姿态解算

写在开头

> 为什么要求高精度的姿态解算？
> 云台姿态估计用于求解云台系到惯性系的坐标变换矩阵与姿态欧拉角，准确的姿态信息对目标运动估计和云台运动控制至关重要。

> 消除零漂有什么作用？
> 这个问题用一个例子来反驳，在比赛期间不会产生长时间的静止情况，细微的零漂不会影响到操作手的操作，也就是说可以通过操作来忽略零漂造成的车体转动。但对于一个自瞄系统来说，精确的姿态角是计算的基础。

I2C

I2C是一种半双工、双向二线制同步串行总线。需要两根信号线，数据线SDA（date）和时钟线SCL（clock）。
I2C通信具有几类信号：

- 开始信号
- 结束信号
- 数据信号
- 应答信号

管脚 功能 封装引脚
SCL I2C时钟线 PA8
SDA I2C数据线 PC9

RSTN IST8310的RESET，低电平重启 PG6
DRDY IST8310的数据准备(data ready)中断信号 PG3

HAL_I2C_Mem_Read函数

从寄存器读取数据

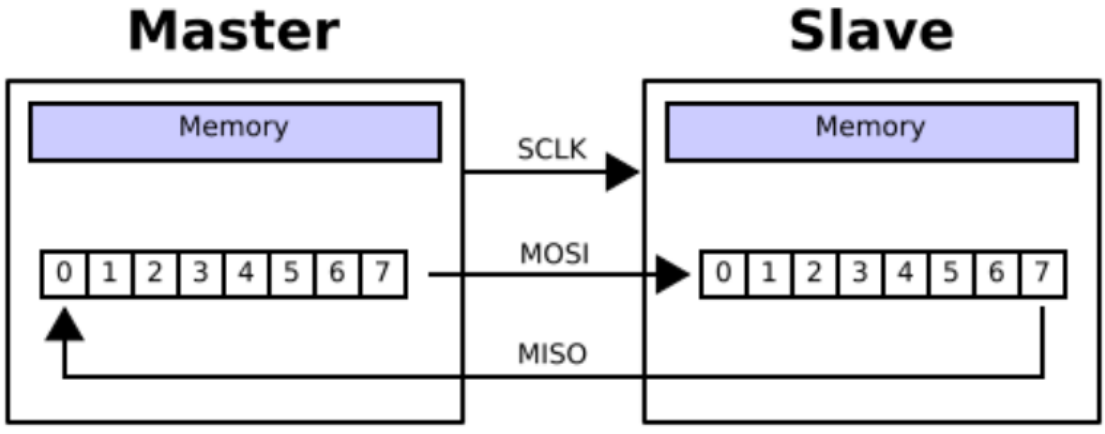
```
HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

参数1：hi2c —— I2C句柄
参数2：DevAddress —— I2C从机地址
参数3：MemAddress —— 寄存器地址
参数4：MemAddSize —— 寄存器地址增加大小
参数5：pData —— 数据指针
参数6：Size —— 数据长度
参数7：Timeout —— 超时时间

SPI

SPI协议是一种高速的，全双工，同步的通信总线。

1. SPI传输需要一个时钟，它是同步通信。
2. SPI以主从方式工作，称为MOSI，（Master output Slave input）



3. NSS就是片选，是SPI从设备是否被选中的，只有片选信号为预先规定的使能信号时（高电位或低电位），对此 SPI 从设备的操作才有效。如果从机没有被选中，主机发送数据从机是不会接收的。
4. Rx FIFO，Tx FIFO：发送缓冲和接收缓冲，当高速通信的时候，数据来不及处理就可以放在缓冲区里面，可以节省一定的时间去处理其他事情。
5. CRC controller：CRC校验，是一种数据检测方式。

四元数

四元数与三维旋转

https://krasjet.github.io/quaternion/quaternion.pdf

复数的相乘其实是旋转与缩放变换的复合

proju (v) 是向量的投影

atan2(b, a)的范围比atan更广

> Theorem 10: 3D 旋转公式（四元数型，一般情况）
>
> 任意向量 $\mathbf{v}$ 沿着以单位向量定义的旋转轴 $\mathbf{u}$ 旋转 θ 度之后的 $\mathbf{v}'$ 可以使用四元数乘法来获得．令 $v = [0, \mathbf{v}]$，$q = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$，那么：
>
> $$v' = qvq^* = qvq^{-1}$$

3D旋转的矩阵形式

> Theorem 11: 3D 旋转公式（矩阵型）
>
> 任意向量 $\mathbf{v}$ 沿着以单位向量定义的旋转轴 $\mathbf{u}$ 旋转 θ 角度之后的 $\mathbf{v}'$ 可以使用矩阵乘法来获得．令 $a = \cos\left(\frac{1}{2}\theta\right)$，$b = \sin\left(\frac{1}{2}\theta\right)u_x$，$c = \sin\left(\frac{1}{2}\theta\right)u_y$，$d = \sin\left(\frac{1}{2}\theta\right)u_z$，那么：
>
> $$\mathbf{v}' = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 \end{bmatrix} \mathbf{v}$$

欧拉公式

$$\cos(\theta) + i\sin(\theta) = e^{i\theta}.$$

万向节死锁(Gimbal Lock)

!! 数学公式推导
https://krasjet.github.io/quaternion/bonus_gimbal_lock.pdf
视频讲解 无伤理解万向死锁现象
https://www.bilibili.com/video/BV1Nr4y1j7kn/?spm_id_from=333.880.my_history.page.click&vd_source=520a72caac2636537f34e1047ea8b17a

由于旋转特定的角度会失去一个自由度，导致需要增加一个向量矩阵才能对那个轴进行旋转。

数学证明如下：

$$
\begin{aligned}
E(\alpha, \frac{\pi}{2}, \beta) &= R_z(\beta)R_y(\frac{\pi}{2})R_x(\alpha) \\
&= \begin{bmatrix} 0 & \cos(\beta)\cdot\sin(\alpha) - \cos(\alpha)\cdot\sin(\beta) & \sin(\alpha)\cdot\sin(\beta) + \cos(\alpha)\cdot\cos(\beta) \\ 0 & \sin(\alpha)\cdot\sin(\beta) + \cos(\alpha)\cdot\cos(\beta) & \cos(\alpha)\cdot\sin(\beta) - \cos(\beta)\cdot\sin(\alpha) \\ -1 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & \sin(\alpha-\beta) & \cos(\alpha-\beta) \\ 0 & \cos(\alpha-\beta) & -\sin(\alpha-\beta) \\ -1 & 0 & 0 \end{bmatrix} \\
&= R_y(\frac{\pi}{2})R_x(\alpha-\beta)
\end{aligned}
$$

旋转矩阵

https://www.cnblogs.com/zhoug2020/p/7842808.html

- 矩阵二维旋转

$$
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}
$$

- 矩阵三维旋转
  ps:绕x轴旋转，没有平移

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 由于矩阵的相乘一般不可交换，所以旋转的顺序会影响最终方向

互补滤波

ω_x ω_y ω_z为陀螺仪测得的角速度,当我们知道前一时刻四元数的值，便可以通过选代法去更新下一时刻四元数的值，以此一步步更新各个时刻的四元数值，也就是说，在初始时刻给定一个四元数的初值（假设零时刻给定q0=1,q1=0,q2=0,q3=0），通过陀螺仪不断测得的角速度去更新四元数，通过四元数就可求得姿态变换矩阵，求出姿态角，也通过四元数不断的更新，这样也就实现了物体姿态的实时更新。

通过叉乘得出偏差角，利用PI调节来消除误差

状态反馈

状态反馈相对于输出反馈的控制效果更好
状态反馈是系统的状态变量通过比例环节传送到输入端去的反馈方式。但是状态变量往往不能从系统外部直接测量到，因此需要更加复杂的技术分析。

- 实验结果
  K436.62 = K0+0.0010696 = 0.0054196 = 1.245885K0
  436.62/4095 = 0.1066227

K240 = K0 + 0.00078225 = 0.00513225 = 1.17982758K0
240/4095=0.05860

K97 = K0 + 0.0003635 = 0.0047135 = 1.08356321K0
97/4095 = 0.023687423

K0=0.00435

K-246 = K0 - 0.000783 = 0.003567

K-452 = K0 - 0.0016673 = 0.00268267

证明线性拟合不可取

DWT

在Cortex-M里面有一个外设叫DWT(Data Watchpoint and Trace)，是用于系统调试及跟踪
它有一个32位的寄存器叫CYCCNT，它是一个向上的计数器，记录的是内核时钟运行的个数，内核时钟跳动一次，该计数器就加1，精度非常高，决定内核的频率是多少，如果是F103系列，内核时钟是72M，那精度就是1/72M = 14ns，而程序的运行时间都是微秒级别的，所以14ns的精度是远远够的。最长能记录的时间为：60s=2的32次方/72000000(假设内核频率为72M，内核跳一次的时间大概为1/72M=14ns)，而如果是H7这种400M主频的芯片，那它的计时精度高达2.5ns（1/400000000 = 2.5），而如果是 i.MX RT1052这种比较牛逼的处理器，最长能记录的时间为： 8.13s=2的32次方/528000000 (假设内核频率为528M，内核跳一次的时间大概为1/528M=1.9ns) 当CYCCNT溢出之后，会清0重新开始向上计数。

卡方检测

卡方检测的作用

使用的情况：当事实与期望不符合情况下使用卡方分布进行检验，看是否系统出了问题，还是属于正常波动。利用卡方分布分析结果，排除可疑结果

卡方检测的自由度

一个式子中独立变量的个数称为这个式子的“自由度”,确定一个式子自由度的方法是:若式子包含有n个独立的随机变量,和由它们所构成的k个样本统计量,则这个表达式的自由度为n-k。

样本统计量：样本统计量（简称统计量）指的是样本的函数，并且此函数不含有未知参数。常见的统计量有：样本均值，样本方差，样本极差等。

- 例子1：比如中包含ξ1，ξ2，…，ξn这n个独立的随机变量，同时还有它们的平均数ξ这一统计量，因此自由度为n-1。
- 例子2：有m×n个数据，总共就有m×n个自由度，扣除均值数目，剩下(m-1)(n-1)个自由度。

卡方检测的应用

在姿态解算中的应用

在四元数更新中，角速度信号存在微小的偏差，经过积分运算之后，变化形成积累误差。这个误差会随着时间延长逐步增加。需要通过重力加速度纠偏。由于重力加速度由加速计测量，当机体处于机动状态时，加速度计测量值产生误差。定义残差为ek。

1.当|ek|<a1时，认为系统不处于机动状态，加速度量测具有较高可信度，因此量测噪声矩阵Rk保持原始值即可。

2.当a1<|ek|<a2时，系统机动使加速度量测与估计值相差较大，应适当增大量测噪声矩阵Rko

3.当|ek|>a2时，系统机动过大，应设置量测噪声矩阵，为无穷大。

在自瞄中的应用

若跟踪过程中miniPC识别的目标发生了切换，卡尔曼滤波器得到的位置量测会与当前位置估计有显著差异，直接进行量测更新会得到一个极大的速度估计值。为解决该问题，通过卡方检验判断跟踪目标是否发生变化。

DSP矩阵运算

#define arm_mat_add_f32

