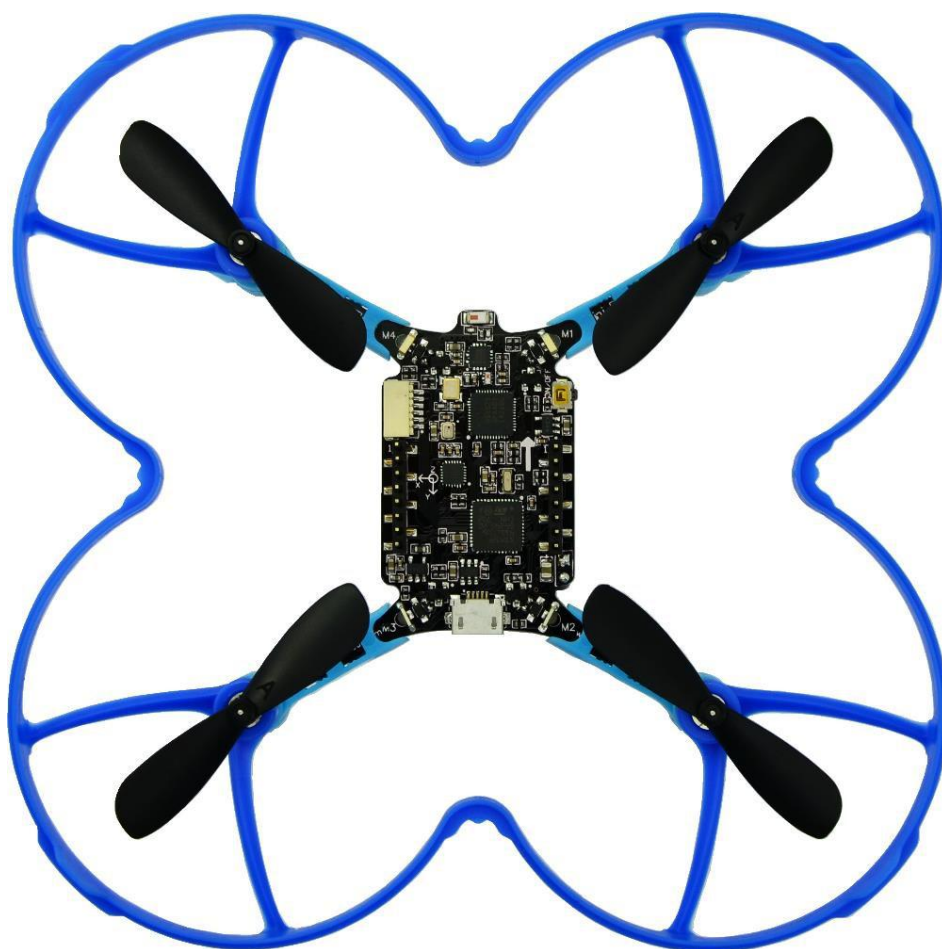


西电航协微型四轴开发指南



主编&项目策划:	西安电子科技大学 电子工程学院 吕瑞涛
硬件设计:	西安电子科技大学 电子工程学院 王晗宇
机械设计:	西安电子科技大学 空间科学与技术学院 程天宇
软件设计:	西安电子科技大学 电子工程学院 吕瑞涛 李妍榕

目 录

目录

1. 四轴飞行器的基本组成.....	5
1.1 电机.....	5
1.2 MOS 管和电调.....	5
1.3 航模电池.....	6
1.4 正反桨.....	6
1.5 四轴机架.....	7
1.6 四轴遥控器.....	7
1.7 四轴飞控.....	8
2. 四轴飞行器的基本工作原理.....	9
2.1 垂直上升和垂直下降.....	9
2.2 向前运动和向后运动.....	9
2.3 向左运动和向右运动.....	10
2.4 顺时针改变航向和逆时针改变航向.....	10
3. 航模常见术语.....	11
3.1 X 模式和+模式.....	11
3.2 姿态角 pitch/roll/yaw.....	11
3.3 有头模式和无头模式.....	12
4. 微型四轴硬件资源.....	13
4.1 系统框架.....	13
4.2 主控 MCU 接口.....	13
4.3 传感器接口.....	14
4.4 电机接口.....	15
4.5 无线通信和 PA 接口.....	15
4.6 电源接口.....	16
4.7 LED 接口.....	17
4.8 下载接口.....	17
5. STM32 基础知识.....	18
5.1 STM32 时钟树原理.....	18

5.2 STM32 GPIO 原理.....	18
5.3 STM32 NVIC 原理.....	19
5.4 STM32 EXTI 原理.....	20
5.5 STM32 ADC 和 DMA 原理.....	21
5.5.1 STM32 ADC 原理.....	21
5.5.2 STM32 DMA 原理.....	21
5.5.3 DMA 在 AD 转换中的作用.....	22
5.6 STM32 PWM 原理.....	22
6. 四轴飞行器算法讲解.....	23
6.1 姿态解算简介.....	23
6.2 飞行器姿态表示方法.....	23
6.2.1 欧拉角	23
6.2.2 四元数.....	24
6.3 软件姿态解算.....	25
6.4 PID 控制理论.....	28
6.5 四轴常用的 2 种 PID.....	30
6.5.1 单级 PID.....	30
6.5.2 串级 PID.....	30
6.6 串级 PID 参数调试.....	31
7. 微型四轴软件原理.....	32
7.1 STM32G030 程序解读.....	32
7.1.1 时钟树配置.....	32
7.1.2 LED 指示灯配置.....	33
7.2 姿态解算和 PID 算法流程图.....	34
7.3 基于四元数的姿态解算互补滤波算法.....	35
7.4 角度环 PID 和角速度环 PID.....	36
7.5 姿态控制量和油门值整合.....	41
7.6 蓝牙无线串口通讯协议.....	42
7.7 手机控制软件框架.....	43
8. 微型四轴二次开发.....	44

8.1 MDK 开发环境搭建.....	44
8.2 CubeMX 开发环境搭建.....	45
8.3 固件下载.....	45
8.4 MWC 地面站使用.....	47
8.5 微型四轴 PID 调试.....	48
9. 微型四轴开发常见问题.....	53
9.1 STM32 虚拟串口问题.....	53
9.2 CubeMX 代码生成问题.....	55
9.3 四轴飞行过程往一边偏.....	56
9.4 四轴定高飞行，高度定不住.....	57
9.5 下载器不能下载调试代码.....	57

1. 四轴飞行器的基本组成

四轴飞行器主要是由电机、电调、电池、浆叶、机架、遥控器、飞控组成。下面我们分别以我们微型四轴及市场上常见的 DIY 大四轴来介绍这些部件。

1.1 电机

电机分为有刷电机和无刷电机两类，分别如图 1.1.1 和图 1.1.2 所示。



图 1.1.1 空心杯电机



图 1.1.2 无刷电机

有刷电机体积小价格便宜而且控制简单，比较适用于做微型四轴。我们设计的微型四轴就是采用了图 1.1.1 空心杯有刷电机，型号为 716。

无刷电机体积稍大，载重能力强，比较适用于 DIY 大四轴。无刷电机控制需要搭配专门的无刷电机电调模块。如图 1.1.2 无刷电机，2212 1200KV，22 表示电机的外转子直径为 22mm；12 表示转子的高度为 12mm；1200KV 表示电压每增加 1V 电机的转速增加 1200r/min。

1.2 MOS 管和电调

微型四轴和大四轴一般分别使用 MOS 管和电调驱动电机，MOS 管和电调分别如图 1.2.1 和 1.2.2 所示：

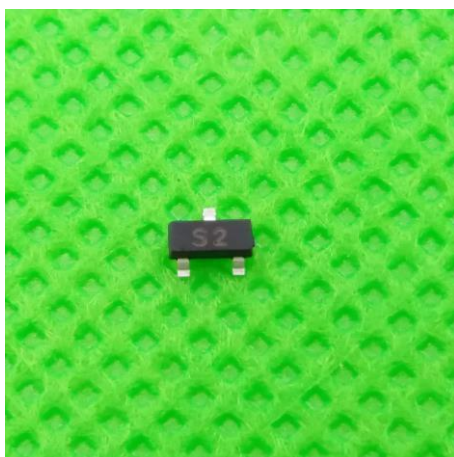


图 1.2.1 MOS 管



图 1.2.2 无刷电机电调模块

电调即为电子调速器，控制电机转动、停止及转速。

有刷电机电调通常只需要一个 MOS 管，飞控输出 PWM 即可控制电机。我们设计的微型四轴采用如图 1.2.1 所示 SOT-23 封装的 MOS 管，型号为：SI2302。

无刷电机电调模块如图 1.2.2 所示。无刷电机电调模块内部通常由一个 MCU 和三相桥电路组成，MCU 通过控制三相桥来实现无刷电机换相。同样，无刷电机电调模块也只需飞控输出 PWM 即可控制电机。

1.3 航模电池

微型四轴选用的锂电池如图 1.3.1 所示，参数为 250mAh 20C 1S。DIY 大四轴常用的锂电池如图 1.3.2 所示。



图 1.3.1 250mAh 20C 1S 锂电池



图 1.3.2 2200mAh 20C 3S 锂电池

下面我们对这些参数做一下解释。

电池容量 mAh: 锂电池的容量，如 2000mAh 的电池，以 2000mA 放电，可持续放电 1 小时；以 1000mA 放电，可持续放电 2 小时。

电池节数: 电池 2S、3S、4S 代表锂电池节数。锂电池 1 节的标准电压为 3.7V，3S 代表有 3 节 3.7V 的电池在里面，电压为 11.1V。

电池放电倍率 C: 四轴一般使用的动力锂电池，也是放电倍率 C 较大。普通锂电池放电倍率通常为 1C 或者 0.5C，动力电池则为十几二十 C。如标称 250mAh 20C 的电池，它的放电电流为 $250\text{mA} \times 20$ ，即为 5A 放电电流。

1.4 正反桨

微型四轴选用如图 1.4.1 所示，直径为 46mm 的桨叶，带 A 字的桨为正桨。图 1.4.2 1038 正反桨叶为 DIY 大四轴所用的桨叶，1038 表示直径 10 英寸，螺距 3.8 英寸。

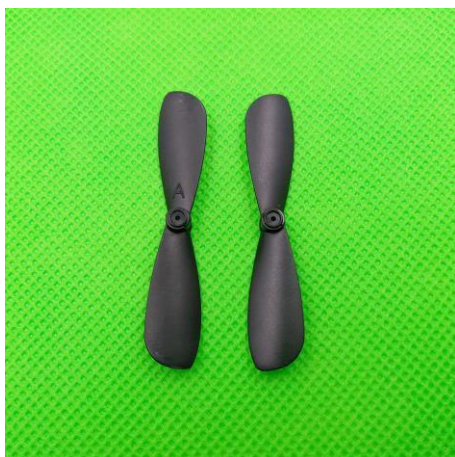


图 1.4.1 46mm 正反浆叶

四轴飞行器为什么需要正反浆叶？

答：浆叶旋转时会产生自旋力导致四轴自旋，为了抵消自旋力相隔电机的浆叶旋转方向要不一样，但是浆的风都是要往下吹，这就出现了正反浆的说法。通常顺时针转的叫正浆，逆时针转的是反浆。

图 1.4.2 1038 正反浆叶

1.5 四轴机架

机架一般采用质轻、硬度好的材料，大四轴常采用碳纤维材质或尼龙材质作为机架。小四轴使用硬塑料材质或 PCB 都可以。微型四轴直接采用轻木板作为支架，支架集成控制电路一体不仅小巧美观而且功能齐全，如图 1.5.1 所示。DIY 大四轴常用的碳纤维支架，如图 1.5.2 所示。



图 1.5.1 PCB 机架



图 1.5.2 碳纤维支架

1.6 四轴遥控器

微型四轴遥控器采用蓝牙控制，简便快捷。如图 1.6.2 所示的商业遥控器采用的是日本手方式。商业遥控也有美国手的，购买时根据自己的操作习惯区分购买。



图 1.6.2 商业遥控器

遥控器用于发送各种控制指令，常用通信频段为 2.4G。四轴飞行器的控制至少需要 4 个通道，功能越多需要的通道数越多。遥控器按操作习惯可分为美国手和日本手两种。什么是遥控器通道？

答：通道就是可以遥控器控制的动作路数。四轴在控制过程中需要控制的动作路数有：上下、左右、前后、旋转。所以最低得 4 通道遥控器。

什么是美国手、日本手？

答：油门摇杆在左边、方向摇杆在右边是美国手；相反，油门摇杆在右边、方向摇杆在左边是日本手。

1.7 四轴飞控

微型四轴飞控如图 1.7.1 所示。国外 APM 飞控如图 1.7.2 所示，



图 1.7.1 微型四轴飞控

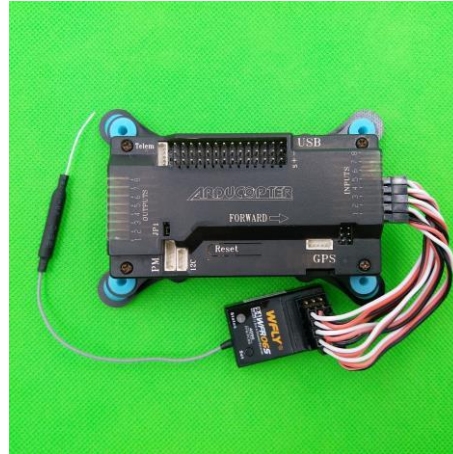


图 1.7.2 APM 飞控和遥控器接收机

飞控是四轴飞行器的核心部件，主要由 MCU、陀螺仪、加速度计、磁力计、气压计、无线接收模块等组成。飞控采集各路传感器数据，经过姿态解算求出姿态，然后对比遥控器发过来的控制命令数据，再经过 PID 计算出控制量，最后将控制量转换成 PWM 信号，分别控制各个电机。

遥控器接收机是接收遥控信号的，它与遥控器是配套的。

2. 四轴飞行器的基本工作原理

四轴飞行器基本原理是通过飞控控制四个电机旋转带动浆叶产生升力，分别控制每一个电机和浆叶产生不同的升力从而控制飞行器的姿态和位置。四轴在空中可以实现八种运动，分别为垂直上升、垂直下降、向前运动、向后运动、向左运动、向右运动、顺时针改变航向、逆时针改变航向。下面分别介绍实现的这些动作的原理。

2.1 垂直上升和垂直下降

当四轴飞行在空中自稳后，M1、M2、M3、M4 四个电机同时转速增大或同时转速减小，即可发生垂直上升运动或垂直下降运动，分别如图 2.1.1 和图 2.1.2 所示：。

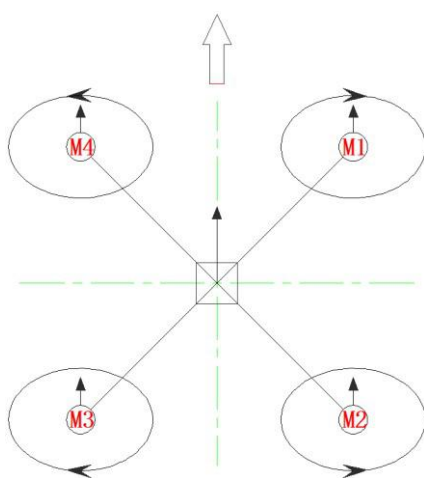


图 2.1.1 垂直上升

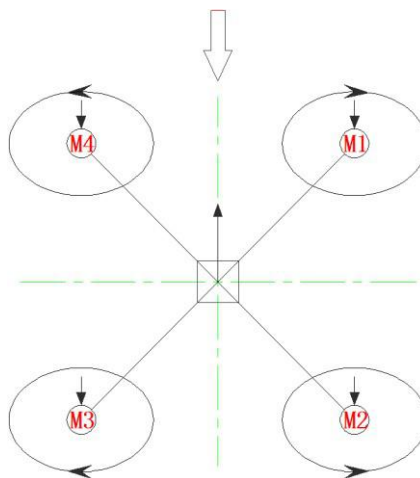


图 2.1.2 垂直下降

2.2 向前运动和向后运动

如图 2.2.1 所示，当四轴飞行在空中自稳后，M2、M3 转速增大 M1、M4 转速不变或减小即可实现向前运动。相反，如图 2.2.2 所示，M2、M3 转速减小或不变 M1、M4 转速增加即可实现向后运动。

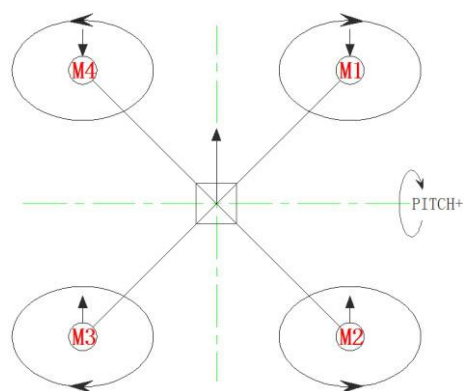


图 2.2.1 向前运动

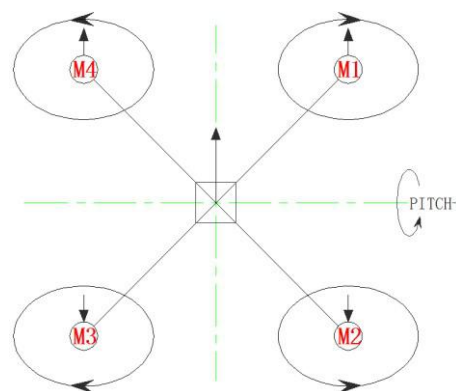


图 2.2.2 向后运动

2.3 向左运动和向后运动

如图 2.3.1 所示，当四轴飞行在空中自稳后，M1、M2 转速增大 M3、M4 转速不变或减小即可实现向左运动。相反，如图 2.3.2 所示，M1、M2 转速减小或不变 M3、M4 转速增加即可实现向右运动。

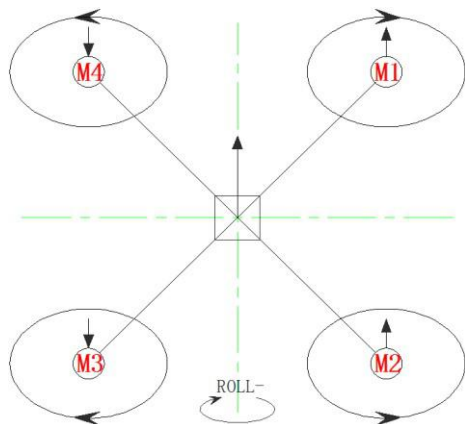


图 2.3.1 向左运动

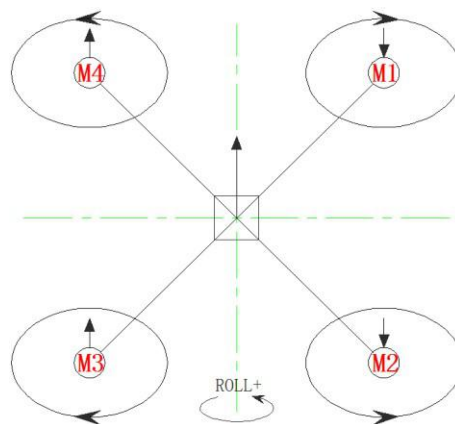


图 2.3.2 向右运动

2.4 顺时针改变航向和逆时针改变航向

如图 2.4.1 所示，当四轴飞行在空中自稳后，M1、M3 转速增大 M2、M4 转速不变或减小即可实现顺时针改变航向。相反，如图 2.4.2 所示，M1、M3 转速减小或不变 M2、M4 转速增加即可实现逆时针改变航向。

