

Table of Contents

章节1:文档阅读指引

PSDK开发者手册	1.1
-----------	-----

章节2:安全须知

章节3:基本介绍

章节4:选型指南

章节5:快速入门

开发须知	5.1
运行示例程序	5.2
RTOS示例程序	5.2.1
Linux示例程序	5.2.2

章节6:功能集合

[基础功能]	6.1
消息订阅	6.1.1
自定义控件	6.1.2
云台功能	6.1.3
基础相机功能	6.1.4
[高级功能]	6.2

章节7:API手册

SDK内核	7.1
ziyan_core	7.1.1
ziyan_error	7.1.2
ziyan_typedef	7.1.3
ziyan_version	7.1.4
基础功能	7.2
高级功能	7.3

Test

fffff

[TOC]

Chapter 1: Getting Started

内容...

Chapter 2: Advanced Topics

内容...

使用PSDK开发负载设备前，请认真阅读本文档中的内容，避免操作不当意外损毁无人机、硬件平台或负载设备。

开发与测试

使用PSDK开发负载设备时，为保护开发者免受意外，请注意如下事项：

- 在使用PSDK开发负载设备或测试基于PSDK开发的负载设备时，请取下无人机上的桨叶。
- 无人机电机在转动时，请勿靠近。
- 请勿向无人机电源输出接口输入大功率电流。

免责声明

- 在使用基于PSDK开发的负载设备前，请检查使用负载设备所在地区的法律法规。**因使用PSDK引发的安全问题或法律纠纷与ZIYAN无关，ZIYAN不承担任何连带责任。**
- 使用基于PSDK开发的负载设备执行飞行任务时，请按照用户手册中的说明操作无人机，确保飞机在执行任务时处于良好的飞行状态，降低无人机潜在的安全隐患和出现安全事故的可能性。
- **ZIYAN不会以任何方式和任何理由获取第三方用户的私人数据。** 开发者应开发出**保护用户隐私数据的功能，例如数据加密和混淆等**。因使用ZIYAN PSDK造成用户隐私数据泄露的事件，ZIYAN将不负任何法律责任。

在注册PSDK企业账号后，请下载PSDK提供的示例代码并在示例代码中补充应用信息，通过编译、调试和烧录等操作获得示例程序。在绑定Ziyan的硬件平台后，即可运行示例程序，借助示例程序了解使用PSDK开发负载设备的方法。

获取示例代码

在注册成为 ZIYAN PSDK 的企业用户后，即可下载 Ziyan PSDK 开发包，在 PSDK 开发包中获取提供的示例代码，借助示例代码了解使用 PSDK 开发负载设备的方法，使用示例代码快速开发出功能完善的出负载设备。

创建负载应用

获取使用 PSDK 开发负载产品的权限后，请在用户中心创建负载应用，获取应用 ID 和应用秘钥，如 图. 填写应用信息 所示。

注意：为提高您的开发效率，请在示例代码中**正确**填写应用的名称、ID、Key和开发者账号，否则编译后的示例程序将无法正常运行。

运行 RTOS 示例代码

说明：本文以STM32F4 Discovery 开发板为例，介绍运行 RTOS 示例代码的步骤和方法。

烧录 Bootloader

1. 使用 Keil MDK IDE 打开位于 `sample/sample_c/platform/rtos_freertos/stm32f4_discovery/project/mdk_bootloader/` 目录下的工程文件 `mdk_bootloader.uvprojx`。
2. 使用 Keil MDK IDE 编译工程为示例程序。
3. 将编译后的示例程序**烧录**到负载设备中（如 STM32F4_discovery 开发板）。

相关参考

- 实现 Bootloader： `platform/rtos_freertos/stm32f4_discovery/bootloader`
- Bootloader 工程目录： `platform/rtos_freertos/stm32f4_discovery/project/mdk_bootloader`

补充应用信息

1. 使用 Keil IDE 打开位于 `sample/sample_c/platform/rtos_freertos/stm32f4_discovery/project/mdk/` 目录下的工程文件 `mdk_app.uvprojx`。
2. 在 `sample/sample_c/platform/rtos_freertos/application/dji_sdk_app_info.h` 文件中补充应用的名称、ID、Key、License、开发者账号和指定波特率。

```
#define USER_APP_NAME          "your_app_name"
#define USER_APP_ID            "your_app_id"
#define USER_APP_KEY           "your_app_key"
#define USER_APP_LICENSE       "your_app_license"
#define USER_DEVELOPER_ACCOUNT "your_developer_account"
#define USER_BAUD_RATE         "921600"
```

编译并烧录

- 使用 Keil MDK IDE 编译示例代码为示例程序。
- 编译示例代码后，将编译后的程序**烧录**到负载设备中（如 STM32F4_discovery 开发板）。
- 如需调试示例程序，请将串口调试工具的波特率设置为：`921600`。

运行 Linux 示例代码

说明：本文以Nvidia Jetson Nano为例，介绍运行Linux示例代码的步骤和方法。

补充应用信息

- 在 `samples/sample_c/platform/linux/nvidia_jetson_nano/application/dji_sdk_app_info.h` 文件中替换应用的名称、ID、Key、License、开发者账号和指定波特率。

```
#define USER_APP_NAME          "your_app_name"
#define USER_APP_ID            "your_app_id"
#define USER_APP_KEY           "your_app_key"
#define USER_APP_LICENSE       "your_app_license"
#define USER_DEVELOPER_ACCOUNT "your_developer_account"
#define USER_BAUD_RATE         "921600"
```

- 在 `samples/sample_c/platform/linux/nvidia_jetson_nano/hal/hal_uart.h` 文件的 `LINUX_UART_DEV1` 和 `LINUX_UART_DEV2` 宏中填写对应的串口名称。

```
#define LINUX_UART_DEV1        "/dev/your_com"
#define LINUX_UART_DEV2        "/dev/your_com"
```

- 通过 `ifconfig` 命令，查看当前与无人机通讯的网口设备名称，并填写到 `samples/sample_c/platform/linux/nvidia_jetson_nano/hal/hal_network.h` 文件的 `LINUX_NETWORK_DEV` 宏中。

```
#define LINUX_NETWORK_DEV      "your_network_name"
```

编译示例程序

- 编译示例代码

进入示例代码工程的根目录下：`Payload-SDK/`，使用如下命令将示例代码编译为示例程序。

```
1. mkdir build
2. cd build
3. cmake ..
4. make
```

根目录位置说明：

```
Payload-SDK/
├─ CMakeLists.txt
├─ EULA.txt
├─ LICENSE.txt
├─ README.md
├─ build/
├─ doc/
├─ psdk_lib/
├─ samples/
└─ tools/
```

- 执行 C 语言示例程序

- 进入示例程序的目录：`cd build/bin/`
- 使用 `sudo ./dji_sdk_demo_linux` 命令运行示例程序

- 执行 C++ 语言示例程序

- 进入示例程序的目录：`cd build/bin/`
- 使用 `sudo ./dji_sdk_demo_linux_cxx` 命令运行示例程序

消息订阅

概述

PSDK 的信息管理功能包含信息获取和消息订阅功能，基于 PSDK 开发的负载设备具有信息获取功能，能够主动获取到飞行器的型号、负载设备挂载的位置以及用户使用的移动端 App 等信息，加载不同的配置文件，方便用户使用负载设备。具有消息订阅功能的负载设备，能够记录用户订阅的数据信息，方便用户实现更广泛的应用。

基础概念

信息获取

信息获取是指负载设备能够**主动获取并记录**飞行器上如飞行器型号、硬件平台类型和负载设备挂载位置等数据信息。

说明：将使用 PSDK 开发的负载设备安装到飞行器上，在开机初始化 5s 后才能够获取到飞行器正确的数据信息。

使用 PSDK 开发的负载设备在初始化后，即可获取到如下信息：

- 基本信息：飞行器型号、硬件平台类型和负载挂载位置
- 移动端 App 信息：App 的系统语言和 App 的屏幕类型

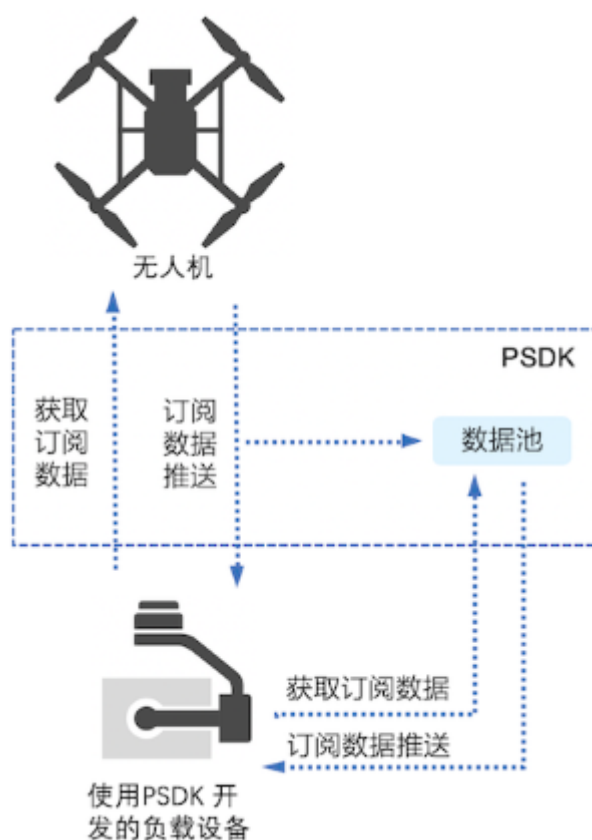
消息订阅

飞行器上的各个部件根据飞行器实际的飞行状况，会实时产生大量的数据信息并被飞行器推送给其他模块，用户使用具有消息订阅功能的负载设备，能够指定所需订阅的数据信息。

订阅流程

订阅数据项后，负载设备即可获得订阅的信息，具体流程如图 1. 消息订阅 所示。

图1: 消息订阅



订阅项

使用 PSDK 消息订阅功能可订阅的数据信息如表 1. 飞行器订阅项 所示。

注意：如果需要在代码工程中搜索订阅项，请替换下的 *_ 为 ZIYAN_SUBSCRIPTION_TOPIC_，譬如 *_QUATERNION 拓展为 ZIYAN_SUBSCRIPTION_TOPIC_QUATERNION。

表1: 飞行器订阅项

数据订阅 TOPIC	影S3	F15
姿态四元数 *_QUATERNION	最大 50Hz	最大 50Hz
相对地面加速度 *_ACCELERATION_GROUND	最大 50Hz	最大 50Hz
相对机体加速度 *_ACCELERATION_BODY	最大 50Hz	最大 50Hz
原始加速度 *_ACCELERATION_RAW	最大 50Hz	最大 50Hz
速度 *_VELOCITY	最大 50Hz	最大 50Hz
融合角速度 *_ANGULAR_RATE_FUSIONED	最大 50Hz	最大 50Hz
原始角速度 *_ANGULAR_RATE_RAW	最大 50Hz	最大 50Hz
融合高度 *_ALTITUDE_FUSED	最大 50Hz	最大 50Hz
气压计高度 *_ALTITUDE_BAROMETER	最大 50Hz	最大 50Hz
Home 点高度 *_ALTITUDE_OF_HOMEPOINT	最大 1Hz	最大 10Hz
融合相对地面高度 *_HEIGHT_FUSION	最大 10Hz	最大 50Hz
相对地面高度 *_HEIGHT_RELATIVE	最大 50Hz	-
融合位置坐标 *_POSITION_FUSED	最大 50Hz	-
GPS 日期（年月日） *_GPS_DATE	最大 5Hz	最大 50Hz
GPS 时间（时分秒） *_GPS_TIME	最大 5Hz	最大 50Hz
GPS 位置 *_GPS_POSITION	最大 5Hz	最大 50Hz
GPS 速度 *_GPS_VELOCITY	最大 5Hz	最大 50Hz
GPS 信息 *_GPS_DETAILS	最大 5Hz	最大 50Hz
GPS 信号强度 *_GPS_SIGNAL_LEVEL	最大 50Hz	最大 50Hz
RTK 位置 *_RTK_POSITION	最大 5Hz	最大 50Hz
RTK 速度 *_RTK_VELOCITY	最大 5Hz	最大 50Hz
RTK 航向角 *_RTK_YAW	最大 5Hz	最大 50Hz
RTK 位置信息 *_RTK_POSITION_INFO	最大 5Hz	最大 50Hz
RTK 航向信息 *_RTK_YAW_INFO	最大 5Hz	最大 50Hz
指南针信息 *_COMPASS	最大 20Hz	最大 50Hz

数据订阅 TOPIC	影S3	F15
遥控摇杆信息 *_RC	最大 50Hz	最大 50Hz
飞行状态 *_STATUS_FLIGHT	最大 50Hz	最大 50Hz
飞行模式状态 *_STATUS_DISPLAYMODE	最大 50Hz	最大 50Hz
带标记遥控遥感信息 *_RC_WITH_FLAG_DATA	最大 50Hz	最大 50Hz
RTK 连接状态 *_RTK_CONNECT_STATUS	最大 50Hz	最大 50Hz
飞行异常信息 *_FLIGHT_ANOMALY	最大 50Hz	最大 50Hz
避障数据 *_AVOID_DATA	最大 100Hz	最大 50Hz
返航点设置状态 *_HOME_POINT_SET_STATUS	最大 50Hz	最大 50Hz
返航点信息 *_HOME_POINT_INFO	最大 50Hz	最大 50Hz

订阅规则

- 订阅项（Topic）支持的数据订阅频率范围有：1Hz，5Hz，10Hz，50Hz，100Hz，200Hz，400Hz，每个订阅项的订阅频率范围不完全相同，每个订阅项不支持重复订阅。
- 指定订阅频率时，任何参数的订阅频率不能小于或等于 0，相同订阅频率的主题的数据长度总和须小于或等于 242。

注意：在 ZIYAN Assistant 2 中使用模拟器模拟负载设备的工作状态时，模拟器将 **无法获取** GPS 信息和 RTK 信息等传感器原始数据，但开发者可订阅如融合位置、融合海拔高度或相对高度等融合数据。

使用消息订阅功能

PSDK 支持通过注册回调和接口调用两种方式订阅飞行器对外推送的数据信息：

- 通过调用 `ZiyanSubscription_GetLatestValueOfTopic()` 获取飞行器最新产生的订阅项的数据信息及其对应的时间。
- 通过调用 `ZiyanSubscription_SubscribeTopic()` 接口指定订阅频率和订阅项，通过构造并注册回调函数，获取飞行器最新产生的订阅项的数据信息及其对应的时间。

说明：使用订阅功能将接收到订阅项的数据与该数据产生时飞行器系统的时间，该时间**暂不支持**与负载设备上的时间实现同步。M300 和 M350 机型得到的是飞行器时间，其它机型得到的是 PSDK 本地时间。

消息订阅功能模块初始化

使用 PSDK 开发的负载设备如需订阅飞行器上的状态信息，需要先调用 `ZiyanSubscription_Init()` 初始化消息订阅模块。

```
ziyanStat = ZiyanSubscription_Init();
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("init data subscription module error.");
    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}
```