```
arm_status arm_mat_add_f32(

  const arm_matrix_instance_f32 * pSrcA,//第1个参数是矩阵A的源地址。
```

```
  const arm_matrix_instance_f32 * pSrcB,//第2个参数是矩阵B的源地址。


  arm_matrix_instance_f32 * pDst//第3个参数是矩阵A + 矩阵B计算结果存储的地址。

  )//返回值，ARM_MATH_SUCCESS表示成功，ARM_MATH_SIZE_MISMATCH表示矩阵大小不一
```

#define Matrix_Init arm_mat_init_f32

```
void arm_mat_init_f32(

  arm_matrix_instance_f32 * S,//arm_matrix_instance_f32类型矩阵结构体指针变量

  uint16_t nRows,//矩阵的行数

  uint16_t nColumns,//矩阵的列数

  float32_t * pData//矩阵数据地址

  )
```

#define malloc

malloc是动态内存分配函数，用于申请一块连续的指定大小的内存块区域以void*类型返回分配的内存区域地址

malloc只开辟空间，不进行类型检查，只是在使用的时候进行类型的强转。

```
//申请一块长度为10个int的地址
int *p = NULL;
int n = 10;
p = (int *)malloc(sizeof(int)*n);
```

#define memset

memset是一个初始化函数，作用是将某一块内存中的全部设置为指定的值

```
void *memset(

 void *s, //s指向要填充的内存块

 int c, //c指向要被设置的值

 size_t n//是要被设置该值的字符数
);
```

#define memcpy

memcpy函数是C/C++语言中的一个用于内存复制的函数.

