ESP-Drone

Aug 14, 2024

CONTENTS

[??]

ESP-Drone is an open source **drone solution** based on Espressif **ESP32/ESP32-S2/ESP32-S3** Wi-Fi chip, which can be controlled over a **Wi-Fi** network using a mobile APP or gamepad. ESP-Drone supports multiple flight modes, including *Stabilize mode*, *Height-hold mode*, and *Position-hold mode*. With **simple hardware**, **clear and extensible code architecture**, ESP-Drone can be used in **STEAM education** and other fields. The main code is ported from **Crazyflie** open source project with **GPL3.0** protocol.

This document describes the development guide for ESP-Drone project.

CHAPTER

ONE

ESP-DRONE SUPPORT POLICY

From December 2022, we will offer limited support on this project, but Pull Request is still welcomed!

CHAPTER

TWO

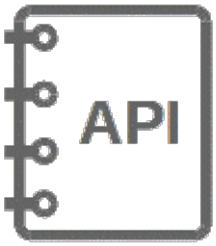
SUPPORTED VERSIONS AND CHIPS

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

CHAPTER

THREE

GET STARTED

		
Get Started	Develop Guide	H/W Reference

3.1 Get Started

[??]

3.1.1 ESP-Drone Overview

ESP-Drone is an ESP32/ESP32-S2/ESP32-S3 based flying development board provided by Espressif. ESP-Drone is equipped with Wi-Fi key features, which allows this drone to be connected to and controlled by an APP or a gamepad over a Wi-Fi network. This drone comes with simple-structured hardware, clear codes, and supports functional extension. Therefore, ESP-Drone can be used in STEAM education. Part of the codes is from Crazyflie open source project under GPL3.0 license.

Main Features

ESP-Drone has the following features:

- Stabilize mode: keep the drone stable to achieve smooth flight.
- Height-hold mode: control thrust output to keep the drone flying at a fixed height.
- Position-hold mode: keep the drone flying at a fixed position.
- PC debugging: use cfclient for static/dynamic debugging.
- Controlled by APP: easily controlled over Wi-Fi by your mobile APP.

Fig. 1: ESP-Drone

- Controlled by gamepad: easily controlled via the gamepad by cfclient.

Main Components

ESP-Drone 2.0 consists of a main board and several extension boards:

- **Main board**: integrates an ESP32-S2 module, necessary sensors for basic flight, and provides hardware extension interfaces.
- **Extension boards**: integrate extension sensors via hardware extension interfaces of the main board, to implement advanced flight.

No.	Modules	Main Components	Function	Interfaces	Mount Location
1	Main board - **ESP32-S2**	ESP32-S2-WROVER + MPU6050	Basic flight	I2C, SPI, GPIO, extension interfaces	
2	Extension board - **Position-hold module**	PMW3901 + VL53L1X	Indoor position-hold flight	SPI + I2C	Mount at bottom, facing to the ground.
3	Extension board - **Pressure module**	MS5611 pressure module	Height-hold flight	I2C or MPU6050 slave	Mount at the top or at the bottom
4	Extension board - **Compass module**	HMC5883 compass	Advanced flight mode, such as head-free mode	I2C or MPU6050 slave	Mount at the top or at the bottom

For more information, please refer to Hardware Reference.

3.1.2 ESP-IDF Overview

ESP-IDF is the IoT Development Framework provided by Espressif for ESP32/ESP32-S2/ESP32-S3.

- ESP-IDF is a collection of libraries and header files that provides the core software components that are required to build any software projects on ESP32/ESP32-S2.
- ESP-IDF also provides tools and utilities that are required for typical developer and production use cases, like build, flash, debug and measure.

For more information, please check ESP-IDF Programming Guide.

3.1.3 Crazyflie Overview

Crazyflie is a Bitcraze open-source quadcopter, with the following features:

- Support various sensor combinations for advanced flight modes, such as Height-hold mode and Position-hold mode.
- Based on FreeRTOS, which allows users to break down complex drone systems into multiple software tasks with various priorities.
- Customized full-featured cfclient and CRTP communication protocol, for debug, measurement, and control purposes.

Fig. 2: A swarm of drones exploring the environment, avoiding obstacles and each other. (Guus Schoonewille, TU Delft)

For more information, see Crazyflie.

3.1.4 Preparations

Assemble Hardware

Please follow the steps below to assemble ESP32-S2-Drone V1.2.

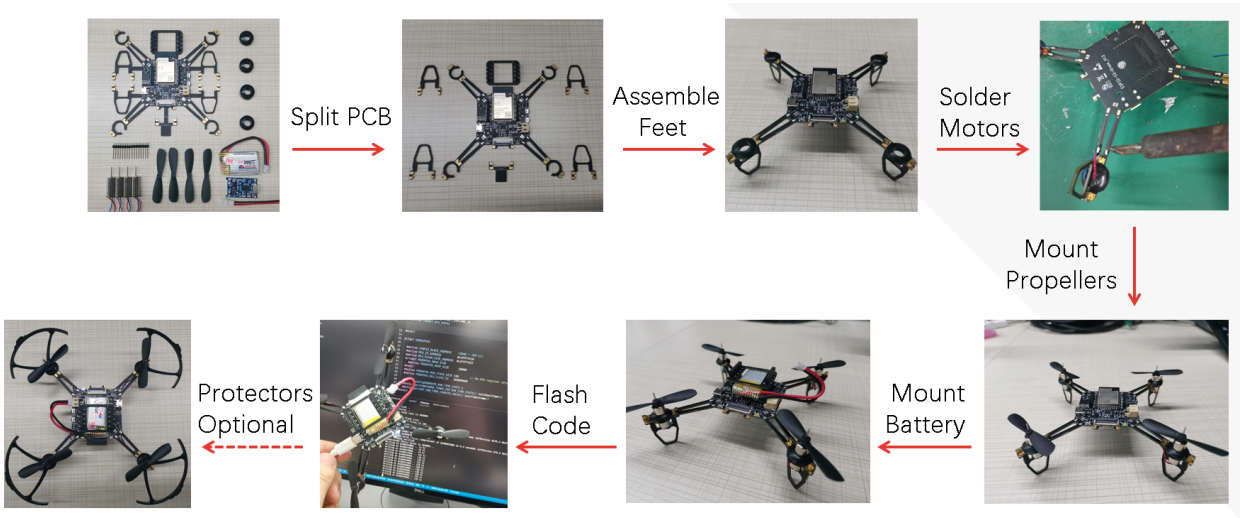


Fig. 3: ESP32-S2-Drone V1.2 Assemble Flow

Hardware overview and pin allocation are available at: Hardware Reference.

Download and Install ESP-Drone APP

ESP-Drone APP is available for Android and iOS.

For Android, please scan the QR below to download ESP-Drone APP.

For iOS, please search and download the ESP-Drone APP in App Store.

iOS APP source code: ESP-Drone-iOS

Android APP source code: ESP-Drone-Android

Install cfclient

This step is optional, only for advanced debugging.

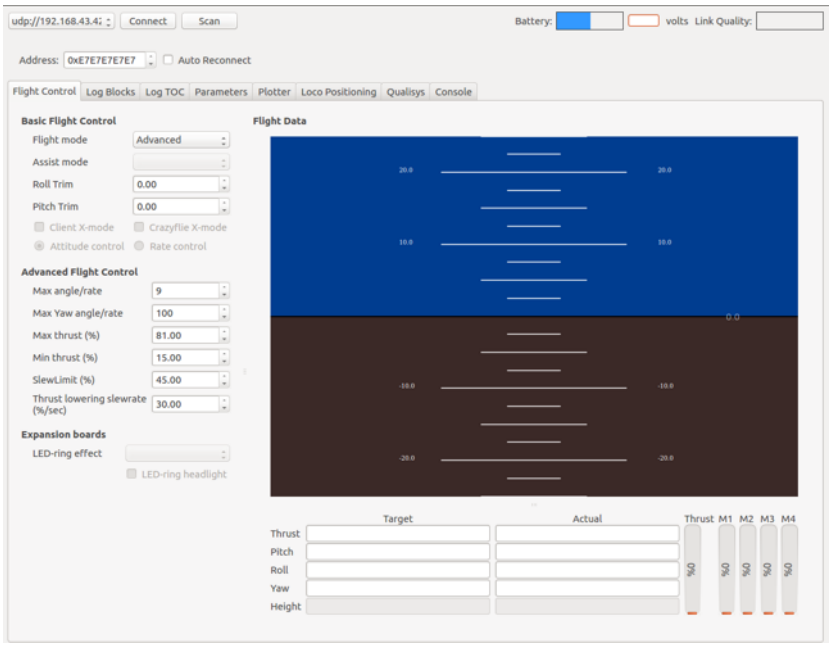


Fig. 4: cfclient Interface

1. Install cfclient

1.1 Download the source code

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

1.2 Check into the directory

```
cd crazyflie-clients-python
```

1.3 Install cfclient

```
pip3 install -e .
```

1.4 Start cfclient

```
cfclient
```

2. Configure the controllers

2.1 Configure the four main dimensions of controls: `Roll, Pitch, Yaw, Thrust`.

2.2 Configure button `Assisted control` for flight mode switching.

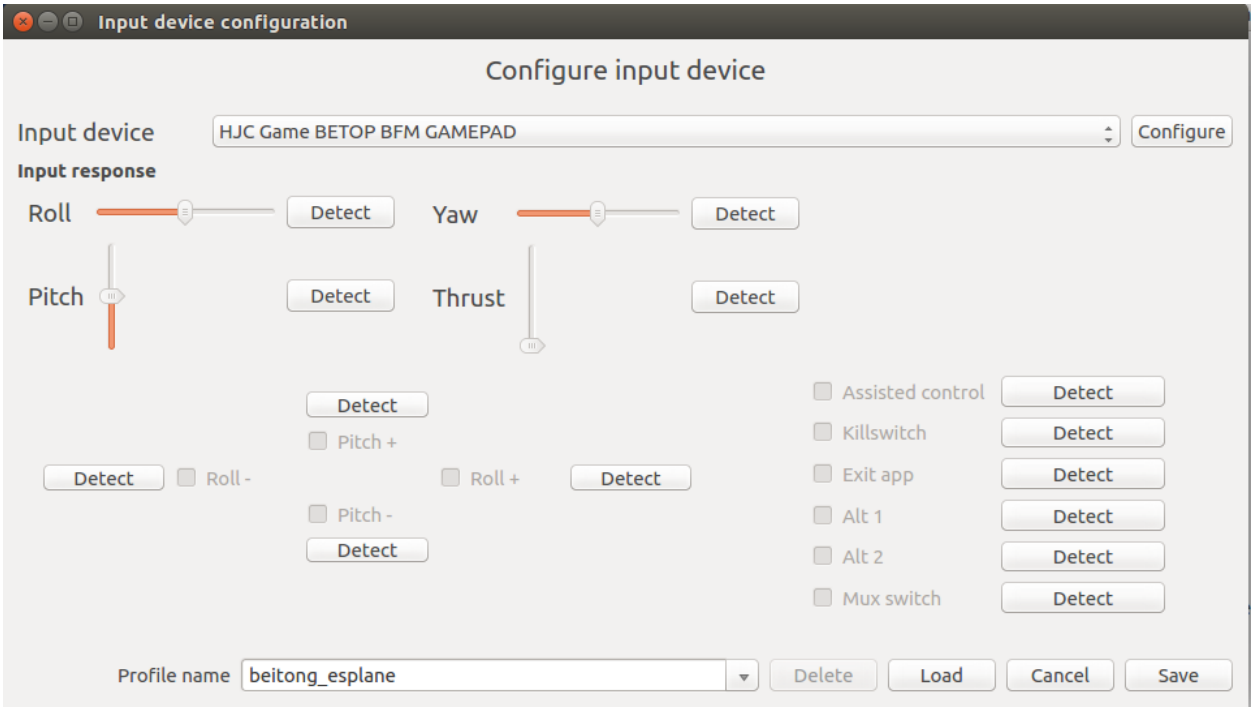


Fig. 5: Controller Configuration

3.1.5 ESP-Drone APP Guide

Establish Wi-Fi connection

- Scan Wi-Fi AP on your mobile. ESP-Drone device works as a Wi-Fi AP with the following SSID:

```
SSID: ESP-DRONE_XXXX (XXXX is configured based on MAC) PASSWORD: 12345678
```

- Click this AP, connect your mobile to this AP.

Now a Wi-Fi connection is established between your mobile and your drone.

Customize settings

In this step, you can customize the flight settings according to your application scenarios, or use the default configuration below.

```
```
Default configuration:

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
```

(continues on next page)
```

(continued from previous page)

```
        2. max yaw angle: 90
        3. max thrust: 90
        4. min thrust: 25
        5. X-Mode: true
Controller settings
    7. use full travel for thrust: false
    8. virtual joystick size: 100
App settings
    9. Screen rotation lock: true
    10. full screen mode:true
    11. show console: true
```

Flight Control

- Click “Connect” button/icon at your APP. When the connection is established successfully between your drone and APP, the LED on the drone blinks GREEN.
- Slide “Thrust” slightly to take off the drone.
- Control the flight by moving your fingers on the APP.

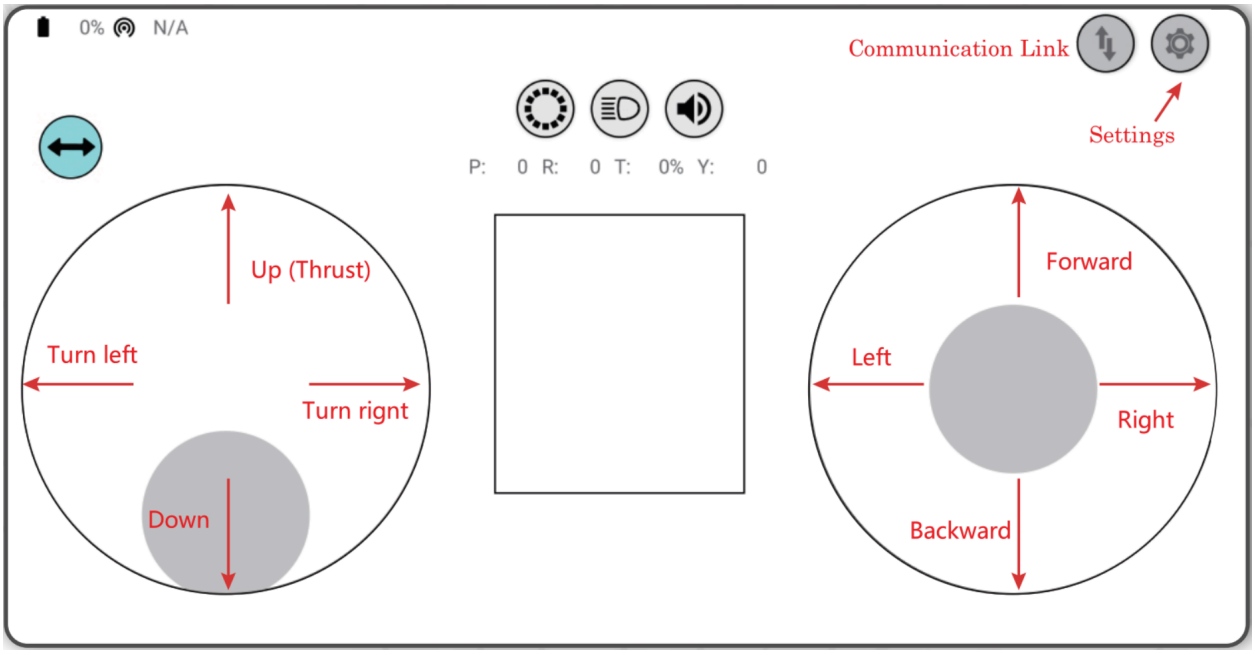


Fig. 6: Android APP Interface

3.1.6 PC cfclient Guide

Cfclient is the PC client for `Crazeflie` source project, which has fully implemented the functions defined in `CRTP` and makes the drone debugging faster. ESP-Drone customizes this cfclient to meet functional design needs.

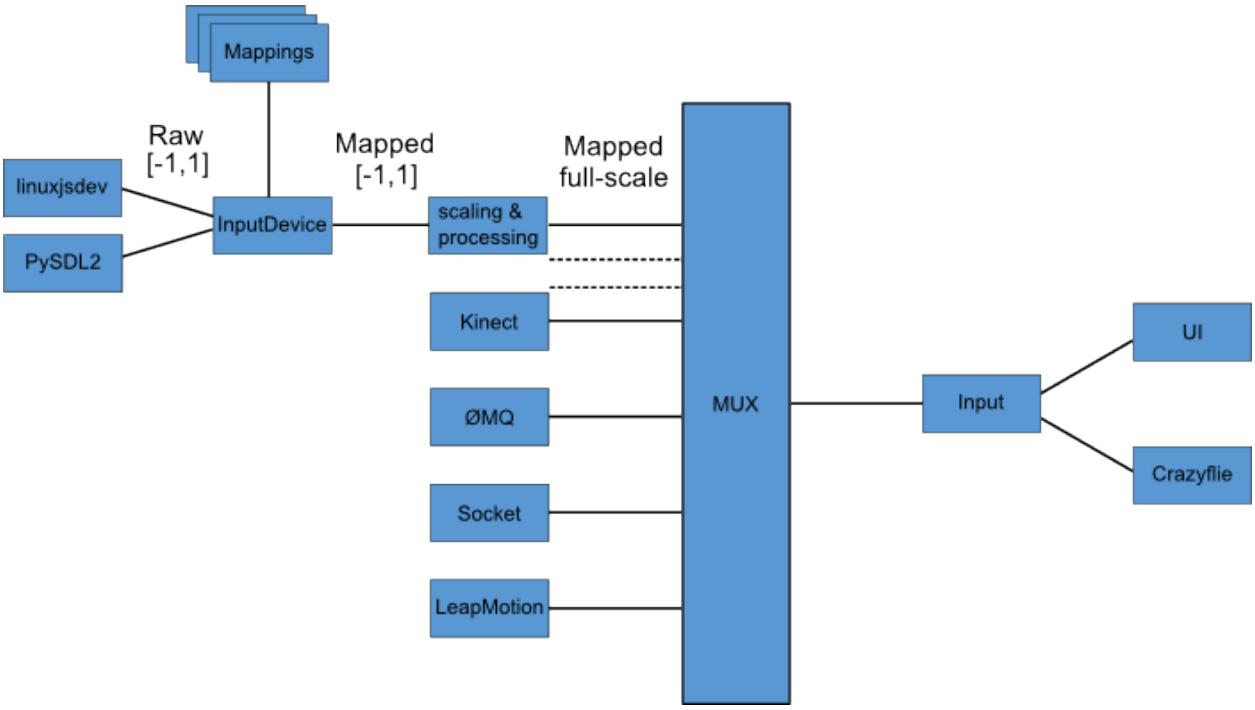


Fig. 7: Cfclient Architecture

In this project, we have configuration files and cache files. JSON file is used to store configuration information. For more information about the configuration, please refer to User Configuration File.

