 200)`

示例说明 设置下拉框选中箭头颜色的 rgb 值为（120, 120, 120），透明度为 200

`dropdown_object.set_item_color(r, g, b, a)`

描述 设置下拉框中未被选中条目的字体颜色

参数

- **r** (*int*) – 字体颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 字体颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 字体颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 字体颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_Dropdown.set_item_color(120, 120, 120, 200)`

示例说明 下拉框中未被选中条目的字体颜色的 rgb 值为（120, 120, 120），透明度为 200

`dropdown_object.set_item_background_color(r, g, b, a)`

描述 设置下拉框中未被选择的条目的背景色

参数

- **r** (*int*) – 背景颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – 背景颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – 背景颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – 背景颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_DropDown.set_item_background_color(200, 200, 200, 230)`

示例说明 设置下拉框中未被选择的条目的背景色的 rgb 值为 (200, 200, 200) ，透明度为 230

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

描述 设置下拉框中选中图标的颜色

参数

- **r** (*int*) – checkmark 颜色的 r 值，范围为 [0, 255]
- **g** (*int*) – checkmark 颜色的 g 值，范围为 [0, 255]
- **b** (*int*) – checkmark 颜色的 b 值，范围为 [0, 255]
- **a** (*int*) – checkmark 颜色的透明度，范围为 [0, 255]

返回 无

示例 `my_DropDown.set_item_checkmark_color(200, 200, 200, 230)`

示例说明 设置下拉框中 checkmark 颜色的 rgb 值为 (200, 200, 200) ，透明度为 230

CHAPTER 6

发射器

`ir_blaster_ctrl.set_fire_count(count)`

描述 设置红外光束的发射频率，即每秒射出的红外光束次数

参数 `color_enum` (*int*) – 发射频率，即每秒射出的红外光束次数，范围为 [1:8]

返回 无

示例 `ir_blaster_ctrl.set_fire_count(4)`

示例说明 设置红外光束的发射频率为 4

`ir_blaster_ctrl.fire_once()`

描述 控制发射器只发射一次红外光束

参数 `void` – 无

返回 无

示例 `ir_blaster_ctrl.fire_once()`

示例说明 控制发射器只发射一次红外光束

`ir_blaster_ctrl.fire_continuous()`

描述 控制发射器持续发射红外光束

参数 `void` – 无

返回 无

示例 `ir_blaster_ctrl.fire_continuous()`

示例说明 控制发射器持续发射红外光束

`ir_blaster_ctrl.stop()`

描述 停止发射红外光束

参数 `void` – 无

返回 无

示例 `ir_blaster_ctrl.stop()`

示例说明 停止发射红外光束

CHAPTER 7

拓展机构

7.1 机械爪

`gripper_ctrl.open()`

描述 控制机械爪打开

参数 `void` – 无

返回 无

示例 `gripper_ctrl.open()`

示例说明 控制机械爪打开

`gripper_ctrl.close()`

描述 控制机械爪关闭

参数 `void` – 无

返回 无

示例 `gripper_ctrl.close()`

示例说明 控制机械爪关闭

`gripper_ctrl.stop()`

描述 控制机械爪停止运动

参数 `void` – 无

返回 无

示例 `gripper_ctrl.stop()`

示例说明 控制机械爪停止运动

`gripper_ctrl.update_power_level(level)`

描述 设置机械爪力度档位

参数 `level` (*int*) – 机械爪的力度档位，范围为 [1:4] 档，默认为 1

返回 无

示例 `gripper_ctrl.update_power_level(1)`

示例说明 设置机械爪力度档位为 1

`gripper_ctrl.is_closed()`

描述 获取机械爪夹紧状态

参数 `void` – 无

返回 机械爪夹紧状态，若机械爪夹紧则返回 true，否则返回 false

返回类型 bool

示例 `ret = gripper_ctrl.is_closed()`

示例说明 获取机械爪夹紧状态

`gripper_ctrl.is_open()`

描述 获取机械爪张开状态

参数 `void` – 无

返回 机械爪张开状态，若机械爪完全张开则返回 true，否则返回 false

返回类型 bool

示例 `ret = gripper_ctrl.is_open()`

示例说明 获取机械爪张开状态

7.2 机械臂

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

描述 设置机械臂运动的相对位置

参数

- `x` (*int32*) – 设置机械臂水平运动的距离，正数为向前运动，负数为向后运动，精确度为 1 mm

- **y** (*int32*) – 设置机械臂垂直运动的距离，正数为向上运动，负数为向下运动，精确度为 1 mm
- **wait_for_complete** (*bool*) – 是否等待执行完成，默认为 True