```
void *memcpy(
 void *destin, //目标地址
 void *source, //源地址
 unsigned n //数据长度
);
```

姿态解算算法对比

姿态角解算

姿态解算算法主要数据形式有三种，欧拉角，四元数，旋转矩阵。各自有优缺点。

算法	优点	缺点	其他
四元数	直接求解四元数微分方程，四个未知参数，计算量小	漂移比较严重	可用于变换时保存姿态角
欧拉角	直接求解欧拉微分方程，得到pitch、roll、yaw，直观容易理解	求解方程困难，当pitch接近90度时候会出现退化现象	
旋转矩阵	求解姿态矩阵微分方程，避免退化现象	参数量多	可用于向量的表示和变换

!! https://krasjet.github.io/quaternion/bonus_gimbal_lock.pdf

由于欧拉角存在万向节死锁(Gimbal Lock)，旋转矩阵计算量大，四元数求解计算量小，只需解决漂移问题便可以在大多数飞行器上使用，故四元数解算的方法被广泛的应用。

!! Q:为什么会产生漂移？
姿态解算是根据IMU数据求解无人机在空中的姿态，也叫做IMU数据融合。得到陀螺仪的角速率后，按照道理可以用陀螺仪的角速率积分

> 得到角度，因为陀螺仪为三轴传感器，可以得到滚转、俯仰、偏航的角度。但是由于积分的作用，会随着时间产生非常大的积分误差，因而产生偏移。

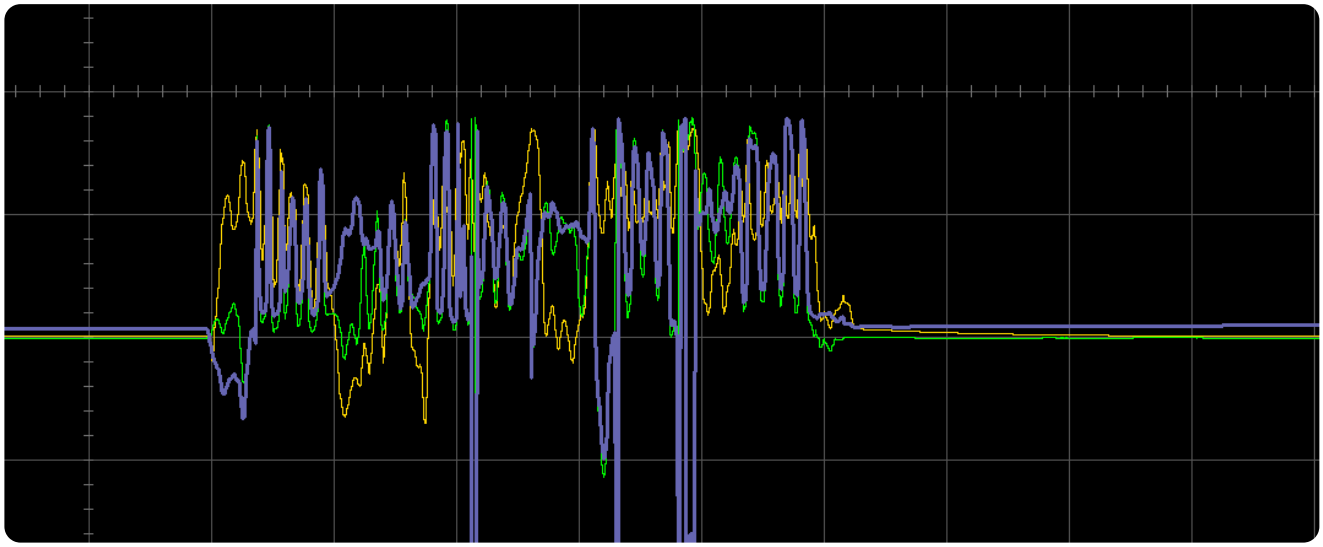
消除漂移

准确快速的姿态解算是无人机姿态控制和位置控制的基础。现在常用的姿态解算方法主要有**互补滤波**、**扩展卡尔曼滤波**。相比较两种方法，互补滤波较为简单，容易调试。接下来先解释互补滤波。

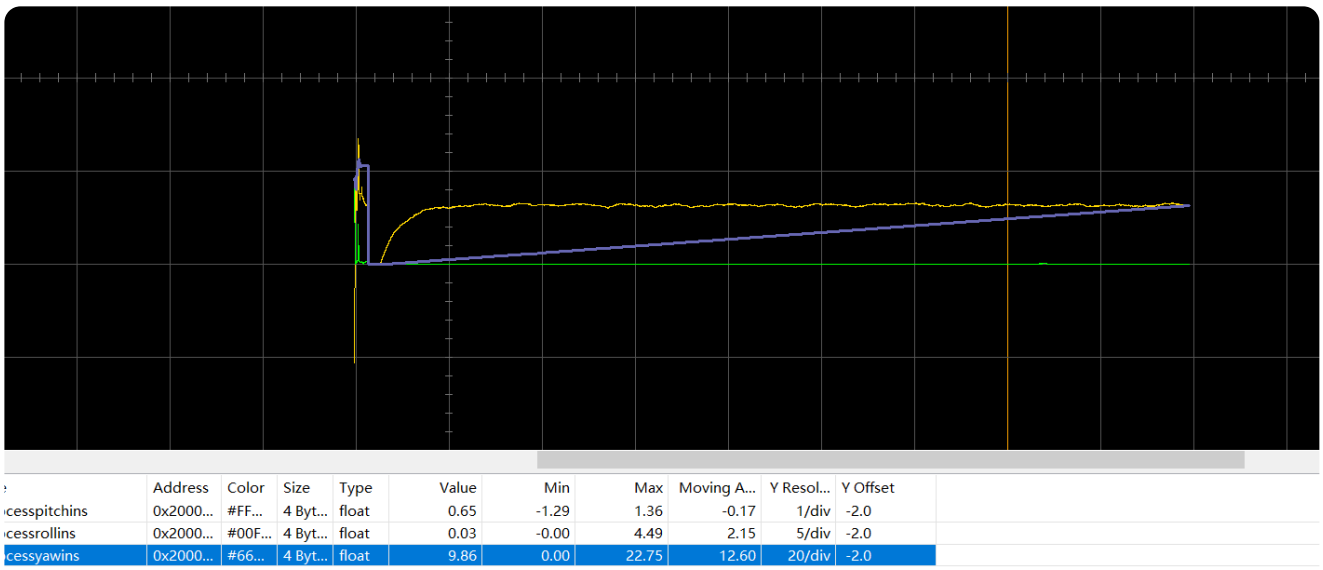
互补滤波

互补滤波是我们队内目前使用的纠正漂移的算法，由于参数固定的原因，仍然会产生微量的零漂。

对队内代码的测试：
在较大加速度情况下，不影响校正



在静止情况下，yaw轴漂移速率为8.8°每分钟



	Address	Color	Size	Type	Value	Min	Max	Moving A...	Y Resol...	Y Offset
cesspitchins	0x2000...	#FF...	4 Byt...	float	0.65	-1.29	1.36	-0.17	1/div	-2.0
cessrollins	0x2000...	#00F...	4 Byt...	float	0.03	-0.00	4.49	2.15	5/div	-2.0
cessyawins	0x2000...	#66...	4 Byt...	float	9.86	0.00	22.75	12.60	20/div	-2.0

根据重力加速度更新四元数

根据 $\boldsymbol{g} = \boldsymbol{C}_n^b \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$，在无运动加速度情况下有：

$$\begin{bmatrix} 2\left(q_1 q_3 - q_0 q_2\right) \\ 2\left(q_2 q_3 + q_0 q_1\right) \\ 1 - 2\left(q_1^2 + q_2^2\right) \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

我们把机体坐标系记为b系(body)，导航坐标系记为n系(Navigation)。学习完《四元数与三维旋转》四元数与三维旋转我们可以知道，向量乘以变换矩阵实际上就实现了对这个向量的旋转和伸缩。C_n^b就是一个把重力加速度从导航坐标系旋转到机体坐标系的矩阵，由四元数q1,q2,q3,q4组成。