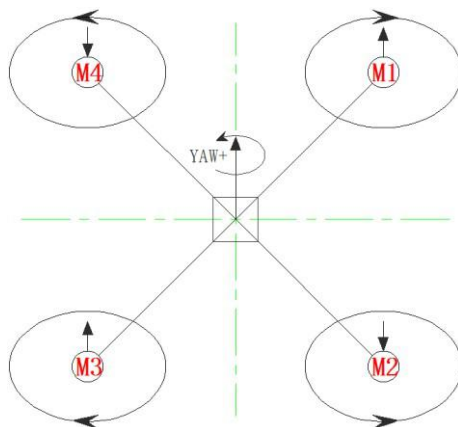


图 2.4.1 顺时针改变航向

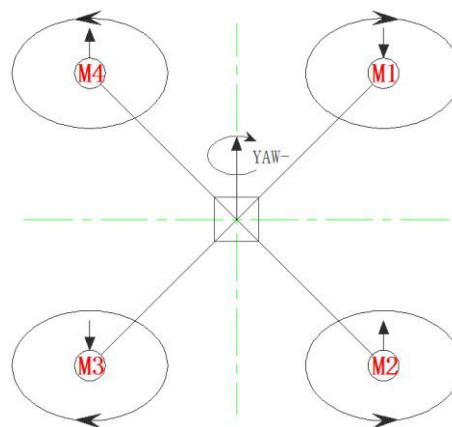


图 2.4.2 逆时针改变航向

3. 航模常见术语

3.1 X 模式和+模式

从结构形式上四轴飞行器可分为十字模式和 X 模式。十字模式如下图 3.1.1 所示，X 模式如下图 3.1.2 所示。对于姿态测量和控制来说，两种结构差别不大。如果考虑安装航拍摄像机，为了视线不被挡住，通常采用 X 模式。

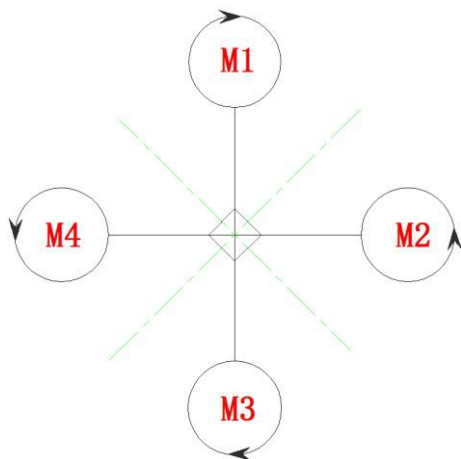


图 3.1.1 十字模式

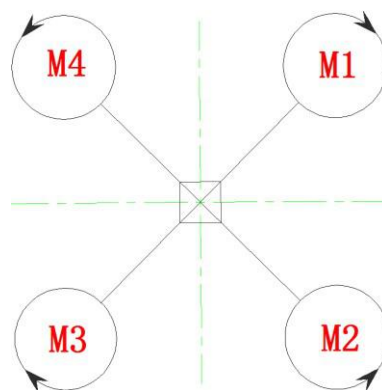


图 3.1.2 X 模式

3.2 姿态角 pitch/roll/yaw

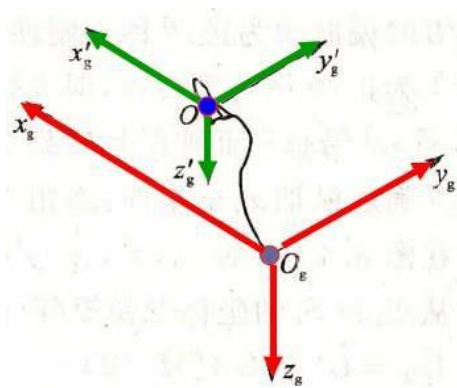


图 3.2.1 地面坐标系

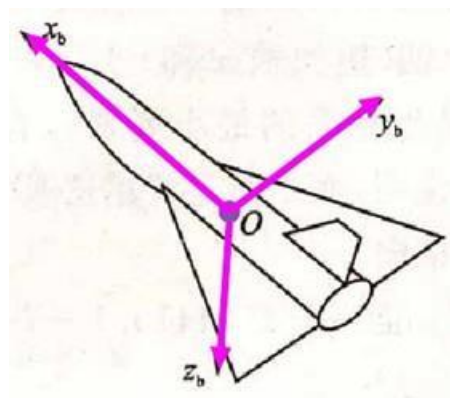


图 3.2.2 机体坐标系

地面坐标系，如图 3.2.1 所示：

- 在地面上选一点 O_g ;
 - 使 X_g 轴在水平面内并指向某一方向;
 - Z_g 轴垂直于地面并指向地心;
 - Y_g 轴在水平面内垂直于 X_g 轴，其指向按右手定则确定;
- 机体坐标系，如图 3.2.2 所示：
- 原点 O 取在飞机质心处，坐标系与飞机固连;
 - X 轴在飞机对称平面内并平行于飞机的设计轴线指向机头;
 - Y 轴垂直于飞机对称平面指向机身右方;
 - Z 轴在飞机对称平面内，与 X 轴垂直并指向机身下方;

姿态角（pitch、roll、yaw）是飞行器的机体坐标系与地面坐标系的夹角，也叫做欧拉角（Euler Angle）。飞行器在空中任何一种姿态都可以通过姿态角旋转后得到。姿态角的旋转关系图如下图 3.2.3 所示。



图 3.2.3 姿态角旋转关系图

俯仰角（**pitch**）：机体坐标系 X 轴与水平面的夹角，围绕 X 轴旋转。当 X 轴的正半轴位于过坐标原点的水平面之上（抬头）时，俯仰角为正，否则为负。

偏航角（**yaw**）：机体坐标系 X 轴在水平面上投影与地面坐标系 X_g 轴之间的夹角，围绕 Y 轴旋转。机头右偏航为正，反之为负。

滚转角（**roll**）：机体坐标系 Z 轴与通过机体 Z 轴的铅垂面间的夹角，围绕 Z 轴旋转。机体向右滚为正，反之为负。

3.3 有头模式和无头模式

有头模式：飞行器运动的前后左右以自身的坐标系为参考坐标系。当飞行器旋转（航向角改变）后，前进方向始终是机头的方向。

无头模式：飞行器运动的前后左右以地面坐标系为参考坐标系。当飞行器旋转（航向角改变）后，前进方向始终是初始设定的前进方向。

4. 微型四轴硬件资源

4.1 系统框架

微型四轴系统框架如图 4.1.1 所示：

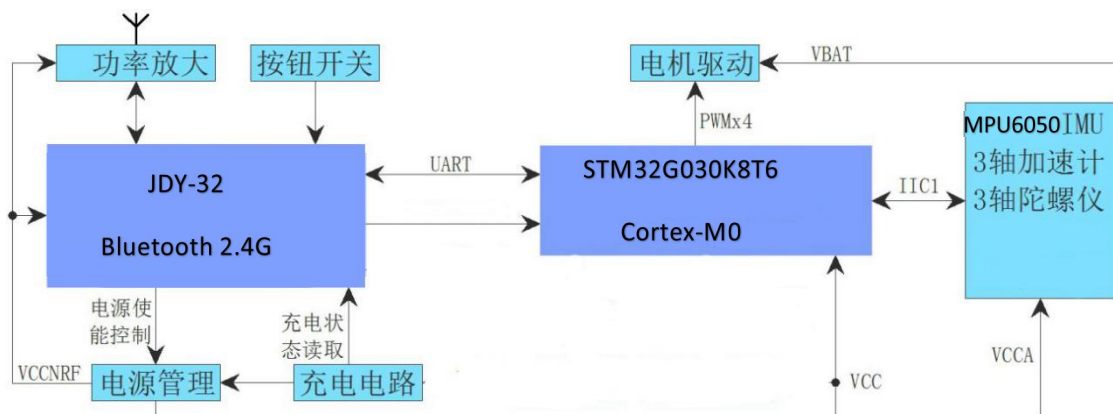


图 4.1.1 系统框架

可以看到，微型四轴采用 STM32G030K8T6 MCU 控制，JDY-32 蓝牙模块主要负责无线通信，Cortex-M0 内核 MCU 主要负责传感器读取，数据融合，PID 控制和电机控制等。而这 2 个模块之间的通信方式为 UART。电路原理详细分析请接着往下看。

4.2 主控 MCU 接口

主控 MCU STM32G030K8T6 电路图如图 4.2.1 所示：

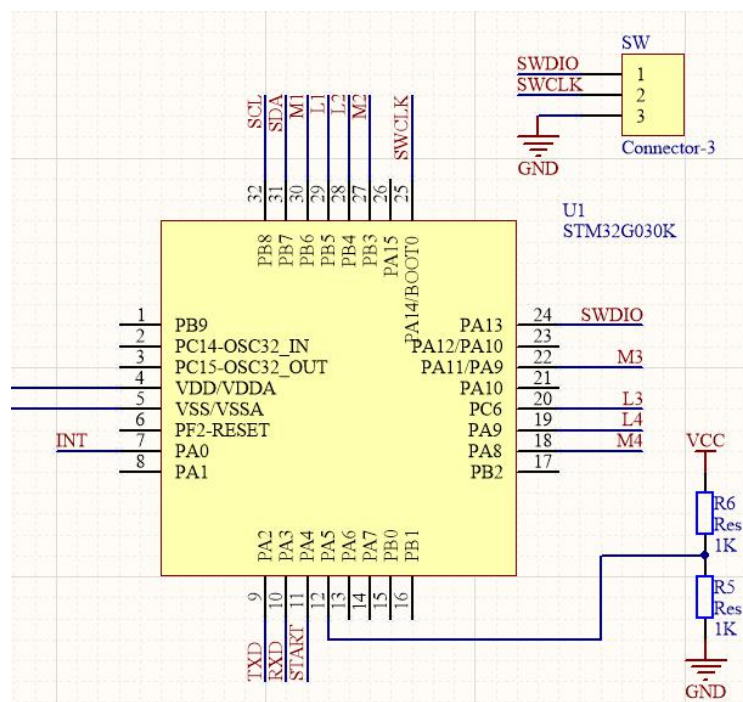


图 4.2.1 主控 MCU

主控 MCU 采用 STM32G030K8T6, 该 32-bit Cortex-M0 内核芯片比较强大, 集成 FPU 和 DSP 指令, 并具有 64KB FLASH、3 个 16 位定时器、2 个 32 位定时器、2 个 DMA 控制器、5 个 SPI (全双工 I2S)、3 个 IIC、2 个串口、1 个 USB (支持 HOST/SLAVE)、16 通道 12 位 ADC、1 个 RTC (带日历功能)、1 个 SDIO 接口、以及 36 个通用 IO 口等。

主控 MCU 为四轴飞行器的大脑, 对飞行器稳定飞行起着至关重要的作用。它同时承担着多种责任, 包括: 传感器数据读取、数据融合、PID 控制、电机控制、无线通信等。

主控 MCU 通过 UART2 串口连接了一个蓝牙模块, 此模块可以用作微型四轴与上位机通信, 也可用于安卓设备控制该四轴。

4.3 传感器接口

微型四轴板载一颗六轴传感器 MPU6050, 电路图如图 4.3.1 所示:

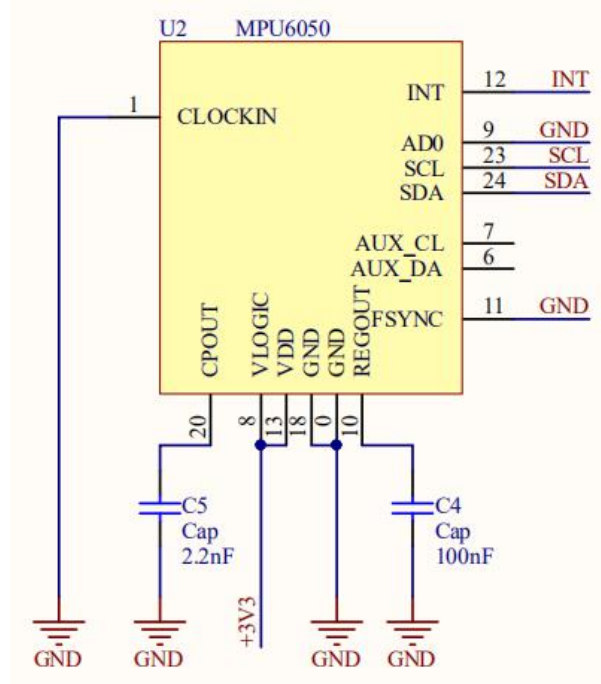


图 4.3.1 MPU6050 六轴传感器

六轴传感器芯片为 MPU6050, 该芯片内部集成了 3 轴加速度传感器、3 轴陀螺仪传感器。并且自带 DMP(Digital Motion Processor), 支持 MPL。此传感器用于测量四轴的姿态数据。

MPU6050 支持 SPI 和 IIC 通信。为方便开发, 这 2 颗传感器我们都使用 IIC 通信, MPU6050 挂接在 MCU 的硬件 IIC1 接口 (IMU_SCL:PB8, IMU_SDA:PB7) 上, 而且我们使用模拟 IIC 的通信方式, 因为模拟 IIC 的通信方式已经能满足我们要求。我们可以看到 MPU6050 的 IIC 地址为 0xD2。

4.4 电机接口

微型四轴板载四个便携式插拔电机接口，电路图如图 4.4.1 所示：

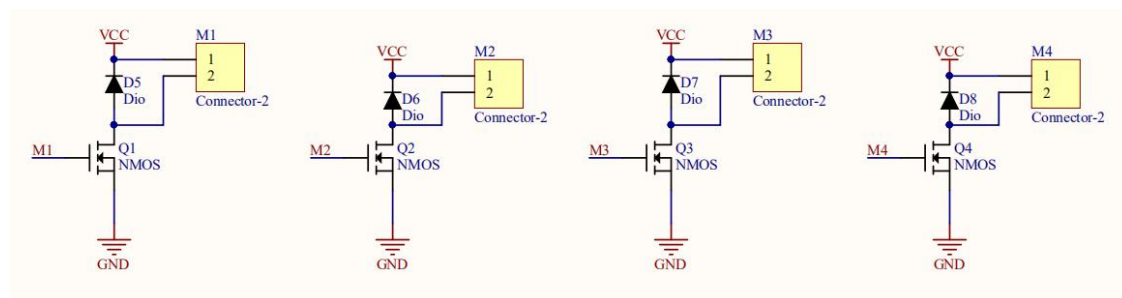


图 4.4.1 电机接口

微型四轴采用微型高速 716 空心杯电机，电机转速高达 45000r/min，能够为飞行器提供充沛的动力。电机采用 NMOS 管 SI2302，3V 门级驱动电压下，导通电阻只有几十毫欧，驱动电流高达 3A，轻松驱动 716 空心杯电机，从而带动飞行器飞行。

4.5 无线通信和 PA 接口

微型四轴板载一颗蓝牙无线通信芯片 JDY-32，电路图如图 4.5.1 所示：

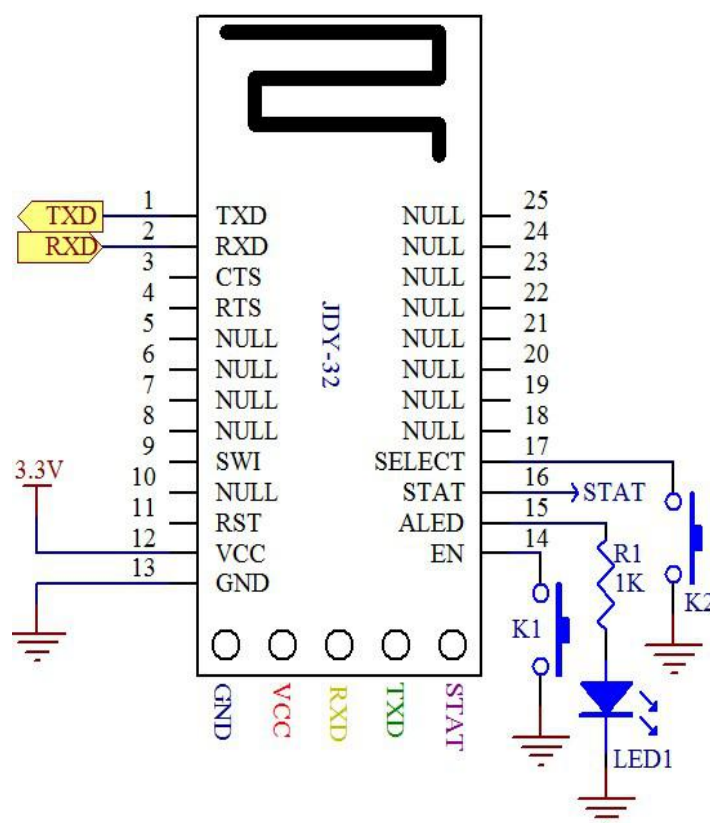


图 4.5.1 蓝牙无线通信芯片 JDY-32

JDY-32 双模蓝牙基于蓝牙 3.0 SPP + 蓝牙 4.2 BLE 设计，这样可以支持 Windows、Linux、ios、android 数据透传，工作频段 2.4GHZ，调制方式 GFSK，最大发射功率 5db，最大发射距离 40 米，支持用户通过 AT 命令修改设备名、波特率等指令，方便快捷使用灵活。JDY-32 支持经典蓝牙与 BLE 协议、可以与支持蓝牙的电脑（台式、笔记本）、手机（android、IOS）通信。

该蓝牙模块通信接口为 UART，工作频段为 2.4GHz，工作电压为 1.8V-3.6V（建议 3.3V），采用 Bluetooth 4.2 BLE/Bluetooth 3.0 SPP 的蓝牙版本。它和主控 MCU 的通信是通过 UART2（NRF_RX:PA2, NRF_TX:PA3, NRF_FLOW_CTRL:PA0）。该模块内置 PCB 天线，传输距离 ≤ 40 m，发射功率最大为 5db。

该模块的串口指令集如下表所示：

注：JDY-32 模块串口发送 AT 指令务必加上\r\n

序列	指令	功能	默认
1	AT+VERSION	版本号	JDY-32-V1.0
2	AT+RST	软复位	
3	AT+DISC	AT 指令断开连接	
4	AT+MAC	查询 BLE 的 MAC 地址	
5	AT+MACS	查询 SPP 的 MAC 地址	
6	AT+BAUD	波特率	9600
7	AT+NAME	BLE 广播名设置与查询	JDY-32-LE
8	AT+NAMES	SPP 广播名设置与查询	JDY-32-SPP
9	AT+TYPE	SPP 密码连接类型	2
10	AT+PIN	SPP 连接密码	1234
11	AT+DEFAULT	恢复出厂设置	

4.6 电源接口

微型四轴板载的电源接口部分，其电路图如图 4.6.1 所示：

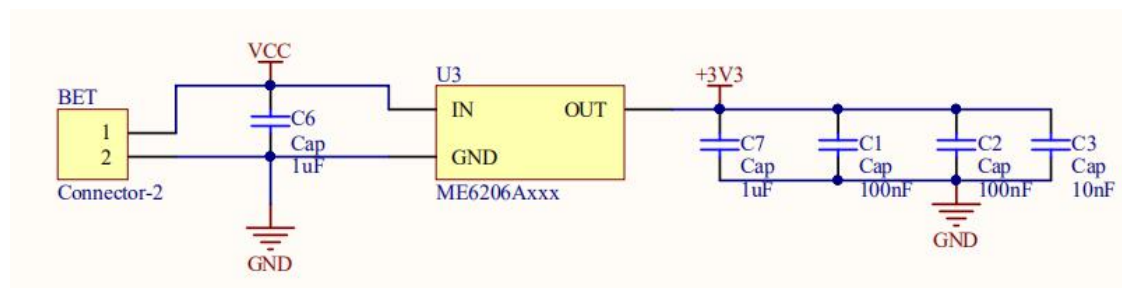


图 4.6.1 电源接口

ME6206A 是高纹波抑制率、低功耗、低压差，具有过流和短路保护的 CMOS 降压型线性稳压器（LDO）。这些器件具有很低的静态偏置电流，它们能在输入、输出电压差极小的情况下提供 300mA 的输出电流，并且仍能保持良好的调整率。该器件输入电压高达 6.0V，输出