返回 无

示例 `robotic_arm_ctrl.move(40, 50, True)`

示例说明 设置机械臂向前移动 20 mm，向上移动 30 mm，等待执行完成

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

描述 设置机械臂运动到绝对坐标

参数

- **x** (*int32*) – 设置机械臂水平运动的坐标值，精确度为 1 mm
- **y** (*int32*) – 设置机械臂垂直运动的坐标值，精确度为 1 mm
- **wait_for_complete** (*bool*) – 是否等待执行完成，默认为 True

返回 无

示例 `robotic_arm_ctrl.moveto(40, 50, True)`

示例说明 设置机械臂移动到（x=40mm，y=50mm）的绝对坐标，等待执行完成

`robotic_arm_ctrl.get_position()`

描述 获取机械臂位置

参数 **void** – 无

返回 机械臂的绝对坐标，精确度为 1 mm

返回类型 列表 [x, y], x 和 y 为 int32 类型

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

示例说明 获取机械臂的绝对坐标

`robotic_arm_ctrl.recenter()`

描述 设置机械臂回中

参数 **void** – 无

返回 无

示例 `robotic_arm_ctrl.recenter()`

示例说明 设置机械臂回中

7.3 舵机

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

描述 获取舵机旋转角度

参数 **servo_id** (*uint8*) – 舵机编号，范围为 [1:4]

返回 舵机角度，精确度为 0.1 度

返回类型 int32

示例 `angle = servo_ctrl.get_angle(1)`

示例说明 获取编号为 1 的舵机旋转角度

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

描述 设置舵机旋转角度

参数

- **servo_id** (*uint8*) – 舵机编号，范围为 [1:4]
- **angle** (*int32*) – 旋转角度，精确度为 0.1 度，正数为顺时针旋转，负数为逆时针旋转
- **wait_for_complete** (*bool*) – 是否等待执行完成，默认为 True

返回 无

示例 `servo_ctrl.set_angle(1, 900, True)`

示例说明 设置编号为 1 的舵机顺时针旋转 90°，等待执行完成

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

描述 设置舵机回中

参数

- **servo_id** (*uint8*) – 舵机编号，范围为 [1:4]
- **wait_for_complete** (*bool*) – 是否等待执行完成，默认为 True

返回 无

示例 `servo_ctrl.recenter(1, True)`

示例说明 设置编号为 1 的舵机回中，等待执行完成

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

描述 设置舵机旋转速度

参数

- **servo_id** (*uint8*) – 舵机编号，范围为 [1:4]

- `speed` (*int32*) – 旋转速度，精确度为 1 度/秒，正数为顺时针旋转，负数为逆时针旋转

返回 无

示例 `servo_ctrl.set_speed(1, 5)`

示例说明 设置编号为 1 的舵机顺时针旋转，旋转速度为 5 度/秒

CHAPTER 8

智能

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

描述 设置视觉标签识别颜色

参数 **`color_enum`** – 标签颜色类型，详细见表格*`color_enum`*

返回 无

示例 `vision_ctrl.marker_detection_color_set(rm_define.marker_detection_color_red)`

示例说明 设置视觉标签识别颜色为红色

`color_enum`

rm_define.marker_detection_color_red	红色
rm_define.marker_detection_color_green	绿色
rm_define.marker_detection_color_blue	蓝色

CHAPTER 9

装甲板

def ir_hit_detection_event(msg):

描述 当检测到机器人受到红外光束攻击时，运行函数内程序

参数 **msg** – 函数内部的消息参数

返回 无

示例

```
# 当检测到机器人受到红外光束攻击时，运行函数内程序

def ir_hit_detection_event(msg):
    pass
```