```
// Estimated direction of gravity
// 估计的重力加速度方向
        halfvx = q[1] * q[3] - q[0] * q[2];
        halfvy = q[0] * q[1] + q[2] * q[3];
        halfvz = q[0] * q[0] - 0.5f + q[3] * q[3];
```

通过以上代码，我们便将重力加速度从n系转移到b系。将在b系的理论重力加速度(v^)和加速计测量的量测重力加速度(v¯)进行向量积。理论重力加速度向量和实际重力加速度向量均是向量，反应向量间夹角关系的运算有两种：内积（点乘）和外积（叉乘），考虑到向量外积模的大小与向量夹角呈正相关，故通过计算外积来得到向量方向差值 θ：

$$|\boldsymbol{\rho}| = |\overline{\boldsymbol{v}}| \cdot |\hat{\boldsymbol{v}}| \cdot \sin\theta$$

考虑到实际情况中理论向量 v^ 和测量向量 v¯ 偏差角不会超过5°，而当θ在±5°内时，sinθ 与θ的值非常接近，因此上式可进一步简化为：

$$|\boldsymbol{\rho}| = \theta$$

用这个向量偏差可以构件PI补偿器来计算角速度补偿值

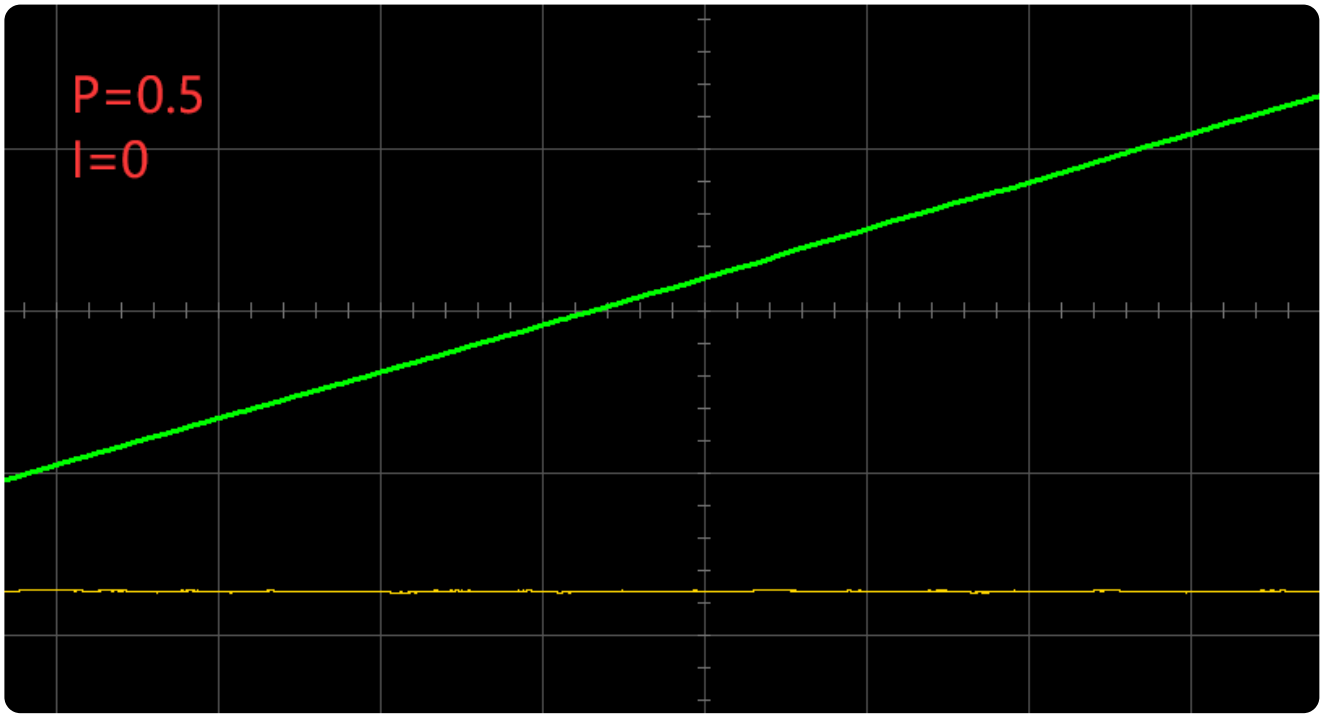
```
// Compute and apply integral feedback if enabled
        if(twoKi > 0.0f) {
// integral error scaled by Ki  对误差进行积分运算
            integralFBx += twoKi * halfex * (1.0f / sampleFreq);
            integralFBy += twoKi * halfey * (1.0f / sampleFreq);
            integralFBz += twoKi * halfez * (1.0f / sampleFreq);
            gx += integralFBx;  // apply integral feedback
            gy += integralFBy;
            gz += integralFBz;
        }
        else {
            integralFBx = 0.0f; // prevent integral windup
            integralFBy = 0.0f;
            integralFBz = 0.0f;
        }