1.5V - 5.0V。因为是线性稳压器，所以输出纹波很小，同时低压差的功能就可以保证输入电压在很低的条件下输出电压也非常稳定。

4.7 LED 接口

微型四轴板载 4 颗高亮的绿色 LED，其电路图如图 4.7.1 所示：

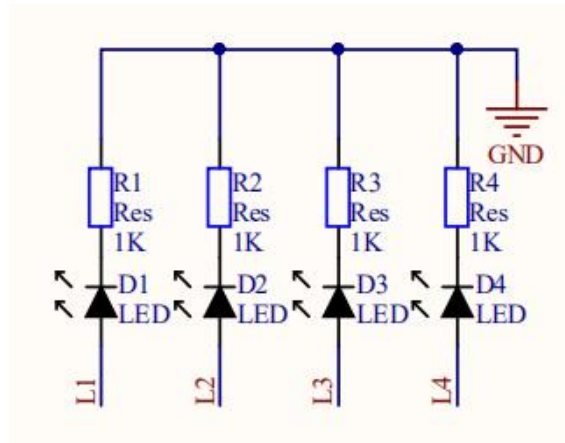


图 4.7.1 LED 接口

4.8 下载接口

微型四轴板载 SWD 仿真及下载接口，电路图如图 4.9.1 所示：

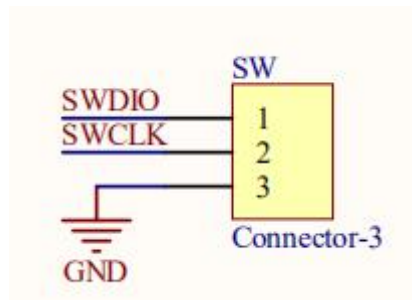


图 4.9.1 SWD 仿真下载接口

微型四轴为客户预留了代码下载和调试接口，方便客户二次开发。该接口支持 STM32 的代码下载和调试。

5. STM32 基础知识

5.1 STM32 时钟树原理

我们可以把 MCU 的运行比作人体的运行一样，人最重要的是什么？是心跳！心脏的周期性收缩将血液泵向身体各处。心脏对于人体好比时钟对于 MCU，微控制器（MCU）的运行要靠周期性的时钟脉冲来驱动，而这个脉冲的始源往往是由外部晶体振荡器提供时钟输入，最终转换为多个外部设备的周期性运作。这种时钟“能量”的传递路径犹如大树的养分由主干流向个分支，因此称为时钟树。

在 STM32 中每个外设都有其单独的时钟，在使用某个外设之前必须打开该外设的时钟，为什么要这么麻烦来设置每一个外设的时钟而不是将所有外设的时钟统一打开？因为 STM32 的外设繁多，外设的运作所需要的最佳时钟各不相同，如果所有时钟同时运行会给 MCU 带来极大的负载，所以 STM32 采取自助餐式的时钟管理方式——随用随开。

各类时钟简括：

1.HSE 时钟（高速外部时钟）：来源为外部无源晶振，通常速度 8M。由 RCC_CR 时钟控制寄存器中的 16:HSEON 控制。

2.HSI 时钟（高速内部时钟）：来源为芯片内部，大小为 8M，当 HSE 故障时，系统时钟会自动切换到 HSI，知道 HSE 启动成功，相当于 HSE 的替补。由 RCC_CR 时钟控制寄存器的位 0:HSION 控制。

3.PLLCLK（锁相环时钟）：来源为 HSI/2、HSE 经过倍频所得。由 CFGR(时钟配置寄存器)中 PLLXTPRE、PLLMUL 控制。

4.SYSCLK（系统时钟）：来源为 HSI、HSE、PLLCLK，最高速度为 72M。由 CFGR 中的 SW 控制。

5.HCLK（AHB 高速总线时钟）：来源由系统时钟分频得到，速度最高为 72M。由 CFGR 中的 HPRE 控制。

6.PCLK1（APB1 低总线时钟）：来源为 HCLK 分频得到，速度最高为 36M，为 APB1 总线上的外设提供时钟。由 RCC_CFGR 时钟配置寄存器的 PPRE1 位控制。

7.PCLK2（APB2 高总线时钟）：来源为 HCLK 分频得到，速度最高为 72M，为 APB2 总线上的外设提供时钟。由 RCC_CFGR 时钟配置寄存器的 PPRE2 位控制。

8.RTC 时钟：来源为 HSE_RTC（HSE 分频得到）、LSE、LSI，为芯片内部的 RTC 外设提供时钟。由 RCC 备份域控制寄存器 RCC_BDCR 中 RTCSEL 控制。

5.2 STM32 GPIO 原理

STM32G0 每组通用 I/O 端口包括 4 个 32 位配置寄存器（MODER、OTYPER、OSPEEDR 和 PUPDR）、2 个 32 位数据寄存器（IDR 和 ODR）、1 个 32 位置位/复位寄存器（BSRR）、1 个 32 位锁定寄存器（LCKR）和 2 个 32 位复用功能选择寄存器（AFRH 和 AFRL）等。GPIO 可以配置成以下 8 种工作模式：

浮空输入：此端口在默认情况下什么都不接，呈高阻态，这种设置在数据传输时用的比较多。

上拉输入：上拉输入模式与浮空输入模式相比，仅仅是在数据通道上部，接入了一个上拉电阻，这个上拉电阻的阻值介于 30K~50K 欧姆，CPU 可以随时在输入数据寄存器的另一端，读出 I/O 端口的电平状态。这种模式的好处在于我们什么都不输入时，由于内部上拉电阻的原因，处理器会觉得我们输入了高电平，这就避免了不确定的输入。该端口在默认情况下输入为高电平。

下拉输入：下拉输入模式与浮空输入模式相比，仅仅是在数据通道上部，接入了一个下拉电阻。与上拉输入模式类似，这种模式的好处在于外部没有输入时，由于内部下拉电阻的原因，我们的处理器会觉得我们输入了低电平。

模拟功能：STM32 的模拟输入通道的配置很简单，信号从 I/O 端口直接进入 ADC 模块。此时，所有的上拉、下拉电阻和施密特触发器，均处于断开状态，因此输入数据寄存器将不能反映端口上的电平状态，也就是说，模拟输入配置下，信号不经过输入数据寄存器，CPU 不能在输入数据寄存器上读到有效的数据。该输入模式，使我们可以获得外部的模拟信号。

开漏输出：开漏输出不可以直接输出高电平，开漏输出的输出端相当于三极管的集电极，要得到高电平状态需要上拉电阻才行。

推挽输出：推挽输出可以输出高、低电平，连接数字器件；推挽结构一般是指两个三极管分别受两个互补信号的控制，总是在一个三极管导通的时候另一个截止。高低电平由 IC 的电源决定。

开漏复用输出：GPIO 的基本功能是普通的 I/O，而 STM32 有自己的各个功能模块，这些内置外设的外部引脚是与标准 GPIO 复用的，当作为这些模块的功能引脚时就叫复用。开漏复用输出功能模式与开漏输出模式相比，不同的是输出控制电路的输入，是和片上外设的输出信号相连即与复用功能的输出端相连，此时，输出数据寄存器在输出通道被断开。

推挽复用输出：推挽复用输出功能模式与推挽输出模式相比，不同的是输出控制电路的输入，是和片上外设的输出信号相连，即与复用功能的输出端相连，而输出数据寄存器在输出通道被断开。

5.3 STM32 NVIC 原理

NVIC 的全称是 Nested vectored interrupt controller，即嵌套向量中断控制器。

通常，在接收到来自外围硬件（相对于中央处理器和内存）的异步信号，或来自软件的同步信号之后，处理器将会进行相应的硬件 / 软件处理。发出这样的信号称为进行中断请求（interrupt request, IRQ）。硬件中断导致处理器通过一个运行信息切换（context switch）来保存执行状态（以程序计数器和程序状态字等寄存器信息为主）；软件中断则通常作为 CPU 指令集中的一个指令，以可编程的方式直接指示这种运行信息切换，并将处理导向一段中断处理代码。中断在计算机多任务处理，尤其是即时系统中尤为有用。

对于 M3 和 M4 内核的 MCU，每个中断的优先级都是用寄存器中的 8 位来设置的。8 位的话就可以设置 $2^8 = 256$ 级中断，实际中用不了这么多，所以芯片厂商根据自己生产的芯片做出了调整。比如 ST 的 STM32F1xx 和 F4xx 只使用了这个 8 位中的高四位[7:4]，低四位取零，这样 $2^4 = 16$ ，只能表示 16 级中断嵌套。

对于这个 NVIC，有个重要的知识点就是优先级分组，抢占优先级和子优先级，下面就以 STM32 为例进行介绍，STM32G030 是只使用了这个 8 位寄存器的高四位[7:4]。

优先级分组	抢占优先级	子优先级	高 4 位使用情况描述
NVIC_PriorityGroup_0	0 级抢占优先级	0-15 级子优先级	0bit 用于抢占优先级 4bit 全用于子优先级
NVIC_PriorityGroup_1	0-1 级抢占优先级	0-7 级子优先级	1bit 用于抢占优先级 3bit 用于子优先级
NVIC_PriorityGroup_2	0-3 级抢占优先级	0-3 级子优先级	2bit 用于抢占优先级 2bit 用于子优先级
NVIC_PriorityGroup_3	0-7 级抢占优先级	0-1 级子优先级	3bit 用于抢占优先级 1bit 用于子优先级
NVIC_PriorityGroup_4	0-15 级抢占优先级	0 级子优先级	4bit 全用于抢占优先级 0bit 用于子优先级

图 5.3 NVIC 优先级

具有高抢占式优先级的中断可以在具有低抢占式优先级的中断服务程序执行过程中被响应，即中断嵌套，或者说高抢占式优先级的中断可以抢占低抢占式优先级的中断的执行。在抢占式优先级相同的情况下，有几个子优先级不同的中断同时到来，那么高子优先级的中断优先被响应。

在抢占式优先级相同的情况下，如果有低子优先级中断正在执行，高子优先级的中断要等待已被响应的低子优先级中断执行结束后才能得到响应，即子优先级不支持中断嵌套。Reset、NMI、Hard Fault 优先级为负数，高于普通中断优先级，且优先级不可配置。

5.4 STM32 EXTI 原理

EXTI (External interrupt/event controller) —外部中断/事件控制器，管理了控制器的 20 个中断/事件线。每个中断/事件线都对应有一个边沿检测器，可以实现输入信号的上升沿检测和下降沿的检测。EXTI 可以实现对每个中断/事件线进行单独配置，可以单独配置为中断或者事件，以及触发事件的属性。

外部中断的触发方式有：电平触发和跳沿触发。

边沿触发和电平触发基本就是触发器和锁存器的区别。触发器是边沿触发，只有当时钟上升（或下降）的一瞬间，触发器会读取并锁存输入信号。输出信号仅在时钟信号上升（或下降）的一瞬间会发生变化。锁存器是电平触发，只要使能（enable）信号处于高电平（或低电平），输出就会随着输入信号变化，直到使能信号变为低电平（或高电平）时，输出才会锁存，不再随输入变化。

STM32 上每个 IO 都可以作为外部中断输入，其中断控制器支持 22 个外部中断/时间请求。

EXTI 线 0~15：对应外部 IO 口的输入中断。

EXTI 线 16：连接到 PVD 输出。

EXTI 线 17：连接到 RTC 闹钟事件。

EXTI 线 18：连接到 USB OTG FS 唤醒事件。

EXTI 线 19: 连接到以太网唤醒事件。

EXTI 线 20: 连接到 USB OTG HS (在 FS 中配置) 唤醒事件。

EXTI 线 21: 连接到 RTC 入侵和时间戳事件。

EXTI 线 22: 连接到 RTC 唤醒事件。

每个外部中断线可以独立的配置触发方式（上升沿，下降沿或者双边沿触发），触发/屏蔽，专用的状态位。但是值得注意的是：外部 IO 口有 16 条中断线，但并不是能设置 16 个外部中断，一组 IO 口对应一根中断线，每个 IO 口都可以使用这跟中断线，但是在同一时刻，只能响应一个端口的事件触发，不能同时响应所有 GPIO 端口的事件，也就是分时复用。

IO 口外部中断在中断向量表中只分配了 7 个中断向量，也就是只能使用 7 个中断服务函数。

中断线 0 ~ 4 各对应一个中断函数

中断线 5 ~ 9 共用中断函数 EXTI9_5_IRQHandler

中断线 10 ~ 15 共用中断函数 EXTI15_10_IRQHandler

5.5 STM32 ADC 和 DMA 原理

5.5.1 STM32 ADC 原理

ADC（Analog-to-Digital Converter，模/数转换器）。也就是将模拟信号转换为数字信号进行处理，在存储或传输时，模数转换器几乎必不可少。

STM32 ADC 特点

12 位逐次逼近型的模拟数字转换器。

最多带 3 个 ADC 控制器

最多支持 18 个通道，可最多测量 16 个外部和 2 个内部信号源。

支持单次和连续转换模式

转换结束，注入转换结束，和发生模拟看门狗事件时产生中断。

通道 0 到通道 n 的自动扫描模式

自动校准

采样间隔可以按通道编程

规则通道和注入通道均有外部触发选项

转换结果支持左对齐或右对齐方式存储在 16 位数据寄存器

ADC 转换时间：最大转换速率 1us。（最大转换速度为 1MHz，在 ADCCLK=14M，采样周期为 1.5 个 ADC 时钟下得到。）

ADC 供电要求：2.4V-3.6V

ADC 输入范围：VREF- ≤ VIN ≤ VREF+

5.5.2 STM32 DMA 原理

DMA(Direct Memory Access: 直接内存存取)是一种可以大大减轻 CPU 工作量的数据转移方式。

CPU 有转移数据、计算、控制程序转移等很多功能，但其实转移数据（尤其是转移大量数据）是可以不需要 CPU 参与。比如希望外设 A 的数据拷贝到外设 B，只要给两种外设提供一条数据通路，再加上一些控制转移的部件就可以完成数据的拷贝。

DMA 就是基于以上设想设计的，它的作用就是解决大量数据转移过度消耗 CPU 资源的问题。有了 DMA 使 CPU 更专注于更加实用的操作--计算、控制等。

DMA 的作用就是实现数据的直接传输，而去掉了传统数据传输需要 CPU 寄存器参与的环节，主要涉及四种情况的数据传输（外设到内存、内存到外设、内存到内存、外设到外设），但本质上是一样的，都是从内存的某一区域传输到内存的另一区域（外设的数据寄存器本质上就是内存的一个存储单元）。

5.5.3 DMA 在 AD 转换中的作用

ADC 采集到的数据是不能直接用的，单次采集会出现不准确的数据，所以要多次采样才能减小系统误差。也可以说，单次 ADC 采集无意义。多次采样造成了数据传输量大，因此我们需要独立于 Cortex 内核的 DMA 模块，实现通信“桥梁”的作用，将外设映射的寄存器“连接”起来，这样我们就可以高速访问各个寄存器，其传输不受 CPU 的支配。

5.6 STM32 PWM 原理

脉冲宽度调制(PWM),是英文“Pulse Width Modulation”的缩写,简称脉宽调制,是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。简单一点,就是对脉冲宽度的控制,PWM 原理如图 14.1.1 所示:

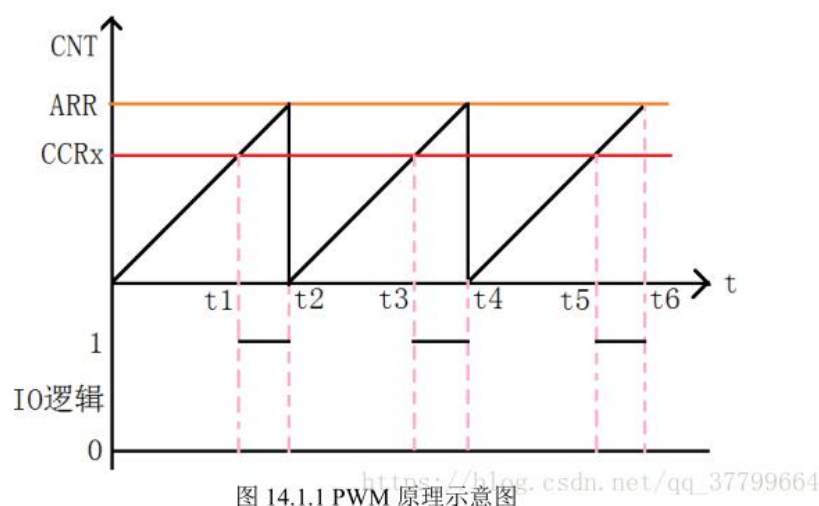


图 14.1.1 PWM 原理示意图

PWM 是一种对模拟信号电平进行数字编码的方法。通过高分辨率计数器的使用，方波的占空比被调制用来对一个具体模拟信号的电平进行编码。PWM 信号仍然是数字的，因为在给定的任何时刻，满幅值的直流供电要么完全有(ON)，要么完全无(OFF)。电压或电流源是以一种通(ON)或断(OFF)的重复脉冲序列被加到模拟负载上去的。通的时候即是直流供电被加到负载上的时候，断的时候即是供电被断开的时候。只要带宽足够，任何模拟值都可以使用 PWM 进行编码。

STM32 的定时器除了 TIM6 和 7。其他的定时器都可以用来产生 PWM 输出。其中高级定时器 TIM1 和 TIM8 可以同时产生多达 7 路的 PWM 输出。而通用定时器也能同时产生多达 4 路的 PWM 输出，这样，STM32 最多可以同时产生 30 路 PWM 输出！

6. 四轴飞行器算法讲解

6.1 姿态解算简介

姿态解算指控制器读取自身传感器数据，实时计算四轴飞行器的姿态角，比如横滚角（roll），俯仰角（pitch），偏航角（yaw）的信息，控制器根据这些信息即可计算 4 个电机的输出量，使飞行器保持平衡稳定或者保持一定的倾斜角往设定方向飞行。姿态解算是飞行器飞行的关键技术之一，解算速度和精度直接关系到飞行器飞行中的稳定性和可靠性。

姿态解算的方法有很多，比较典型的方法有扩展型卡尔曼滤波算法（EKF），互补滤波算法等。扩展型卡尔曼滤波算法具有高精度的特点，但其计算量大，并要求建立精确的动力学模型，因此并不是微型四轴飞行器的最佳选择；而互补滤波算法对系统的要求相对较低，计算量也不大，在微型四轴飞行器的姿态解算中应用最广。

6.2 飞行器姿态表示方法

飞行器姿态有多种表示方式，常见的是四元数，欧拉角，矩阵和轴角。他们各自有其自身的优点，在不同的领域使用不同的表示方式。在四轴飞行器中使用到了四元数和欧拉角。

6.2.1 欧拉角

用来确定定点转动刚体位置的 3 个一组独立角参量，由章动角 θ 、旋进角（即进动角） ψ 和自转角 j 组成，为莱昂哈德·欧拉首先提出而得名。

对于在三维空间里的一个参考系，任何坐标系的取向，都可以用三个欧拉角来表现。参考系又称为实验室参考系，是静止不动的。而坐标系则固定于刚体，随着刚体的旋转而旋转。

如图 5.2.1 所示：设定 xyz 轴为参考系的参考轴。称 xy-平面与 XY-平面的相交为交点线，用英文字母（N）代表。zxy 顺规的欧拉角可以静态地这样定义： α 是 x-轴与交点线的夹角， β 是 z-轴与 Z-轴的夹角， γ 是交点线与 X-轴的夹角。