通过构造回调函数获取飞行器上的信息

1. 构造回调函数
通过构造回调函数接收飞行器推送的信息。
注意：为避免出现内存踩踏事件，须将数据地址的类型强制转换为订阅项数据结构中的指针类型。

```
static T_ZiyanReturnCode ZiyanTest_SubscriptionReceiveQuaternionCallback(const uint8_t *data, uint16_t dataSize, const T_ZiyanSubscriptionQuaternion *quaternion)
{
    T_ZiyanSubscriptionQuaternion *quaternion = (T_ZiyanSubscriptionQuaternion *) data;
    ziyan_f64_t pitch, yaw, roll;

    USER_UTIL_UNUSED(dataSize);

    pitch = (ziyan_f64_t) asinf(-2 * quaternion->q1 * quaternion->q3 + 2 * quaternion->q0 * quaternion->q2) * 57.3;
    roll = (ziyan_f64_t) atan2f(2 * quaternion->q2 * quaternion->q3 + 2 * quaternion->q0 * quaternion->q1, -2 * quaternion->q1 * quaternion->q3 - 2 * quaternion->q0 * quaternion->q2);
    yaw = (ziyan_f64_t) atan2f(2 * quaternion->q1 * quaternion->q2 + 2 * quaternion->q0 * quaternion->q3, -2 * quaternion->q1 * quaternion->q3 - 2 * quaternion->q0 * quaternion->q2);

    if (s_userSubscriptionDataShow == true) {
        USER_LOG_INFO("receive quaternion data.");

        USER_LOG_INFO("timestamp: millisecond %u microsecond %u.", timestamp->millisecond, timestamp->microsecond);
        USER_LOG_INFO("quaternion: %f %f %f %f.\r\n", quaternion->q0, quaternion->q1, quaternion->q2, quaternion->q3);
        USER_LOG_INFO("euler angles: pitch = %.2f roll = %.2f yaw = %.2f.", pitch, yaw, roll);
        ZiyanTest_WidgetLogAppend("pitch = %.2f roll = %.2f yaw = %.2f.", pitch, yaw, roll);
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

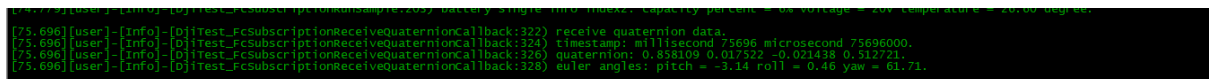
1. 注册回调函数

注册回调函数接收无人机产生并对外推送的数据信息，下述代码以 1Hz 的频率订阅无人机“无人机飞行速度”和“无人机 GPS 坐标”，如图. 订阅结果 (1) 所示。

说明：使用订阅功能订阅无人机上的数据信息时，订阅频率只能为“最大订阅频率”的约数。

```
ziyanStat = ZiyanSubscription_SubscribeTopic(ZIYAN_SUBSCRIPTION_TOPIC_QUATERNION, ZIYAN_DATA_SUBSCRIPTION_TOPIC_1_HZ,
                                             ZiyanTest_SubscriptionReceiveQuaternionCallback);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Subscribe topic quaternion error.");
    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}
```

图. 订阅结果 (1)



```
[75.696] (user) [Info] [0]Test_FcSubscriptionReceiveQuaternionCallback(322) receive quaternion data.
[75.696] (user) [Info] [0]Test_FcSubscriptionReceiveQuaternionCallback(324) timestamp: millisecond 75696 microsecond 75696000.
[75.696] (user) [Info] [0]Test_FcSubscriptionReceiveQuaternionCallback(326) quaternion: 0.858109 0.017522 -0.021438 0.512721.
[75.696] (user) [Info] [0]Test_FcSubscriptionReceiveQuaternionCallback(328) euler angles: pitch = -3.14 roll = 0.46 yaw = 61.71.
```

在线程函数中获取无人机上的信息

非回调方式发起订阅，回调参数设置成NULL。

```
ziyanStat = ZiyanSubscription_SubscribeTopic(ZIYAN_SUBSCRIPTION_TOPIC_VELOCITY, ZIYAN_DATA_SUBSCRIPTION_TOPIC_1_HZ,
                                             NULL);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Subscribe topic velocity error.");
    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}

ziyanStat = ZiyanSubscription_SubscribeTopic(ZIYAN_SUBSCRIPTION_TOPIC_GPS_POSITION, ZIYAN_DATA_SUBSCRIPTION_TOPIC_1_HZ,
                                             NULL);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Subscribe topic gps position error.");
    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
}
```

通过数据订阅线程函数获取飞行器推送的信息并打印在终端上。下述代码以 1Hz 的频率，订阅飞行器最新产生的飞行器飞行速度和飞行器 GPS 坐标，以及该数据对应的时间，如图. 订阅结果 (2) 所示。

```

ziyanStat = Ziyansubscription_GetLatestValueOfTopic(ZIYAN_SUBSCRIPTION_TOPIC_VELOCITY,
                                                    (uint8_t *) &velocity,
                                                    sizeof(T_ZiyansubscriptionVelocity),
                                                    &timestamp);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get value of topic velocity error.");
} else {
    USER_LOG_INFO("velocity: x = %f y = %f z = %f healthFlag = %d.", velocity.data.x, velocity.data.y,
                  velocity.data.z, velocity.health);
}

ziyanStat = Ziyansubscription_GetLatestValueOfTopic(ZIYAN_SUBSCRIPTION_TOPIC_GPS_POSITION,
                                                    (uint8_t *) &gpsPosition,
                                                    sizeof(T_ZiyansubscriptionGpsPosition),
                                                    &timestamp);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("get value of topic gps position error.");
} else {
    USER_LOG_INFO("gps position: x = %d y = %d z = %d.", gpsPosition.x, gpsPosition.y, gpsPosition.z);
}

```

图. 订阅结果 (2)

```

[74.739][user]-[Info]-[OjTest_FcSubscriptionRunSample:163] velocity: x = -0.000444 y = 0.001184 z = 0.000243 healthFlag = 1, timestamp ms = 74739 us = 74739000.
[74.739][user]-[Info]-[OjTest_FcSubscriptionRunSample:173] gps position: x = 0 y = 0 z = 214000.
[74.758][user]-[Info]-[OjTest_FcSubscriptionRunSample:189] battery single info index1: capacity percent = 94% voltage = 25V temperature = 28.80 degree.
[74.768][user]-[Info]-[UserFcSubscription_Task:267] velocity: x -0.000444 y 0.001184 z 0.000243, healthFlag 1.
[74.768][user]-[Info]-[UserFcSubscription_Task:279] gps position: x 0 y 0 z 214000.
[74.768][user]-[Info]-[UserFcSubscription_Task:294] gps total satellite number used: 0 0 0.
[74.779][user]-[Info]-[OjTest_FcSubscriptionRunSample:203] battery single info index2: capacity percent = 6% voltage = 20V temperature = 26.60 degree.

```

概述

"自定义控件"是一个将"负载设备的功能"封装为按钮、开关以及范围条等控件的功能。使用Ziyan Pilot或基于GSDK开发的移动端App能够识别负载设备中控件的配置信息并生成UI控件，方便用户快速设置负载设备的参数并控制负载设备执行指定的动作。同时，Ziyan Pilot或基于GSDK开发的移动端App还能以浮窗的形式显示负载设备的状态信息。此外，用户还能根据使用需求，将负载设备的功能映射到遥控器上的预留按键上，通过使用遥控器上的预留按键，以更便捷的方式控制负载设备。

基础概念

控件

主界面的控件

- 动作栏控件：动作栏支持按钮、开关、范围条和选择列表四个控件类型，最多支持设置8个自定义控件。
- 浮窗：实时显示负载设备的状态信息。

图1. 主界面

配置界面的控件

用户在配置界面能够操作配置界面中的控件，如按钮、开关、范围条、选择列表、文本输入框、整型值输入框，如 图2. 配置界面 所示。

图2. 配置界面

控件配置文件

说明

- 控件配置文件的路径：`sample/sample_c/module_sample/widget/widget_file`
- Ziyan Pilot系统语言为中文时，控件配置文件为 `cn_big_screen`
- Ziyan Pilot系统语言为英文时，控件配置文件为 `en_big_screen`
- 不同语言下的配置信息如控件编号、数量与类型等需要保持一致。

注意：控件配置文件包含静态配置文件和控件UI 图标，建议先在静态配置文件中配置控件属性，再设计控件图标。

配置控件属性

`widget_config.json` 是一个用于配置控件静态属性的文件，修改 `widget_config.json` 文件时，请务必严格遵守JSON的语法规则，否则配置文件将无法使用。

提示：

- JSON文件中的配置项被包裹在一个{}中，通过key-value的方式表达数据；
- JSON的Key 必须包裹在一个双引号中，请勿丢失Key 值的双引号；
- JSON的值只支持数字（含浮点数和整数）、字符串、Bool值(如true和false)、数组(需要包裹在[]中)和对象(需要包裹在{}中)

注意：下述代码中 “//” 后的内容为代码注释，在实际的JSON配置文件请勿添加该内容。

```

{
  "version": { // 自定义控件配置文件版本,用户不可更改此版本号
    "major" : 1,
    "minor" : 0
  },
  "main_interface": { // ZIYAN Pilot 主界面控件设置
    "floating_window": { // 浮窗配置
      "is_enable": true // 浮窗是否显示, true : 显示, false : 隐藏
    },
    "widget_list": [ // 主界面动作栏控件列表
      {
        "widget_index": 0, // 控件编号
        "widget_type": "button", // 控件类型, 主界面Action控件支持 "button" : 按钮, "switch" : 开关, "range" : 范围
        "widget_name": "Button_1", // 控件名称
        "icon_file_set": { // 控件图标文件集
          "icon_file_name_selected" : "icon_button1.png", // 选中状态下控件图标文件名称
          "icon_file_name_unselected" : "icon_button1.png" // 未选中状态下控件图标文件名称
        }
      },
      {
        "widget_index": 1,
        "widget_type": "button",
        "widget_name": "Button_2",
        "icon_file_set": {
          "icon_file_name_selected" : "icon_button2.png",
          "icon_file_name_unselected" : "icon_button2.png"
        }
      },
      {
        "widget_index": 2,
        "widget_type": "list",
        "widget_name": "List",
        "list_item": [
          {
            "item_name": "Item_1",
            "icon_file_set": {
              "icon_file_name_selected" : "icon_list_item1.png",
              "icon_file_name_unselected" : "icon_list_item1.png"
            }
          },
          {
            "item_name": "Item_2",
            "icon_file_set": {
              "icon_file_name_selected" : "icon_list_item2.png",
              "icon_file_name_unselected" : "icon_list_item2.png"
            }
          }
        ]
      },
      {
        "widget_index": 3,
        "widget_type": "switch",
        "widget_name": "Switch",
        "icon_file_set": {
          "icon_file_name_selected" : "icon_switch_select.png",
          "icon_file_name_unselected" : "icon_switch_unselect.png"
        }
      },
      {
        "widget_index": 4,
        "widget_type": "scale",
        "widget_name": "Scale",
        "icon_file_set": {
          "icon_file_name_selected" : "icon_scale.png",
          "icon_file_name_unselected" : "icon_scale.png"
        }
      }
    ],
    "config_interface": {
      "text_input_box": { // 文本输入框
        "widget_name": "TextInputBox", // 文本输入框名称
        "placeholder_text": "Please input message", // 文本输入框占位符文本
        "is_enable": false // 文本输入框是否显示, false : 不显示, true : 显示
      },
      "widget_list": [

```

```
{
  "widget_index": 5,
  "widget_type": "button",
  "widget_name": "Button 5"
},
{
  "widget_index": 6,
  "widget_type": "scale",
  "widget_name": "Scale 6"
},
{
  "widget_index": 7,
  "widget_type": "int_input_box",
  "widget_name": "Integer Input Box 7",
  "int_input_box_hint": "unit:s"
},
{
  "widget_index": 8,
  "widget_type": "switch",
  "widget_name": "Switch 8"
},
{
  "widget_index": 9,
  "widget_type": "list",
  "widget_name": "List 9",
  "list_item": [
    {
      "item_name": "Item 1"
    },
    {
      "item_name": "Item 2"
    },
    {
      "item_name": "Item 3"
    },
    {
      "item_name": "Item 4"
    }
  ]
}
]
```

RTOS 配置文件转换

RTOS 系统不支持文件系统，请按如下步骤将配置文件转换为 .h 类型的文件：

- 使用 `tools/file2c` 目录下的工具 `file2c`，将**所有的**自定义控件配置文件生成 `.h` 头文件，详细说明请参见目录下 `Readme.txt` 文件；
- `.h` 头文件在 `sample/api_sample/widget/widget_file_c` 目录下；
- 调用接口 `ZiyanWidget_RegDefaultUiConfigByBinaryArray` 和 `ZiyanWidget_RegUiConfigByBinaryArray` 调用控件配置文件。

配置参考文件

- 中文配置文件：file_binary_array_list_cn.h 和 file_binary_array_list_cn.c
- 英文配置文件：file_binary_array_list_en.h 和 file_binary_array_list_en.c

使用自定义控件功能

开发加载设备的自定义控件功能，需要先完成自定义控件的初始化，获取控件配置文件所在的目录，配置控件在不同系统语言下显示的配置文件，通过设置控件处理函数列表，应用自定义控件功能，最终实现自定义控件功能。

1. 控件初始化

使用“自定义控件”功能前，需要使用如下代码初始化负载设备的控件。

```
ZiyanStat = ZiyanWidget_Init();
if (ZiyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Ziyan test widget init error, stat = 0x%08llx", ziyanStat);
    return ZiyanStat;
}
```


2. 设置控件配置信息

在Linux 和RTOS 系统上开发负载设备时，需要设置控件的配置信息，如控件默认的配置文件和不同系统语言对应的控件配置文件。确保Ziyan Pilot 能够获取控件的配置信息并正确地显示在Ziyan Pilot上。

说明： 不同系统语言下的控件配置文件中的配置项如控件编号、控件数量与控件类型等需要保持一致。

设置负载设备的控件参数（Linux）

```
//Step 2 : Set UI Config (Linux environment)
char curFileDirPath[WIDGET_DIR_PATH_LEN_MAX];
char tempPath[WIDGET_DIR_PATH_LEN_MAX];
ziyanStat = ZiyanUserUtil_GetCurrentFileDirPath(__FILE__, WIDGET_DIR_PATH_LEN_MAX, curFileDirPath);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Get file current path error, stat = 0x%08llx", ziyanStat);
    return ziyanStat;
}

if (s_isWidgetFileDirPathConfigured == true) {
    snprintf(tempPath, WIDGET_DIR_PATH_LEN_MAX, "%swidget_file/en_big_screen", s_widgetFileDirPath);
} else {
    snprintf(tempPath, WIDGET_DIR_PATH_LEN_MAX, "%swidget_file/en_big_screen", curFileDirPath);
}

//set default ui config path
ziyanStat = ZiyanWidget_RegDefaultUiConfigByDirPath(tempPath);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Add default widget ui config error, stat = 0x%08llx", ziyanStat);
    return ziyanStat;
}

//set ui config for English language
ziyanStat = ZiyanWidget_RegUiConfigByDirPath(ZIYAN_MOBILE_APP_LANGUAGE_ENGLISH,
                                              ZIYAN_MOBILE_APP_SCREEN_TYPE_BIG_SCREEN,
                                              tempPath);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Add widget ui config error, stat = 0x%08llx", ziyanStat);
    return ziyanStat;
}

//set ui config for Chinese language
if (s_isWidgetFileDirPathConfigured == true) {
    snprintf(tempPath, WIDGET_DIR_PATH_LEN_MAX, "%swidget_file/cn_big_screen", s_widgetFileDirPath);
} else {
    snprintf(tempPath, WIDGET_DIR_PATH_LEN_MAX, "%swidget_file/cn_big_screen", curFileDirPath);
}

ziyanStat = ZiyanWidget_RegUiConfigByDirPath(ZIYAN_MOBILE_APP_LANGUAGE_CHINESE,
                                              ZIYAN_MOBILE_APP_SCREEN_TYPE_BIG_SCREEN,
                                              tempPath);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Add widget ui config error, stat = 0x%08llx", ziyanStat);
    return ziyanStat;
}
}
```

设置负载设备的控件参数（RTOS）

