步兵sdk的程序基本架构

阅读官方文档

文档分为5个蓝字板块，其中我们主要用到明文SDK说明的内容

1. 机械臂与机械爪

2. 舵机

3. 红外深度传感器

4. 传感器转接模块

5. UART 接口

明文 SDK 说明

1. 明文 SDK 介绍

2. 接入方式

3. 明文协议

4. 编队控制

PYTHON 编程说明

1. Python 编程介绍

2. Python 功能介绍

3. Python API

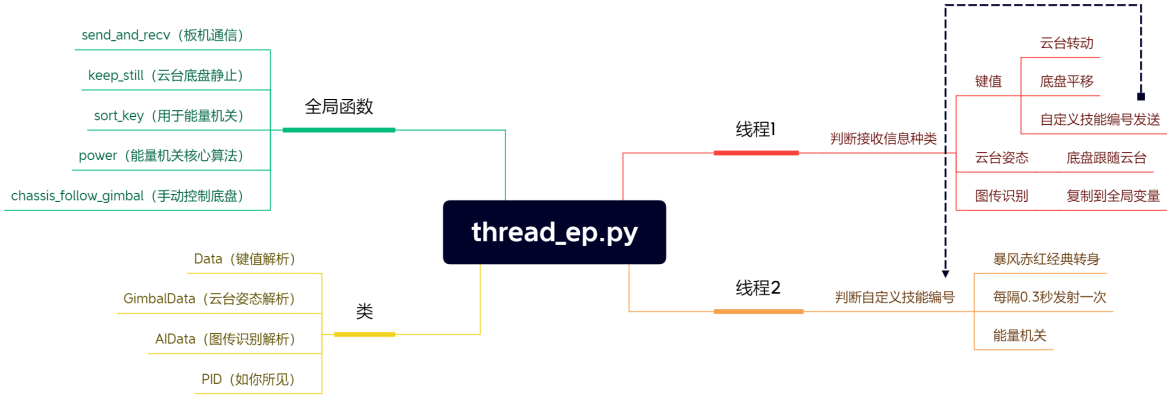
版本说明

版本说明

备注：

请详细阅读明文SDK介绍中的内容，建立起EP与第三方模块通信的概念

架构图



建立连接

由于步兵底盘跟随云台的特殊功能需求，底盘的运动数据时刻发生改变，这需要较高的控制频率。在EP的三种硬件接入方式中，控制频率由高到低如下所示：

- usb连接（≈40Hz）
- wifi连接（≈20Hz）
- uart连接（≈10Hz）

所以usb连接是最好的选择。

建立两个元组存储端口信息

```
# 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
# 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
address_ctrl = ("192.168.42.2", int(40923))  #控制命令
address_push = ("0.0.0.0", int(40924))  #推送消息
```

启动EP与第三方硬件的连接

```
# 与机器人控制命令端口建立 TCP 连接 （用于发送控制命令）
sock_ctrl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Connecting sock_ctrl...")
sock_ctrl.connect(address_ctrl)
print("Connected!")

# 与机器人消息推送端口建立 UDP 连接 （用于接收推送信息）
sock_push = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("Connecting sock_push...")
sock_push.bind(address_push)
print("Connected!")
```

构建交互函数

```
def send_and_recv(s, string, en_print=False):
    with send_recv_lock:
        s.send(string.encode('utf-8'))  #将数据编码成utf-8
        if en_print == True:
            print("send: " + string)
        try:    #使用试错语法，有时候s.recv(1024)会莫名其妙报错，为避免程序死机使用try
            buf = s.recv(1024)  #1024是一次接收的数据位数上限，一般1024够用
        except:
            pass
        if en_print == True:    #用于调试
            print("back: " + buf.decode('utf-8'))   #将接收到的数据解码成utf-8打印到
控制台
    return buf  #返回接收到的数据
```

这个函数的作用是向EP的控制端口发送信息并接收返回的信息（一般是"ok;"，具体看各个命令的文档）。

s是用户选择的交互端口

with send_recv_lock:是py多线程编程中互斥锁的用法。*send_recv_lock*是一个互斥锁，定义在全局。

```
send_recv_lock = threading.Lock()
```

使用动机

当多个线程同时向EP发送信息时，可能会导致错误，为避免发生这种情况，使用*with send_recv_lock:*就相当于告诉所有线程：我已经获取了发送权限，你们谁都别抢。

使用效果

自己去试

让EP进入状态（发送初始化EP的命令）

```
send_and_recv(sock_ctrl, "command;")    #进入sdk模式
send_and_recv(sock_ctrl, "quit;")       #退出sdk模式
send_and_recv(sock_ctrl, "command;")    #sdk再重进是为了解决图传被卡住的bug（DJI的锅）
send_and_recv(sock_ctrl, "robot mode free;")
send_and_recv(sock_ctrl, "game_msg on;")    #打开键值推送
send_and_recv(sock_ctrl, "blaster bead 1;")     #设置单次发射数量
```

