ESP-Drone

2023 年 11 月 24 日

Contents

[English]

ESP-Drone 是基于乐鑫 ESP32/ESP32-S2/ESP32-S3 开发的小型无人机解决方案，可使用手机 APP 或游戏手柄通过 Wi-Fi 网络进行连接和控制。目前已支持自稳定飞行、定高飞行、定点飞行等多种模式。该方案硬件结构简单，代码架构清晰完善，方便功能扩展，可用于 STEAM 教育等领域。控制系统代码来自 Crazyflie 开源工程，使用 GPL3.0 开源协议。

本文档包含 ESP-Drone 基本的开发和使用说明。

CHAPTER 1

ESP-Drone 支持政策

自 2022 年 12 月起，我们仅提供有限的支持，但是 Pull Request 仍然是被欢迎的！

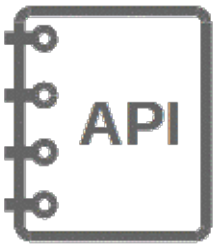
CHAPTER 2

支持的版本和芯片

ESP-Drone	Dependent ESP-IDF	Target	Support State
master	release/v4.4 , release/v5.0	ESP32-S2 / ESP32-S3	Limiter Support

CHAPTER 3

快速入门

		
快速入门	开发指引	硬件参考

3.1 快速入门

[English]

3.1.1 项目简介

ESP-Drone 是基于乐鑫 ESP32/ESP32-S2/ESP32-S3 开发的小型无人机解决方案，可使用手机 APP 或游戏手柄通过 Wi-Fi 网络进行连接和控制。该方案硬件结构简单，代码架构清晰，支持功能扩展，可用于 STEAM 教育等领域。项目部分代码来自 Crazyflie 开源工程，继承 GPL3.0 开源协议。

图 1: ESP-Drone 无人机

主要特性

ESP-Drone 具备以下特性：

- 支持自稳定模式 (Stabilize mode)：自动控制机身水平，保持平稳飞行。
- 支持定高模式 (Height-hold mode)：自动控制油门输出，保持固定高度。
- 支持定点模式 (Position-hold mode)：自动控制机身角度，保持固定空间位置。
- 支持 PC 上位机调试：使用 cfclient 上位机进行静态/动态调试。
- 支持 APP 控制：使用手机 APP 通过 Wi-Fi 轻松控制。
- 支持游戏手柄 (gamepad) 控制：通过 cfclient 使用游戏手柄轻松控制。

主要组件

ESP-Drone 2.0 使用模块化的设计思路，由主控板和扩展板组成。

- **主控制板**：搭载 ESP32-S2 模组和支持基础飞行的必要传感器，并提供硬件扩展接口。
- **扩展板**：搭载扩展传感器，可对接主控制板的硬件扩展接口，支持高级飞行功能。

序号	模块名	主要元器件	功能	接口	安装位置
1	主控制板 - **ESP32-S2**	ESP32-S2-WROVER + MPU6050	基础飞行	提供 I2C SPI GPIO 扩展接口	
2	扩展板 - **定点模块**	PMW3901 + VL53L1X	室内定点飞行	SPI + I2C	底部，面向地面
3	扩展板 - **气压定高模块**	MS5611 气压	气压定高	I2C 或 MPU6050 从机	顶部或底部
4	扩展板 - **指南针模块**	HMC5883 罗盘	无头模式等高级模式	I2C 或 MPU6050 从机	顶部或底部

详情可查阅：硬件参考。

3.1.2 ESP-IDF 简介

ESP-IDF 是乐鑫为 ESP32/ESP32-S2/ESP32-S3 提供的物联网开发框架。

- ESP-IDF 包含一系列库及头文件，提供了基于 ESP32/ESP32-S2/ESP32-S3 构建软件项目所需的核心组件。
- ESP-IDF 还提供了开发和量产过程中最常用的工具及功能，例如：构建、烧录、调试和测量等。

详情可查阅：ESP-IDF 编程指南。

3.1.3 Crazyflie 简介

Crazyflie 是来自 Bitcraze 开源工程的四旋翼飞行器，具备以下特性：

- 支持多种传感器组合，可以轻松实现定高模式、定点模式等高级飞行模式。
- 基于 FreeRTOS 编写，将复杂的无人机系统，分解成多个具有不同优先级的软件任务。
- 设计了功能完备的 cfclient 上位机和 CRTP 通信协议，便于实现调试、测量和控制。

详情可查阅 Crazyflie 官网。

图 2: 多架无人机同时探索周围环境，灵活避开障碍物，同时避开其它无人机。A swarm of drones exploring the environment, avoiding obstacles and each other. (Guus Schoonewille, TU Delft)

3.1.4 准备工作

硬件组装

请按照下述步骤组装 ESP32-S2-Drone V1.2：

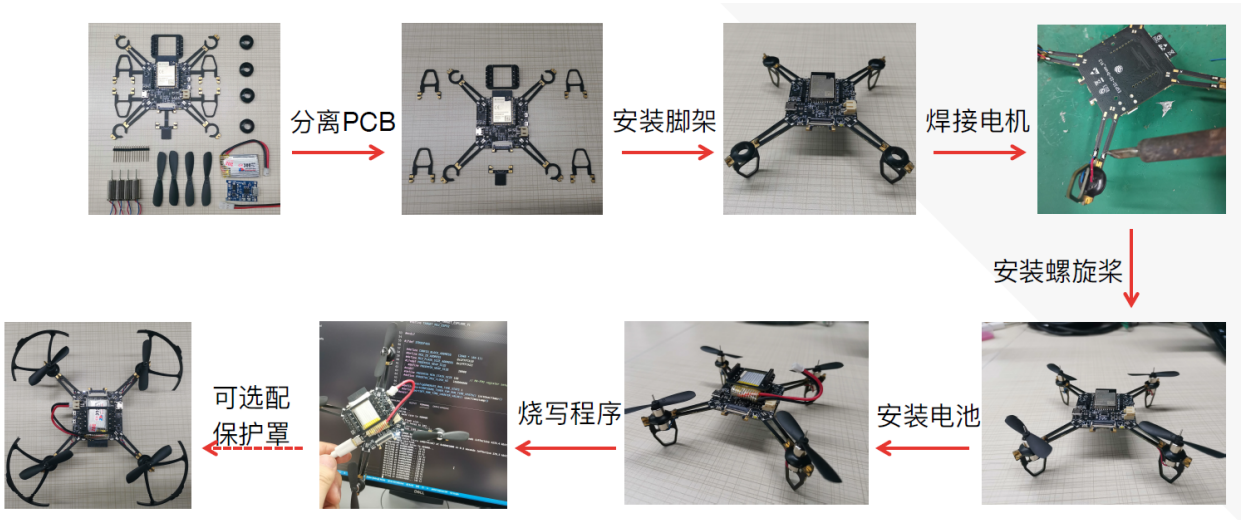


图 3: ESP32-S2-Drone V1.2 组装流程

硬件介绍和管脚资源分配可查阅：硬件参考。

安装 ESP-Drone APP

ESP-Drone APP 同时支持 Android 系统和 iOS 系统。

扫描下方二维码，下载 Android APP：

下载 iOS APP：

在 App Store 中搜索 ESP-Drone，点击下载并安装。

iOS APP 源代码：ESP-Drone-iOS

Android APP 源代码：ESP-Drone-Android

安装 cfclient

安装 cfclient 为可选步骤，用于实现高级调试，非必须使用。

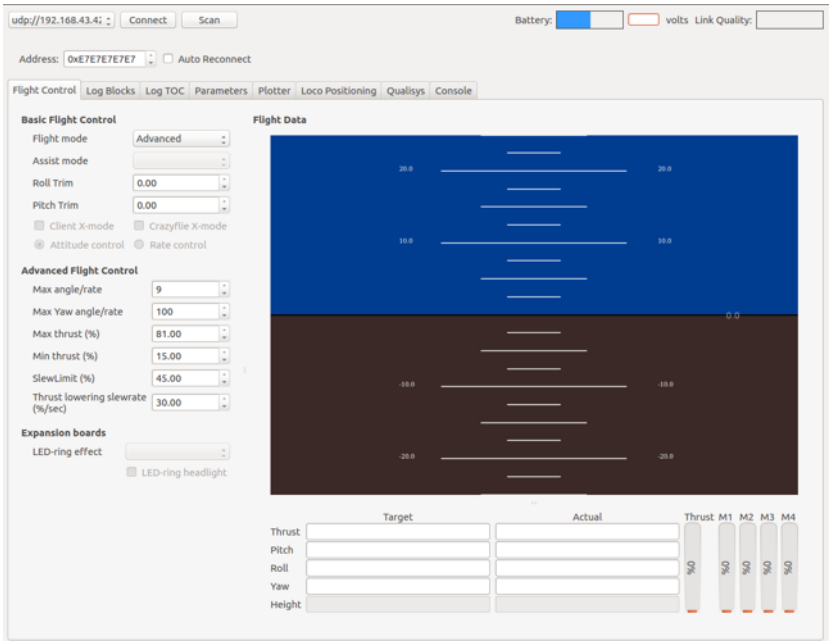


图 4: cfclient 上位机界面

1. 安装 CRTP 协议支持包

1.1 下载源代码

```
git clone -b esp-drone  https://github.com/qljz1993/crazyflie-lib-python.git
```

1.2 进入源码目录，安装依赖

```
pip3 install -r requirements.txt
```

1.3 安装 CRTP 包

```
pip3 install -e .
```

2. 安装 cfclient

2.1 下载源代码

```
git clone -b esp-drone https://github.com/qljz1993/crazyflie-clients-python.git
```

2.2 进入源码目录，安装依赖

```
sudo apt-get install python3 python3-pip python3-pyqt5 python3-pyqt5.qtsvg
```

2.3 安装 cfclient 客户端

```
pip3 install -e .
```

2.4 启动客户端

```
python3 ./bin/cfclient
```

3. 配置遥控器

3.1 配置 4 个控制轴：`Roll` 、`Pitch`、`Yaw`、`Thrust`。

3.2 配置一个按键为 `Assisted control`，用于飞行模式切换。

3.1.5 手机 APP 使用指南

Wi-Fi 连接

- 手机扫描 Wi-Fi AP。ESP-Drone 设备用作 AP，其 SSID 及密码如下：

```
SSID: ESP-DRONE_XXXX (XXXX 根据 MAC 设置) PASSWORD: 12345678
```

- 点击该 AP，手机与 ESP-Drone 设备建立 Wi-Fi 连接。

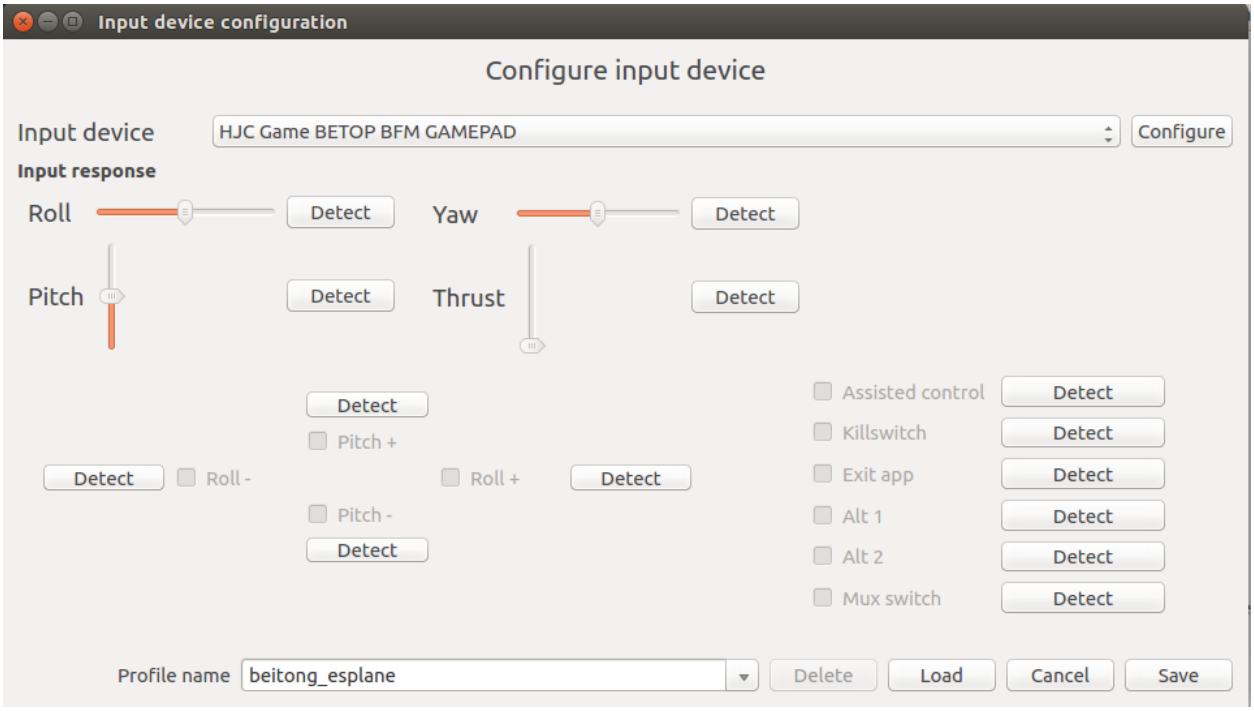


图 5: 遥控器配置

个性化设置

在该步骤中，您可以按照具体的应用场景进行个性化配置，或使用默认配置。

```
```
默认配置:

Flight control settings
    1. Mode: Mode2
    2. Deadzone: 0.2
    3. Roll trim: 0.0
    4. Pitch trim: 0.0
    5. Advanced flight control : true
    6. Advanced flight control preferences
        1. max roll/pitch angle: 15
        2. max yaw angle: 90
        3. max thrust: 90
        4. min thrust: 25
        5. X-Mode: true
Controller settings
    1. use full travel for thrust: false
    2. virtual joystick size: 100
App settings
```

(下页继续)
```

(续上页)

```
    1. Screen rotation lock: true
    2. full screen mode:true
    3. show console: true
```
```

#### 控制飞行

- 打开 APP，点击 *Connect* 按钮，连接小飞机。连接成功，小飞机绿灯闪烁。
- 轻推油门，小飞机起飞。
- 在 APP 上滑动，控制小飞机方向。

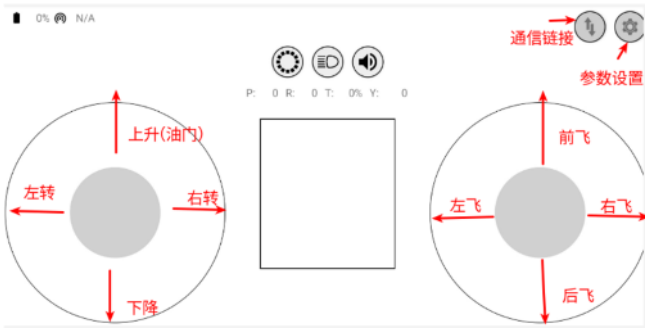



图 6: Android APP 用户界面

### 3.1.6 PC cfclient 使用指南

cfclient 是 `Crazeflie` 源工程的上位机，完全实现了 `CRTP` 协议中定义的功能，可以加快飞机的调试过程。ESP-Drone 项目对该上位机进行裁剪和调整，满足功能设计需求。

项目中有很多相关的文件，例如配置文件和缓存文件，其中 JSON 文件用来存储配置信息。关于配置信息中内容的解读，可参考：User Configuration File。

#### 飞行设置

#### 基本飞行设置 (Basic Flight Control)

1. 飞行模式 (Flight mode)：基本模式和高级模式
   - 基本模式 (Normal mode)：初学者使用。
   - 高级模式 (Advanced mode)：设置解锁最大角度，设置最大油门。
2. 自动模式 (Assisted mode)
```

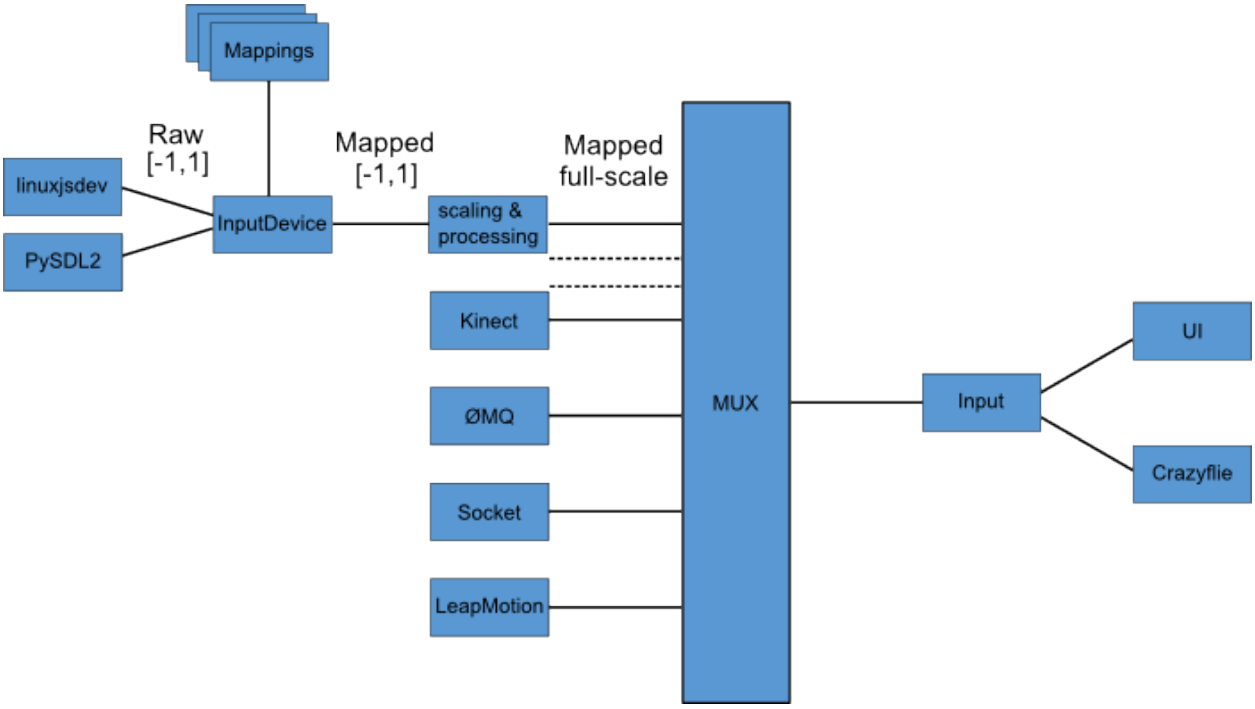


图 7: cfclient 架构

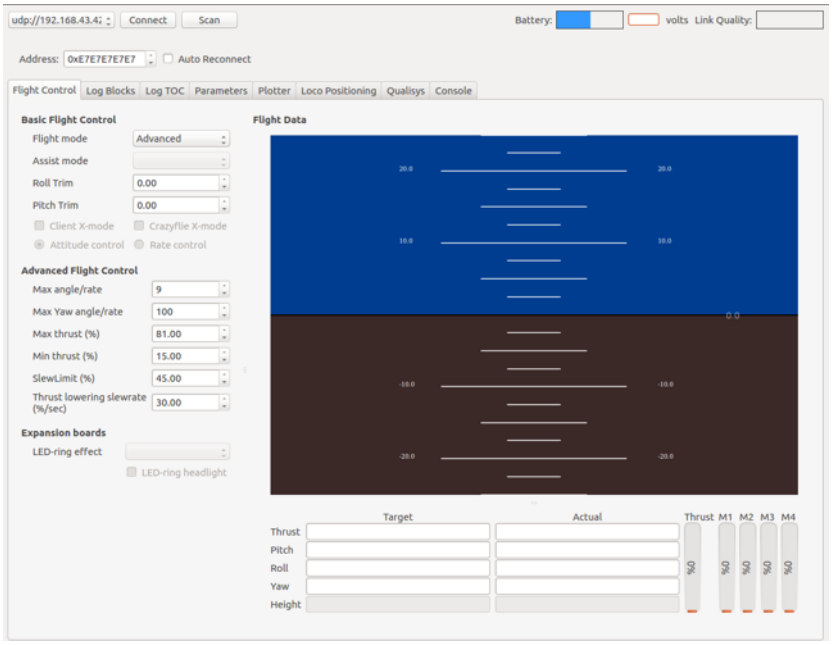


图 8: cfclient 控制台界面

- 定海拔模式 (Altitude-hold mode)：保持飞行海拔，需要气压计支持。
- 定点模式 (Position-hold mode)：保持当前位置，需要光流和 TOF 支持。
- 定高模式 (Height-hold mode)：保持相对高度，触发时保持高于地面 40 cm，需要 TOF 支持。
- 悬停模式 (Hover mode)：触发时保持高于地面 40 cm，并悬停在起飞点，需要光流和 TOF 支持。

3. 角度修正 (Trim)
   - 翻滚角修正 (Roll Trim)：用于弥补传感器水平安装误差。
   - 俯仰角修正 (Pitch Trim)：用于弥补传感器水平安装误差。

注意，在自动模式下，油门摇杆变为高度控制摇杆。

高级飞行设置 (Advanced Flight Control)

1. 最大倾角 (Max angle)：设置最大允许的俯仰和翻滚角度：roll/pitch。
2. 最大自旋速度 (Max yaw rate)：设置允许的偏航速度：yaw。
3. 最大油门 (Max thrust)：设置最大油门。
4. 最小油门 (Min thrust)：设置最小油门。
5. 压摆极限 (Slew limit)：防止油门骤降，油门低于该值时，下降速度将被限定。
6. 压摆率 (Slew rate)：油门到压摆极限之后的最大下降率。