```
//Step 2 : Set UI Config (RTOS environment)
T_ZiyanWidgetBinaryArrayConfig enWidgetBinaryArrayConfig = {
    .binaryArrayCount = g_EnBinaryArrayCount,
    .fileBinaryArrayList = g_EnFileBinaryArrayList
};

//set default ui config
ziyanStat = ZiyanWidget_RegDefaultUiConfigByBinaryArray(&enWidgetBinaryArrayConfig);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("Add default widget ui config error, stat = 0x%08llx", ziyanStat);
    return ziyanStat;
}
}
```

3. 注册控件处理函数列表

通过 `ZiyanWidget_RegHandlerList` 接口注册负载设备的某项功能和对应的控件参数，将负载设备的功能绑定到指定的控件上。

```
ziyanStat = ZiyWidget_RegHandlerList(s_widgetHandlerList, s_widgetHandlerListCount);  
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {  
    USER_LOG_ERROR("Set widget handler list error, stat = 0x%08llx", ziyStat);  
    return ziyStat;  
}
```

说明

- 仅使用SkyPort 开发的负载设备支持使用PSDK 的云台功能。
- X-Port 支持开发者使用ZIYAN Assistant 2 (2.0.11 及以上版本) 调整配置参数。

概述

使用PSDK 的“云台控制”功能，开发者需要先设计负载设备的云台并开发出控制云台的程序，将云台的控制函数注册到PSDK 指定的接口后，用户通过使用ZIYAN Pilot、基于GSDK 开发的移动端App 及遥控器即可控制基于PSDK 开发的具有云台功能的负载设备，同时获得负载设备的相关信息，如姿态等。

基础概念

云台状态信息

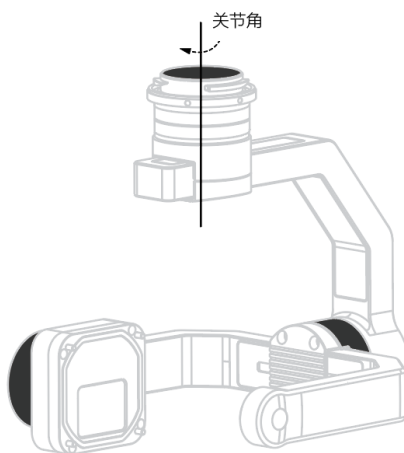
使用PSDK 开发的云台类负载设备需要按照指定的要求上报云台的状态、当前姿态和校准状态等信息，方便用户移动端App 或机载计算机根据云台的状态，实现精准控制。有关获取云台状态的方法和相关详情请参见**PSDK API 文档**。

关节角与姿态角

云台关节与云台关节角

云台的关节如 图1.云台关节 所示，云台关节是云台上带动负载设备转动的结构件：云台电机，云台关节角即云台电机转动的角度。本教程使用机体坐标系描述云台的关节角。

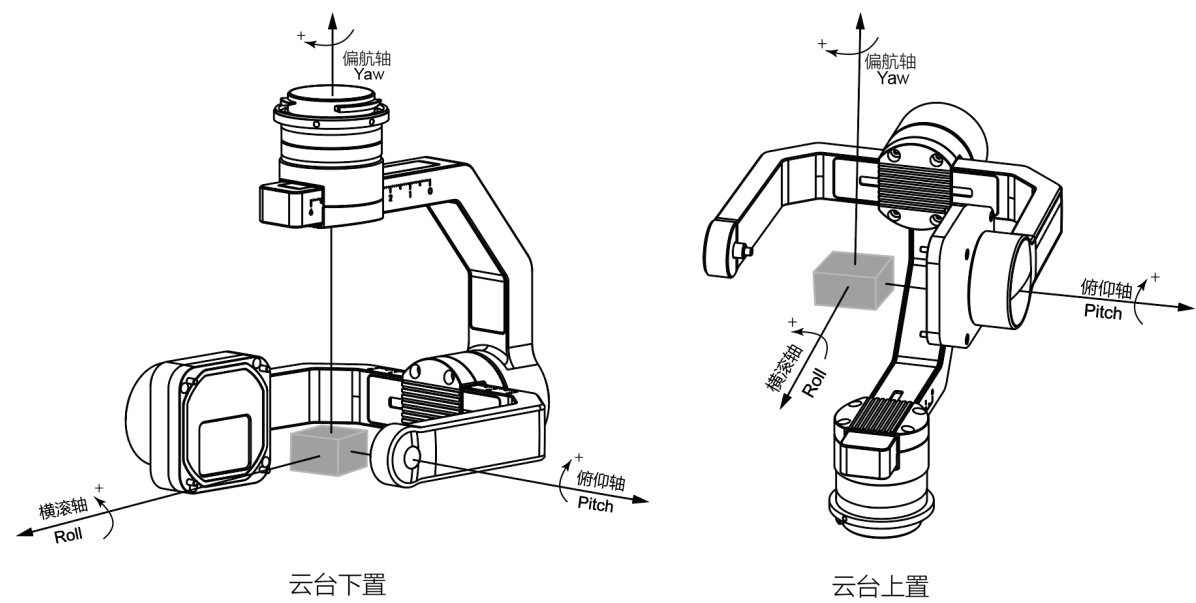
图1.云台关节



云台姿态与云台姿态角

云台的姿态如 图2.云台姿态 所示，根据用户的控制指令，云台能够调整姿态；云台姿态角即使用大地坐标系（NED，北东地坐标系）描述云台上**负载设备**的角度，该角度也称为欧拉角。

图2.云台姿态



云台模式

云台模式决定了云台跟随无人机运动时的转动方式：

- 自由模式：当无人机的姿态改变时，云台将不会转动。
- FPV 模式：当无人机的姿态发生改变时，云台会转动航向轴与横滚轴，确保负载设备当前的视场角不会发生改变。
- YAW 跟随模式：在该模式下，云台的航向轴会跟随无人机的航向轴转动。

说明：在以上三种模式下，无人机系统中的其他模块（航线飞行时的云台控制指令）、遥控器和移动端App，能够控制云台转动。

云台控制

控制方式

云台转动的控制方式分为以下三种：

- 相对角度控制：使用PSDK 开发的云台根据用户指定的角度，在规定的时间内，转动指定的角度。
- 绝对角度控制：使用PSDK 开发的云台根据用户的指令，在规定的时间内，从当前的位置转动到指定的位置。
- 速度控制：用户可控制使用PSDK 开发的云台的转动速度。

说明

- 在角度控制模式下，云台转动的时间受云台最大旋转速度和最大加速度限制，实际的转动角度受云台限位角度的限制。
- 在速度控制模式下，云台根据用户指定的速度转动0.5s，当云台转动到限位角时，将会停止转动。

控制权限

有关云台控制权限的详细说明请参见 表1.云台控制权限。

- 控制规则：
 - 优先级低的控制动作在优先级高的控制动作结束后才能控制云台；
 - 高优先级的控制动作可抢占低优先级控制动作的控制权；
 - 同等优先级的控制场景按照开始控制的时间先后顺序抢占控制权，开始控制时间较晚的控制场景不能夺取控制权。
- 权限释放
 - 控制模块完成对云台的控制后会释放控制权；
 - 若控制模块完成对云台的控制后未释放控制权，使用PSDK 开发的云台将在云台转动结束后指定的时间内自动释放控制权。

表1.云台控制权限

权限角色	权限等级	云台运动持续时间	权限超时释放时间
遥控器云台摇杆	1	0	500 ms
GSDK 云台控制指令	1	由控制命令指定	2000 ms
ZIYAN App 云台控制指令	2	0	500 ms
负载协同指令	2	由控制命令指定	
GSDK 云台速度指令	2	0	500 ms
航线飞行时的云台控制指令	2	由控制命令指定	

平滑度

云台的平滑度是指云台响应动作的缓急，使用PSDK 开发的云台支持用户通过ZIYAN Pilot和基于GSDK 开发的移动端App 通过设置平滑控制系数，实现云台的缓启停。

- 平滑控制系数：平滑控制系数决定云台转动的最大加速度。
- 云台转动的最大加速度= $10000 \times (0.8 \wedge (1 + X)) \text{ deg/s}^2$ (X 为平滑控制系数)

最大速度百分比

- 最大速度百分比：云台的最大速度百分比决定云台旋转的最大速度。
- 云台实际最大的转动速度= 默认最大速度 × 最大速度百分比

说明：开发者根据实际的使用需要，可设置云台默认的最大云台转动运动速度。

角度微调

使用PSDK 开发的云台支持用户通过ZIYAN Pilot和基于GSDK 开发的移动端App，精细化地调整云台关节角的角度，调整结果还可作为校准参数存储在负载设备中，用于降低云台的各类误差。

云台限位功能

为避免云台在工作时，因结构干涉导致云台意外损坏或干扰无人机的执行飞行任务，请务必为云台设置机械限位和软件限位。

- 机械限位：机械限位由云台类负载设备的物理形态和设计结构决定，详情请参见[标准声明](#)。
- 软件限位：开发者可根据实际的使用需求设置软件限位：
 - 设置云台俯仰轴、横滚轴和航向轴的欧拉角角度限制；
 - 设置俯仰轴欧拉角扩展角角度限制；
 - 设置云台关节角限制。

提示：

- 使用俯仰轴角度范围扩展功能后，可将云台俯仰轴的欧拉角角度限制设置为默认限制或扩展限制。
- 当云台的关节角达到限位时，使用ZIYAN Pilot 以及基于GSDK 开发的移动端App 将接收到云台转动到限位角的提示信息。

云台复位

使用PSDK 开发的云台类负载设备支持用户通过ZIYAN Pilot和基于GSDK 开发的移动端App 复位云台，将云台的姿态复位为初始状态。

- 航向轴复位：将云台航向轴的角度复位为无人机航向轴角度与云台航向轴微调角度的和。
- 俯仰轴与航向轴复位：将云台俯仰轴的角度复位为微调的角度，将云台航向轴的角度复位为无人机航向轴角度与云台航向轴微调角度的和。
- 重置云台的偏航轴和俯仰轴：将云台偏航轴的角度重置为无人机偏航轴和云台微调角度的和。重置云台俯仰轴为-90°与云台微调角度的和（云台下置），90°与云台微调角度的和（云台上置）。
- 重置云台的偏航轴为-90°与云台微调角度的和（云台下置），90°与云台微调角度的和（云台上置）。

实现云台功能

请开发者根据选用的开发平台以及行业应用实际的使用需求，按照PSDK 中的结构体 `T_ZiyangGimbalCommonHandler` 构造实现云台类负载设备控制功能的函数，将云台控制功能的函数注册到PSDK 中指定的接口后，用户通过使用ZIYAN Pilot 或基于GSDK 开发的移动端App 能够控制基于PSDK 开发的云台类负载设备执行指定的动作。

```

s_commonHandler.GetSystemState = GetSystemState;
s_commonHandler.GetAttitudeInformation = GetAttitudeInformation;
s_commonHandler.GetCalibrationState = GetCalibrationState;
s_commonHandler.GetRotationSpeed = GetRotationSpeed;
s_commonHandler.GetJointAngle = GetJointAngle;

s_commonHandler.Rotate = ZiyanTest_GimbalRotate;
s_commonHandler.StartCalibrate = StartCalibrate;
s_commonHandler.SetControllerSmoothFactor = SetControllerSmoothFactor;
s_commonHandler.SetPitchRangeExtensionEnabled = SetPitchRangeExtensionEnabled;
s_commonHandler.SetControllerMaxSpeedPercentage = SetControllerMaxSpeedPercentage;
s_commonHandler.RestoreFactorySettings = RestoreFactorySettings;
s_commonHandler.SetMode = SetMode;
s_commonHandler.Reset = Reset;
s_commonHandler.FineTuneAngle = FineTuneAngle;

```

使用云台控制功能

使用云台控制功能，需要先实现云台控制功能，再实现云台限位功能，根据云台模式调整云台的姿态、目标角度和限位标志，最后实现云台校准功能校准云台。

说明：使用PSDK 开发负载设备的云台功能时，请使用SkyPort V2 或 SkyPort ，若您所用的开发工具为X-Port，请阅读[X-Port 功能](#)。

使用云台管理功能

1. 云台控制功能模块初始化

使用“云台控制”功能前，需要先初始化云台控制功能模块，确保云台控制功能可正常运行。

```

ziyanStat = ZiyanGimbal_Init();
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("init gimbal module error: 0x%08lX", ziyanStat);
}

```

2. 注册云台控制功能

使用PSDK 的云台控制功能控制云台类负载设备时，开发者需要将控制云台的函数注册到指定的接口中。

```

ziyanStat = ZiyanGimbal_RegCommonHandler(&s_commonHandler);
if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("gimbal register common handler error: 0x%08lX", ziyanStat);
}

```

3. 获取云台的状态信息

为方便用户控制云台执行相应的动作，需调用 GetSystemState 接口获取云台的状态。

```

static T_ZiyanReturnCode GetSystemState(T_ZiyanGimbalSystemState *systemState)
{
    T_ZiyanOsalHandler *osalHandler = ZiyanPlatform_GetOsalHandler();

    if (osalHandler->MutexLock(s_commonMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex lock error");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    *systemState = s_systemState;

    if (osalHandler->MutexUnlock(s_commonMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex unlock error");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

4. 构造回调函数计算云台的转动速度

构造回调函数计算云台的转动速度、调整云台的姿态并记录云台转动的目标角度和转动速度。

```

T_ZiyanReturnCode ZiyanTest_GimbalRotate(E_ZiyanGimbalRotationMode rotationMode,
                                          T_ZiyanGimbalRotationProperty rotationProperty,
                                          T_ZiyanAttitude3d rotationValue)
{
    T_ZiyanReturnCode ziyanStat;
    T_ZiyanReturnCode returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    T_ZiyanAttitude3d targetAttitudeDTemp = {0};
    T_ZiyanAttitude3f targetAttitudeFTemp = {0};
    T_ZiyanAttitude3d speedTemp = {0};
    T_ZiyanOsahandler *osalHandler = ZiyanPlatform_GetOsahandler();

    USER_LOG_DEBUG("gimbal rotation value invalid flag: pitch %d, roll %d, yaw %d.",
                    rotationProperty.rotationValueInvalidFlag.pitch,
                    rotationProperty.rotationValueInvalidFlag.roll,
                    rotationProperty.rotationValueInvalidFlag.yaw);

    if (osalHandler->MutexLock(s_attitudeMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex lock error");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    if (osalHandler->MutexLock(s_commonMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex lock error");
        returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
        goto out2;
    }

    switch (rotationMode) {
        case ZIYAN_GIMBAL_ROTATION_MODE_RELATIVE_ANGLE:
            USER_LOG_INFO("gimbal relative rotate angle: pitch %d, roll %d, yaw %d.", rotationValue.pitch,
                           rotationValue.roll, rotationValue.yaw);
            USER_LOG_DEBUG("gimbal relative rotate action time: %d.",
                            rotationProperty.relativeAngleRotation.actionTime);

            if (s_rotatingFlag == true) {
                USER_LOG_WARN("gimbal is rotating.");
                goto out1;
            }

            targetAttitudeDTemp.pitch =
                rotationProperty.rotationValueInvalidFlag.pitch == true ? s_attitudeInformation.attitude.pitch : (
                    s_attitudeInformation.attitude.pitch + rotationValue.pitch);
            targetAttitudeDTemp.roll =
                rotationProperty.rotationValueInvalidFlag.roll == true ? s_attitudeInformation.attitude.roll : (
                    s_attitudeInformation.attitude.roll + rotationValue.roll);
            targetAttitudeDTemp.yaw =
                rotationProperty.rotationValueInvalidFlag.yaw == true ? s_attitudeInformation.attitude.yaw : (
                    s_attitudeInformation.attitude.yaw + rotationValue.yaw);

            targetAttitudeFTemp.pitch = targetAttitudeDTemp.pitch;
            targetAttitudeFTemp.roll = targetAttitudeDTemp.roll;
            targetAttitudeFTemp.yaw = targetAttitudeDTemp.yaw;
            ZiyanTest_GimbalAngleLegalization(&targetAttitudeFTemp, s_aircraftAttitude, NULL);
            targetAttitudeDTemp.pitch = targetAttitudeFTemp.pitch;
            targetAttitudeDTemp.roll = targetAttitudeFTemp.roll;
            targetAttitudeDTemp.yaw = targetAttitudeFTemp.yaw;

            s_targetAttitude = targetAttitudeDTemp;
            s_rotatingFlag = true;
            s_controlType = TEST_GIMBAL_CONTROL_TYPE_ANGLE;

            ziyanStat = ZiyanTest_GimbalCalculateSpeed(s_attitudeInformation.attitude, s_targetAttitude,
                                                       rotationProperty.relativeAngleRotation.actionTime, &s_speed);
            if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
                USER_LOG_ERROR("calculate gimbal rotation speed error: 0x%08lX.", ziyanStat);
                returnCode = ziyanStat;
                goto out1;
            }

            break;
        case ZIYAN_GIMBAL_ROTATION_MODE_ABSOLUTE_ANGLE:
            USER_LOG_INFO("gimbal absolute rotate angle: pitch %d, roll %d, yaw %d.", rotationValue.pitch,
                           rotationValue.roll, rotationValue.yaw);
            USER_LOG_DEBUG("gimbal absolute rotate action time: %d.",
                            rotationProperty.absoluteAngleRotation.actionTime);
            if (rotationProperty.absoluteAngleRotation.jointAngleValid) {