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

描述 等待机器人受到红外光束攻击时，执行下一条指令

参数 **condition_enum** – 事件类型，rm_define.cond_ir_hit_detection 表示机器人受到红外光束攻击

返回 无

示例 armor_ctrl.cond_wait(rm_define.cond_ir_hit_detection)

示例说明 等待机器人受到红外光束攻击时，执行下一条指令

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

描述 判断机器人是否受到红外光束攻击

参数 `condition_enum` – 事件类型，`rm_define.cond_ir_hit_detection` 表示机器人受到红外光束攻击

返回 机器人是否受到红外光束攻击，受到攻击时返回真，否则返回假。

返回类型 bool

示例 `if armor_ctrl.check_condition(rm_define.cond_ir_hit_detection):`

示例说明 如果机器人受到红外光束攻击时，执行下一条指令

CHAPTER 10

传感器

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

描述 开启红外深度传感器测距功能

参数 **`port_id`** (*int*) – 红外深度传感器模块编号，范围为 [1:4]

返回 无

示例 `ir_distance_sensor_ctrl.enable_measure(1)`

示例说明 开启 1 号红外深度传感器测距功能

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

描述 关闭红外深度传感器测距功能

参数 **`port_id`** (*int*) – 红外深度传感器模块编号，范围为 [1:4]

返回 无

示例 `ir_distance_sensor_ctrl.disable_measure(1)`

示例说明 关闭 1 号红外深度传感器测距功能

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

描述 获取红外深度传感器测距信息

参数 **`port_id`** (*int*) – 红外深度传感器模块编号，范围为 [1:4]

返回 红外深度传感器前方障碍物的距离，精确度为 1 cm

返回类型 uint16

示例 `ir_distance_sensor_ctrl.get_distance_info(1)`

示例说明 获取 1 号红外深度传感器测距信息

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

描述 当检测到红外深度传感器模块前方障碍物距离满足条件时，运行函数内程序

参数

- `port_id` (*int*) – 红外深度传感器模块编号，范围为 [1:4]
- `compare_type` – 比较类型，可以为 eq, ge, gt, le, lt, 分别表示等于，大于等于，大于，小于等于，小于
- `dist` – 用于比较的距离，精确度为 1 cm，范围为 5~500 cm，误差率为 5%

返回 无

示例

```
# 当检测到 1 号红外深度传感器前方障碍物距离小于 10 cm 时，运行函数内程序

def ir_distance_1_lt_10_event(msg):
    pass
```

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

描述 等待红外深度传感器模块前方障碍物距离满足条件时，执行下一条指令

参数

- `'ir_distance_[port_id]_[compare_type]_[dist]'` – 用于距离比较的字符串，含模块编号，比较类型和距离
- `port_id` (*int*) – 红外深度传感器模块编号，范围为 [1:4]
- `compare_type` – 比较类型，可以为 eq, ge, gt, le, lt, 分别表示等于，大于等于，大于，小于等于，小于
- `dist` – 用于比较的距离，精确度为 1 cm，范围为 5~500 cm，误差率为 5%

返回 无

示例 `ir_distance_sensor_ctrl.cond_wait('ir_distance_1_gt_50')`

示例说明 等待 1 号红外深度传感器模块前方障碍物距离大于 50 cm 时，执行下一条指令

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

描述 判断红外深度传感器模块前方障碍物距离是否满足条件

参数

- `'ir_distance_[port_id]_[compare_type]_[dist]'` – 用于距离比较的字符串，含模块编号，比较类型和距离

- **port_id** (*int*) – 红外深度传感器模块编号，范围为 [1:4]
- **compare_type** – 比较类型，可以为 eq, ge, gt, le, lt, 分别表示等于，大于等于，大于，小于等于，小于
- **dist** – 用于比较的距离，精确度为 1 cm，范围为 5~500 cm，误差率为 5%