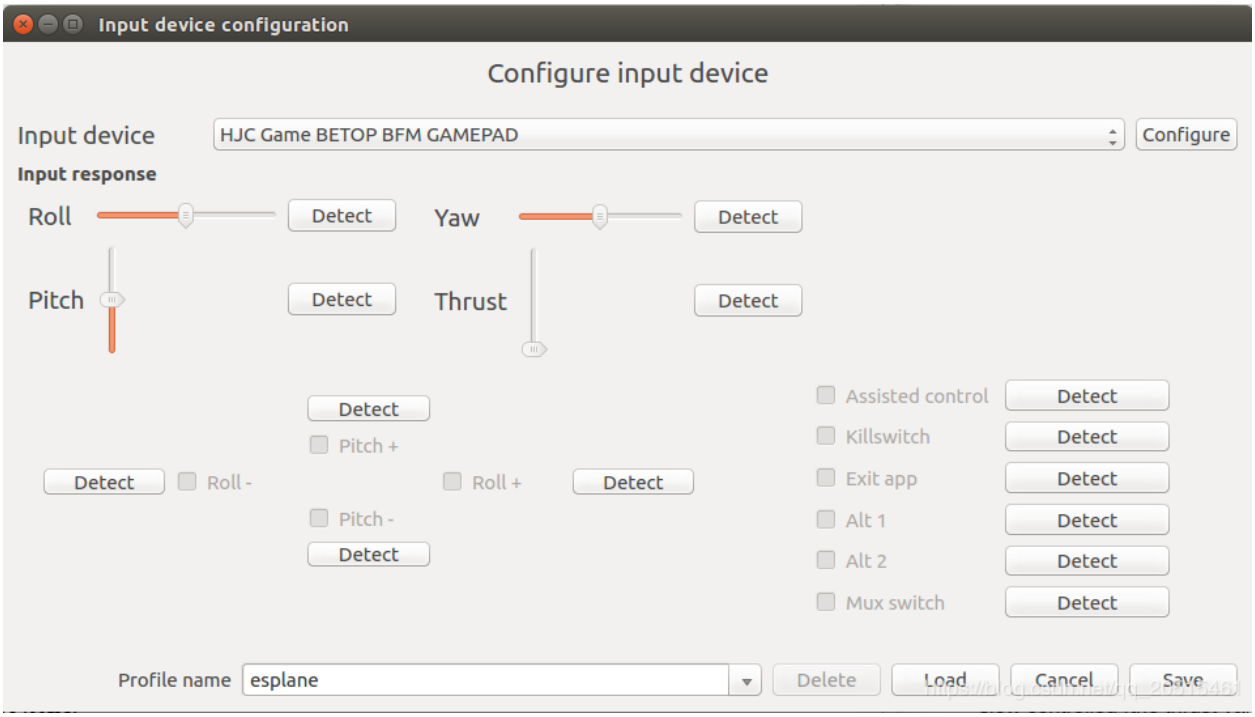
遥控器设置 (Configure Input Device)

按照提示绑定遥控器摇杆与各个控制通道：

飞行数据 (Flight Data)

驾驶仪可以看到当前飞机姿态，右下方显示对应的详细数据。

1. 目标角度 (Target)
2. 测量角度 (Actual)
3. 当前油门值 (Thrust)
4. 电机实际输出 (M1/M2/M3/M4)



在线参数修改

在线调整 PID 参数

注意事项

1. 修改的参数实时生效，避免了频繁烧录固件。
2. 可在代码中通过宏定义，配置哪些参数可被上位机实时修改。
3. 注意，参数在线修改仅用于调试，掉电不保存。

飞行数据监控

配置要监控的参数

实时波形绘制

陀螺仪加速度计实时数据监测：

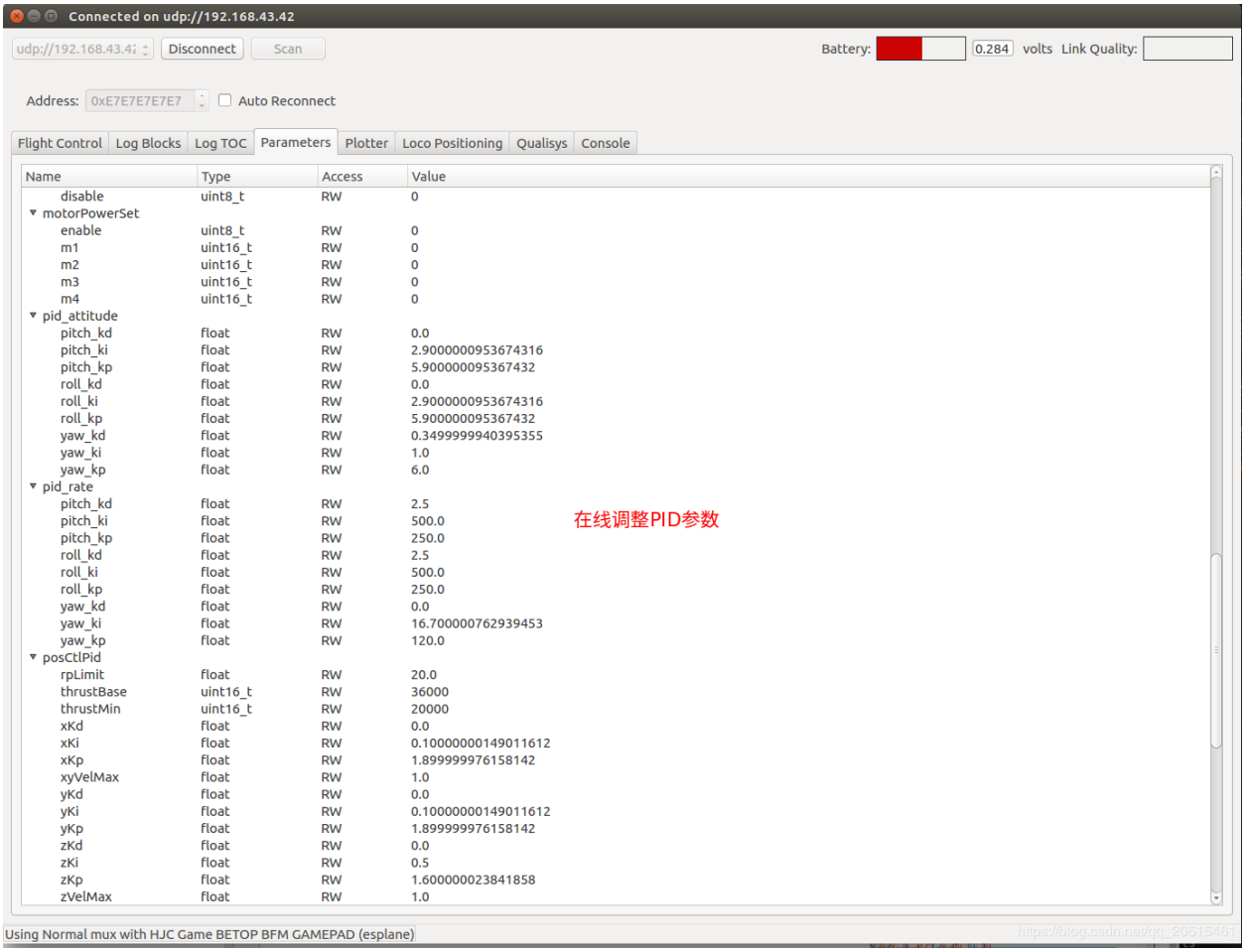


图 9: cfclient PID 参数调整

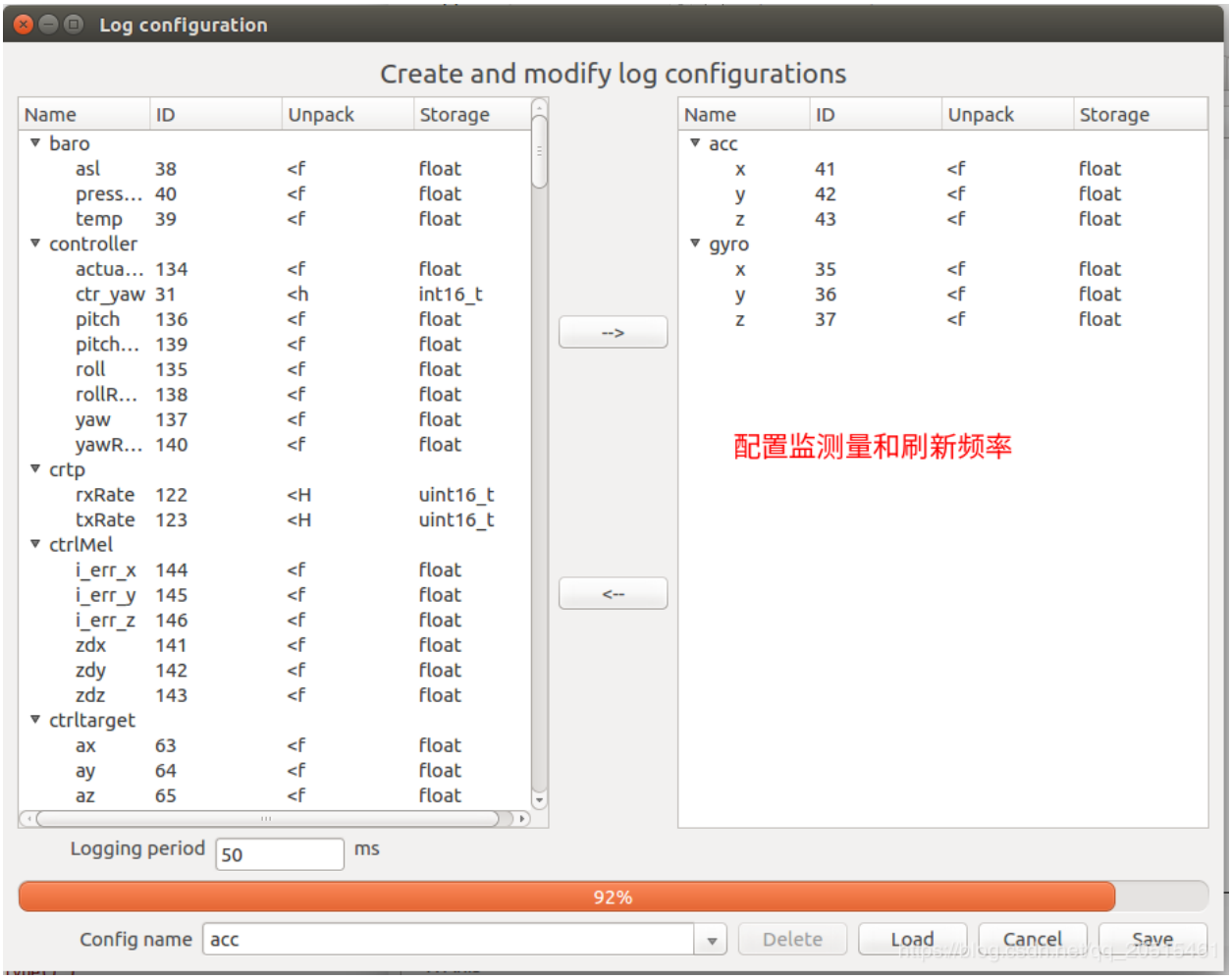

Log configuration
Create and modify log configurations
Name
ID
Unpack
Storage
baro
asl 38 <f float
press... 40 <f float
temp 39 <f float
controller
actua... 134 <f float
ctr_yaw 31 <h int16_t
pitch 136 <f float
pitch... 139 <f float
roll 135 <f float
rollR... 138 <f float
yaw 137 <f float
yawR... 140 <f float
crtp
rxRate 122 <H uint16_t
txRate 123 <H uint16_t
ctrlMel
i_err_x 144 <f float
i_err_y 145 <f float
i_err_z 146 <f float
zdx 141 <f float
zdy 142 <f float
zdz 143 <f float
ctrltarget
ax 63 <f float
ay 64 <f float
az 65 <f float
-->
<--
acc
x 41 <f float
y 42 <f float
z 43 <f float
gyro
x 35 <f float
y 36 <f float
z 37 <f float
配置监测量和刷新频率
Logging period
50
ms
92%
Config name
acc
Delete
Load
Cancel
Save

图 10: 监控参数配置

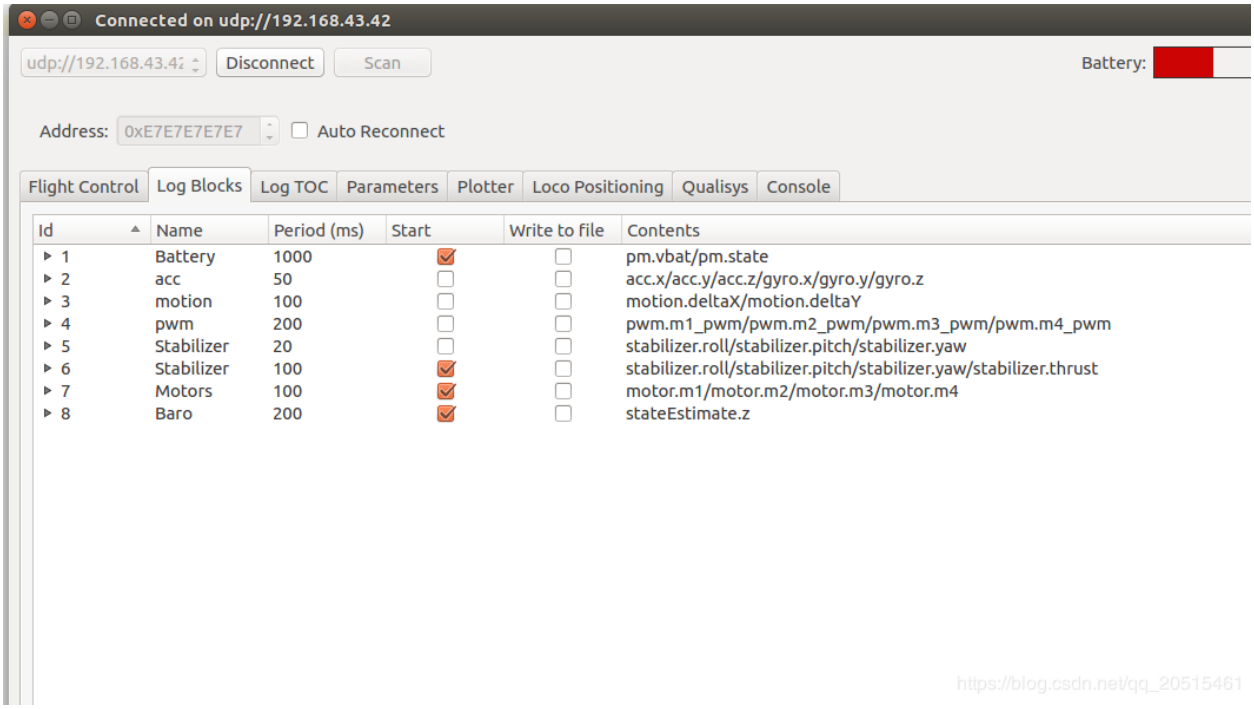


图 11: 参数配置区

3.1.7 螺旋桨方向

- 按照下图所示位置，安装 A、B 螺旋桨。
- 飞行器上电自检时，检查螺旋桨转向是否正确。

3.1.8 起飞前检查

- 将小飞机头部朝前放置，尾部天线朝向自己；
- 将小飞机置于水平面上，待机身稳定时上电；
- 观察上位机水平面是否置平；
- 观察通信建立以后，小飞机尾部绿灯是否快速闪烁；
- 观察小飞机头部红灯是否熄灭，亮起代表电量不足；
- 轻推左手小油门，检查飞机是否能快速响应；
- 轻推右手方向，检查方向控制是否正确；
- 起飞吧！

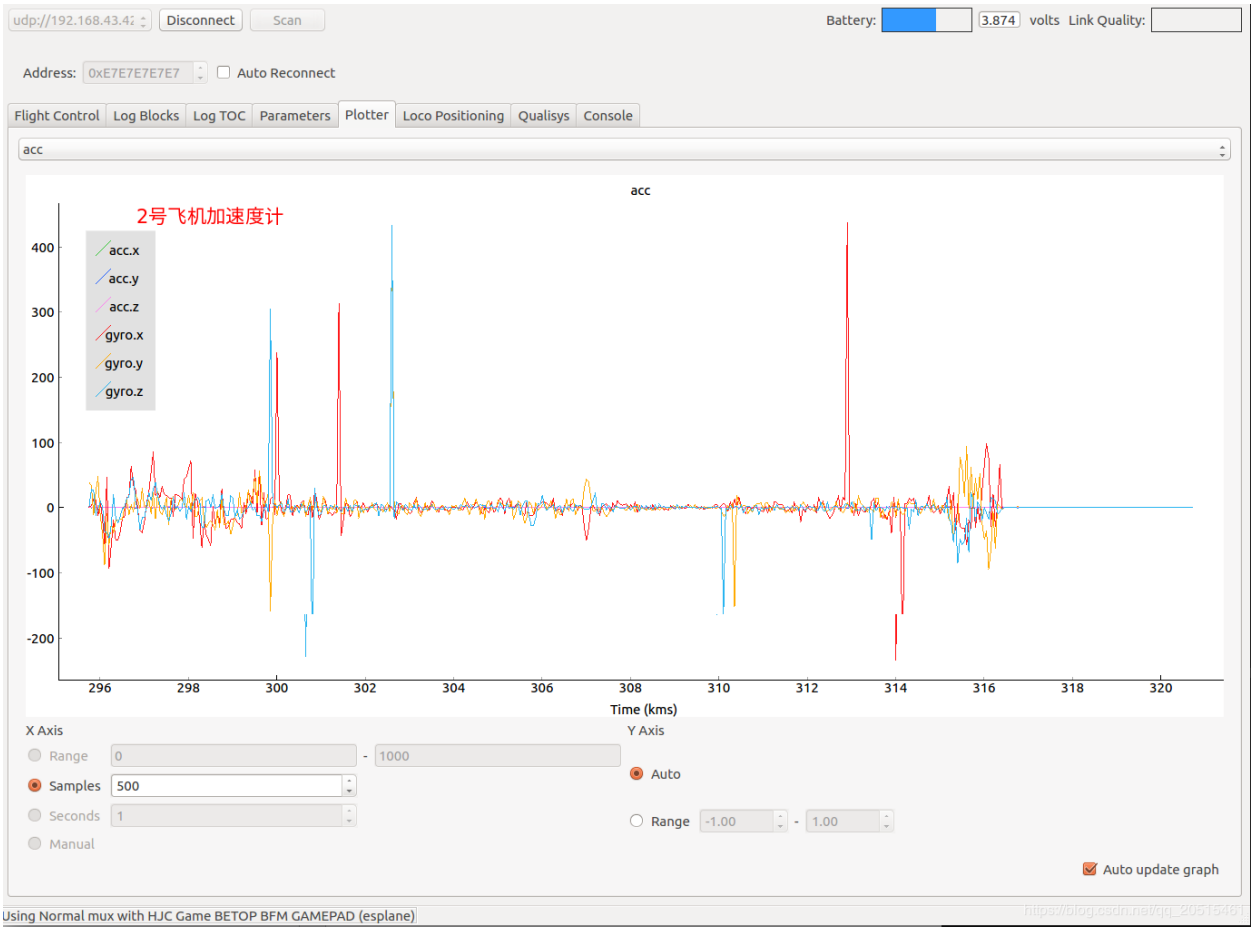


图 12: 陀螺仪加速度计数据监测

图 13: 螺旋桨方向示图

3.2 搭建开发环境

[English]

3.2.1 ESP-IDF 环境搭建

请参照 ESP-IDF 编程指南，按照步骤设置 ESP-IDF。

注意事项：

- 推荐安装 ESP-IDF *release/v4.4* 分支；
- 请完成 ESP-IDF 所有安装步骤;
- 建议首先编译一个 ESP-IDF 示例程序，用以检查安装的完整性。

3.2.2 获取项目源代码

测试版本代码，目前放在 GitHub 仓库，可使用 git 工具获取：

```
git clone https://github.com/espressif/esp-drone.git
```

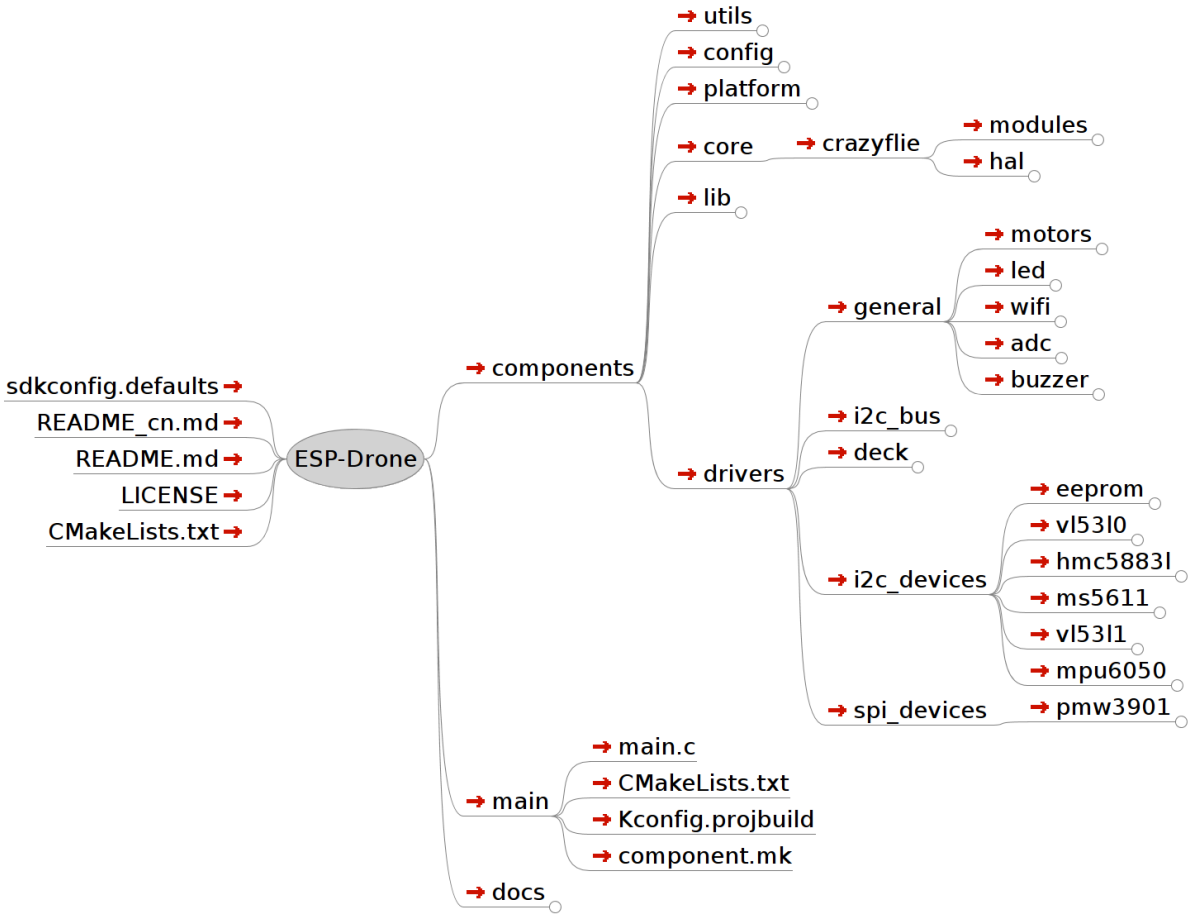
项目软件主要由飞控内核、硬件驱动和依赖库组成：

- 飞控内核来自 Crazyflie 开源工程，主要包括硬件抽象层和飞控程序。
- 硬件驱动按照硬件接口进行了文件结构划分，包括 I2C 设备和 SPI 设备等。
- 依赖库包括 ESP-IDF 提供的默认组件，以及来自第三方的 DSP 等。