我们在第四行把ep的整机模式设置成自由模式，为什么不是 底盘跟随云台 呢？因为向EP发送底盘控制类的命令（例如以特定速度移动）会迫使EP退出底盘跟随云台的模式，最终还是变成自由模式。云台跟随底盘 也是同理，向EP发送云台控制类的命令会迫使EP退出原有模式，变成自由模式。所以所幸一开始就设置成自由模式，至于底盘跟随云台的代码就自己写。

线程1（手动阶段基本控制）

在这个线程，需要完成

- 对键值的接收和解码
- 对图传识别信息的接收和解码
- 实现WASD移动和疾跑

- 通知对应线程执行用户调用的相关技能（能量机关、转身等）

全局变量声明

```
global mk_list  #图传识别信息数组
global block_skill  #特定技能的编号
global std_pitch    #云台当前pitch轴
global std_yaw  #云台当前yaw轴
global is_blocked_blaster   #是否关闭发射器发射权限
global is_blocked_man_drive_chassis #是否关闭底盘跟随云台权限
global is_blocked_man_drive_gimbal  #是否关闭鼠标控制云台权限
```

键值数据类

接收到的键值是以一个字符串接收的，如下所示：

```
game msg push [0, 6, 1, 0, 0, 255, 1, 199];
```

根据 面向对象 的编程思想，我们可以把每次接收到的信息看成一个对象，键值就是它的属性，所以构建一下类

```
class Data:
    def __init__(self, raw):
        self.raw = raw
        substr = raw[15:-2]
        data_str_list = substr.split(',')
        self.cmd_id = int(data_str_list[0])
        self.len = int(data_str_list[1])
        self.mouse_press = int(data_str_list[2])
        self.mouse_x = int(data_str_list[3])
        self.mouse_y = int(data_str_list[4])
        self.seq = int(data_str_list[5])
        self.key_num = int(data_str_list[6])
        self.keys = []

        if not self.key_num == 0:
            for i in range(self.key_num):
                self.keys.append(int(data_str_list[i + 7]))

    def print_data(self):
        output = "["+str(time.time())+"] "
        output += "cmd_id: " + str(self.cmd_id) + ","
        output += "len: " + str(self.len) + ","
        output += "mouse_press: " + str(self.mouse_press) + ","
        output += "mouse_x: " + str(self.mouse_x) + ","
        output += "mouse_y: " + str(self.mouse_y) + ","
        output += "seq: " + str(self.seq) + ","
        output += "key_num: " + str(self.key_num) + ","

        if self.key_num > 0:
            output += "keys: " + " ".join(map(str, self.keys))

        print(output)
```

在线程1中，把接收的数据交给Data类处理：

```
data1 = Data(buf)
```

于是我们就得到了一批关键数据，包括

```
data1.mouse_x    #鼠标横向移动的速度
data1.mouse_y    #鼠标纵向移动的速度
data1.keys[]     #键盘按下的按键
data1.mouse_press    #鼠标按下了哪个键
```

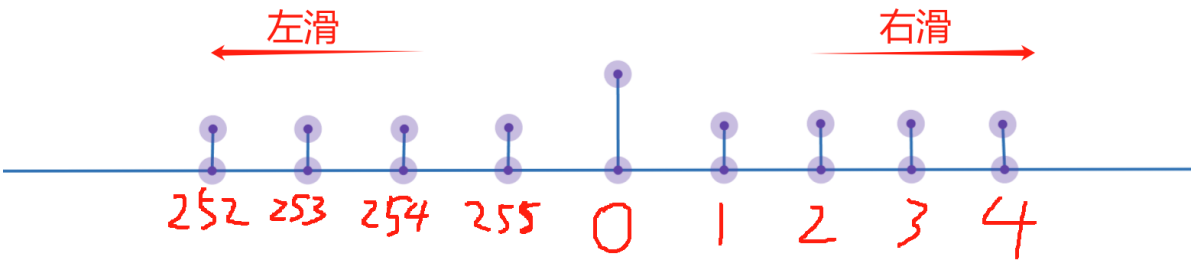
然后就可以根据这些数据计算出云台和底盘分别应该怎么运动

云台控制

鼠标移动速度的数据比较粗糙，如果直接映射到云台上的效果就是云台转动一卡一卡的。所以使用均值滤波平滑数据后再映射

```
# 将鼠标移动加速度数据转化为正负数
data1.mouse_x = data1.mouse_x - 255 if data1.mouse_x > 125 else data1.mouse_x
data1.mouse_y = data1.mouse_y - 255 if data1.mouse_y > 125 else data1.mouse_y
# 均值滤波
data1.mouse_x = (data1.mouse_x + data0.mouse_x) / 2
data1.mouse_y = (data1.mouse_y + data0.mouse_y) / 2
```