// Apply proportional feedback
        gx += twoKp * halfex;
        gy += twoKp * halfey;
        gz += twoKp * halfez;
    }
```

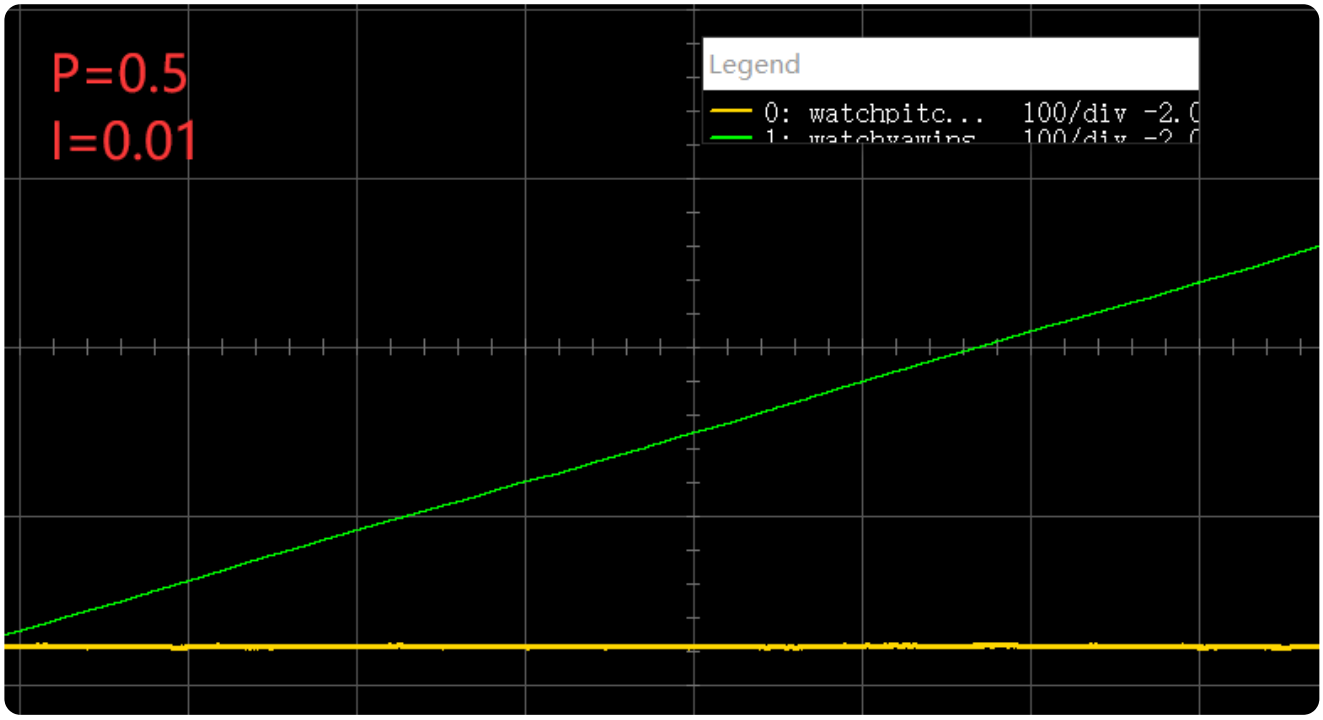
$$GyroError = K_P \cdot \text{error} + K_I \cdot \int \text{error}$$

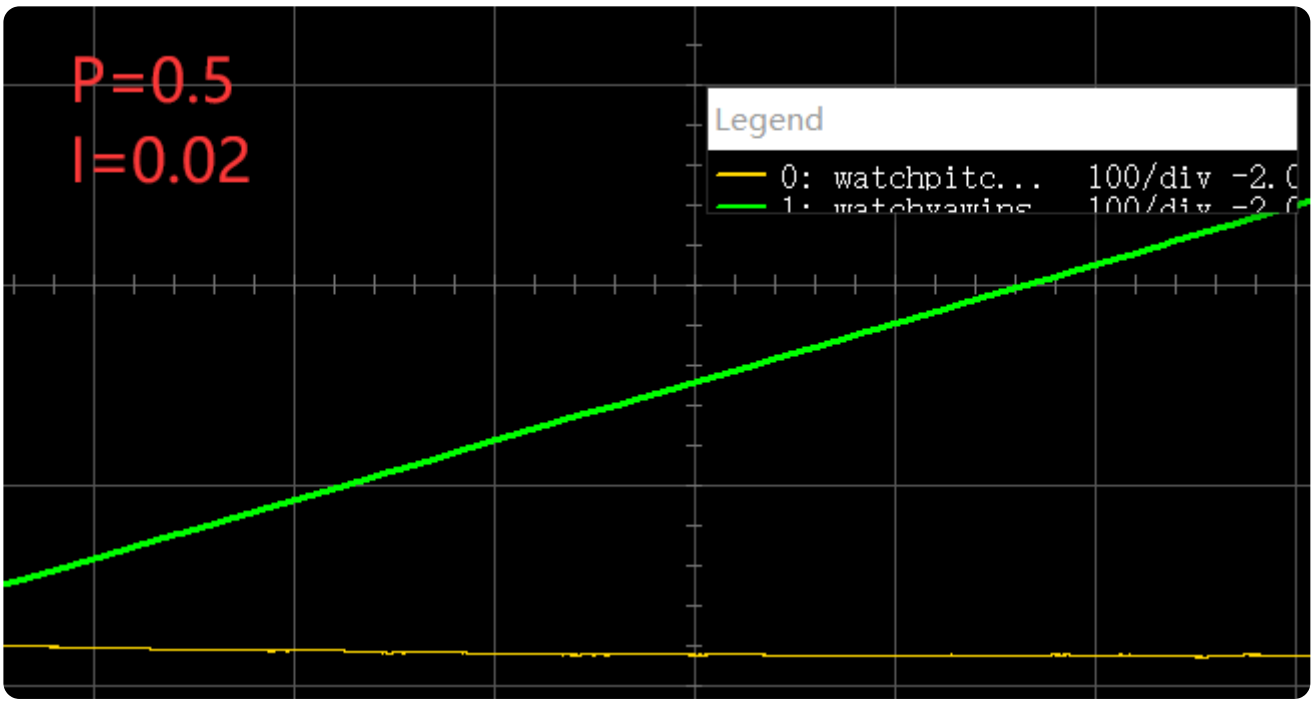
其中，比例项用于控制传感器的“可信度”，积分项用于消除静态误差。KP越大，意味着通过加速度计得到误差后补偿越显著，即是越信任加速度计。反之KP越小时，加速度计对陀螺仪的补偿作用越弱，也就越信任陀螺仪。而积分项则用于消除角速

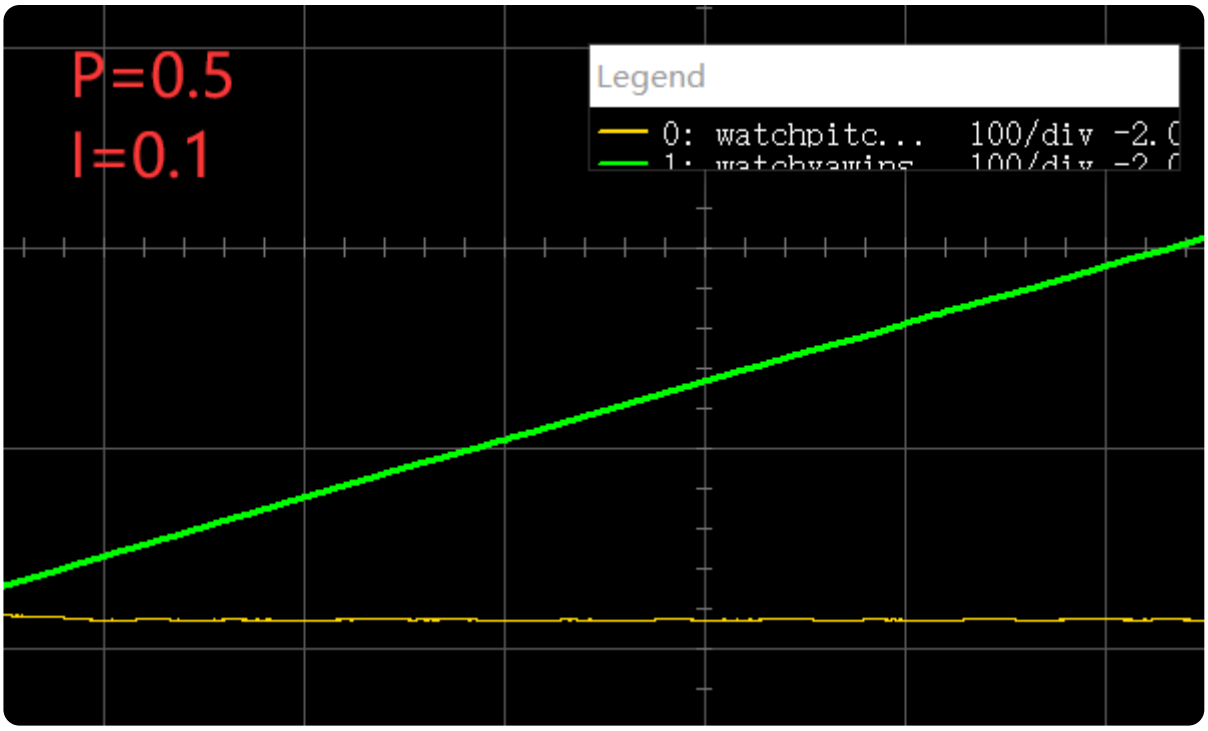
度测量值中的有偏噪声，故对于经过零篇矫正的角速度测量值，一般选取很小的KI。最后将补偿值补偿给角速度测量值，带入四元数差分方程中即可更新当前四元数。

下面进行一些调参尝试，绿色曲线为yaw轴









通过调参我们可以发现，在静止情况下，加大I也就是积分可以抑制陀螺仪测量的角速度所产生的有偏噪声，但并没有通过调参来消除零漂。并且在较大加速度的情况下，积分产生的误差将会难以接受。

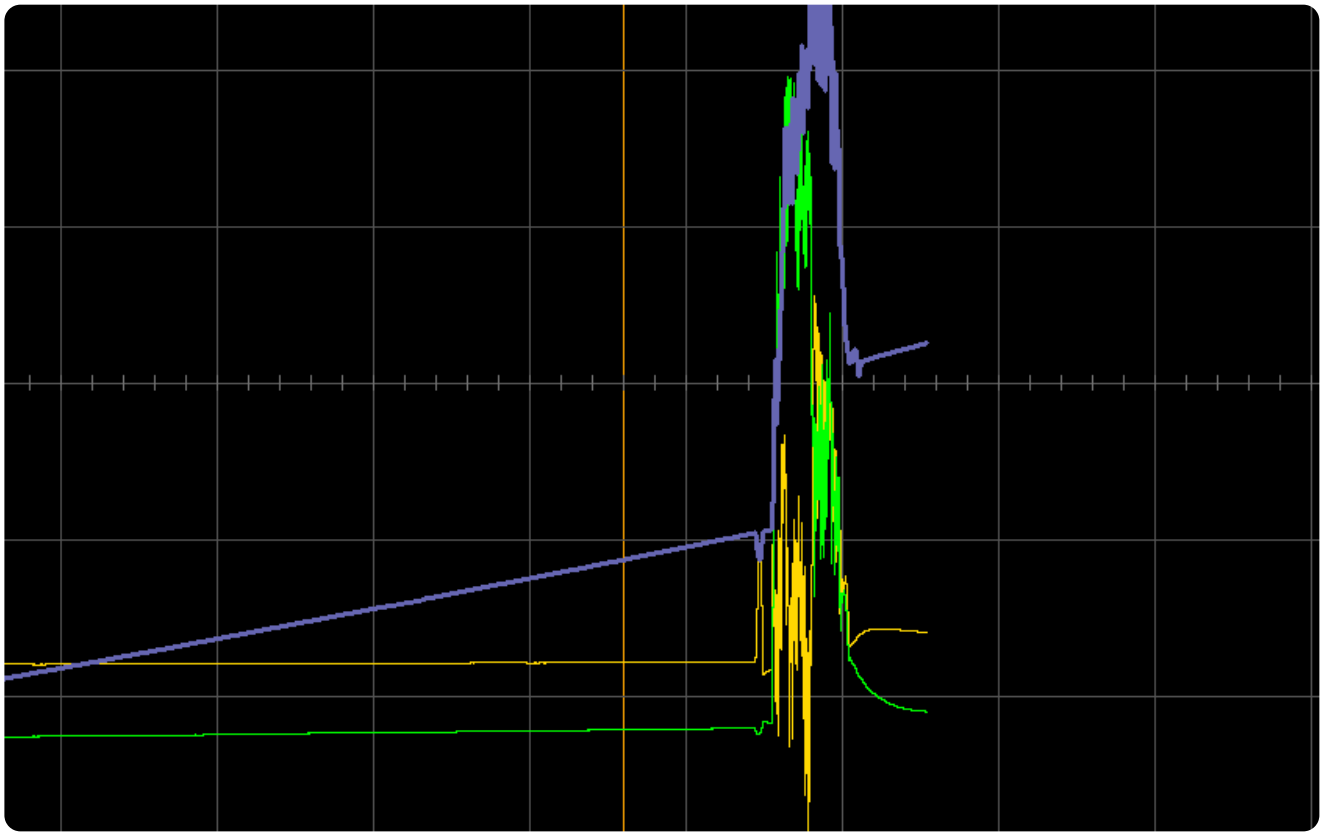
队内目前的代码参数设置为P=0.5，I=0.0，因此不会产生过大加速度严重影响积分导致发散的情况。综合网上开源的算法和经验，我们可以限制积分上限输出来避免发散，加入卡方检测来禁止过大加速度对纠偏的影响。