代码文件结构如下所示：

```
.
├── components                          | 项目组件目录
│   ├── config                                | 系统 task 配置
│   │   └── include
│   ├── core                                   | 系统内核目录
│   │   └── crazyflie                    | crazyflie 内核
│   │       ├── hal                           | 硬件抽象代码
│   │       └── modules                |   飞行控制代码
│   ├── drivers                               | 硬件驱动目录
│   │   ├── deck                             | 硬件扩展接口驱动
│   │   ├── general                         | 一般设备目录
│   │   │   ├── adc                          | ADC 驱动，用于电压监测
│   │   │   ├── buzzer                     | 蜂鸣器驱动，用于状态反馈
│   │   │   ├── led                          | LED 驱动，用于状态反馈
│   │   │   ├── motors                     | 电机驱动，用于推力输出
```

(下页继续)


ESP-Drone
sdkconfig.defaults
README_cn.md
README.md
LICENSE
CMakeLists.txt
components
utils
config
platform
core
crazyflie
modules
hal
lib
drivers
general
motors
led
wifi
adc
buzzer
i2c_bus
deck
i2c_devices
eeprom
vl53l0
hmc5883l
ms5611
vl53l1
mpu6050
spi_devices
pmw3901
main
main.c
CMakeLists.txt
Kconfig.projbuild
component.mk
docs

(续上页)

```
|   |   |   └── wifi                    | Wi-Fi 驱动，用于通信
|   |   ├── i2c_bus                   | I2C 驱动
|   |   ├── i2c_devices             | I2C 设备目录
|   |   |   ├── eeprom              | eeprom 驱动，用于参数存储
|   |   |   ├── hmc5883l            | hmc5883l 磁罗盘传感器
|   |   |   ├── mpu6050             | mpu6050 陀螺仪加速度计传感器
|   |   |   ├── ms5611              | ms5611 气压传感器
|   |   |   ├── vl53l0                  | Vl53l0 激光传感器（最大测距 2 m）
|   |   |   └── vl53l1                  |  Vl53l1 激光传感器（最大测距 4 m）
|   |   └── spi_devices             | SPI 设备目录
|   |       └── pmw3901             | pmw3901 光流传感器
|   ├── lib                                   | 外部库目录
|   |   └── dsp_lib                     | dsp 库
|   ├── platform                          | 用于支持多平台
|   └── utils                                 | 工具函数目录
├── CMakeLists.txt                    | 工具函数
├── LICENSE                             | 开源协议
├── main                                    | 入口函数
├── README.md                         | 项目说明
└── sdkconfig.defaults              | 默认参数
```

详情可查阅：espdrone_file_structure

3.2.3 源代码风格

两种方式检索同一区域（union）

实现两种检索方式检索同一片内存区域，可以使用：

```
typedef union {
  struct {
        float x;
        float y;
        float z;
  };
  float axis[3];
} Axis3f;
```

使用枚举类型计数

以下枚举类型成员 SensorImplementation_COUNT，始终可以代表枚举类型中成员的个数。巧妙利用了枚举类型第一个成员默认为 0 的特点。

```
typedef enum {
  #ifdef SENSOR_INCLUDED_BMI088_BMP388
  SensorImplementation_bmi088_bmp388,
  #endif

  #ifdef SENSOR_INCLUDED_BMI088_SPI_BMP388
  SensorImplementation_bmi088_spi_bmp388,
  #endif

  #ifdef SENSOR_INCLUDED_MPU9250_LPS25H
  SensorImplementation_mpu9250_lps25h,
  #endif

  #ifdef SENSOR_INCLUDED_MPU6050_HMC5883L_MS5611
  SensorImplementation_mpu6050_HMC5883L_MS5611,
  #endif

  #ifdef SENSOR_INCLUDED_BOSCH
  SensorImplementation_bosch,
  #endif

  SensorImplementation_COUNT,
} SensorImplementation_t;
```

紧凑的数据类型

```
struct cppmEmuPacket_s {
  struct {
      uint8_t numAuxChannels : 4;   // Set to 0 through MAX_AUX_RC_CHANNELS
      uint8_t reserved : 4;
  } hdr;
  uint16_t channelRoll;
  uint16_t channelPitch;
  uint16_t channelYaw;
  uint16_t channelThrust;
  uint16_t channelAux[MAX_AUX_RC_CHANNELS];
} __attribute__((packed));
```

`__attribute__((packed))` 的作用是：使编译器取消结构在编译过程中的优化对齐，而按照实际占用字节数进行对齐。这是 GCC 特有的语法，与操作系统无关，与编译器有关。GCC 和 VC（在 Windows 下）的编译器为非紧凑模式，TC 的编译器为紧凑模式。例如：

```
在 TC 下：struct my{ char ch; int a;} sizeof(int)=2;sizeof(my)=3;（紧凑模式）
在 GCC 下：struct my{ char ch; int a;} sizeof(int)=4;sizeof(my)=8;（非紧凑模式）
```

(下页继续)

(续上页)

```
在 GCC 下：struct my{ char ch; int a;}__attrubte__ ((packed)) sizeof(int)=4;sizeof(my)=5
```

3.3 开发指引

[English]

3.3.1 硬件参考

[English]

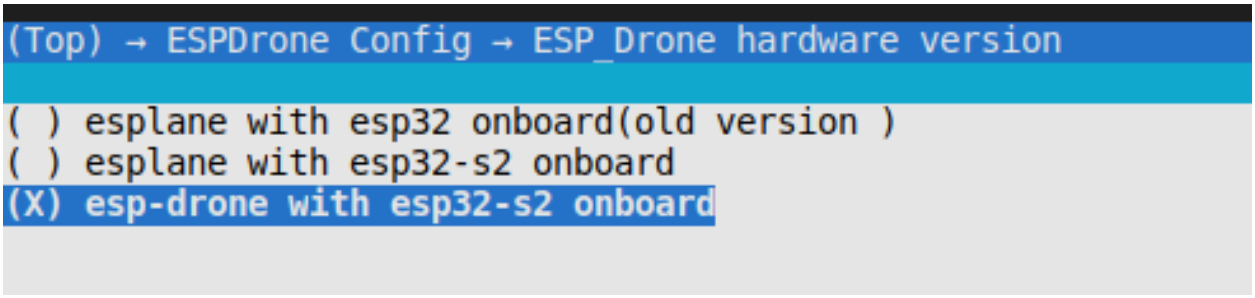
已支持硬件

已支持硬件清单

开发板名	主要配置	备注
ESP32-S2-Drone V1.2	ESP32-S2-WROVER + MPU6050	一体化
ESPlane-V2-S2	ESP32-S2-WROVER + MPU6050	需安装脚架
ESPlane-FC-V1	ESP32-WROOM-32D + MPU6050	需安装机架

硬件切换方法

- `esp_drone` 仓库代码已支持多种硬件，可通过 `menuconfig` 进行切换，见下图。



- 默认情况下，`set-target` 设为 `esp32s2` 后，硬件自动切换为 `ESP32_S2_Drone_V1_2`。
- 默认情况下，`set-target` 设为 `esp32` 后，硬件自动切换为 `ESPlane_FC_V1`。

注意事项

1. ESPlane-FC-V1 为老版本硬件。

2. ESPlane-FC-V1 使用 ESP-Drone 新版本代码，需要对硬件进行改动，即使用跳线，将模组 GPIO14 连接到 MPU6050 INT 管脚。
3. ESPlane-FC-V1 防止上电时 IO12 触发 flash 电压切换，使用 `espefuse.py` 将 flash 电压固定到 3.3 V：

   ```
   espefuse.py --port /dev/ttyUSB0 set_flash_voltage 3.3V
   ```

   注意，仅有第一个连接到总线的设备可以使用 CS0 管脚。

ESP32-S2-Drone V1.2

图 14: ESP32-S2-Drone V1.2 外观图

- 主板原理图：SCH_Mainboard_ESP32_S2_Drone_V1_2

- 主板 PCB：PCB_Mainboard_ESP32_S2_Drone_V1_2

基础配置

基础配置清单

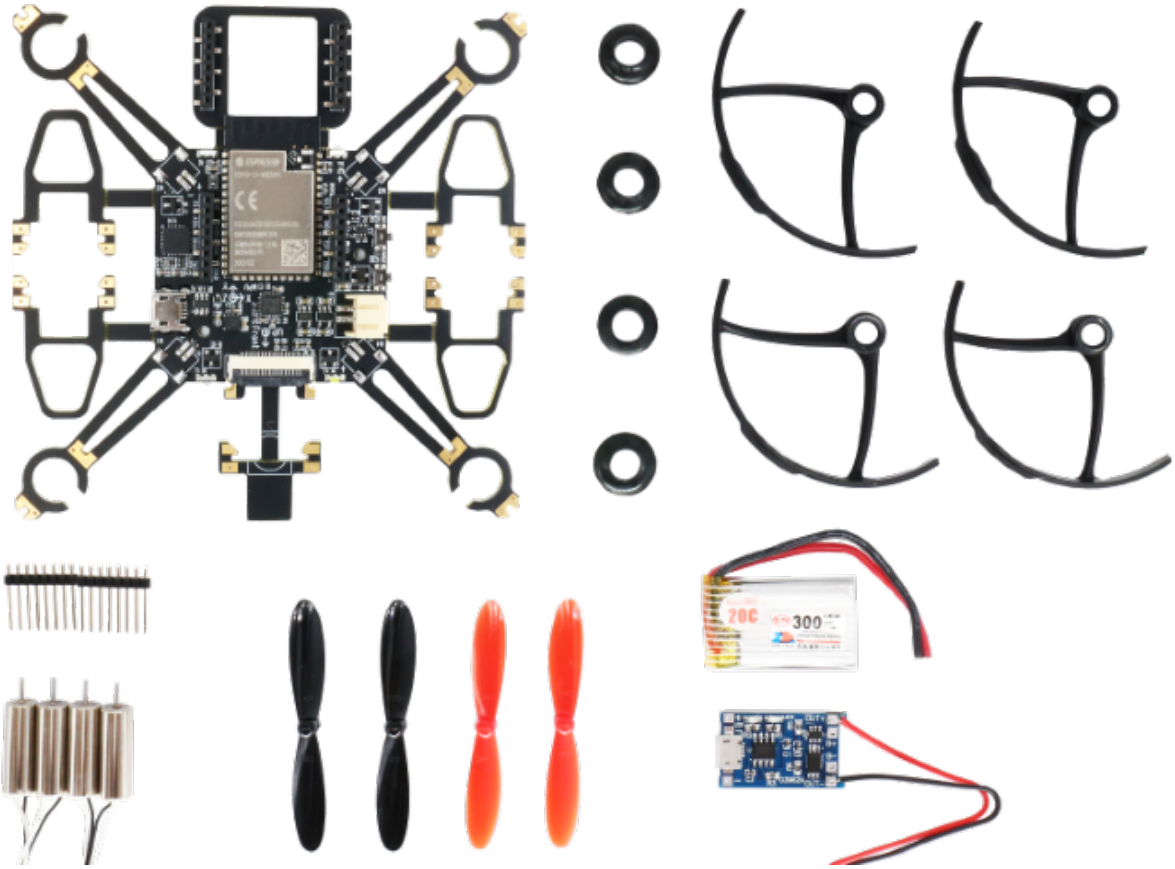


图 15: ESP32-S2-Drone V1.2 基础配置清单

基础配置清单	数量	备注
主板	1	ESP32-S2-WROVER + MPU6050
716 电机	4	可配置 720 电机
716 电机橡胶圈	4	
46mm 螺旋桨 A	2	可配置 55mm 桨
46mm 螺旋桨 B	2	
300mAh 1s 锂电池	1	可配置 350mAh 高倍率
1s 锂电池充电板	1	
8-pin 25 mm 排针	2	

注解： 更换720电机之后，需要在`menuconfig->ESPDrone Config->motors config`将`motor type`修改为`brushed 720 motor`。

主控制器

芯片型号	模组型号	备注
ESP32-S2	ESP32-S2-WROVER	模组内置 4 MB flash，2 MB PSRAM

传感器

传感器	接口	备注
MPU6050	I2C0	主板传感器

指示灯

状态	LED	动作
POWER_ON	WHITE	常亮
SENSORS CALIBRATION	BLUE	慢速闪烁
SYSTEM READY	BLUE	正常闪烁
UDP_RX	GREEN	闪烁
LOW_POWER	RED	常亮

按键

按键	IO	功能
SW1	GPIO1	Boot, Normal
SW2	EN	Reset

主板 IO 定义

管脚	功能	备注
GPIO11	I2C0_SDA	MPU6050 专用
GPIO10	I2C0_SCL	MPU6050 专用
GPIO37	SPI_MISO	MISO
GPIO35	SPI_MOSI	MOSI
GPIO36	SPI_CLK	SCLK
GPIO34	SPI_CS0	CS0*
GPIO40	I2C1_SDA	VL53L1X
GPIO41	I2C1_SCL	VL53L1X
GPIO12	interrupt	MPU6050 interrupt
GPIO39	BUZ_1	BUZZ+
GPIO38	BUZ_2	BUZZ-
GPIO8	LED_RED	LED_1
GPIO9	LED_GREEN	LED_2
GPIO7	LED_BLUE	LED_3
GPIO5	MOT_1	
GPIO6	MOT_2	
GPIO3	MOT_3	
GPIO4	MOT_4	
GPIO2	ADC_7_BAT	VBAT/2
GPIO1	EXT_IO1	

摄像头接口

管脚	功能
GPIO13	CAM_VSYNC
GPIO14	CAM_HREF
GPIO15	CAM_Y9
GPIO16	CAM_XCLK
GPIO17	CAM_Y8
GPIO18	CAM_RESET
GPIO19	CAM_Y7
GPIO20	CAM_PCLK
GPIO21	CAM_Y6
GPIO33	CAM_Y2
GPIO45	CAM_Y4
GPIO46	CAM_Y3

扩展配置

扩展板	主要传感器	功能	接口	安装位置
扩展板- 定点模块	PMW3901 + VL53L1X	室内定点飞行	SPI + I2C	底部，面向地面
扩展板-气压定高模块	MS5611 气压	气压定高	I2C 或 MPU6050 从机	顶部或底部
扩展板 -指南针模块	HMC5883 罗盘	无头模式等高级模式	I2C 或 MPU6050 从机	顶部或底部

扩展板 IO 定义

左管脚	IO	右管脚	IO
SPI_CS0	GPIO34	VDD_33	IO
SPI_MOSI	GPIO35	I2C0_SDA	GPIO11
SPI_CLK	GPIO36	I2C0_SCL	GPIO10
SPI_MISO	GPIO37	GND	
GND		AUX_SCL	
I2C1_SDA	GPIO40	AUX_SDA	
I2C1_SCL	GPIO41	BUZ_2	GPIO38
EXT_IO1	GPIO1	BUZ_1	GPIO39

ESPlane-V2-S2

- 主板原理图：SCH_ESPlane_V2_S2
- 主板 PCB：PCB_ESPlane_V2_S2

ESPlane-FC-V1

- 主板原理图：Schematic_ESPlane_FC_V1
- 主板 PCB：PCB_ESPlane_FC_V1

图 16: ESPlane-V2-S2 外观图

图 17: ESPlane-FC-V1 外观图

基础配置

基础配置清单

配置清单	数量	备注
主板	1	ESP32-WROOM-32D + MPU6050
机架	1	
46 mm 螺旋桨 A	2	
46 mm 螺旋桨 B	2	
300 mAh 1s 锂电池	1	
1s 锂电池充电板	1	

传感器

传感器	接口	备注
MPU6050	I2C0	必需

指示灯

```
#define LINK_LED          LED_BLUE
//#define CHG_LED           LED_RED
#define LOWBAT_LED        LED_RED
//#define LINK_DOWN_LED   LED_BLUE
#define SYS_LED           LED_GREEN
#define ERR_LED1          LED_RED
#define ERR_LED2          LED_RED
```

状态	LED	动作
SENSORS READY	BLUE	常亮
SYSTEM READY	BLUE	常亮
UDP_RX	GREEN	闪烁

主板 IO 定义

管脚	功能	备注
GPIO21	SDA	I2C0 数据
GPIO22	SCL	I2C0 时钟
GPIO14	SRV_2	MPU6050 中断
GPIO16	RX2	
GPIO17	TX2	
GPIO27	SRV_3	BUZZ+
GPIO26	SRV_4	BUZZ-
GPIO23	LED_RED	LED_1
GPIO5	LED_GREEN	LED_2
GPIO18	LED_BLUE	LED_3
GPIO4	MOT_1	
GPIO33	MOT_2	
GPIO32	MOT_3	
GPIO25	MOT_4	
TXD0		
RXD0		
GPIO35	ADC_7_BAT	VBAT/2

扩展配置

ESPlane + PMW3901 管脚配置

管脚	功能	备注
GPIO21	SDA	I2C0 数据
GPIO22	SCL	I2C0 时钟
GPIO12	MISO/SRV_1	HSPI
GPIO13	MOSI	HSPI
GPIO14	SCLK/SRV_2	HSPI [STRIKEOUT:MPU6050 中断]
GPIO15	CS0*	HSPI
GPIO16	RX2	
GPIO17	TX2	
GPIO19	interrupt	MPU6050 中断
GPIO27	SRV_3	BUZZ+
GPIO26	SRV_4	BUZZ-
GPIO23	LED_RED	LED_1
GPIO5	LED_GREEN	LED_2
GPIO18	LED_BLUE	LED_3
GPIO4	MOT_1	
GPIO33	MOT_2	
GPIO32	MOT_3	
GPIO25	MOT_4	
TXD0		
RXD0		
GPIO35	ADC_7_BAT	VBAT/2

3.3.2 驱动程序

[English]

本节将主要介绍 ESP-Drone 中使用到的 I2C 驱动和 SPI 驱动。

I2C 设备驱动 (i2c_devices)

I2C 驱动涵盖 MPU6050 传感器驱动和 VL53LXX 传感器驱动。后续部分将从传感器主要特性、关键寄存器和编程注意事项进行介绍。

MPU6050 传感器

概述

MPU6050 是一款整合性 6 轴运动处理组件，内带 3 轴陀螺仪、3 轴加速度传感器和数字运动处理器（DMP: Digital Motion Processor）。

工作原理

- 陀螺仪：当陀螺仪围绕任何感应轴旋转时，由于科里奥利效应，会产生振动。这种振动可以被电容式传感器检测到，传感器所得到的信号被放大，解调和滤波，产生与角速度成比例的电压。
- 电子加速度计：加速沿着一条特定轴在相应的检测质量上位移，则电容式传感器会检测到电容的变化。

测量范围

- 可配置陀螺仪测量范围：±250 °/sec、±500 °/sec、±1000 °/sec、±2000 °/sec
- 可配置加速度计测量范围：±2 g、±4 g、±8 g、±16 g

AUX I2C 接口

- MPU6050 具有一个辅助 I2C 总线，用于与片外 3-轴数字磁力计或其它传感器进行通信。
- AUX I2C 接口支持两种工作模式：I2C Master 模式或 Pass-Through 模式。

MPU6050 FIFO

MPU6050 包含一个可通过串行接口访问的 1024 字节 FIFO 寄存器。FIFO 配置寄存器决定哪个数据写入 FIFO，包括：陀螺仪数据、加速计数据、温度读数、辅助传感器读数和 FSYNC 输入。

数字低通滤波器 (DLPF)

MPU6050 自带低通滤波器，可通过配置寄存器 26 控制低通滤波频段，减少高频干扰，但会降低传感器输入速率。开启 DLPF 加速度计输出 1 kHz，关闭 DLPF 可以输出 8 kHz。

FSYNC 帧同步采样管脚

寄存器 26 EXT_SYNC_SET，用于配置外部帧同步管脚的采样。

数字运动处理器 (DMP)

- MPU6050 内部存在一个数字运动处理单元 (Digital Motion Processor, DMP)，可以计算四元数等，减轻主 CPU 压力。
- DMP 可通过管脚触发中断。

MPU6050 方向定义

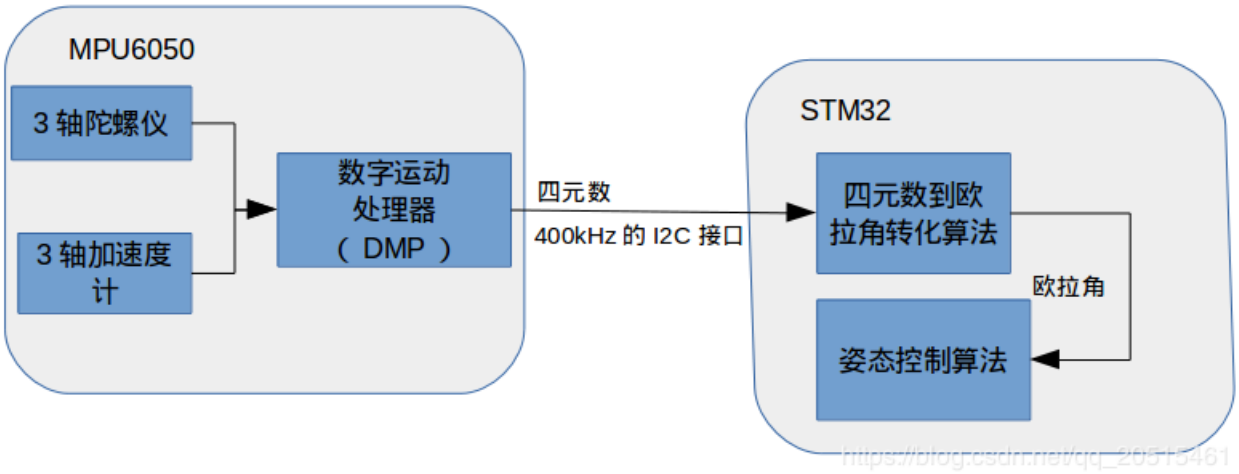


图 18: MPU6050 DMP

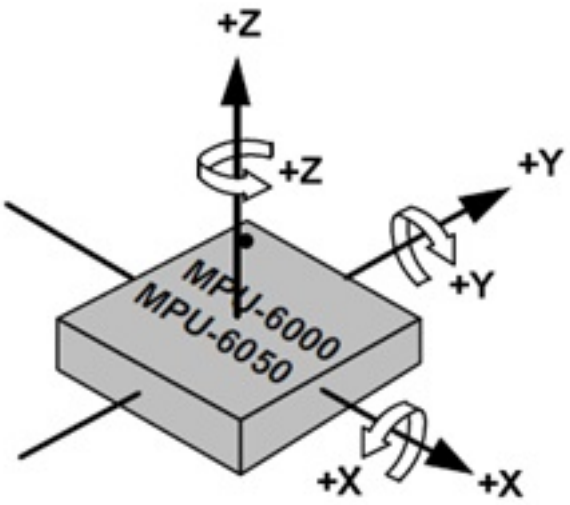


图 19: MPU6050 X、Y、Z 方向

MPU6050 初始化步骤

1. 恢复寄存器默认值：设置 PWR_MGMT_1 bit7 为 1，恢复后 bit7 为 0，bit6 自动设置为 1，进入 sleep 模式；
2. 设置 PWR_MGMT_1 bit6 为 0，唤醒传感器；
3. 设置时钟源；
4. 设置量程：分别设置陀螺仪和加速度计量程；
5. 设置采样率；
6. 设置数字低通滤波器（可选）。