```

```

        USER_LOG_DEBUG("gimbal joint angles: pitch %d, roll %d, yaw %d.",
                        rotationProperty.absoluteAngleRotation.jointAngle.pitch,
                        rotationProperty.absoluteAngleRotation.jointAngle.roll,
                        rotationProperty.absoluteAngleRotation.jointAngle.yaw);
    }

    if (s_rotatingFlag == true) {
        USER_LOG_WARN("gimbal is rotating.");
        goto out1;
    }

    targetAttitudeDTemp.pitch =
        rotationProperty.rotationValueInvalidFlag.pitch == true ? s_attitudeInformation.attitude.pitch
        : rotationValue.pitch;
    targetAttitudeDTemp.roll =
        rotationProperty.rotationValueInvalidFlag.roll == true ? s_attitudeInformation.attitude.roll
        : rotationValue.roll;
    targetAttitudeDTemp.yaw =
        rotationProperty.rotationValueInvalidFlag.yaw == true ? s_attitudeInformation.attitude.yaw
        : rotationValue.yaw;

    targetAttitudeFTemp.pitch = targetAttitudeDTemp.pitch;
    targetAttitudeFTemp.roll = targetAttitudeDTemp.roll;
    targetAttitudeFTemp.yaw = targetAttitudeDTemp.yaw;
    ZiyanTest_GimbalAngleLegalization(&targetAttitudeFTemp, s_aircraftAttitude, NULL);
    targetAttitudeDTemp.pitch = targetAttitudeFTemp.pitch;
    targetAttitudeDTemp.roll = targetAttitudeFTemp.roll;
    targetAttitudeDTemp.yaw = targetAttitudeFTemp.yaw;

    s_targetAttitude = targetAttitudeDTemp;
    s_rotatingFlag = true;
    s_controlType = TEST_GIMBAL_CONTROL_TYPE_ANGLE;

    ziyangStat = ZiyanTest_GimbalCalculateSpeed(s_attitudeInformation.attitude, s_targetAttitude,
        rotationProperty.absoluteAngleRotation.actionTime, &s_speed);
    if (ziyangStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("calculate gimbal rotation speed error: 0x%08lX.", ziyangStat);
        returnCode = ziyangStat;
        goto out1;
    }

    break;
case ZIYAN_GIMBAL_ROTATION_MODE_SPEED:
    USER_LOG_INFO("gimbal rotate speed: pitch %d, roll %d, yaw %d.", rotationValue.pitch,
        rotationValue.roll, rotationValue.yaw);

    if (s_rotatingFlag == true && s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
        USER_LOG_WARN("gimbal is rotating.");
        goto out1;
    }

    memcpy(&speedTemp, &rotationValue, sizeof(T_ZiyanAttitude3d));
    ZiyanTest_GimbalSpeedLegalization(&speedTemp);
    s_speed = speedTemp;

    if (rotationValue.pitch != 0 || rotationValue.roll != 0 || rotationValue.yaw != 0) {
        s_rotatingFlag = true;
        s_controlType = TEST_GIMBAL_CONTROL_TYPE_SPEED;
    } else {
        s_rotatingFlag = false;
    }

    break;
default:
    USER_LOG_ERROR("gimbal rotation mode invalid: %d.", rotationMode);
    returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT;
    goto out1;
}

out1:
if (osalHandler->MutexUnlock(s_commonMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("mutex unlock error");
    returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    goto out2;
}

out2:

```



```

    if (osalHandler->MutexUnlock(s_attitudeMutex) != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("mutex unlock error");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_UNKNOWN;
    }

    return returnCode;
}

```

5. 控制云台转动

负载设备根据云台的姿态和转动速度，将相对角度控制量、绝对角度控制量或速度控制量转换为控制云台转动的速度，根据该速度控制云台转动，如图3.云台控制所示。

```

nextAttitude.pitch =
    (float) s_attitudeHighPrecision.pitch + (float) s_speed.pitch / (float) PAYLOAD_GIMBAL_TASK_FREQ;
nextAttitude.roll =
    (float) s_attitudeHighPrecision.roll + (float) s_speed.roll / (float) PAYLOAD_GIMBAL_TASK_FREQ;
nextAttitude.yaw = (float) s_attitudeHighPrecision.yaw + (float) s_speed.yaw / (float) PAYLOAD_GIMBAL_TASK_FREQ;

if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
    nextAttitude.pitch =
        (nextAttitude.pitch - s_targetAttitude.pitch) * s_speed.pitch >= 0 ? s_targetAttitude.pitch
        : nextAttitude.pitch;
    nextAttitude.roll = (nextAttitude.roll - s_targetAttitude.roll) * s_speed.roll >= 0 ? s_targetAttitude.roll
        : nextAttitude.roll;
    nextAttitude.yaw =
        (nextAttitude.yaw - s_targetAttitude.yaw) * s_speed.yaw >= 0 ? s_targetAttitude.yaw : nextAttitude.yaw;
}

ZiyanTest_GimbalAngleLegalization(&nextAttitude, s_aircraftAttitude, &s_attitudeInformation.reachLimitFlag);
s_attitudeInformation.attitude.pitch = nextAttitude.pitch;
s_attitudeInformation.attitude.roll = nextAttitude.roll;
s_attitudeInformation.attitude.yaw = nextAttitude.yaw;

s_attitudeHighPrecision.pitch = nextAttitude.pitch;
s_attitudeHighPrecision.roll = nextAttitude.roll;
s_attitudeHighPrecision.yaw = nextAttitude.yaw;

if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
    if (memcmp(&s_attitudeInformation.attitude, &s_targetAttitude, sizeof(T_ZiyanAttitude3d)) == 0) {
        s_rotatingFlag = false;
    }
} else if (s_controlType == TEST_GIMBAL_CONTROL_TYPE_SPEED) {
    if ((s_attitudeInformation.reachLimitFlag.pitch == true || s_speed.pitch == 0) &&
        (s_attitudeInformation.reachLimitFlag.roll == true || s_speed.roll == 0) &&
        (s_attitudeInformation.reachLimitFlag.yaw == true || s_speed.yaw == 0)) {
        s_rotatingFlag = false;
    }
}

```

图3.云台控制

```

[587.281][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.381][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.482][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.582][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.682][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.782][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.848][module_user]-[Debug]- gimbal rotate speed: pitch -69, roll 0, yaw 0.
[587.858][module_user]-[Debug]- gimbal rotate speed: pitch -171, roll 0, yaw 0.
[587.879][module_user]-[Debug]- gimbal rotate speed: pitch -284, roll 0, yaw 0.
[587.882][module_user]-[Debug]- gimbal attitude: pitch 0, roll 0, yaw 0.
[587.899][module_user]-[Debug]- gimbal rotate speed: pitch -409, roll 0, yaw 0.
[587.900][module_user]-[Debug]- gimbal rotate speed: pitch -556, roll 0, yaw 0.
[587.920][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[587.930][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[587.951][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[587.961][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[587.982][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[587.982][module_user]-[Debug]- gimbal attitude: pitch -28, roll 0, yaw 0.
[588.002][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.002][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.023][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.027][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.038][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.058][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.068][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.076][module_user]-[Debug]- gimbal attitude: pitch -94, roll 0, yaw 0.
[588.089][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.099][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.109][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.130][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.140][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.161][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.171][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.177][module_user]-[Debug]- gimbal attitude: pitch -160, roll 0, yaw 0.
[588.181][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.202][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.212][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.223][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.233][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.243][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.264][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.274][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.277][module_user]-[Debug]- gimbal attitude: pitch -226, roll 0, yaw 0.
[588.294][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.305][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.315][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.335][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.346][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.366][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.376][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.377][module_user]-[Debug]- gimbal attitude: pitch -292, roll 0, yaw 0.
[588.387][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.407][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.418][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.438][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.448][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.459][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.477][module_user]-[Debug]- gimbal attitude: pitch -358, roll 0, yaw 0.
[588.479][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.489][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.510][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.520][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.531][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.551][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.561][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.577][module_user]-[Debug]- gimbal attitude: pitch -424, roll 0, yaw 0.
[588.582][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.592][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.
[588.592][module_user]-[Debug]- gimbal rotate speed: pitch -660, roll 0, yaw 0.

```

调整云台处于不同模式时的参数

开发者能够根据实际的使用需求调整云台处于不同模式时的姿态、目标角度和到达限位标志。

- 在FPV 模式下：
 - 调整云台姿态：将无人机姿态的变化量叠加到云台的横滚轴和航向轴上；
 - 调整云台转动的目标角度：云台旋转时，将无人机的姿态变化量叠加到云台的横滚轴和航向轴上。
- 在YAW 模式下：
 - 调整云台姿态：将无人机姿态的变化量叠加到云台的航向轴上；

- 调整云台转动的目标角度：云台旋转时，将无人机的姿态变化量叠加到云台的航向轴上。
- 调整云台姿态与目标角度的转动范围，负载设备能够计算出云台的达到限位标志。

```
switch (s_systemState.gimbalMode) {
    case PSDK_GIMBAL_MODE_FREE:
        break;
    case PSDK_GIMBAL_MODE_FPV:
        s_attitudeInformation.attitude.roll += (s_aircraftAttitude.roll - s_lastAircraftAttitude.roll);
        s_attitudeInformation.attitude.yaw += (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);

        s_attitudeHighPrecision.roll += (float) (s_aircraftAttitude.roll - s_lastAircraftAttitude.roll);
        s_attitudeHighPrecision.yaw += (float) (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);

        if (s_rotatingFlag == true && s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
            s_targetAttitude.roll += (s_aircraftAttitude.roll - s_lastAircraftAttitude.roll);
            s_targetAttitude.yaw += (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);
        }
        break;
    case PSDK_GIMBAL_MODE_YAW_FOLLOW:
        s_attitudeInformation.attitude.yaw += (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);

        s_attitudeHighPrecision.yaw += (float) (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);

        if (s_rotatingFlag == true && s_controlType == TEST_GIMBAL_CONTROL_TYPE_ANGLE) {
            s_targetAttitude.yaw += (s_aircraftAttitude.yaw - s_lastAircraftAttitude.yaw);
        }
        break;
    default:
        PsdkLogger_UserLogError("gimbal mode invalid: %d.", s_systemState.gimbalMode);
}
s_lastAircraftAttitude = s_aircraftAttitude;

attitudeFTemp.pitch = s_attitudeInformation.attitude.pitch;
attitudeFTemp.roll = s_attitudeInformation.attitude.roll;
attitudeFTemp.yaw = s_attitudeInformation.attitude.yaw;
PsdkTest_GimbalAngleLegalization(&attitudeFTemp, s_aircraftAttitude, &s_attitudeInformation.reachLimitFlag);
s_attitudeInformation.attitude.pitch = attitudeFTemp.pitch;
s_attitudeInformation.attitude.roll = attitudeFTemp.roll;
s_attitudeInformation.attitude.yaw = attitudeFTemp.yaw;

PsdkTest_GimbalAngleLegalization(&s_attitudeHighPrecision, s_aircraftAttitude, NULL);

attitudeFTemp.pitch = s_targetAttitude.pitch;
attitudeFTemp.roll = s_targetAttitude.roll;
attitudeFTemp.yaw = s_targetAttitude.yaw;
PsdkTest_GimbalAngleLegalization(&attitudeFTemp, s_aircraftAttitude, NULL);
s_targetAttitude.pitch = attitudeFTemp.pitch;
s_targetAttitude.roll = attitudeFTemp.roll;
s_targetAttitude.yaw = attitudeFTemp.yaw;
```

使用云台校准功能

为方便用户更加精准地控制云台，建议开发者在开发具有云台功能的负载设备时实现云台校准功能。

1. 使用云台校准功能

如需使用云台校准功能，请先实现云台校准功能，并将云台校准功能的函数，注册到指定的接口中，方便用户使用ZIYAN Pilot 或基于GSDK 开发的移动端App 校准具有云台功能的负载设备。

```

static T_PsdkReturnCode StartCalibrate(void)
{
    T_PsdkReturnCode psdkStat;

    PsdkLogger_UserLogDebug("start calibrate gimbal.");

    psdkStat = PsdkOsal_GetTimeMs(&s_calibrationStartTime);
    if (psdkStat != PSDK_RETURN_CODE_OK) {
        PsdkLogger_UserLogError("get start time error: %lld.", psdkStat);
    }

    if (PsdkOsal_MutexLock(s_calibrationMutex) != PSDK_RETURN_CODE_OK) {
        PsdkLogger_UserLogError("mutex lock error");
        return PSDK_RETURN_CODE_ERR_UNKNOWN;
    }

    s_calibrationState.calibratingFlag = true;
    s_calibrationState.progress = 0;
    s_calibrationState.stage = PSDK_GIMBAL_CALIBRATION_STAGE_PROCESSING;

    if (PsdkOsal_MutexUnlock(s_calibrationMutex) != PSDK_RETURN_CODE_OK) {
        PsdkLogger_UserLogError("mutex unlock error");
        return PSDK_RETURN_CODE_ERR_UNKNOWN;
    }

    return PSDK_RETURN_CODE_OK;
}

```

2. 云台校准状态更新

负载设备执行云台校准功能后，将记录云台的校准状态，基于GSDK 开发的移动端App 能够获取云台的校准状态。

```

// calibration
if (s_calibrationState.calibratingFlag != true)
    goto unlockCalibrationMutex;

progressTemp = (currentTime - s_calibrationStartTime) * 100 / PAYLOAD_GIMBAL_CALIBRATION_TIME_MS;
if (progressTemp >= 100) {
    s_calibrationState.calibratingFlag = false;
    s_calibrationState.currentCalibrationProgress = 100;
    s_calibrationState.currentCalibrationStage = PSDK_GIMBAL_CALIBRATION_STAGE_COMPLETE;
}

unlockCalibrationMutex:
if (PsdkOsal_MutexUnlock(s_calibrationMutex) != PSDK_RETURN_CODE_OK) {
    PsdkLogger_UserLogError("mutex unlock error");
    continue;
}