值得注意的是鼠标移速的特殊数据表达，以横向速度为例：



所以才有了这两行代码：

```
data1.mouse_x = data1.mouse_x - 255 if data1.mouse_x > 125 else data1.mouse_x
data1.mouse_y = data1.mouse_y - 255 if data1.mouse_y > 125 else data1.mouse_y
```

接下来只需要一个简单的P控制就能实现云台转动了

```
# 鼠标控制云台运动
if is_blocked_man_drive_gimbal == False:
    send_and_recv(sock_ctrl, "gimbal speed p {0} y {1} wait_for_complete
false;".format(data1.mouse_y * 22 +        int(proPitch), data1.mouse_x * 16 +
int(proYaw)))
```

云台角度类

根据键值数据类的经验，我们给接收到的云台角度也写了个类

```
class GimbalData:
    def __init__(self, raw):
        substr = raw[21:-1]   # 消息推送
        data_str_list = substr.split(' ')
        self.pitch = float(data_str_list[0])
        self.yaw = float(data_str_list[1])

    def print_data(self):
        print(f"[{time.time()}] pitch:{self.pitch}° yaw:{self.yaw}°")
```

用于底盘跟随云台的pid类

```
class PID:
    def __init__(self, kp, ki, kd, maxIntergral, maxOutput, deadZone,
errLpfRatio):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.integral = 0
        self.error = 0
        self.last_error = self.error
        self.output = 0
        self.maxIntergral = maxIntergral
        self.maxOutput = maxOutput
        self.deadZone = deadZone
        self.errLpfRatio = errLpfRatio
    def update(self, error):
        self.last_error = self.error if self.last_error>self.deadZone else 0
        self.error = error*self.errLpfRatio+self.last_error*(1-self.errLpfRatio)
        self.integral += self.error
        if self.integral > self.maxIntergral:
            self.integral = self.maxIntergral
        elif self.integral < -self.maxIntergral:
            self.integral = -self.maxIntergral
        derivative = (error - self.last_error)
        self.output = self.kp * error + self.ki * self.integral + self.kd *
derivative
        if self.output > self.maxOutput:
            self.output = self.maxOutput
        elif self.output < -self.maxOutput:
            self.output = -self.maxOutput
        return self.output
```

- kp,ki,kd：pid基本的3个参数
- maxIntergral：最大积分
- maxOutput：最大输出
- deadZone：死区
- errLpfRatio：低通滤波系数，越接近1，则当前误差的权重更大，否则更看重上一次误差

使用时先实例化一个pid对象

```
pid_chassis_stop = PID(1.6,0,0,0,1000,1,0)
```

将误差输入函数并获取输出

```
z = pid_chassis_stop.update(yaw) if (x==0 and y==0) else
pid_chassis_move.update(yaw)
```

底盘控制函数

由于底盘暂时不具备写成类的需求（类一般有有多个属性、多个固有函数），所以用单独一个全局函数的形式写

```
def chassis_follow_gimbal(keys, yaw):
    move_speed = 150  # 行走速度
    rush_speed = 1000  # 疾跑速度
    speed_tr = move_speed  # 默认平移速度等于行走速度
    x, y, z = 0, 0, 0  # 底盘运动三维速度
    tr_dir = 0  # 底盘坐标系的运动方向

    if 16 in keys:  # shift键
        speed_tr = rush_speed
    else:
        speed_tr = move_speed

    if (87 in keys):  # W键
        x = speed_tr
    elif (83 in keys):  # S键
        x = -1 * speed_tr
    else:
        x = 0

    if (68 in keys):  # D键
        y = speed_tr
    elif (65 in keys):  # A键
        y = -1 * speed_tr
    else:
        y = 0

    # 将云台坐标系下运动方向转化为底盘坐标系下运动方向
    if y == 0:
        tr_dir = yaw
    elif x == 0:
        tr_dir = yaw + 90
    else:
        tr_dir = yaw + 45

    z = pid_chassis_stop.update(yaw) if (x==0 and y==0) else
pid_chassis_move.update(yaw)  # 根据云台与底盘夹角比例控制Z轴
    x *= math.cos(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算X方向速度分量
    y *= math.sin(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算Y方向速度分量

    cos45 = math.cos(math.radians(45))
    send_and_recv(sock_ctrl, "chassis wheel w1 {0} w2 {1} w3 {2} w4
{3};".format(-y*cos45+x*cos45-z*cos45,

y*cos45+x*cos45+z*cos45,
```

```
-y*cos45+x*cos45+z*cos45,

y*cos45+x*cos45-z*cos45))
```