MPU6050 关键寄存器

寄存器典型值

寄存器	典型值	功能
PWR_MGMT_1	0x00	正常启用
SMPLRT_DIV	0x07	陀螺仪采样率 125 Hz
CONFIG	0x06	低通滤波器频率为 5 Hz
GYRO_CONFIG	0x18	陀螺仪不自检，输出满量程范围为 ±2000 °/s
ACCEL_CONFIG	0x01	加速度计不自检，输出的满量程范围为 ±2 g

寄存器 117 WHO_AM_I - 设备地址

[6:1] 保存设备地址，默认为 0x68，不反映 AD0 管脚值。

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

寄存器 107 PWR_MGMT_1 - 电源管理 1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

- DEVICE_RESET：置位此位，则寄存器使用默认值。
- SLEEP：置位此位，则 MPU6050 将置于睡眠模式。

- CYCLE：置位此位，且 SLEEP 设置为禁用时，MPU6050 将进入循环模式 (CYCLE)。在循环模式下，MPU6050 进入睡眠，达到 LP_WAKE_CTRL（寄存器 108）设定的时间后，从睡眠模式唤醒，完成一次对活动传感器的采样，然后再进入睡眠模式，依次循环。

寄存器 26 CONFIG - 配置数字低通滤波器

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1A	26	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		

- 数字低通滤波器 (DLPF) 取值与滤波频段关系如下：

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

寄存器 27 - GYRO_CONFIG 陀螺仪量程配置

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

- XG_ST：X 轴陀螺仪自检
- FS_SEL：用于配置陀螺仪量程，具体信息见下表：

寄存器 28 ACCEL_CONFIG - 配置加速度计量程

寄存器 25 SMPRT_DIV - 采样速率分频器

该寄存器指定陀螺仪输出速率的分频器，用于为 MPU6050 生成采样速率。传感器寄存器的输出、FIFO 输出和 DMP 采样都是基于采样率。陀螺仪输出速率除以 (1 + SMPLRT_DIV) 得到采样率，公式如下：

Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Type: Read/Write

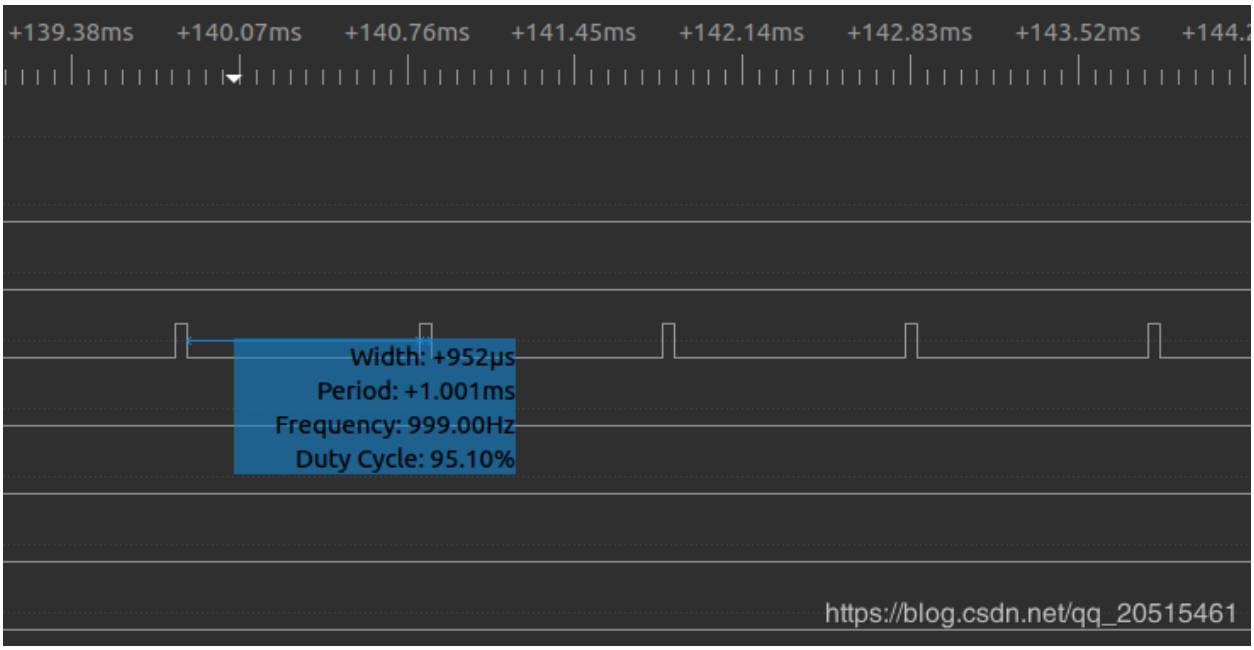
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
19	25	SMPLRT_DIV[7:0]							

其中，当 DLPF 禁用时，即 DLPF_CFG = 0 或 7 时，陀螺仪输出速率为 8 kHz；当 DLPE 使能时，见寄存器 26，陀螺仪输出速率为 1 kHz。注意，在不开启 DLPF 的情况下，设置 SMPLRT_DIV 为 7 可以使芯片产生 1kHz 的中断信号。



寄存器 59 ~ 64 - 加速度计测量值

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

- 大端序存放：地址低位存放数据高位，地址高位存放数据低位。
- 补码存放：测量值为有符号整数，因此采用补码方式存放。

寄存器 65 ~ 66 - 温度测量

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

寄存器 67 ~ 72 - 陀螺仪测量值

Type: Read Only

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

VL53LXX 传感器

概述

VL53L1X 是 ST 公司提供的一款 ToF 测距和姿态检测传感器。

工作原理

VL53L0X/VL53L1X 芯片内部集成了激光发射器和 SPAD 红外接收器。芯片通过探测光子发送和接收时间差，计算光子飞行距离，最远测量距离可达两米，适合中短距离测量的应用。

图 20: VL53LXX

测量区域 - ROI

VL53L0X/VL53L1X 的测量值为测量区域中的最短距离，测量区域可以根据使用场景进行放大或缩小，较大的探测范围可能会引起测量值的波动。

测量区域的配置参见 VL53LXX Datasheet 中 2.4 Ranging Description 和 2.8 Sensing Array Optical Center。

测量距离

- VL53L0X 传感器存在 **3 ~ 4 cm 的盲区**，有效测量范围为 3 ~ 200 cm，精度 ±3%。
- VL53L1X 是 VL53L0X 的升级版本，探测距离可达 400 cm。
- VL53LXX 测量距离与光线环境有关，黑暗环境下探测距离更远；在室外强光下，激光传感器可能会受到很大的干扰，导致测量精度降低，因此室外需要结合气压定高。

测量频率

- VL53L0X 响应频率最快可达 50 Hz，测量误差 ±5%。

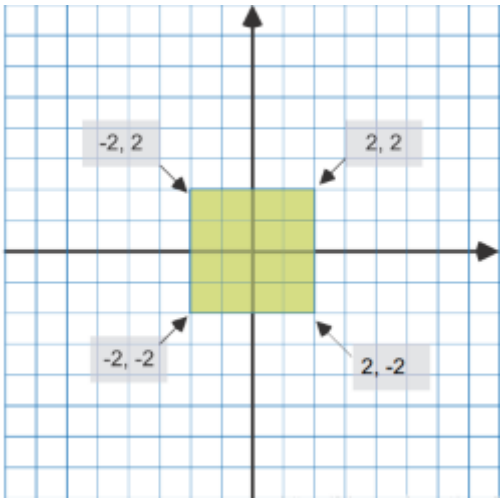


图 21: ROI

精度模式	测量时间预算范围（ms）	测距性能（m）	典型应用
默认	30	1.2	标准
高精度	200	1.2　精度<±3%	精确测量
长距离	33	2	长距离，只适用于黑暗条件（无红外线）
高速	20	1.2　精度　±5%	高速，精度不优先

Table 4. Maximum distance vs. Distance mode under ambient light

Distance mode	Max. distance in dark (cm)	Max. distance under strong ambient light (cm)
Short	136	135
Medium	290	76
Long	360	73

Test conditions: timing budget = 100 ms, white target 88 %, dark = no IR ambient, ambient light = 200 kcps/SPAD.

- VL53L1X I2C 最高时钟频率可达 400 kHz，上拉电阻需要根据电压和总线电容值选择。具体信息请参考 VL53LXX Datasheet。

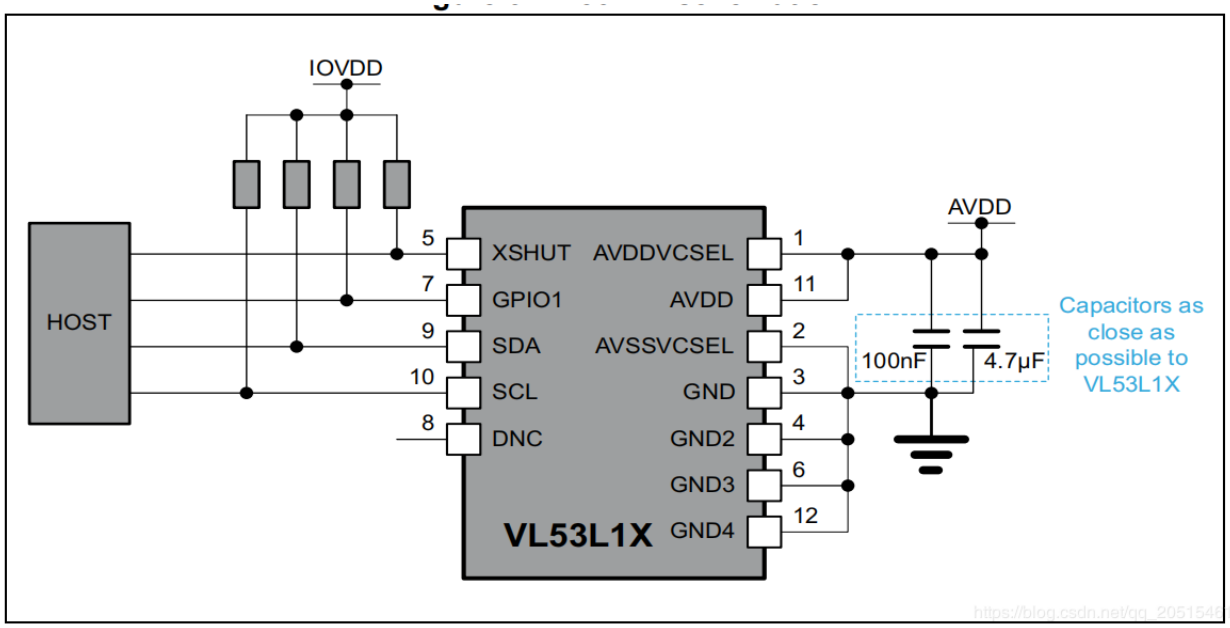


图 22: VL53L1X

- XSHUT 为输入管脚，用于模式选择（休眠），需要上拉电阻防止漏电流。
- GPIO1 为中断输出管脚，用于输出测量 dataready 中断。

工作模式

通过设置 XSHUT 管脚的电平，可以切换传感器进入 HW Standby 模式或 SW Standby 模式，实现有条件启动，降低待机功耗。如果主机放弃管理传感器模式，可将 XSHUT 管脚默认设置为上拉。

- HW Standby：XSHUT 拉低，传感器电源关闭。
- SW Standby：XSHUT 拉高，进入 boot 和 SW Standby 模式。

VL53LXX 初始化步骤

1. 等待硬件初始化完成；
2. 数据初始化；
3. 静态初始化，装载数据；
4. 设置测量距离模式；
5. 设置单次测量最长等待时间；
6. 设置测量频率（时间间隔）；
7. 设置测量区域 ROI（可选）；

Figure 7. Power up and boot sequence

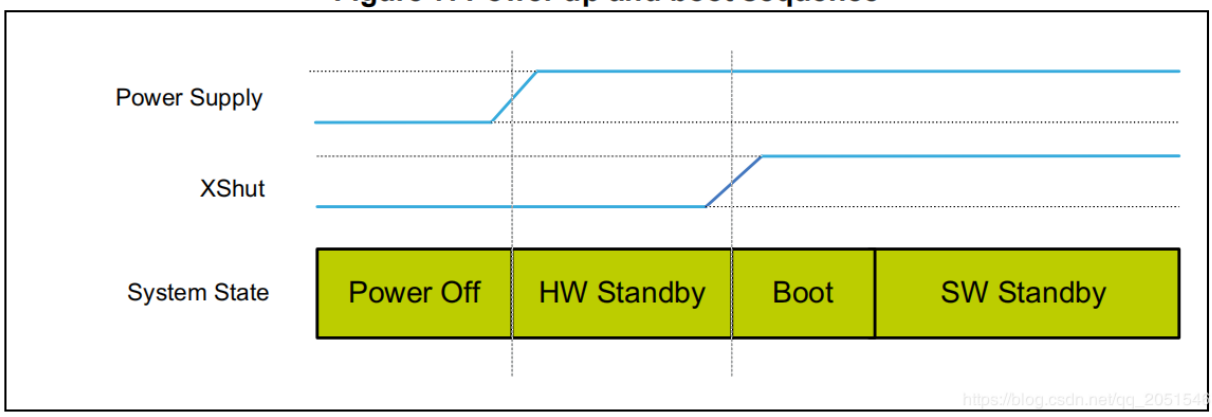
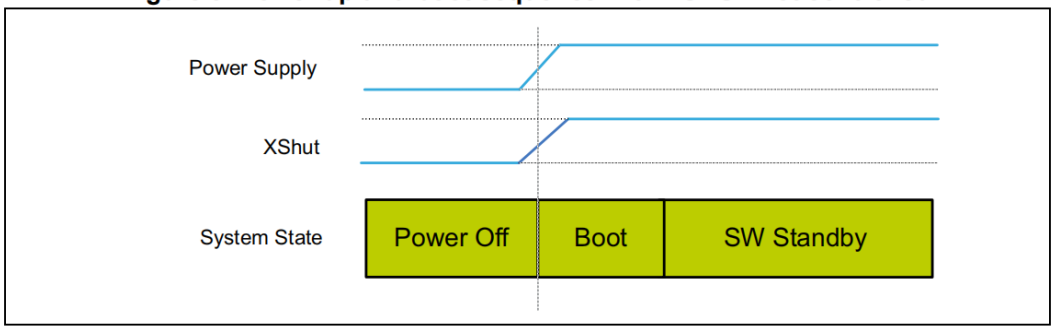


图 23: HW Standby

Figure 8. Power up and boot sequence with XSHUT not controlled



Note: Boot duration is 1.2 ms max.

Note: In all cases, XSHUT has to be raised only when the power supply is tied on.

图 24: SW Standby

8. 启动测量。

```
/*init  vl53l1 module*/
void vl53l1_init()
{

    Roi0.TopLeftX = 0;     //测量目标区（可选）最小 4*4，最大 16*16。
    Roi0.TopLeftY = 15;
    Roi0.BotRightX = 7;
    Roi0.BotRightY = 0;
    Roi1.TopLeftX = 8;
    Roi1.TopLeftY = 15;
    Roi1.BotRightX = 15;
    Roi1.BotRightY = 0;

    int status = VL53L1_WaitDeviceBooted(Dev); //等待硬件初始化完成。
    status = VL53L1_DataInit(Dev); //数据初始化，上电后立刻执行。
    status = VL53L1_StaticInit(Dev); //静态初始化，装载参数。
    status = VL53L1_SetDistanceMode(Dev, VL53L1_DISTANCEMODE_LONG);//设置测量模式。
    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(Dev, 50000); //根据测量模式确
定最长等待时间。
    status = VL53L1_SetInterMeasurementPeriodMilliSeconds(Dev, 100); //设置测量间隔。

    status = VL53L1_SetUserROI(Dev, &Roi0); //设置测量区域 ROI
    status = VL53L1_StartMeasurement(Dev); //启动测量
    if(status) {
        printf("VL53L1_StartMeasurement failed \n");
        while(1);
    }

}
```

注意，上述初始化步骤除 VL53L1_SetUserROI 外，其余步骤不可缺少。

VL53LXX 测距步骤

轮询测量模式

轮询测量流程图：

- 注意，完成一次测量和读取后，需要使用 `VL53L1_ClearInterruptAndStartMeasurement` 清除中断标志并重新开始。
- 轮询测量有两种方法，如上图所示：一种是阻塞方式 (Drivers polling mode)；一种是非阻塞方式 (Host polling mode)。以下代码为阻塞测量方式：

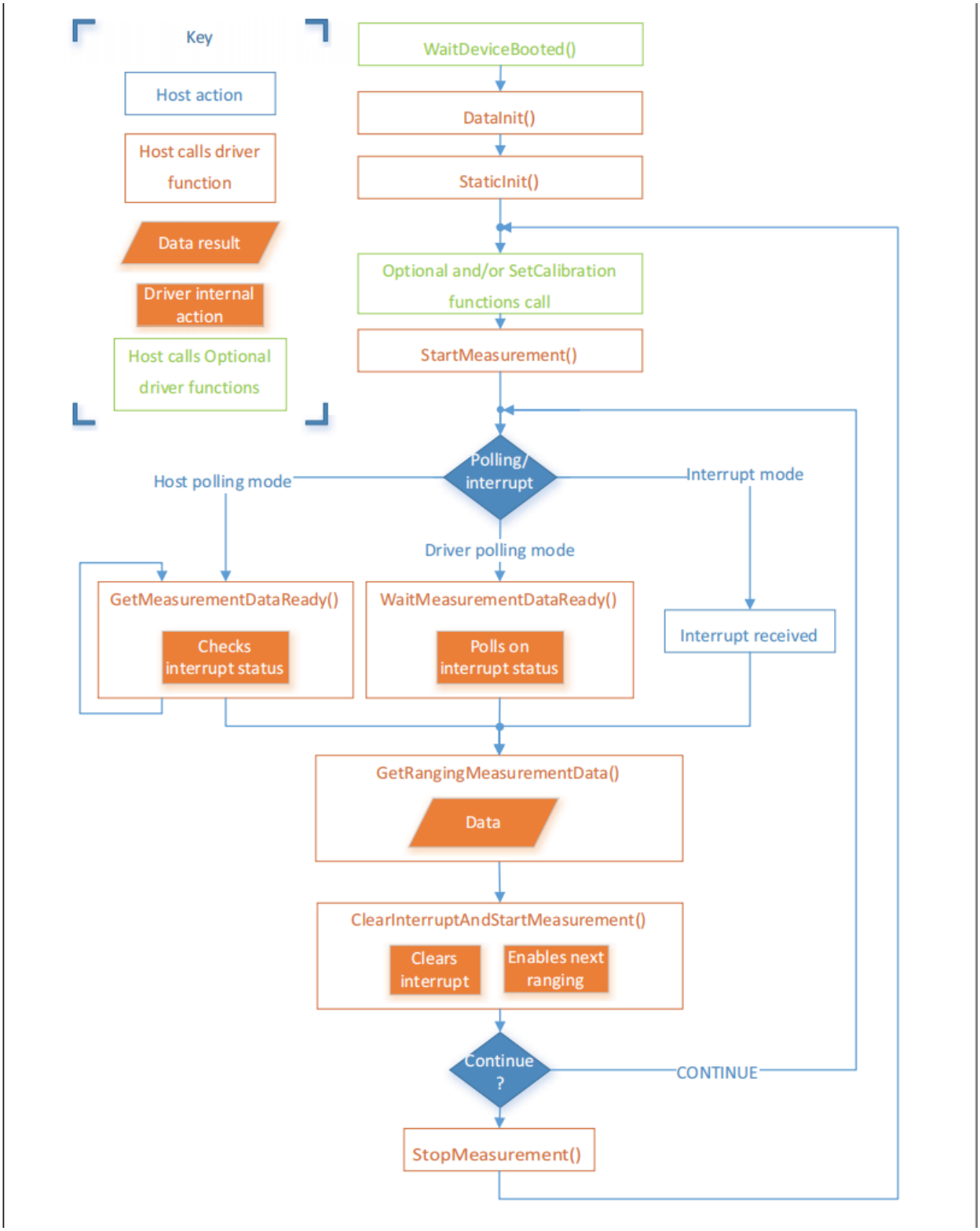


图 25: VL53LXX 测量流程

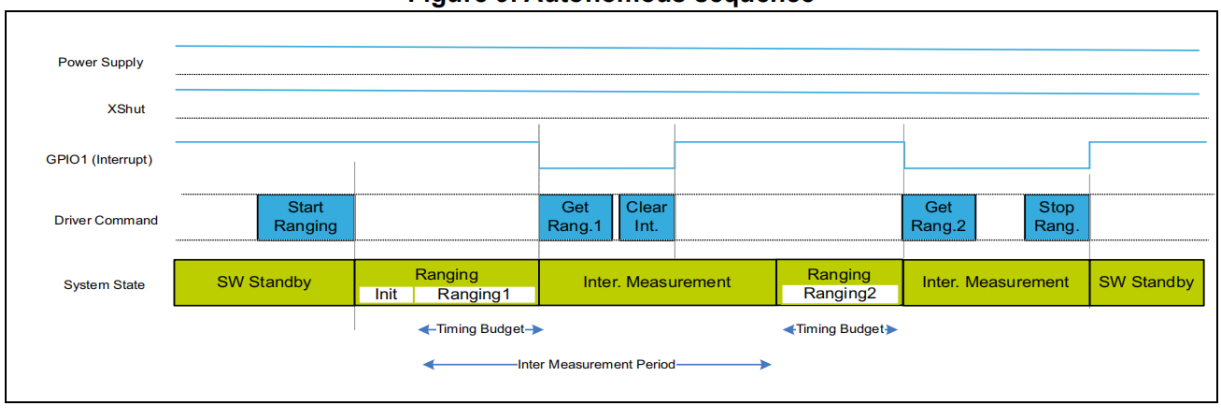
```
/* Autonomous ranging loop*/
static void
AutonomousLowPowerRangingTest(void)
{
    printf("Autonomous Ranging Test\n");

    static VL53L1_RangingMeasurementData_t RangingData;
    VL53L1_UserRoi_t Roi1;
    int roi = 0;
    float left = 0, right = 0;
    if (0/*isInterrupt*/) {
    } else {
        do // polling mode
            {
                int status = VL53L1_WaitMeasurementDataReady(Dev); //等待测量结果
                if(!status) {
                    status = VL53L1_GetRangingMeasurementData(Dev, &RangingData); //获
取单次测量数据
                    if(status==0) {
                        if (roi & 1) {
                            left = RangingData.RangeMilliMeter;
                            printf("L %3.1f R %3.1f\n", right/10.0, left/10.0);
                        } else
                            right = RangingData.RangeMilliMeter;
                    }
                    if (++roi & 1) {
                        status = VL53L1_SetUserROI(Dev, &Roi1);
                    } else {
                        status = VL53L1_SetUserROI(Dev, &Roi0);
                    }
                    status = VL53L1_ClearInterruptAndStartMeasurement(Dev); //释放中断
                }
            }
        while (1);
    }
    //  return status;
}
```

中断测量模式

中断测量模式需要使用中断管脚 GPIO1，在数据 ready 时，GPIO1 管脚电平将被拉低，通知主机读取数据。

Figure 9. Autonomous sequence



Note: Timing budget and inter measurement timings are the parameters set by the user, using a dedicated driver function.