Flight Settings

Basic Flight Control

1. Flight mode: normal and advanced modes
    - Normal mode: for beginners.
    - Advanced mode: unlock the maximum angle and the maximum thrust.
2. Assisted mode
    - Altitude-hold mode: maintain flight altitude. To implement this mode, a barometric pressure sensor is needed.
    - Position-hold mode: maintain current flight position. To implement this mode, an optical flow sensor and a Time of Flight (TOF) sensor are needed.
    - Height-hold mode: keep flight height. Note: to apply this mode, the drone should fly at 40 cm or higher over the ground, and a TOF is needed.
    - Hover mode: stay and hover at 40 cm or higher over the take-off point. To implement this mode, a optical flow sensor and a TOF are needed.
3. Trim

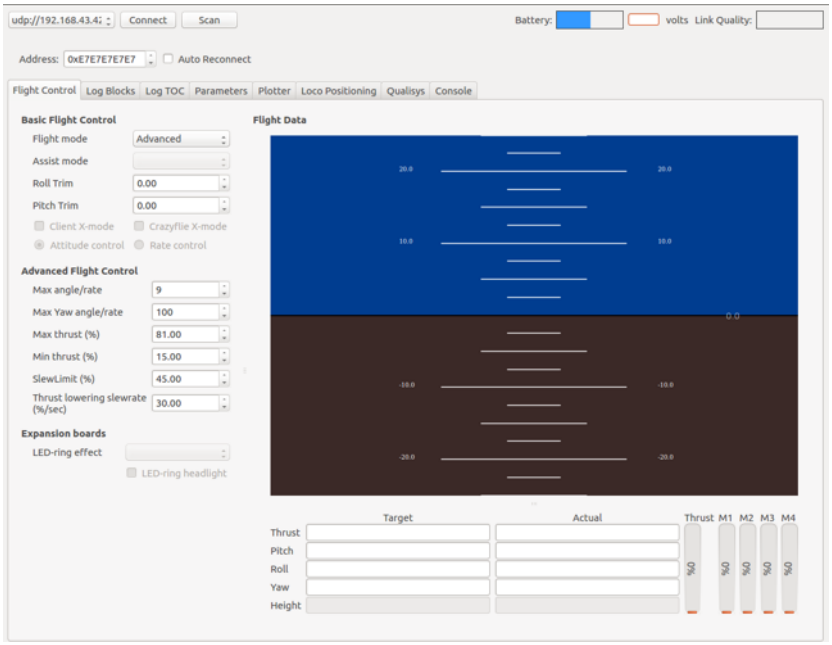


Fig. 8: Cfclient Console Interface

- Roll Trim: trim the rotation around a horizontal axis going through the drone from back to front. This rotation literally rolls the drone and moves it left and right. Roll trim is used to compensate for the level installation deviation of sensors.
- Pitch Trim: trim the rotation around a horizontal axis going through the drone from left to right. This rotation tilts the drone and moves it forwards or backwards. Pitch trim is used to compensate for the level installation deviation of sensors.

Note that in assisted mode, the thrust controller works as a height controller.

Advanced Flight Control

1. Max angle: set the maximum pitch and roll rotation: roll/pitch.
2. Max yaw rate: set the allowed yaw: `yaw`.
3. Max thrust: set the maximum thrust.
4. Min thrust: set the minimum thrust.
5. Slew limit: prevent sudden drop of thrust. When the thrust drops below this limit, the rates below `Slew rate` will not be allowed.
6. Slew rate: this is the maximum rate when the thrust is below `slew limit`.

Configure Input Device

Follow the prompts, route the controllers to each channel.

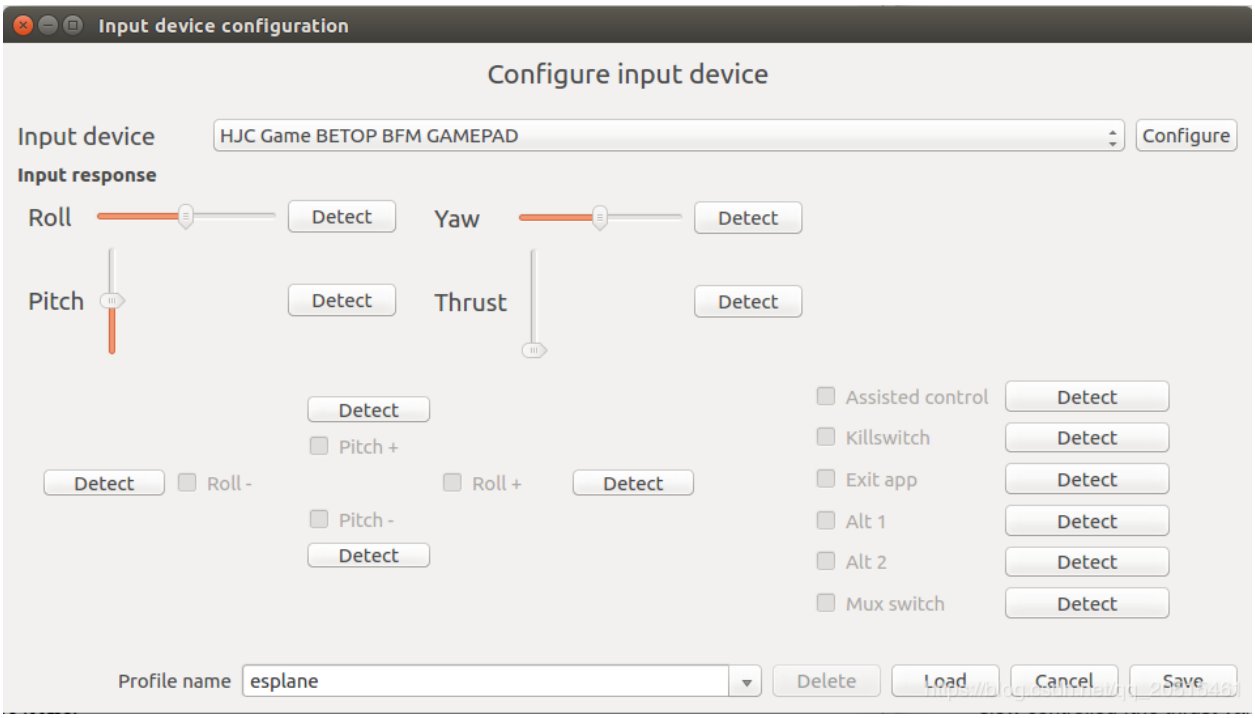


Fig. 9: Cfclient Input Device Configuration

Flight Data

On the tab “Flight Control” of cfclient, you can check the drone status. The detailed information is shown at the bottom right, including:

1. Target: target angle
2. Actual: measured angle
3. Thrust: current thrust value
4. M1/M2/M3/M4: actual output of motors

Tune Online Parameters

Tune PID parameters online

Note

1. The modified parameters take effect in real time, which avoids frequent flash of firmware.
2. You can define in your code which parameters can be modified by PC in real time.
3. Note that modifying parameters online is only for debugging purpose. The modified parameters will not be saved at power down.

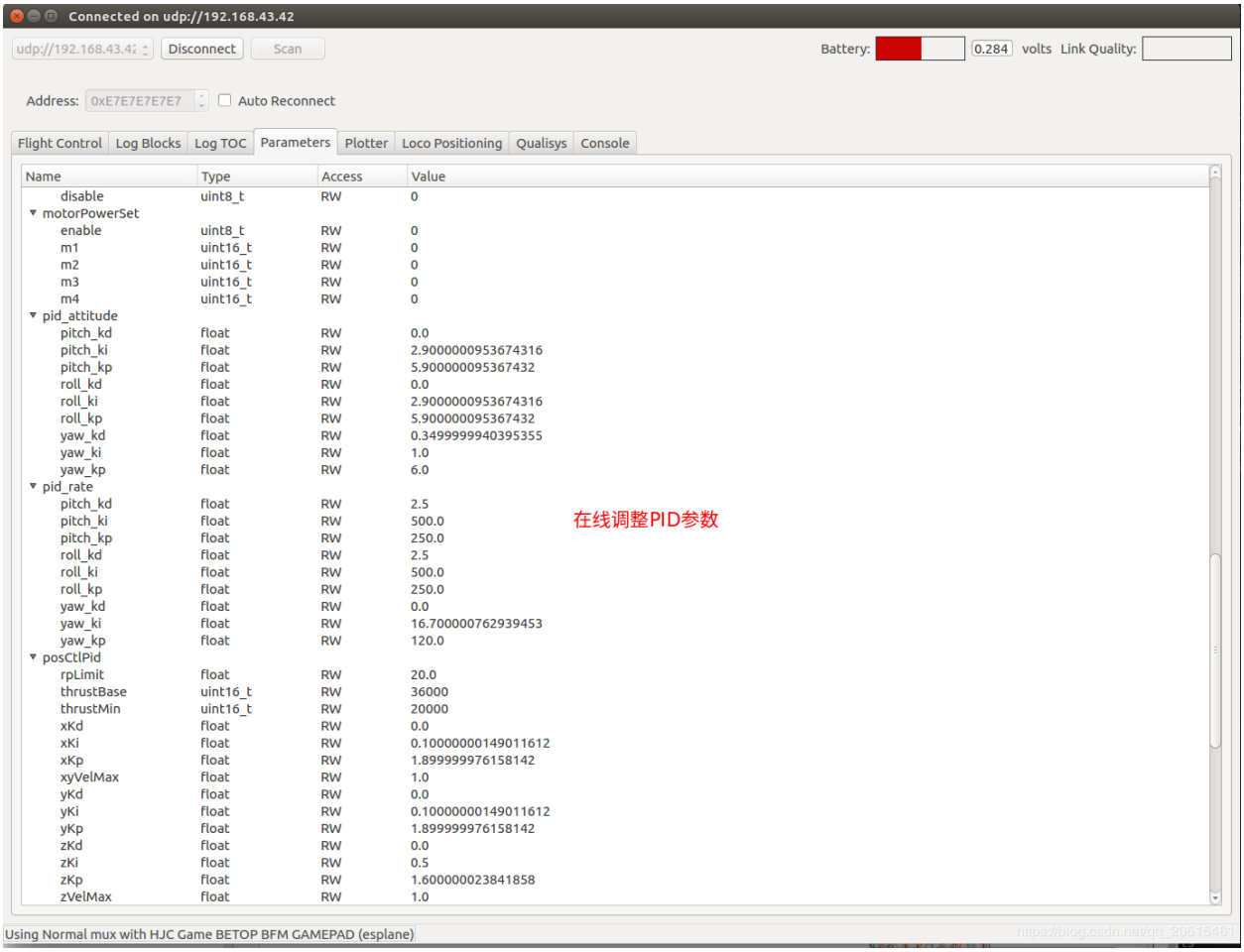


Fig. 10: Cfclient PID Parameters Tunning

Monitor Flight Data

Configure the parameters to monitor at Tab Log configuration and Tab Log Blocks:

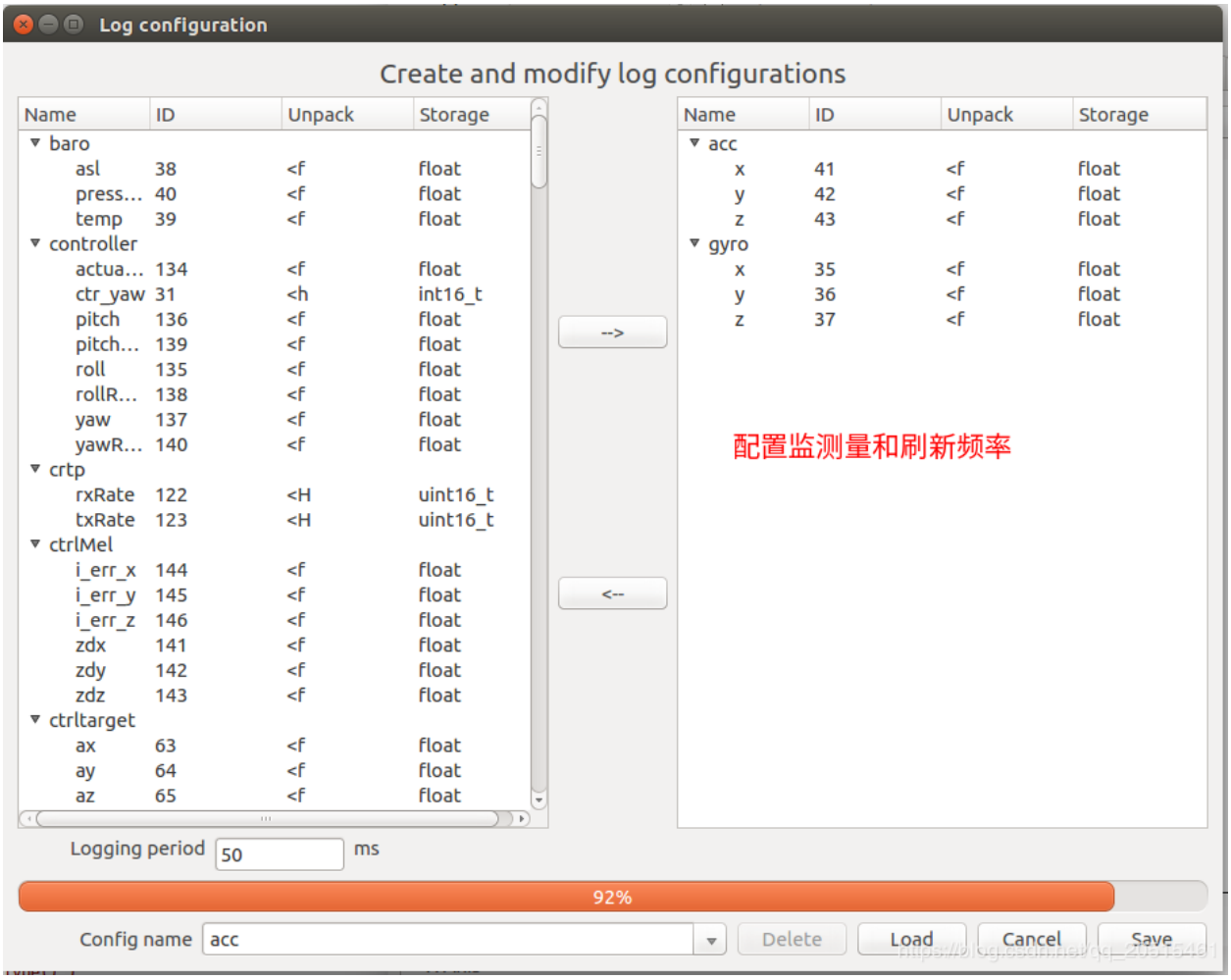


Fig. 11: Log Configuration

Configure real-time waveform drawing at Tab Plotter, to monitor gyro accelerometer data.

3.1.7 Propeller Direction

- Install A and B propellers according to the figure below.
- During the power-on self-test, check if the propellers spin properly.

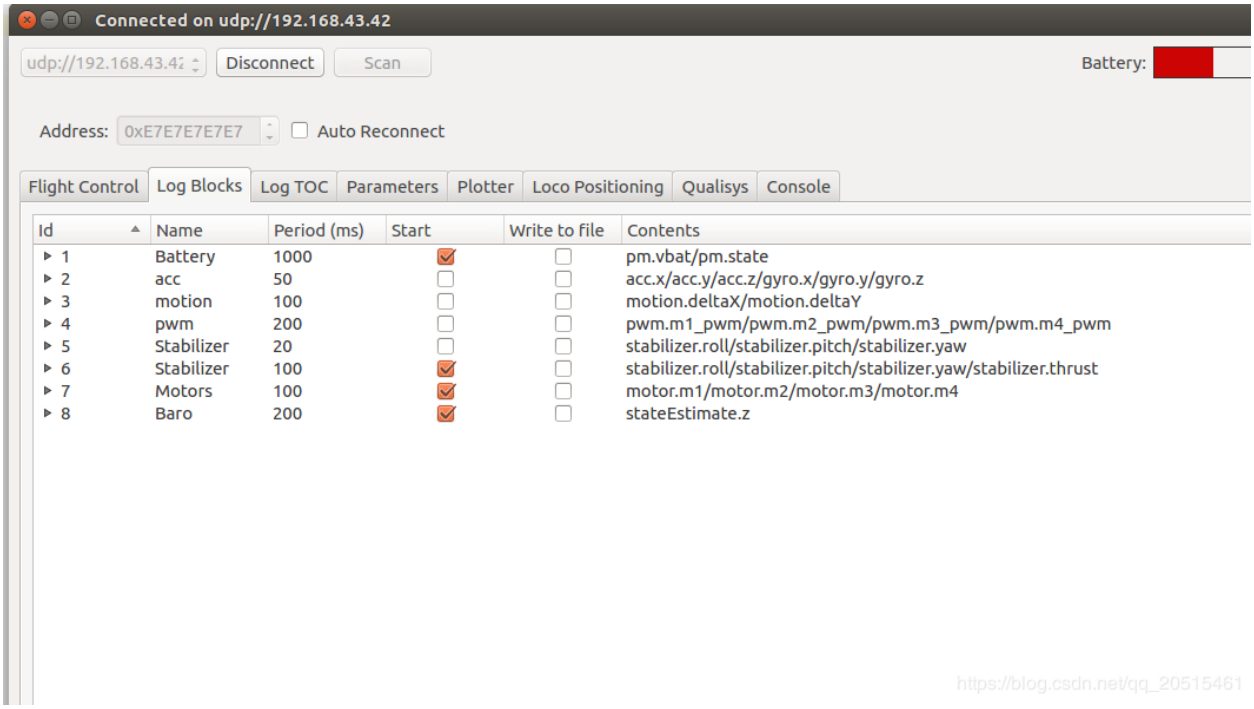


Fig. 12: Log Blocks

3.1.8 Preflight Check

- Place the drone with its head on the front, and its tail (i.e. the antenna part) at the back.
- Place the drone on a level surface and power it up when the drone stays still.
- Check on the cfclient if the drone is placed level.
- After the communication is established, check if the LED at the drone tail blinks GREEN fast.
- Check if the LED on the drone head blinks RED, which indicates battery LOW.
- Slide forward the Trust controller slightly at the left side of your APP (i.e. the commands controlled by your left finger), to check if the drone can respond the command quickly.
- Move your finger at the right command area of the APP (i.e. the commands controlled by your right finger), to check if the direction control works well.
- Go fly and have fun!

3.2 Set up Development Environment

[??]