很可惜地，对于夹角的顺序和标记，夹角的两个轴的指定，并没有任何常规。科学家对此从未达成共识。每当用到欧拉角时，我们必须明确的表示出夹角的顺序，指定其参考轴。

实际上，有许多方法可以设定两个坐标系的相对取向。欧拉角方法只是其中的一种。此外，不同的作者会用不同组合的欧拉角来描述，或用不同的名字表示同样的欧拉角。因此，使用欧拉角前，必须先做好明确的定义。

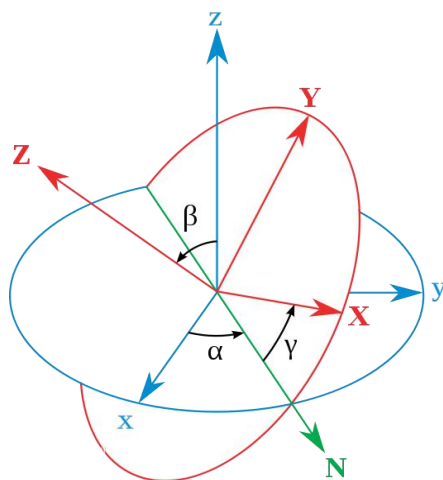


图 5.2.1 欧拉角

本小节内容部分摘自[百度百科-欧拉角](https://baike.baidu.com/item/欧拉角)

6.2.2 四元数

四元数（Quaternions）是由爱尔兰数学家哈密顿(William Rowan Hamilton, 1805-1865) 在 1843 年发明的数学概念。明确地说，四元数是复数的不可交换延伸。如把四元数的集合考虑成多维实数空间的话，四元数就代表着一个四维空间，相对于复数为二维空间。

单位四元数与欧拉角、旋转矩阵是等价的，但又不同于欧拉角表示，四元数表示法没有奇异点的问题，正因为这个优点，单位四元数在姿态估计的核心算法中非常常见。比起三维旋转矩阵四元数更能方便的给出旋转的转轴和旋转角。

关于四元数的计算，四元数可以理解为一个实数和一个向量的组合，也可以理解为四维的向量。这里用一个圈表示 q 是一个四元数（很可能不是规范的表示方式）。

$$\mathring{q} = \{w, \vec{v}\} = [w \ x \ y \ z]^T$$

四元数的长度（模）与普通向量相似。

$$|\mathring{q}| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

下面是对四元数的单位化，单位化的四元数可以表示一个旋转。

$$\hat{\mathring{q}} = \frac{\mathring{q}}{|\mathring{q}|} = \left[\frac{w}{|\mathring{q}|} \ \frac{x}{|\mathring{q}|} \ \frac{y}{|\mathring{q}|} \ \frac{z}{|\mathring{q}|} \right]^T$$

四元数相乘，旋转的组合就靠它了。

$$\mathring{q}_0 = \mathring{q}_1 \cdot \mathring{q}_2$$

$$\begin{cases} w_0 = w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ x_0 = w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2 \\ y_0 = w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2 \\ z_0 = w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2 \end{cases}$$

旋转的“轴角表示”转“四元数表示”。这里创建一个运算 $q(w, \theta)$ ，用于把绕单位向量 w 转 θ 角的旋转表示为四元数。

$$\{\hat{w}, \theta\} \rightarrow \hat{q}(\hat{w}, \theta) = \left\{ \cos\left(\frac{\theta}{2}\right), \hat{w} \cdot \sin\left(\frac{\theta}{2}\right) \right\}$$

通过 $q(w, \theta)$ ，引伸出一个更方便的运算 $q(f, t)$ 。有时需要把向量 f 的方向转到向量 t 的方向，这个运算就是生成表示对应旋转的四元数的（后面会用到）。

$$\hat{q}(\vec{f}, \vec{t}) = \hat{q}\left(\frac{\vec{f} \times \vec{t}}{|\vec{f} \times \vec{t}|}, \text{atan2}(|\vec{f} \times \vec{t}|, \vec{f} \cdot \vec{t})\right)$$

然后是“四元数表示”转“矩阵表示”。再次创造运算，用 $R(q)$ 表示四元数 q 对应的矩阵（后面用到）。

$$\hat{q} \rightarrow R(\hat{q}) = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

多个旋转的组合可以用四元数的乘法来实现。

$$R(\hat{q}_0) \cdot R(\hat{q}_1) = R(\hat{q}_0 \cdot \hat{q}_1)$$

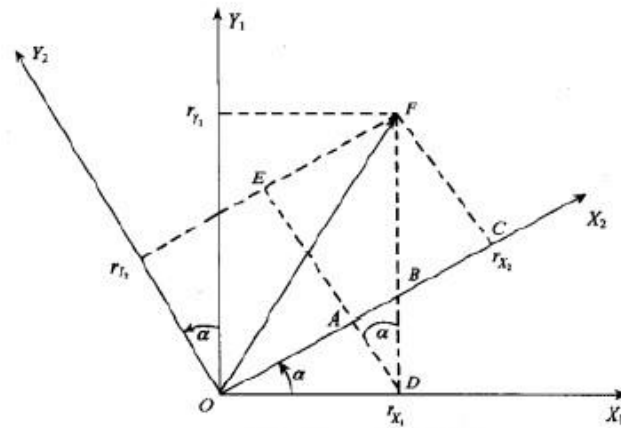
“四元数表示”转“欧拉角表示”。用于显示。

$$\begin{aligned} \hat{q} &\rightarrow \{\psi, \theta, \varphi\} \\ \begin{cases} \psi = \text{atan2}(2wz + 2xy, 1 - 2y^2 - 2z^2) \\ \theta = \arcsin(2wy - 2zx) \\ \varphi = \text{atan2}(2wx + 2yz, 1 - 2x^2 - 2y^2) \end{cases} \\ \begin{cases} \psi & : \text{偏航角(yaw), z轴} \\ \theta & : \text{俯仰角(pitch), y轴} \\ \varphi & : \text{滚转角(roll), x轴} \end{cases} \end{aligned}$$

本小节内容部分摘自[百度百科-四元数](#)

6.3 软件姿态解算

我们先来看看如何用欧拉角描述一次平面旋转(坐标变换)，如图 5.3.1 所示：



坐标系间的变换关系

图 5.3.1 欧拉角平面旋转

设坐标系绕旋转 α 角后得到坐标系,在空间中有一个矢量在坐标系中的投影为,在内的投影为由于旋转绕进行,所以 Z 坐标未变,即有:

$$\begin{aligned}
 r_{x2} &= OA + AB + BC \\
 &= OD \cos \alpha + BD \sin \alpha + BF \sin \alpha \\
 &= r_{x1} \cos \alpha + r_{y1} \sin \alpha \\
 r_{y2} &= DE - AD \\
 &= DF \cos \alpha - OD \sin \alpha \\
 &= r_{y1} \cos \alpha - r_{x1} \sin \alpha \\
 r_{z2} &= r_{z1}
 \end{aligned}$$

转换成矩阵形式表示为:

$$\begin{bmatrix} r_{x2} \\ r_{y2} \\ r_{z2} \end{bmatrix} = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix}$$

整理一下:

$$r^1 = \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix} \quad r^2 = \begin{bmatrix} r_{x2} \\ r_{y2} \\ r_{z2} \end{bmatrix} \quad \text{旋转阵 } C_1^2 = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

上面仅仅是绕一根轴的旋转, 如果三维空间中的欧拉角旋转要转三次:

$$O-X_nY_nZ_n \xrightarrow{\text{绕}Z_n\text{轴旋转}\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^b C_1^2 C_n^1 = \begin{bmatrix} \cos \gamma & 0 & -\sin \gamma \\ 0 & 1 & 0 \\ \sin \gamma & 0 & \cos \gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos \gamma \cos \psi + \sin \gamma \sin \psi \sin \theta & -\cos \gamma \sin \psi + \sin \gamma \cos \psi \sin \theta & -\sin \gamma \cos \theta \\ \sin \psi \cos \theta & \cos \psi \cos \theta & \sin \theta \\ \sin \gamma \cos \psi - \cos \gamma \sin \psi \sin \theta & -\sin \gamma \sin \psi - \cos \gamma \cos \psi \sin \theta & \cos \gamma \cos \theta \end{bmatrix}$$

上面得到了一个表示旋转的方向余弦矩阵，不过要想用欧拉角解算姿态，其实我们套用欧拉角微分方程就行了：

$$\begin{bmatrix} \dot{\gamma} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos \theta} \begin{bmatrix} \cos \theta & \sin \gamma \sin \theta & \cos \gamma \sin \theta \\ 0 & \cos \theta \cos \gamma & -\sin \gamma \cos \theta \\ 0 & \sin \gamma & \cos \gamma \cos \theta \end{bmatrix}^{-1} \bullet \begin{bmatrix} \omega_{EbX}^b \\ \omega_{EbY}^b \\ \omega_{EbZ}^b \end{bmatrix}$$

上式中左侧是本次更新后的欧拉角,对应 roll、pitch、yaw。右侧是上个周期测算出来的角度，三个角速度由直接安装在四轴飞行器的三轴陀螺仪在这个周期转动的角度，单位为弧度，计算间隔为 T 陀螺角速度，比如 0.02 秒 0.01 弧度/秒=0.0002 弧度。因此求解这个微分方程就能解算出当前的欧拉角。

前面介绍了什么是欧拉角，而且欧拉角微分方程解算姿态关系简单明了，概念直观容易理解，那么我们为什么不用欧拉角来表示旋转而要引入四元数呢？

一方面是因为欧拉角微分方程中包含了大量的三角运算，这给实时解算带来了一定的困难。而且当俯仰角为 90 度时方程式会出现神奇的“GimbalLock”。所以欧拉角方法只适用于水平姿态变化不大的情况，而不适用于全姿态飞行器的姿态确定。四元数法只求解四个未知量的线性微分方程组，计算量小，易于操作，是比较实用的工程方法。

我们知道在平面(x,y)中的旋转可以用复数来表示，同样的三维中的旋转可以用单位四元数来描述。我们来定义一个四元数：

$$\mathbf{q} = a + \vec{u} = q_0 + q_1 i + q_2 j + q_3 k$$

我们可以把它写成 $\mathbf{q} = a + \vec{u}$ ，其中 a 是标量，表示三维空间中的旋转轴。 \vec{u} 是标量，表示旋转角度。那么就是绕轴旋转 w 度，所以一个四元数可以表示一个完整的旋转。只有单位四元数才可以表示旋转，至于为什么，因为这就是四元数表示旋转的约束条件。

而刚才用欧拉角描述的方向余弦矩阵用四元数描述则为：

$$C_n^b = \begin{bmatrix} q_1^2 + q_0^2 - q_3^2 - q_2^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_2^2 - q_3^2 + q_0^2 - q_1^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_3^2 - q_2^2 - q_1^2 + q_0^2 \end{bmatrix}$$

更多参考资料请看：

秦永元《惯性导航》 袁信、郑锴的《捷联式惯性导航原理》，邓正隆的《惯性技术》

6.4 PID 控制理论

当今的闭环自动控制技术都是基于反馈的概念以减少不确定性。反馈理论的要素包括三个部分：测量、比较和执行。测量关键的是被控变量的实际值，与期望值相比较，用这个偏差来纠正系统的响应，执行调节控制。在工程实际中，应用最为广泛的调节器控制规律为比例、积分、微分控制，简称 PID 控制，又称 PID 调节。

PID 控制器（比例-积分-微分控制器）是一个在工业控制应用中常见的反馈回路部件，由比例单元 P、积分单元 I 和微分单元 D 组成。PID 控制的基础是比例控制；积分控制可消除稳态误差，但可能增加超调；微分控制可加快大惯性系统响应速度以及减弱超调趋势。这个理论和应用的关键是，做出正确的测量和比较后，如何才能更好地纠正系统。

PID（比例（proportion）、积分（integral）、导数（derivative））控制器作为最早实用化的控制器已有近百年历史，现在仍然是应用最广泛的工业控制器。PID 控制器简单易懂，使用中不需精确的系统模型等先决条件，因而成为应用最为广泛的控制器，典型的单级 PID 控制器如图 5.4.1 所示：

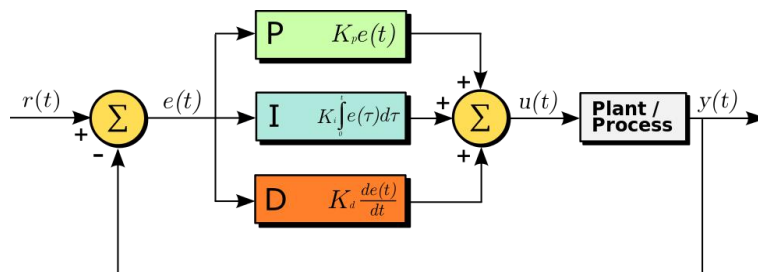


图 5.4.1 单级 PID 控制器结构框图比例（P）控制

比例控制是一种最简单的控制方式。其控制器的输出与输入误差信号成比例关系。当仅有比例控制时系统输出存在稳态误差。比例项输出：

$$P_{out} = K_p e(t)$$

不同比例增益 K_p 下，受控变量的阶跃响应（ K_i , K_d 维持稳定值）如图 5.4.2 所示：

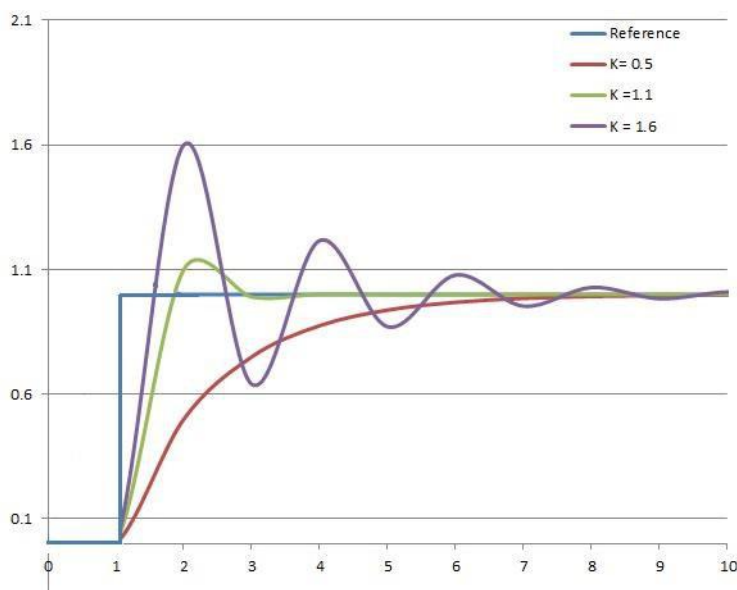


图 5.4.2 不同比例增益下，系统对阶跃信号的响应积分（I）控制

在积分控制中，控制器的输出与输入误差信号的积分成正比关系。对于只有比例控制的系统存在稳态误差，为了消除稳态误差，在控制器中必须引入“积分项”。积分项是误差对时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到等于零。因此，比例积分(PD)控制器，可以使系统在进入稳态后无稳态误差。积分项输出：

$$I_{out} = K_i \int_0^t e(\tau) d\tau$$

不同积分增益 K_i 下，受控变量的阶跃响应 (K_p, K_d 维持稳定值) 如图 5.4.3 所示：

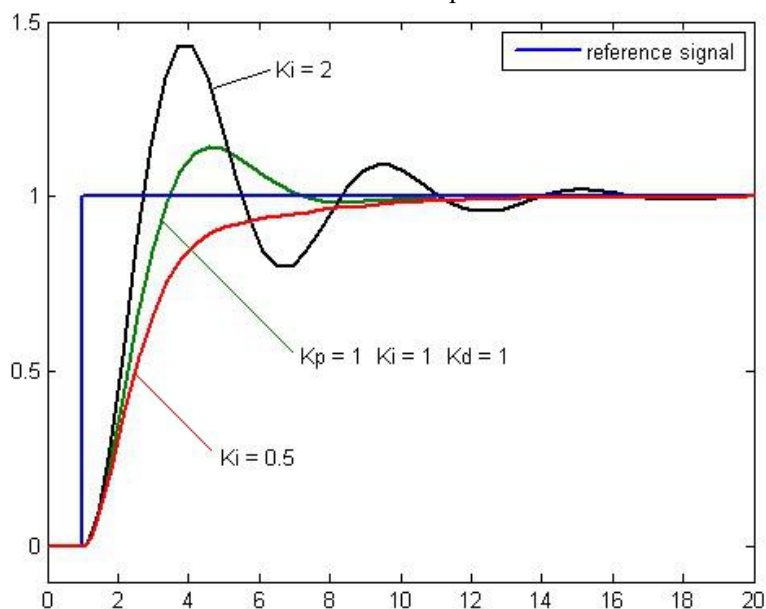


图 5.4.3 不同积分增益下，系统对阶跃信号的响应微分（D）控制

在微分控制中，控制器的输出与输入误差信号的微分成正比关系。微分调节就是偏差值的变化率。使用微分调节能够实现系统的超前控制。如果输入偏差值线性变化，则在调节器输出侧叠加一个恒定的调节量。大部分控制系统不需要调节微分时间。因为只有时间滞后的系统才需要附加这个参数。微分项输出：

$$D_{out} = K_d \frac{d}{dt} e(t)$$

不同微分增益 K_d 下，受控变量的阶跃响应（ K_p , K_i 维持稳定值）如图 5.4.4 所示：

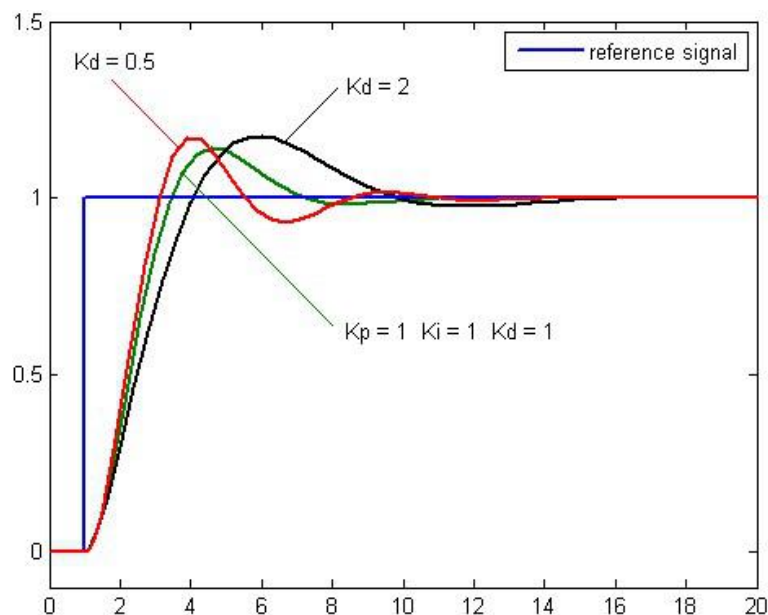


图 5.4.4 不同微分增益下，系统对阶跃信号的响应

综上可得到 PID 控制数学表达式：

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

更多内容请参考[维基百科-PID_controller](#)。

6.5 四轴常用的 2 种 PID

6.5.1 单级 PID

单级 PID 控制框图，如图 5.5.1 所示：



图 5.5.1 单级 PID 框图

期望角度就是遥控器控制飞行器的角度值，反馈当前角度就是传感器测得的飞行器角度，这里的角度指的是 Roll/Pitch/Yaw 三个角度，而且在 PID 控制计算的时候，是相互独立的。

6.5.2 串级 PID

角度单环 PID 控制算法仅仅考虑了飞行器的角度信息，如果想增加飞行器的稳定性(增加阻尼)并提高它的控制品质，我们可以进一步的控制它的角速度，于是角度/角速度-串级 PID 控制算法应运而生。在这里，相信大多数朋友已经初步了解了角度单环 PID 的原理，