Ps：我是按照mc键位写的

代码解析

- 车头方向是x轴正方向
- 车右方向是y轴正方向
- 由于pid误差驱动的特性，底盘无法瞬间跟上云台，很多时候云台与底盘存在角度差，这就造成操作手按下方向键与实际运动方向不符的情况。

  ```
  x *= math.cos(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算X方向速度分量
  y *= math.sin(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算Y方向速度分量
  ```

  这两行代码的作用就是把视角坐标系的方向映射到底盘坐标系的方向，这样在理论上，底盘始终服务于视角，但是受限于反馈频率和电机响应速度等，实际效果是底盘不会乱动，而是短暂停滞在原地（一般只出现在大幅扭动视角的时候）

线程2（技能）

基本思路：朴素的方法，在线程1中写一组判断，按下对应按键就将“技能代号”写入全局变量block_skill，在线程2中轮询block_skill并执行相应代码段。

线程2主要代码

```
# 线程2：阻塞技能
def thread2_func():
    while True:
        # 获取队列中的数据
        global mk_list
        global block_skill
        global std_pitch
        global std_yaw
        global is_blocked_man_drive_chassis
        global is_blocked_man_drive_gimbal
        global is_blocked_blaster
        if block_skill == 1:    # 暴风赤红经典转身
            is_blocked_man_drive_gimbal = True
            # time.sleep(0.1)
            send_and_recv(sock_ctrl, "gimbal speed p 0 y 450 wait_for_complete
false;")
            time.sleep(0.4)
            send_and_recv(sock_ctrl, "gimbal speed p 0 y 0 wait_for_complete
false;")
            is_blocked_man_drive_gimbal = False
            block_skill = 0
        if block_skill == 2:    # 每隔0.3秒发射一次
            while(block_skill == 2):
                send_and_recv(sock_ctrl, "blaster fire;")
                time.sleep(0.3)
        if block_skill == 3:    # 能量机关
            is_blocked_man_drive_chassis = True   #锁定底盘手动遥控
            is_blocked_man_drive_gimbal = True    #锁定云台手动遥控
```

```
            keep_still()    # 保持底盘、云台静止
            while(block_skill == 3):
                if len(mk_list) >= 5:   #识别到符合要求的标签
                    send_and_recv(sock_ctrl, "gimbal moveto p 0 y 0 vp 180 vy
180 wait_for_complete false;",True)
                    time.sleep(0.3)
                    atk = power(mk_list,0,0,block_skill)    #核心算法和击打操作
                    if atk == 0:
                        send_and_recv(sock_ctrl, "gimbal moveto y 0 vy 540
wait_for_complete false;",True)
                        is_blocked_man_drive_chassis = False   #解锁底盘手动遥控
                        is_blocked_man_drive_gimbal = False    #解锁云台手动遥控
                        block_skill = 0     # 复位技能标识变量
                        break
                time.sleep(0.1)     #必须加，不然多线程处理不过来
            is_blocked_man_drive_chassis = False   #锁定底盘手动遥控
            is_blocked_man_drive_gimbal = False    #锁定云台手动遥控
        time.sleep(0.1)     #必须加，不然多线程处理不过来
```

线程1中的键值判断

```
# 按键技能部分
if 9 in data1.keys and 9 not in data0.keys:   #TAB键     云台旋转180°
    block_skill = 1
if 70 in data1.keys and 70 not in data0.keys:   #F键     每隔0.3秒射一次
    block_skill = 2 if not block_skill == 2 else 0
if 71 in data1.keys and 71 not in data0.keys:     #E键     能量机关
    block_skill = 3 if not block_skill == 3 else 0
```

如何继续?

因为时间赶，阻塞技能的代码没有经过优化，目前仍能优化：

- 将技能封装为函数，增强可读性
- 将常用的代码段写为外部文件中的函数（例如自一的几种路线的移动代码），增强可读性
- 在执行时间较长的技能中设置检查block_skill的语句，保证操作手取消执行后，程序可以在合适的时机停止
- 善用try except语法，尽量避免死机（报错）