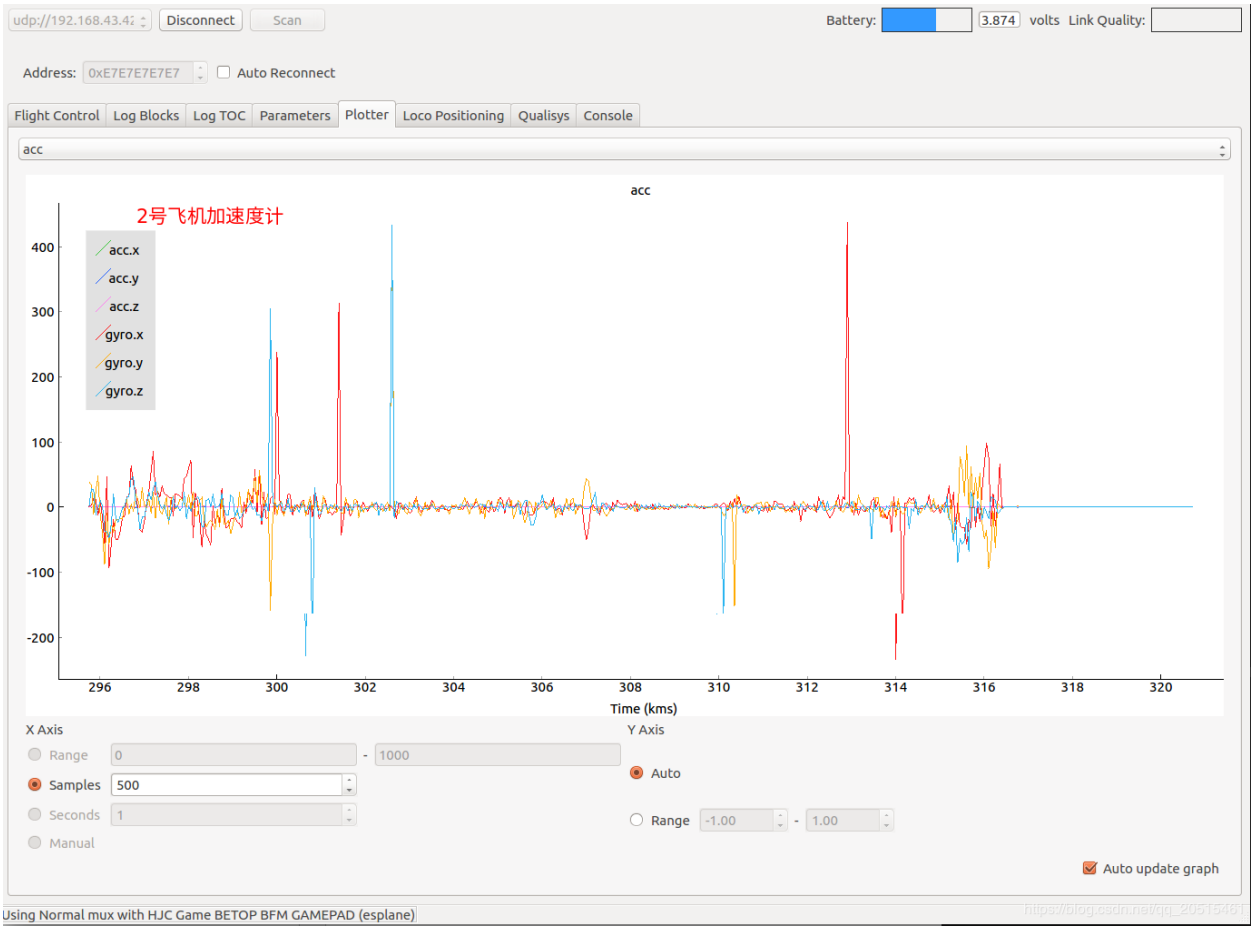


Fig. 13: Accelerometer Logging

Fig. 14: Propeller Check

3.2.1 Set up ESP-IDF Environment

Please refer to ESP-IDF Programming Guide and set up ESP-IDF environmnet step by step.

- ESP-IDF branch `release /v4.4` is suggested.
- Please follow and complete all setup steps.
- Build a example of ESP-IDF to make sure the setup is successful.

3.2.2 Get Project Source Code

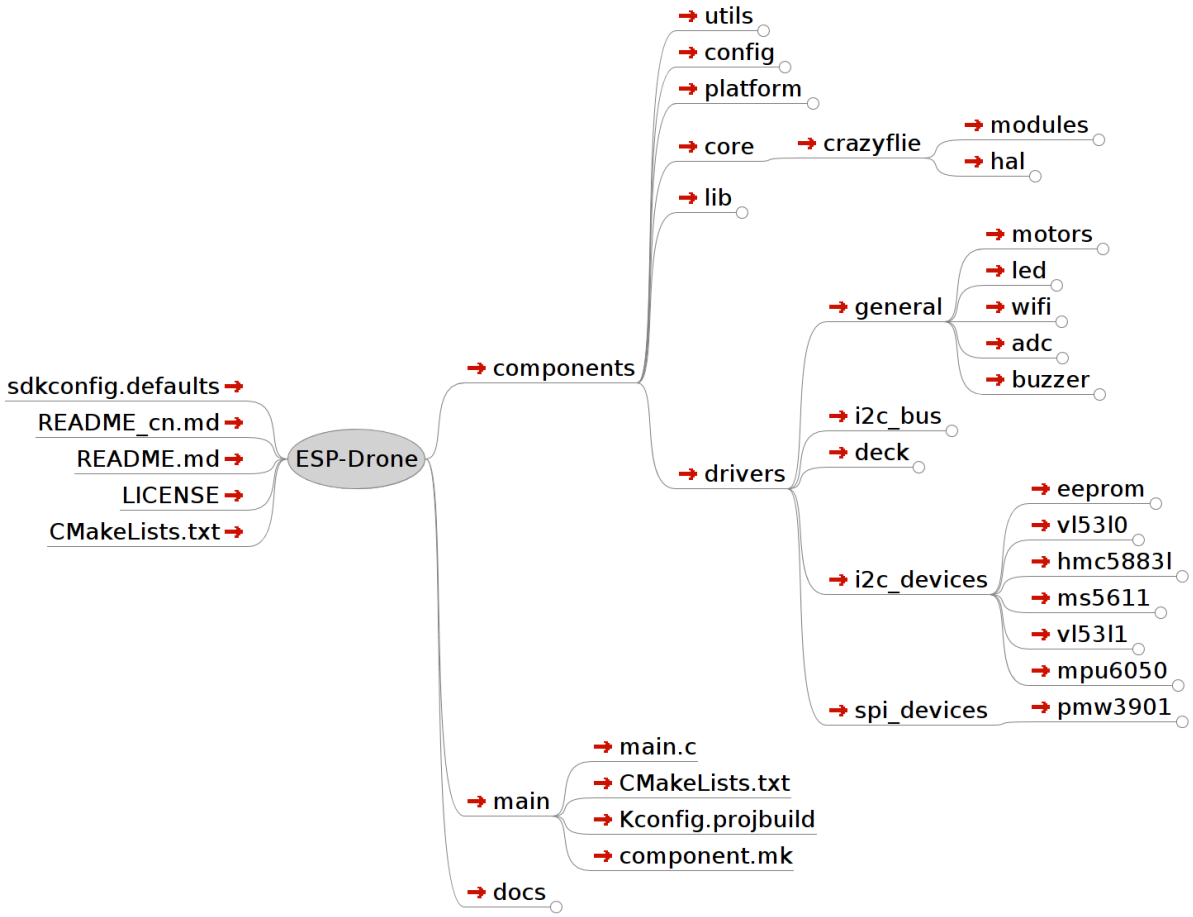
Beta code, currently in the GitHub repository, is available by using git:

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

The project software mainly consists of a flight control kernel, hardware drivers, and dependency libraries:

- The flight control kernel is from Crazyflie, mainly including hardware abstraction layer and flight control program.
- Hardware drivers are structured in files according to hardware interfaces, including I2C devcies and SPI devices.
- Dependency libraries include the default components provided by ESP-IDF, as well as DSP from third parties.

The code file structure is as follows:



```
.
├── components                          | project components directory
│   ├── config                                | system task configuration
│   │   └── include
│   ├── core                                   | system kernel directory
│   │   └── crazyflie                    | Crazyflie kernel
│   │       ├── hal                           | hardware abstraction code
│   │       └── modules                |  flight control code
│   ├── drivers                               | hardware driver directory
│   │   ├── deck                            | hardware extention interface driver
│   │   ├── general                        | general device directory
│   │   │   ├── adc                         | ADC driver for voltage monitoring
│   │   │   ├── buzzer                    | buzzer driver for status feedback
│   │   │   ├── led                           | LED driver for status feedback
│   │   │   ├── motors                   | motor driver for thrust output
│   │   │   └── wifi                          | Wi-Fi driver for communication
│   │   ├── i2c_bus                        | I2C driver
│   │   ├── i2c_devices                 | I2C device directory
│   │   │   ├── eeprom                 | EEPROM driver for parameter storage
│   │   │   ├── hmc5883l               | HMC5883l magnetic compass sensor
│   │   │   ├── mpu6050                | MPU6050 gyroscopic accelerometer sensor
│   │   │   ├── ms5611                   | MS5611 air pressure sensor
│   │   │   ├── vl53l0                      | Vl53l0 laser sensor (maximum distance 2 m)
│   │   │   └── vl53l1                      |  Vl53l1 laser sensor（maximum distance 4 m）
│   │   └── spi_devices                 | SPI devices directory
│   │       └── pmw3901                 | PMW3901 optical flow sensor
│   ├── lib                                            | external repository directory
│   │   └── dsp_lib                          | DSP repository
│   ├── platform                               | support multi-platform
│   └── utils                                        | utility function directory
├── CMakeLists.txt                          | utility function
├── LICENSE                                       | open source protocol
├── main                                             | entry function
├── README.md                                | project description
└── sdkconfig.defaults                    | default parameter
```

For more information, please refer to: espdrone_file_structure.

3.2.3 Source Code Style

Two ways to search the same area（union)

The same memory area can be searched in two ways:

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

Counting using enumeration types

The first member in an enumeration defaults to 0, so the member SensorImplementation_COUNT can always represent the total number of defined enumeration members before it.

```
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;
```

Packed Data Type

```
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));
```

The purpose of `__attribute__ ((packed))` is to disable the optimized alignment when compiling `struct`. By such way, `struct` is aligned based on its actual bytes. This is a syntax specific to GCC, which has nothing to do with your operating system but has to do with compiler. In Windows operating system, GCC and VC compiler do not support packed mode while TC compiler supports such mode.

```
In TC：struct my{ char ch; int a;} sizeof(int)=2;sizeof(my)=3; (compact mode)
In GCC：struct my{ char ch; int a;} sizeof(int)=4;sizeof(my)=8; (non-compact mode)
In GCC：struct my{ char ch; int a;}__attrubte__ ((packed)) sizeof(int)=4;sizeof(my)=5
```

3.3 Develop Guide

[??]

3.3.1 Drivers

[??]

This section covers I2C drivers and SPI drivers used in ESP-Drone.

I2C Drivers

I2C drivers include MPU6050 sensor driver and VL53LXX sensor driver. The following part describes the two sensors in their main features, key registers and programming notes, etc.

MPU6050 Sensor

Overview

The MPU6050 is a 6-axis Motion Tracking device that combines a 3-axis gyroscope, a 3-axis accelerometer, and a Digital Motion Processor (DMP).

How It Works

- Gyroscope: vibration occurs due to Coriolis effect when the gyroscope rotates around any sensing axis. Such vibration can be detected by a capacitive sensor. This capacitive sensor can amplify, demodulate, and filter such vibration signal, then generate a voltage proportional to the angular velocity.
- Accelerometer: when displacement accelerates along a specific axis over the corresponding detection mass, the capacitive sensor detects a change in capacitance.

Measure Range

- Gyroscope full-scale range: ±250 °/sec, ±500 °/sec, ±1000 °/sec, ±2000 °/sec
- Accelerometer full-scale range: ±2 g, ±4 g, ±8 g, ±16 g

AUX I2C Interface

- MPU6050 has an auxiliary I2C bus for communication with external 3-Axis magnetometer or other sensors.
- The AUX I2C interface supports two operation modes: I2C Master mode or Pass-Through mode.

MPU6050 FIFO

The MPU6050 contains a 1024-byte FIFO register that is accessible via the Serial Interface. The FIFO configuration register determines which data is written into the FIFO. Possible choices include gyroscope data, accelerometer data, temperature readings, auxiliary sensor readings, and FSYNC input.

Digital Low-Pass Filter (DLPF)

The MPU6050 has its own low-pass filter. Users can configure *Register 26 CONFIG - Configure DLPF* to control bandwidth, to lower high-frequency interference, but the configuration may reduce sensor input rate. Enable the DLPF: accelerometer outputs 1 kHz signal; disable the DLPF: accelerometer outputs 8 kHz signal.

Frame Synchronization Sampling Pin (FSYNC)

The field EXT_SYNC_SET in Register 26 is used to configure the external Frame Synchronization (FSYNC) pin sampling.

Digital Motion Processing (DMP)

- The MPU6050 has its own DMP inside, which can be used to calculate quaternion to offload processing power from the main CPU.
- The DMP can trigger an interrupt via a pin.

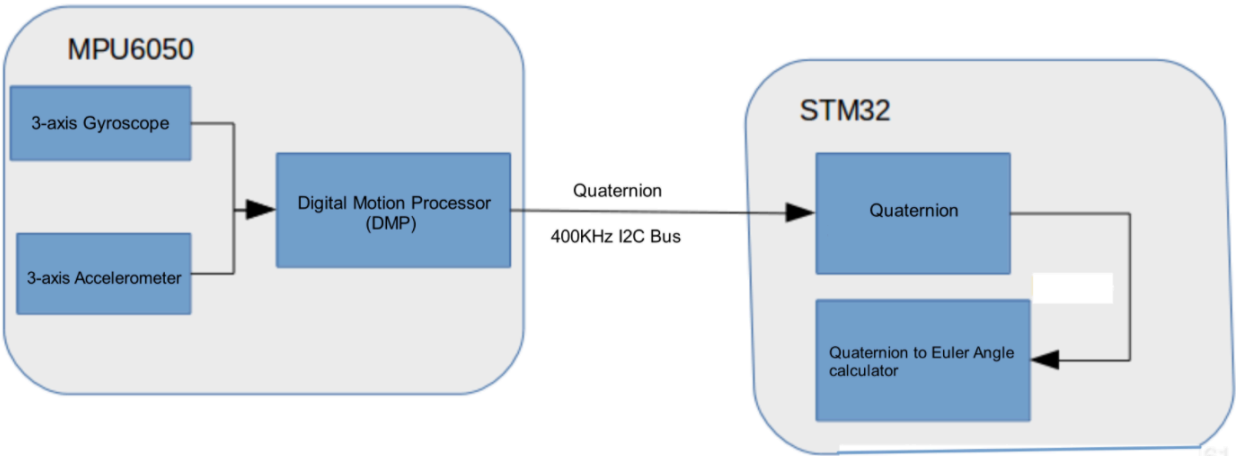


Fig. 15: MPU6050 DMP

MPU6050 Orientation

The MPU6050 defines its orientation as follows.

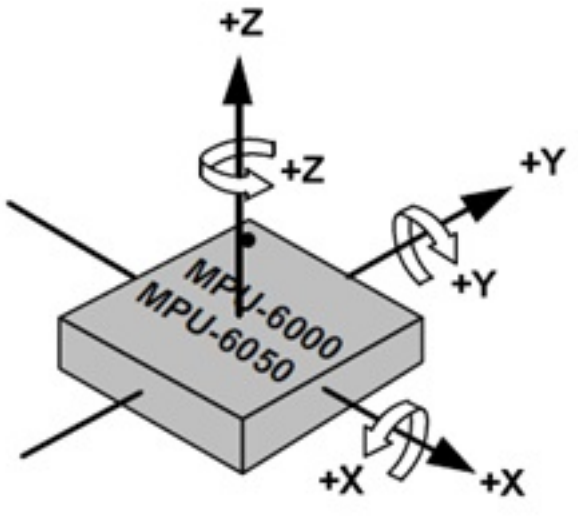


Fig. 16: MPU6050 Axis of X, Y, Z

MPU6050 Initialization

1. Restore register defaults:
   - Set PWR_MGMT_1 bit7 to 1,
   - and then to 0 after register defaults are restored.
   - Bit6 is automatically set to 1, putting MPU6050 into sleep mode;
2. Set PWR_MGMT_1 bit6 to 0, to wake up the sensor;
3. Set the clock source;
4. Set the range for the gyroscope and accelerometer;
5. Set the sample rate;
6. Set Digital Low-Pass Filter (optional).

MPU6050 Key Registers

Typical Register Values

Register	Typical Value	Feature
PWR_MGMT_1	0x00	Normal Enable
SMPLRT_DIV	0x07	Gyroscope sample rate: 125 Hz
CONFIG	0x06	DLPF filter frequency: 5 Hz
GYRO_CONFIG	0x18	Gyroscope does not conduct self-test, and its full-scale output is ±2000 °/s
ACCEL_CONFIG	0x01	Accelerometer does not conduct self-test, and its full-scale output is ±2 g.

Register 117 WHO_AM_I - Device Address

Bits [6:1] store the device address, and default to 0x68. The value of the AD0 pin is not reflected in this register.

Type: Read Only

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

Register 107 PWR_MGMT_1 - Power Management 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: if this bit is set, the register will use the default value.
- SLEEP: if this bit is set, MPU6050 will be put into sleep mode.
- CYCLE: when this bit (CYCLE) is set to 1 while SLEEP is disabled, the MPU6050 will be put into Cycle mode. In Cycle mode, the MPU6050 cycles between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by LP_WAKE_CTRL (Register 108).

Register 26 CONFIG - Configure DLPF

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]		

The DLPF is configured by DLPF_CFG. The accelerometer and gyroscope are filtered according to the value of DLPF_CFG as shown in the table below.

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

Register 27 - GYRO_CONFIG - Configure Gyroscope's Full-Scale Range

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: if this bit is set, X-axis gyroscope performs self-test.
- FS_SEL: select the full scale of the gyroscope, see the table below for details.

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

Register 28 ACCEL_CONFIG - Configure Accelerometer's Full-Scale Range

Users can configure the full-scale range for the accelerometer according to the table below.

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*

Register 25 SMPRT_DIV - Sample Rate Divider

The register specifies the divider from the gyroscope output rate used to generate the Sample Rate for the MPU6050. The sensor register output, FIFO output, and DMP sampling are all based on the Sample Rate. The Sample Rate is generated by dividing the gyroscope output rate by (1 + SMPLRT_DIV).

Type: Read/Write

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

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

Gyroscope Output Rate = 8 kHz when the DLPF is disabled (DLPF_CFG = 0 0r 7), and 1 kHz when the DLPF is enabled (see Register 26). Note: if SMPLRT_DIV is set to 7 when the DLPF is disabled, the chip can generate an interrupt signal of 1 kHz.

Registers 59 ~ 64 - Accelerometer Measurements

- Save data in big-endian: higher data bits are stored at low address, and lower bits at high address.
- Save data in complement: the measurement value is a signed integer, so it can be stored in complement.

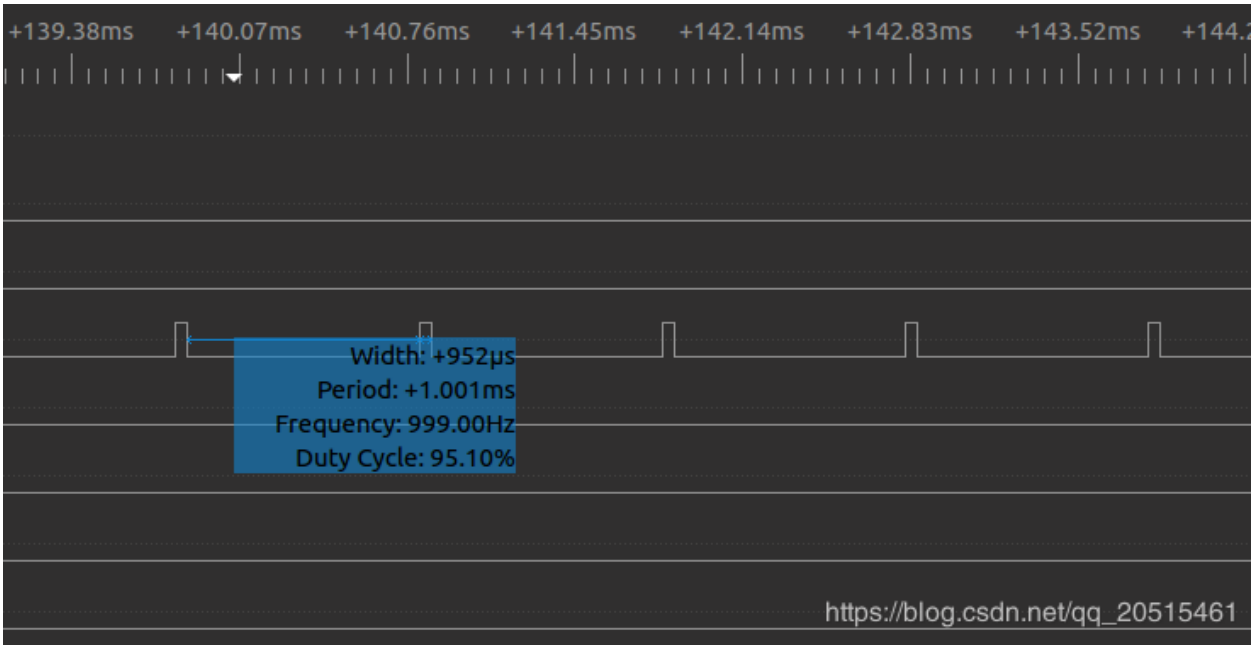
Registers 65 ~ 66 - Temperature Measurement

Registers 67 ~ 72 - Gyroscope Measurements

VL53LXX Sensor

Overview

VL53L1X is a Time-of-Flight ranging and gesture detection sensor provided by ST.