```

在ZIYAN Pilot 以及基于GSDK 开发的移动端App 中使用“云台自动校准”功能后，使用PSDK 开发的负载设备将接收到云台校准命令并校准云台，如 图4 和图5 所示。

图4.云台校准（1）

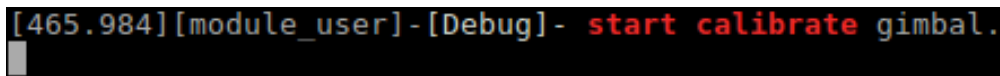
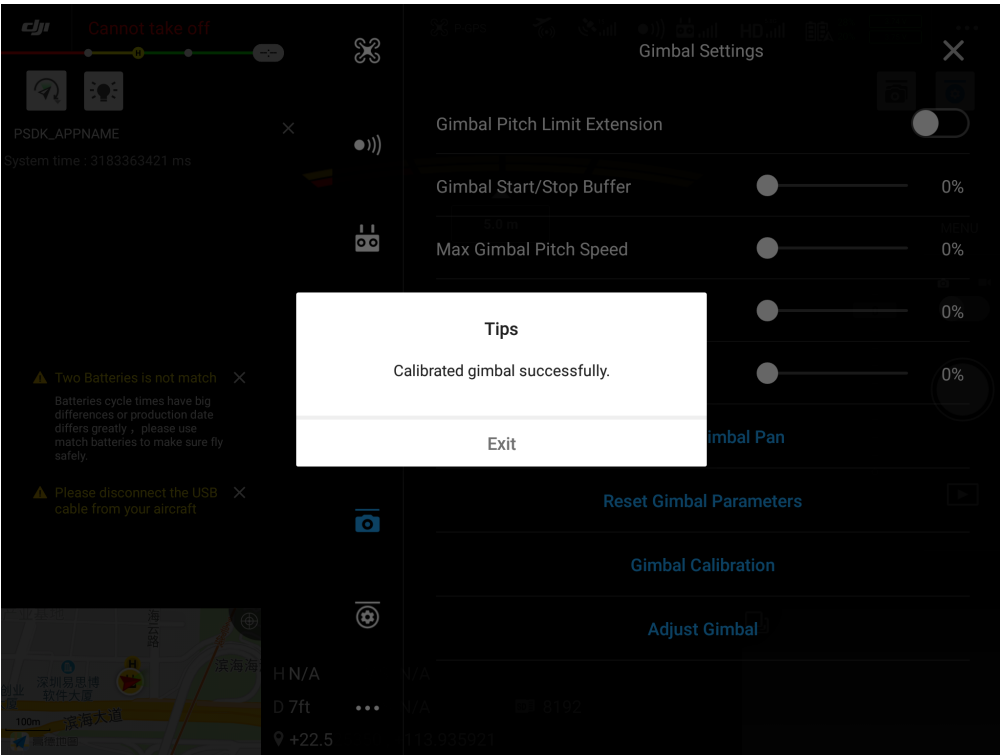


图5.云台校准（2）



概述

为满足开发者对相机类负载设备的控制需求，PSDK 提供了**控制**相机执行拍照、录像、变焦及对焦等功能的接口，开发者需**先实现**相机拍照、录像以及测光等功能，再通过注册 PSDK 相机类的接口，开发出功能完善的相机类负载设备；通过使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App，用户能够控制使用 PSDK 开发的相机类负载设备执行指定的动作，获取负载设备中的信息和资源。

- 基础功能：设置相机模式、拍照、录像、获取相机状态、SD 卡管理
- 高级功能：变焦、测光、对焦、视频流传输、回放下载、媒体库管理

基础概念介绍

相机模式

使用相机类功能前，需要先设置相机类负载设备的模式，不同的模式指定了相机类负载设备在执行某个任务时的工作逻辑。

- 拍照：在该模式下，用户能够触发相机类负载设备拍摄照片。
- 录像：在该模式下，用户能够触发相机类负载设备录制影像。
- 视频回放：在该模式下，用户可以在移动端 App 上播放或下载负载设备上的媒体文件。

注意：相机只能在一种模式中执行相应的操作，如在录像模式下仅能录像无法拍照。

拍照模式

使用 PSDK 开发的相机类负载设备支持以下拍照模式：

- 单拍：下发拍照命令后，相机拍摄单张照片。
- 连拍：下发拍照命令后，相机将连续拍摄指定数量的照片；当前支持 2/3/5/7/10/14 张连拍。
- 定时拍照：下发拍照命令时，相机按照指定的时间间隔，拍摄指定数量的照片。
 - 当前支持 2/3/5/7/10 秒时间间隔；
 - 当前最大支持指定拍摄 254 张照片，当拍照张数为 255 的时候，相机将不间断地拍摄照片。

对焦模式

- 自动对焦：Auto Focus，简称 AF。在自动对焦模式下，相机类负载设备根据光电传感器获取的影像状态（漫发射），计算出对焦结果实现对焦功能，获取清晰的影像。
- 手动对焦：Manual Focus，简称 MF。在手动对焦模式下，用户通过调节对焦环能够获得清晰的影像。

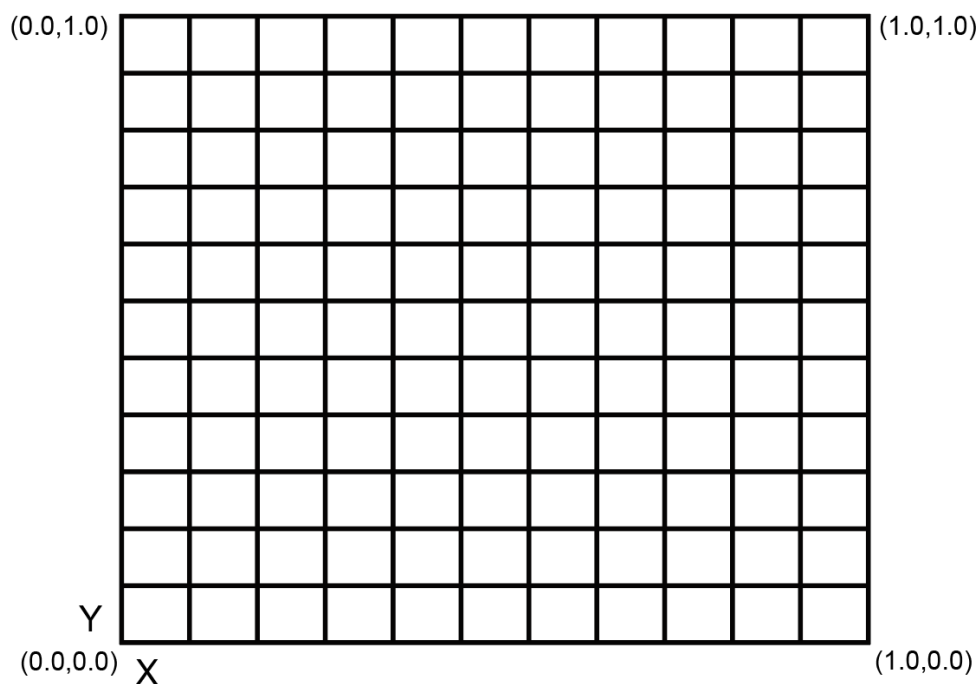
对焦点

说明：相机类负载设备的对焦点默认为传感器的中心位置。

在控制相机类负载设备对焦，需要先设置对焦点，该对焦点的值为当前对焦点在相机画面中的纵横比系数，如 图 1. 对焦点 所示。

- 在自动对焦模式下，开发者需要制定相机对焦的策略，设置对焦点（该对焦点也称“目标点”）。
- 在手动对焦模式下，用户可根据实际需要调整对焦点，获得所需的影像。

图 1. 对焦点



对焦环

使用 PSDK 开发的具有变焦环（光学变焦）的相机类负载设备通过调用 `SetFocusRingValue` 接口，设置对焦环的值：

- 对焦环的值默认为 0，表示无穷大和最接近的可能焦距。
- 对焦环的值不为 0 时，用户可根据相机的实际参数设置对焦环的值。

对焦助手

在 AF 和 MF 模式下，对焦助手通过数字变焦的方式，能够放大用户指定的对焦区域，调用 `SetFocusAssistantSettings` 接口可设置对焦助手的状态，使用对焦助手，用户能够查看当前相机类负载设备的对焦质量。

变焦模式

- 光学变焦，通过改变光学镜头的结构实现变焦，光学变焦倍数越大，能拍摄的景物就越远，反之则近；
- 数码变焦，处理器使用特定的算法，通过改变传感器上每个像素的面积，实现数码变焦；
- 连续变焦，相机类负载设备控制镜头以指定的速度沿指定的方向运动，相机类负载设备先控制镜头执行光学变焦，当光学变焦达到上限后，相机类负载设备再执行数码变焦，以此实现连续变焦的功能。当前变焦倍数 = 当前光学变焦倍数 × 当前数码变焦倍数；
- 指点变焦，用户指定某一目标点后，基于 PSDK 开发的相机类负载设备能够控制云台转动，使指定的目标处于画面中心，控制相机类负载设备按照预设的变焦倍数放大图像。

变焦方向

- ZOOM_IN：变焦倍数减小，图像由远到近
- ZOOM_OUT：变焦倍数增大，图像由近到远

变焦速度

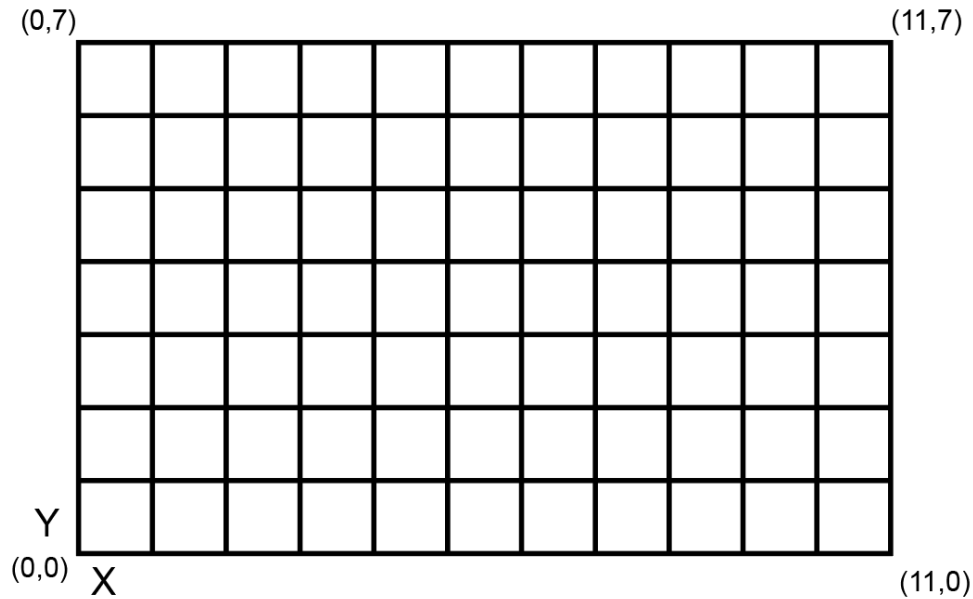
- SLOWEST：以最慢的速度变焦
- SLOW：以较慢的速度变焦
- MODERATELY_SLOW：以比正常速度稍慢的速度变焦
- NORMAL：镜头以正常的速度变焦
- MODERATELY_FAST：以比正常速度稍快的速度变焦
- FAST：以较快的速度变焦
- FASTEST：以最快的速度变焦

测光模式

- 平均测光，通过对画面整体亮度的分析，计算出画面的平均亮度的值，适合光照均匀的拍照场景；
- 中央重点测光，仅对图像传感器中间区域测光，适合拍摄框架式构图的照片；

- 点测光，对在以“指定的点”为中心的范围内测光，如 图 1. 点测光区域 所示，通过该方式能获得准确的测光结果，确保指定的对象能够曝光正确，适合光照复杂的拍摄场景。

图 2. 点测光区域



图像传感器被分为 12 列 8 行定义的 96 个点区域。行索引范围是[0,7]，其中值在图像上从上到下递增；列索引范围是[0, 11]，其中值从左到右增加。

变焦模式

- 光学变焦，通过改变光学镜头的结构实现变焦，光学变焦倍数越大，能拍摄的景物就越远，反之则近；
- 数码变焦，处理器使用特定的算法，通过改变传感器上每个像素的面积，实现数码变焦；
- 连续变焦，相机类负载设备控制镜头以指定的速度沿指定的方向运动，相机类负载设备先控制镜头执行光学变焦，当光学变焦达到上限后，相机类负载设备再执行数码变焦，以此实现连续变焦的功能。当前变焦倍数 = 当前光学变焦倍数 × 当前数码变焦倍数；
- 指点变焦，用户指定某一目标点后，基于 PSDK 开发的相机类负载设备能够控制云台转动，使指定的目标处于画面中心，控制相机类负载设备按照预设的变焦倍数放大图像。

变焦方向

- ZOOM_IN：变焦倍数减小，图像由远到近
- ZOOM_OUT：变焦倍数增大，图像由近到远

变焦速度

- SLOWEST：以最慢的速度变焦
- SLOW：以较慢的速度变焦
- MODERATELY_SLOW：以比正常速度稍慢的速度变焦
- NORMAL：镜头以正常的速度变焦
- MODERATELY_FAST：以比正常速度稍快的速度变焦
- FAST：以较快的速度变焦
- FASTEST：以最快的速度变焦

媒体文件管理

使用 PSDK 开发的相机类负载设备能够根据用户的指令，执行文件删除或下载等操作。

媒体文件预览功能

使用 PSDK 开发的相机类负载设备支持用户使用 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 预览负载设备中的媒体文件。

- 静态预览：预览单个文件或文件列表
 - 缩略图，预览文件列表
 - 图像：负载设备按照文件的原始比例生成缩略图，请将预览图的宽度设置为 100 像素
 - 视频：截取视频某一帧的画面
 - 截屏图，预览单个文件
 - 图像：按原始比例，建议缩放图像成宽为 600 像素的预览图
 - 视频：截取视频某一帧的画面

- 原始文件，如需获得相机类负载设备中原始的媒体文件，请使用下载功能获取指定的媒体文件。
- 动态预览（视频预览）：播放、暂停、停止、跳转（快进、快退和进度拖动）

说明：支持动态预览的文件格式：MP4、JPG、DNG 和 MOV，编码格式请参见[视频标准](#)。

实现相机类基础功能

请开发者根据选用的**开发平台**以及行业应用实际的使用需求，按照 PSDK 中的结构体 `T_ZiyanCameraCommonHandler` 构造实现相机类负载设备设置相机模式、拍照和录像等功能的函数，将相机功能的函数注册到 PSDK 中指定的接口后，用户通过使用 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 能够控制基于 PSDK 开发的相机类负载设备执行相应的动作。

```
// 获取负载设备系统当前的状态
s_commonHandler.GetSystemState = GetSystemState;
// 实现设置相机类负载设备模式的功能
s_commonHandler.SetMode = SetMode;
s_commonHandler.GetMode = ZiyanTest_CameraGetMode;
// 实现开始或停止录像的功能
s_commonHandler.StartRecordVideo = StartRecordVideo;
s_commonHandler.StopRecordVideo = StopRecordVideo;
// 实现开始或停止拍照的功能
s_commonHandler.StartShootPhoto = StartShootPhoto;
s_commonHandler.StopShootPhoto = StopShootPhoto;
// 实现设置相机类负载设备的拍照功能
s_commonHandler.SetShootPhotoMode = SetShootPhotoMode;
s_commonHandler.GetShootPhotoMode = GetShootPhotoMode;
s_commonHandler.SetPhotoBurstCount = SetPhotoBurstCount;
s_commonHandler.GetPhotoBurstCount = GetPhotoBurstCount;
s_commonHandler.SetPhotoTimeIntervalSettings = SetPhotoTimeIntervalSettings;
s_commonHandler.GetPhotoTimeIntervalSettings = GetPhotoTimeIntervalSettings;
// 实现 SD 卡管理功能
s_commonHandler.GetSDCardState = GetSDCardState;
s_commonHandler.FormatSDCard = FormatSDCard;
```