但是依旧无法理解串级 PID 究竟有什么不同。其实很简单：它就是两个 PID 控制算法，只不过把他们串起来了(更精确的说是套起来)。那这么做有什么用？答案是，它增强了系统的抗干扰性(也就是增强稳定性)，因为有两个控制器控制飞行器，它会比单个控制器控制更多的变量，使得飞行器的适应能力更强。

串级 PID 控制框图，如图 5.5.2 所示：

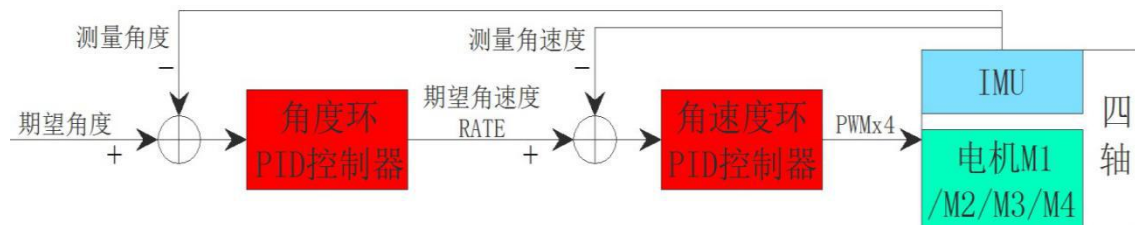


图 5.5.2 串级 PID 框图

期望角度来自遥控数据，反馈角度来自传感器，二者的偏差作为外环角度环的输入，角度环 PID 输出角速度的期望值；角速度期望值减去传感器反馈的角速度得到角速度偏差值，这个值作为内环角速度环的输入，角速度环 PID 输出姿态控制量，控制量转换为 PWM 去控制电机，从而控制四轴。

6.6 串级 PID 参数调试

内环 P：从小到大，拉动四轴越来越困难，越来越感觉到四轴在抵抗你的拉动；到比较大的数值时，四轴自己会高频震动，肉眼可见，此时拉扯它，它会快速的振荡几下，过几秒钟后稳定；继续增大，不用加人为干扰，自己发散翻机。**特别注意：只有内环 P 的时候，四轴会缓慢的往一个方向下掉，这属于正常现象。这就是系统角速度静差。**

内环 I：前述 PID 原理可以看出，积分只是用来消除静差，因此积分项系数个人觉得没必要弄的很大，因为这样做会降低系统稳定性。从小到大，四轴会定在一个位置不动，不再往下掉；继续增加 I 的值，四轴会不稳定，拉扯一下会自己发散。**特别注意：增加 I 的值，四轴的定角度能力很强，拉动他比较困难，似乎像是在钉钉子一样，但是一旦有强干扰，它就会发散。这是由于积分项太大，拉动一下积分速度快，给的补偿非常大，因此很难拉动，给人一种很稳定的错觉。**

内环 D：这里的微分项 D 为标准的 PID 原理下的微分项，即本次误差-上次误差。在角速度环中的微分就是角加速度，原本四轴的震动就比较强烈，引起陀螺的值变化较大，此时做微分就更容易引入噪声。因此一般在这里可以适当做一些滑动滤波或者 IIR 滤波。从小到大，飞机的性能没有多大改变，只是回中的时候更加平稳；继续增加 D 的值，可以肉眼看到四轴在平衡位置高频震动(或者听到电机发出滋滋的声音)。前述已经说明 D 项属于辅助性项，因此如果机架的震动较大，D 项可以忽略不加。

外环 P：当内环 PID 全部整定完成后，飞机已经可以稳定在某一位置而不动了。此时内环 P，从小到大，可以明显看到飞机从倾斜位置慢慢回中，用手拉扯它然后放手，它会慢慢回中，达到平衡位置；继续增大 P 的值，用遥控器给不同的角度给定，可以看到飞机跟踪的速度和响应越来越快；继续增加 P 的值，飞机变得十分敏感，机动性能越来越强，有发散的趋势。

本节内容来自 CSDN 网友 Nemo 之家博客[四轴 PID 讲解](#)

7. 微型四轴软件原理

7.1 STM32G030 程序解读

7.1.1 时钟树配置

本项目使用 STM32CubeMX 配置时钟树，使用 STM32CubeMX 的一个好处就是图形化配置，CubeMX 能自动检测时钟树配置的冲突并自动给出解决方案。在时钟配置方面，主要了解清楚高级外设总线（APB1、APB2）控制哪些外设。

首先配置 RCC（复位和时钟控制），选择外部高速时钟，如下图所示：

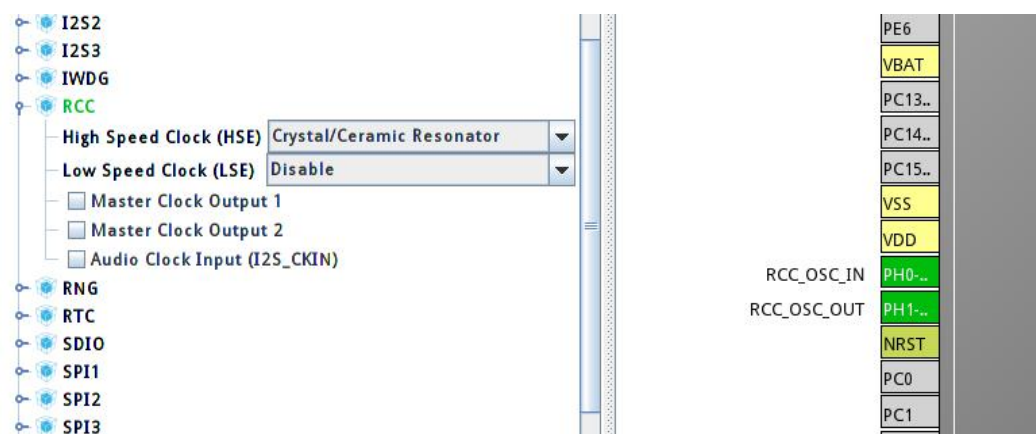


图 7.1.1 STM32CubeMX 时钟配置

通过查询 MCU 芯片手册，我们可以知道 TIM1 挂在 APB2 总线上，TIM2 挂在 APB1 总线上。

进入时钟配置，STM32G030 开发板外部晶振是 8MHz 的，在 Input frequency 输入 8，在 HCLK 那里会提示最大频率 168MHz，输入 168，然后自动寻求配置，注意第一次配置出来的还是内部时钟 HSI 的，要选择为 HSE，配置好后如下图。

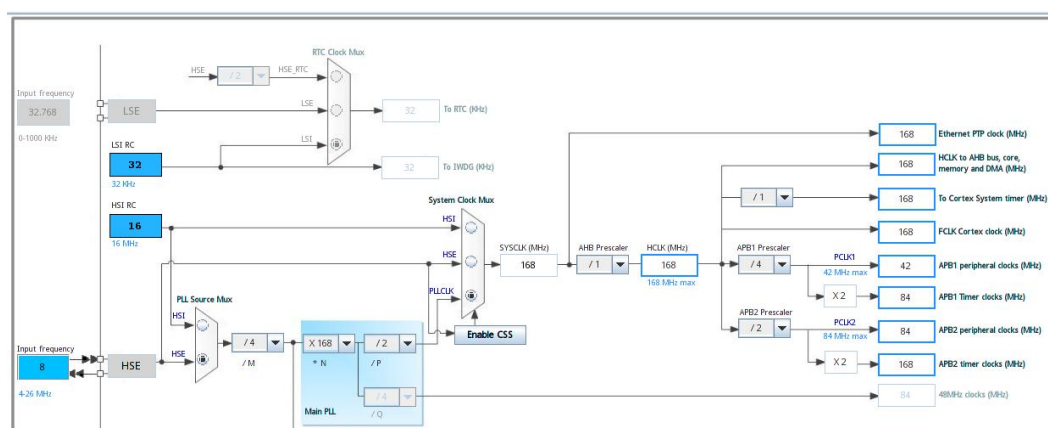


图 7.1.2 STM32CubeMX 时钟树配置

设置定时器，以 TIM1 为例，APB2 时钟频率 $168\text{MHz}=168,000,000\text{Hz}$ ，选取预分频系数 (Prescaler)（16 位存储，预分频系数最大 65535）为 16799，则定时器的时钟频率为

$168,000,000 / 16800 = 10000 \text{ Hz}$ ，频率为 1 万赫兹，选取计数周期(Counter Period, 16 位存储)为 9999，所以定时周期为 1s。

牢记如下公式：

$$\text{时钟周期} = \text{时钟频率} / ((\text{时钟预分频系数} + 1) * (\text{计数器自动重装载值} + 1))$$

在这里计数器自动重装载值(AutoReload Register)等于计数周期(Counter Period)，因为计算机计数是从 0 开始的，因此这里我们需要给位于上式分母的两个值各加一。

同理 TIM2 也是同样计算，如果 APB 的时钟频率改变，一样安装上面的方法进行设置。

7.1.2 LED 指示灯配置

通过搜索框找到引脚的位置，这里我们分别设置 PB5, PB4, PC6, PA9 为 GPIO 输出模式（即 GPIO_Output），并输入用户标签分别为 L1、L2、L3、L4（这里为了方便按照机臂 LED 顺序分别标注）。选中后，配置为输出模式后，该引脚变为绿色。

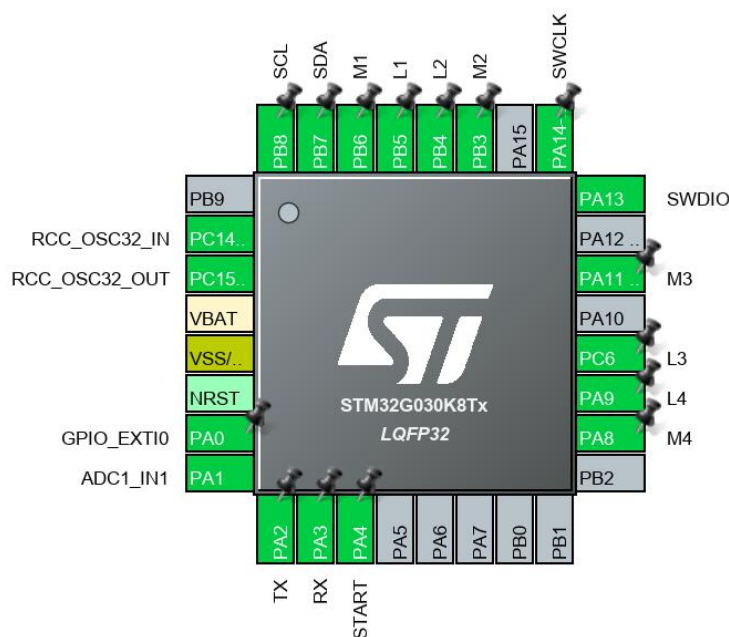


图 7.1.2 STM32CubeMX 引脚配置

接下来分别对两个 IO 进行详细配置，点击左边的 System Core，然后选择 GPIO，这时候右边可以选中具体的引脚。我们分别配置 PB5、PB4、PC6、PA9 引脚的默认输出电平、GPIO 工作模式、上下拉(这里配置为 No pull-up and no pull-down)。

Configuration								
Group By Peripherals								
GPIO ADC I2C RCC SYS TIM USART NVIC								
Search Signals								
Search (Ctrl+F)								
<input type="checkbox"/> Show only Modified Pins								
Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-do	Maximum output spe	Fast Mode	User Label	Modified
PA0	n/a	n/a	External Interrupt Mo	No pull-up and no pu	n/a	n/a		<input type="checkbox"/>
PA4	n/a	n/a	Analog mode	No pull-up and no pu	n/a	n/a	START	<input checked="" type="checkbox"/>
PA9	n/a	Low	Output Push Pull	No pull-up and no pu	Low	Disable	L4	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output Push Pull	No pull-up and no pu	Low	n/a	L2	<input checked="" type="checkbox"/>
PB5	n/a	Low	Output Push Pull	No pull-up and no pu	Low	n/a	L1	<input checked="" type="checkbox"/>
PC6	n/a	Low	Output Push Pull	No pull-up and no pu	Low	n/a	L3	<input checked="" type="checkbox"/>

图 7.1.3 GPIO 引脚配置

使用手机控制时需要 WiFi 摄像头模块的支持。有关 ATKP 格式的说明，大家可以参考《MiniFly 遥控器开发指南.pdf》。stabilizerTask 才是我们的重点，主要包括姿态解算和 PID 控制输出，下面我们就讲解一下姿态解算和 PID 算法流程。

7.2 姿态解算和 PID 算法流程图

微型四轴的姿态解算和 PID 算法流程图如图 6.3.1 所示：

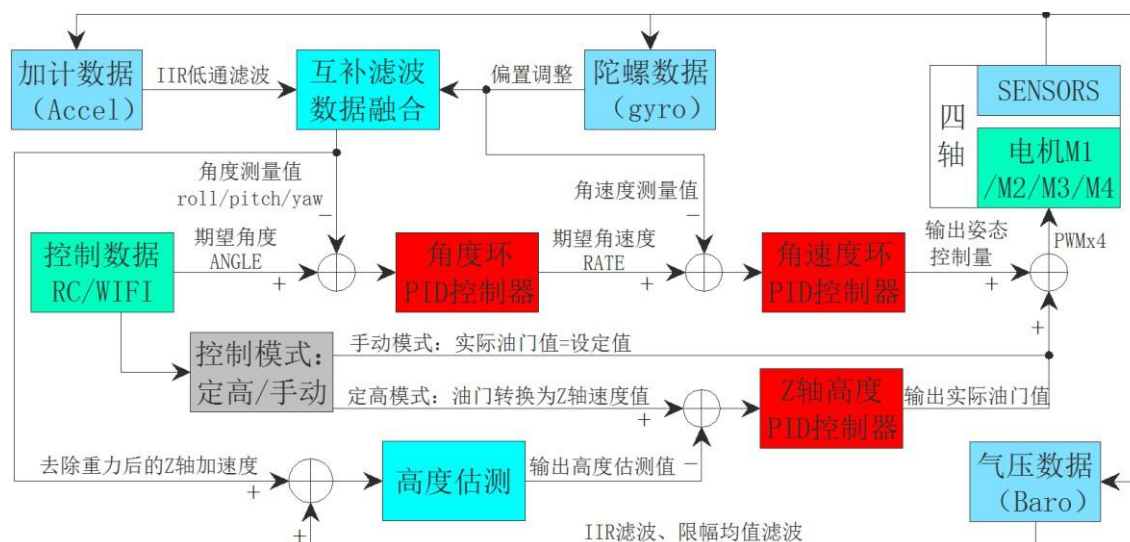


图 6.3.1 算法流程图

关于姿态解算，微型四轴采用互补滤波算法进行姿态解算，更新周期 500Hz。MCU 通过 IIC(模拟 IIC)读取加速计和陀螺仪数据寄存器，然后对加速计数据 IIR 低通滤波，对陀螺仪数据加偏置调整，然后对加计数据和陀螺数据进行融合，输出姿态数据 (roll/pitch/yaw)，互补滤波算法具体内容请看 7.4 节。

角度环 PID 控制器，更新周期 500Hz。期望角度来自手机蓝牙发送的指令，测量角度来自数据融合，期望角度减去测量角度得到偏差角度，这个偏差值作为角度环的输入，经过角度环 PID 后输出期望角速度，角度环 PID 详细内容请看 7.4 节。

角速度环 PID 控制器，更新周期 500Hz。测量角速度来自陀螺数据，期望角速度减去测量角速度得到一组偏差值，这组偏差值作为角速度环的输入，经过角速度环 PID 后输出的姿态控制量，用作控制电机，角速度环 PID 详细内容请看 7.5 节。

得到实际油门值和姿态控制量数据，我们就可以把油门值和姿态控制量数据整合，整合周期 1000Hz，然后通过控制 PWM 控制电机，从而控制四轴，关于油门值和姿态控制量数据融合详细内容请看 6.7 节。

7.3 基于四元数的姿态解算互补滤波算法

微型四轴的姿态解算算法采用四元数的互补滤波算法。在一个 IMU 系统中，一般集成有加速度计传感器、陀螺仪传感器、磁传感器等。目前常见的飞控系统中只使用一个姿态传感器芯片，这个芯片集成了加速度计、陀螺仪以及磁传感器，四轴使用一颗 6 轴传感器芯片 MPU6050。接下来说说如何把传感器数据通过四元数和欧拉角解算成飞行器的姿态。

姿态解算关键代码如下：

```
void IMUUpdate(float gx, float gy, float gz, float ax, float ay, float az) /*数据融合 互补滤波*/
{ float norm;
  float vx, vy,
  vz;
  float ex, ey,
  ez;

  //四元数积分，求得当前的姿态

  float q0_last=q0;
  float q1_last = q1;
  float q2_last = q2;
  float q3_last = q3;
```

下面把四元数换算成方向余弦中的第三行的三个元素。 $vecxZ$ 、 $vecyZ$ 、 $veczZ$ 就是上一次的欧拉角（四元数）的机体坐标参考系换算出来的重力的单位向量。

```
//把加速度计的三维向量转成单位向量
norm = invSqrt(ax*ax + ay*ay + az*az);
ax = ax * norm;
ay = ay * norm;
az = az * norm;

//估计重力加速度方向在飞行器坐标系中的表示，为四元数表示的旋转矩阵的第三行
vx = 2*(q1*q3 - q0*q2);
vy = 2*(q0*q1 + q2*q3);
vz = q0*q0 - q1*q1 - q2*q2 + q3*q3;
```

ax 、 ay 、 az 是机体坐标参照系上，加速度计测出来的重力向量，也就是实际测出来的重力向量， vx 、 vy 、 vz 是根据陀螺仪数据推算出的重力向量，它们都是机体坐标参照系上的重力向量。

那它们之间的误差向量，就是陀螺积分后的姿态和加计测出来的姿态之间的误差。向量间的误差，可以用向量叉积（也叫向量外积、叉乘）来表示， ex 、 ey 、 ez 就是两个重力向量的叉积。

```
//加速度计读取的方向与重力加速度方向的差值，用向量叉乘计算
ex = ay*vz - az*vy;
ey = az*vx - ax*vz;
ez = ax*vy - ay*vx;
```

这个叉积向量仍旧是位于机体坐标系上的，而陀螺积分误差也是在机体坐标系，而且叉积的大小与陀螺积分误差成正比，正好拿来纠正陀螺。（你可以自己拿东西想象一下）由于陀螺是对机体直接积分，所以对陀螺的纠正量会直接体现在对机体坐标系的纠正。

用叉积误差来做 PI 修正陀螺零偏

```
//误差累积，已与积分常数相乘
exInt = exInt + ex*Ki;
eyInt = eyInt + ey*Ki;
ezInt = ezInt + ez*Ki;

//用叉积误差来做 PI 修正陀螺零偏，即抵消陀螺读数中的偏移量
gx = gx + Kp*ex + exInt;
gy = gy + Kp*ey + eyInt;
gz = gz + Kp*ez + ezInt;
```

四元数微分方程，其中 $q0_last$ 、 $q1_last$ 、 $q2_last$ 、 $q3_last$ 分别为上一次的四元数值 $q0$ 、 $q1$ 、 $q2$ 、 $q3$ ， $halfT$ 为测量的半周期， gx 、 gy 、 gz 为陀螺仪角速度，以下都是已知量，这里使用了一阶毕卡算法求解四元数微分方程：

```
//一阶近似算法，四元数运动学方程的离散化形式和积分
q0 = q0_last + (-q1_last*gx - q2_last*gy - q3_last*gz)*halfT;
q1 = q1_last + (q0_last*gx + q2_last*gz - q3_last*gy)*halfT;
q2 = q2_last + (q0_last*gy - q1_last*gz + q3_last*gx)*halfT;
q3 = q3_last + (q0_last*gz + q1_last*gy - q2_last*gx)*halfT;
```