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]							

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]							

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]							

How It Works

The VL53L0X/VL53L1X chip is internally integrated with a laser transmitter and a SPAD infrared receiver. By detecting the time difference between photon sending and receiving, the chip calculates the flight distance of photons, and the maximum measuring distance can reach two meters, which is suitable for short- and medium-range measurement applications.

Fig. 17: VL53LXX

Measurement Area - Region-of-Interest (ROI)

The VL53L0X/VL53L1X measures the shortest distance in the measurement area, which can be zoomed in or out depending on actual scenario. But a large detection range may cause fluctuations in the measurement.

For more information about configuration on measurement area, see the sections 2.4 Ranging Description and 2.8 Sensing Array Optical Center in VL53LXX Datasheet.

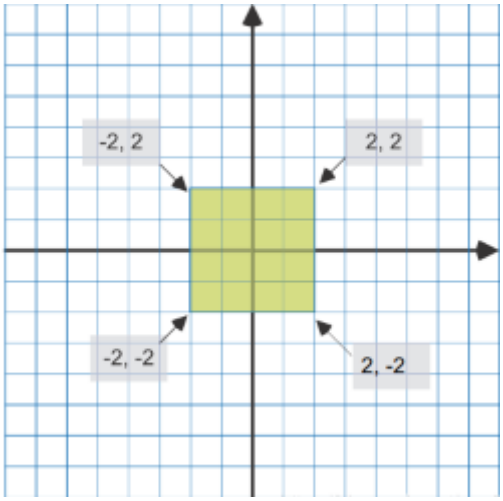


Fig. 18: ROI

Measure Distance

- The VL53L0X sensor has a blind spot of **3 ~ 4 cm**, with an effective measurement range of 3 ~ 200 cm and an accuracy of ±3%.
- The VL53L1X is an upgraded version of the VL53L0X with a detection distance of up to 400 cm.

Precision Mode	Measuring Time (ms)	Measuring Range (m)	Applications
Default	30	1.2	Standard
High Precision	200	1.2 <+-3%	Precise distance measurement
Long Range	33	2	Long distance, only works in dark environment
High Speed	20	1.2 +-5%	Speed priority with low precision

- VL53LXX measurement distance/performance is related to the environment. The detection distance is farther in the dark conditions. But in outdoor bright conditions, the laser sensor may be subject to a lot of interference, resulting in reduced measurement accuracy. For such reason, the height should be set based on the outdoor air pressure.

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.

Measurement Frequency

- The VL53L0X ranging frequency is up to 50 Hz, with a measurement error of ±5%.
- The VL53L1X I2C has a maximum clock frequency of 400 kHz, and the pull-up resistor needs to be selected based on voltage and bus capacitance values. For more information, see VL53LXX Datasheet.
- XSHUT, input pin for mode selection (sleep), must always be driven to avoid leakage current. A pull-up resistor is needed.
- GPIO1, interrupt output pin for measuring dataready interrupts.

Work Mode

By setting the level of the XSHUT pin, the sensor can be switched into HW Standby mode or SW Standby mode for conditional boot and for lowering the standby power consumption. If the host gives up the management of sensor modes, the XSHUT pin can be set to pull up by default.

- HW Standby: XSHUT pulls down and the sensor power is off.
- SW Standby: XSHUT pulls up, then the sensor enters Boot and SW Standby modes.

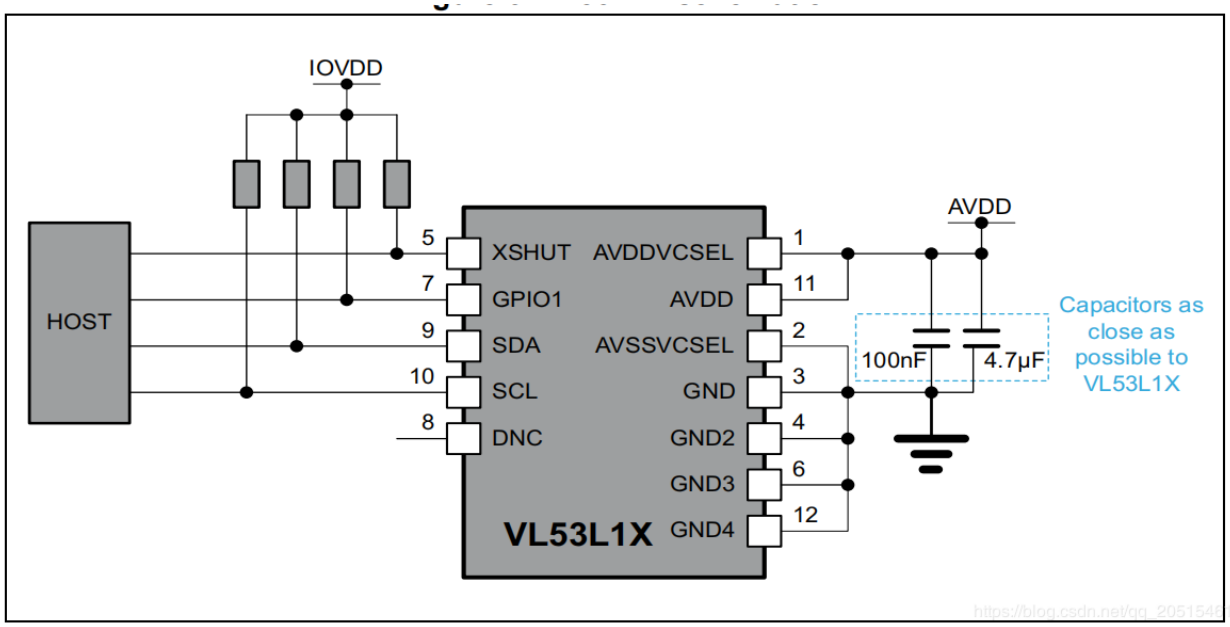


Fig. 19: VL53L1X

Figure 7. Power up and boot sequence

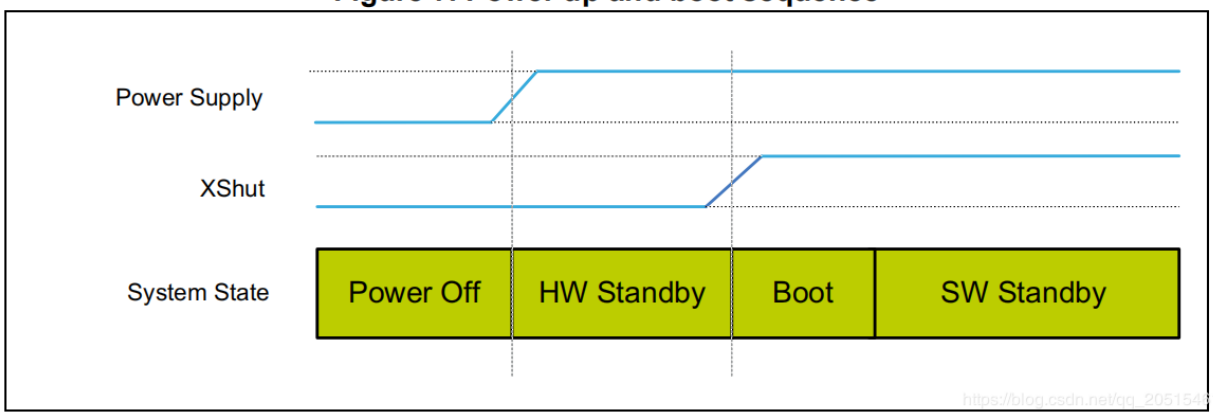
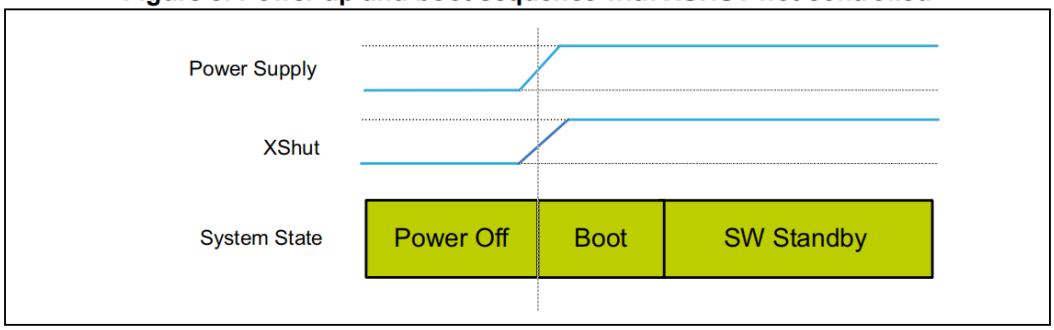


Fig. 20: HW Standby

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.

Fig. 21: SW Standby

VL53LXX Initialization

1. Wait for the initialization of the hardware to complete
2. Data initialization
3. Static initialization, loading data
4. Set ranging mode
5. Set the maximum wait time for a single measurement
6. Set the measurement frequency (interval)
7. Set measurement area ROI (optional)
8. Start the measurement

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

    Roi0.TopLeftX = 0;    //Measurement target area (optional). Min: 4*4, Max: 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); // Wait for the initialization of the␣
↪hardware to complete
    status = VL53L1_DataInit(Dev); // Data initialization, is executed immediately␣
↪after wake up;
    status = VL53L1_StaticInit(Dev); // Static initialization, loading parameters.
    status = VL53L1_SetDistanceMode(Dev, VL53L1_DISTANCEMODE_LONG);// Set ranging␣
↪mode;
    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(Dev, 50000); // Set the␣
↪maximum wait time based on the measurement mode.
    status = VL53L1_SetInterMeasurementPeriodMilliSeconds(Dev, 100); // Set the␣
↪measurement interval.
```

(continues on next page)

(continued from previous page)

```
    status = VL53L1_SetUserROI(Dev, &Roi0); //Set measurement area ROI
    status = VL53L1_StartMeasurement(Dev); //Start the measurement
    if(status) {
        printf("VL53L1_StartMeasurement failed \n");
        while(1);
    }

}
```

Note: except the step VL53L1_SetUserROI, the other initialization steps can not be skipped.

VL53LXX Ranging Step

The ranging can be done in two kinds of modes: polling mode and interrupt mode.

Polling Mode Workflow

Note:

- Each time when the measurement and value reading are done, clear the interrupt flag using `VL53L1_ClearInterruptAndStartMeasurement`, and then start the measurement again.
- Polling mode provides two methods as shown in the figure above: Drivers polloing mode and Host polling mode. We take the Drivers polling mode as an example.

```
/* 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); // Waiting for the␣
↪measurement result
                if(!status) {
                    status = VL53L1_GetRangingMeasurementData(Dev, &RangingData); //␣
↪Get a single measurement data
                    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);
```

(continues on next page)

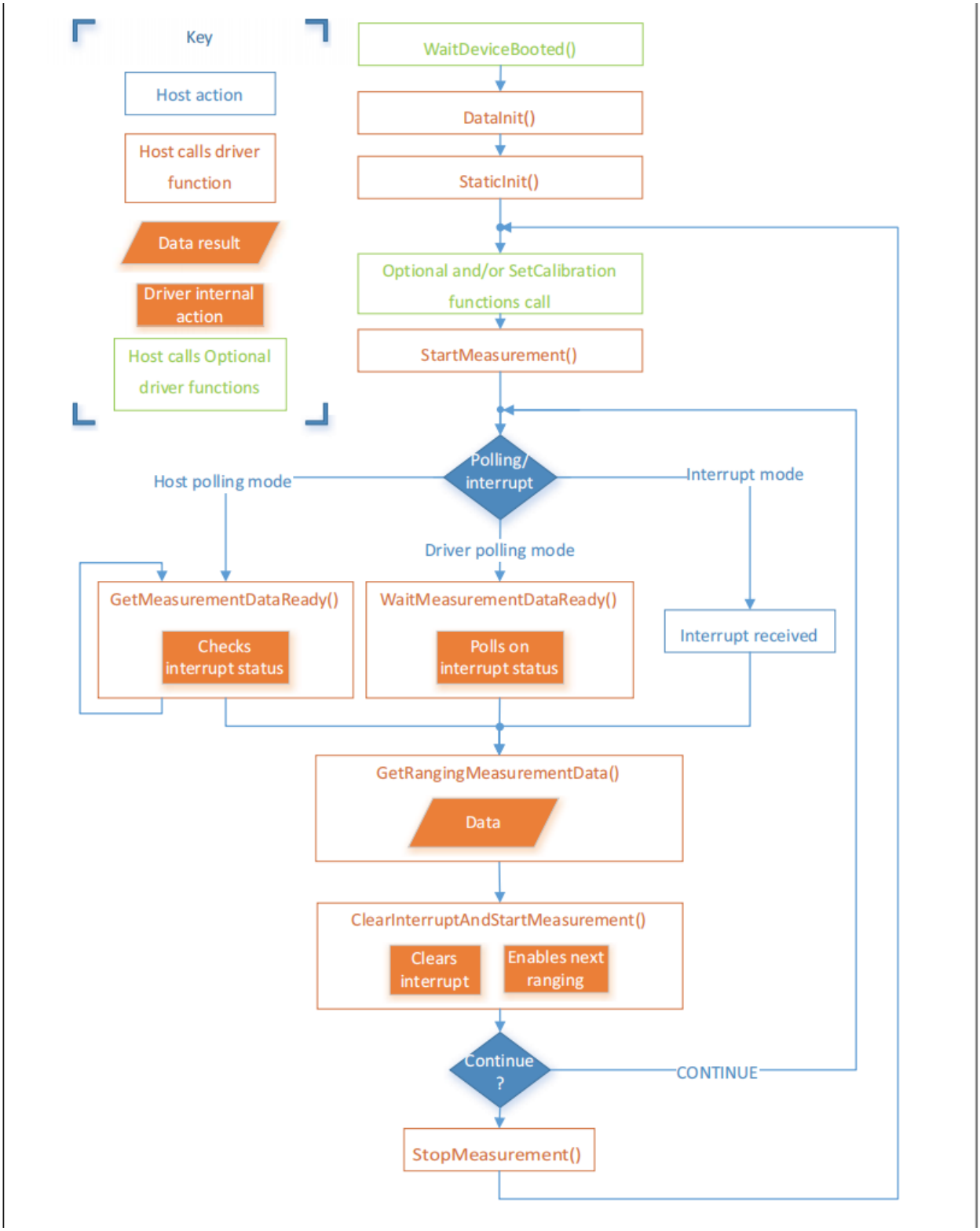


Fig. 22: VL53LXX Measurement Sequence

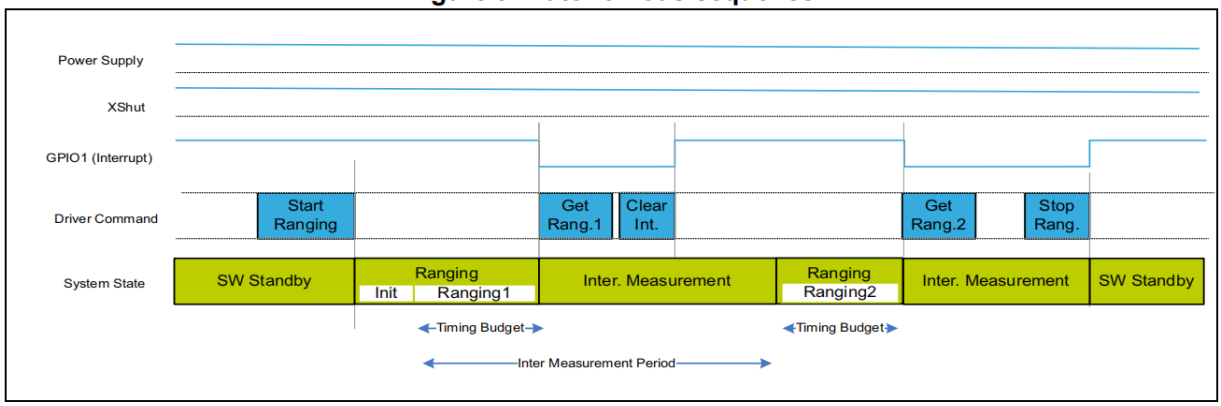
(continued from previous page)

```
                    } else {
                        status = VL53L1_SetUserROI(Dev, &Roi0);
                    }
                    status = VL53L1_ClearInterruptAndStartMeasurement(Dev); //␣
↪Release the interrupt
                }
            }
        while (1);
    }
    //  return status;
}
```

Interrupt Mode Workflow

Interrupt is controlled by the pin GPIO1. When the data is ready, pulling down GPIO1 can inform the host to read the data.

Figure 9. Autonomous sequence



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

Fig. 23: VL53LXX Autonomous Sequence

VL53LXX Sensor Calibration

If a mask is installed over the sensor receiver, or if the sensor is mounted behind a transparent cover, the sensor needs to be calibrated due to changes in the transmission rate. You can write a calibration program and call APIs based on the calibration process, or you can measure the calibration value directly using the official PC GUI.

Use Official APIs to Write Calibration Program

Calibration process is shown below. The order of calls should be exactly the same.