图 26: VL53LXX 自主测量流程

VL53LXX 传感器校准

如果传感器接收器上方安装了光罩，或传感器安装在透明盖板背后，由于透光率的变化，需要对传感器进行校准。您可以根据校准流程调用 API 编写校准程序，也可以使用官方提供的 GUI 上位机直接测量出校准值。

使用官方 API 编写校准程序

校准流程：调用顺序要完全一致。

```
/*VL53L1 模块校准 */
static VL53L1_CalibrationData_t vl53l1_calibration(VL53L1_Dev_t *dev)
{
    int status;
    int32_t targetDistanceMilliMeter = 703;
    VL53L1_CalibrationData_t calibrationData;
    status = VL53L1_WaitDeviceBooted(dev);
    status = VL53L1_DataInit(dev);                                    //设备初始化
    status = VL53L1_StaticInit(dev);                                  // 为给定用例,
加载设备设置。
    status = VL53L1_SetPresetMode(dev,VL53L1_PRESETMODE_AUTONOMOUS);
    status = VL53L1_PerformRefSpadManagement(dev);
    status = VL53L1_PerformOffsetCalibration(dev,targetDistanceMilliMeter);
    status = VL53L1_PerformSingleTargetXTalkCalibration(dev,targetDistanceMilliMeter);
    status = VL53L1_GetCalibrationData(dev,&calibrationData);

    if (status)
    {
        ESP_LOGE(TAG, "vl53l1_calibration failed \n");
```

(下页继续)

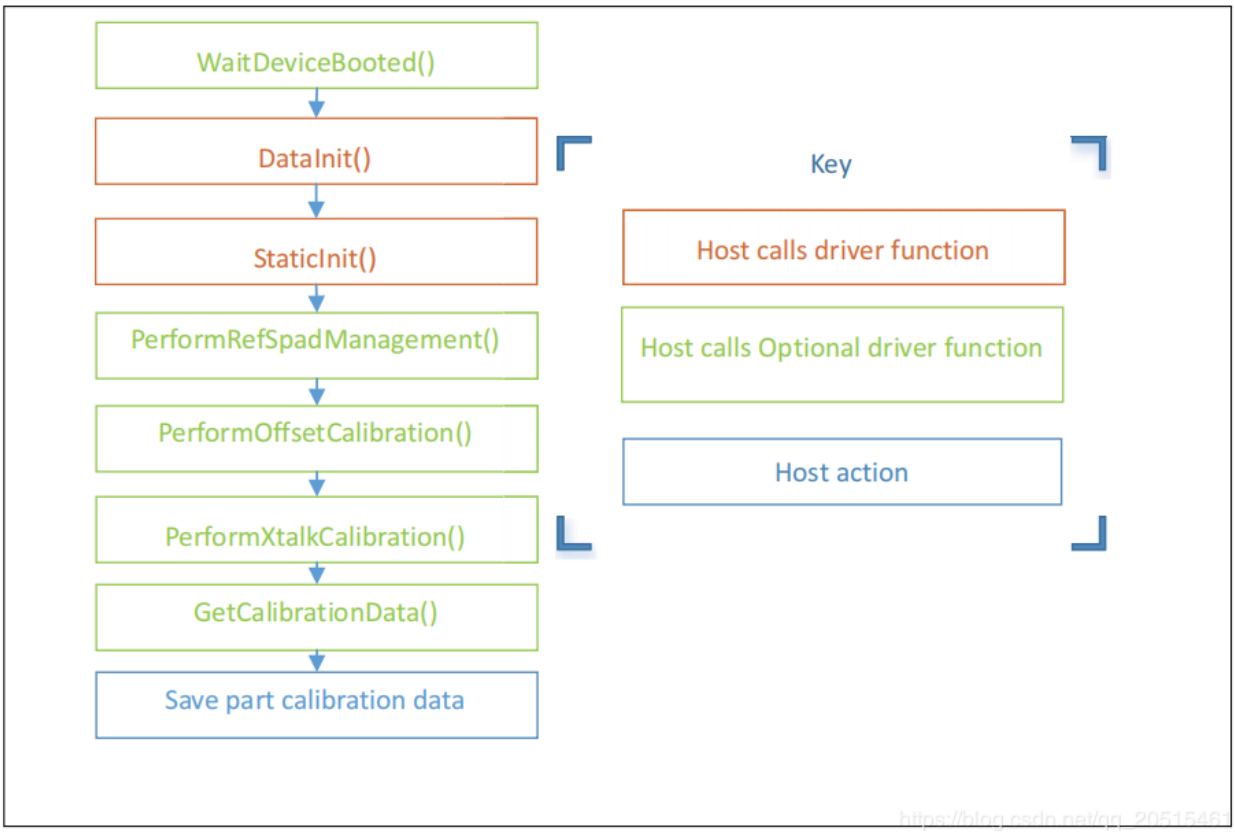


图 27: VL53LXX 校准流程

(续上页)

```
        calibrationData.struct_version = 0;
        return calibrationData;

    }else
    {
        ESP_LOGI(TAG, "vl53l1_calibration done ! version = %u \n",calibrationData.
↪struct_version);
        return calibrationData;
    }

}
```

使用官方 GUI 上位机校准传感器

官方提供了用于配置和校准传感器的 GUI 上位机，配合 ST 官方 `STM32F401RE nucleo` 开发板连接传感器，使用软件校准得到基准值后，初始化时填入即可。

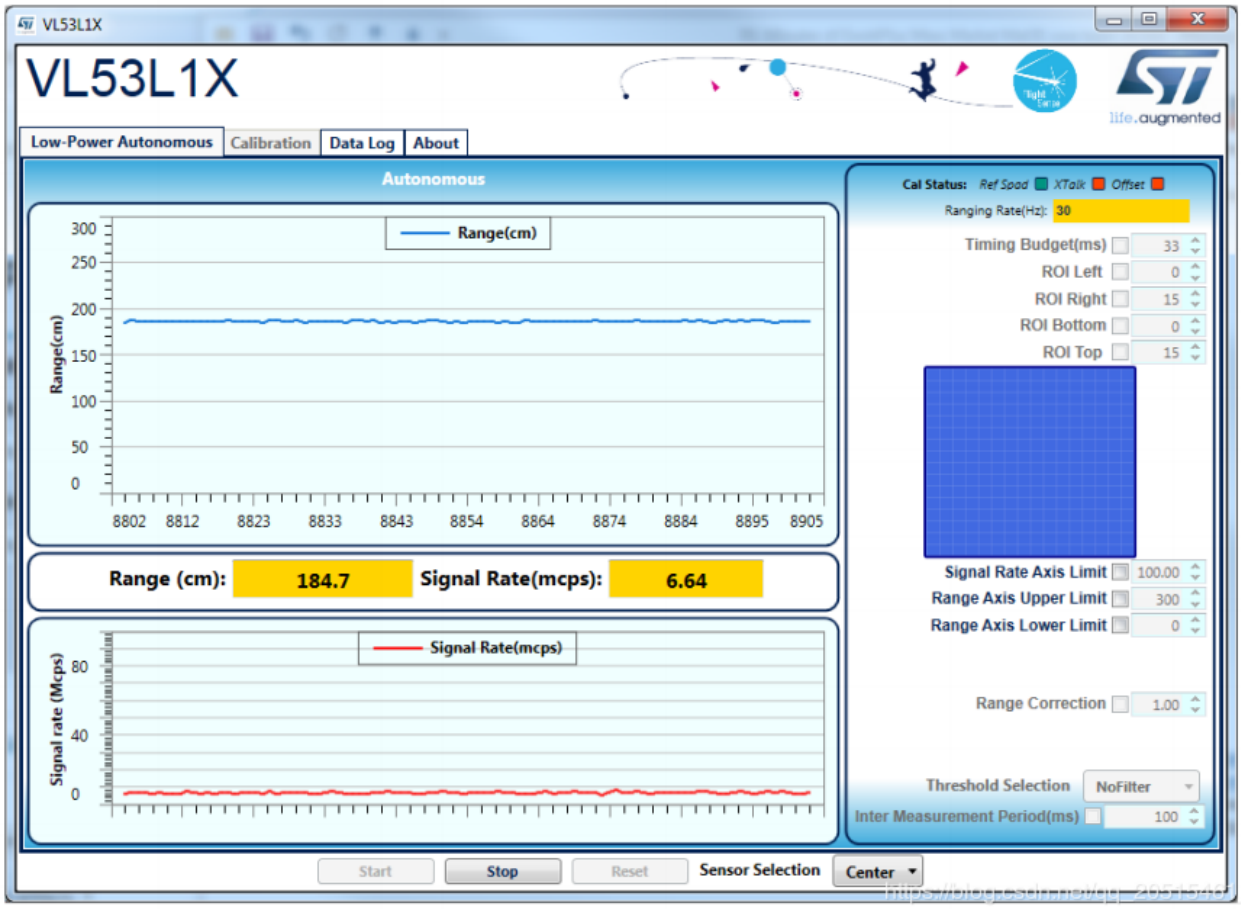


图 28: STSW-IMG008

更多信息，可参考 STSW-IMG008: Windows Graphical User Interface (GUI) for VL53L1X Nucleo packs. Works with P-NUCLEO-53L1A1。

VL53L1X 例程

例程说明

1. 实现功能：通过 VL53L1X 检测到高度变化（持续 1 秒），红灯亮起。高度恢复正常值（持续 1 秒），绿灯亮起。
2. 可配置参数：通过 `make menuconfig` 设置 I2C 号码、端口号、LED 端口号。
3. 例程解析见代码注释和用户手册。

注意事项

1. 该例程只适用于 VL53L1X，VL53L0X 为老版本硬件，不适用本例程。
2. 官方标称 400 cm 测量距离，为黑暗环境下测得。室内正常灯光环境，可以保证 10 ~ 260 cm 范围的有效测量。
3. 初始化函数 `vl53l1\_init (VL53L1\_Dev\_t \*)` 中部分参数，需要根据实际使用环境确定，还有优化的空间。
4. 传感器安装位置应确保在检测位置正上方。
5. 模块上电时自动矫正基准高度，如果基准高度有变化，需要重新上电重置参数。

例程仓库

点击 esp32-vl53l1x-test 查看例程，或使用 git 工具下载例程：

```
git clone https://github.com/qljz1993/esp32-vl53l1x-test.git
```

SPI 设备驱动 (spi_devices)

PMW3901 传感器

PMW3901 是 PixArt 公司最新的高精度低功耗光学追踪模组，可直接获取 X-Y 方向运动速度信息，实现对地高度 8 cm 以上进行有效测量。PWM3901 工作电流小于 9 mA，工作电压为 VDD (1.8 ~ 2.1 V)，VDDIO (1.8 ~ 3.6 V)，使用 4 线 SPI 接口通信。

主要参数

参数	参数值
供电电压 (V)	VDD：1.8 ~ 2.1 V；VDDIO：1.8 ~ 3.6 V
工作范围 (mm)	80 ~ +∞
接口	4 线 SPI @ 2 MHz
封装	28 管脚 COB 封装，尺寸：6 x 6 x 2.28 mm

封装和管脚图

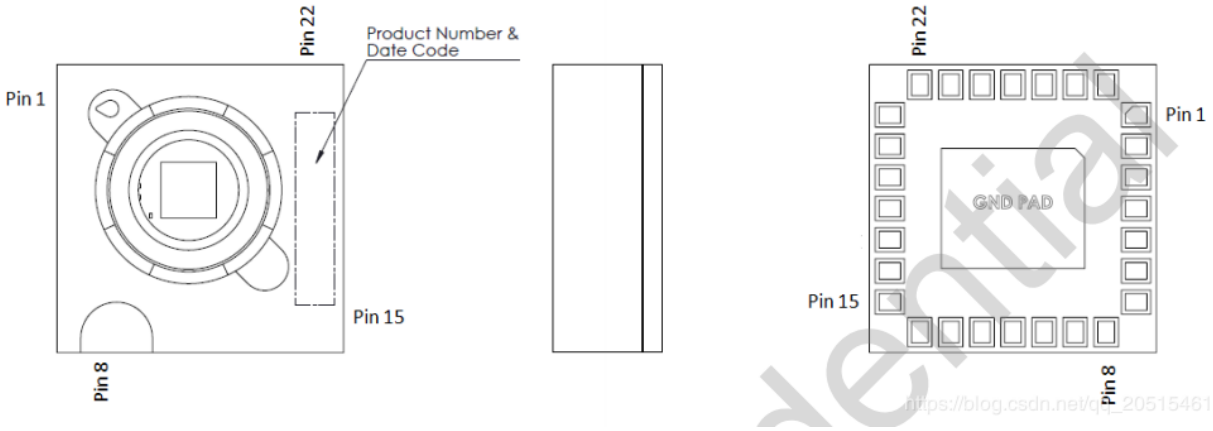


图 29: PMW3901 封装

传感器工作电压较低，与 3.3 V 的 ESP32 通信，需要 VDD 和 VDDIO 提供不同的电压。

上电启动流程

上电流程

PMW3901MB 虽然可执行内部上电自复位，但仍建议每次上电时，对 Power_Up_Reset 寄存器执行写操作。具体顺序如下:

1. 首先对 VDDIO 供电，然后对 VDD 供电，中间延迟不应超过 100 ms。注意确保供电稳定。
2. 等待至少 40 ms。
3. 先拉高，然后再拉低 NCS，以复位 SPI 口。
4. 写 0x5A 到 Power_Up_Reset 寄存器，或切换至 NRESET 管脚。
5. 等待至少 1 ms。
6. 无论运动管脚状态如何，一次读取寄存器 0x02、0x03、0x04、0x05 和 0x06。
7. 请参考 PWM3901MB Datasheet 章节 8.2 性能优化寄存器来配置所需的寄存器，以实现芯片的最佳性能。

掉电流程

Pin No.	Signal Name	Type	Description
Functional Group:		Power Supplies	
2	VDD	Power	Input power supply
3	VDDIO	Power	I/O reference voltage
4	VREG	Power	Internal voltage output
1	GND	Ground	Ground
21	GND	Ground	Ground
Functional Group:		Control Interface	
16	MOSI	Input	Serial data input
17	SCLK	Input	Serial data clock
18	MISO	Output	Serial data output
19	NCS	Input	Chip select
Functional Group:		Functional I/O	
7	NRESET	Input	Hardware reset (Active low)
15	MOTION	Output	Motion interrupt (Active low)
20	LED_N	Input	External LED control pin (Active low) (Refer **Appendix B** for more details)
Functional Group:		Special Function Pin	
5 - 6	NC	NC	No connection (float)
8 - 14	NC	NC	No connection (float)
22 - 28	NC	NC	No connection (float)
29*	GND PAD	Ground Pad	Bottom of COB package must be connected to circuit ground

图 30: PMW3901 管脚映射

通过写 Shutdown 寄存器，可将 PMW3901MB 设置为 Shutdown 模式。Shutdown 模式下 PMW3901MB 只对上电启动指令（写 0x5A 到寄存器 0x3A）进行响应，不响应其它访问操作。同一 SPI 总线下的其它设备不受 PMW3901MB Shutdown 模式的影响，在 NCS 管脚不冲突的情况下可以正常访问。

从 Shutdown 模式复位：

1. 拉高，然后再拉低 NCS，以复位 SPI 口；
2. 写 0x5A 到 Power_Up_Reset 寄存器，或切换至 NRESET 管脚；
3. 等待至少 1 ms；
4. 无论运动管脚状态如何，一次性读取寄存器 0x02、0x03、0x04、0x05 和 0x06；
5. 请参考 PWM3901MB Datasheet 章节 8.2 性能优化寄存器来配置所需的寄存器，以实现芯片的最佳性能。

更多信息见 PixArt 其它产品助力 IoT。

部分代码解读

关键结构体

```
typedef struct opFlow_s
{
    float pixSum[2]; /* 累积像素 */
    float pixComp[2]; /* 像素补偿 */
    float pixValid[2]; /* 有效像素 */
    float pixValidLast[2]; /* 上一次有效像素 */
    float deltaPos[2]; /*2 帧之间的位移 单位: cm*/
    float deltaVel[2]; /* 速度 单位 cm/s*/
    float posSum[2]; /* 累积位移 单位 cm*/
    float velLpf[2]; /* 速度低通 单位 cm/s*/
    bool isOpFlowOk; /* 光流状态 */
    bool isDataValid; /* 数据有效 */
} opFlow_t;
```

- 累积像素：飞行器起飞后的累积像素；
- 像素补偿：补偿由于飞行器倾斜导致的像素误差；
- 有效像素：指经过补偿的实际像素；
- 2 帧之间的位移：即由像素转换出来的实际位移，单位 cm；
- 速度：表示瞬时速度，由位移变化量微分得到，单位 cm/s；
- 累积位移：实际位移，单位 cm；
- 速度低通：对速度进行低通，增加数据平滑性；
- 光流状态：检查光流是否正常工作；
- 数据有效：在一定高度范围内，数据有效。

```
typedef struct motionBurst_s {
  union {
    uint8_t motion;
    struct {
      uint8_t frameFrom0    : 1;
      uint8_t runMode       : 2;
      uint8_t reserved1     : 1;
      uint8_t rawFrom0      : 1;
      uint8_t reserved2     : 2;
      uint8_t motionOccured : 1;
    };
  };
```

(下页继续)

(续上页)

```
  uint8_t observation;
  int16_t deltaX;
  int16_t deltaY;

  uint8_t squal;

  uint8_t rawDataSum;
  uint8_t maxRawData;
  uint8_t minRawData;

  uint16_t shutter;
} __attribute__((packed)) motionBurst_t;
```

- motion：运动信息，可以根据不同的位去判断运动信息，包括帧判别、运行模式和运动信息检测等；
- observation：用于检测 IC 是否出现 EFT/B 或 ESD 问题。传感器正常工作时，读取出来的值为 0xBF；
- deltaX 和 deltaY：光流检测到图像 X 和 Y 方向的运动信息；
- squal：指运动信息质量，即运动信息的可信度；
- rawDataSum：原数据求和，可用于对一帧数据求平均值；
- maxRawData 和 minRawData：最大和最小原始数据；
- shutter：是一个实时自动调整的值，确保平均运动数据保持在正常可操作范围以内。shutter 可搭配 squal，用来判断运动信息是否可用。

编程注意事项

- 如果连续 1 s 内光流数据都为 0，说明出现故障，需要做挂起光流任务等处理；
- 注意，传感器镜头必须朝下安装。由于地方位置固定，根据相对运动，**传感器采集的位移数据与飞机实际运动方向相反**；
- 注意，只有在定高模式测试稳定时，才能进入定点模式。精确的高度信息，用于确定图像像素和实际距离的对应关系；
- 手动测试倾角补偿，实现通过补偿使飞行器有一定的倾角时，传感器输出基本不变化；
- 有了倾角补偿和运动累积像素即可以得到实际累积像素。通过相关计算可以得到：
  - 2 帧之间变化像素 = 实际积累像素 - 上次实际像素；
  - 2 帧之间的位移变化 = 2 帧之间变化像素 x 系数。对系数的限制：高度小于 5 cm，光流即无法工作，所以系数设置为 0；
  - 对上述位移积分，得到四轴到起飞点的位移；对上述位移微分，得到瞬时速度。注意对速度进行低通增加数据的平滑性，对速度进行限幅处理，增加数据安全性。

- 通过光流就得到了四轴的位置信息和速度信息，然后：
  - 上述位置信息和速度信息融合加速计 (state_estimator.c)，即可得到估测位置和速度；
  - 估测位置和速度参与 PID 运算，即可用于水平方向位置控制。请参考 position_pid.c，查看位置环和速度环 PID 的处理过程。

通过上述过程即可实现水平定点控制。

3.3.3 飞控系统

[English]