最后根据四元数方向余弦阵和欧拉角的转换关系，把四元数转换成欧拉角 $pitch$ 、 $roll$ 、 yaw 以及去除重力加速度后的 Z 轴加速度：

```
//四元数规范化
norm = invSqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
q0 = q0 * norm;
q1 = q1 * norm;
q2 = q2 * norm;
q3 = q3 * norm;

out_angle.yaw += filter_gyro.z * RawData_to_Angle * 0.001f;
//四元数转欧拉角
void Get_Eulerian_Angle(struct _out_angle *angle)
{
    angle->pitch = -atan2(2.0f*(q0*q1 + q2*q3), q0*q0 - q1*q1 - q2*q2 + q3*q3)*Radian_to_Angle;
    angle->roll = asin(2.0f*(q0*q2 - q1*q3))*Radian_to_Angle;
}
```

7.4 角度环 PID 和角速度环 PID

在讲角度 PID 之前，首先讲一下 PID 更新函数，PID 采用的标准 PID，其数学公式如下：

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

将数学公式转换为 C 代码，PID 更新函数是这样的：

```

float pidUpdate(PidObject* pid, const float error)
{
    float output;

    pid->error = error;    pid->integ
+= pid->error * pid->dt;    if (pid-
>integ > pid->iLimit)
    {
        pid->integ = pid->iLimit;
    }
    else if (pid->integ < pid->iLimitLow)
    {
        pid->integ = pid->iLimitLow;
    }
} pid->deriv = (pid->error - pid->prevError) / pid->dt;

pid->outP = pid->kp * pid->error; pid->outI =
pid->ki * pid->integ; pid->outD = pid->kd * pid-
>deriv;

output = pid->outP + pid->outI + pid->outD; pid->prevError =
pid->error;

return output;

```

PidObject 为 PID 对象结构体数据类型，第一个参数为将被更新的 PID 结构体对象，第二个参数则是偏差（期望值-测量值），积分项为偏差对时间的积分，微分项则是偏差对时间的微分，然后函数里面有三个参数 pid->kp, pid->ki, pid->kd 分别指的是该 pid 对象的比例项，积分项和微分项系数，每个 pid 对象都有属于自己的 PID 系数，PID 初始化 pid 对象的时候会设定一组默认的系数，同时这组系数是可以调整的，我们常说的 PID 参数整定，其实就是调整这组系数，让它满足你的系统。

接下来我们说说角度环 PID，其函数原型如下

```

void Control_Angle(struct _out_angle *angle, struct _Rc *rc)
{
    static struct _out_angle control_angle;

    static struct _out_angle last_angle;

    if(rc->ROLL>1490 && rc->ROLL<1510)
        rc->ROLL=1500;

    if(rc->PITCH>1490 && rc->PITCH<1510)
        rc->PITCH=1500;

    if(rc->AUX1>1495 && rc->AUX1<1505)
        rc->AUX1=1500;
}

```

```

if(rc->AUX2>1495 && rc->AUX2<1505)

    rc->AUX2=1500;

control_angle.roll = angle->roll - (rc->ROLL -1500)/13.0f + (rc->AUX2 -1500)/100.0f;
control_angle.pitch = angle->pitch - (rc->PITCH -1500)/13.0f - (rc->AUX1 -1500)/100.0f;

if(control_angle.roll > angle_max)    //ROLL

    roll.integral += angle_max;

if(control_angle.roll < -angle_max)

    roll.integral += -angle_max;

else

    roll.integral += control_angle.roll;

if(roll.integral > angle_integral_max)

    roll.integral = angle_integral_max;

if(roll.integral < -angle_integral_max)

    roll.integral = -angle_integral_max;

if(control_angle.pitch > angle_max)//PITCH

    pitch.integral += angle_max;

if(control_angle.pitch < -angle_max)

    pitch.integral += -angle_max;

else

    pitch.integral += control_angle.pitch;

if(pitch.integral > angle_integral_max)

    pitch.integral = angle_integral_max;

if(pitch.integral < -angle_integral_max)

    pitch.integral = -angle_integral_max;

if(rc->THROTTLE<1200)//油门较小时，积分清零

{

    roll.integral = 0;

    pitch.integral = 0;

}

roll.output = roll.kp *control_angle.roll + roll.ki *roll.integral + roll.kd *(control_angle.roll -
last_angle.roll);

pitch.output = pitch.kp*control_angle.pitch + pitch.ki*pitch.integral + pitch.kd*(control_angle.pitch-
last_angle.pitch);

last_angle.roll=control_angle.roll;

```

```

    last_angle.pitch=control_angle.pitch;
}

```

attitude_t 是一个姿态数据结构类型，函数参数 actualAngle 是一个结构体指针，指向实际角度结构体变量（数据融合输出值）state->attitude, desiredAngle 指向期望角度结构体变量（设置的角度）attitudeDesired, outDesiredRate 则是角度环的输出，指向期望角速度结构体变量 rateDesired。此处 PID 更新的分别是角度环的 3 个 pid 对象结构体，输入偏差为期望角度减去测量角度（actualAngle- desiredAngle）。YawError 处理那段，是为了快速计算并限制其范围在-180° 到+180°。

然后是角速度环 PID，其函数原型如下：

```

/* 角速度环 PID */
void Control_Gyro(struct _SI_float *gyro,struct _Rc *rc,uint8_t Lock)
{
    static struct _out_angle control_gyro;
    static struct _out_angle last_gyro;
    int16_t throttle1,throttle2,throttle3,throttle4;
    if(rc->YAW>1400 && rc->YAW<1600)
        rc->YAW=1500;
    if(rc->AUX3>1495 && rc->AUX3<1505)
        rc->AUX3=1500;
    control_gyro.roll = -roll.output - gyro->y*Radian_to_Angle;
    control_gyro.pitch = pitch.output - gyro->x*Radian_to_Angle;
    if(rc->AUX4 & Lock_Mode)
        control_gyro.yaw = - gyro->z*Radian_to_Angle - (rc->AUX3 -1500)/100.0f;//锁尾模式
    else
        control_gyro.yaw = -(rc->YAW-1500)/2.0f - gyro->z*Radian_to_Angle + (rc->AUX3 -1500)/50.0f;//非
锁尾模式

    if(control_gyro.roll > gyro_max) //GYRO_ROLL
        gyro_roll.integral += gyro_max;
    if(control_gyro.roll < -gyro_max)
        gyro_roll.integral += -gyro_max;
    else
        gyro_roll.integral += control_gyro.roll;
    if(gyro_roll.integral > gyro_integral_max)
        gyro_roll.integral = gyro_integral_max;
    if(gyro_roll.integral < -gyro_integral_max)

```



```

        gyro_roll.integral = -gyro_integral_max;

    if(control_gyro.pitch > gyro_max)//GYRO_PITCH

        gyro_pitch.integral += gyro_max;

    if(control_gyro.pitch < -gyro_max)

        gyro_pitch.integral += -gyro_max;

    else

        gyro_pitch.integral += control_gyro.pitch;

    if(gyro_pitch.integral > gyro_integral_max)

        gyro_pitch.integral = gyro_integral_max;

    if(gyro_pitch.integral < -gyro_integral_max)

        gyro_pitch.integral = -gyro_integral_max;

    gyro_yaw.integral += control_gyro.yaw;

    if(gyro_yaw.integral > gyro_integral_max)

        gyro_yaw.integral = gyro_integral_max;

    if(gyro_yaw.integral < -gyro_integral_max)

        gyro_yaw.integral = -gyro_integral_max;

    if(rc->THROTTLE<1200)//油门较小时，积分清零

    {
        gyro_yaw.integral = 0;
    }

    gyro_roll.output = gyro_roll.kp *control_gyro.roll + gyro_roll.ki *gyro_roll.integral + gyro_roll.kd
*(control_gyro.roll -last_gyro.roll );

    gyro_pitch.output = gyro_pitch.kp*control_gyro.pitch + gyro_pitch.ki*gyro_pitch.integral +
gyro_pitch.kd*(control_gyro.pitch-last_gyro.pitch);

    gyro_yaw.output = gyro_yaw.kp *control_gyro.yaw + gyro_yaw.ki *gyro_yaw.integral + gyro_yaw.kd
*(control_gyro.yaw -last_gyro.yaw );

    last_gyro.roll =control_gyro.roll;

    last_gyro.pitch=control_gyro.pitch;

    last_gyro.yaw =control_gyro.yaw;

    if(rc->THROTTLE>1200 && Lock==0){

        throttle1 = rc->THROTTLE - 1050 + gyro_pitch.output + gyro_roll.output - gyro_yaw.output;

        throttle2 = rc->THROTTLE - 1050 + gyro_pitch.output - gyro_roll.output + gyro_yaw.output;

        throttle3 = rc->THROTTLE - 1050 - gyro_pitch.output + gyro_roll.output + gyro_yaw.output;

        throttle4 = rc->THROTTLE - 1050 - gyro_pitch.output - gyro_roll.output - gyro_yaw.output;

    }

    else{

        throttle1=0;

```

```

        throttle2=0;

        throttle3=0;

        throttle4=0;

    }

    Motor_Out(throttle1,throttle2,throttle3,throttle4);
}

```

Axis3f 是一个 3 轴数据结构体类型，control_t 是控制数据结构体类型。参数 actualRate 指向 3 轴陀螺结构体变量 sensors->gyro，desiredRate 则指向角度环 PID 的输出 rateDesired，control_t 则指向 control 结构体变量，control 结构体包含了角速度环的输出数据—姿态控制量数据，这个数据用作控制电机。此处 PID 更新的分别是角速度环的 3 个 pid 对象结构体，输入偏差为期望角速度减去测量角速度（actualRate- desiredRate），pidOutLimit 函数用作限制姿态控制量的调整范围（-32768~+32767），防止调整量过大，难以控制。

7.5 姿态控制量和油门值整合

微型四轴为 X 模式飞行，电机转向和姿态解算正方向（箭头指示正方向）如图 6.7.1 所示：

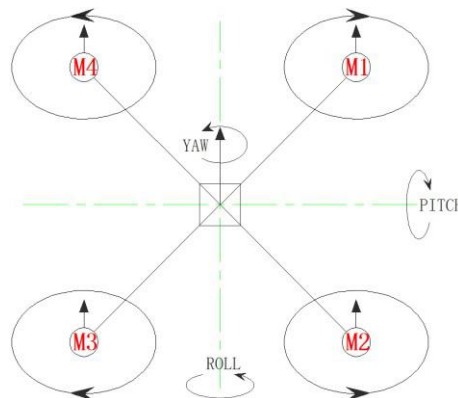


图 6.7.1 电机转向和姿态解算方向

rc->THROTTLE 为油门控制量，这个值增大四轴升高，减小则下降。rc->ROLL，rc->PITCH，rc->YAW 为 PID 输出的姿态控制量。油门控制量和姿态控制量整合后控制电机，整合代码在 power_control.c 文件的函数 powerControl() 中实现，代码如下：

```

void RC_Limit(struct _Rc *rc)
{
    rc->THROTTLE = (rc->THROTTLE<=1000)?1000:rc->THROTTLE;

    rc->THROTTLE = (rc->THROTTLE>=2000)?2000:rc->THROTTLE;

    rc->PITCH = (rc->PITCH<=1000)?1000:rc->PITCH;

    rc->PITCH = (rc->PITCH>=2000)?2000:rc->PITCH;

    rc->ROLL = (rc->ROLL<=1000)?1000:rc->ROLL;

    rc->ROLL = (rc->ROLL>=2000)?2000:rc->ROLL;
}

```

```

rc->YAW = (rc->YAW<=1000)?1000:rc->YAW;
rc->YAW = (rc->YAW>=2000)?2000:rc->YAW;

rc->AUX1 = (rc->AUX1<=1000)?1000:rc->AUX1;
rc->AUX1 = (rc->AUX1>=2000)?2000:rc->AUX1;

rc->AUX2 = (rc->AUX2<=1000)?1000:rc->AUX2;
rc->AUX2 = (rc->AUX2>=2000)?2000:rc->AUX2;

rc->AUX3 = (rc->AUX3<=1000)?1000:rc->AUX3;
rc->AUX3 = (rc->AUX3>=2000)?2000:rc->AUX3;
}

```

如图 6.7.1 所示，微型四轴右上角为 M1 机臂，在代码中对应电机为 motorPWM.m1，然后顺时针依次为 M2，M3，M4。如果四轴受外力干扰导致 PITCH 轴向前倾斜 3 度，那么 M1 和 M4 需要提高升力，M2 和 M3 减小升力来恢复平衡状态，所以有以下规则：

Roll 方向（向右为正）旋转，为了恢复平衡，则 M3 和 M4 同侧出力，M1 和 M2 反向出力（m1 和 m2 的 Roll 为-，m3 和 m4 的 Roll 为+）；

Pitch 方向（向前为正）旋转，为了恢复平衡，则 M2 和 M3 同侧出力，M1 和 M4 反向出力（m1 和 m4 的 Pitch 为-，m2 和 m3 的 Pitch 为+）；

Yaw 方向（逆时针为正）旋转，为了恢复平衡，则 M1 和 M3 同侧出力，M2 和 M4 反向出力，（m2 和 m4 的 Yaw 为-，m1 和 m3 的 Yaw 为+）；

bool 型变量 motorSetEnable，为 true，使能手动设置电机占空比，这样可以方便单独调试某几个电机，默认不使能。

motorsSetRatio() 当然就是设定对应电机定时器通道占空比的函数了，设定的占空比作用到 MOS 管，然后控制电机，从而控制四轴。

7.6 蓝牙无线串口通讯协议

APP 和飞控之间的通信协议使用了 MWC 飞控协议(MSP, Multiwii Serial Protocol)，详见 [MSP 协议格式](#)。

MWC 协议规定了飞控和上位机（或者手机 APP）信息交流的基本格式。

MWC 具体实现，可以查看 Android APP 源代码中的 Protocol.java 文件。

串口通信采用 UART 通讯方式，数组名为 Bluetooth_RXDATA，一次发送八个字节，低八位读取。若发送的位数超过 8 位，则使用两个数组元素来存放一个通道的数据。

通信协议格式如下所示：

串口数组	作用
Bluetooth_RXDATA[0]	数据校验，内容为\$

Bluetooth_RXDATA[1]	数据校验，内容为 M
Bluetooth_RXDATA[2]	发送标志， 若为四轴发送给上位机，则内容为> 若为上位机发送给四轴，则内容为<
Bluetooth_RXDATA[3]	串口长度，用于校验
Bluetooth_RXDATA[4]	功能帧标志
Bluetooth_RXDATA[5] ~ Bluetooth_RXDATA[20]	取决于功能帧的标志

7.7 手机控制软件框架

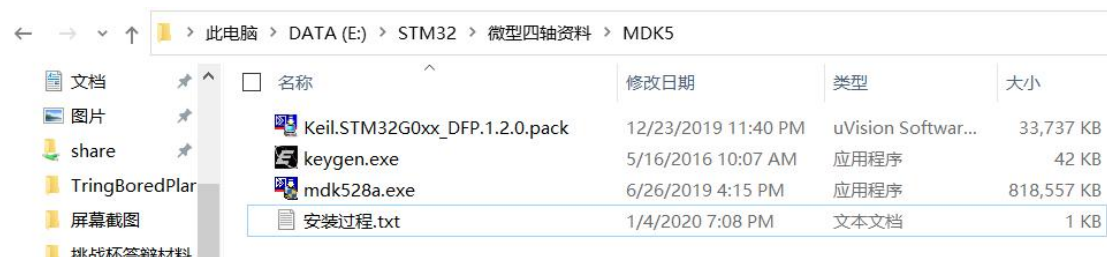
Android 手机控制软件采用 Android Studio 开发，并采用最新版本 29 版本的 Android SDK，操作系统最高支持 Android 9.0，最低支持 Android 4.3.0。BLE 技术需要 Android 4.3 及以上系统支持。源码采用 Java 语言编写而成，通过使用 Bluetooth BLE4.0 协议，可通过 Android 手机实现近距离控制配套的微型四轴。

源码的主要 java 代码文件在

8. 微型四轴二次开发

8.1 MDK 开发环境搭建

我们微型四轴源码工程是基于 MDK5 开发，下面我们先安装这个软件。MDK5 安装源文件在微型四轴资料中有提供，如下图 7.1.1 所示。



名称	修改日期	类型	大小
Keil.STM32G0xx_DFP.1.2.0.pack	12/23/2019 11:40 PM	uVision Softwar...	33,737 KB
keygen.exe	5/16/2016 10:07 AM	应用程序	42 KB
mdk528a.exe	6/26/2019 4:15 PM	应用程序	818,557 KB
安装过程.txt	1/4/2020 7:08 PM	文本文档	1 KB

图 7.1.1 MDK 安装源文件目录

MDK 安装过程就不详细说明，安装前先看“安装过程.txt”。安装完 MDK 软件之后还需要安装芯片包（图 7.1.1 中 .pack 文件），直接双击芯片包文件即可安装（先安装 MDK 软件）。所有都安装完成之后打开微型四轴源码工程如下图 7.1.2 所示。

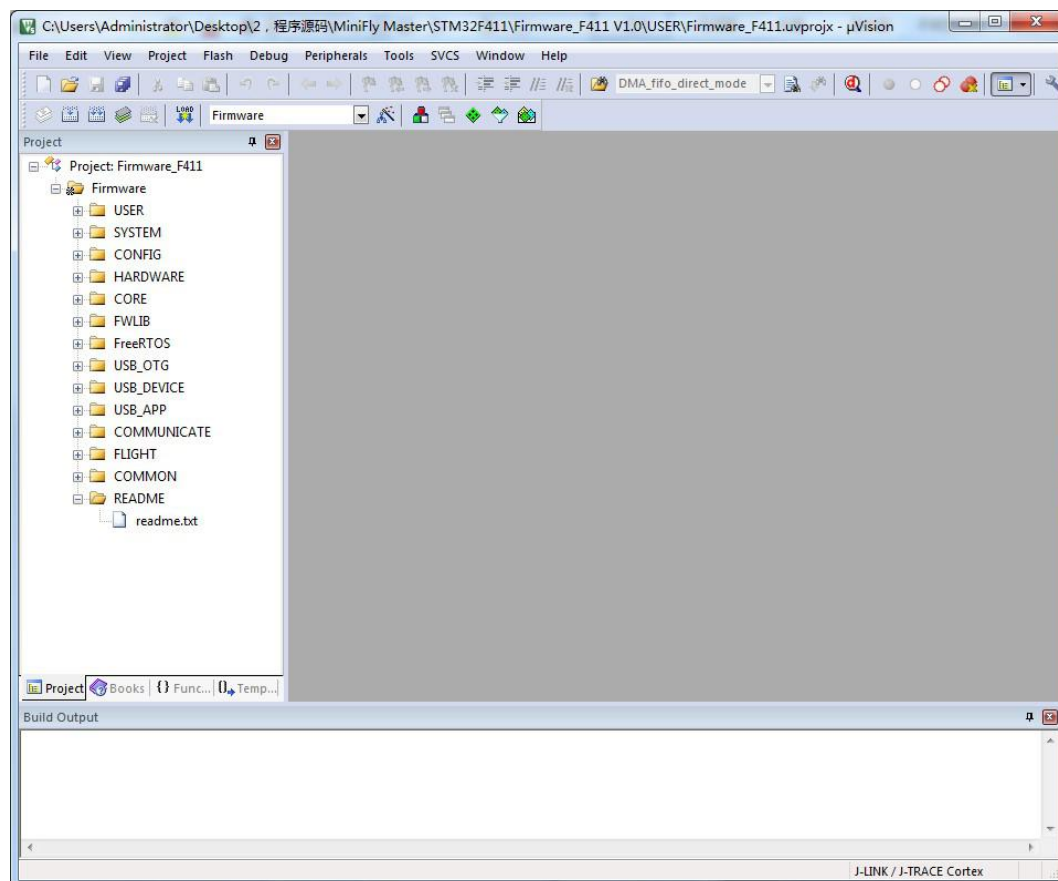


图 7.1.2 源码工程示图

8.2 CubeMX 开发环境搭建

STM32CubeMX 是 STM32 芯片图形化配置工具，允许用户使用图形化向导生成 C 初始化代码，可以大大减轻开发工作，时间和费用。STM32CubeMX 几乎覆盖了 STM32 全系列芯片。