```
/* VL53L1 Module Calibration*/
static VL53L1_CalibrationData_t vl53l1_calibration(VL53L1_Dev_t *dev)
{
    int status;
    int32_t targetDistanceMilliMeter = 703;
    VL53L1_CalibrationData_t calibrationData;
    status = VL53L1_WaitDeviceBooted(dev);
```

(continues on next page)

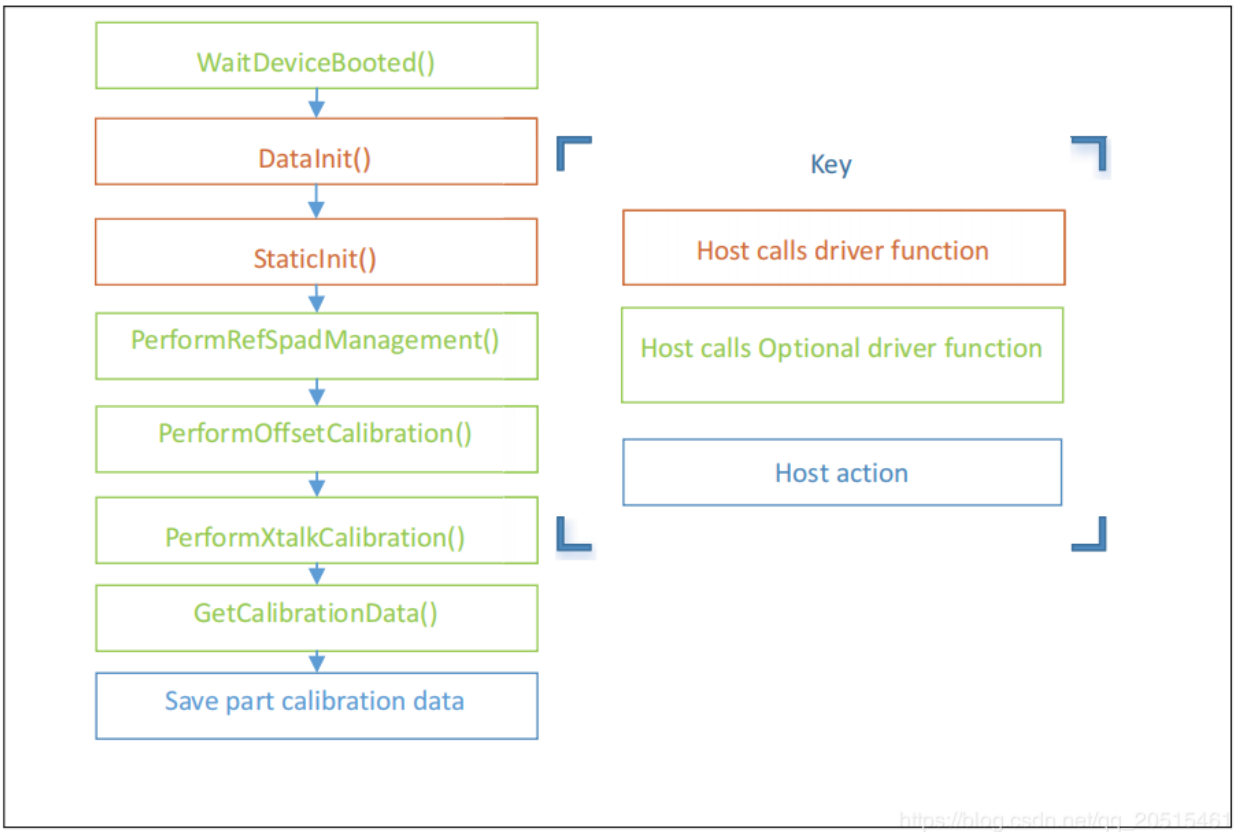


Fig. 24: VL53LXX Calibration Sequence

(continued from previous page)

```
    status = VL53L1_DataInit(dev);                                          // Device␣
↪initialization
    status = VL53L1_StaticInit(dev);                                        // Load␣
↪device settings for a given use case.
    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");
        calibrationData.struct_version = 0;
        return calibrationData;

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

}
```

Use the Official PC GUI to Calibrate the Sensor

STM has provided a PC GUI to configure and calibrate the sensor.

- Connect the sensor to `STM32F401RE nucleo` development board provided by STM.
- Use softwere to conduct calibration, and get the reference value.
- Enter this value during initialization.

For more information, see STSW-IMG008: Windows Graphical User Interface (GUI) for VL53L1X Nucleo packs. Works with P-NUCLEO-53L1A1.

VL53L1X Example

Example Description

- Implementation: VL53L1X detects the height changes (last for 1 second) and the red light turns on. Height returns to normal value (last for 1 second), and the green light turns on.
- Parameters: set the I2C number, port number, LED port number via `make menuconfig`.
- Example analysis can be found in code notes and user manual.

Notes

- This example applies only to VL53L1X, not to its older version hardware VL53L0X.
- According to STM document, the measurement distance is 400 cm in dark environment. In indoor environment, the measurement distance can be 10 to 260 cm.
- Some of the parameters in the initialization function `vl53l1\_init (VL53L1\_Dev\_t \*)` need to be determined according to the actual usage environment, and there is room for optimization.
- Make sure the sensor is installed right above the detection position.

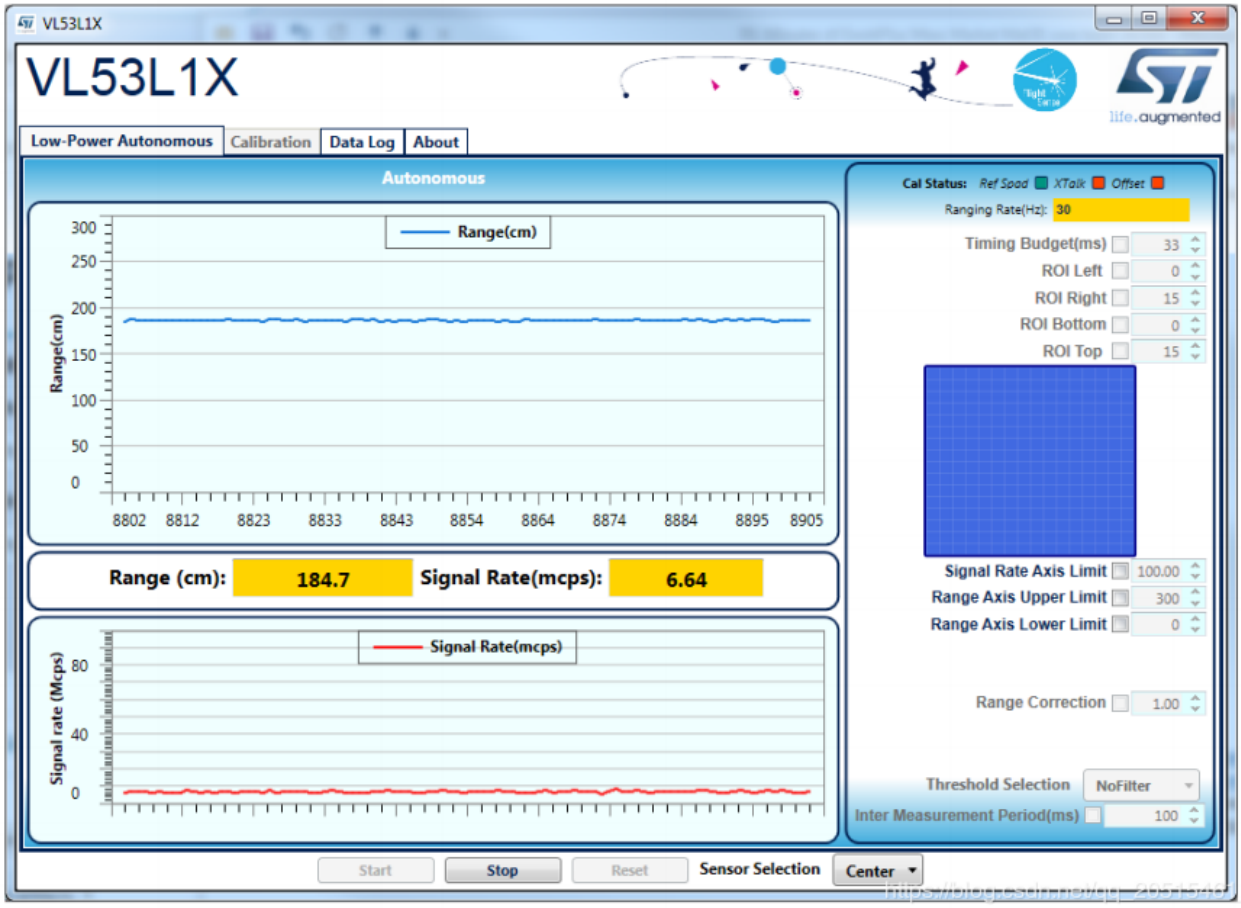


Fig. 25: PC GUI Calibration

- Base height is automatically calibrated when the module is powered on. If the base height changes, the parameters need to be reset again.

Example Repository

Click esp32-vl53l1x-test to check the example, or download the example using git tool:

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

SPI Driver

PMW3901 Sensor

Overview

The PMW3901 is PixArt's latest high-precision, low-power optical navigation module that provides X-Y motion information with a wide range of 8 cm to infinity. The PWM3901 works with an operating current of less than 9 mA, an operating voltage of VDD (1.8 to 2.1 V), a VDDIO (1.8 to 3.6 V), and uses a 4-wire SPI interface for communication.

Main Parameters

Parameter	Value
Supply voltage (V)	VDD: 1.8 ~ 2.1 V; VDDIO: 1.8 ~ 3.6 V
Working range (mm)	80 ~ +∞
Interface	4-line SPI @ 2 MHz
Package	28-pin COB package, size: 6 x 6 x 2.28 mm

Package and Pin Layout

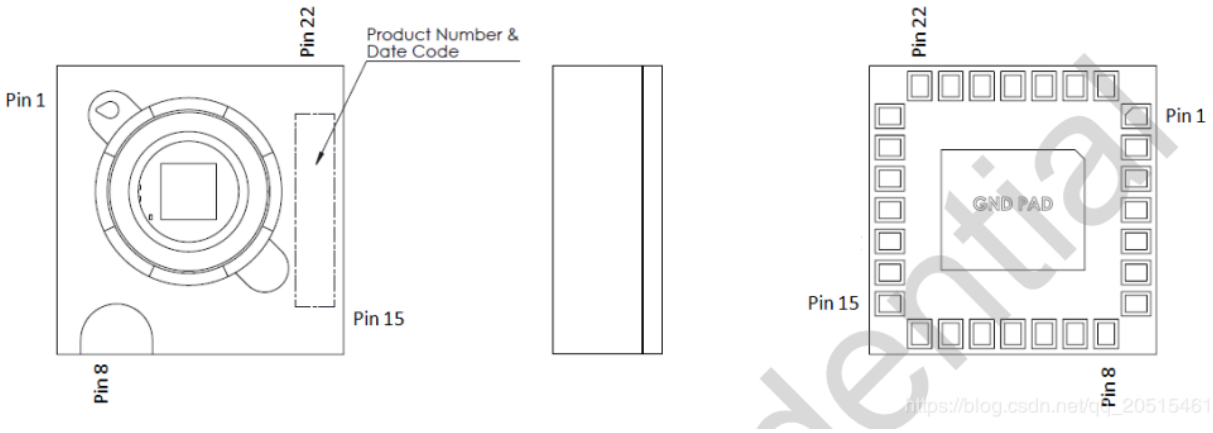


Fig. 26: PMW3901 Package

The sensor works at a low operating voltage and to communicates with the ESP32 at 3.3 V requires different voltages from VDD and VDDIO.

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

Fig. 27: PMW3901 Pinmap

Power States and Sequence

Power-Up Sequence

Although PMW3901MB performs an internal power up self-reset, it is still recommended that the Power_Up_Reset register is written every time power is applied. The appropriate sequence is as follows:

1. Apply power to VDDIO first and followed by VDD, with a delay of no more than 100 ms in between each supply. Ensure all supplies are stable.
2. Wait for at least 40 ms.
3. Drive NCS high, and then low to reset the SPI port.
4. Write 0x5A to Power_Up_Reset register (or alternatively, toggle the NRESET pin).
5. Wait for at least 1 ms.
6. Read from registers 0x02, 0x03, 0x04, 0x05, and 0x06 one time regardless of the motion pin state.
7. Refer to PWM3901MB Datasheet Section 8.2 Performance Optimization Registers to configure the required registers to achieve optimum performance of the chip.

Power-Down Sequence

PMW3901MB can be set to Shutdown mode by writing to Shutdown register. The SPI port should not be accessed when Shutdown mode is asserted, except the power-up command (writing 0x5A to register 0x3A). Other ICs on the same SPI bus can be accessed, as long as the chip's NCS pin is not asserted.

To de-assert Shutdown mode:

1. Drive NCS high, and then low to reset the SPI port.

2. Write 0x5A to Power_Up_Reset register (or alternatively, toggle the NRESET pin).
3. Wait for at least 1 ms.
4. Read from registers 0x02, 0x03, 0x04, 0x05, and 0x06 one time regardless of the motion pin state.
5. Refer to PWM3901MB Datasheet Section 8.2 Performance Optimization Registers to configure the required registers for optimal chip performance.

For more information, see Connected Home Appliances and IoT.

Code Interpretation

Key Structs

```
typedef struct opFlow_s
{
    float pixSum[2]; /*accumulated pixels*/
    float pixComp[2]; /*pixel compensation*/
    float pixValid[2]; /*valid pixels*/
    float pixValidLast[2]; /*last valid pixel*/
    float deltaPos[2]; /*displacement between 2 frames, unit: cm*/
    float deltaVel[2]; /*velocity, unit: cm/s*/
    float posSum[2]; /*accumulated displacement, unit: cm*/
    float velLpf[2]; /*low-pass velocity, unit cm/s*/
    bool isOpFlowOk; /*optical flow*/
    bool isDataValid; /* valid data */
} opFlow_t;
```

- Accumulated pixels: accumulated pixels after the drone takes off.
- Pixel compensation: compensation for pixel error caused by drone tilt.
- Valid pixels: the actual pixels that have been compensated.
- Displacement between 2 frames: the actual displacement converted from pixels, unit: cm.
- Velocity: instantaneous velocity, obtained by differentiating on displacement changes, unit: cm/s.
- Accumulated displacement: actual displacement, unit: cm.
- Low-pass velocity: low-pass operation on velocity increases data smoothness.
- Optical flow status: check if this optical flow sensor is working properly.
- Valid data: data is valid at a certain height.

```
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;
    };
  };

  uint8_t observation;
```

(continues on next page)

(continued from previous page)

```
  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: motion information, can be obtained according to the bits frame detection (frameFrom0), operating mode (runMode) and motion detection (motionOccured).
- observation: to check if the IC has EFT/B or ESD issues. When the sensor works properly, the value should be 0xBF.
- deltaX and deltaY: the sensor has detected the motion of the image at X and Y directions.
- squal: the quality of motion information, i.e, the credibility of motion information;
- rawDataSum: the sum of raw data, can be used to average the data of a frame.
- maxRawData and minRawData: the maximum and minimum of the raw data;
- shutter: a real-time auto-adjusted value, to ensure that the average motion data remains within the normal operating range. Shutter can be used together with `squal` to check whether the motion information is available.

Programming Notes

- If the sensor data is 0 and lasts for 1 second, it indicates that an error occurs. If so, optical flow tasks should be suspended.
- The sensor lens must be mounted facing down. Due to relative motion, the displacement data collected by the sensor is opposite to the actual motion direction of the drone.
- Enable position-hold mode only when the height-hold mode test is stable. Accurate height information is used to determine the relation between image pixels and actual distance.
- Manual test on tilt compensation can remain the sensor output nearly unchanged even when the drone tilts on certain direction.
- With tilt compensation and motion accumulated pixels, the actual accumulated pixels can be obtained. After several calculations, you can get:
  - Pixel changes between 2 frames = actual accumulated pixels - last actual pixels;
  - Displacement changes between 2 frames = Pixel changes between 2 frames x coefficient. Note when the height is less than 5 cm, the optical flow will stop working, so the coefficient should be set to 0;
  - Integrate on the above displacement changes, to obtain the displacement from the four axes to the take-off point. Differentiate on the above displacement changes, to obtain the instantaneous velocity. Conduct low-pass operation on velocity, to increase data smoothness. Limit amplitude on velocity, to enhance data security.
- The position and velocity information of the four axes are obtained through the optical flow, and then:
  - The above location and speed information together with the accelerometer (state_estimator.c) can be used to estimate the position and speed;

  - Estimated position and speed are involved in PID operations and can be used for horizontal position control. Refer to position_pid.c to see how the position ring and speed ring PID are handled.

Finally, horizontal position-hold control can be achieved through the above process.

3.3.2 Flight Control System

[??]

Startup Process

For source file, please check start_from_app_main.

Task Management

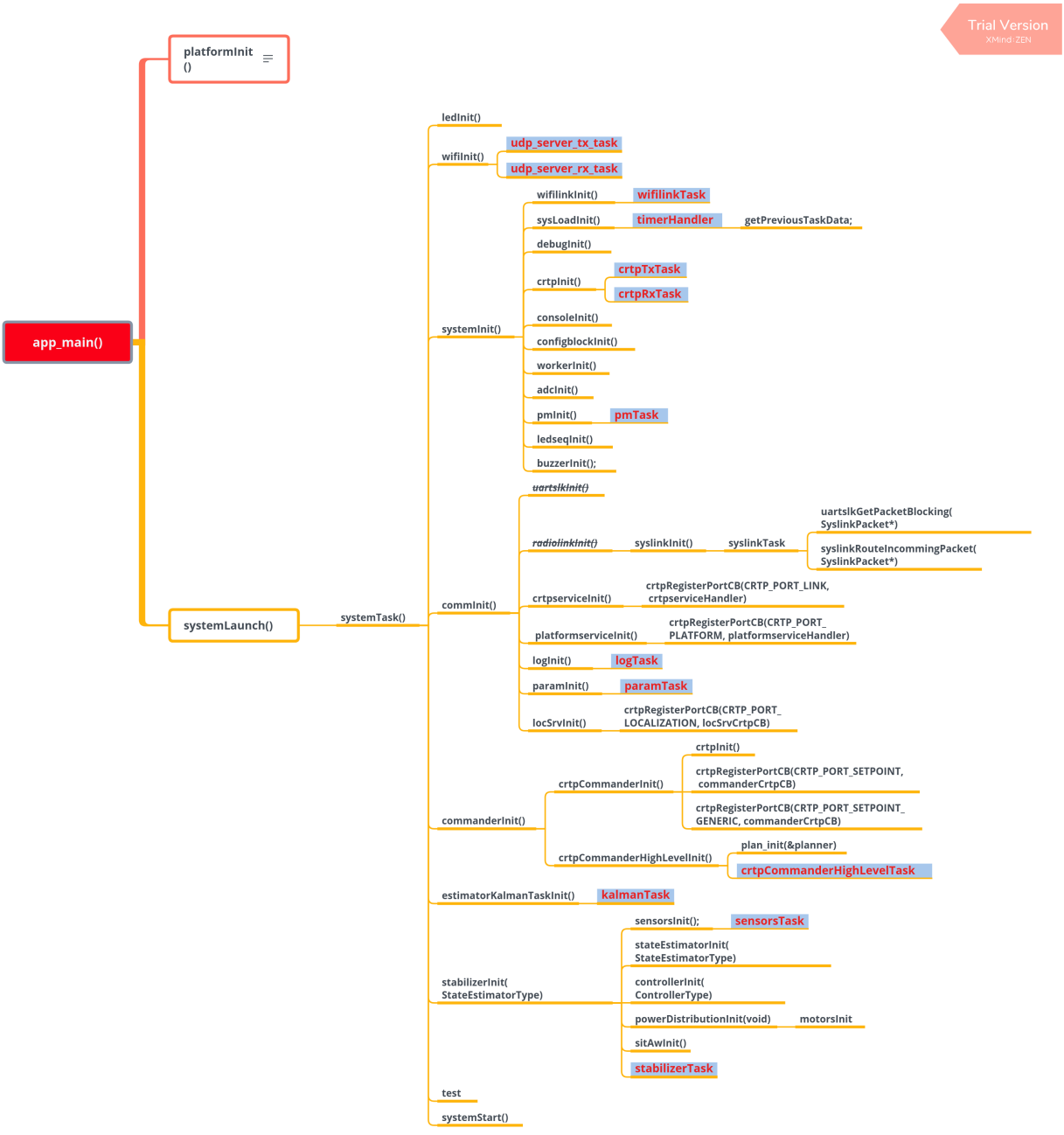
Tasks

The following TASKs are started when the system is operating properly.