基础功能初始化

使用 PSDK 开发负载设备的相机功能时，必须要初始化相机模块并注册相机类的功能。

相机类功能模块初始化

在使用相机类功能前，必须先调用接口 `ZiyanPayloadCamera_Init` 初始化相机类负载设备，确保相机类负载设备可正常工作。

```
T_PsdkReturnCode returnCode;

returnCode = ZiyanPayloadCamera_Init();
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("payload camera init error:0x%08llx", returnCode);
}
```

注册相机类基础功能

开发者实现相机类负载设备设置相机模式、拍照和录像等功能后，需要通过 `ZiyanPayloadCamera_RegCommonHandler` 注册相机类基础功能。

```
returnCode = ZiyanPayloadCamera_RegCommonHandler(&s_commonHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register common handler error:0x%08llx", returnCode);
}
```

使用 SD 卡管理功能

说明：

- 使用 X-Port 开发的相机类负载设备需要实现存储媒体文件的功能并将该功能注册到 PSDK 指定的接口中。X-Port 上的存储卡仅支持存放 X-Port 日志信息，无法存储负载设备产生的媒体文件。
- 本教程以模拟SD卡管理功能为例，介绍使用 PSDK SD 卡管理功能的使用方法，如需开发具有 SD 卡管理功能的负载设备，请调用负载设备系统的接口实现 SD 卡管理功能。

SD 卡模块初始化

使用 SD 卡管理功能，需要开发者先开发并注册操作 SD 卡功能的函数，通过初始化 SD 卡管理模块，获取 SD 卡的状态信息。

```

/* Init the SDCard parameters */
s_cameraSDCardState.isInserted = true;
s_cameraSDCardState.isVerified = true;
s_cameraSDCardState.totalSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
s_cameraSDCardState.remainSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
s_cameraSDCardState.availableCaptureCount = SDCARD_TOTAL_SPACE_IN_MB / SDCARD_PER_PHOTO_SPACE_IN_MB;
s_cameraSDCardState.availableRecordingTimeInSeconds = SDCARD_TOTAL_SPACE_IN_MB / SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;

```

获取 SD 卡的当前状态

基于 PSDK 开发的负载设备控制程序调用 `GetSDCardState` 接口能够获取负载设备上 SD 卡当前的状态，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的 APP 能够查看负载设备中 SD 卡的状态信息。

```

// 预估可拍照张数和可录像时长的功能。
if (s_cameraState.isRecording) {
    s_cameraState.currentVideoRecordingTimeInSeconds++;
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;
    if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
        s_cameraSDCardState.remainSpaceInMB = 0;
        s_cameraSDCardState.isFull = true;
    }
}
// 获取 SD 卡的状态
static T_ZiyanReturnCode GetSDCardState(T_ZiyanCameraSDCardState *sdCardState)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    memcpy(sdCardState, &s_cameraSDCardState, sizeof(T_ZiyanCameraSDCardState));

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

使用 SD 卡格式化功能

基于 PSDK 开发的负载设备控制程序调用 `FormatSDCard` 接口能够控制负载设备执行 SD 卡格式化，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的 APP 可获取负载设备中 SD 卡的状态信息并控制负载设备执行 SD 卡格式化功能，如图 1. SD 卡管理功能 所示。

```

static T_ZiyanReturnCode FormatSDCard(void)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("format sdcard");

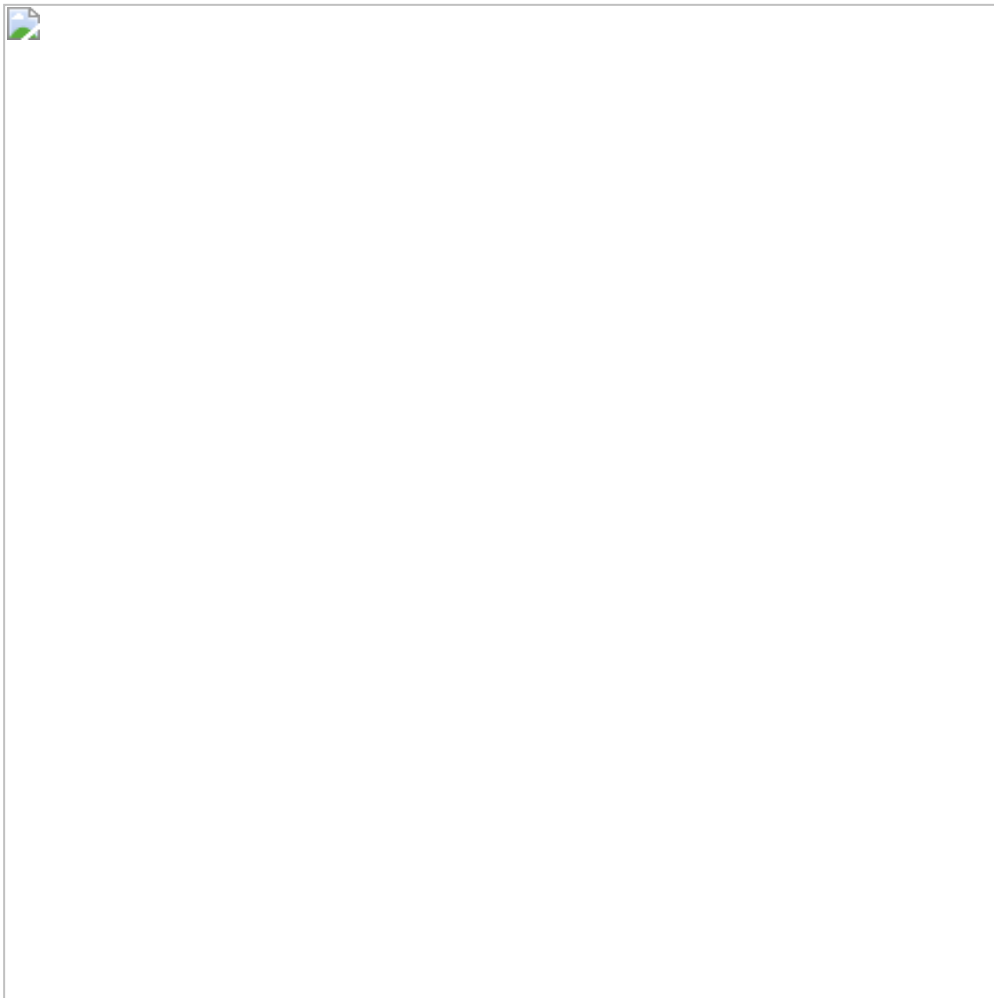
    memset(&s_cameraSDCardState, 0, sizeof(T_ZiyanCameraSDCardState));
    s_cameraSDCardState.isInserted = true;
    s_cameraSDCardState.isVerified = true;
    s_cameraSDCardState.totalSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
    s_cameraSDCardState.remainSpaceInMB = SDCARD_TOTAL_SPACE_IN_MB;
    s_cameraSDCardState.availableCaptureCount = SDCARD_TOTAL_SPACE_IN_MB / SDCARD_PER_PHOTO_SPACE_IN_MB;
    s_cameraSDCardState.availableRecordingTimeInSeconds =
        SDCARD_TOTAL_SPACE_IN_MB / SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

图 3. SD 卡管理功能



使用模式设置功能

基于 PSDK 开发的负载设备控制程序调用 SetMode 和 GetMode 接口能够设置相机的模式，用户使用 ZIYAN Pilot 能够切换相机类负载设备的工作模式，如图 2. 设置相机模式 所示。

```
static T_ZiyanReturnCode GetSystemState(T_ZiyanCameraSystemState *systemState)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *systemState = s_cameraState;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode SetMode(E_ZiyanCameraMode mode)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraState.cameraMode = mode;
    USER_LOG_INFO("set camera mode:%d", mode);

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

T_ZiyanReturnCode ZiyanTest_CameraGetMode(E_ZiyanCameraMode *mode)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

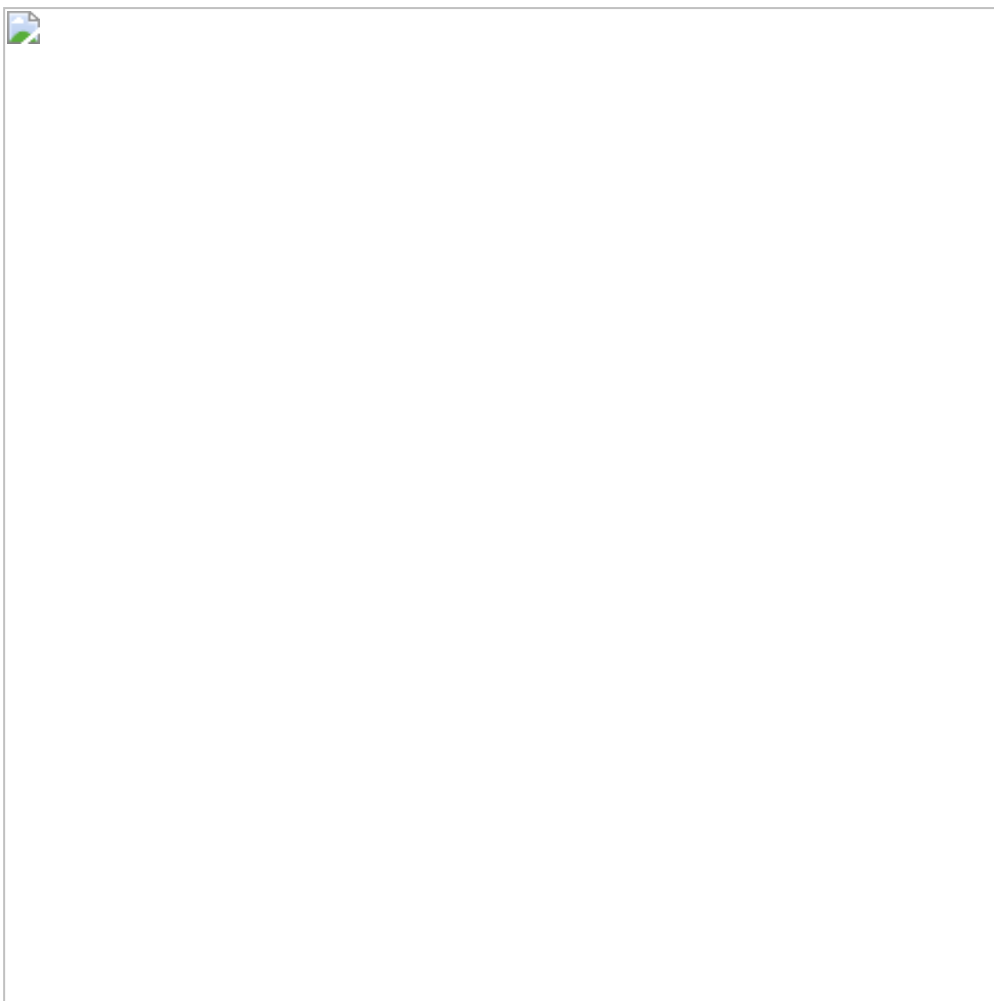
    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *mode = s_cameraState.cameraMode;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

图 4. 设置相机类负载设备的模式



使用拍照功能

说明：

- 使用拍照功能前，用户需要在 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 上将相机类负载设备的工作模式设置为拍照模式。
- 使用 PSDK 开发的负载设备在拍照时，会向 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 返回拍照状态（用于如触发移动端 App 拍照声音等功能）。

设置相机类负载设备的拍照模式

基于 PSDK 开发的负载设备控制程序调用 `SetShootPhotoMode` 和 `GetShootPhotoMode` 接口能够设置并获取相机类负载设备的模式，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可设置并获取相机类负载设备的拍照模式。

```

static T_ZiyanReturnCode SetShootPhotoMode(E_ZiyanCameraShootPhotoMode mode)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraShootPhotoMode = mode;
    USER_LOG_INFO("set shoot photo mode:%d", mode);

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetShootPhotoMode(E_ZiyanCameraShootPhotoMode *mode)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *mode = s_cameraShootPhotoMode;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

控制相机单拍

基于 PSDK 开发的负载设备控制程序调用 StartShootPhoto 和 StopShootPhoto 接口控制相机类负载设备拍摄单张照片，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可控制相机类负载设备拍摄单张照片。

```

static T_ZiyanReturnCode StartShootPhoto(void)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("start shoot photo");
    s_cameraState.isStoring = true;

    if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_SINGLE) {
        s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_SINGLE_PHOTO;
    } else if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_BURST) {
        s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_BURST_PHOTO;
    } else if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
        s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_INTERVAL_PHOTO;
        s_cameraState.isShootingIntervalStart = true;
        s_cameraState.currentPhotoShootingIntervalTimeInSeconds = s_cameraPhotoTimeIntervalSettings.timeIntervalSeconds;
    }

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode StopShootPhoto(void)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("stop shoot photo");
    s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_PHOTO_IDLE;
    s_cameraState.isStoring = false;
    s_cameraState.isShootingIntervalStart = false;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

控制相机连拍

基于 PSDK 开发的负载设备控制程序调用 SetPhotoBurstCount 和 GetPhotoBurstCount 接口控制相机类负载设备连拍，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可设置相机类负载设备的连拍张数，控制相机类负载设备拍摄指定数量的照片。

```

static T_ZiyanReturnCode SetPhotoBurstCount(E_ZiyanCameraBurstCount burstCount)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraBurstCount = burstCount;
    USER_LOG_INFO("set photo burst count:%d", burstCount);

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetPhotoBurstCount(E_ZiyanCameraBurstCount *burstCount)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *burstCount = s_cameraBurstCount;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

控制相机定时拍照

基于 PSDK 开发的负载设备控制程序调用 SetPhotoTimeIntervalSettings 和 GetPhotoTimeIntervalSettings 接口控制相机类负载设备定时拍照，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可设置相机类负载设备的拍照间隔，控制相机类负载设备按照指定的时间间隔拍摄照片。


```

static T_ZiyanReturnCode SetPhotoTimeIntervalSettings(T_ZiyanCameraPhotoTimeIntervalSettings settings)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_cameraPhotoTimeIntervalSettings.captureCount = settings.captureCount;
    s_cameraPhotoTimeIntervalSettings.timeIntervalSeconds = settings.timeIntervalSeconds;
    USER_LOG_INFO("set photo interval settings count:%d seconds:%d", settings.captureCount,
        settings.timeIntervalSeconds);
    s_cameraState.currentPhotoShootingIntervalCount = settings.captureCount;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetPhotoTimeIntervalSettings(T_ZiyanCameraPhotoTimeIntervalSettings *settings)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    memcpy(settings, &s_cameraPhotoTimeIntervalSettings, sizeof(T_ZiyanCameraPhotoTimeIntervalSettings));

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

拍照状态管理

在 ZIYAN Pilot 以及使用 GSDK 开发的移动端 App 中点击“拍照”按钮后，使用 PSDK 开发的相机类负载设备在自定义时间内（如 0.5s）在线程中执行拍照、照片存储和内存状态更新的操作。

确认拍照状态

使用 PSDK 开发的相机类负载设备在执行完拍照动作后，需要获取负载设备的拍照状态。