根据磁力计数据更新四元数

#操作系统

vTaskDelay/osDelay()函数用于阻塞延时，调用函数后，任务将进入阻塞状态。阻塞时长由系统时钟节拍（SysTick）和节拍数(函数实参)决定，并且阻塞时长由该函数调用时开始计算。因此，vTaskDelay/osDelay()函数并不适用于周期性执行任务的场合。

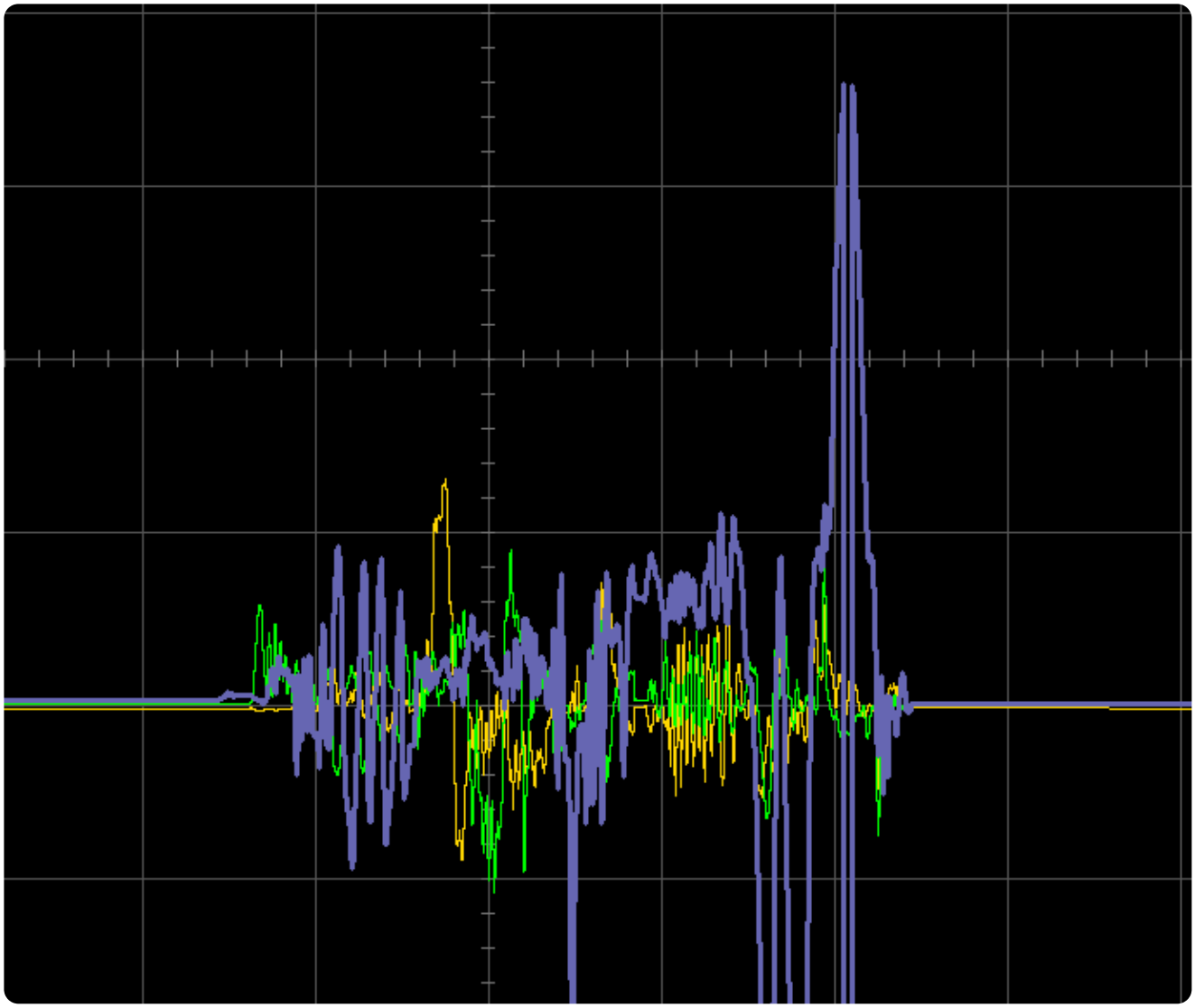
尝试用磁力计解决零漂,反复实验效果不稳定。



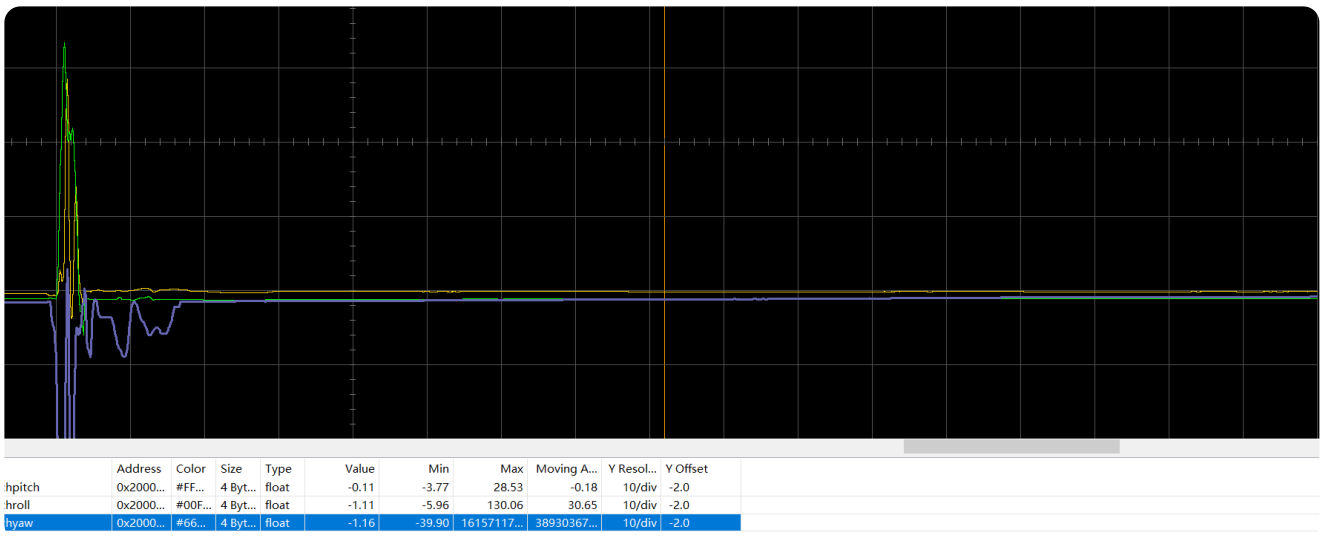
由于固定参数的适应性差，自适应互补滤波的资料尚未了解，我们即将采用拓展卡尔曼来纠偏，并消除零漂。

拓展卡尔曼

对于开源代码的测试：
通过手动摇晃测试较大加速度对纠偏的影响，从下图中我们可以看到，经过较大的加速度，纠偏不会出现错误。



通过较长时间的静止测试yaw轴零漂速率。



	Address	Color	Size	Type	Value	Min	Max	Moving A...	Y Resol...	Y Offset
hpitch	0x2000...	#FF...	4 Byt...	float	-0.11	-3.77	28.53	-0.18	10/div	-2.0
hroll	0x2000...	#00F...	4 Byt...	float	-1.11	-5.96	130.06	30.65	10/div	-2.0
hyaw	0x2000...	#66...	4 Byt...	float	-1.16	-39.90	16157117...	38930367...	10/div	-2.0

拓展卡尔曼更新流程图