返回 是否满足条件，满足条件时返回真，否则返回假。

返回类型 bool

示例

```
# 当检测到 1 号红外深度传感器前方障碍物距离小于 10 cm 时，运行函数内程序

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

CHAPTER 11

转接模块

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

描述 获取传感器转接模块相应端口模拟引脚的 ADC 值

参数

- **`board_id`** (*int*) – 传感器转接模块编号，范围为 [1:6]
- **`port_num`** (*uint8*) – 传感器转接模块上的端口号，范围为 [1:2]
- **`wait_for_complete`** (*bool*) – 是否等待执行完成，默认为 True

返回 传感器转接模块相应端口模拟引脚的 ADC 值，范围为 [0:1023]

返回类型 uint16

示例 `ret = sensor_adapter_ctrl.get_sensor_adapter_adc(1, 2)`

示例说明 获取 1 号传感器转接模块 2 号端口模拟引脚的 ADC 值

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

描述 获取传感器转接模块相应端口引脚的脉冲持续时间

参数

- **`board_id`** (*int*) – 传感器转接模块编号，范围为 [1:6]
- **`port_num`** (*uint8*) – 传感器转接模块上的端口号，范围为 [1:2]

返回 传感器转接模块相应端口引脚的脉冲持续时间，精确度为 1 ms

返回类型 uint32

示例 `ret = sensor_adapter_ctrl.get_sensor_pulse_period(1, 2)`

示例说明 获取 1 号传感器转接模块 2 号端口引脚脉冲持续时间

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

描述 当检测到传感器转接模块相应端口引脚跳变为高电平/低电平/双向，运行函数内程序

参数

- `board_id` (*int*) – 传感器转接模块编号，范围为 [1:6]
- `port_num` (*uint8*) – 传感器转接模块上的端口号，范围为 [1:2]
- `judge_type` – 触发条件，可以为 high, low, trigger，分别表示高电平，低电平还是双向跳变

返回 无

示例

```
# 当检测到 1 号传感器转接模块 2 号端口引脚跳变为高电平时，运行函数内程序

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)`

描述 等待传感器转接模块相应端口引脚脉冲为（高/低/跳变）时，执行下一条指令

参数

- `board_id` (*int*) – 传感器转接模块编号，范围为 [1:6]
- `port_num` (*uint8*) – 传感器转接模块上的端口号，范围为 [1:2]
- `judge_type` – 触发条件，可以为 high, low, trigger，分别表示高电平，低电平还是双向跳变

返回 无

示例 `sensor_adapter_ctrl.cond_wait(rm_define.cond_sensor_adapter1_port2_high_event)`

示例说明 等待 1 号传感器转接模块 2 号端口引脚为高电平时，执行下一条指令

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

描述 判断传感器转接模块相应端口引脚脉冲是否为（高/低/跳变）

参数

- `board_id` (*int*) – 传感器转接模块编号，范围为 [1:6]
- `port_num` (*uint8*) – 传感器转接模块上的端口号，范围为 [1:2]