```

if (s_cameraState.shootingState != PSDK_CAMERA_SHOOTING_PHOTO_IDLE &&
    photoCnt++ > TAKING_PHOTO_SPENT_TIME_MS_EMU / (1000 / PAYLOAD_CAMERA_EMU_TASK_FREQ)) {
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = PSDK_CAMERA_SHOOTING_PHOTO_IDLE;
    photoCnt = 0;
}

```

存储照片

相机类负载设备在执行完拍照后，使用 PSDK 开发的相机类负载设备将相机拍摄的照片存储在**相机类负载设备**上的内存卡中。

- 存储单拍模式下相机类负载设备拍摄的照片

```

if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_SINGLE) {
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_PHOTO_IDLE;
}

```

- 存储连拍模式下相机类负载设备拍摄的照片

```

else if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_BURST) {
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB * s_cameraBurstCount;
    s_cameraState.isStoring = false;
    s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_PHOTO_IDLE;
}

```

- 存储定时拍照模式下相机类负载设备拍摄的照片

```

else if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
    if (isStartIntervalPhotoAction == true) {
        s_cameraState.isStoring = false;
        s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_PHOTO_IDLE;
        s_cameraSDCardState.remainSpaceInMB =
            s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
    }
}

```

检查存储空间

为确保相机类负载设备中的 SD 卡在相机类负载设备执行拍照动作后，有充足的存储空间存储照片或视频，建议在使用 PSDK 开发的相机类负载设备中添加检查 SD 卡存储空间的功能。

- 检查相机类负载设备执行单拍和连拍后 SD 卡剩余的存储空间。

```

if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
    s_cameraSDCardState.remainSpaceInMB = 0;
    s_cameraSDCardState.isFull = true;
}

```

- 检查相机类负载设备执行定时拍照后 SD 卡剩余的存储空间

```

if (s_cameraShootPhotoMode == ZIYAN_CAMERA_SHOOT_PHOTO_MODE_INTERVAL) {
    if (isStartIntervalPhotoAction == true) {
        s_cameraState.isStoring = false;
        s_cameraState.shootingState = ZIYAN_CAMERA_SHOOTING_PHOTO_IDLE;
        s_cameraSDCardState.remainSpaceInMB =
            s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_PHOTO_SPACE_IN_MB;
    }
}

```

使用遥控器可以控制相机类负载设备执行拍照动作。

在单拍模式下，可执行拍照动作。

在连拍模式下，设置相机类负载设备的连拍张数后，相机类负载设备即可执行连拍动作。

在定时拍照模式下，设置相机类负载设备拍照的间隔时间，相机类负载设备可执行定时拍照动作。

使用录像功能

说明：

- 相机类负载设备在录像的过程中无法拍照和测光；
- 开发者可根据用户的使用需要，设置相机类负载设备录像时如 ISO、曝光以及对焦等参数的默认值；
- 使用相机类负载设备的录像功能前，用户需要在 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 上将相机类负载设备的模式设置为录像模式。

控制相机录像

基于 PSDK 开发的负载设备控制程序调用 StartRecordVideo 和 StopRecordVideo 接口控制相机类负载设备录像，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可控制相机类负载设备录像。

```

static T_ZiyanReturnCode StartRecordVideo(void)
{
    T_ZiyanReturnCode ziyanStat;
    T_ZiyanReturnCode returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    T_ZiyanOsalHandler *osalHandler = ZiyanPlatform_GetOsalHandler();

    ziyanStat = osalHandler->MutexLock(s_commonMutex);
    if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", ziyanStat);
        return ziyanStat;
    }

    if (s_cameraState.isRecording != false) {
        USER_LOG_ERROR("camera is already in recording state");
        returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT_IN_CURRENT_STATE;
        goto out;
    }

    s_cameraState.isRecording = true;
    USER_LOG_INFO("start record video");

out:
    ziyanStat = osalHandler->MutexUnlock(s_commonMutex);
    if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", ziyanStat);
        return ziyanStat;
    }

    return returnCode;
}

static T_ZiyanReturnCode StopRecordVideo(void)
{
    T_ZiyanReturnCode ziyanStat;
    T_ZiyanReturnCode returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
    T_ZiyanOsalHandler *osalHandler = ZiyanPlatform_GetOsalHandler();

    ziyanStat = osalHandler->MutexLock(s_commonMutex);
    if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", ziyanStat);
        return ziyanStat;
    }

    if (s_cameraState.isRecording != true) {
        USER_LOG_ERROR("camera is not in recording state");
        returnCode = ZIYAN_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT_IN_CURRENT_STATE;
        goto out;
    }

    s_cameraState.isRecording = false;
    s_cameraState.currentVideoRecordingTimeInSeconds = 0;
    USER_LOG_INFO("stop record video");

out:
    ziyanStat = osalHandler->MutexUnlock(s_commonMutex);
    if (ziyanStat != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", ziyanStat);
        return ziyanStat;
    }

    return returnCode;
}

```

录像状态更新

使用 PSDK 开发的相机类负载设备控制程序，默认以 10Hz 的频率更新相机的状态。

说明：相机开始录像后，ZIYAN Pilot 及基于 GSDK 开发的移动端 App 会显示当前正在录像的时间，相机停止录像时，该时间将归 0。

```

if (s_cameraState.isRecording) {
    s_cameraState.currentVideoRecordingTimeInSeconds++;
    s_cameraSDCardState.remainSpaceInMB =
        s_cameraSDCardState.remainSpaceInMB - SDCARD_PER_SECONDS_RECORD_SPACE_IN_MB;
    if (s_cameraSDCardState.remainSpaceInMB > SDCARD_TOTAL_SPACE_IN_MB) {
        s_cameraSDCardState.remainSpaceInMB = 0;
        s_cameraSDCardState.isFull = true;
    }
}

static T_ZiyanReturnCode GetSystemState(T_ZiyanCameraSystemState *systemState)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    *systemState = s_cameraState;

    returnCode = osalHandler->MutexUnlock(s_commonMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

在 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 上向负载设备发送录像指令后（也可通过遥控器向负载设备发送录像指令），相机类负载设备根据用户发送的指令控制负载设备录像。

实现对焦功能

请开发者根据选用的**开发平台**以及行业应用实际的使用需求，按照 PSDK 中的结构体 `T_ZiyanCameraFocusHandler` 构造实现相机类负载设备对焦功能的函数，将对焦功能的函数注册到 PSDK 中指定的接口后，用户通过使用 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 能够控制基于 PSDK 开发的相机类负载设备对焦。

```

// 实现设置对焦模式的功能
s_focusHandler.SetFocusMode = SetFocusMode;
s_focusHandler.GetFocusMode = GetFocusMode;
// 实现设置对焦点的功能
s_focusHandler.SetFocusTarget = SetFocusTarget;
s_focusHandler.GetFocusTarget = GetFocusTarget;
// 实现设置对焦助手的功能
s_focusHandler.SetFocusAssistantSettings = SetFocusAssistantSettings;
s_focusHandler.GetFocusAssistantSettings = GetFocusAssistantSettings;
// 实现设置对焦环的功能
s_focusHandler.SetFocusRingValue = SetFocusRingValue;
s_focusHandler.GetFocusRingValue = GetFocusRingValue;
s_focusHandler.GetFocusRingValueUpperBound = GetFocusRingValueUpperBound;

```

使用对焦功能

注册对焦功能

开发者实现相机类负载设备的对焦功能后，需要通过 `ZiyanPayloadCamera_RegFocusHandler` 注册对焦功能，方便用户通过使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 控制相机类负载设备对焦。

```

returnCode = ZiyanPayloadCamera_RegFocusHandler(&s_focusHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register adjustable focal point handler error:0x%08llx", returnCode);
    return returnCode;
}

```

设置对焦模式

基于 PSDK 开发的负载设备控制程序调用 SetFocusMode 和 GetFocusMode 接口能够设置相机类负载设备的对焦模式，用户使用 ZIYAN Pilot 能够切换相机类负载设备的对焦模式。

```
static T_ZiyanReturnCode SetFocusMode(E_ZiyanCameraFocusMode mode)
{
    USER_LOG_INFO("set focus mode:%d", mode);
    s_cameraFocusMode = mode;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetFocusMode(E_ZiyanCameraFocusMode *mode)
{
    *mode = s_cameraFocusMode;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

设置对焦点

基于 PSDK 开发的负载设备控制程序调用 SetFocusTarget 和 GetFocusTarget 接口能够设置相机类负载设备的对焦点，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够设置或获取相机类负载设备的对焦点。

```
static T_ZiyanReturnCode SetFocusTarget(T_ZiyanCameraPointInScreen target)
{
    USER_LOG_INFO("set focus target x:%.2f y:%.2f", target.focusX, target.focusY);
    memcpy(&s_cameraFocusTarget, &target, sizeof(T_ZiyanCameraPointInScreen));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetFocusTarget(T_ZiyanCameraPointInScreen *target)
{
    memcpy(target, &s_cameraFocusTarget, sizeof(T_ZiyanCameraPointInScreen));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

设置对焦环

基于 PSDK 开发的负载设备控制程序调用 SetFocusRingValue 、 GetFocusRingValueUpperBound 和 GetFocusRingValueUpperBound 接口能够设置相机类负载设备对焦环的值，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够设置或获取相机类负载设备对焦环的当前的值和最大值。

```
static T_ZiyanReturnCode SetFocusRingValue(uint32_t value)
{
    USER_LOG_INFO("set focus ring value:%d", value);
    s_cameraFocusRingValue = value;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetFocusRingValue(uint32_t *value)
{
    *value = s_cameraFocusRingValue;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetFocusRingValueUpperBound(uint32_t *value)
{
    *value = FOCUS_MAX_RINGVALUE;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

使用对焦助手

基于 PSDK 开发的负载设备控制程序调用 SetFocusAssistantSettings 和 GetFocusAssistantSettings 接口能够设置相机类负载设备对焦环的值，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够设置或获取相机类负载设备对焦助手的状态。

```
static T_ZiyanReturnCode SetFocusAssistantSettings(T_ZiyanCameraFocusAssistantSettings settings)
{
    USER_LOG_INFO("set focus assistant setting MF:%d AF:%d", settings.isEnabledMF, settings.isEnabledAF);
    memcpy(&s_cameraFocusAssistantSettings, &settings, sizeof(T_ZiyanCameraFocusAssistantSettings));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetFocusAssistantSettings(T_ZiyanCameraFocusAssistantSettings *settings)
{
    memcpy(settings, &s_cameraFocusAssistantSettings, sizeof(T_ZiyanCameraFocusAssistantSettings));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

实现相机类负载设备的对焦功能后，在自动对焦模式下，相机类负载设备需要设置对焦点；在手动对焦模式下，用户可根据实际需要调整对焦点。

说明：若按钮为黄色，表示当前为测光功能，点击后，即可切换为对焦模式。

实现测光功能

请开发者根据选用的开发平台以及行业应用实际的使用需求，按照 PSDK 中的结构体 T_ZiyanCameraExposureMeteringHandler 构造实现相机类负载设备测光功能的函数，将测光功能的函数注册到 PSDK 中指定的接口后，用户通过使用 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 能够控制相机类负载设备测光。

```
// 实现设置测光模式的功能
s_exposureMeteringHandler.SetMeteringMode = SetMeteringMode;
s_exposureMeteringHandler.GetMeteringMode = GetMeteringMode;
// 实现控制负载设备测光的功能
s_exposureMeteringHandler.SetSpotMeteringTarget = SetSpotMeteringTarget;
s_exposureMeteringHandler.GetSpotMeteringTarget = GetSpotMeteringTarget;
```

使用测光功能

注册测光功能

开发者实现相机类负载设备的测光功能后，需要通过 PsdkPayloadCamera_RegExposureMeteringHandler 注册测光功能；调用指定的接口后，用户通过使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 即可控制相机类负载设备测光，如图 2. 测光功能

```
returnCode = ZiyanPayloadCamera_RegExposureMeteringHandler(&s_exposureMeteringHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register exposure metering handler error:0x%08llx", returnCode);
    return returnCode;
}
```

设置测光模式

基于 PSDK 开发的负载设备控制程序调用 SetMeteringMode 和 GetMeteringMode 接口能够设置或获取相机类负载设备的测光模式，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的 APP 能够查看负载设备的测光模式，如图 3. 指点测光和图 4. 中央重点测光所示。

```
static T_ZiyanReturnCode SetMeteringMode(E_ZiyanCameraMeteringMode mode)
{
    USER_LOG_INFO("set metering mode:%d", mode);
    s_cameraMeteringMode = mode;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetMeteringMode(E_ZiyanCameraMeteringMode *mode)
{
    *mode = s_cameraMeteringMode;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

设置测光对象

相机类负载设备在调用 SetSpotMeteringTarget 和 GetSpotMeteringTarget 接口后，使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 可设置或获取相机类负载设备的测光对象。

```
static T_ZiyanReturnCode SetSpotMeteringTarget(T_ZiyanCameraSpotMeteringTarget target)
{
    USER_LOG_INFO("set spot metering area col:%d row:%d", target.col, target.row);
    memcpy(&s_cameraSpotMeteringTarget, &target, sizeof(T_ZiyanCameraSpotMeteringTarget));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetSpotMeteringTarget(T_ZiyanCameraSpotMeteringTarget *target)
{
    memcpy(target, &s_cameraSpotMeteringTarget, sizeof(T_ZiyanCameraSpotMeteringTarget));

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

说明：在 ZIYAN Pilot 上使用相机类负载设备的测光功能时，若按钮为绿色，表示当前为对焦功能，点击后，即可切换为测光模式。

实现变焦功能

请开发者根据选用的开发平台以及行业应用实际的使用需求，按照 PSDK 中的结构体 T_ZiyanCameraTapZoomHandler 构造实现相机类负载设备变焦功能的函数，将变焦功能的函数注册到 PSDK 中指定的接口后，用户通过使用 ZIYAN Pilot 或基于 GSDK 开发的移动端 App 能够控制相机类负载设备变焦。

```
// 实现控制负载设备执行数码变焦的功能
s_digitalZoomHandler.SetDigitalZoomFactor = SetDigitalZoomFactor;
s_digitalZoomHandler.GetDigitalZoomFactor = GetDigitalZoomFactor;
// 实现控制负载设备执行光学变焦的功能
s_opticalZoomHandler.SetOpticalZoomFocalLength = SetOpticalZoomFocalLength;
s_opticalZoomHandler.GetOpticalZoomFocalLength = GetOpticalZoomFocalLength;
s_opticalZoomHandler.GetOpticalZoomFactor = GetOpticalZoomFactor;
s_opticalZoomHandler.GetOpticalZoomSpec = GetOpticalZoomSpec;
s_opticalZoomHandler.StartContinuousOpticalZoom = StartContinuousOpticalZoom;
s_opticalZoomHandler.StopContinuousOpticalZoom = StopContinuousOpticalZoom;
// 实现控制负载设备执行指点变焦的功能
s_tapZoomHandler.GetTapZoomState = GetTapZoomState;
s_tapZoomHandler.SetTapZoomEnabled = SetTapZoomEnabled;
s_tapZoomHandler.GetTapZoomEnabled = GetTapZoomEnabled;
s_tapZoomHandler.SetTapZoomMultiplier = SetTapZoomMultiplier;
s_tapZoomHandler.GetTapZoomMultiplier = GetTapZoomMultiplier;
s_tapZoomHandler.TapZoomAtTarget = TapZoomAtTarget;
```

使用变焦功能

注册变焦功能

开发者实现相机类负载设备的变焦功能后，需要通过注册接口注册各个变焦功能的函数；基于 PSDK 开发的负载设备通过调用指定的接口，即可控制相机类负载设备执行变焦，方便用户通过使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 控制相机类负载设备变焦。

- 注册数码变焦功能

```
returnCode = ZiyanPayloadCamera_RegDigitalZoomHandler(&s_digitalZoomHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register digital zoom handler error:0x%08llx", returnCode);
    return returnCode;
}
```