读取陀螺仪数据，获得角速度
ωx,ωy,ωz

计算当前的离散时间
状态转移矩阵Ak

估计‘先验’系统状态q^¯k

‘先验’噪声协方差计算P¯k

计算h1

计算雅可比矩阵Hk1

读取加速度计数据Zk1
ax,ay,az

计算‘后验’系统状态q^k

计算‘后验’协方差矩阵Pk

状态外推方程

$$\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}\mathbf{q}$$

四元数微分方程

对四元数微分方程进行离散化

$$\begin{aligned}\mathbf{q}_{k+1} &= \mathbf{q}_k + \dot{\mathbf{q}}_k \Delta t \\ &= (\boldsymbol{I}_{4\times 4} + \frac{1}{2}\boldsymbol{\Omega}_k \Delta t)\mathbf{q}_k\end{aligned}$$

$$A_k = (I + A_{\mathrm{TC}}T) = \left(I + \frac{1}{2}\Omega_{nb}^{n}T\right).$$

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

计算先验估计 Ak为状态转移矩阵

$$\hat{\mathrm{q}}_{\mathrm{k}}^{-} = \mathrm{A}_{\mathrm{k}}\hat{\mathrm{q}}_{\mathrm{k-1}}$$

计算先验协方差 Qk为过程噪声矩阵

$$P_k^- = A_k P_{k-1} A_k^{\mathrm{T}} + Q_k.$$