- Load: CPU occupancy
- Stack Left: remaining stack space
- Name: task name
- PRI: task priority

TASKs are described as follows:

- PWRMGNT: monitor system voltage
- CMDHL: application layer - process advanced commands based on CRTP protocol
- CRTP-RX: protocol layer - decode CRTP flight protocol
- CRTP-TX: protocol layer - decode CRTP flight protocol
- UDP-RX: transport layer - receive UDP packet
- UDP-TX: Transport Layer - send UDP packet
- WIFILINK: work with CRTP protocol layer and UDP transport layer
- SENSORS: read and pre-process sensor data
- KALMAN: estimate the drone's status according to sensor data, including the drone's angle, angular velocity, and spatial position. This TASK consumes a large amount of CPU resources on the ESP chip and users should be careful about its priority allocation.
- PARAM: modify variables remotely according to CRTP protocol
- LOG: monitor variables on real-time according to CRTP protocol
- MEM: modify memory remotely according to CRTP protocol
- STABILIZER: self stabilize its thread, and control the process of flight control program
- SYSTEM: control system initialization and self-test process


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();
uartslkInit()
radiolinkInit()
syslinkInit()
syslinkTask
uartslkGetPacketBlocking(SyslinkPacket*)
syslinkRouteIncommingPacket(SyslinkPacket*)
commInit()
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()

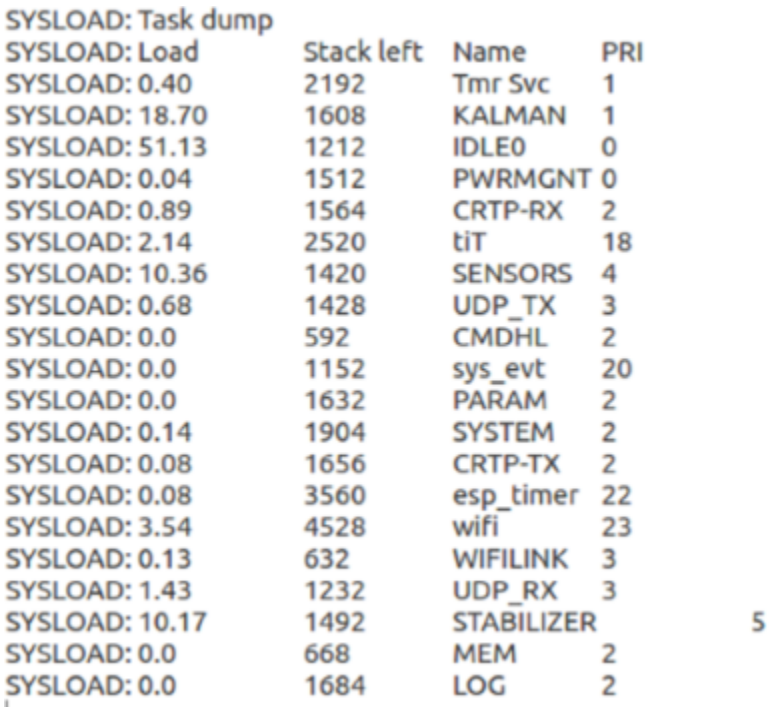


Fig. 28: Task Dump

Configure Task Stack Size

Users can modify the stack size in `components/config/include/config.h`, or modify `BASE_STACK_SIZE` in `menucfg`. When the target is `ESP32`, you can adjust the `BASE_STACK_SIZE` to 2048, to avoid memory overlapping. When the target is `ESP32-S2`, modify the value to `1024`.

```
//Task stack size
#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
#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
```

(continues on next page)

(continued from previous page)

```
#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
```

Configure Task Priority

The priority of system tasks can be configured in `components/config/include/config.h`. `ESP32`, for its dual-core advantage, has more computing resources than `ESP32-S2`, therefore, its time-consuming `KALMAN_TASK` can be set to a higher priority. But if the target is `ESP32-S2`, please lower the priority of the `KALMAN_TASK`, otherwise task watchdog will be triggered due to the failure of releasing enough CPU resources.

```
//Task priority: the higher the number, the higher the priority.
#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
#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
```

(continues on next page)

(continued from previous page)

```
  #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
```

Key Tasks

Except the system default enabled tasks, such as Wi-Fi TASK, the task with the highest priority is `STABILIZER_TASK`, highlighting the importance of this task. `STABILIZER_TASK` controls the entire process from sensor data reading, attitude calculation, target receiving, to final motor power output, and drives the algorithms at each stage.

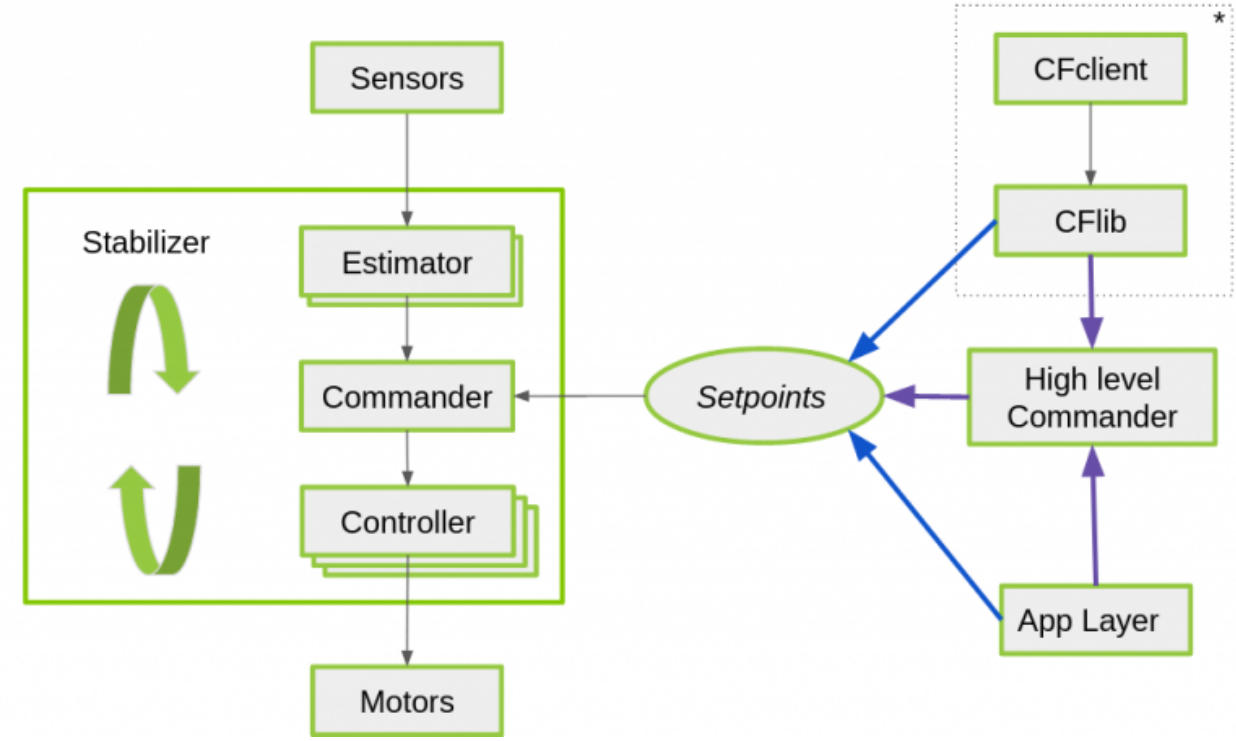


Fig. 29: stabilizerTask Process

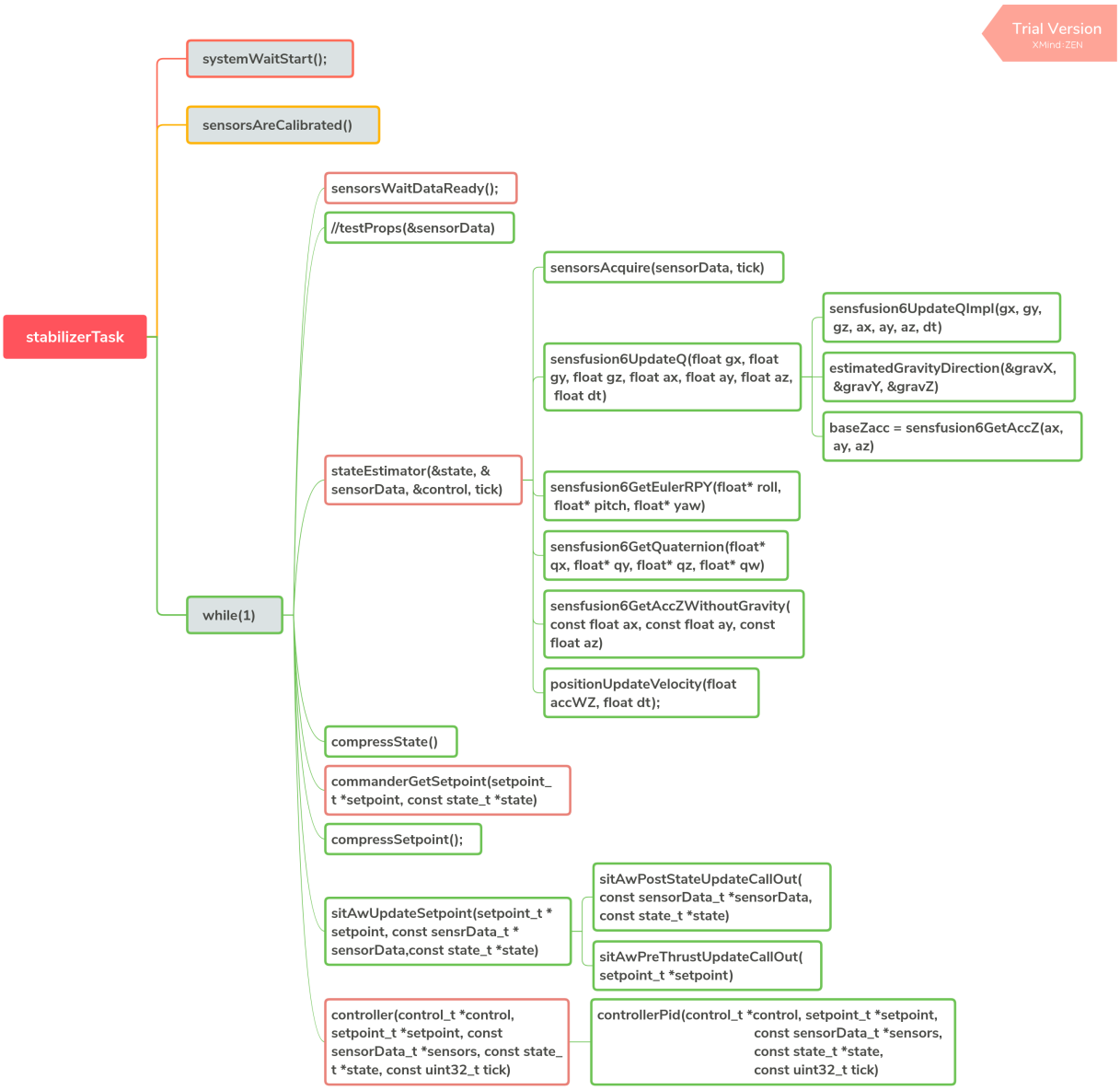


Fig. 30: stabilizerTask

Sensor Driver

The sensor driver code can be found in `components\drivers`. `drivers` applies a file structure similar to that used in esp-iot-solution. In such structure, drivers are classified by the bus they belong to, including `i2c_devices`, `spi_devices`, and `general`. For more information, please refer to Drivers.

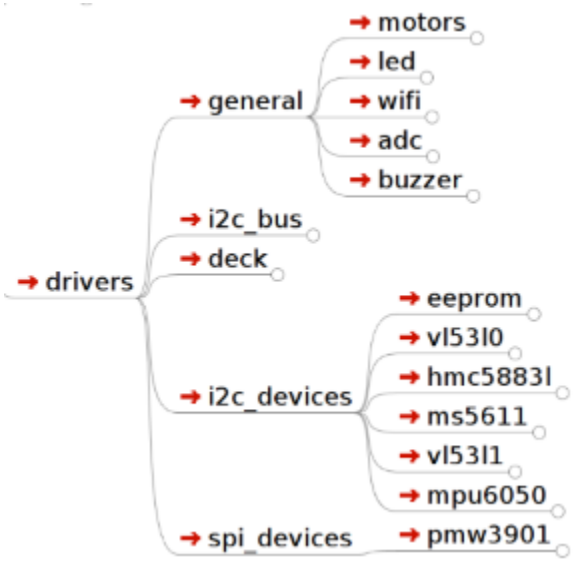


Fig. 31: Drivers File Structure

Sensor Hardware Abstraction

`components\core\crazyflie\hal\src\sensors.c` provides hardware abstraction for the sensors. Users are free to combine sensors to interact with upper-level application by implementing the sensor interfaces defined by the hardware abstraction layer.

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

The sensor abstraction interfaces implemented by ESP-Drone are listed in `components/core/crazyflie/hal/src/sensors_mpu6050_hm5883L_ms5611.c`, which can interact with the upper-level application by the following assignment process.

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

Sensor Calibration

Gyroscope Calibration

Due to large temperature drift, the gyroscope needs to be calibrated before each use, to calculate its reference values in current environment. ESP-Drone continues the gyroscope calibration scheme provided by Crazyflie 2.0. At the first power-up, the variance and average at the three axes of the gyroscope are calculated.

The detailed gyroscope calibration is as follows:

1. Store the latest 1024 sets of gyroscope measurements into a ring buffer with a maximum length of 1024.
2. Calculate the variance of the gyroscope output values, to check whether the drone is placed level and gyroscope is working properly.
3. If Step 2 is OK, calculate the average of the 1024 sets of the gyroscope output values as its calibration value.

Below is the source code for gyroscope base calculation:

```
/**
 * 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.
```

(continues on next page)

(continued from previous page)

```
 * 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;
}
```

Trim Gyroscope Output Values

```
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); // LPF Filter, to avoid␣
↪high-frequency interference
```

Accelerometer Calibration

Gravitational Acceleration (g) Calibration

The values of g are generally different at various latitudes and altitudes of the Earth, so an accelerometer is required to measure the actual g. The accelerometer calibration scheme provided by Crazyflie 2.0 can be your reference, and its g-value calibration process is as follows:

1. Once the gyroscope calibration is complete, the accelerometer calibration is performed immediately.
2. Store 200 sets of accelerometer measurements to the Buffer.
3. Calculate the value of g at rest by synthesizing the weight of g on three axes.

For more information, see g Values at Various Latitudes and Altitudes.

Calculate g values at rest

```
/**
 * 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;
}
```

Trim accelerometer measurements by the actual g values

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

Calibrate the Drone at Horizontal Level

Ideally, the accelerometer is installed horizontally on the drone, allowing the 0 position to be used as the drone's horizontal surface. However, due to the inevitable inclination of the accelerometer when installed, the flight control system can not estimate the horizontal position accurately, resulting in the drone flying in a certain direction. Therefore, a certain calibration strategy needs to be set to balance this error.

1. Place the drone on a horizontal surface, and calculate `cosRoll`, `sinRoll`, `cosPitch`, and `sinPitch`. Ideally, `cosRoll` and `cosPitch` are 1. `sinPitch` and `sinRoll` are 0. If the accelerometer is not installed horizontally, `sinPitch` and `sinRoll` are not 0. `cosRoll` and `cosPitch` are not 1.
2. Store the `cosRoll`, `sinRoll`, `cosPitch`, and `sinPitch` obtained in Step 1, or the corresponding `Roll` and `Pitch` to the drone for calibration.