- 注册光学变焦功能

```
returnCode = ZiyanPayloadCamera_RegOpticalZoomHandler(&s_opticalZoomHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register optical zoom handler error:0x%08llx", returnCode);
    return returnCode;
}
```

- 注册指点变焦功能

```

returnCode = ZiyenPayloadCamera_RegTapZoomHandler(&s_tapZoomHandler);
if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
    USER_LOG_ERROR("camera register tap zoom handler error:0x%08lX", returnCode);
    return returnCode;
}

```

使用数码变焦功能

基于 PSDK 开发的负载设备控制程序调用 SetDigitalZoomFactor 和 GetDigitalZoomFactor 接口能够控制负载设备执行数码变焦，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够控制相机类负载设备的执行数码变焦，同时获取负载设备数码变焦的系数。

```

static T_ZiyenReturnCode SetDigitalZoomFactor(ziyen_f32_t factor)
{
    T_ZiyenReturnCode returnCode;
    T_ZiyenOsHandler *osalHandler = ZiyenPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("set digital zoom factor:%.2f", factor);
    s_cameraDigitalZoomFactor = factor;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

T_ZiyenReturnCode ZiyenTest_CameraGetDigitalZoomFactor(ziyen_f32_t *factor)
{
    T_ZiyenReturnCode returnCode;
    T_ZiyenOsHandler *osalHandler = ZiyenPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    *factor = s_cameraDigitalZoomFactor;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

使用光学变焦功能

基于 PSDK 开发的负载设备控制程序调用 SetOpticalZoomFocalLength 和 GetOpticalZoomFocalLength 接口能够控制负载设备执行光学变焦，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够控制相机类负载设备执行光学变焦，同时获取负载设备光学变焦的系数。

- 设置光学变焦相机的焦距


```

static T_ZiyanReturnCode SetOpticalZoomFocalLength(uint32_t focalLength)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("set optical zoom focal length:%d", focalLength);
    s_isOpticalZoomReachLimit = false;
    s_cameraDigitalZoomFactor = ZOOM_DIGITAL_BASE_FACTOR;
    s_cameraOpticalZoomFocalLength = ZOOM_OPTICAL_FOCAL_MIN_LENGTH;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

T_ZiyanReturnCode ZiyanTest_CameraGetOpticalZoomFactor(ziyan_f32_t *factor)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    //Formula:factor = currentFocalLength / minFocalLength
    *factor = (ziyan_f32_t) s_cameraOpticalZoomFocalLength / ZOOM_OPTICAL_FOCAL_MIN_LENGTH;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

- 获取相机类负载设备的变焦系数

获取相机类负载设备当前的光学焦距后，根据变焦系数的计算公式，能够计算相机类负载设备当前的变焦系数（变焦系数 = 当前焦距 ÷ 最短焦距）。

```

T_ZiyanReturnCode ZiyanTest_CameraGetOpticalZoomFactor(ziyan_f32_t *factor)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    //Formula:factor = currentFocalLength / minFocalLength
    *factor = (ziyan_f32_t) s_cameraOpticalZoomFocalLength / ZOOM_OPTICAL_FOCAL_MIN_LENGTH;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

- 获取光学变焦的范围

```
static T_ZiyanReturnCode GetOpticalZoomSpec(T_ZiyanCameraOpticalZoomSpec *spec)
{
    spec->maxFocalLength = ZOOM_OPTICAL_FOCAL_MAX_LENGTH;
    spec->minFocalLength = ZOOM_OPTICAL_FOCAL_MIN_LENGTH;
    spec->focalLengthStep = ZOOM_OPTICAL_FOCAL_LENGTH_STEP;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

使用连续变焦功能

基于 PSDK 开发的负载设备控制程序调用 StartContinuousOpticalZoom 和 StopContinuousOpticalZoom 接口能够控制负载设备开始或停止执行连续变焦，用户使用 ZIYAN Pilot 以及基于 GSDK 开发的移动端 App 能够控制相机类负载设备执行连续变焦。

- 控制相机类负载设备开始变焦

```
static T_ZiyanReturnCode StartContinuousOpticalZoom(E_ZiyanCameraZoomDirection direction, E_ZiyanCameraZoomSpeed speed)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsalHandler *osalHandler = ZiyanPlatform_GetOsalHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("start continuous optical zoom direction:%d speed:%d", direction, speed);
    s_isStartContinuousOpticalZoom = true;
    s_cameraZoomDirection = direction;
    s_cameraZoomSpeed = speed;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

- 控制相机类负载设备停止变焦

```
static T_ZiyanReturnCode StopContinuousOpticalZoom(void)
{
    T_ZiyanReturnCode returnCode;
    T_ZiyanOsalHandler *osalHandler = ZiyanPlatform_GetOsalHandler();

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    USER_LOG_INFO("stop continuous optical zoom");
    s_isStartContinuousOpticalZoom = false;
    s_cameraZoomDirection = ZIYAN_CAMERA_ZOOM_DIRECTION_OUT;
    s_cameraZoomSpeed = ZIYAN_CAMERA_ZOOM_SPEED_NORMAL;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08llx.", returnCode);
        return returnCode;
    }

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

- 控制相机持续变焦

```

if (s_isStartContinuousOpticalZoom == true) {
    tempDigitalFactor = s_cameraDigitalZoomFactor;
    tempFocalLength = (int32_t) s_cameraOpticalZoomFocalLength;
    if (s_isOpticalZoomReachLimit == false) {
        if (s_cameraZoomDirection == ZIYAN_CAMERA_ZOOM_DIRECTION_IN) {
            tempFocalLength += ((int) s_cameraZoomSpeed - ZIYAN_CAMERA_ZOOM_SPEED_SLOWEST + 1) * ZOOM_OPTICAL_FOCAL_LENGTH_1;
        } else if (s_cameraZoomDirection == ZIYAN_CAMERA_ZOOM_DIRECTION_OUT) {
            tempFocalLength -= ((int) s_cameraZoomSpeed - ZIYAN_CAMERA_ZOOM_SPEED_SLOWEST + 1) * ZOOM_OPTICAL_FOCAL_LENGTH_1;
        }

        if (tempFocalLength > ZOOM_OPTICAL_FOCAL_MAX_LENGTH) {
            s_isOpticalZoomReachLimit = true;
            tempFocalLength = ZOOM_OPTICAL_FOCAL_MAX_LENGTH;
        }

        if (tempFocalLength < ZOOM_OPTICAL_FOCAL_MIN_LENGTH) {
            tempFocalLength = ZOOM_OPTICAL_FOCAL_MIN_LENGTH;
        }
    } else {
        if (s_cameraZoomDirection == ZIYAN_CAMERA_ZOOM_DIRECTION_IN) {
            tempDigitalFactor += (ziyan_f32_t) ZOOM_DIGITAL_STEP_FACTOR;
        } else if (s_cameraZoomDirection == ZIYAN_CAMERA_ZOOM_DIRECTION_OUT) {
            tempDigitalFactor -= (ziyan_f32_t) ZOOM_DIGITAL_STEP_FACTOR;
        }

        if (tempDigitalFactor > (ziyan_f32_t) ZOOM_DIGITAL_MAX_FACTOR) {
            tempDigitalFactor = (ziyan_f32_t) ZOOM_DIGITAL_MAX_FACTOR;
        }

        if (tempDigitalFactor < (ziyan_f32_t) ZOOM_DIGITAL_BASE_FACTOR) {
            s_isOpticalZoomReachLimit = false;
            tempDigitalFactor = ZOOM_DIGITAL_BASE_FACTOR;
        }
    }
    s_cameraOpticalZoomFocalLength = (uint16_t) tempFocalLength;
    s_cameraDigitalZoomFactor = tempDigitalFactor;
}

```

持续按住变焦按钮可改变变焦倍数，如图 1. 连续变焦 所示。

- T: 放大焦距（放大变焦倍数）
- W: 缩小焦距（缩小变焦倍数）
- R: 还原相机的焦距

说明： 根据实际的使用需要，可设置相机类负载设备默认的变焦倍数，当前为 1.0。

图 15. 连续变焦



实现指点变焦功能

当用户开始使用“指点变焦”功能后，使用 PSDK 开发的相机类负载设备将根据用户指定的目标点位置以及当前的焦距，先控制云台旋转，将目标对象置于画面中心，再控制负载设备变焦。

通过注册回调函数的方式实现指点变焦功能

- 设置指点变焦的变焦系数

```
static T_ZiyanReturnCode SetTapZoomMultiplier(uint8_t multiplier)
{
    USER_LOG_INFO("set tap zoom multiplier: %d.", multiplier);
    s_tapZoomMultiplier = multiplier;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

static T_ZiyanReturnCode GetTapZoomMultiplier(uint8_t *multiplier)
{
    *multiplier = s_tapZoomMultiplier;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}
```

- 获取指点变焦的对象

使用 PSDK 开发的相机类负载设备通过 TapZoomAtTarget 接口获取用户在移动端 App 中指定的变焦对象，在确认指点变焦功能开启后，根据目标点位置和混合焦距，计算云台转动角度，控制相机转动。

```

static T_ZiyanReturnCode TapZoomAtTarget(T_ZiyanCameraPointInScreen target)
{
    T_ZiyanReturnCode returnCode;
    E_ZiyanGimbalRotationMode rotationMode;
    T_ZiyanGimbalRotationProperty rotationProperty = {0};
    T_ZiyanAttitude3d rotationValue = {0};
    float hybridFocalLength = 0; // unit: 0.1mm
    T_ZiyanOsHandler *osalHandler = ZiyanPlatform_GetOsHandler();

    USER_LOG_INFO("tap zoom at target: x %f, y %f.", target.focusX, target.focusY);

    if (s_isTapZoomEnabled != true) {
        USER_LOG_WARN("tap zoom is not enabled.");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SYSTEM_ERROR;
    }

    if (s_isTapZooming || s_isStartTapZoom) {
        USER_LOG_WARN("The last tap zoom process is not over.");
        return ZIYAN_ERROR_SYSTEM_MODULE_CODE_NONSUPPORT_IN_CURRENT_STATE;
    }

    rotationMode = ZIYAN_GIMBAL_ROTATION_MODE_RELATIVE_ANGLE;
    rotationProperty.relativeAngleRotation.actionTime = TAP_ZOOM_DURATION / 10;

    returnCode = osalHandler->MutexLock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    /* Calculation formula: rotation angle = arctan((coordinate of target in sensor - coordinate of center point in
    * sensor) / hybrid focal length). Here, suppose that images of all pixels of sensor are displayed in screen,
    * and that center of the image sensor coincides with center of rotation of the gimbal, and that optical axis of
    * camera coincides with x-axis of gimbal. */
    hybridFocalLength = (ziyan_f32_t) s_cameraOpticalZoomFocalLength * s_cameraDigitalZoomFactor;

    returnCode = osalHandler->MutexUnlock(s_zoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    rotationValue.pitch = (int32_t) (
        atan2f((target.focusY - CENTER_POINT_IN_SCREEN_Y_VALUE) * IMAGE_SENSOR_Y_SIZE, hybridFocalLength) * 1800 /
        ZIYAN_PI);
    rotationValue.yaw = (int32_t) (
        atan2f((target.focusX - CENTER_POINT_IN_SCREEN_X_VALUE) * IMAGE_SENSOR_X_SIZE, hybridFocalLength) * 1800 /
        ZIYAN_PI);

    returnCode = osalHandler->MutexLock(s_tapZoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("lock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_tapZoomNewestGimbalRotationArgument.rotationMode = rotationMode;
    s_tapZoomNewestGimbalRotationArgument.rotationProperty = rotationProperty;
    s_tapZoomNewestGimbalRotationArgument.rotationValue = rotationValue;
    s_tapZoomNewestTargetHybridFocalLength = (uint32_t) (hybridFocalLength * (float) s_tapZoomMultiplier);

    returnCode = osalHandler->MutexUnlock(s_tapZoomMutex);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("unlock mutex error: 0x%08lX.", returnCode);
        return returnCode;
    }

    s_isStartTapZoom = true;

    return ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS;
}

```

在线程中实现指点变焦功能

为避免负载设备在旋转云台和控制变焦时，阻塞负载设备控制程序的主线程，请在线程中实现指点变焦功能。

```

if (s_isStartTapZoom) {
    s_isStartTapZoom = false;
    s_isTapZooming = true;

    returnCode =osalHandler->GetTimeMs(&s_tapZoomStartTime);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("get start time error: 0x%08lX.", returnCode);
    }

    returnCode = Ziyantest_CameraRotationGimbal(s_tapZoomNewestGimbalRotationArgument);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS)
        USER_LOG_ERROR("rotate gimbal error: 0x%08lX.", returnCode);
    else
        s_cameraTapZoomState.isGimbalMoving = true;

    returnCode = Ziyantest_CameraHybridZoom(s_tapZoomNewestTargetHybridFocalLength);
    if (returnCode == ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        s_cameraTapZoomState.zoomState = (ziyan_f32_t) s_tapZoomNewestTargetHybridFocalLength >
            ((ziyan_f32_t) s_cameraOpticalZoomFocalLength *
            s_cameraDigitalZoomFactor)
            ? ZIYAN_CAMERA_TAP_ZOOM_STATE_ZOOM_IN
            : ZIYAN_CAMERA_TAP_ZOOM_STATE_ZOOM_OUT;
    } else if (returnCode == ZIYAN_ERROR_SYSTEM_MODULE_CODE_OUT_OF_RANGE) {
        USER_LOG_ERROR("hybrid zoom focal length beyond limit.");
        s_cameraTapZoomState.zoomState = ZIYAN_CAMERA_TAP_ZOOM_STATE_ZOOM_LIMITED;
    } else {
        USER_LOG_ERROR("hybrid zoom error: 0x%08lX.", returnCode);
    }
} else if (s_isTapZooming) {
    returnCode =osalHandler->GetTimeMs(&currentTime);
    if (returnCode != ZIYAN_ERROR_SYSTEM_MODULE_CODE_SUCCESS) {
        USER_LOG_ERROR("get start time error: 0x%08lX.", returnCode);
    }

    if ((currentTime - s_tapZoomStartTime) >= TAP_ZOOM_DURATION) {
        s_cameraTapZoomState.zoomState = ZIYAN_CAMERA_TAP_ZOOM_STATE_IDLE;
        s_cameraTapZoomState.isGimbalMoving = false;
        s_isTapZooming = false;
    }
}
}

```


PSDK获取版本信息相关功能的头文件为 `ziyan_version.h`，本文档描述了 `ziyan_version.h` 文件中的宏定义。

宏定义

- ZIYAN SDK 主版本号

ZIYAN SDK 主版本号，当有不兼容的 API 更改时。范围从 0 到 99。

```
#define ZIYAN_VERSION_MAJOR    1
```

- ZIYAN SDK 次版本号

ZIYAN SDK 次版本号，当以向后兼容的方式添加功能时会发生变化。范围从 0 到 99。

```
#define ZIYAN_VERSION_MINOR    0
```

- ZIYAN SDK 修改版本号

ZIYAN SDK 修改版本号，当有向后兼容的错误修复更改时。范围从 0 到 99。

```
#define ZIYAN_VERSION_MODIFY    0
```

- ZIYAN SDK 调试版本信息

ZIYAN SDK 调试版本信息，发布版本为0，当调试版本发布变化时。范围从 0 到 255。

```
#define ZIYAN_VERSION_BETA      0
```

- ZIYAN SDK 构建版本号，范围为0~65535

ZIYAN SDK 版本构建信息，当 jenkins 触发构建更改时。范围从 0 到 65535。

```
#define ZIYAN_VERSION_BUILD     1929
```