目前，ST 公司已基于 STM32CubeMX 的基础上，融入 eclipse 开发生态环境，通过 arm 交叉编译器设计出新一代嵌入式开发 IDE——STM32CubeIDE。

首先安装 Java 运行环境，双击 JavaSetup8u151 在线下载安装包，全部点击默认安装即可。（需要注意，STM32CubeMX 的 Java 运行环境版本必须是 V1.7 及以上，如果你的电脑安装过 V1.7 以下版本，请先删掉后重新安装最新版本），然后安装 STM32CubeMX，直接双击 STM32CubeMX 安装包，默认安装在 C 盘，可以改变安装路径。安装之后打开该软件，接下来我们安装 STM32 库，点击 Help->Install New Libraries 或者按快捷键 ALT + U。这里我们推荐使用离线下载（在线下载速度较慢而且不支持断点续传功能），也可使用我们提供的 STM32G030 系列的库函数。注意：固件库的固件要与开发板的主控型号对应，否则无法使用。

8.3 固件下载

首先我们给下载器安装驱动。下载器驱动文件路径：“微型四轴资料\ST LINK 驱动及教程\STLINK 驱动”。驱动安装过程这里就不详细说明，安装完成后将 USB 线连接好下载

器并插入电脑，然后打开设备管理器，在通用串行总线控制器中即可找到 ST-Link driver，如下图 7.2.1 所示。



图 7.2.1 下载器驱动

然后将下载器连接到四轴，下载器通过 4Pin 线连接到四轴，如下图 7.2.2 所示。

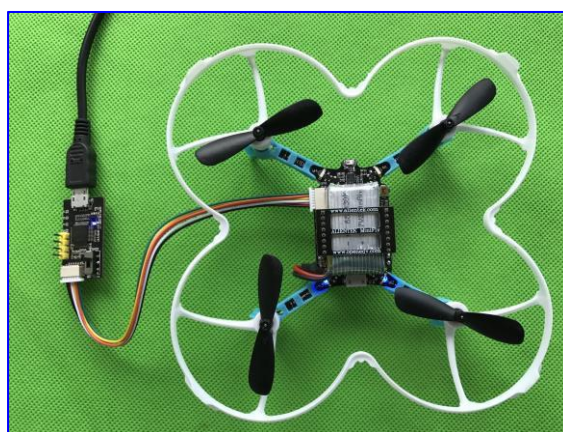


图 7.2.2 下载器连接图

然后打开程序源码。驱动安装完成之后，我来看看程序源码文件。程序源码目录树如下



图 7.2.3 程序源码目录树

编译固件并下载。解压 CubeMX_Code.zip 并打开程序工程，点击编译，等待编译完成再点击下载。操作步骤如下图 7.2.5 所示。下载完成后如图 7.2.6 所示。所有程序源码下载都可

以按照此方法执行，这里就不重复。

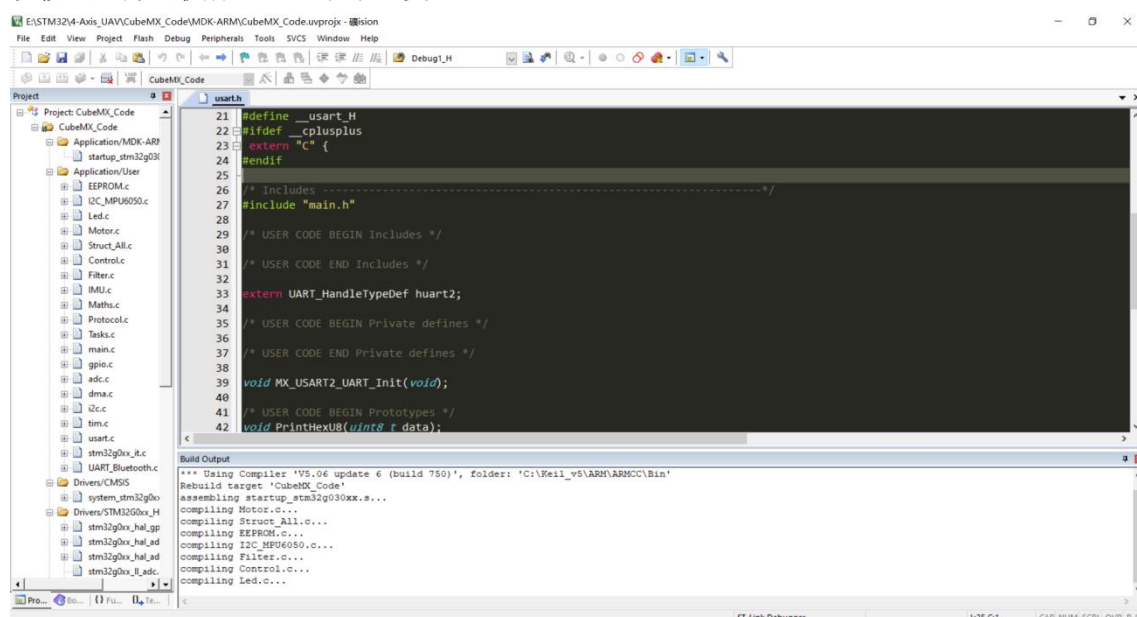


图 7.2.4 固件下载操作步骤示意图

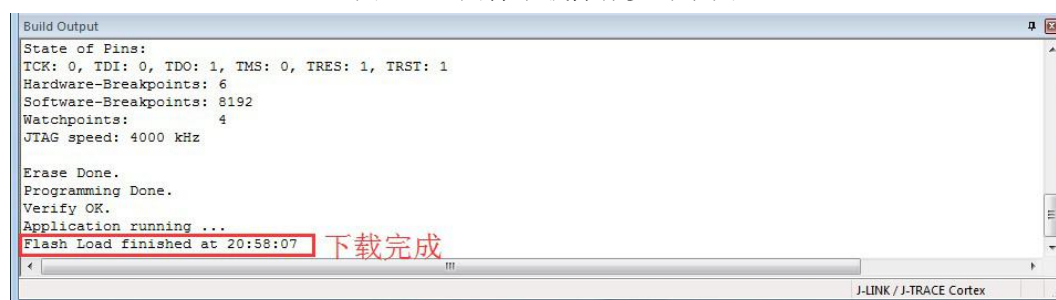


图 7.2.5 程序下载完成示意图

提示：下载时，芯片必须是上电状态。

8.4 安卓地面站使用

我们微型四轴兼容了功能强大的 MWC 地面站，该软件路径：“微型四轴资料\匿名四轴上位机”。MWC 地面站支持很多通信方式，这里我们使用的是蓝牙无线串口方式通信。下面介绍怎么使用匿名科创地面站查看飞控姿态，显示数据波形及 PID 调试。

第一步，打开四轴飞行器，使用杜邦线将 USB-TTL 串口连接至蓝牙无线传输模块，而后再将 USB-TTL 无线串口模块连接至电脑端，在设备管理器中找到端口号。注意：使用遥控器和地面站通信时，只有四轴是开机状态才有数据上传。

第二步，打开软件选择串口通信

图 7.3.1 串口通信设置图

打开软件后点击左边一列图标中的程序设置，即可显示程序设置界面，如图 7.3.1 所示。我们选择通信连接方式为 COM 方式，然后选择端口号，设置波特率为 500000，最后点击右下角的打开连接，即可通信。当有数据上传时，RX:显示接收到数据的个数。

第三步，查看飞控状态

图 7.3.2 飞控状态图

点击左边一列图标的飞控状态即可显示飞控状态界面，如图 7.3.2 所示。飞控状态显示主要包括当前四轴姿态（PIT\ROL\YAW）、接收机（遥控器控制数据）、电机输出（PWM）、传感器原始数据（ACC\GYR\MAG）、气压高度（单位 cm）、电池电压。飞行模式和 GPS 信息功能未使用。

第四步，查看数据波形

图 7.3.3 数据波形图

点击左边一列图标的数据波形即可显示数据波形界面，如图 7.3.3 所示。首先选中我们要显示的数据，再点击飞控数据即可显示数据波形。

第五步，PID 调试

图 7.3.4 PID 调试图

点击左边一列图标的飞控设置即可显示飞控设置界面，如图 7.3.4 所示。飞控设置包括 PID 设置、飞行模式设置、传感器校准，这里我们只使用了 PID 设置的功能。读取飞控：读取四轴当前 PID 参数。写入飞控：将显示的所有 PID 参数写入到飞控。恢复默认值：将四轴 PID 参数恢复成出厂默认值。这里我们总共使用了 7 组 PID，分别是 roll 速率、pitch 速率、yaw 速率、roll 角度、pitch 角度、yaw 角度、高度位置。

注意：显示的 PID 数值是实际数值的 10 倍。由于地面站 PID 数据传输只能是 int16 类型不能浮点型，所以将实际数值乘以 10 再上传，同时写入时也除以了 10。

8.5 微型四轴 PID 调试

众所周知 PID 调试是四轴最难的一部分，也是最核心的一部分，PID 参数是否合理直接影响四轴飞行的效果。下面就介绍一下我们微型四轴的 PID 调试方法。

第一步，修改 G030 固件代码

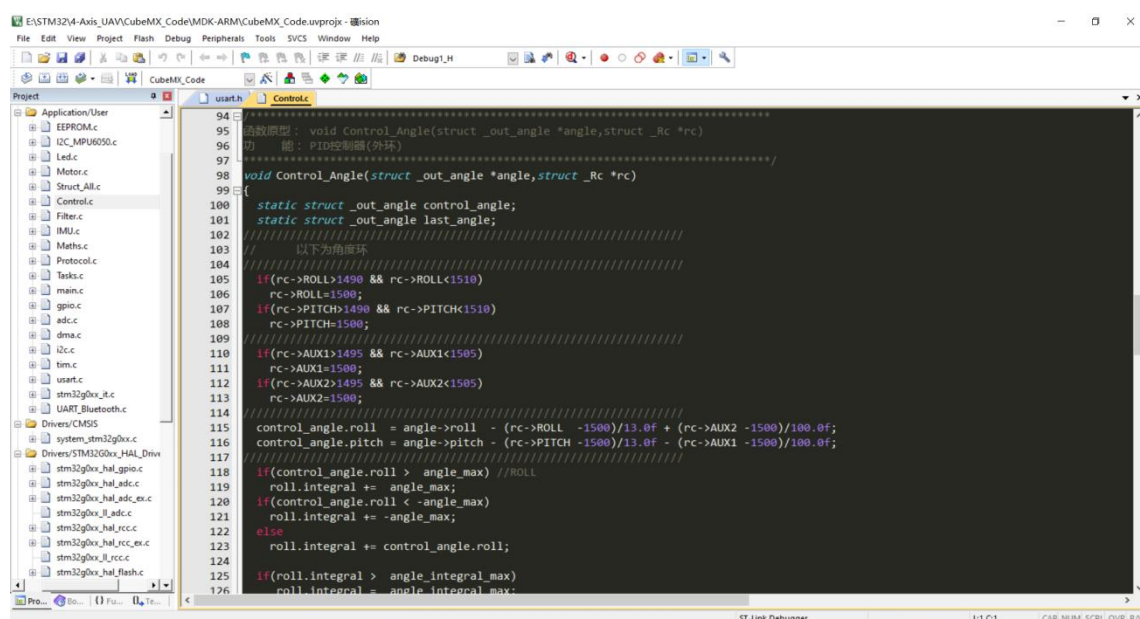


图 7.4.1 PID 控制函数示意图

打开 CubeMX_Code 源码文件夹，打开 MDK-ARM/CubeMX_Code.uvprojx 工程文件，PID 控制器的内外环函数位于 Application/User 文件夹下的 Control.c，如上图所示。修改后编译固件并下载到四轴。

第二步，搭建四轴调试平台



图 7.4.2 MiniFly 四轴调试平台搭建示图

如上 7.4.2 所示，使用两条细绳一端拴住微型四轴用于固定电池的排针，另一端拴住椅子的两端扶手，并保证微型四轴可以 360 度旋转不会碰到椅子面。绳子拴住的排针最好是四轴中间位置的排针，排针不在机身中间，所以机身中间的排针不一定是排针中间的那一根。最好选择扶手较高的椅子，这样机身离椅子面足够的距离排风。**注意：四轴开机后一直晃动陀螺仪是校准不通过的，需要使用物体卡住使其静止等待校准通过。**

第三步，遥控器连接上位机

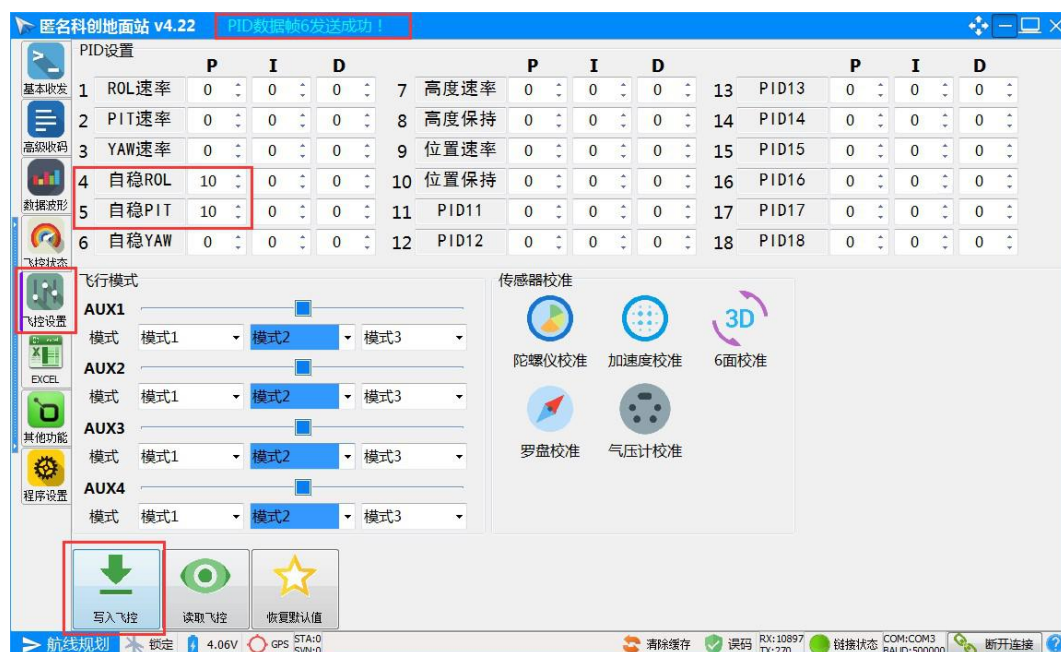


图 7.4.3 PID 初始设置示意图

使用 USB 线将遥控器连接到上位机，点击右下角打开连接，然后并打开飞控设置，设置自稳 ROL 和自稳 PIT 的 P 为 10，点击写入飞控，上方会显示“PID 数据 6”发送成功，如图 7.4.3 所示。串级 PID 调试一般先调内环（速率环），所以先将外环（角度环）PID 的 P 设置为 1，I 和 D 设置为 0。**注意：上位机显示是放大了 10 倍，后面的调试值均是上位机显示的值。**

第四步，调内环 pitch 和 roll 的 P

设置遥控器为手动模式，然后解锁长按 KEY_R 键进入调试界面。按照第三步操作后，我们给内环 ROL 速率和 PIT 速率的 P 同时设置一个值，然后推动油门至 50 左右观察并记录现象。下面是测试我们不同 P 的值和对应的现象：

P = 10 时，四轴乱晃无法固定于某一个角度，用手轻轻干扰四轴会绕绳子旋转，说明 P 值太小，力度不够。

P = 100 时，四轴晃动幅度减小但还是无法固定于某一个角度，用手轻轻干扰四轴还会绕绳子旋转，说明 P 值还是太小，力度不够。

P = 1000 时，四轴已经不晃动了，用手轻轻干扰明显感觉有回复力，说明 P 值已经慢慢趋向理想值。

P = 5000 时，四轴机身晃动严重已经震荡，说明 P 值已经过大，由此可确定理想的 P 值应该在 1000~5000 范围内。

P = 2500 时，四轴不晃动并且能固定在某一个角度了，机身也不震荡，用手去干扰能感觉到明显的回复力。说明 P 值已经很接近理想值了。

P = 3500 时，四轴机身轻微晃动说明已经轻微震荡，用手去干扰能感觉到很强的回复力。说明 P 值已经稍大于理想值了。

P = 3200 时，四轴没有晃动，用手去干扰能感觉到很强的回复力并有点晃动。说明 P 值就是我们的理想值了，晃动问题后期可以通过加点 D 抑制。

第五步，调内环 pitch 和 roll 的 D

根据第四步内环 P 已经调节好,但如果四轴受到干扰则会震荡,说明稳定性不好,这时我们就需要通过加 D 来增强稳定性。按照第四步的操作方法,设置 P 值为 3200,测试不同的 D 值对应的现象:

$D=10$ 时,用手去干扰四轴,能感觉到很强的回复力并有点晃动,说明 D 值有点小,抑制不了干扰。

$D=100$ 时,用手去干扰四轴,能感觉没有晃动了但四轴需要长时间才能回复到水平,说明 D 值太大了,抑制了 P 的调整。

$D=50$ 时,用手去干扰四轴,能感觉没有晃动了四轴也能较快回复到水平,说明 D 值接近理想值了。

就这样我们暂时把 D 值调好了,因为后面整机试飞时可能还会调节 D 值,这里先不用精确调节至某一个值。

第六步,调外环 pitch 和 roll 的 P

内环调好了,下面我们开始调外环。外环的主要作用是控制四轴姿态响应快慢,下面我们测试不同的 P 值,观察四轴的响应速度。按照前面操作方法,我们测试 P 值和现象如下:

$P=10$ 时,方向摇杆往前打到最大,发现四轴慢慢倾斜,最终达到设定角度。响应速度不够快,说明 P 太小了。

$P=100$ 时,方向摇杆往前打到最大,四轴瞬间倾斜了达到设定角度,而且力量很大。说明 P 值太大了。

$P=50$ 时,方向摇杆往前打到最大,四轴较快的达到设定角度,不是很快也不是很慢。说明 P 值是我们理想值了。

第七步,调内环 yaw 的 PID

调节内环 yaw 的 PID 前需将第一步注释掉的 `ENABLE_PID_TUNING` 宏定义重新注释回来,并且编译下载至四轴。然后设置 yaw 外环 P 为 10,内环 P 设定为某一个值,慢慢推油门让四轴起飞至水平,推油门摇杆时不要向左或向右打到航向角。飞至水平后,推油门至 50,然后用手轻轻向左边或向右拨动四轴,感受四轴的回复力,并且听电机的声音。下面是我们设置不同内环 P 的值时对应的现象:

当 $P=10$ 时,四轴几乎没有回复力,受干扰后左右摆晃,电机声音差异不大。说明 P 值太小了,没有修正的回复力。

当 $P=100$ 时,四轴有一点点回复力,受干扰后左右摆晃,电机声音差异不大。说明 P 值还是太小。

当 $P=1000$ 时,四轴有明显的回复力,受干扰后不会摆晃,电机声音差异很大。说明 P 值接近理想值了。

当 $P=5000$ 时,四轴有明显的回复力,受干扰后两个电机停止转动,电机声音差异非常大。说明 P 值已经太大了。

跟以上调试现象和经验我们最终选定 P 值为 1200,然后再逐渐添加一点了积分 I ,没有添加 D 。内环的反馈是陀螺仪,航向角也是由陀螺仪积分而得,陀螺仪在四轴飞行过程有高频干扰,高频干扰使得 D 具有相反作用,所以我们没有添加 D 。