Use the calibration value to trim accelerometer measurements:

```
/**
 * 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
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

The above process can be deduced via the resolution of a force and Pythagorean Theorem.

Attitude Calculation

Supported Attitude Calculation Algorithms

- Complementary filtering
- Kalman filtering

Attitude calculation used in ESP-Drone is from `Crazyflie`. ESP-Drone firmware has been tested for complementary filtering and Kalman filtering to efficiently calculate flight attitudes, including the angle, angular velocity, and spatial position, providing a reliable state input for the control system. Note that in position-hold mode, you must switch to Kalman filtering algorithm to ensure proper operation.

Crazyflie Status Estimation can be found in State estimation: To be or not to be!

Complementary Filtering

Kalman Filtering

Fight Control Algorithms

Supported Controller

The control system code used in ESP-Drone is from `Crazyflie`, and continues all its algorithms. Please note that ESP-Drone has only tested and tuned the parameters for PID controller. When using other controllers, tune your parameters while ensuring safety.

For more information, please refer to Out of Control.

In the code, you can modify the input parameters of `controllerInit(ControllerType controller)` to switch the controller.

Customized controllers can also be added by implementing the following controller interfaces.

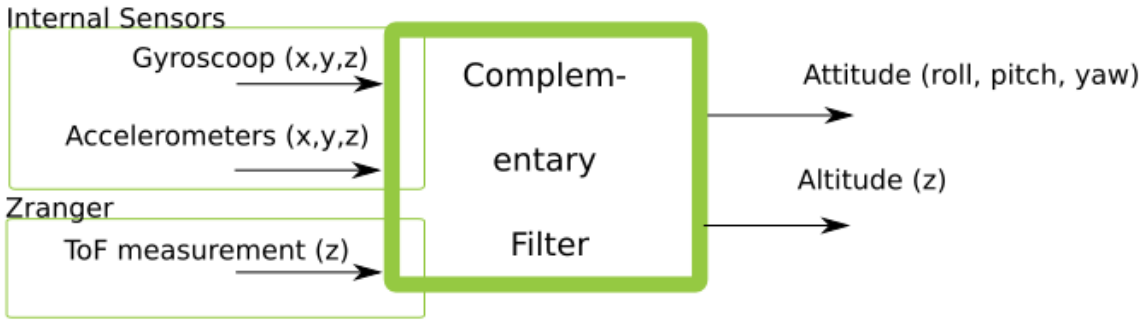


Fig. 32: Complementary Filtering

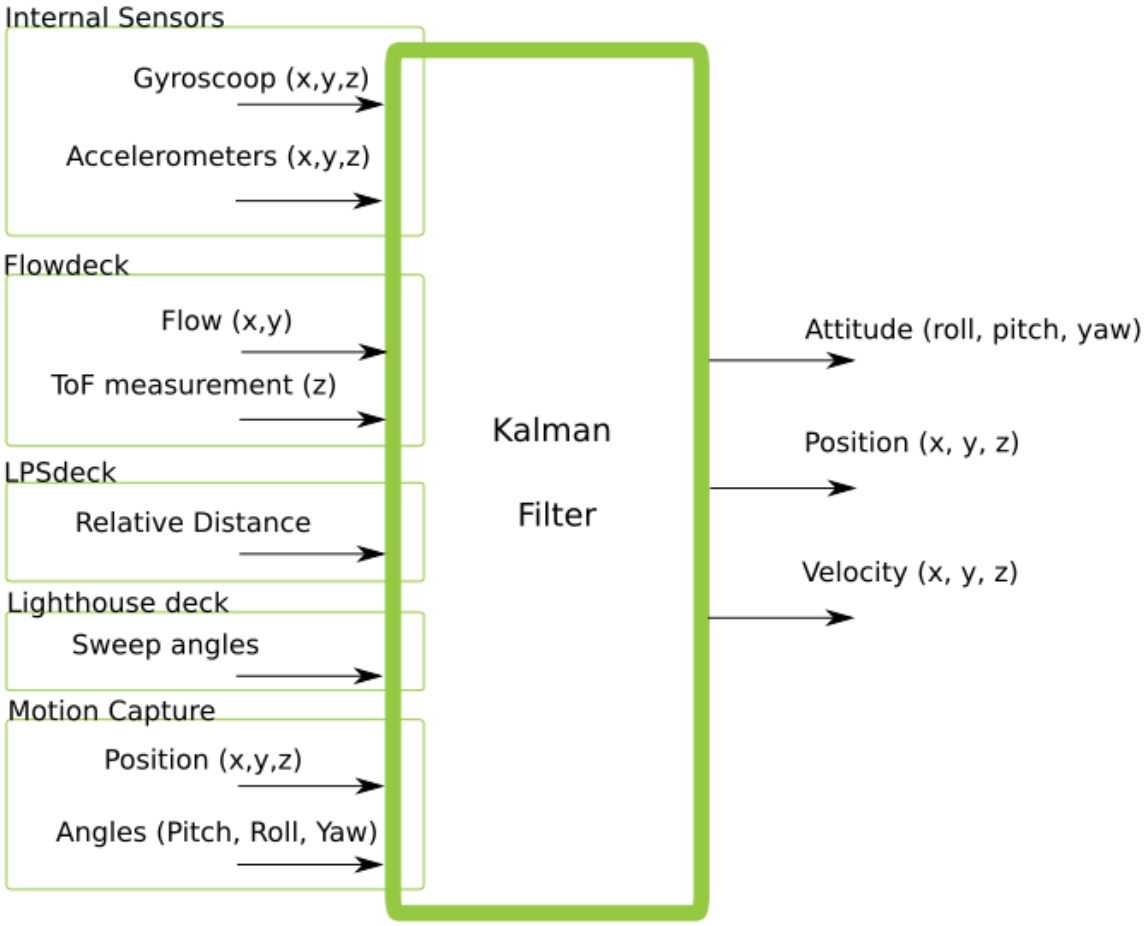


Fig. 33: Extended Kalman Filtering

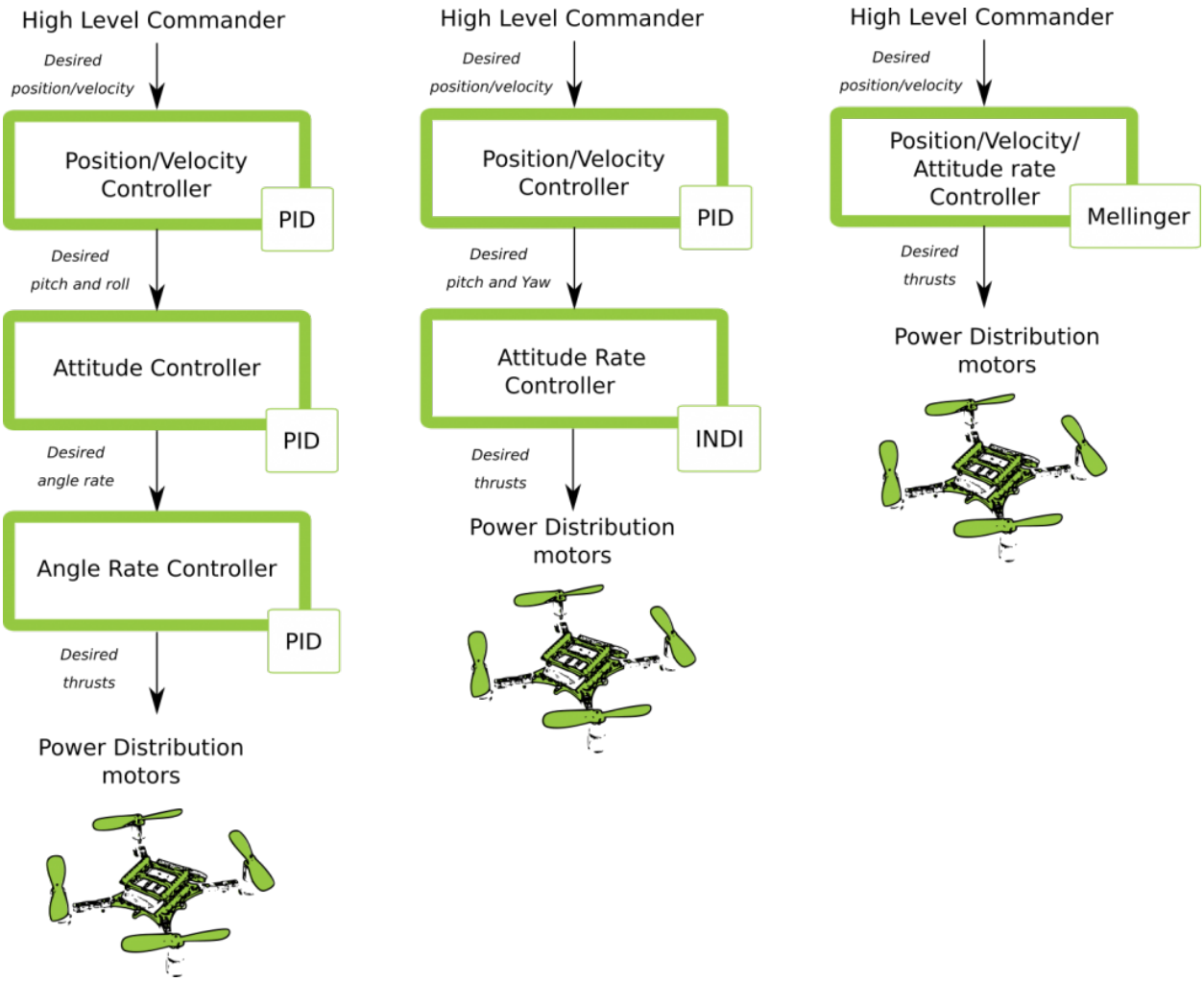


Fig. 34: Possible Controller Pathways

```
static ControllerFcns controllerFunctions[] = {
  {.init = 0, .test = 0, .update = 0, .name = "None"}, // Any
  {.init = controllerPidInit, .test = controllerPidTest, .update = controllerPid, .
↪name = "PID"},
  {.init = controllerMellingerInit, .test = controllerMellingerTest, .update =␣
↪controllerMellinger, .name = "Mellinger"},
  {.init = controllerINDIInit, .test = controllerINDITest, .update = controllerINDI, .
↪name = "INDI"},
};
```

PID Controller

How the PID controller works

The PID controller (proportional-integral-derivative controller), consists of a proportional unit, an integral unit, and a derivative unit, corresponding to the current error, past cumulative error and future error, respectively, and then controls the system based on the error and the error rate of change. The PID controller is considered to be the most suitable controller because of its negative feedback correction. By adjusting the PID controller's three parameters, you can adjust the speed of the system's response to the error, the degree of the controller's overswing and shake, so that the system can reach the optimal state.

This drone system has three control dimensions: `pitch`, `roll`, and `yaw`, so it is necessary to design a PID controller with a closed loop as shown in the figure below.

For each control dimension, a string-level PID controller is provided, consisting of a Rate controller and an Attitude controller. Rate controller is used to control the speed of angle correction based on the input of angular velocity. Attitude controller is used to control the drone to fly at a target angle based on the input of fitting angles. The two controllers can work together at various frequencies. Of course, you can also choose to use only one single-level PID controller, where pitch and roll control dimensions are controlled by Attitude by default, and yaw controlled by Rate.

```
You can modify the parameters in crtp_commander_rpyt.c:
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)
```

Implementation Code

```
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)) { // This macro controls PID calculation␣
↪frequency based on the interrupts triggered by MPU6050
    // Rate-controled YAW is moving YAW angle setpoint
    if (setpoint->mode.yaw == modeVelocity) {                                        ␣
↪           //rate mode, correct yaw
      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 mode
```

(continues on next page)

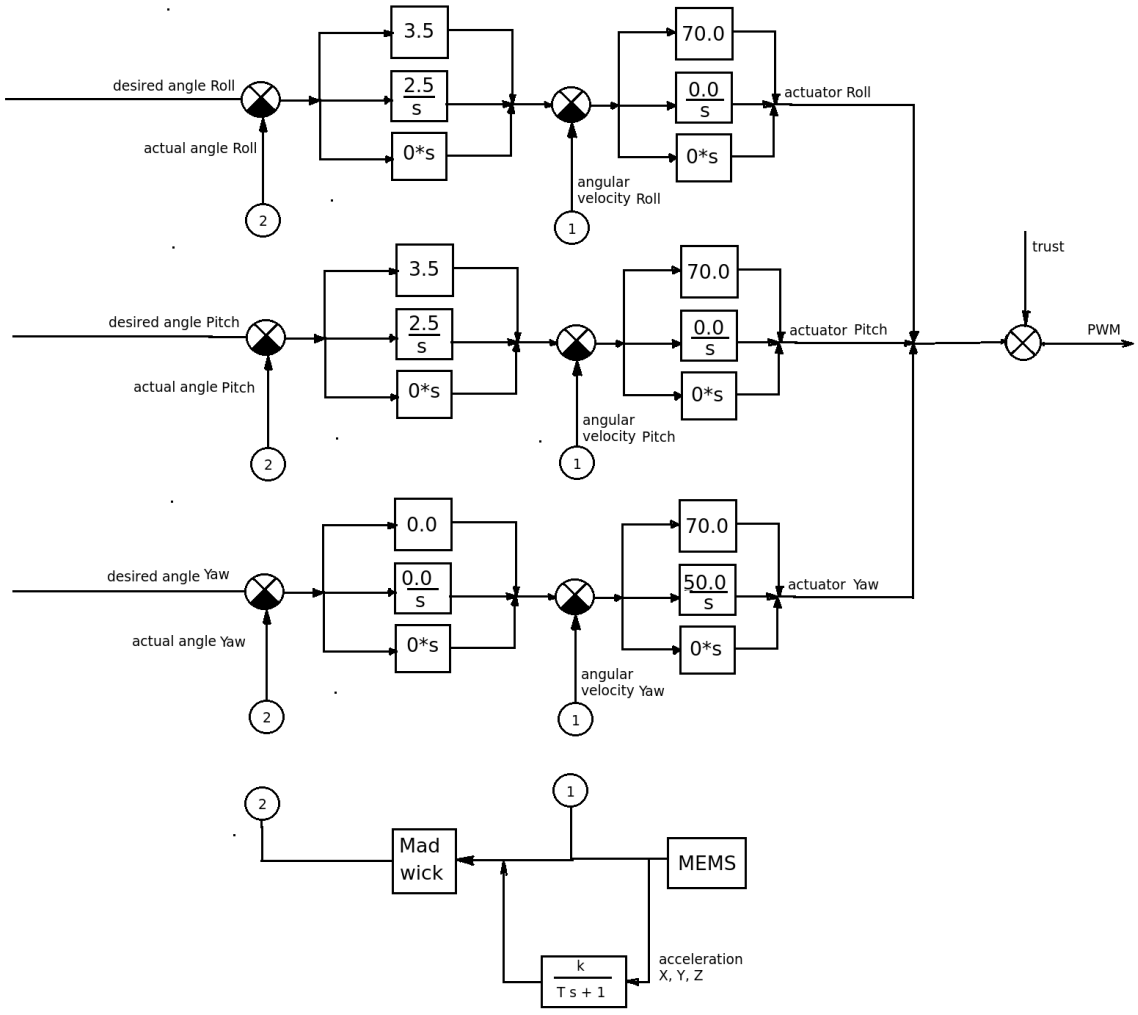


Fig. 35: Crazyflie Control System

(continued from previous page)

```
      attitudeDesired.yaw = setpoint->attitude.yaw;
    }
  }

  if (RATE_DO_EXECUTE(POSITION_RATE, tick)) {                                    ␣
↪       //Position control
    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();
    }

    // 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;
```

(continues on next page)

(continued from previous page)

```
  }

  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 Controller

Mellinger controller is an **all-in-one** controller that directly calculates the required thrust allocated to all motors, based on the target position and the speed vector on the target position.

For your reference: Minimum snap trajectory generation and control for quadrotors.

INDI Controller

An INDI controller is a controller that immediately processes angle rates to determine data reliability. This controller can be used together with a traditional PID controller, which provides a faster angle processing than a string-level PID controller.

For your reference: Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles.

PID Parameter Tuning

Crazyflie Rate PID is tuned as follows:

1. Adjust `Rate` mode first: set `rollType`, `pitchType`, and `yawType` to `RATE`;
2. Adjust `ATTITUDE` mode: set the `KP`, `KI`, and `KD` of `roll`, `pitch`, and `yaw` to `0.0`, and only remain the parameters of `Rate` unchanged.
3. Adjust `RATE` mode: set the `KI`, `KD` of `roll`, `pitch` and `yaw` to `0.0`. Set the `KP` first.
4. Burn the code and start the `KP` adjustment online using the param function of cfclient.
5. Note that the modified parameters using cfclient will not be saved when power down;
6. During PID tuning, shake (over-tuning) may happen, please be careful.
7. Hold the drone to make sure it can only roll around `pitch` axis. Gradually increase the `KP` of `pitch`, till the drone starts shaking back and forth.
8. If the drone shakes intensely, slightly lower `KP`, generally 5%-10% lower than the shake's critical point.
9. Tune the `roll` and `yaw` in the same way.

10. Adjust `KI` to eliminate steady-state errors. If only with proportional adjustment, but without this parameter, the drone may swing up and down at Position 0 due to the interference such as gravity. Set the initial value of `KI` to 50% of `KP`.
11. When the `KI` increases to certain value, the drone starts shaking. But compared with the shake caused by `KI`, that caused by `KP` is more low-frequency. Keep in mind the point when the drone starts shaking, and mark this `KI` as the critical point. The final `KI` should be 5%-10% lower than this critical point.
12. Tune the `roll` and `yaw` in the same way.
13. In general, the value of `KI` should be over 80% of the `KP`.

`Rate PID` parameter tuning is done now.

Let's start the tuning of Attitude PID

1. First ensure that `Rate PID` tuning is completed.
2. Adjust `rollType`, `pitchType`, and `yawType` to `ANGLE`, i.e. the drone is in attitude mode now.
3. Set the `KI` and `KD` of `roll` and `pitch` to `0.0`, and then set the `KP`, `KI`, and `KD`of `Yaw` to `0.0`.
4. Burn the code and start the `KP` tuning online using the param function of cfclient.
5. Set the `KP` of `roll` and `pitch` to `3.5`. Check for any existing instability, such as shakes. Keep increasing the KP until the limit is reached;
6. If the `KP` already is causing drone instability, or the value is over `4`, please lower the `KP` and `KI` of `RATE` mode by 5% ~ 10%. By such way, we have more freedom to tune the Attitude mode.
7. If you still need to adjust the KI, please slowly increase KI again. If some low-frequency shakes occur, it indicates that your drone is in an unstable state.

3.3.3 Communication Protocols

[??]

Communication Hierarchy

Terminal	Mobile/PC	ESP-Drone
Application Layer	APP	Flight Control Firmware
Protocol Layer	CRTP	CRTP
Transport Layer	UDP	UDP
Physical Layer	Wi-Fi STA (Station)	Wi-Fi AP (Access Point)

Wi-Fi Communication

Wi-Fi Performance

ESP32 Wi-Fi Performance

Item	Parameter
Mode	STA mode, AP mode, STA+AP mode
Protocol	IEEE 802.11b/g/n, and 802.11 LR (Espressif). Support switching over software
Security	WPA, WPA2, WPA2-Enterprise, WPS
Main Feature	AMPDU, HT40, QoS
Supported Distance	1 km under the Exclusive Agreement of Espressif
Transfer Rate	20 Mbit/s TCP throughput, 30 Mbit/s UDP

For other parameters, see ESP32 Wi-Fi Feature List.

ESP32-S2 Wi-Fi Performance

Item	Parameter
Mode	STA mode, AP mode, STA+AP mode
Protocol	IEEE 802.11b/g/n. Support switch over software
Security	WPA, WPA2, WPA2-Enterprise, WPS
Main Feature	AMPDU, HT40, QoS
Supported Distance	1 km under the Exclusive Agreement of Espressif
Transfer Rate	20 Mbit/s TCP throughput, 30 Mbit/s UDP

For other parameters, see ESP32-S2 Wi-Fi Feature List.

Wi-Fi Programming Framework

Wi-Fi Programming Framework Based on ESP-IDF

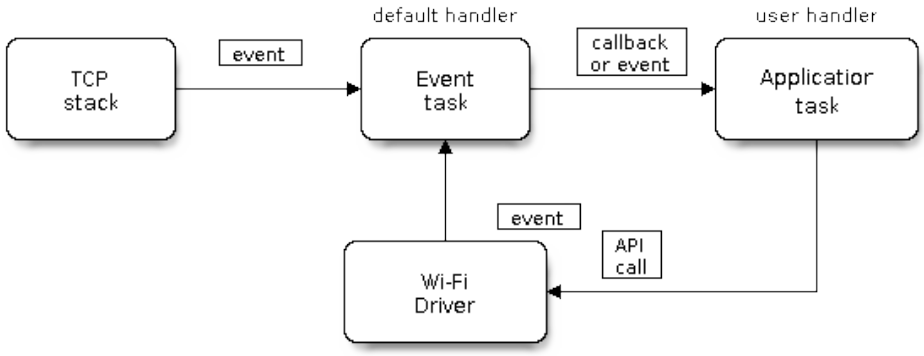


Fig. 36: Wi-Fi Programming Example

General Programming Process

1. At application layer, call Wi-Fi Driver API to initialize Wi-Fi.
2. Wi-Fi driver is transparent to developers. When an event occurs, Wi-Fi driver sends an `event` to default event loop. Applications can write and register `handle` program on demand.

3. Network interface component esp_netif provides the `handle` program associated with the Wi-Fi driver `event` by default. For example, when ESP32 works as an AP, and a user connects to this AP, esp_netif will automatically start the DHCP service.

For the detailed usage, please refer to the code `\components\drivers\general\wifi\wifi_esp32.c`.

Note: Before Wi-Fi initialization, please use `WIFI_INIT_CONFIG_DEFAULT` to obtain the initialization configuration struct, and customize this struct first, then start the initialization. Be aware of problems caused by unitialized members of the struct, and pay special attention to this issue when new structure members are added to the ESP-IDF during update.

AP Mode Workflow

Increase Wi-Fi Communication Distance

Navigate to `Component config>>PHY>>Max WiFi TX power (dBm)`, and update `Max WiFi TX power` to `20`. This configuration increases the PHY gain and Wi-Fi communication distance.

UDP Communication

UDP Port

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

UDP Packet Structure

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

- The packet transmitted by the UDP: CRTP + verification information.
- CRTP: As defined by the CRTP packet structure, it contains Header and Data, as detailed in the CRTP protocol section.
- CKSUM: the verification information. Its size is 1 byte, and this CKSUM is incremented by CRTP packet byte.

