User Manual

ATK-PMW3901 光流模块用户手册

超轻低功耗光流模块

用户手册

https://telesky.tmall.com/

目 录

1. 特性参数

ATK-PMW3901 是 ALIENTEK 推出的一款超轻多功能低功耗光流模块（以下简称光流模块），此模块集成一个高精度低功耗光学追踪传感器 PMW3901 和一个高精度激光传感器 VL53LXX（2m 版本使用 VL53L0X, 4m 版本使用 VL53L1X，以下统称 VL53LXX），PMW3901 光流传感器负责测量水平移动，VL53LXX 激光传感器负责测量距离。4m 版本光流模块是 2m 版本光流的升级版本，测量距离更远，抗干扰能力更强。MiniFly 搭配概模块即可实现稳定悬停（遥控器设置为定点模式）。

ATK-PMW3901 光流模块各项参数如表 1.1 所示：

ATK-PMW3901	
接口特性	3.3V~4.2VDC @20mA
通信接口	PMW3901 传感器: SPI2 @2Mhz
	VL53LXX 传感器: 模拟 IIC@400Khz(max)
测量范围	PMW3901 传感器: > 8cm (建议 400cm 以内使用)
	VL53L0X 传感器: 3cm~200cm（30Hz）
	VL53L1X 传感器: 4cm~400cm（30Hz）
输出速率	PMW3901 传感器: > 100Hz(>60Lux)
	VL53L0X 传感器: 30Hz (长距离模式，室内)
	VL53L1X 传感器: 30Hz (长距离模式，室内)
尺寸	长 x 宽：27.5mm x 16.5mm
重量	重量 1.6g

表 1. 1 ATK-PMW3901 光流模块参数

此模块搭配 MiniFly 使用时请注意以下事项：

1) 请先将 MiniFly 以及遥控器固件升级到固件 V1.3 及以上版本，固件版本升级请参考“ATK-MiniFly 微型四轴固件说明_V1.3.pdf”，升级完成之后，遥控器设置控制模式为定点模式；
2) 四轴固件 1.3 及以上版本支持用户打开/关闭激光传感器功能，用户可在遥控器端开启/关闭 VL53LXX（遥控器固件先升级到 V1.3 或者以上版本）。
3) 搭配该模块定点时，会用到激光传感器测距，测距模式为长距离模式，可以测量 2m/4m，但是长距离测量模式要求在黑暗且无红外光的环境，在室外强光下，激光传感器会受到很大干扰，测量精度急剧降低，所以室外飞行建议使用气压定高。
4) 激光测量的距离为测量参考平面到激光传感器的距离，MiniFly 在定点或者定高飞行且激光可测范围内，MiniFly 会根据激光测量高度实时调整自身高度，比如改变激光测量的参考平面，拿一个障碍物放到模块下面，那么 MiniFly 会自动升高到以新的参考平面为基准的设定高度，但是不建议这样做，因为这样同时会影响到光流的数据测量。
5) 光流传感器需要一定的光照条件（>60Lux）,光线过低会影响定点效果；
6) 1.2 及以上版本的固件，扩展模块支持热插拔，也就是四轴开机之后再插上扩展模块就可以直接使用，不用重启四轴。

2. 使用说明

2.1 模块硬件说明

ATK-PMW3901 光流模块实物图如图 2.1.1（2m 版本）和图 2.1.2（4m 版本）所示：

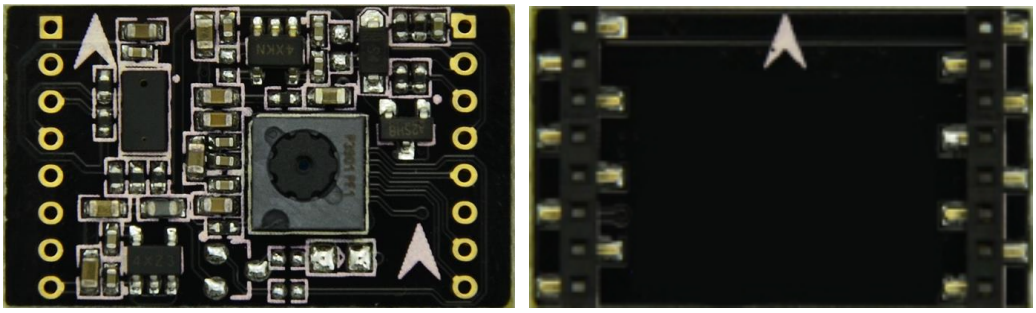


图 2.1.1 2m 版本光流模块实物图

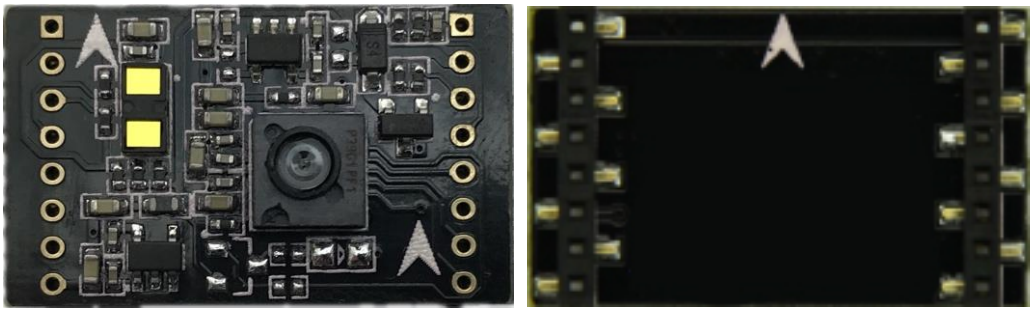


图 2.1.2 4m 版本光流模块实物图

现在改进了光流镜头，用的是图 2.1.2 的镜头，新的镜头无需盖片，而之前的光流模块，我们有在光流模组上增加一个盖片，防止光学镜片进入灰尘，影响传感器精度，如图 2.1.3 所示：