- **judge_type** – 触发条件，可以为 high, low, trigger，分别表示高电平，低电平还是双向跳变

返回 是否满足条件，满足条件时返回真，否则返回假。

返回类型 bool

示例

```
# 如果 1 号传感器转接模块 2 号端口引脚正在跳变时，执行下一条指令

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*)

描述 设置串口的波特率、数据位、校验位以及停止位属性

参数

- **baud_rate** – 设置波特率，可选波特率为 9600、19200、38400、57600、115200
- **data_bit** – 设置数据位，可选的数据位为 cs7、cs8
- **odd_even_crc** – 设置奇偶校验，详细见表格*odd_even_crc*
- **stop_bit** – 设置停止位，可选的停止位为 1、2

返回 无

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

示例说明 设置串口的波特率为 9600，数据位 8 位，不使用奇偶校验，停止位为 1 位

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

描述 发送字符串信息，自动添加换行 `'\n'`

参数 **msg_string** (*string*) – 需要发送的字符串信息，发送时字符串后自动添加 `'\n'`

返回 无

示例 `serial_ctrl.write_line('RoboMaster EP')`

示例说明 向串口写入 `'RoboMaster EP\n'` ，最后的换行自动添加，用户只需要发送 `'RoboMaster EP'`

`serial_ctrl.write_string(msg_string)`

描述 发送字符串信息

参数 `msg_string` (*string*) – 需要发送的字符串信息

返回 无

示例 `serial_ctrl.write_string('RoboMaster EP')`

示例说明 向串口写入 `'RoboMaster EP'`

`serial_ctrl.writ_numbers(key, value)`

描述 将参数以键值对的形式组成字符串，并通过串口发送出去

参数

- `key` (*string*) – 需要发送的关键字
- `value` (*uint32*) – 需要发送的值

返回 无

示例 `serial_ctrl.writ_numbers('x', 12)`

示例说明 向串口中写入字符串 `'x:12'`

`serial_ctrl.read_line()`

描述 从串口中读取以 `'\n'` 结尾的字符串

参数 `void` – 无

返回 通过串口读取到的字符串

返回类型 string

示例 `recv = serial_ctrl.read_line()`

示例说明 从串口读取一行以 `'\n'` 结尾的字符串

`serial_ctrl.read_string()`

描述 从串口中读取字符串（字符串可以不以 `'\n'` 结尾）

参数 `void` – 无

返回 通过串口读取到的字符串

返回类型 string

示例 `recv = serial_ctrl.read_line()`

示例说明 从串口读取一个字符串

`serial_ctrl.read_until(stop_sig)`