完整版

```
import threading
import time
import socket
import math
import sys
import string
import serial
from queue import Queue


# 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
# 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
address_ctrl = ("192.168.42.2", int(40923))  #控制命令
address_push = ("0.0.0.0", int(40924))  #推送消息
```

```
# 设置串口参数
# serial_port = "COM3"  # 请根据你的系统和连接方式修改串口名称
# baud_rate = 115200
# ser = serial.Serial(serial_port, baud_rate)

def send_and_recv(s, string, en_print=False):
    with send_recv_lock:
        s.send(string.encode('utf-8'))
        if en_print == True:
            print("send: " + string)
        try:
            buf = s.recv(1024)
        except:
            pass
        if en_print == True:
            print("back: " + buf.decode('utf-8'))
    return buf

def keep_still():
    send_and_recv(sock_ctrl, "gimbal speed p 0 y 0 wait_for_complete false;")
    send_and_recv(sock_ctrl, "chassis wheel w1 0 w2 0 w3 0 w4 0;")

def sort_key(sublist):
    return sublist[0]

def power(mk1,std_pitch,std_yaw,block_skill):
    calc = [[lambda a,b,c,d,f:[b,f,c,f,d]if(b[0]+c[0]+d[0]==24)else[],
             lambda a,b,c,d,f:[a,f,b,f,c,f,d]if(a[0]+b[0]+c[0]+d[0]==24)else[],
             lambda a,b,c,d,f:[a,b,c,f,d]if(b[0]*10+c[0]+d[0]==24)else[],
             lambda a,b,c,d,f:[a,b,f,c,f,d]if(a[0]*10+b[0]+c[0]+d[0]==24)else[],
             lambda a,b,c,d,f:
[a,b,f,c,d]if(a[0]*10+b[0]+c[0]*10+d[0]==24)else[]],

             [lambda a,b,c,d,f:[b,c,f,d]if(b[0]*10+c[0]-d[0]==24)else[],
             lambda a,b,c,d,f:[a,b,f,c,d]if(a[0]*10+b[0]-c[0]*10-d[0]==24)else[],
             lambda a,b,c,d,f:[a,b,f,c,f,d]if(a[0]*10+b[0]-c[0]-d[0]==24)else[]],

             [lambda a,b,c,d,f:[b,c,f,d]if((b[0]*10+c[0])/d[0]==24)else[],
             lambda a,b,c,d,f:
[a,b,c,f,d]if((a[0]*100+b[0]*10+c[0])/d[0]==24)else[]],

             [lambda a,b,c,d,f:[c,f,d]if(c[0]*d[0]==24)else[],
             lambda a,b,c,d,f:[b,c,f,d]if(b[0]*10*c[0]*d[0]==24)else[],
             lambda a,b,c,d,f:[b,f,c,f,d]if(b[0]*c[0]*d[0]==24)else[],
             lambda a,b,c,d,f:[a,f,b,f,c,f,d]if(a[0]*b[0]*c[0]*d[0]==24)else[]]]

    order = [[0, 1, 2, 3], [0, 1, 3, 2], [0, 2, 1, 3], [0, 2, 3, 1],
             [0, 3, 1, 2], [0, 3, 2, 1], [1, 0, 2, 3], [1, 0, 3, 2],
             [1, 2, 0, 3], [1, 2, 3, 0], [1, 3, 0, 2], [1, 3, 2, 0],
             [2, 0, 1, 3], [2, 0, 3, 1], [2, 1, 0, 3], [2, 1, 3, 0],
             [2, 3, 0, 1], [2, 3, 1, 0], [3, 0, 1, 2], [3, 0, 2, 1],
             [3, 1, 0, 2], [3, 1, 2, 0], [3, 2, 0, 1], [3, 2, 1, 0]]
    flag = 0
    for [a,b,c,d] in order:
        if flag == 1:break
        try:
            for func in calc[mk1[4][0]%10]:
                if flag == 1:break
```

```
                for target in func(mk1[a],mk1[b],mk1[c],mk1[d],mk1[4]):
                    if block_skill == 0:
                        flag = 1
                        break
                    if target != []:
                        flag = 1
                        target_pitch=target[2]-target[3]+math.atan((1-
2*target[2])*math.tan(32/180*math.pi))*180/math.pi

 target_yaw=math.atan((2*target[1]-1)*math.tan(45/180*math.pi))*180/math.pi
                        send_and_recv(sock_ctrl, "gimbal moveto p
"+str(target_pitch+std_pitch)+" y "+str(target_yaw+std_yaw)+" vp 540 vy
540;",True)
                        time.sleep(0.08)
                        send_and_recv(sock_ctrl, "blaster fire;")
                        time.sleep(0.06)
            return 0
        except:
            print("list index out of range !!!")
            return 1

class Data:
    def __init__(self, raw):
        self.raw = raw
        substr = raw[15:-2]
        data_str_list = substr.split(',')
        self.cmd_id = int(data_str_list[0])
        self.len = int(data_str_list[1])
        self.mouse_press = int(data_str_list[2])
        self.mouse_x = int(data_str_list[3])
        self.mouse_y = int(data_str_list[4])
        self.seq = int(data_str_list[5])
        self.key_num = int(data_str_list[6])
        self.keys = []

        if not self.key_num == 0:
            for i in range(self.key_num):
                self.keys.append(int(data_str_list[i + 7]))

    def print_data(self):