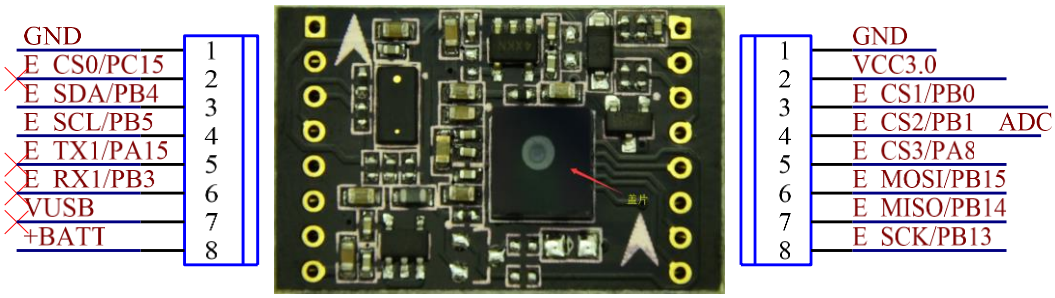


图 2.1.3 光流加盖片效果

注意：

现在使用的光流模块，使用新的镜头，不再使用盖片，如果用户使用的是早期的光流镜头 ，盖片上还有一层保护膜，使用之间请撕掉这个保护膜，且不能刮花保护盖片。

光流模块电路图如下图 2.1.4 所示。

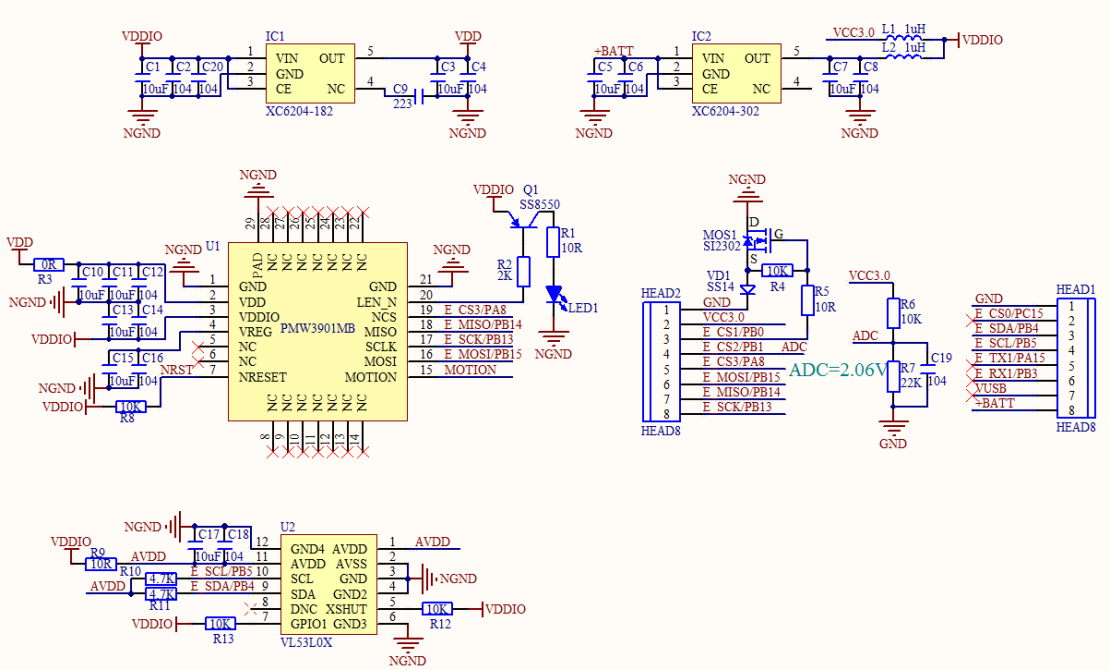


图 2.1.4 ATK-PMW3901 光流模块电路图

可以看到，光流模块原理图还是比较简单的，一颗光流模组 PMW3901,一颗激光测距传感器 VL53LXX，2 颗 LDO，一组 2.0 间距排母，一个 ADC 采集点，以及电阻、电容、二三极管和 MOS 管。

PMW3901 是 PixArt 公司最新的高精度低功耗光学追踪模组，可直接获取 xy 方向运动信息，测量范围 8cm 以上，工作电流< 9mA，工作电压 VDD(1.8~2.1VDC) 和 VDDIO(1.8~3.6VDC)，使用 4 线 SPI 接口通信。原理图可以看出，2 颗 LDO，一颗 1.8V 的 LDO 提供给 VDD，一颗 3.0V 的 LDO 提供给 VDDIO。SPI 接口使用硬件 SPI2，片选信号 PA8，另外 LED1 为红外 LED，默认没有焊接。

VL53L0X 是 ST 公司的第二代激光测距传感器，该芯片集成了激光发射器和 SPAD 红外接收器，采用 FlightSense™技术，通过接收器接收到的光子时间来计算距离，实现更快、更远、更精确的测距功能，该传感器工作电压 AVDD 由 3.0V 的 LDO 经过一个 10R 电阻得到，大概是 2.8V，也恰好是 VL53LXX 工作最佳电压，该传感器测量距离 3cm~200cm，使用 IIC 通信接口，最大通信速率 400K，我们使用模拟 IIC（SDA/PB4,SCL/PB5）和飞控端通信。

而 VL53L1X 是 ST 公司的第三代激光测距传感器，除了拥有第二代传感器的技术外，还集成了物理红外滤波器和光学元件，无论目标颜色和反射率如何，都可以进行距离测量，抗干扰能力更强，VL53L1X 和 VL53L0X Pin-to-pin 兼容，VL53L1X 是 VL53L0X 的升级版，测量距离更远（4~400cm）,测量速度更快（短距离 20ms，长距离 33ms）,抗干扰更强。