描述 从串口中读取字符串，直到匹配到指定的结束字符 `'stop_sig'`

参数 `stop_sig` – 指定的结束字符，参数类型为字符，范围为 [`'\n'` | `'$'` | `'#'` | `'.'` | `':'` | `';'`]

返回 通过串口读取到的匹配字符串

返回类型 string

示例 `serial_ctrl.read_until('#')`

示例说明 从串口中读取字符串，直到匹配到 `'#'` 停止读取

`odd_even_crc`

none	不使用奇偶校验
odd	使用奇校验
even	使用偶校验

CHAPTER 13

拓展模块使用说明

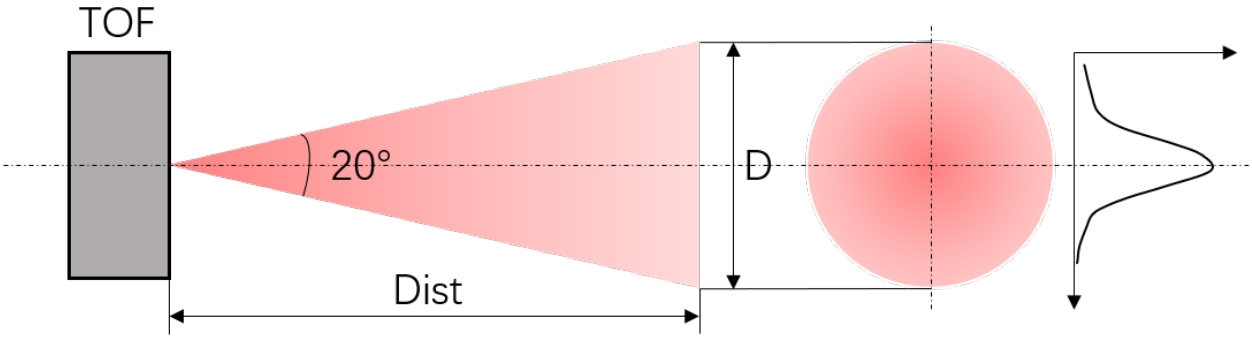
13.1 红外深度传感器

1. 介绍：

红外深度传感器的设计是基于 TOF(Time of Flight) 即飞行时间原理。即传感器发出经调制的近红外光，遇物体后反射，传感器通过计算光线发射和反射时间差或相位差，来计算距离物体的距离。

2. 产品特性

红外深度传感器的探测面积见下图，



其发出的是一个角度为 20° 的圆锥光，这个光斑 D 与距离 Dist 的关系：

$$\mathbf{D=2\times Dist\times tan(10)}$$

要实现最佳测试效果，应保证目标物的尺寸要至少等于 TOF 光斑的尺寸。

小技巧: 如果目标物小于光斑大小，那么应保证目标物尽量在光斑的中心位置，因为光斑内的光强分布并不是均匀的，而是呈一个类高斯分布，中间光强大，四周光强小，为了保证返回光能量足够，应尽量保证目标物在光斑中心。

3. 引脚说明

编号	引脚	功能	对应连接项
1	VCC	供电	电源正极
2	GND	供电	电源地
3	TX	发送	RX
4	RX	接收	TX

4. 通讯协议和数据格式

通讯接口	波特率	数据位	停止位	奇偶校验
UART	115200	8	1	none

控制命令输入:

`ir_distance_sensor_measure_on`

描述 打开红外深度传感器数据输出

`ir_distance_sensor_measure_off`

描述 关闭红外深度传感器数据输出

数据输出:

`ir distance:xxx`

描述 红外深度传感器数据格式

小技巧: 命令格式都以字符串形式输入和输出

13.2 舵机

1. 介绍

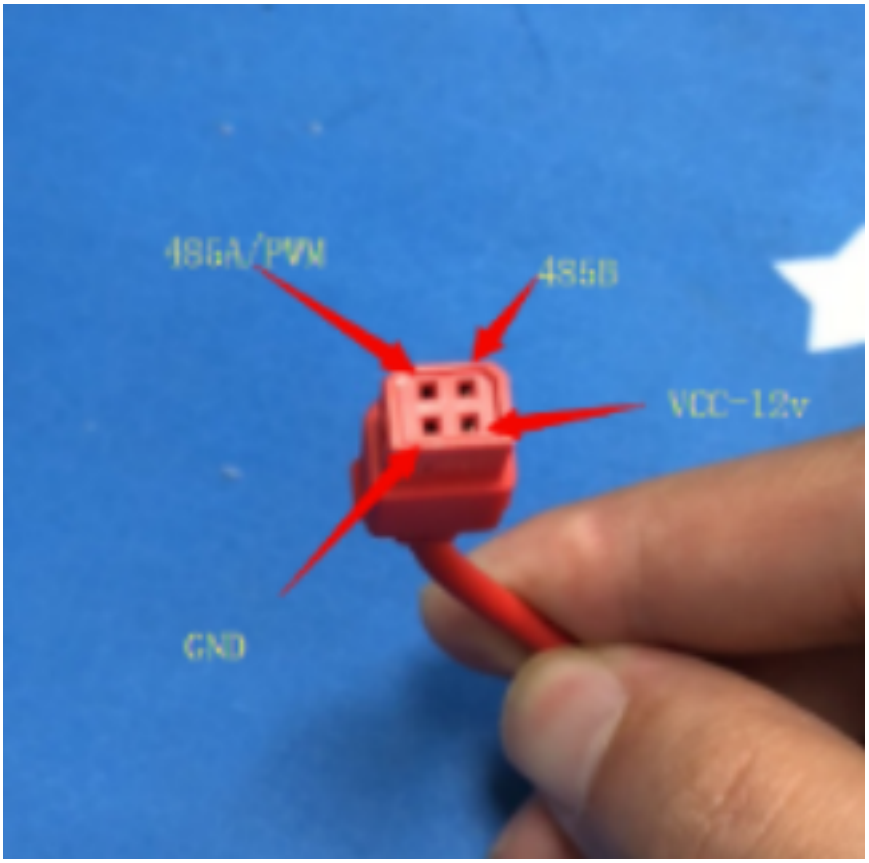
舵机的油门控制方式除了支持 485 控制，还可以进行 PWM 控制，控制模式包含：速度模式和角度模式。

注解: 舵机的控制模式

舵机的控制模式需要通过官方的编程接口（Scratch / Python ）进行切换，并且会记录在舵机内部，不会随着舵机掉电而重置。使用 PWM 控制前请确认舵机当前的控制模式。

2. 引脚说明

舵机上的 485 管脚和 PWM 管脚复用，如下图所示



3. 控制说明

在 PWM 控制方式下，舵机对应的输入输出

控制模式	脉冲周期	油门范围	舵机输出
角度模式	50Hz	2.5%~12.5%	0°~360°
速度模式	50hz	2.5%~7.5%	49rpm~0rpm, 顺时针
		7.5%~12.5%	0rpm~—49rpm, 逆时针

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

索引

L

M

O

R

S

T

V