h1为重力加速度g经过旋转从导航坐标系旋转到机体坐标系的矩阵

$$h_1(q_k) = \hat{g} = R_n^b \begin{bmatrix} 0 \\ 0 \\ |g| \end{bmatrix} = |g| \begin{bmatrix} 2q_1q_3 - 2q_0q_2 \\ 2q_0q_1 + 2q_2q_3 \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

雅可比矩阵计算公式

$$H_{k1} = \frac{\partial h_{1[i]}}{\partial q_{[j]}} = \begin{bmatrix} -2q_2 & 2q_3 & -2q_0 & 2q_1 \\ 2q_1 & 2q_0 & 2q_3 & 2q_2 \\ 2q_0 & -2q_1 & -2q_2 & 2q_3 \end{bmatrix}.$$

这里得到的后验qk实际上就是我们更新完成的四元数
对该四元数反解，即可得到roll，pitch，yaw三个角度

$$\hat{\mathrm{q}}_{\mathrm{k}} = \hat{\mathrm{q}}_{\mathrm{k}}^{-} + \mathrm{K}_{\mathrm{k}}\left(\mathrm{z}_{\mathrm{k}} - \mathrm{h}(\hat{\mathrm{q}}_{\mathrm{k}}^{-}, 0)\right)$$

$$\mathrm{P}_{\mathrm{k}} = (\mathrm{I} - \mathrm{K}_{\mathrm{k}}\mathrm{H}_{\mathrm{k}})\mathrm{P}_{\mathrm{k}}^{-}$$