此模块同 ATK-LED-RING 灯环模块，ATK-WIFI-MODULE 摄像头模块一样，有一个 ADC 采集脚，根据不同的 ADC 电压值来识别不同的模块 ID，然后飞控根据模块 ID 来控制模块，光流模块的 ADC 值为 2.06V，ADC 的值由电阻 R6 和 R7 分压得到。模块的电源通过 MOS1 控制通断，控制信号为 E_SCL/PB0，高电平时，MOS 管导通。

另外可以看到电感 L1 和 L2，通过 L1 和 L2 来选择 VDDIO，如果使用 L1，那么 VDDIO 电源来自于飞控 VCC3.0，如果使用 L2，那么 VDDIO 电源来自光流模块 3.0V LDO IC2，默认焊接 L2，这样可以得到更稳定的电源。

2.2 模块和 MiniFly 连接示意图

模块自带 2x8 排母，用来对接 MiniFly 的 2x8 排针，但是要注意模块是有方向的，我们需要按照模块箭头和四轴机头箭头保持一致的方式安装模块，安装效果如图 2.2.1 所示：

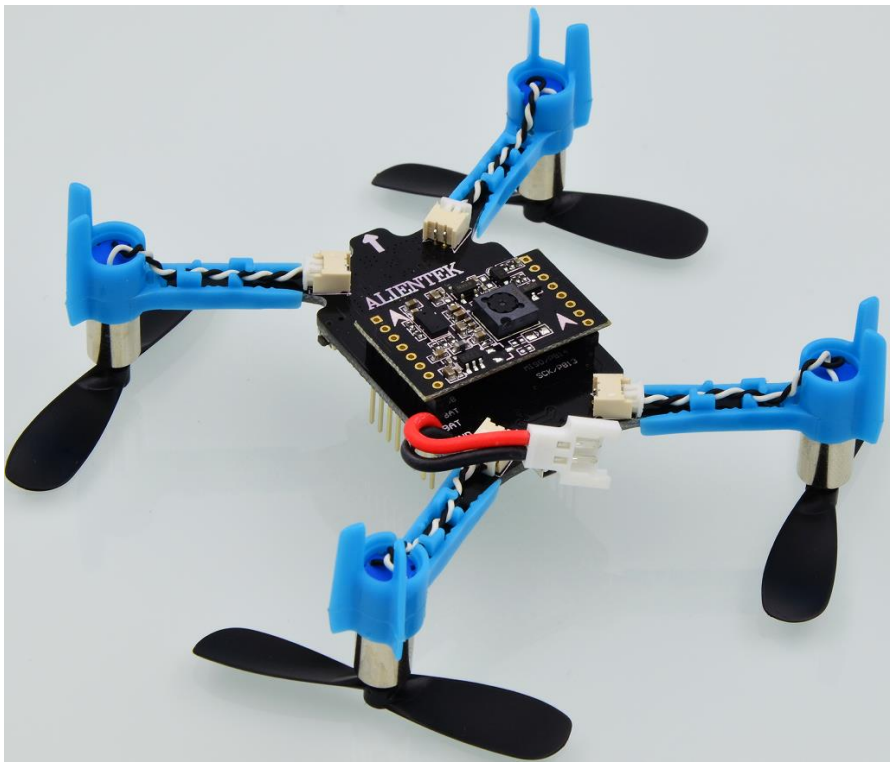


图 2.2.1 模块装机图

2.3 PMW3901 光流传感器和 MiniFly 通信

PMW3901 光流传感器和 MiniFly 之间采用 SPI 的通信方式，使用硬件 SPI2，光流传感器和飞控通信源码主要分 2 个部分，底层 SPI 驱动和光流数据通信。

底层 SPI 驱动配置比较简单，2 线全双工，2M 波特率，主机模式，这部分就不贴代码了，下去看 spi.c 即可。

然后 SPI 的收发都使用 DMA 的方式，接收 DMA 使用 DMA1 数据流 3，发送 DMA 使用 DMA1 数据流 4，详细内容请看源码 spi.c。需要注意的是，SPI 的 DMA 发送中断和灯环的中断共用 DMA1 数据流 4 中断，所以我们需要在 DMA1 数据流 4 中断里面进行分开处理，处理方式如下：

```
static enum expModuleID lastModuleID = NO_MODULE;
void    DMA1_Stream4_IRQHandler(void)
{
    if(getModuleID() == LED_RING)
    {
        lastModuleID = LED_RING;
        ws2812DmaIsr();
    }
    else if(getModuleID() == OPTICAL_FLOW)
```

```
        {
            lastModuleID = OPTICAL_FLOW;
            spiTxDmaIsr();
        }
        else if(getModuleID() == NO_MODULE)
        {
            if(lastModuleID == LED_RING)
            {
                ws2812DmaIsr();
                DMA_ITConfig(DMA1_Stream4, DMA_IT_TC, DISABLE);
                DMA_Cmd(DMA1_Stream4,DISABLE);
            }
            else if(lastModuleID == OPTICAL_FLOW)
            {
                spiTxDmaIsr();
                DMA_ITConfig(DMA1_Stream3, DMA_IT_TC, DISABLE);
                DMA_ITConfig(DMA1_Stream4, DMA_IT_TC, DISABLE);
                DMA_Cmd(DMA1_Stream3,DISABLE);
                DMA_Cmd(DMA1_Stream4,DISABLE);
            }
        }
    }
```

可以看到，中断里面，我们根据模块的 ID 来分开处理，getModuleID()就是读取当前模块的 ID，现在 MiniFly 支持 ATK-LED-RING 灯环模块，ATK-WIFI-MODULE 摄像头模块以及 ATK-PMW3901 光流模块，可以看到中断里多了对未检测到模块（NO_MODULE）做处理的内容，这部分内容的作用就是，保证我们在拔掉模块后，可以继续进入中断，完成拔掉模块之前没有执行完的中断内容。