CKSUM Calculation Method

```
#take python as an example: Calculate the raw cksum and add it to the end of the␣
↪packet
 raw = (pk.header,) + pk.datat
 cksum = 0
 for i in raw:
         cksum += i
 cksum %= 256
 raw = raw + (cksum,)
```

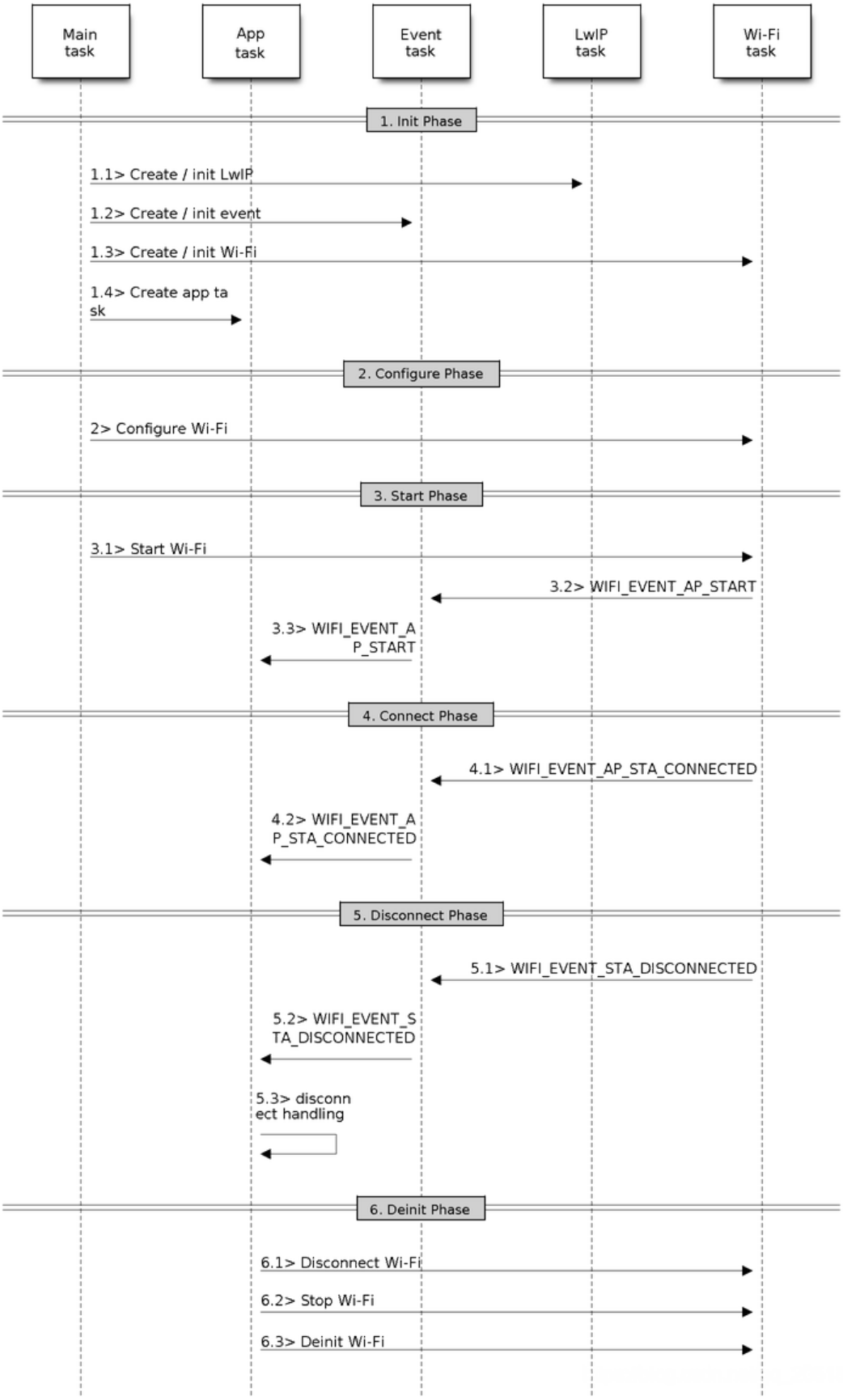


Fig. 37: Sample Wi-Fi Event Scenarios in AP Mode

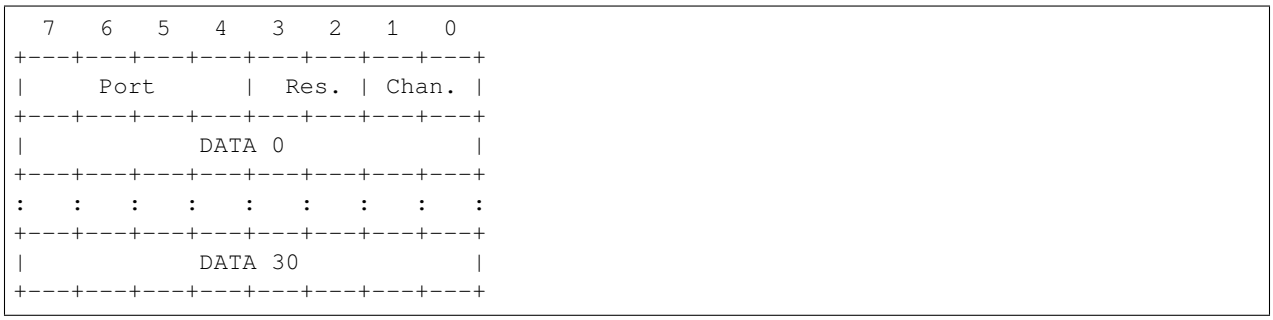
CRTP Protocol

The ESP-Drone project continues the CRTP protocol used by the Crazyflie project for flight instruction sending, flight data passback, parameter settings, etc.

CRTP implements a stateless design that does not require a handshake step. Any command can be sent at any time, but for some log/param/mem commands, the TOC (directory) needs to be downloaded to assist the host in sending the information correctly. The implemented Python API (cflib) can download param/log/mem TOC to ensure that all functions are available.

CRTP Packet Structure

The 32-byte CRTP packet contains one byte of Header and 31 bytes of Payload. Header records the information about the ports (4 bits), channels (2 bits), and reserved bits (2 bits).



Field	Byte	Bit	Description
Header	0	0 ~ 1	Target data channel
	0	2 ~ 3	Reserved for transport layer
	0	4 ~ 7	Target data port
Data	1 ~ 31	0 ~ 7	The data in this packet

Port Allocation

Port	Target	Purpose
0	Console	Read console text that is printed to the console on the Crazyflie using consoleprintf.
2	Parameters	Get/set parameters from the Crazyflie. Parameters are defined using a macro in the Crazyflie source-code
3	Commander	Send control set-points for the roll/pitch/yaw/thrust regulators
4	Memory access	Access non-volatile memories like 1-wire and I2C (only supported for Crazyflie 2.0)
5	Data logging	Set up log blocks with variables that will be sent back to the Crazyflie at a specified period. Log variables are defined using a macro in the Crazyflie source-code
6	Localization	Packets related to localization
7	Generic Setpoint	Allows to send setpoint and control modes
13	Platform	Used for misc platform control, like debugging and power off
14	Client-side debugging	Debugging the UI and exists only in the Crazyflie Python API and not in the Crazyflie itself.
15	Link layer	Used to control and query the communication link

Most of the modules in the firmware that are connected to the port are implemented as tasks. If an incoming CRTP packet is delivered in the messaging queue, the task is blocked in the queue. At startup, each task and other modules need to be registered for a predefined port at the communication link layer.

Details of the use of each port can be found at CRTP - Communicate with Crazyflie.

Supported Package by CRTP Protocol

cflib is a Python package supported by CRTP protocol, and provides an application-layer interface for communication protocols that can be used to build an upper PC, to communicate with Crazyflie and Crazyflie 2.0 quadcopters. Each component in the firmware that uses the CRTP protocol has a script corresponding to it in cflib.

- Source repository: crazyflie-lib-python.
- cflib repository specially for ESP-Drone: qljz1993/crazyflie-lib-python. Please checkout to `esplane` branch.

Application Development Based on CRTP Protocol

Examples for Various Platform

1. crazyflie2-ios-client
2. crazyflie2-windows-uap-client
3. crazyflie-android-client
4. User Guide on Android
5. Development Guide on Android

cfclient

cfclient is the upper PC for `Crazeflie` project, which has fully implemented the functions defined in `CRTP` Protocol, and speeds up the debug process for the drone. The ESP-Drone project tailors and adjusts the upper PC to meet functional design needs.

For detailed information about cfclient, please refer to cfclient.

3.3.4 Hardware Reference

[??]

Supported Hardware

Supported Hardware List

Development Board	Main Components	Notes
ESP32-S2-Drone V1.2	ESP32-S2-WROVER + MPU6050	All in one
ESPlane-V2-S2	ESP32-S2-WROVER + MPU6050	Four feet should be installed
ESPlane-FC-V1	ESP32-WROOM-32D + MPU6050	A drone frame should be installed

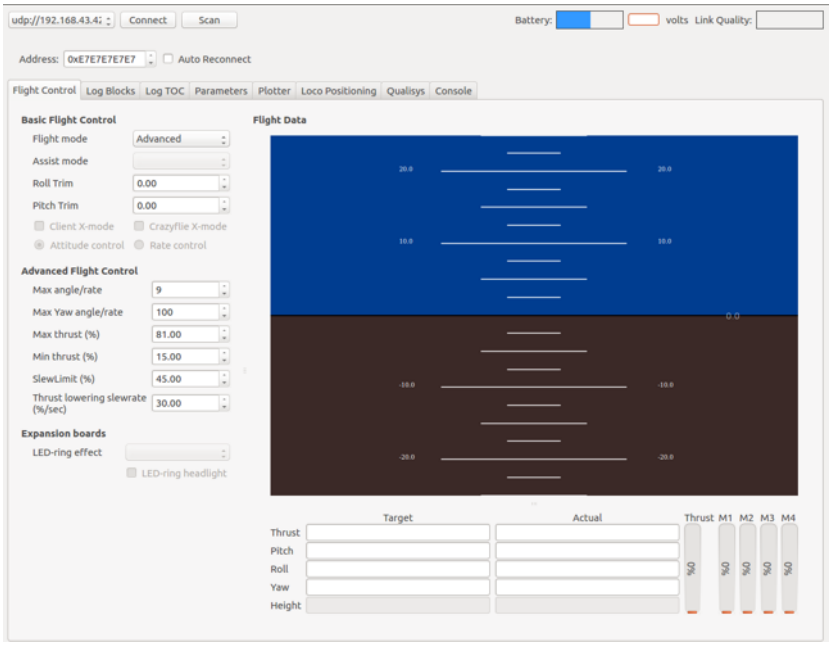
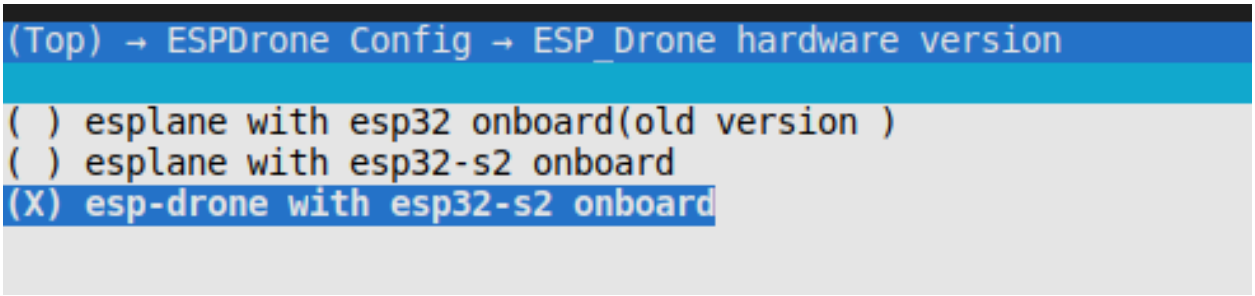


Fig. 38: Cfclient Control Interface

Configure Target Hardware

- Code in `esp_drone` repository supports a wide variety of hardware, which can be changed through `menuconfig`.



- By default, when set `set-target` as `esp32s2`, target hardware will be changed to `ESP32_S2_Drone_V1_2` automatically.
- By default, when set `set-target` as `esp32`, target hardware will be changed to `ESPlane_FC_V1` automatically.

Notes

1. ESPlane-FC-V1 is an old version hardware.
2. To use ESP-Drone new version code on ESPlane-FC-V1, please connect GPIO14 of ESP32-WROOM-32D to INT pin of MPU6050 by a jumper.
3. To avoid flash voltage switching triggered by IO12 when ESPlane-FC-V1 is powered on, please fix the flash voltage to 3.3 V by running `espefuse.py` script

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

   note * Only the first device attaching to the bus can use CS0 pin.
   ```

ESP32-S2-Drone V1.2

Fig. 39: ESP32-S2-Drone V1.2 Overview

- Main Board Schematic：SCH_Mainboard_ESP32_S2_Drone_V1_2
- Main Board PCB：PCB_Mainboard_ESP32_S2_Drone_V1_2

Basic Component

Basic Component List

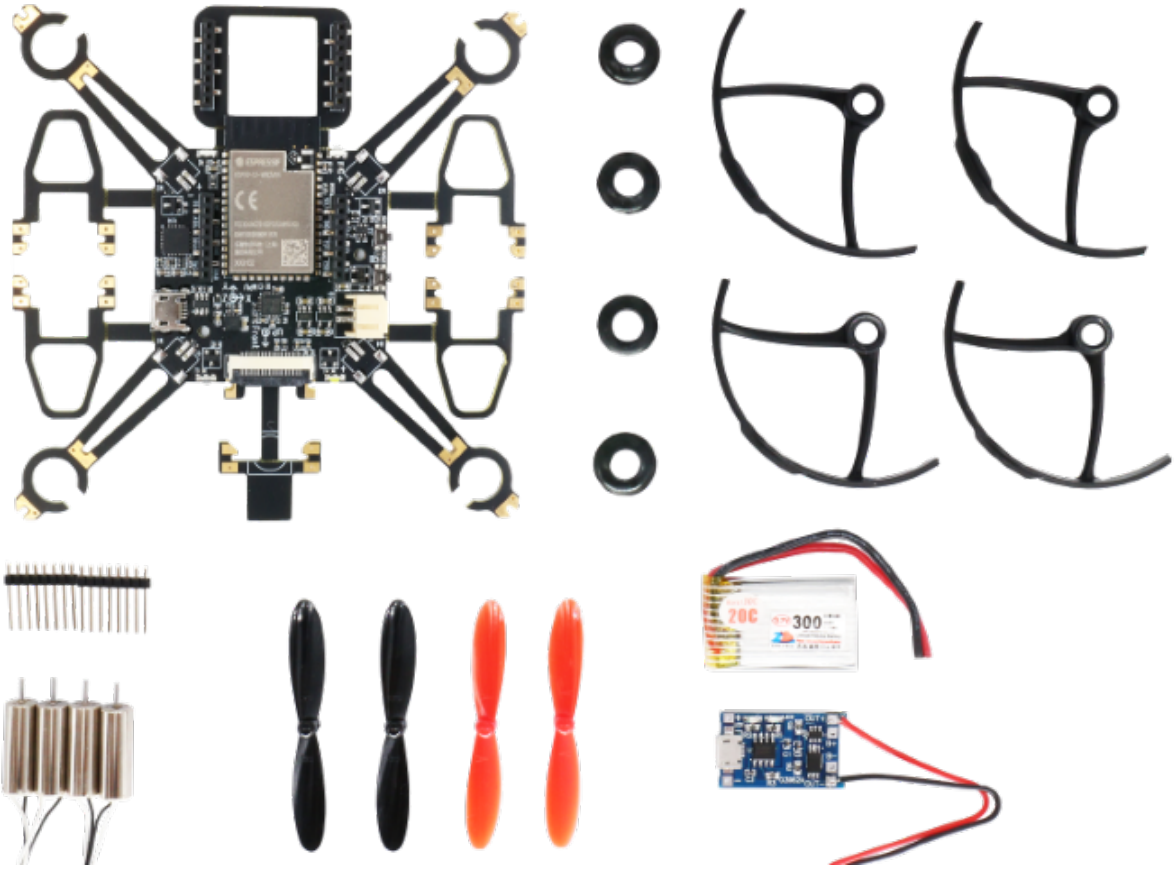


Fig. 40: Component List for ESP32-S2-Drone V1.2

Basic Component List	Number	Notes
Main board	1	ESP32-S2-WROVER + MPU6050
716 motor	4	Optional: 720 motor
716 motor rubber ring	4	
46mm propeller A	2	Optional: 55mm propeller
46mm propeller B	2	
300mAh 1s LiPo battery	1	Optional: 350mAh
1s LiPo battery charging panel	1	
8-pin 25 mm male pins	2	

Note: Set `motor type` as `brushed 720 motor` through `menuconfig->ESPDrone Config->motors config` if you use 720 motor.

Main Controller

Chip	Module	Notes
ESP32-S2	ESP32-S2-WROVER	4 MB flash, 2 MB PSRAM in module

Sensor

Sensor	Interface	Notes
MPU6050	I2C0	Main board Sensor

LEDs

Status	LED	Action
POWER_ON	WHITE	Fully lit
SENSORS CALIBRATION	BLUE	Blinking slowly
SYSTEM READY	BLUE	Blinking
UDP_RX	GREEN	Blinking
LOW_POWER	RED	Fully lit

Buttons

Buttons	IO	Function
SW1	GPIO1	Boot, Normal
SW2	EN	Reset

Definition of Main Board IO

Pins	Function	Notes
GPIO11	I2C0_SDA	Only for MPU6050
GPIO10	I2C0_SCL	Only for MPU6050
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	

Camera Interface

Pins	Function
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

Extension Components

Extension Board	Main Sensor	Function	Interfaces	Mount Location
Position-hold module	PMW3901 + VL53L1X	Indoor position-hold flight	SPI + I2C	Mount at bottom, facing to the ground.
Pressure module	MS5611 pressure module	Height-hold flight	I2C or MPU6050 slave	Mount at the top or at the bottom
Compass module	HMC5883 compass	Advanced flight mode, such as head-free mode	I2C or MPU6050 slave	Mount at the top or at the bottom

Definition of Extension Board IO

Left Pins	IO	Right pins	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

- Main Board Schematic：SCH_ESPlane_V2_S2
- Main Board PCB：PCB_ESPlane_V2_S2

ESPlane-FC-V1

- Main Board Schematic：Schematic_ESPlane_FC_V1
- Main Board PCB：PCB_ESPlane_FC_V1

Basic Component

Basic Component List

Basic Component List	Number	Notes
Main board	1	ESP32-WROOM-32D + MPU6050
Drone frame	1	
46 mm propeller A	2	
46 mm propeller B	2	
300 mAh 1s LiPo battery	1	
1s LiPo battery charging panel	1	

Fig. 41: ESPlane-V2-S2 Overview

Fig. 42: esplane_fc_v1

Sensor

Sensor	Interface	Notes
MPU6050	I2C0	Must

LEDs

```
#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
```

Status	LED	Action
SENSORS READY	BLUE	Fully lit
SYSTEM READY	BLUE	Fully lit
UDP_RX	GREEN	Blinking

Definition of Main Board IO

Pins	Function	Notes
GPIO21	SDA	I2C0 data
GPIO22	SCL	I2C0 clock
GPIO14	SRV_2	MPU6050 interrupt
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

Components of Extension Board

ESPlane + PMW3901 Pins Allocation

Pins	Function	Notes
GPIO21	SDA	I2C0 data
GPIO22	SCL	I2C0 clock
GPIO12	MISO/SRV_1	HSPI
GPIO13	MOSI	HSPI
GPIO14	SCLK/SRV_2	HSPI
GPIO15	CS0*	HSPI
GPIO16	RX2	
GPIO17	TX2	
GPIO19	interrupt	MPU6050 interrupt
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

3.4 Third-Party Codes

[??]

The third-party codes adopted in ESP-Drone project and the licenses are as follows:

Components	Licenses	Source Codes	Commit ID
core/crazyflie	GPL-3.0	Crazyflie	tag_2021_01 b448553
lib/dsp_lib		esp32-lin	6fa39f4cd5f7782b3a2a052767f0fb06be2378ff

3.5 Acknowledgement

1. Thanks Bitcraze for providing the amazing drone codes: Crazyflie
2. Thanks Espressif for providing ESP32 and ESP-IDF
3. Thanks WhyEngineer for providing DSP lib esp-dsp