        output = "["+str(time.time())+"] "
        output += "cmd_id: " + str(self.cmd_id) + ","
        output += "len: " + str(self.len) + ","
        output += "mouse_press: " + str(self.mouse_press) + ","
        output += "mouse_x: " + str(self.mouse_x) + ","
        output += "mouse_y: " + str(self.mouse_y) + ","
        output += "seq: " + str(self.seq) + ","
        output += "key_num: " + str(self.key_num) + ","

        if self.key_num > 0:
            output += "keys: " + " ".join(map(str, self.keys))

        print(output)

class GimbalData:
    def __init__(self, raw):
        substr = raw[21:-1]   # 消息推送
        data_str_list = substr.split(' ')
```

```
        self.pitch = float(data_str_list[0])
        self.yaw = float(data_str_list[1])

    def print_data(self):
        print(f"[{time.time()}] pitch:{self.pitch}° yaw:{self.yaw}°")

class UI:
    def __init__(self,mode):
        self.mode = mode
        self.is_setting = False

class AIData:
    def __init__(self, raw):
        self.raw = raw
        self.num = 0
        self.marker_list = []
        substr = raw[15:-1]
        data_str_list = substr.split(' ')
        if len(data_str_list) > 1:
            float_list = [float(x) for x in data_str_list]
            self.num = int(float_list[0])
            for i in range(self.num):
                new_marker = [int(float_list[i * 5 + 1]),
                              float(float_list[i * 5 + 2]),
                              float(float_list[i * 5 + 3]),
                              float(float_list[i * 5 + 4]),
                              float(float_list[i * 5 + 5])]
                new_marker[0] = 15 if new_marker[0] == 38 else new_marker[0]
                self.marker_list.append(new_marker)
            self.marker_list = sorted(self.marker_list, key=sort_key)
            try:
                for i in range(4):
                    self.marker_list[i][0] -= 10
            except:
                pass
    def print_marker(self):
        print("["+str(time.time())+"] "+str(self.marker_list))

class PID:
    def __init__(self, kp, ki, kd, maxIntergral, maxOutput, deadZone,
errLpfRatio):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.integral = 0
        self.error = 0
        self.last_error = self.error
        self.output = 0
        self.maxIntergral = maxIntergral
        self.maxOutput = maxOutput
        self.deadZone = deadZone
        self.errLpfRatio = errLpfRatio
    def update(self, error):
        self.last_error = self.error if self.last_error>self.deadZone else 0
        self.error = error*self.errLpfRatio+self.last_error*(1-self.errLpfRatio)
        self.integral += self.error
        if self.integral > self.maxIntergral:
            self.integral = self.maxIntergral
```

```
        elif self.integral < -self.maxIntergral:
            self.integral = -self.maxIntergral
        derivative = (error - self.last_error)
        self.output = self.kp * error + self.ki * self.integral + self.kd *
derivative
        if self.output > self.maxOutput:
            self.output = self.maxOutput
        elif self.output < -self.maxOutput:
            self.output = -self.maxOutput
        return self.output

class Kalmentfilter:
    def __init__(self, emea, eest):
        self.z = 0
        self.x = 0
        self.x0 = self.x
        self.eest = eest
        self.eest0 = self.eest
        self.emea = emea
        self.kk = 0
    def update(self, z):
        self.z = z
        self.kk = self.eest0/(self.eest0+self.emea)
        self.x = self.x0 + self.kk*(self.z-self.x0)
        self.eest = (1-self.kk)*self.eest0
        return self.x

pid_chassis_stop = PID(1.6,0,0,0,1000,1,0)
pid_chassis_move = PID(1.5,5,0,0,1000,1,0)

def chassis_follow_gimbal(keys, yaw):
    move_speed = 150  # 行走速度
    rush_speed = 1000  # 疾跑速度
    speed_tr = move_speed  # 默认平移速度等于行走速度
    x, y, z = 0, 0, 0  # 底盘运动三维速度
    tr_dir = 0  # 底盘坐标系的运动方向

    if 16 in keys:  # shift键
        speed_tr = rush_speed
    else:
        speed_tr = move_speed

    if (87 in keys):  # W键
        x = speed_tr
    elif (83 in keys):  # S键
        x = -1 * speed_tr
    else:
        x = 0

    if (68 in keys):  # D键
        y = speed_tr