然后再来说光流数据通信，通信相关内容在 optical_flow.c 里面，主要包括光流电源控制函数 opticalFlowPowerControl，读写寄存器函数 registerRead，registerWrite，突发读取 12 字节运动数据函数 readMotion，寄存器初始化函数 InitRegisters，光流任务函数 opticalFlowTask，读取光流数据函数 getFlowData，以及光流初始化函数 opticalFlowInit，我们主要讲解下几个比较重要的函数。

光流初始化函数，源码如下：

```
    void opticalFlowInit(void)
    {
        if (!isInit) /*第一次初始化通用 IO*/
        {
            GPIO_InitTypeDef GPIO_InitStructure;

            //初始化 CS 引脚
            RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);//使能时钟
            RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);//使能时钟

            GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
```

```
            GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
            GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
            GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
            GPIO_Init(GPIOA, &GPIO_InitStructure);

            GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
            GPIO_Init(GPIOB, &GPIO_InitStructure);
        }
        else
        {
            resetOpFlowData();
            opFlow.isOpFlowOk = true;
        }

        opticalFlowPowerControl(true);    /*打开电源*/
        vTaskDelay(50);

        NCS_PIN = 1;
        spi2Init();
        vTaskDelay(40);

        uint8_t chipId = registerRead(0);
        uint8_t invChipId = registerRead(0x5f);
    //  printf("Motion chip is: 0x%x\n", chipId);
    //  printf("si pihc noitoM: 0x%x\n", invChipId);

        // 上电复位
        registerWrite(0x3a, 0x5a);
        vTaskDelay(5);

        InitRegisters();
        vTaskDelay(5);

        if (isInit)
        {
            vTaskResume(opFlowTaskHandle);     /*恢复光流任务*/
        }
        else if(opFlowTaskHandle == NULL)
        {
            xTaskCreate(opticalFlowTask,     "OPTICAL_FLOW",     300,     NULL,     4,
&opFlowTaskHandle);  /*创建光流模块任务*/
        }

        vl53lxxInit();  /*初始化 vl53lxx*/
```

```
        isInit = true;
    }
```

isInit 是个静态 bool 变量，用于判断是否是第一次使用光流模块，如果是第一次使用（isInit=false）则初始化相关 GPIO 引脚，然后打开光流电源，初始化 SPI2 接口，初始化光流寄存器，然后再次判断是否第一次使用光流模块，如果第一次使用，则创建光流通信任务，如果之前已经创建了光流通信任务，则直接将挂起的任务恢复即可，最后初始化激光传感器 VL53LXX(vl53l0x 或者 vl53l1x)，关于激光传感器的控制，我在 2.4 章节给大家说明。这样光流初始化就完成了。

接着我们看下光流通信任务，源码如下：

```
    void opticalFlowTask(void *param)
    {
        static u16 count = 0;
        u32 lastWakeTime = getSysTickCnt();

        opFlow.isOpFlowOk = true;

        while(1)
        {
            vTaskDelayUntil(&lastWakeTime, 10);       /*100Hz 10ms 周期延时*/

            readMotion(&currentMotion);

            if(currentMotion.minRawData == 0 && currentMotion.maxRawData == 0)
            {
                if(count++ > 100 && opFlow.isOpFlowOk == true)
                {
                    count = 0;
                    opFlow.isOpFlowOk = false;       /*光流出错*/
                    vTaskSuspend(opFlowTaskHandle);    /*挂起光流任务*/
                }
            }else
            {
                count = 0;
            }
            /*连续 2 帧之间的像素变化，根据实际安装方向调整  (pitch:x)   (roll:y)*/
            int16_t pixelDx = currentMotion.deltaY;
            int16_t pixelDy = -currentMotion.deltaX;

            if (ABS(pixelDx) < OULIER_LIMIT && ABS(pixelDy) < OULIER_LIMIT)
            {
                opFlow.pixSum[X] += pixelDx;
                opFlow.pixSum[Y] += pixelDy;
            }else
```

```
            {
                outlierCount++;
            }
        }
    }
```

先说一下 opFlow 光流结构体，定义如下：

```
typedef struct opFlow_s
{
    float pixSum[2];        /*累积像素*/
    float pixComp[2];       /*像素补偿*/
    float pixValid[2];      /*有效像素*/
    float pixValidLast[2];  /*上一次有效像素*/

    float deltaPos[2];      /*2 帧之间的位移  单位 cm*/
    float deltaVel[2];      /*速度  单位 cm/s*/
    float posSum[2];        /*累积位移  单位 cm*/
    float velLpf[2];        /*速度低通  单位 cm/s*/

    bool isOpFlowOk;        /*光流状态*/
    bool isDataValid;       /*数据有效*/

} opFlow_t;
```

看注释大家也应该大概明白各变量的含义了，我再简单说明一下：
累积像素，就是自四轴起飞后的累积像素；
像素补偿，就是补偿由于飞机倾斜导致的像素误差；
有效像素，指经过补偿的实际像素；
2 帧之间的位移，这个就是由像素转换出来的实际位移，单位 cm；
速度，这个速度是瞬时速度，由位移变化量微分得到，单位 cm/s；
累积位移，实际位移，单位 cm；
速度低通，对速度进行低通，增加数据平滑性；
光流状态，光流是否正常工作；
数据有效，在一定高度范围内，数据有效；

接着看光流任务内容，可以看到，我们设定光流传感器更新速率 100Hz，然后就是使用突发模式一次性读取多字节运动数据了，这个运动数据是 1 个由 12 个字节组成的结构体，此结构体定义如下：

```
typedef __packed struct motionBurst_s
{
    __packed union
    {
        uint8_t motion;
        __packed 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;
        int16_t deltaX;
        int16_t deltaY;

        uint8_t squal;

        uint8_t rawDataSum;
        uint8_t maxRawData;
        uint8_t minRawData;

        uint16_t shutter;
    } motionBurst_t;
```