系统启动流程

源文件见：start_from_app_main。

系统任务管理

系统任务简介

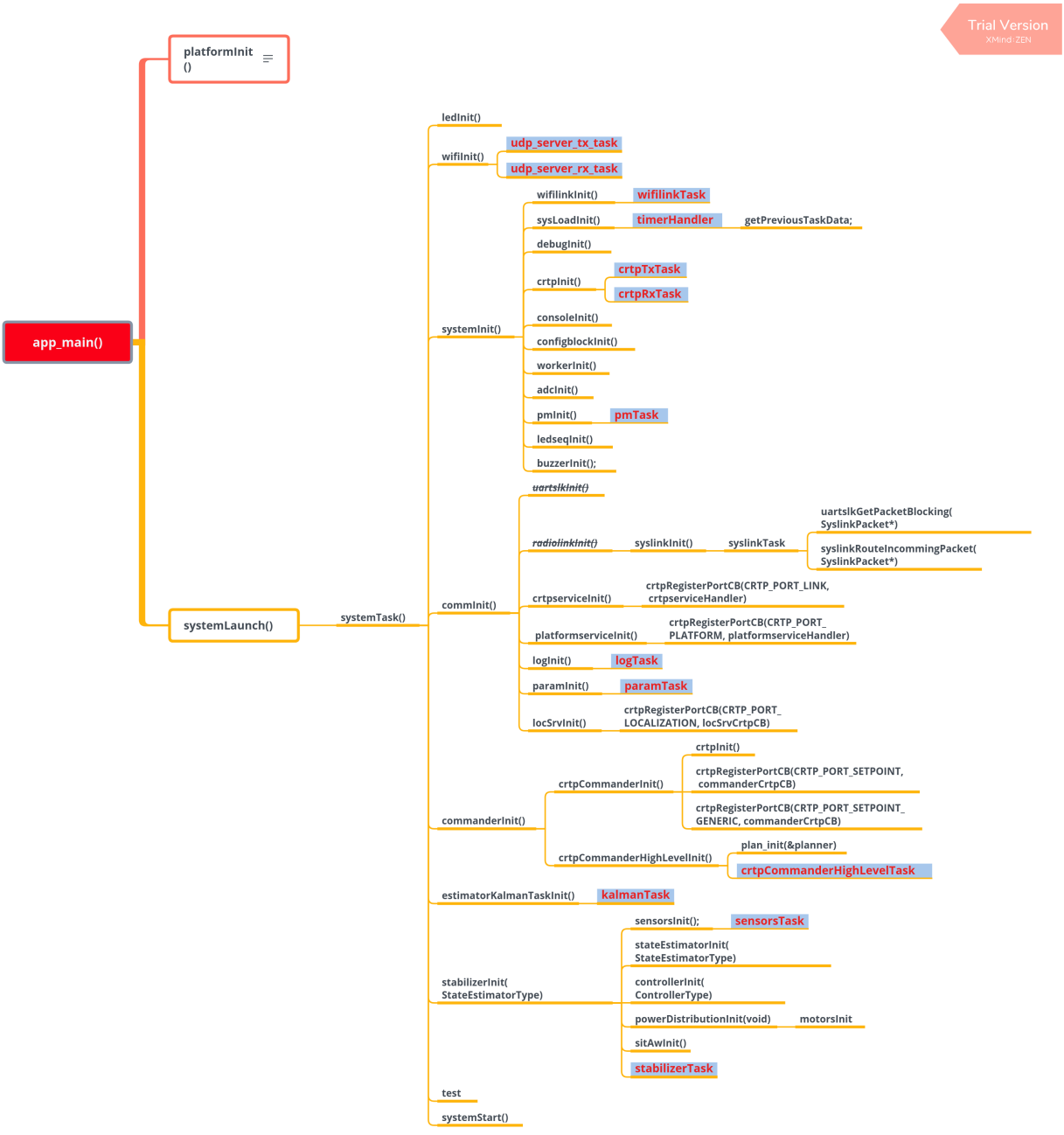
系统正常运行时，将启动以下 TASK。

其中，

- Load：CPU 占用率
- Stack Left：剩余堆栈空间
- Name：TASK 名称
- PRI：TASK 优先级

TASK 具体描述如下：

- PWRMGNT：系统电压监测
- CMDHL：应用层-处理根据 CRTP 协议构成的高级命令
- CRTP-RX：协议层-CRTP 飞行协议解码
- CRTP-TX：协议层-CRTP 飞行协议解码
- UDP-RX：传输层-UDP 包接收
- UDP-TX：传输层-UDP 包发送
- WIFILINK：对接 CRTP 协议层和 UDP 传输层
- SENSORS：传感器数据读取和预处理


Trial Version
XMind:ZEN
app_main()
platformInit()
systemLaunch()
systemTask()
ledInit()
wifiInit()
udp_server_tx_task
udp_server_rx_task
systemInit()
wifilinkInit()
wifilinkTask
sysLoadInit()
timerHandler
getPreviousTaskData;
debugInit()
crtpInit()
crtpTxTask
crtpRxTask
consoleInit()
configblockInit()
workerInit()
adcInit()
pmInit()
pmTask
ledseqInit()
buzzerInit();
commInit()
uartslkinit()
radiolinkInit()
syslinkInit()
syslinkTask
uartslkGetPacketBlocking(SyslinkPacket*)
syslinkRouteIncommingPacket(SyslinkPacket*)
crtpserviceInit()
crtpRegisterPortCB(CRTP_PORT_LINK, crtpserviceHandler)
platformserviceInit()
crtpRegisterPortCB(CRTP_PORT_PLATFORM, platformserviceHandler)
logInit()
logTask
paramInit()
paramTask
locSrvInit()
crtpRegisterPortCB(CRTP_PORT_LOCALIZATION, locSrvCrtpCB)
commanderInit()
crtpCommanderInit()
crtpInit()
crtpRegisterPortCB(CRTP_PORT_SETPOINT, commanderCrtpCB)
crtpRegisterPortCB(CRTP_PORT_SETPOINT_GENERIC, commanderCrtpCB)
crtpCommanderHighLevelInit()
plan_init(&planner)
crtpCommanderHighLevelTask
estimatorKalmanTaskInit()
kalmanTask
stabilizerInit(StateEstimatorType)
sensorsInit();
sensorsTask
stateEstimatorInit(StateEstimatorType)
controllerInit(ControllerType)
powerDistributionInit(void)
motorsInit
sitAwInit()
stabilizerTask
test
systemStart()

```
SYSLOAD: Task dump
SYSLOAD: Load      Stack left   Name       PRI
SYSLOAD: 0.40      2192         Tmr Svc    1
SYSLOAD: 18.70     1608         KALMAN     1
SYSLOAD: 51.13     1212         IDLE0      0
SYSLOAD: 0.04      1512         PWRMGNT    0
SYSLOAD: 0.89      1564         CRTP-RX    2
SYSLOAD: 2.14      2520         tiT        18
SYSLOAD: 10.36     1420         SENSORS    4
SYSLOAD: 0.68      1428         UDP_TX     3
SYSLOAD: 0.0       592          CMDHL      2
SYSLOAD: 0.0       1152         sys_evt    20
SYSLOAD: 0.0       1632         PARAM      2
SYSLOAD: 0.14      1904         SYSTEM     2
SYSLOAD: 0.08      1656         CRTP-TX    2
SYSLOAD: 0.08      3560         esp_timer  22
SYSLOAD: 3.54      4528         wifi       23
SYSLOAD: 0.13      632          WIFILINK   3
SYSLOAD: 1.43      1232         UDP_RX     3
SYSLOAD: 10.17     1492         STABILIZER          5
SYSLOAD: 0.0       668          MEM        2
SYSLOAD: 0.0       1684         LOG        2
```

- KALMAN：使用传感器数据进行飞机状态估计，包括飞机角度、角速度、空间位置的估计。该 TASK 在 ESP 芯片上 CPU 资源消耗较大，应注意优先级的分配。
- PARAM：使用 CRTP 协议远程修改变量
- LOG：使用 CRTP 协议实时监控变量
- MEM：使用 CRTP 协议远程修改存储器
- STABILIZER：自稳定线程，控制飞控程序运行流程
- SYSTEM：控制系统初始化和自检流程

任务堆栈空间配置

用户可以在 `components/config/include/config.h` 中直接修改空间大小，也可以在 `menucfg` 中修改 `BASE_STACK_SIZE` 大小。使用 `ESP32` 时可将 `BASE_STACK_SIZE` 调整为 2048，减小踩内存的概率；使用 `ESP32-S2` 时，建议将该值调整为 `1024`。

```
//任务堆栈空间大小
#define SYSTEM_TASK_STACKSIZE         (4* configBASE_STACK_SIZE)
#define ADC_TASK_STACKSIZE            configBASE_STACK_SIZE
#define PM_TASK_STACKSIZE             (2*configBASE_STACK_SIZE)
#define CRTP_TX_TASK_STACKSIZE        (2*configBASE_STACK_SIZE)
#define CRTP_RX_TASK_STACKSIZE        (2* configBASE_STACK_SIZE)
#define CRTP_RXTX_TASK_STACKSIZE      configBASE_STACK_SIZE
```

(下页继续)

(续上页)

```
#define LOG_TASK_STACKSIZE            (2*configBASE_STACK_SIZE)
#define MEM_TASK_STACKSIZE            (1 * configBASE_STACK_SIZE)
#define PARAM_TASK_STACKSIZE          (2*configBASE_STACK_SIZE)
#define SENSORS_TASK_STACKSIZE        (2 * configBASE_STACK_SIZE)
#define STABILIZER_TASK_STACKSIZE     (2 * configBASE_STACK_SIZE)
#define NRF24LINK_TASK_STACKSIZE      configBASE_STACK_SIZE
#define ESKYLINK_TASK_STACKSIZE       configBASE_STACK_SIZE
#define SYSLINK_TASK_STACKSIZE        configBASE_STACK_SIZE
#define USBLINK_TASK_STACKSIZE        configBASE_STACK_SIZE
#define WIFILINK_TASK_STACKSIZE        (2*configBASE_STACK_SIZE)
#define UDP_TX_TASK_STACKSIZE   (2*configBASE_STACK_SIZE)
#define UDP_RX_TASK_STACKSIZE   (2*configBASE_STACK_SIZE)
#define UDP_RX2_TASK_STACKSIZE   (1*configBASE_STACK_SIZE)
#define PROXIMITY_TASK_STACKSIZE      configBASE_STACK_SIZE
#define EXTRX_TASK_STACKSIZE          configBASE_STACK_SIZE
#define UART_RX_TASK_STACKSIZE        configBASE_STACK_SIZE
#define ZRANGER_TASK_STACKSIZE        (1* configBASE_STACK_SIZE)
#define ZRANGER2_TASK_STACKSIZE       (2* configBASE_STACK_SIZE)
#define FLOW_TASK_STACKSIZE           (2* configBASE_STACK_SIZE)
#define USDLOG_TASK_STACKSIZE         (1* configBASE_STACK_SIZE)
#define USDWRITE_TASK_STACKSIZE       (1* configBASE_STACK_SIZE)
#define PCA9685_TASK_STACKSIZE        (1* configBASE_STACK_SIZE)
#define CMD_HIGH_LEVEL_TASK_STACKSIZE (1* configBASE_STACK_SIZE)
#define MULTIRANGER_TASK_STACKSIZE    (1* configBASE_STACK_SIZE)
#define ACTIVEMARKER_TASK_STACKSIZE   configBASE_STACK_SIZE
#define AI_DECK_TASK_STACKSIZE        configBASE_STACK_SIZE
#define UART2_TASK_STACKSIZE          configBASE_STACK_SIZE
```

任务优先级配置

系统 TASK 优先级可以在 components/config/include/config.h 中进行配置。由于 ESP32 具有双核优势，相比 ESP32-S2 计算资源更加富余，可将高耗时的 KALMAN_TASK 优先级调高。在使用 ESP32-S2 时，需要将高耗时的 KALMAN_TASK 优先级调低，否则难以释放足够的 CPU 资源，将触发 task watchdog。

```
// 任务优先级，数字越大，优先级越高。
#define STABILIZER_TASK_PRI     5
#define SENSORS_TASK_PRI        4
#define ADC_TASK_PRI            3
#define FLOW_TASK_PRI           3
#define MULTIRANGER_TASK_PRI    3
#define SYSTEM_TASK_PRI         2
#define CRTP_TX_TASK_PRI        2
```

(下页继续)

(续上页)

```
#define CRTP_RX_TASK_PRI        2
#define EXTRX_TASK_PRI          2
#define ZRANGER_TASK_PRI        2
#define ZRANGER2_TASK_PRI       2
#define PROXIMITY_TASK_PRI      0
#define PM_TASK_PRI             0
#define USDLOG_TASK_PRI         1
#define USDWRITE_TASK_PRI       0
#define PCA9685_TASK_PRI        2
#define CMD_HIGH_LEVEL_TASK_PRI 2
#define BQ_OSD_TASK_PRI         1
#define GTGPS_DECK_TASK_PRI     1
#define LIGHTHOUSE_TASK_PRI     3
#define LPS_DECK_TASK_PRI       5
#define OA_DECK_TASK_PRI        3
#define UART1_TEST_TASK_PRI     1
#define UART2_TEST_TASK_PRI     1
//if task watchdog triggered, KALMAN_TASK_PRI should set lower or set lower flow␣
↪frequency
#ifdef CONFIG_IDF_TARGET_ESP32
  #define KALMAN_TASK_PRI         2
  #define LOG_TASK_PRI            1
  #define MEM_TASK_PRI            1
  #define PARAM_TASK_PRI          1
#else
  #define KALMAN_TASK_PRI         1
  #define LOG_TASK_PRI            2
  #define MEM_TASK_PRI            2
  #define PARAM_TASK_PRI          2
#endif

#define SYSLINK_TASK_PRI        3
#define USBLINK_TASK_PRI        3
#define ACTIVE_MARKER_TASK_PRI  3
#define AI_DECK_TASK_PRI        3
#define UART2_TASK_PRI          3
#define WIFILINK_TASK_PRI       3
#define UDP_TX_TASK_PRI         3
#define UDP_RX_TASK_PRI         3
#define UDP_RX2_TASK_PRI        3
```

关键任务介绍

除了系统默认开启的 TASK（如 Wi-Fi TASK），优先级最高的 TASK 是 `STABILIZER_TASK`，凸显了这个任务的重要性。`STABILIZER_TASK` 控制了从传感器数据读取，到姿态计算，到目标接收，到最终输出电机功率的整个过程，驱动各个阶段的算法运行。

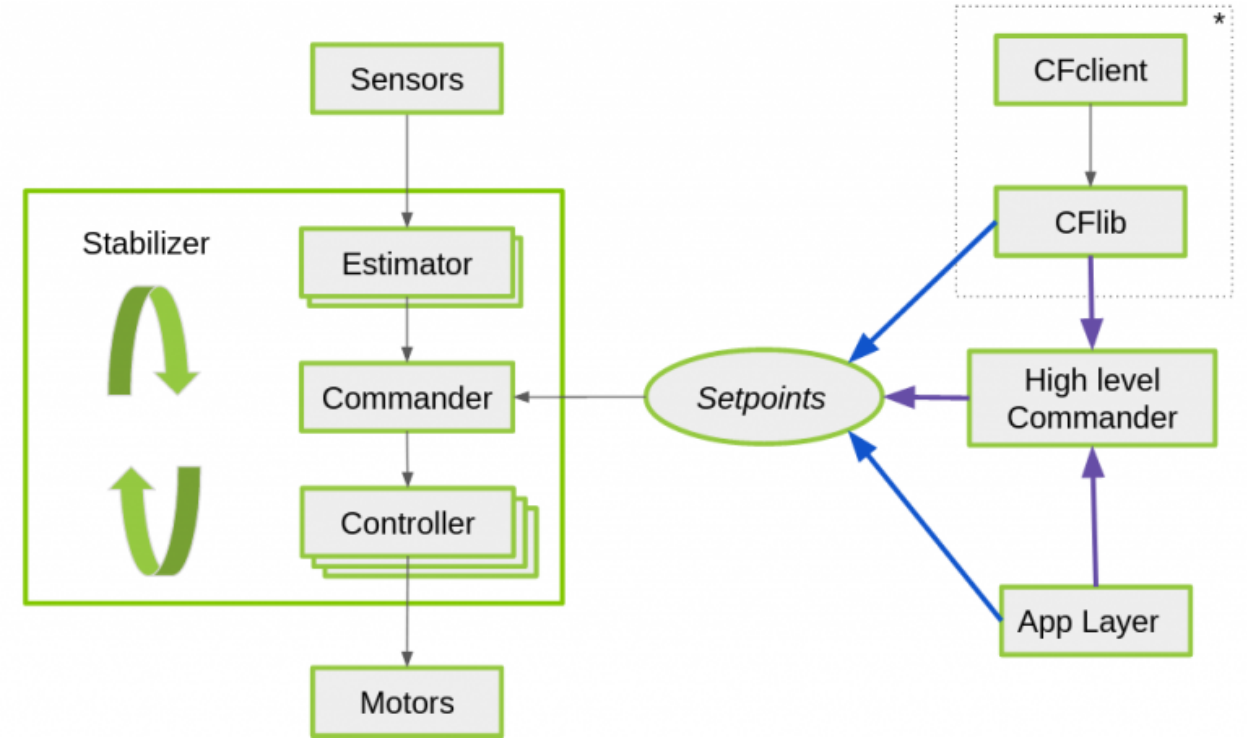


图 31: stabilizerTask 流程

传感器驱动

传感器驱动代码，可以在 `components\drivers` 中查阅。`drivers` 使用了与 esp-iot-solution 类似的文件结构，将驱动程序按照所属总线进行分类，包括 `i2c_devices`、`spi_devices`、`general` 等。具体可参考：驱动程序。

传感器硬件抽象

`components\core\crazyflie\hal\src\sensors.c` 文件对传感器进行了硬件抽象，开发者可以自由组合传感器，通过实现硬件抽象层定义的传感器接口，与上层应用进行对接。

```
typedef struct {
  SensorImplementation_t implements;
  void (*init)(void);
  bool (*test)(void);
```

(下页继续)

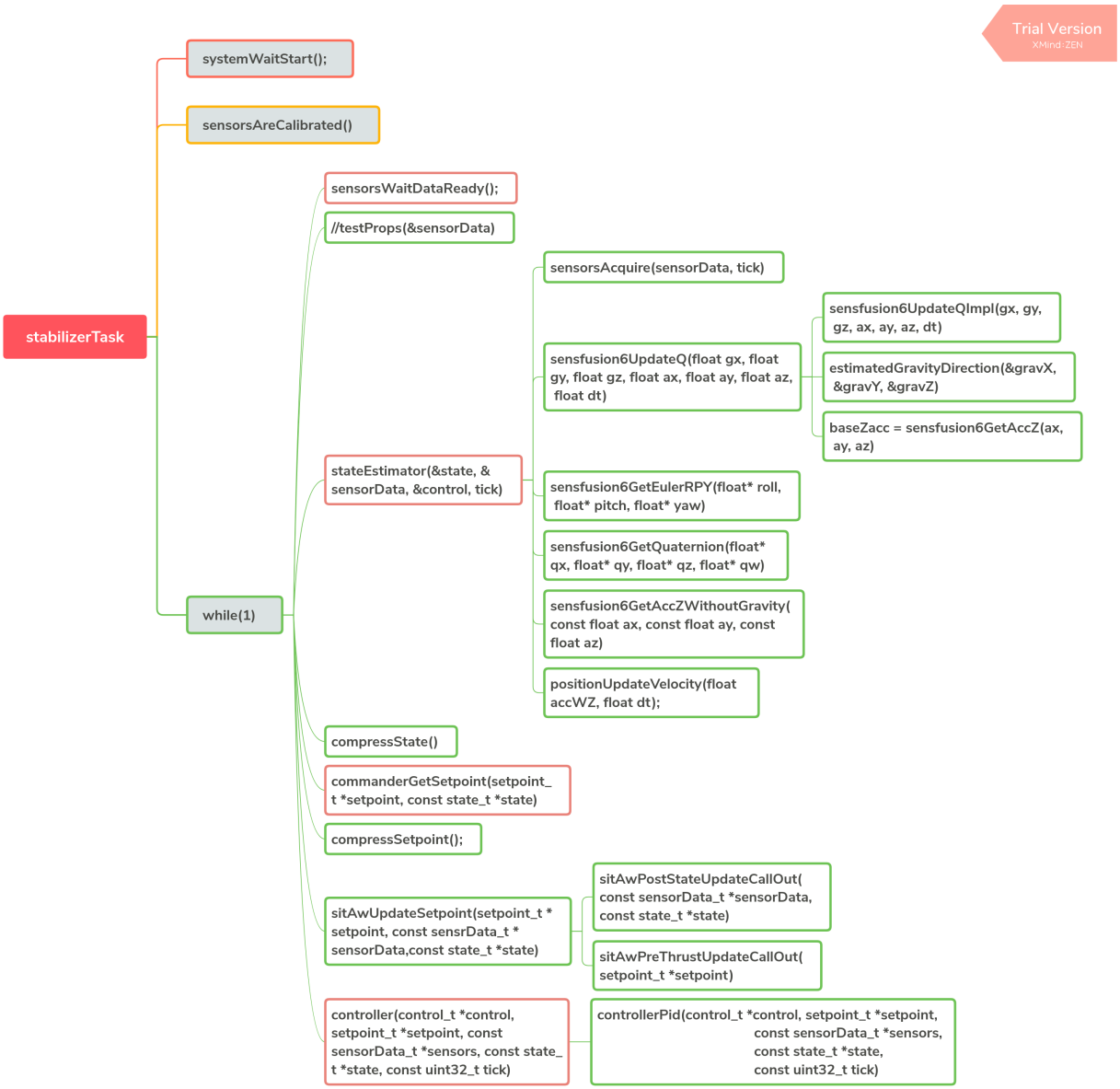


图 32: stabilizerTask

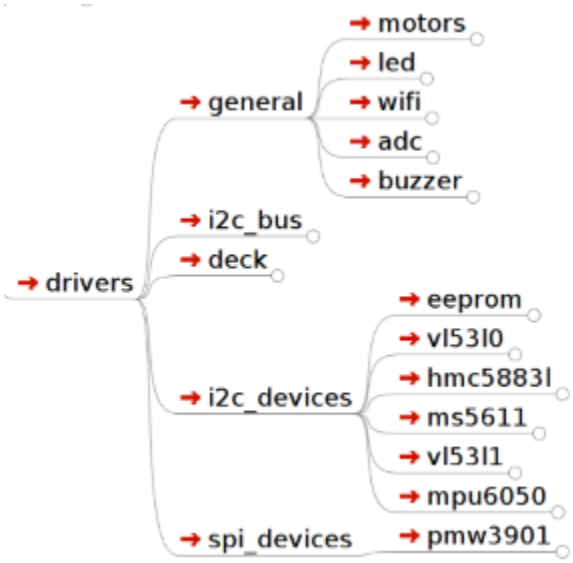


图 33: 驱动文件结构

(续上页)

```
  bool (*areCalibrated)(void);
  bool (*manufacturingTest)(void);
  void (*acquire)(sensorData_t *sensors, const uint32_t tick);
  void (*waitDataReady)(void);
  bool (*readGyro)(Axis3f *gyro);
  bool (*readAcc)(Axis3f *acc);
  bool (*readMag)(Axis3f *mag);
  bool (*readBaro)(baro_t *baro);
  void (*setAccMode)(accModes accMode);
  void (*dataAvailableCallback)(void);
} sensorsImplementation_t;
```