第八步,调外环 yaw 的 PID

调节外环 yaw 的 P 和调节外环 pitch\roll 的 P 方法一样,主要是调节四轴的受控制的响应速度。给外环 yaw 的 P 设定一个值然后推油门和航向角,感受四轴旋转的快慢。这里我们选定 P 的值为 100。

第九步，手动试飞

经过以上步骤调试，四轴基本可以手动飞行了，松开绳子测试手动飞行。飞行过程中如果在无风条件下发现四轴晃动说明内环 pitch 和 roll 的 P 值大了，这时可以通过减小 P 或增加 D 来抑制晃动。如果发现航向角受外力干扰不能快速回复到原本状态，这可能是内环 yaw 的 P 太小了，需要增加 P。如果发现飞行过程会慢慢自旋，说明 yaw 的 I 太大了，需要减小 I 值。经过以上反复调试，我们最终确定我们 MiniFly 四轴 PID 参数如下图 7.4.4 所示：



图 7.4.4 微型四轴 PID 参数

第十步，调定高的 PID

定高 PID 调节是比较揪心的，调试过程我们按照正常的 PID 调试方法先调 P 再调 I 最后调节 D，反复调试发现根本行不通，四轴依旧很难定在一定高度范围内。后来经过分析，我们发现使用气压计定高，气压数据有很大的滞后性，并且四轴飞行过程垂直向下的风也会干扰气压读数。所以我们调试时不管怎么调节 P，出现的现象是要么慢慢的上升或慢慢的下降，要么快速上蹿下跳，根本不受控制。后来我们根据 PID 理论中 D 有抑制超调和干扰的作用，索性把 D 也先加上一点点，果然有效果，四轴上蹿下跳没那么厉害了。继续增加 D 四轴更加稳定了，当 D 增加到很大时，我们发现，推动油门四轴不怎么受控制了，说明 D 值太大了将 P 的作用都抑制了。反复调试后，我们最终选定 PID 的值为 210,0,600。

总结

PID 三项的意义：P 是系统平衡的回复力；I 是消除误差，有辅助 P 的作用；D 是阻尼，抑制超调和干扰的作用。调试时一般先调节 P 找到临界震荡的 P 值，然后减小一点 P 值，增加 I 值消除静态误差，最后增加 D 值抑制干扰。要不要加 I 和 D 需根据实际情况而定。调试定高 PID 时，需要 P 和 D 同时调。PID 调试是比较繁琐的事情，需要耐心观察现象并分析原理。

9. 微型四轴开发常见问题

9.1 STM32 虚拟串口问题

通常驱动安装失败，打开设备管理器，可以看到如下图 1.1 所示：



图 1.1 虚拟串口驱动安装失败示意图

正确安装 STM32 虚拟串口驱动步骤如下：

安装资料包里的 VCP_V1.4.0_Setup.exe，这个驱动在资料包”3，配套软件 3，STM32 USB 虚拟串口驱动”文件夹内，安装完成，在如下路径可以看到如图 1.2 所示：



图 1.2 VCP_V1.4.0_Setup.exe 安装完成示意图

退出 360 等安全软件，并安装虚拟串口补丁，补丁路径如下图 1.3 所示：然后双击运行“双击这个.bat”，因为部分系统缺少配置文件，所以需要使用的.bat 复制配置文件到系统文件夹里。

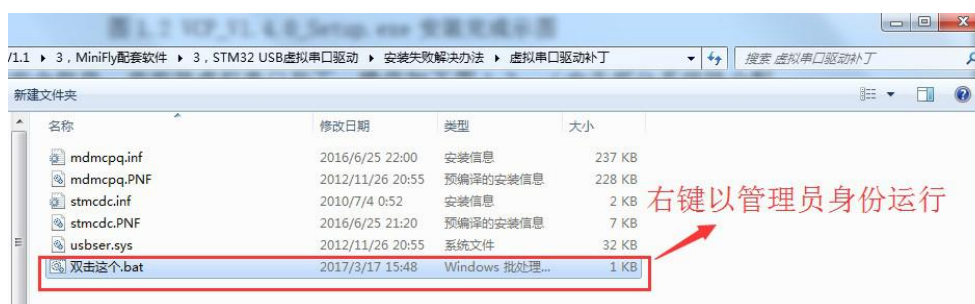


图 1.3 安装虚拟串口补丁示意图

在下图的路径下，运行 dpinst_amd64.exe 或 dpinst_x86.exe（根据自己系统位数选择）如图 1.4 所示。安装成功后如下图 1.5 所示。

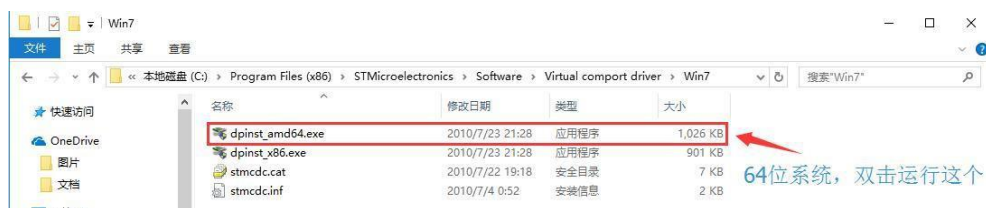


图 1.4 手动执行 dpinst_amd64.exe

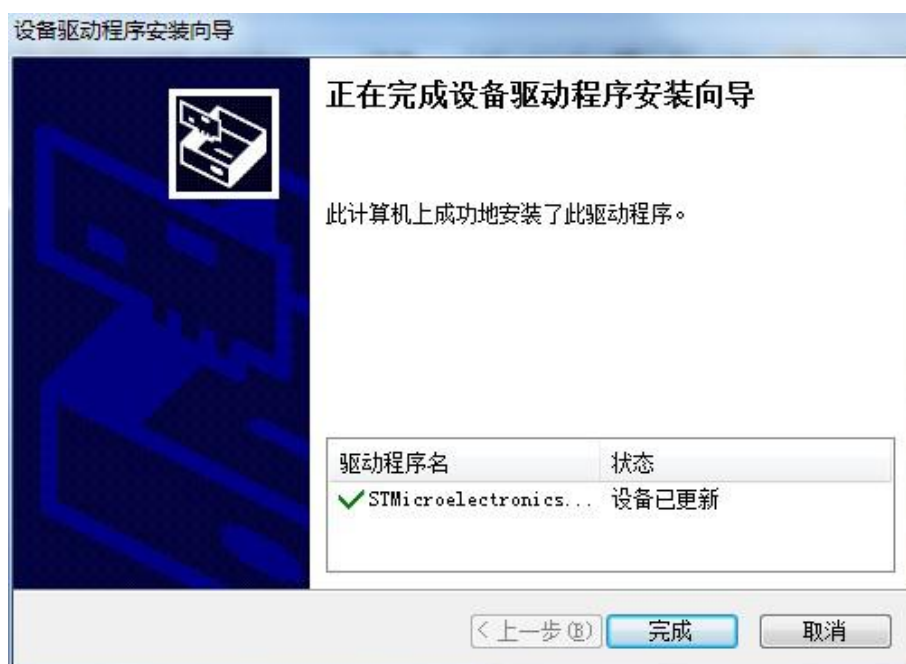


图 1.5 虚拟串口驱动安装成功示图

USB 连接四轴或遥控器，在设备管理 端口可以看到如下图 1.6 所示。

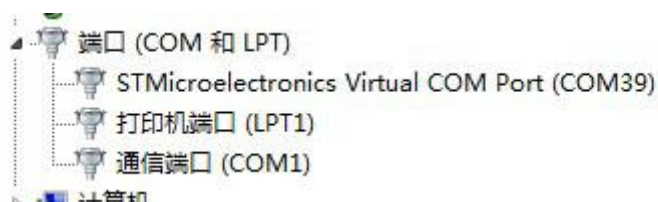


图 1.6 成功安装虚拟串口

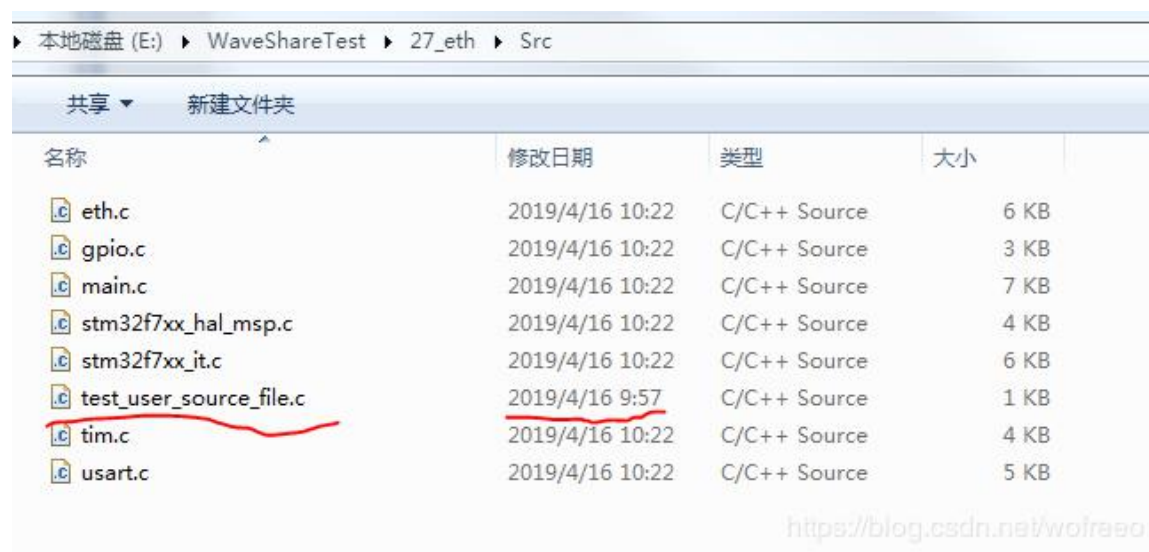
驱动安装成功后，插上 USB，虚拟串口用不了，则检查下 USB 线是否接好，甚至可以换一根 USB 线试试。

9.2 CubeMX 代码生成问题

注意：CubeMX 在重新生成代码后，会覆盖 `/* USER CODE BEGIN */` 和 `/* USER CODE END */` 以外的区域，区域外的代码在重新生成后会被删除！

该工程文件由 STM32CubeMX V5.5.0 版本生成，打开 ioc 工程文件需要升级至 5.5.0 及以上版本。

用户自己添加的 xxx.c 文件，在 CubeMX 重新 Generate 文件时，不会被删掉。CubeMX 只是重新生成了例如 eth.c，gpio.c 之类的文件，不会把用户自己添加的文件删掉。



名称	修改日期	类型	大小
eth.c	2019/4/16 10:22	C/C++ Source	6 KB
gpio.c	2019/4/16 10:22	C/C++ Source	3 KB
main.c	2019/4/16 10:22	C/C++ Source	7 KB
stm32f7xx_hal_msp.c	2019/4/16 10:22	C/C++ Source	4 KB
stm32f7xx_it.c	2019/4/16 10:22	C/C++ Source	6 KB
test_user_source_file.c	2019/4/16 9:57	C/C++ Source	1 KB
tim.c	2019/4/16 10:22	C/C++ Source	4 KB
usart.c	2019/4/16 10:22	C/C++ Source	5 KB

<https://blog.csdn.net/wofreeo>

然而在 keil 工程里却被删掉，因此需要重新添加。为了避免重复添加用户自己的源代码和头文件，可以进行如下操作：

1. 打开 mdk 工程
2. 打开 cubemx 工程，也就是 ioc 后缀文件
3. 进行你的配置，重新生成 mdk 工程（生成的之前 mdk 工程要打开）
4. 点击最小化的 mdk，然后可以看到提示框

更新 xxx.c 文件，选择是

更新 xxx.h 文件，选择是

最重要的一步，提示更新工程文件，选择否

好了，现在芯片配置的代码已经同步过来了，工程文件还是用原来的，添加的文件就还在，重新编译，ok。

9.3 四轴飞行过程往一边偏

如果偏飞得厉害，先检查遥控器微调值是否为 0，如果不为零，就先调整为 0 再试飞看飞行效果。如果通过微调的方式不能纠正偏飞问题，那就是其他原因所致。

电机坏了，电机坏了也分好几种情况，比如像图 2.1 那种情况，电机蓝色底座脱离或者脱落，这种情况导致电机输出动力减弱，甚至有烧坏 MOS 管的可能；还有像图 2.2 那种情况，电机输出轴弯曲，输出轴弯曲，装上桨叶转动起来就会因为不平衡产生很大的震动，震动的直接结果就是导致传感器读数精度变差甚至错误；还有可能是电机长时间工作，轴承或者电刷磨损厉害，导致动力减弱，因为有刷电机是有寿命的，按正常使用来说，这个电机还是有几个月寿命的。对于这种电机坏了的情况，我们只需要换上新的电机即可。

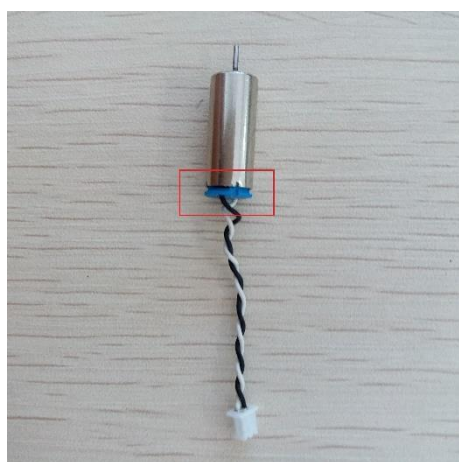


图 2.1 电机蓝色底座脱离

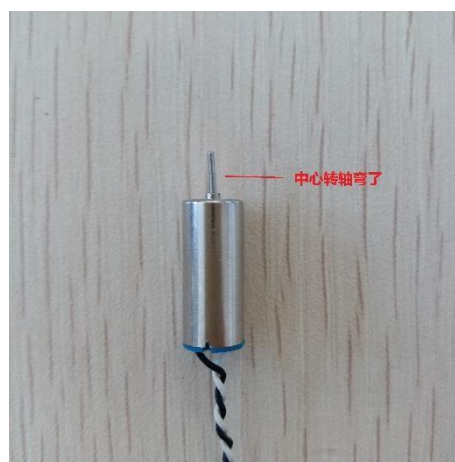


图 2.2 电机中心转轴弯曲

注意：

对于如图 2.1 电机底座脱离的情况，电机就不能再使用了，只能更换新电机，因为电机在这种状态下，不但没有动力输出，甚至会导致 MOS 管烧掉或者短路，MOS 短路后，上电电机不受控制直接就转起来了，如果 MOS 管烧掉那就得重新焊接了。

偏飞也有可能是桨叶不平衡所致，因为空心杯转速高，装上不平衡桨叶高速转动，就会产生很大震动，过大的震动导致传感器度数精度降低甚至错误。怎么看桨叶是否平衡呢？最直接的就是看桨叶是否磨损，变形以及残缺等。如果这种情况，直接换上新的桨叶即可，注意桨叶是有正反的。

偏飞也有可能是信号干扰导致，信号干扰导致四轴接受不到控制信号，从而出现偏飞的情况，如何判断干扰呢？在可控制范围内，遥控器控制四轴飞行，无干扰的时候，遥控器蓝色通讯指示灯常亮；有一点点干扰（不会影响飞行）的时候，遥控器红色指示灯偶尔闪烁一下，蓝色指示灯基本还是常亮；当干扰变严重，遥控器红色指示灯常亮，这时候四轴基本接收不到遥控器控制信号了，偏飞，乱飞的可能性就很大了。

9.4 四轴定高飞行，高度定不住

定高是讨论比较多的一个话题，当然主要讨论的就是定高效果的问题，有不少用户反映定高效果差的问题，比如定高飞不起来，响应缓慢，飞起来上窜下窜等，反正就是有很多问题，当然这里面不排除硬件（电机，桨叶）的问题以及软件（PID）问题。

1.1 版本固件使用串级(位置环+速度环)PID 的方式控制定高，增加稳定性的同时提高了响应速度，定高稳定性也不会受到自重影响。如果定高效果还是很差，那么我们检测下是否是硬件的问题。

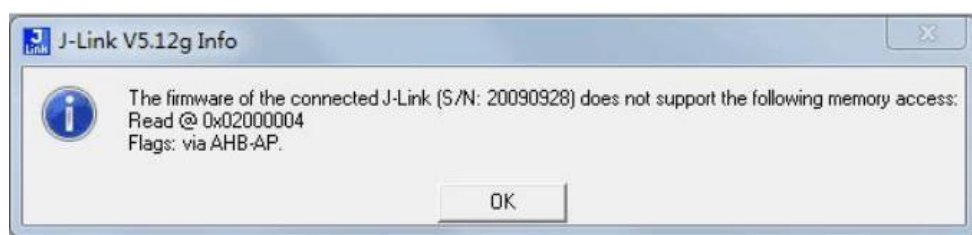
9.5 下载器不能下载调试代码

如果下载器开关设置好了，下载线 2 端也对接好了，还是不能下载，有可能是如下原因：

编译出错，提示代码超过 32K，这种情况是因为 MDK 未破解，这种情况只需要百度一下 MDK 破解。

编译或者下载提示找不到芯片，应该是 MDK 安装之后没有安装相应芯片的 pack 包。请安装芯片型号对应的 Pack 包。

下载或者调试出现如图 5.1 所示的提示：



图

5.1 驱动问题提示

这种情况需要替换一下 jlink 驱动，用我们提供的驱动替换 MDK 安装路径下对应的驱动，替换之前可以先将原来的驱动备份一下。在微型四轴资料包里找到“Segger.zip”，解压出来，然后替换 MDK 安装目录下的 Segger 文件夹，如图 5.2 和 5.3 所示：



图 5.2 新的驱动包

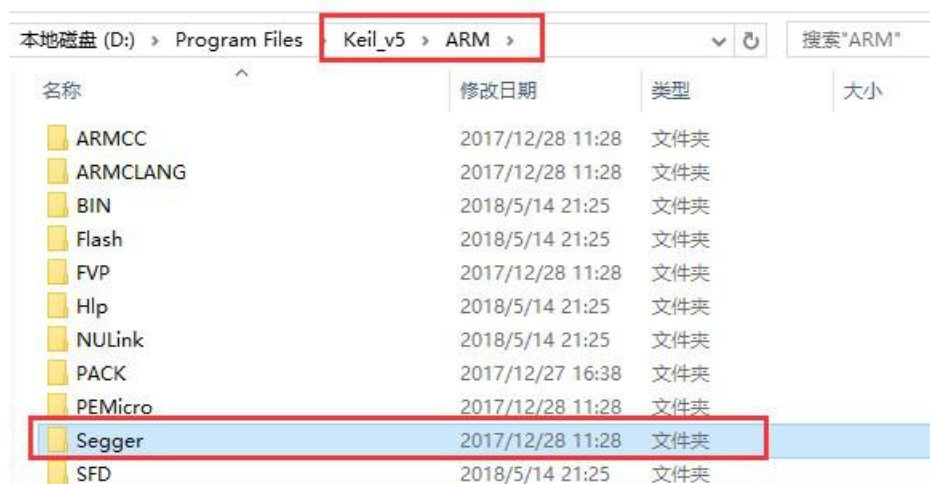


图 5.3 替换掉原来的文件夹

注意：

调试微型四轴时，请务必接上电池四轴供电，不能用下载器的电源给四轴供电并调试，下载器带载能力较弱，只使用它给微型四轴供电调试可能导致下载器烧掉。