简单说明下结构体内容，motion 指的是运动信息，同时又是一个联合体，这样就可以根据不同的位去判断运动信息，包括帧判别，运行模式和运动信息检测等，这个我们暂时不需要用到；然后就是 observation，这个是用于检测 IC 是否出现 EFT/B 或者 ESD 问题，传感器正常工作时，读取出来的值为 0xBF；接着就是我们要用到的 int16_t 类型的运动信息 deltaX, deltaY 了，这个就是光流检测到图像的 X 和 Y 方向的运动信息；然后是 squal 是指运动信息质量，简单说就是运动信息的可信度；rawDataSum 这个是原数据求和，可用作对一帧数据求平均值；至于 maxRawData 和 minRawData ，都知道是最大和最小原始数据了；然后 shutter 是一个实时自动调整的值，目的是保证平均运动数据在正常可操作范围以内，这个值可以搭配 squal，用来判断运动信息是否可用。

接着说光流任务里的内容，100Hz 突发模式读取 12 字节数据，然后对读取的数据进行处理，首先是判断最大最小原始数据，如果连续 1s 时间都为 0，说明光流出故障了，因为正常工作的时候最大最小原始值不为 0，如果出故障了，我们就挂起光流任务；光流正常，我们就继续分析运动数据，先读出 x，y 方向的像素变化，读取的时候需要根据光流的安装方向做相应的调整，调整方式参考光流手册 28 页，上面有描述 x，y 的运动正方向，MiniFly 设置 pitch 方向为 x，roll 方向为 y，需要注意的是：这个运动正方向是光流传感器检测到图像的运动，当我们安装到 MiniFly 之后，以地面为参考，地面是不动的 ，MiniFly 在运动，那么检测到的图像运动是相反的，比如 MiniFly 往前走，光流检测的图像是往后的，因为参考点不同。

读取运动数据之后，我们加了限幅判断，目的是增加数据安全性，接着将原始像素数据求和并低通；如果数据不安全，则使用变量 outlierCount 累加。这些就是光流任务内容了。

我们接着看一下光流数据读取，这儿比较重要了，源码如下：

```
bool getOpFlowData(state_t *state, float dt)
{
    static u8 cnt = 0;
```

```
        float height = 0.01f * getFusedHeight();/*读取高度信息 单位 m*/

        if(opFlow.isOpFlowOk && height<4.0f)    /*4m 范围内，光流可用*/
        {
            cnt= 0;
            opFlow.isDataValid = true;

            float coeff = RESOLUTION * height;
            float tanRoll = tanf(state->attitude.roll * DEG2RAD);
            float tanPitch = tanf(state->attitude.pitch * DEG2RAD);

            opFlow.pixComp[X] = 480.f * tanPitch;     /*像素补偿，负方向*/
            opFlow.pixComp[Y] = 480.f * tanRoll;
            opFlow.pixValid[X] = (opFlow.pixSum[X] + opFlow.pixComp[X]); /* 实 际 输 出
像素*/
            opFlow.pixValid[Y] = (opFlow.pixSum[Y] + opFlow.pixComp[Y]);

            if(height < 0.05f)   /*光流测量范围大于 5cm*/
            {
                coeff = 0.0f;
            }
            opFlow.deltaPos[X] = coeff * (opFlow.pixValid[X] - opFlow.pixValidLast[X]);
    /*2 帧之间位移变化量，单位 cm*/
            opFlow.deltaPos[Y] = coeff * (opFlow.pixValid[Y] - opFlow.pixValidLast[Y]);

            opFlow.pixValidLast[X] = opFlow.pixValid[X]; /*上一次实际输出像素*/
            opFlow.pixValidLast[Y] = opFlow.pixValid[Y];
            opFlow.deltaVel[X] = opFlow.deltaPos[X] / dt;   /*速度 cm/s*/
            opFlow.deltaVel[Y] = opFlow.deltaPos[Y] / dt;

            opFlow.velLpf[X] += (opFlow.deltaVel[X] - opFlow.velLpf[X]) * 0.15f;  /* 速 度
低通 cm/s*/
            opFlow.velLpf[Y] += (opFlow.deltaVel[Y] - opFlow.velLpf[Y]) * 0.15f;  /* 速 度
低通 cm/s*/
            opFlow.velLpf[X] = constrainf(opFlow.velLpf[X], -VEL_LIMIT, VEL_LIMIT);
    /*速度限幅 cm/s*/
            opFlow.velLpf[Y] = constrainf(opFlow.velLpf[Y], -VEL_LIMIT, VEL_LIMIT);
    /*速度限幅 cm/s*/

            opFlow.posSum[X] += opFlow.deltaPos[X];/*累积位移 cm*/
            opFlow.posSum[Y] += opFlow.deltaPos[Y];/*累积位移 cm*/
        }
        else if(opFlow.isDataValid == true)
        {
```

```
            if(cnt++ > 100)      /*超过定点高度，切换为定高模式*/
            {
                cnt = 0;
                opFlow.isDataValid = false;
            }
            resetOpFlowData();
        }
        return opFlow.isOpFlowOk;  /*返回光流状态*/
    }
```

首先读取高度信息，下面会用的，这个高度信息是由激光 vl53lxx 测量高度和气压高度融合得到（激光测高，我在 2.4 章节说明）。接着判断光流工作状态以及高度信息范围，说一下这个融合的高度信息，因为光流传感器也有精度，一定程度上和高度成反比，所以高度达到一定程度，精度就不怎么好了，我们暂时设定可用范围 4m，然后定义变量系数，这个变量系数用做转换像素信息为实际位移信息，该系数由高度和分辨率 RESOLUTION 相乘得到，RESOLUTION 是怎么来的呢，它是一个宏，定义如下：

```
#define RESOLUTION     (0.2131946f)      /*1m 高度下  1 个像素对应的位移，单位 cm*/
```

它的意思就是 1m 高度下，一个像素对应的位移，单位 cm，这个分辨率怎么来的呢，光流手册 42 页有个光流 CPI 和高度的关系图表，根据 CPI 和高度信息，计算得到这个分辨率。因此我们得到高度信息、分辨率以及 2 帧之间的像素变化之后，我们就可以算出 2 帧之间的实际位移了。