ESP-Drone 实现的传感器抽象接口在 `components/core/crazyflie/hal/src/sensors_mpu6050_hm5883L_ms5611.c` 中，通过以下赋值过程与上层应用对接：

```
#ifdef SENSOR_INCLUDED_MPU6050_HMC5883L_MS5611
  {
    .implements = SensorImplementation_mpu6050_HMC5883L_MS5611,
    .init = sensorsMpu6050Hmc5883lMs5611Init,
    .test = sensorsMpu6050Hmc5883lMs5611Test,
    .areCalibrated = sensorsMpu6050Hmc5883lMs5611AreCalibrated,
    .manufacturingTest = sensorsMpu6050Hmc5883lMs5611ManufacturingTest,
    .acquire = sensorsMpu6050Hmc5883lMs5611Acquire,
    .waitDataReady = sensorsMpu6050Hmc5883lMs5611WaitDataReady,
    .readGyro = sensorsMpu6050Hmc5883lMs5611ReadGyro,
```

(下页继续)

(续上页)

```
    .readAcc = sensorsMpu6050Hmc5883lMs5611ReadAcc,
    .readMag = sensorsMpu6050Hmc5883lMs5611ReadMag,
    .readBaro = sensorsMpu6050Hmc5883lMs5611ReadBaro,
    .setAccMode = sensorsMpu6050Hmc5883lMs5611SetAccMode,
    .dataAvailableCallback = nullFunction,
  }
#endif
```

传感器校准过程

陀螺仪校准过程

由于陀螺仪存在较大的温漂，因此每次使用前需要对陀螺仪进行校准，计算当前环境下的陀螺仪基准值。ESP-Drone 延续 Crazyflie 2.0 陀螺仪校准方案，在初次上电时，计算陀螺仪三个轴的方差与平均值。

陀螺仪具体校准过程如下：

1. 使用一个最大长度为 1024 的环形缓冲区，存储最新的 1024 组陀螺仪测量值。
2. 通过计算陀螺仪输出值方差，确认飞行器已经放置平稳并且陀螺仪工作正常。
3. 确认第 2 步正常后，计算静止时 1024 组陀螺仪输出值的平均值，作为陀螺仪的校准值。

陀螺仪基准值计算源代码：

```
/**
 * Adds a new value to the variance buffer and if it is full
 * replaces the oldest one. Thus a circular buffer.
 */
static void sensorsAddBiasValue(BiasObj* bias, int16_t x, int16_t y, int16_t z)
{
  bias->bufHead->x = x;
  bias->bufHead->y = y;
  bias->bufHead->z = z;
  bias->bufHead++;

  if (bias->bufHead >= &bias->buffer[SENSORS_NBR_OF_BIAS_SAMPLES])
  {
    bias->bufHead = bias->buffer;
    bias->isBufferFilled = true;
  }
}

/**
 * Checks if the variances is below the predefined thresholds.
```

(下页继续)

(续上页)

```
 * The bias value should have been added before calling this.
 * @param bias  The bias object
 */
static bool sensorsFindBiasValue(BiasObj* bias)
{
  static int32_t varianceSampleTime;
  bool foundBias = false;

  if (bias->isBufferFilled)
  {
    sensorsCalculateVarianceAndMean(bias, &bias->variance, &bias->mean);

    if (bias->variance.x < GYRO_VARIANCE_THRESHOLD_X &&
        bias->variance.y < GYRO_VARIANCE_THRESHOLD_Y &&
        bias->variance.z < GYRO_VARIANCE_THRESHOLD_Z &&
        (varianceSampleTime + GYRO_MIN_BIAS_TIMEOUT_MS < xTaskGetTickCount()))
    {
      varianceSampleTime = xTaskGetTickCount();
      bias->bias.x = bias->mean.x;
      bias->bias.y = bias->mean.y;
      bias->bias.z = bias->mean.z;
      foundBias = true;
      bias->isBiasValueFound = true;
    }
  }

  return foundBias;
}
```

修正陀螺仪输出值：

```
sensorData.gyro.x = (gyroRaw.x - gyroBias.x) * SENSORS_DEG_PER_LSB_CFG;
sensorData.gyro.y = (gyroRaw.y - gyroBias.y) * SENSORS_DEG_PER_LSB_CFG;
sensorData.gyro.z = (gyroRaw.z - gyroBias.z) * SENSORS_DEG_PER_LSB_CFG;
applyAxis3fLpf((lpf2pData *)(&gyroLpf), &sensorData.gyro); //低通滤波器，去除高频干扰
```

加速度计校准过程

重力加速度校准

在地球不同的纬度和海拔下，重力加速度 g 值一般不同，因此需要使用加速度计对 g 进行实际测量。可参考 Crazyflie 2.0 加速度计校准方案，g 值的校准过程如下：

1. 陀螺仪校准完成后，立刻进行加速度计校准。
2. 使用 Buffer 保存 200 组加速度计测量值。
3. 通过合成重力加速度在三个轴的分量，计算重力加速度在静止状态下的值。

参考：不同地球纬度和海拔下的重力加速度值 g。

计算静止状态下重力加速度值：

```
/**
 * Calculates accelerometer scale out of SENSORS_ACC_SCALE_SAMPLES samples. Should be␣
↪called when
 * platform is stable.
 */
static bool processAccScale(int16_t ax, int16_t ay, int16_t az)
{
    static bool accBiasFound = false;
    static uint32_t accScaleSumCount = 0;

    if (!accBiasFound)
    {
        accScaleSum += sqrtf(powf(ax * SENSORS_G_PER_LSB_CFG, 2) + powf(ay * SENSORS_
↪G_PER_LSB_CFG, 2) + powf(az * SENSORS_G_PER_LSB_CFG, 2));
        accScaleSumCount++;

        if (accScaleSumCount == SENSORS_ACC_SCALE_SAMPLES)
        {
            accScale = accScaleSum / SENSORS_ACC_SCALE_SAMPLES;
            accBiasFound = true;
        }
    }

    return accBiasFound;
}
```

通过实际重力加速度值，修正加速度计测量值：

```
accScaled.x = (accelRaw.x) * SENSORS_G_PER_LSB_CFG / accScale;
accScaled.y = (accelRaw.y) * SENSORS_G_PER_LSB_CFG / accScale;
```

(下页继续)

(续上页)

```
accScaled.z = (accelRaw.z) * SENSORS_G_PER_LSB_CFG / accScale;
```

机身水平校准

理想状态下，加速度传感器在小飞机上完全水平地进行安装，进而可以使用 0 位置作为小飞机的水平面。但由于加速度计在安装时不可避免的存在一定的倾角，导致飞控系统错误估计水平位置，导致小飞机向某个方向偏飞。因此需要设置一定的校准策略来平衡这种误差。

1. 将小飞机放置在一个水平面上，计算小飞机 `cosRoll`、`sinRoll`、`cosPitch`、`sinPitch`。理想状态下 `cosRoll`、`cosPitch` 为 1，`sinPitch`、`sinRoll` 为 0。如果不是水平安装 `sinPitch`、`sinRoll` 不为 0，`cosRoll cosPitch` 不为 1。
2. 将步骤 1 的 `cosRoll`、`sinRoll`、`cosPitch`、`sinPitch` 或对应的 `Roll`、`Pitch` 角度值保存到飞机，用于校准。

利用校准值，对加速度计测量值进行修正：

```
/**
 * Compensate for a miss-aligned accelerometer. It uses the trim
 * data gathered from the UI and written in the config-block to
 * rotate the accelerometer to be aligned with gravity.
 */
static void sensorsAccAlignToGravity(Axis3f *in, Axis3f *out)
{
    //TODO: need cosPitch calculate firstly
    Axis3f rx;
    Axis3f ry;

    // Rotate around x-axis
    rx.x = in->x;
    rx.y = in->y * cosRoll - in->z * sinRoll;
    rx.z = in->y * sinRoll + in->z * cosRoll;

    // Rotate around y-axis
    ry.x = rx.x * cosPitch - rx.z * sinPitch;
    ry.y = rx.y;
    ry.z = -rx.x * sinPitch + rx.z * cosPitch;

    out->x = ry.x;
    out->y = ry.y;
    out->z = ry.z;
}
```

以上过程，可通过力的分解和勾股定理推导。

姿态计算

支持的姿态计算算法

- 互补滤波
- 卡尔曼滤波

ESP-Drone 姿态计算代码来自 `Crazyflie`。ESP-Drone 固件已经对互补滤波和卡尔曼滤波进行了实际测试，可以有效地计算飞行姿态，包括各个自由度的角度、角速度、和空间位置，为控制系统提供了可靠的状态输入。需要注意的是，在定点模式下，必须切换到卡尔曼滤波算法，才能保证工作正常。

Crazyflie 状态估计见 State estimation: To be or not to be!

互补滤波

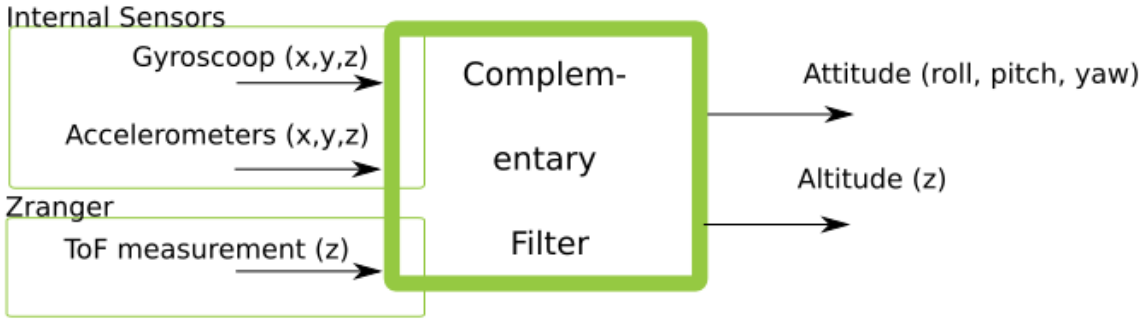


图 34: 互补滤波

互补滤波中文说明可参考 飞控与姿态互补滤波器。

卡尔曼滤波

卡尔曼滤波中文说明可参考 图说卡尔曼滤波，一份通俗易懂的教程。

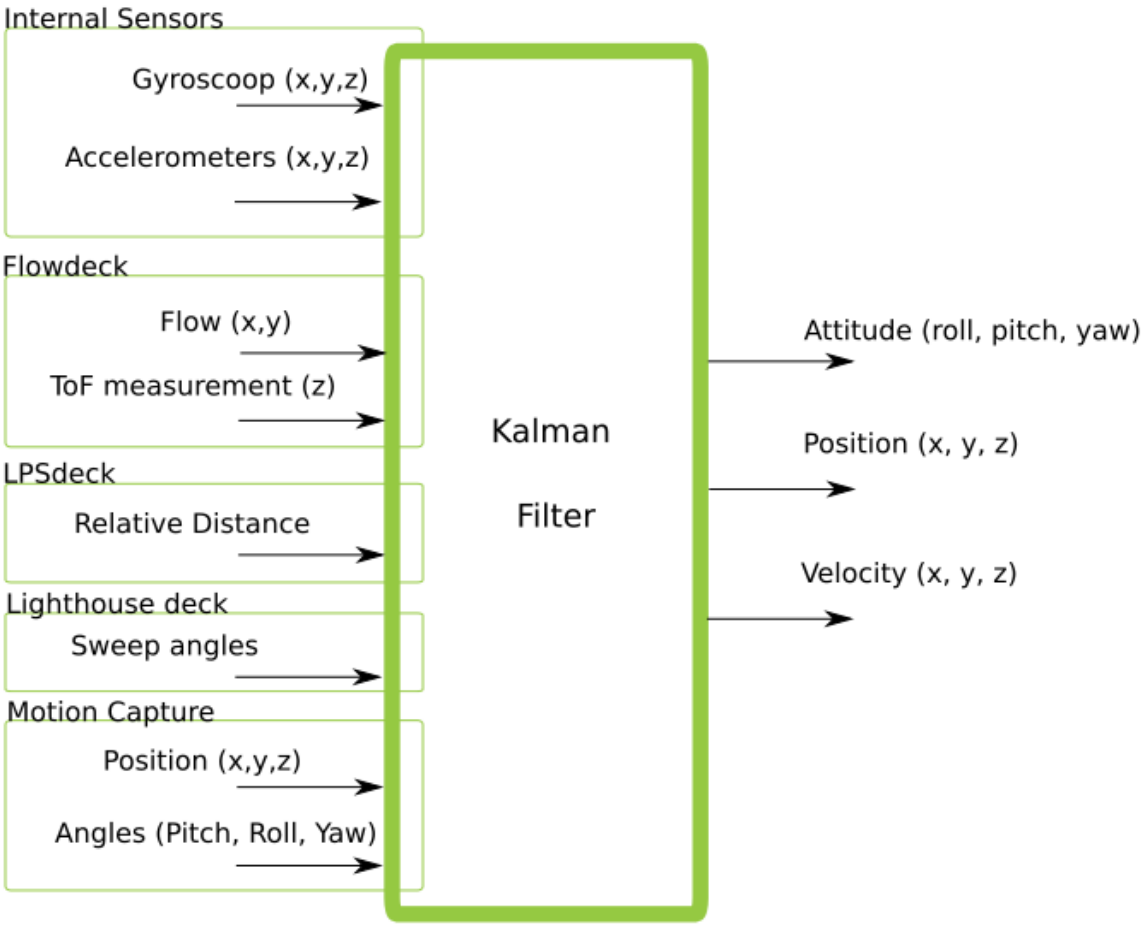

Internal Sensors
Gyroscoop (x,y,z)
Accelerometers (x,y,z)
Flowdeck
Flow (x,y)
ToF measurement (z)
LPSdeck
Relative Distance
Lighthouse deck
Sweep angles
Motion Capture
Position (x,y,z)
Angles (Pitch, Roll, Yaw)
Kalman
Filter
Attitude (roll, pitch, yaw)
Position (x, y, z)
Velocity (x, y, z)

图 35: 卡尔曼滤波

控制算法

已支持的控制器

ESP-Drone 控制系统代码来自 `Crazyflie`，也继承了该工程的所有控制算法。需要注意的是，ESP-Drone 仅对 PID 控制器进行了参数整定和测试。换用其它控制器时，请在确保安全的情况下，自行进行参数整定。

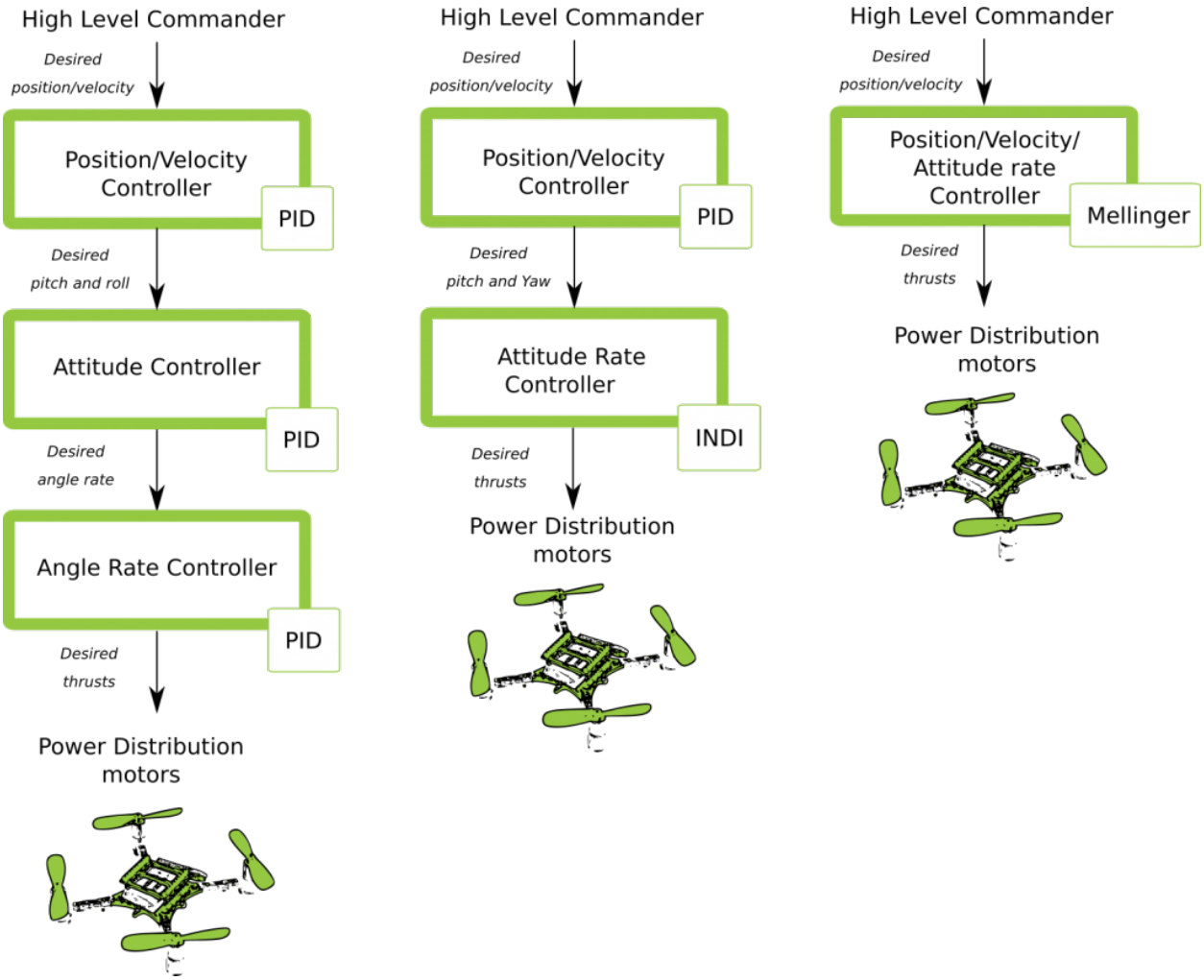


图 36: possible_controller_pathways

详情请参考：Out of Control。

在代码中，可通过修改 `controllerInit(ControllerType controller)` 的传入参数，切换控制器。

也可通过实现以下控制器接口，添加自定义的控制器：

```
static ControllerFcns controllerFunctions[] = {
  {.init = 0, .test = 0, .update = 0, .name = "None"}, // Any
  {.init = controllerPidInit, .test = controllerPidTest, .update = controllerPid, .
↪name = "PID"},
```

(下页继续)

(续上页)

```
  {.init = controllerMellingerInit, .test = controllerMellingerTest, .update =␣
↪controllerMellinger, .name = "Mellinger"},
  {.init = controllerINDIInit, .test = controllerINDITest, .update = controllerINDI, .
↪name = "INDI"},
};
```

PID 控制器

控制原理

PID 控制器（比例-积分-微分控制器），由比例单元 (Proportional)、积分单元 (Integral) 和微分单元 (Derivative) 组成，分别对应当前误差、过去累计误差及未来误差，最终基于误差和误差的变化率对系统进行控制。PID 控制器由于具有负反馈修正作用，一般被认为是最适用的控制器。通过调整 PID 控制器的三类参数，可以调整系统对误差的反应快慢、控制器过冲的程度及系统震荡的程度，使系统达到最优状态。

在飞行器系统中，由于存在 pitch、roll、yaw 三个自由度，因此需要设计如下图所示的具有控制闭环的 PID 控制器。

其中每一个自由度都包括一个串级 PID 控制器：Rate 控制和 Attitude 控制，前者以角速度作为输入量，控制角度修正的速度；后者以拟合后的角度为输入量，控制飞机到达目标角度，两个控制器以不同的频率配合工作。当然，也可以选择只使用单级的 PID 控制，默认情况下 pitch 和 roll 自由度使用 Attitude 控制，yaw 使用 Rate 控制。

```
可以在 crtp_commander_rpyt.c 中调整如下参数选择
static RPYType stabilizationModeRoll  = ANGLE; // Current stabilization type of roll␣
↪(rate or angle)
static RPYType stabilizationModePitch = ANGLE; // Current stabilization type of pitch␣
↪(rate or angle)
static RPYType stabilizationModeYaw   = RATE;  // Current stabilization type of yaw␣
↪(rate or angle)
```

实现代码

```
void controllerPid(control_t *control, setpoint_t *setpoint,
                                       const sensorData_t *sensors,
                                       const state_t *state,
                                       const uint32_t tick)
{
  if (RATE_DO_EXECUTE(ATTITUDE_RATE, tick)) { //该宏定义用于控制 PID 的计算频率，时间基准来
自 MPU6050 触发的中断
    // Rate-controled YAW is moving YAW angle setpoint
    if (setpoint->mode.yaw == modeVelocity) {                                    ␣
↪           //rate 模式，对 yaw 做修正
```

(下页继续)