    elif (65 in keys):  # A键
        y = -1 * speed_tr
    else:
        y = 0

    # 将云台坐标系下运动方向转化为底盘坐标系下运动方向
    if y == 0:
```

```
        tr_dir = yaw
    elif x == 0:
        tr_dir = yaw + 90
    else:
        tr_dir = yaw + 45

    z = pid_chassis_stop.update(yaw) if (x==0 and y==0) else
pid_chassis_move.update(yaw)  # 根据云台与底盘夹角比例控制Z轴
    x *= math.cos(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算X方向速度分量
    y *= math.sin(math.radians(tr_dir))  # 根据底盘坐标系运动方向计算Y方向速度分量

    cos45 = math.cos(math.radians(45))
    send_and_recv(sock_ctrl, "chassis wheel w1 {0} w2 {1} w3 {2} w4
{3};".format(-y*cos45+x*cos45-z*cos45,

y*cos45+x*cos45+z*cos45,

-y*cos45+x*cos45+z*cos45,

y*cos45+x*cos45-z*cos45))

# 创建两个锁对象
lock = threading.Lock()
send_recv_lock = threading.Lock()

# 全局定义
mk_list = []
block_skill = 0
std_pitch = 0
std_yaw = 0
is_blocked_blaster = False
is_blocked_man_drive_chassis = False
is_blocked_man_drive_gimbal = False

# 等待EP启动完成
# for i in range(20,0,-1):
#     print("open sdk in "+str(i)+" seconds...")
#     time.sleep(1)

# 与机器人控制命令端口建立 TCP 连接 （用于发送控制命令）
sock_ctrl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Connecting sock_ctrl...")
sock_ctrl.connect(address_ctrl)
print("Connected!")

# 与机器人消息推送端口建立 UDP 连接 （用于接收推送信息）
sock_push = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("Connecting sock_push...")
sock_push.bind(address_push)
print("Connected!")

# 线程1：接收数据、手动控制
def thread1_func():
    global mk_list  #图传识别信息数组
    global block_skill  #特定技能的编号
    global std_pitch    #云台当前pitch轴
    global std_yaw  #云台当前yaw轴
    global is_blocked_blaster   #是否关闭发射器发射权限
```

```
    global is_blocked_man_drive_chassis #是否关闭底盘跟随云台权限
    global is_blocked_man_drive_gimbal  #是否关闭鼠标控制云台权限
    # 初始化6个数据对象
    data1 = Data("game msg push [0, 6, 0, 0, 0, 12, 0];")
    data0 = data1
    gb_data1 = GimbalData("gimbal push attitude 0 0;")
    gb_data0 = gb_data1
    ai_data1 = AIData("")
    ai_data0 = ai_data1
    #初始化PID
    basePidPitch = PID(400,5,3, 540,540, 0.03,1)
    basePidYaw   = PID(500,5,3, 540,540, 0.03,1)
    # 初始化鼠标运动卡尔曼
    kalx = Kalmentfilter(2, 4)
    kaly = Kalmentfilter(2, 4)
    lost_msg_count = 0    # 统计未收到信息的次数
    while True:
        # 接收信息推送端口
        buf, _ = sock_push.recvfrom(1024)
        buf = buf.decode('utf-8')
        # print("buf:"+buf)
        # 循环内重申两个上一帧数据对象
        data0 = data1
        gb_data0 = gb_data1
        ai_data0 = ai_data1
        proYaw = 0
        proPitch = 0
        # 解析接收到的数据
        if buf.startswith("game"):  # 如果接收到的是键值数据
            lost_msg_count = 0
            data1 = Data(buf)
            # 将鼠标移动加速度数据转化为正负数
            data1.mouse_x = data1.mouse_x - 255 if data1.mouse_x > 125 else
data1.mouse_x
            data1.mouse_y = data1.mouse_y - 255 if data1.mouse_y > 125 else
data1.mouse_y
            # 均值滤波
            data1.mouse_x = (data1.mouse_x + data0.mouse_x) / 2
            data1.mouse_y = (data1.mouse_y + data0.mouse_y) / 2
            # # 卡尔曼滤波
            # data1.mouse_x = kalx.update(data1.mouse_x)
            # data1.mouse_y = kaly.update(data1.mouse_y)
            # 打印键值数据
            data1.print_data()
            # 按键技能部分
            if 9 in data1.keys and 9 not in data0.keys:   #TAB键     云台旋转180°
                block_skill = 1
            if 70 in data1.keys and 70 not in data0.keys:   #F键     每隔0.3秒射一次
                block_skill = 2 if not block_skill == 2 else 0
            if 71 in data1.keys and 71 not in data0.keys:     #E键     能量机关
                block_skill = 3 if not block_skill == 3 else 0
            if data1.mouse_press == 1:
                print("mouse right")
                if len(ai_data1.marker_list) >= 1:
                    for mark in ai_data1.marker_list:
                        if mark[0]==27:
                            proYaw = basePidYaw.update(mark[1]-0.5)
                            proPitch = basePidPitch.update(0.5-mark[2])
```

```
                            print("proYaw:",proYaw,"proPitch:",proPitch)
                            break
                else:
                    proPitch = 0
                    proYaw = 0
                # 鼠标控制云台运动
                if is_blocked_man_drive_gimbal == False:
                    send_and_recv(sock_ctrl, "gimbal speed p {0} y {1}
wait_for_complete false;".format(data1.mouse_y * 22 + int(proPitch),
data1.mouse_x * 16 + int(proYaw)))
            elif buf.startswith('gimbal'):  # 如果接收到的是云台姿态数据
                gb_data1= GimbalData(buf)
                std_pitch,std_yaw = gb_data1.pitch,gb_data1.yaw
                # 打印云台姿态数据
                gb_data1.print_data()
                # 键盘控制底盘及底盘跟随云台
                if is_blocked_man_drive_chassis == False:
                    chassis_follow_gimbal(data1.keys, gb_data1.yaw)
            elif buf.startswith('AI push'):  # 如果接收到的是视觉标签数据
                ai_data1 = AIData(buf)
                ai_data1.print_marker()
                mk_list = ai_data1.marker_list
            # # 每次循环都递增，若收到键值信息会归零
            # lost_msg_count += 1 if lost_msg_count < 30 else 0
            # # 若连续3次未收到键值信息，则锁定云台、底盘的手操
            # if lost_msg_count > 3:
            #     is_blocked_man_drive_chassis = True   #锁定底盘手动遥控
            #     is_blocked_man_drive_gimbal = True    #锁定云台手动遥控
            #     keep_still()
            # elif lost_msg_count < 3:     # 若收到键值信息且lost_flag为True（已经锁定底
盘、云台），则解锁，重置lost_flag
            #     is_blocked_man_drive_chassis = False   #锁定底盘手动遥控
            #     is_blocked_man_drive_gimbal = False    #锁定云台手动遥控
# 线程2：阻塞技能
def thread2_func():
    while True:
        # 获取队列中的数据
        global mk_list
        global block_skill
        global std_pitch
        global std_yaw
        global is_blocked_man_drive_chassis
        global is_blocked_man_drive_gimbal
        global is_blocked_blaster
        if block_skill == 1:    # 暴风赤红经典转身
            is_blocked_man_drive_gimbal = True
            # time.sleep(0.1)
            send_and_recv(sock_ctrl, "gimbal speed p 0 y 450 wait_for_complete
false;")
            time.sleep(0.4)
            send_and_recv(sock_ctrl, "gimbal speed p 0 y 0 wait_for_complete
false;")
            is_blocked_man_drive_gimbal = False
            block_skill = 0
        if block_skill == 2:    # 每隔0.3秒发射一次
            while(block_skill == 2):
                send_and_recv(sock_ctrl, "blaster fire;")
                time.sleep(0.3)
```

```
        if block_skill == 3:    # 能量机关
            is_blocked_man_drive_chassis = True   #锁定底盘手动遥控
            is_blocked_man_drive_gimbal = True    #锁定云台手动遥控
            keep_still()    # 保持底盘、云台静止
            while(block_skill == 3):
                if len(mk_list) >= 5:   #识别到符合要求的标签
                    send_and_recv(sock_ctrl, "gimbal moveto p 0 y 0 vp 180 vy
180 wait_for_complete false;",True)
                    time.sleep(0.3)
                    atk = power(mk_list,0,0,block_skill)    #核心算法和击打操作
                    if atk == 0:
                        send_and_recv(sock_ctrl, "gimbal moveto y 0 vy 540
wait_for_complete false;",True)
                        is_blocked_man_drive_chassis = False   #解锁底盘手动遥控
                        is_blocked_man_drive_gimbal = False    #解锁云台手动遥控
                        block_skill = 0     # 复位技能标识变量
                        break
                time.sleep(0.1)     #必须加，不然多线程处理不过来
            is_blocked_man_drive_chassis = False   #锁定底盘手动遥控
            is_blocked_man_drive_gimbal = False    #锁定云台手动遥控
        time.sleep(0.1)     #必须加，不然多线程处理不过来
# 线程3：UI控制
def thread3_func():
    pass

# 发送初始化控制命令给机器人
send_and_recv(sock_ctrl, "command;")    #进入sdk模式
send_and_recv(sock_ctrl, "quit;")       #退出sdk模式
send_and_recv(sock_ctrl, "command;")    #sdk再重进是为了解决图传被卡住的bug（DJI的锅）
send_and_recv(sock_ctrl, "robot mode free;")
send_and_recv(sock_ctrl, "game_msg on;")    #打开键值推送
send_and_recv(sock_ctrl, "blaster bead 1;")     #设置单次发射数量

def main():
    send_and_recv(sock_ctrl, "AI push marker on;",True)  #打开视觉标签推送
    send_and_recv(sock_ctrl, "gimbal push attitude on;")    #打开云台姿态信息推送

    # 创建线程对象
    thread1 = threading.Thread(target=thread1_func)
    thread2 = threading.Thread(target=thread2_func)
    thread3 = threading.Thread(target=thread3_func)
    # 启动线程
    thread1.start()
    thread2.start()
    # thread3.start()

    # 等待所有线程完成
    thread1.join()
    thread2.join()
    # thread3.join()

while True:
    try:
        main()
    except Exception as e:
        print(f"Caught exception: {e}")
```