接着是倾角补偿计算，倾角补偿就的目的是去除由于机体倾斜导致的像素误差，根据 roll，pitch，计算像素补偿，可以看到有个系数 480，这个 480 如何得到呢，当然是实际测试得到的，测试过程是：保证在某一高度下，调整这个值，使得我们原地前后左右晃动四轴，光流输出数据基本保持不变，那么这个值就是我们需要的，因为四轴在原地，水平方向不应当有位移变化。

有了倾角补偿和运动累积像素，我们就可以得到实际累积像素，减去上次的实际像素，就可以得到 2 帧之间的变化像素，再乘以系数就可以得到 2 帧之间的位移变化，可以看到还有对系数的限制，当高度小于 5cm，光流就无法工作了，所以系数设置为 0。接着对这个位移积分得到四轴到起飞点的位移，对这个位移微分得到瞬时速度，对速度进行低通增加数据的平滑性，对速度进行限幅处理，增加数据安全性。

我们通过光流就得到了四轴的位置信息和速度信息，把这些位置信息和速度信息融合加速计（state_estimator.c），得到估测位置和速度，将估测位置和速度参与 PID 运算，即可用于水平方向位置控制，这部分内容请看 position_pid.c，源码里面可以直接看到位置环和速度环 PID 的处理过程，这样就可以实现水平定点控制了。

后面是对异常数据做处理，比如高度过高（>400cm）一段时间之后，设定光流数据不可用，也就是设定 opFlow.isDataValid 为 false，然后清零光流数据。另外在 commander.c 里面，实时读取 opFlow.isDataValid 的状态，根据 opFlow.isDataValid 的状态设置是否使用定点模式。

2.4 VL53LXX 激光传感器和 MiniFly 通信

VL53LXX（包括 VL53L0X 和 VL53L1X），我们把底层驱动函数分别写到 vl53l0x.c 和 vl53l1x.c，任务函数写到 vl53lxx.c。

激光传感器和 MiniFly 之间采用模拟 IIC 的通信方式，激光传感器和飞控通信源码主要分 2 个部分，底层 IIC 驱动部分和激光测距应用部分。

底层 IIC 驱动配置比较简单，使用 2 个通用 IO（PB4，PB5）来模拟 IIC，代码比较简单，就不贴出来了，下去看 vl53l0x_i2c.c 即可。

VL53L0X 底层驱动代码，这部分内容在 vl53l0x.c 里面，这部分内容基本移植于 crazyflie2.0，这一个源文件已经集成各种使用该传感器需要用到的函数，ST 官方虽然没有提供该传感器的寄存器手册，但是提供了传感器的软件 API（应用编程接口）以及完整的文档和例程源码，例程源码是 ST 官方的 X-NUCLEO-53L0A1 扩展板基于 STM32F401RE 和 STM32L476RG Nucleo 开发板进行开发的，也是需要移植一下才可以使用，如果大家感兴趣，可以参考这个文档：VL53L0XAPIv1.0.2.4823externalx.chm，我们也有单独的 VL53L0X 激光测距模块 ATK-VL53L0X，就是通过调用官方的各种 API 函数来使用该传感器，资料也是开源的，可自行下载参考。这儿我们就移植别人写好的驱动，一个.c 文件搞定，使用起来相对简单，VL53L0X 的寄存器配置特别多，就不贴源码了，可以自己下去看 vl53l0x.c。

VL53L1X 底层驱动代码，这部分内容移植于 ST 官方库，像 VL53L0X 一样，ST 官方提供了 VL53L1X 的，软件 API（应用编程接口）以及完整的文档和例程源码，例程源码是 ST 官方的 X-NUCLEO-53L1A1 扩展板基于 STM32F401RE 和 STM32L476RG Nucleo 开发板进行开发的，这部分 API 函数在源码文件夹 VL53L1X 里面，然后平台接口函数 vl53l1x，用于连接上层 API 和 MCU 的 IIC，对 API 函数感兴趣的用户，可以参考这 2 个文档：VL53L1X_API.chm 以及 VL53L1X API User Manual (UM2356).pdf。

2.4.1. VL53LXX 初始化

VL53LXX 初始化如下：

```
void vl53lxxInit(void)
{
    vl53IICInit();
    delay_ms(10);

    /*vl53l0x 初始化*/
    vl53lxxId = vl53l0xGetModelID();
    if(vl53lxxId == VL53L0X_ID)
    {
        if (isInitvl53l0x)
        {
            reInitvl53l0x = true;
            vTaskResume(vl53l0xTaskHandle);     /*恢复激光测距任务*/
        }
        else /*首次接上 vl53l0x 光流模块*/
        {
            isInitvl53l0x = true;
            xTaskCreate(vl53l0xTask, "VL5310X", 300, NULL, 5, &vl53l0xTaskHandle);
/*创建激光测距模块任务*/
        }
        return;
    }
    delay_ms(10);
```

```
        /*vl53l1x  初始化*/
        VL53L1_RdWord(&dev, 0x010F, &vl53lxxId);
        if(vl53lxxId == VL53L1X_ID)
        {
            if (isInitvl53l1x)
            {
                reInitvl53l1x = true;
                vTaskResume(vl53l1xTaskHandle);     /*恢复激光测距任务*/
            }
            else  /*首次接上 vl53l1x 光流模块*/
            {
                isInitvl53l1x = true;
               xTaskCreate(vl53l1xTask, "VL53L1X", 300, NULL, 5, &vl53l1xTaskHandle);
    /*创建激光测距模块任务*/
            }
            return;
        }

        vl53lxxId = 0;
    }
```

首先初始化模拟 IIC，然后先尝试读取 vl53l0x 的 ID，因为 vl53l0x 和 vl53l1x 的 ID 不同，我们就可以根据 ID 来初始化对应的传感器。然后根据 reInitvl53l1x / reInitvl53l1x 的值判断是否第一次使用 VL53L0X / VL53L1X，如果第一次使用，则创建激光测距任务 vl53l0xTask / vl53l1xTask，否则，恢复之前之间已经创建且挂起的测距任务。