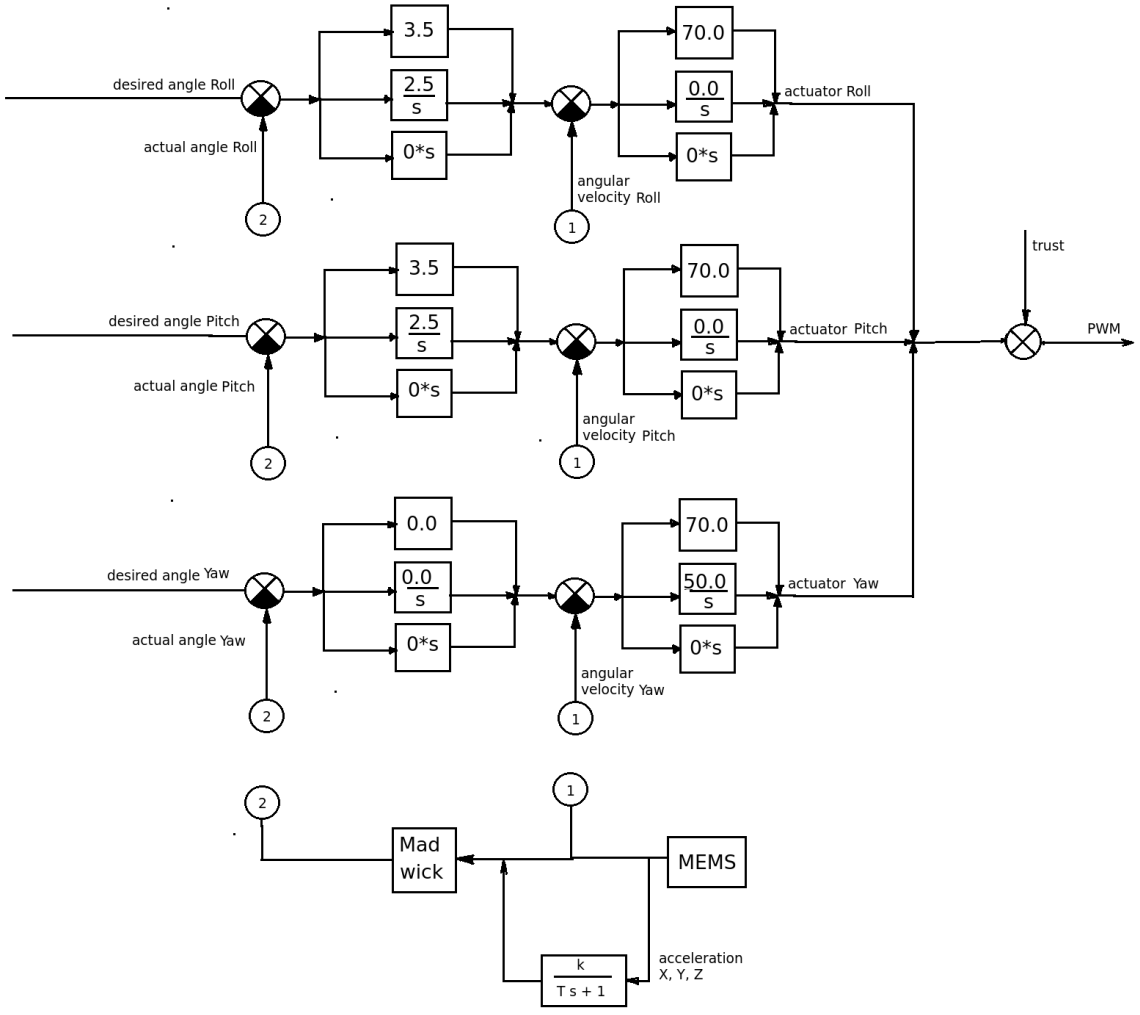


图 37: Crazyflie 控制系统

(续上页)

```
      attitudeDesired.yaw += setpoint->attitudeRate.yaw * ATTITUDE_UPDATE_DT;
      while (attitudeDesired.yaw > 180.0f)
        attitudeDesired.yaw -= 360.0f;
      while (attitudeDesired.yaw < -180.0f)
        attitudeDesired.yaw += 360.0f;
    } else {
↪                                     //attitude 模式
      attitudeDesired.yaw = setpoint->attitude.yaw;
    }
  }

  if (RATE_DO_EXECUTE(POSITION_RATE, tick)) {
↪      //位置控制
    positionController(&actuatorThrust, &attitudeDesired, setpoint, state);
  }

  if (RATE_DO_EXECUTE(ATTITUDE_RATE, tick)) {
    // Switch between manual and automatic position control
    if (setpoint->mode.z == modeDisable) {
      actuatorThrust = setpoint->thrust;
    }
    if (setpoint->mode.x == modeDisable || setpoint->mode.y == modeDisable) {
      attitudeDesired.roll = setpoint->attitude.roll;
      attitudeDesired.pitch = setpoint->attitude.pitch;
    }

    attitudeControllerCorrectAttitudePID(state->attitude.roll, state->attitude.pitch,
↪state->attitude.yaw,
                                         attitudeDesired.roll, attitudeDesired.pitch,
↪attitudeDesired.yaw,
                                         &rateDesired.roll, &rateDesired.pitch, &rateDesired.
↪yaw);

    // For roll and pitch, if velocity mode, overwrite rateDesired with the setpoint
    // value. Also reset the PID to avoid error buildup, which can lead to unstable
    // behavior if level mode is engaged later
    if (setpoint->mode.roll == modeVelocity) {
      rateDesired.roll = setpoint->attitudeRate.roll;
      attitudeControllerResetRollAttitudePID();
    }
    if (setpoint->mode.pitch == modeVelocity) {
      rateDesired.pitch = setpoint->attitudeRate.pitch;
      attitudeControllerResetPitchAttitudePID();
```

(下页继续)

(续上页)

```
    }

    // TODO: Investigate possibility to subtract gyro drift.
    attitudeControllerCorrectRatePID(sensors->gyro.x, -sensors->gyro.y, sensors->gyro.
↪z,
                                     rateDesired.roll, rateDesired.pitch, rateDesired.yaw);

    attitudeControllerGetActuatorOutput(&control->roll,
                                        &control->pitch,
                                        &control->yaw);

    control->yaw = -control->yaw;
  }

  if (tiltCompensationEnabled)
  {
    control->thrust = actuatorThrust / sensfusion6GetInvThrustCompensationForTilt();
  }
  else
  {
    control->thrust = actuatorThrust;
  }

  if (control->thrust == 0)
  {
    control->thrust = 0;
    control->roll = 0;
    control->pitch = 0;
    control->yaw = 0;

    attitudeControllerResetAllPID();
    positionControllerResetAllPID();

    // Reset the calculated YAW angle for rate control
    attitudeDesired.yaw = state->attitude.yaw;
  }
}
```

Mellinger 控制器

Mellinger 控制器是一种 **多合**一控制器，基于目标位置和目标位置速度矢量，直接计算出需要分配给所有电动机的所需推力。

详情可参考论文：Minimum snap trajectory generation and control for quadrotors。

INDI 控制器

INDI 控制器是立即处理角速率以确定信任度的控制器，与传统的 PID 控制器相结合，对于角度处理相比串级 PID 控制器组合的速度要快。

详情可参考论文：Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles。

PID 参数整定

Rate PID 整定

1. 先调整 Rate 模式，将 rollType、pitchType 和 yawType 都调整为 RATE；
2. 将 ATTITUDE 模式对应的 roll、pitch 和 yaw 的 KP、KI 和 KD 调整为 0.0，仅保留 Rate 相关的参数；
3. 将 RATE 模式对应的 roll、pitch 和 yaw 的 KI 和 KD 调整为 0.0，先调整比例控制 KP；
4. 烧写代码，使用 cfclient 的 param 功能开始在线进行 KP 的调整；
5. 注意，使用 cfclient 修改后的参数，掉电不保存；
6. 在 PID 调整期间会出现震荡（超调）的情况，请注意安全；
7. 先固定住飞行器，让其只能进行 pitch 轴的翻转。逐渐增加 pitch 对应的 KP，直到飞机出现前后的震荡；
8. 当出现严重的震荡时，可以稍微降低 KP，以恰好达到震荡的临界点为基础，降低 5-10 个百分点即可确定 KP 参数；
9. 使用同样的方法调整 roll 和 yaw；
10. 调整 KI，该参数用于消除稳态误差。如果不引入该参数，只有比例调整的话，飞行器受到重力等干扰会在 0 位置上下摆动。设置 KI 的初始值为 KP 的 50%；
11. 当 KI 增大到一定程度，也会导致飞机不稳定晃动。但 KI 造成的晃动频率相比 KP 带来的震动，频率更小。以恰好造成震动的临界 KI 为基础，减小 5-10 个百分点，确定最终的 KI 值；
12. 使用同样的方法调整 roll 和 yaw；
13. 一般情况下 KI 的取值为 KP 取值的 80% 以上。

Attitude PID 整定

1. 确保 Rate PID 调整已经完成；
2. 将 rollType、pitchType 和 yawType 都调整为 ANGLE，即飞机已进入 attitude mode；
3. 改变 roll 和 pitch 的 KI 和 KD 为 0.0，将 Yaw 的 KP、KI、KD 都设置为 0.0；
4. 烧写代码，使用 cfclient 的 param 功能开始在线进行 KP 的调整；
5. 将 roll 和 pitch 的 KP 设置为 3.5，查找任何存在的不稳定性，例如振荡。持续增加 KP，直到达到极限；
6. 如果发现 KP 导致不稳定，如果此时已经高于 4，需要将 RATE 模式的 KP 和 KI 稍微降低 5 ~ 10 点。实现调整姿势模式时更加自由；
7. 要调整 KI，请再次缓慢增加 KI。不稳定性的状态是产生低频振荡。

3.3.4 通信协议

[English]

通信层级结构

终端	手机/PC	ESP-Drone
应用层	APP	飞控固件
协议层	CRTP	CRTP
传输层	UDP	UDP
物理层	Wi-Fi STA (Station)	Wi-Fi AP (Access Point)

Wi-Fi 通信

Wi-Fi 性能

ESP32 Wi-Fi 性能

项目	参数
模式	STA 模式、AP 模式、共存模式
协议	IEEE 802.11b、IEEE 802.11g、IEEE 802.11n、802.11 LR（乐鑫）支持软件切换
安全性	WPA、WPA2、WPA2-Enterprise、WPS
主要特性	AMPDU、HT40、QoS
支持距离	乐鑫专属协议下 1 km
传输速率	20 Mbit/s TCP 吞吐量、30 Mbit/s UDP

其它参数见 ESP32 Wi-Fi 特性列表。

ESP32-S2 Wi-Fi 性能

项目	参数
模式	STA 模式、AP 模式、共存模式
协议	IEEE 802.11b、IEEE 802.11g、IEEE 802.11n 支持软件切换
安全性	WPA、WPA2、WPA2-Enterprise、WPS
主要特性	AMPDU、HT40、QoS
支持距离	乐鑫专属协议下 1 km
传输速率	20 Mbit/s TCP 吞吐量、30 Mbit/s UDP

其他参数见 ESP32-S2 Wi-Fi 特性列表。

Wi-Fi 编程框架

基于 ESP-IDF 的 Wi-Fi 编程框架：

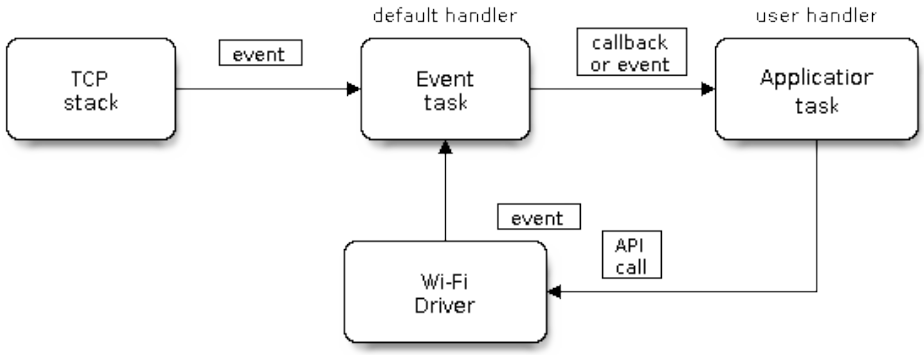


图 38: Wi-Fi 编程模型

一般使用过程如下：

1. 应用层调用 Wi-Fi 驱动 API，进行 Wi-Fi 初始化。
2. Wi-Fi 驱动对开发人员透明。事件发生，则 Wi-Fi 驱动向默认事件循环：default event loop 发布 `event`。应用程序可根据需求编写 `handle` 程序，进行注册。
3. 网络接口组件 esp_netif 提供了一系列 `handle` 程序，与 Wi-Fi 驱动 `event` 默认关联。例如 ESP32 作为 AP，当有用户接入时，esp_netif 将自动启动 DHCP 服务。

具体的使用过程，可查阅代码 `\components\drivers\general\wifi\wifi_esp32.c`。

注解： Wi-Fi 初始化之前应使用 `WIFI_INIT_CONFIG_DEFAULT` 获取初始化配置结构体，对该结构体进行个性化配置，然后进行初始化工作。请注意防范结构体成员未初始化导致的问题，在 ESP-IDF 更新添加了新的结构体成员时，应尤其特别注意这一问题。

AP 模式工作状态图：

提高 Wi-Fi 通信距离

依次进入：`Component config>>PHY>>Max WiFi TX power (dBm)`，将 `Max WiFi TX power` 改为 `20`。该项配置将提高 PHY 增益，提高 Wi-Fi 通信距离。

UDP 通信

UDP 端口号

App	方向	ESP-Drone
192.168.43.42::2399	TX/RX	192.168.43.42::2390

UDP 包结构

```
/* Frame format:
 * +=============+-----+-----+
 * | CRTP                      |    CKSUM    |
 * +=============+-----+-----+
 */
```

- UDP 传输的数据包为：CRTP + 校验信息。
- CRTP：按照 CRTP 包结构定义，包含 Header + Data，具体参考 CRTP 协议部分。
- CKSUM：为校验信息，大小为 1 byte，将 CRTP 包按照 byte 累加即可。

CKSUM 计算方法

```
#python 为例：计算 raw 的 cksum，并将其添加到包尾
 raw = (pk.header,) + pk.datat
 cksum = 0
 for i in raw:
        cksum += i
 cksum %= 256
 raw = raw + (cksum,)
```

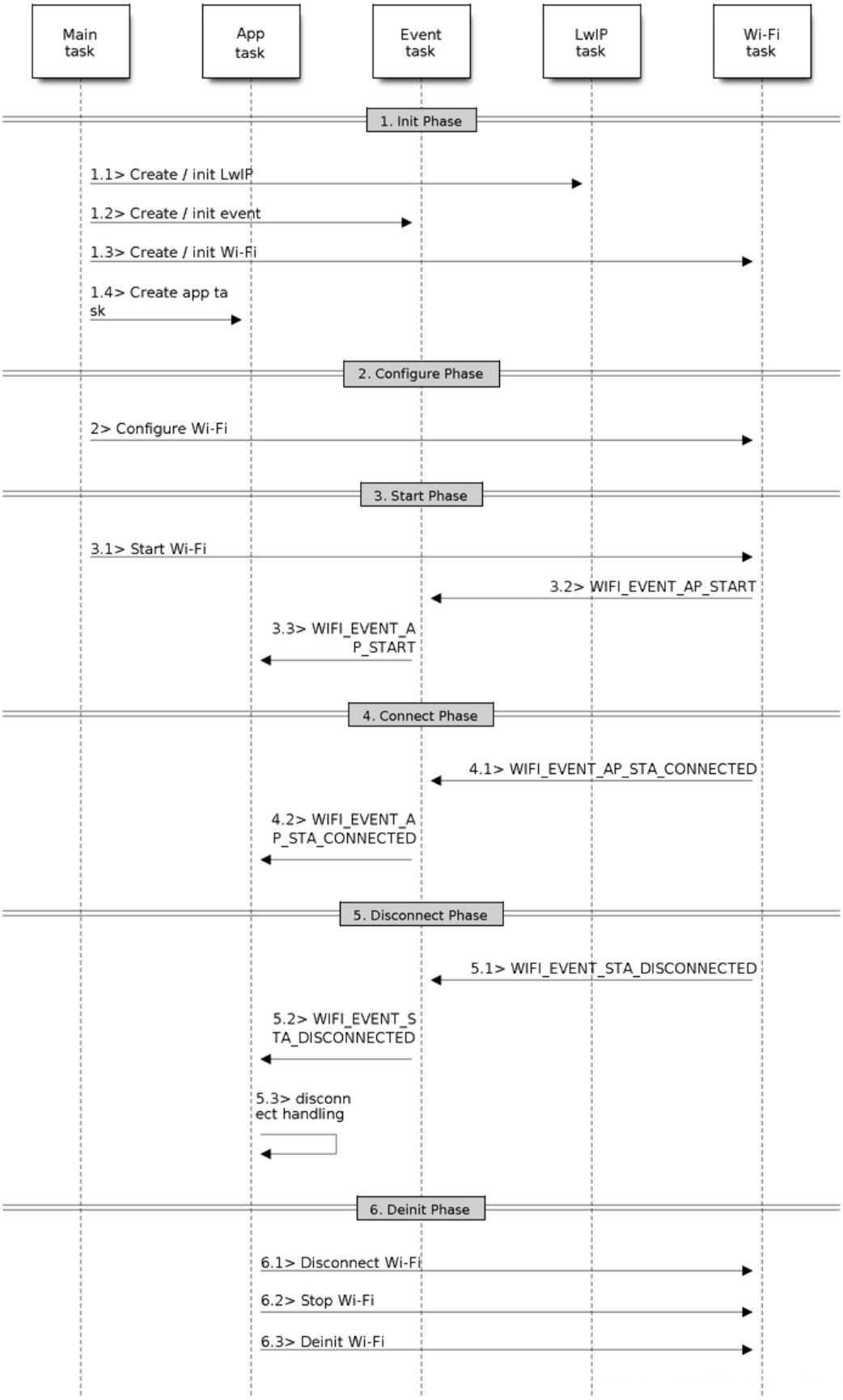


图 39: AP 模式下 Wi-Fi Event 示例

CRTP 协议

ESP-Drone 项目继承 Crazyflie 项目使用的 CRTP 协议，用于飞行指令发送、飞行数据回传、参数设置等。

CRTP 实现了无状态设计，不需要握手步骤。任何命令均可在任意时刻发送，但对于一些 log/param/mem 命令，需下载 TOC (目录)，协助主机正确发送信息。已经实现的 Python API (cflib) 实现下载 param/log/mem TOC，确保能够使用所有功能。

CRTP 包结构

CRTP 包大小为 32 字节，其中包含一个字节的 Header，31 个字节的 Payload。Header 记录端口（4 位）、通道（2 位）、及保留位（2 位）。

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
|     Port      |  Res. | Chan. |
+---+---+---+---+---+---+---+---+
|             DATA 0            |
+---+---+---+---+---+---+---+---+
:   :   :   :   :   :   :   :   :
+---+---+---+---+---+---+---+---+
|             DATA 30           |
+---+---+---+---+---+---+---+---+
```

字段	字节	位	描述
Header	0	0 ~ 1	目标数据通道
	0	2 ~ 3	保留，用于传输层
	0	4 ~ 7	目标数据端口
Data	1 ~ 31	0 ~ 7	数据包中的数据

端口分配

端口号	数据端口	用途
0	Console	Console 使用 consoleprintf 函数将调试信息输出到 PC 端。
2	Parameters	读写 Crazyflie 参数。参数可在源码中用宏表示。
3	Commander	发送 roll/pitch/yaw/thrust 控制指令。
4	Memory access	访问非易失性存储，如 1 线访问和 I2C 访问。仅支持 Crazyflie 2.0
5	Log	设置日志变量。日志变量将定期发送至 Crazyflie，日志变量在 Crazyflie 源码中用宏表示。
6	Localization	本地化相关包
7	Generic Setpoint	运行发送定位点和控制模式
13	Platform	用于 misc platform 控制，如调试和掉电等
14	Client-side debugging	用于调试 PC 端 UI 界面程序，只针对 Crazyflie Python API。
15	Link layer	用于控制和访问通信链路层。

固件中大部分连接到端口的模块，以任务的方式实现。如果有传入的 CRTP 包在信息传递队列中传递，则任务在队列中阻塞。启动时，每个任务及其它模块需要在通信链路层为预定义的端口注册。

各个端口使用详情可参考：CRTP - 与 Crazyflie 通信。

CRTP 协议支持包

cflib 是 CRTP 协议的 Python 支持包，提供了通信协议的应用层接口，可用于构建上位机，与 Crazyflie 和 Crazyflie 2.0 四轴飞行器通信并控制飞行器。固件中每一个使用 CRTP 协议的组件，在 cflib 中都有一个脚本与其对应。

- 源工程仓库地址：crazyflie-lib-python。
- 适配 ESP-Drone 的 cflib 工程仓库地址：qljz1993/crazyflie-lib-python。需要切换到 `esplane` 分支。

基于 CRTP 协议的应用开发

各个平台工程模板

1. crazyflie2-ios-client
2. crazyflie2-windows-uap-client
3. crazyflie-android-client
4. 安卓版本使用指南

5. 安卓版本开发指南

cfclient

cfclient 是 `Crazeflie` 源工程的上位机，完全实现了 `CRTP` 协议中定义的功能，可以加快飞机的调试过程。ESP-Drone 项目对该上位机进行裁剪和调整，满足功能设计需求。

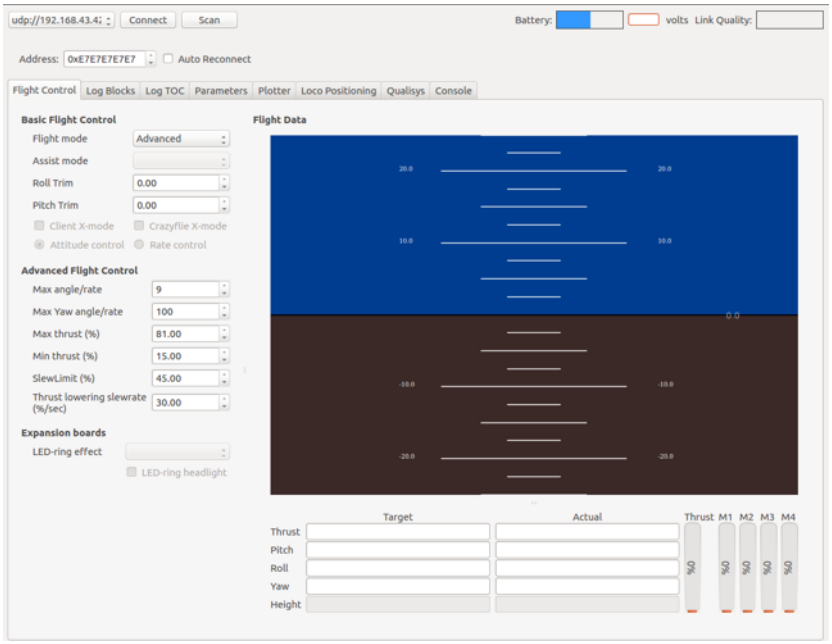


图 40: cfclient 控制台界面

cfclient 具体使用说明可查阅：cfclient。

3.4 第三方代码

[English]

第三方代码及证书如下：

组件	License	源代码	Commit ID
core/crazyflie	GPL-3.0	Crazyflie	tag_2021_01 b448553
lib/dsp_lib		esp32-lin	6fa39f4cd5f7782b3a2a052767f0fb06be2378ff

3.5 致谢

1. 感谢 Bitcraze 开源组织提供很棒的 Crazyflie 无人机项目代码；
2. 感谢乐鑫提供 ESP32 和 ESP-IDF 操作系统；
3. 感谢 WhyEngineer 提供的 dsp 移植库 esp-dsp。