2.4.2. VL53L0X 任务

vl53l0x 任务函数如下：

```
    void vl53l0xTask(void* arg)
    {
        TickType_t xLastWakeTime = xTaskGetTickCount();

        vl53l0xSetParam(); /*设置 vl53l0x  参数*/

        while (1)
        {
            if(reInitvl53l0x == true)
            {
                count = 0;
                reInitvl53l0x = false;
                vl53l0xSetParam(); /*设置 vl53l0x  参数*/
                xLastWakeTime = xTaskGetTickCount();

            }else
            {
```

```
                range_last = vl53l0xReadRangeContinuousMillimeters() * 0.1f; //单位 cm

                if(range_last < VL53L0X_MAX_RANGE)
                    validCnt++;
                else
                    inValidCnt++;

                if(inValidCnt + validCnt == 10)
                {
                    quality += (validCnt/10.f - quality) * 0.1f;    /*低通*/
                    validCnt = 0;
                    inValidCnt = 0;
                }

                if(range_last >= 6550)   /*vl53  错误*/
                {
                    if(++count > 30)
                    {
                        count = 0;
                        vTaskSuspend(vl53l0xTaskHandle);    /*挂起激光测距任务*/

                    }
                }else count = 0;

                vTaskDelayUntil(&xLastWakeTime, measurement_timing_budget_ms);
            }
        }
    }
```

先是定义一些变量，其中变量 validCnt 和 inValidCnt 分别用于统计激光数据是否可用，具体如何统计，我待会儿在下面会给大家说明。然后就开始设置测量参数 vl53l0xSetParam，设置详细内容如下：

```
void vl53l0xSetParam(void)  /*设置 vl53l0x  参数*/
{
    vl53l0xTest();
    vl53l0xSetVcselPulsePeriod(VcselPeriodPreRange, 18);    /*长距离模式 33ms  周期*/
    vl53l0xSetVcselPulsePeriod(VcselPeriodFinalRange, 14);  /*长距离模式 33ms  周期*/
    vl53l0xStartContinuous(0);
}
```

这里主要就是设置测量模式，包括设定 VCSEL 脉冲周期（VcselPeriodPreRange）和设定设定 VCSEL 脉冲周期范围（VcselPeriodFinalRange）；该传感器支持 4 种模式，当这 2 个参数分别设置为 18 和 14，最大测量周期 33ms，就表示长距离测量模式，测量范围可达 2m。需要注意的是，此模式限制在黑暗且无红外光条件下使用，在户外强光下测量精度会急剧下降，此模块使用就是这种模式，所以激光测距最好不要在户外使用；

当这 2 个参数分别设置为 14 和 10 则有可能表示默认模式、高精度模式，或者高速模式，

这几种模式测量范围都只有 1.2m，如何区分这几种模式呢，当然是根据设置传感器最大测量周期来区分这几种模式，当最大测量周期设置为 20ms，则表示高速模式，测量误差±5%；最大测量周期设置为 30ms，表示默认模式，测量精度±4%；最大测量周期设置为 200ms，表示高精度模式，此模式测量精度高，测量误差小于±3%，但耗时多。

接着设置采集模式为连续模式，也就是以最快的速率读取数据。

然后进入 while()循环，先判断 reInitvl53l0x 这个变量，如果为真，表示再次使用激光测距，那么我们需要设置一下测量模式，接着才是以 30Hz 读取激光数据，接着对激光数据分析，先判断激光数据是否有效，判断依据测量值是否小于最大值 VL53L0X_MAX_RANGE，这是一个宏，在 vl53l0x.h 中定义，宏的内容为 220，表示最大测量距离 220cm，如果测量数据有效变量 validCnt 加 1，否则无效变量 inValidCnt 加 1，当有效无效变量相加等于 10，我们进行一次数据可信度计算，计算很简单，将数据有效次数除以总次数，然后低通处理，这样我们得到激光数据可信度 quality，范围 0~1.0；可以看到后边还有一次对测量数据的判断，这个主要用于识别 vl53l0x 是否掉线，当 vl53l0x 掉线（比如拔掉光流模块）之后，这个测量数据一直大于 6550，此时，我们将激光测距任务挂起，直到激光传感器恢复正常。

2.4.3. VL53L1X 任务

vl53l0x 任务函数如下：

```
void vl53l1xTask(void* arg)
{
    int status;
    u8 isDataReady = 0;
    TickType_t xLastWakeTime = xTaskGetTickCount();;
    static VL53L1_RangingMeasurementData_t rangingData;

    vl53l1xSetParam(); /*设置 vl53l1x  参数*/

    while(1)
    {
        if(reInitvl53l1x == true)
        {
            count = 0;
            reInitvl53l1x = false;
            vl53l1xSetParam(); /*设置 vl53l1x  参数*/
            xLastWakeTime = xTaskGetTickCount();
        }else
        {
            status = VL53L1_GetMeasurementDataReady(&dev, &isDataReady);

            if(isDataReady)
            {
                status = VL53L1_GetRangingMeasurementData(&dev, &rangingData);
                if(status==0)
                {
```

```
                    range_last = rangingData.RangeMilliMeter * 0.1f;      /*单位 cm*/
                }
                status = VL53L1_ClearInterruptAndStartMeasurement(&dev);
            }

            if(range_last < VL53L1X_MAX_RANGE)
                validCnt++;
            else
                inValidCnt++;

            if(inValidCnt + validCnt == 10)
            {
                quality += (validCnt/10.f - quality) * 0.1f;    /*低通*/
                validCnt = 0;
                inValidCnt = 0;
            }

            if(getModuleID() != OPTICAL_FLOW)
            {
                if(++count > 10)
                {
                    count = 0;
                    VL53L1_StopMeasurement(&dev);
                    vTaskSuspend(vl53l1xTaskHandle);    /*挂起激光测距任务*/

                }
            }else count = 0;

            vTaskDelayUntil(&xLastWakeTime, 50);
        }
    }
}
```

这部分内容和 vl53l0x 大同小异，先设置 vl53l1x 测量参数 vl53l1xSetParam，设置详细内容如下：

```
int vl53l1xSetParam(void)    /*设置 vl53l1x  参数*/
{
    int status;

    status = VL53L1_WaitDeviceBooted(&dev);
    status = VL53L1_DataInit(&dev);
    status = VL53L1_StaticInit(&dev);
    status = VL53L1_SetDistanceMode(&dev, VL53L1_DISTANCEMODE_LONG);
    status = VL53L1_SetMeasurementTimingBudgetMicroSeconds(&dev, 45000);
    status = VL53L1_SetInterMeasurementPeriodMilliSeconds(&dev, 50);
```

```
        status = VL53L1_StopMeasurement(&dev);
        status = VL53L1_StartMeasurement(&dev);

        return status;
    }
```

这些 API 函数都可以在上面说的文档里面找到说明，简单说你个重要的 API 函数：

VL53L1_SetDistanceMode：设置测量模式，包括 3 个模式，短距离模式(<136cm)、中距离模式(<290cm)和长距离模式(<400cm)，默认设置为长距离模式，测量距离可以到 400cm。

VL53L1_SetMeasurementTimingBudgetMicroSeconds：设置测量时间，单位 us，不同模式对测量时间有不同的要求，测量距离越大，需要的时间越长，短距离模式要求测量时间不小于 20ms，所有工作模式正常工作要求测量时间不小于 33ms，测量时间设置越大，数据越精确，我们默认设置为 45ms。

VL53L1_SetInterMeasurementPeriodMilliSeconds：设置连续测量模式间隔，因为除了测量时间，读取也需要时间，所以这个时间必须大于测量时间，我们设置为 50ms。

参数设置完成后进入 while()循环，先判断 reInitvl53l1x 这个变量，如果为真，表示再次使用激光测距，那么我们需要设置一下测量模式，接着才是以 20Hz 读取激光数据，接着对激光数据分析，先判断激光数据是否有效，判断依据测量值是否小于最大值 VL53L1X_MAX_RANGE，这是一个宏，在 vl53l1x.h 中定义，宏的内容为 410，表示最大测量距离 410cm，如果测量数据有效变量 validCnt 加 1，否则无效变量 inValidCnt 加 1，当有效无效变量相加等于 10，我们进行一次数据可信度计算，计算很简单，将数据有效次数除以总次数，然后低通处理，这样我们得到激光数据可信度 quality，范围 0~1.0；可以看到后边还有一次对测量数据的判断，这个主要用于识别 vl53l1x 是否掉线，当 vl53l1x 掉线（比如拔掉光流模块）一段时间之后，此时，我们将激光测距任务挂起，直到激光传感器恢复正常。

2.4.4. VL53LXX 读取数据

上面激光测距任务内得到了激光数据和可信度，那要如何用上这个数据和可信度呢，当然是要调用相应的函数把数据和可信度送到需要的地方，激光数据读取源码如下：

```
bool vl53lxxReadRange(zRange_t* zrange)
{
    if(vl53lxxId == VL53L0X_ID)
    {
        zrange->quality = quality;          //可信度
        vl53lxx.quality = quality;

        if (range_last != 0 && range_last < VL53L0X_MAX_RANGE)
        {
            zrange->distance = (float)range_last;    //单位[cm]
            vl53lxx.distance =  zrange->distance;
            return true;
        }
    }
    else if(vl53lxxId == VL53L1X_ID)
    {
```

```
            zrange->quality = quality;              //可信度
            vl53lxx.quality = quality;

            if (range_last != 0 && range_last < VL53L1X_MAX_RANGE)
            {
                zrange->distance = (float)range_last;    //单位[cm]
                vl53lxx.distance =  zrange->distance;
                return true;
            }
        }

        return false;
    }
```

可以看到，我们直接根据初始化读取到的 ID 读取距离信息和可行度信息，内容比较简单，先说说这个结构体参数 zRange_t，定义如下：

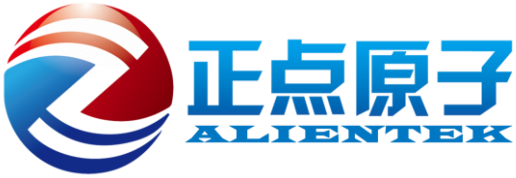
```
    typedef struct zRange_s
    {
        uint32_t timestamp;      //时间戳
        float distance;          //测量距离
        float quality;           //可信度
    } zRange_t;
```

看注释就可以理解了。

先读取可信度，这个可信度是实时更新，而测量数据则需要经过判断，如果测量值不为零且小于 VL53L0X_MAX_RANGE / VL53L1X_MAX_RANGE，则读取测量数据。到此，激光数据测量完成。

激光读取的数据用于高度融合，这部分内容请看 state_estmator.c，源码内有详细的高度融合过程，简单说就是气压和激光数据融合得到高度信息，同时根据 z 轴加速度会估测出一个高度信息和 z 轴速度信息，然后用融合的高度信息去校正估测高度和 z 轴估测速度，最后得到姿态高度和姿态速度（z 轴）。最后，同样也是在源码 position_pid.c 里面进行高度串级 PID 控制，也就是 PID 位置环加 PID 速度环的方式，从而实现定高控制。

最后还有一个函数 setVl53lxxState，这个函数用户是否使能激光传感器，因为考虑到用户带着光流模块室外飞行，但是室外红外光对 VL53LXX 干扰较大，无法正常使用，这个时候，我们就可以通过遥控关闭 VL53LXX（遥控器固件先升级到 V1.3 或以上版本），然后使用气压计定高，这样 MiniFly 就可以户外定点飞行了（注意定点飞行高度<4m）。


正点原子
ALIENTEK