1. IMU 误差来源

> MEMS惯性器件的误差一般分成两类：系统性误差和随机误差。系统性误差本质就是能找到规律的误差，所以可以实时补偿掉，主要包括常值偏移、比例因子、轴安装误差等。但是随机误差一般指噪声，无法找到合适的关系函数去描述噪声，所以很难处理。一般采用时间序列分析法对零点偏移的数据进行误差建模分析，可以用卡尔曼滤波算法减小随机噪声的影响。

IMU的误差来主要来自于三部分，包括噪声(Bias and Noise)、尺度因子(Scale errors)和轴偏差(Axis misalignments)。

加速度计和陀螺仪的测量模型可以用式(1.1)和式(1.2)表达。

$$a^B = T^a K^a (a^S + b^a + \nu^a) \qquad (1.1)$$

$$w^B = T^g K^g (w^S + b^g + \nu^g) \qquad (1.2)$$

中上标a 表示加速度计，g 表示陀螺仪，B 表示正交的参考坐标系，S 表示非正交的选准坐标系。T 表示轴偏差的变换矩阵，K 表示尺度因子，a^S, w^S表示真值，b,ν 分别表示Bias和白噪声。

1.1 IMU噪声来源

$$\tilde{\omega}(t) = \omega（t） + b(t) + n(t) \qquad (1.3)$$

n(t) 表示高斯白噪声，b(t) 表示随机游走噪声

1. 高斯白噪声
   服从高斯分布的一种白噪声，其一阶矩为常数，二阶距随时间会发生变化。一般来说这种噪声是由AD转换器引起的一种外部噪声。因为连续的高斯白噪声服从如下条件(参考An Introduction to Stochastic PDEs中Example 3.56 (White noise))，

$$E(n(t)) \equiv 0 \qquad (1.4)$$
$$E(n(t_1)n(t_2)) = \sigma_g^2 \delta(t_1 - t_2)$$

   其中 δ(t) 表示狄拉克函数。σg 位高斯白噪声的方差，值越大，表示噪声程度越大。将其进一步离散化后，得到

$$n_d[k] = \sigma_{gd}\omega[k] \qquad (1.5)$$

$$\omega[k] \sim N(0,1), \sigma_{gd} = \sigma_g \frac{1}{\sqrt{\triangle t}}。$$

$$n_d[k] = n(t_0 + \Delta t) \simeq \frac{1}{\Delta t}\int_{t_0}^{t_0+\Delta t} n(\tau)d\tau$$

$$\begin{aligned} E(n_d[k]^2) &= E(\frac{1}{\Delta t^2}\int_{t_0}^{t_0+\Delta t}\int_{t_0}^{t_0+\Delta t} n(\tau)n(t)d\tau dt) \\ &= \frac{\sigma_g^2}{\Delta t^2}\int_{t_0}^{t_0+\Delta t}\int_{t_0}^{t_0+\Delta t}\delta(t-\tau)d\tau dt \\ &= \frac{\sigma_g^2}{\Delta t} \end{aligned}$$

$$\text{则有 } \sigma_{gd}^2 = \frac{\sigma_g^2}{\Delta t},\ \text{即 } \sigma_{gd} = \frac{\sigma_g}{\sqrt{\Delta t}}$$

2. 随机游走噪声(这里指零偏Bias)

   随机游走是一个离散模型，可以把它看做是一种布朗运动，或者将其称之为维纳过程。该模型可以看做是高斯白噪声的积分。该噪声参数一般是由传感器的内部构造、温度等变化量综合影响下的结果。

$$\dot{b_g}(t) = \sigma_{bg}\omega(t) \qquad (1.6)$$

   其中 $\omega(t)$ω(t) 是单位的高斯白噪声，在上面提到了。如果把随机游走噪声离散化，可以写作：

$$b_d[k] = b_d[k-1] + \sigma_{bgd}\omega[k] \qquad (1.7)$$

$$\text{其中 } \omega[k] \sim (0,1),\ \sigma_{bgd} = \sigma_{bg}\sqrt{\triangle t}$$

$$b_d[k] \simeq b(t_0) + \int_{t_0}^{t_0+\Delta t} n(t)dt$$

$$\begin{aligned} E((b_d[k]-b_d[k-1])^2) &= E(\int_{t_0}^{t_0+\Delta t}\int_{t_0}^{t_0+\Delta t} n(t)n(\tau)dt\tau) \\ &= \sigma_{bg}^2\int_{t_0}^{t_0+\Delta t}\int_{t_0}^{t_0+\Delta t}\delta(t-\tau)dtd\tau \\ &= \sigma_{bg}^2\Delta t \end{aligned}$$

$$\text{则有 } \sigma_{bgd}^2 = \sigma_{bg}^2\Delta t,\ \text{即}\sigma_{bgd} = \sigma_{bg}\sqrt{\Delta t}$$

由随机游走的分布可以看出，随机游走都是在上一次的噪声的基础之上叠加了一个高斯噪声，所以下一步永远都是随机的。

误差模型部分的内容和推导参考Kalibr wiki和郑凡博客。

1.2 IMU尺度因子

尺度误差：这部分的误差来自于传感器的数字信号向物理量转换的误差，比如说AD向加速度/角速度转换的时候的误差。

$$K^a = \begin{bmatrix} s_x^a & 0 & 0 \\ 0 & s_x^a & 0 \\ 0 & 0 & s_x^a \end{bmatrix}$$
$$K^g = \begin{bmatrix} s_x^g & 0 & 0 \\ 0 & s_x^g & 0 \\ 0 & 0 & s_x^g \end{bmatrix} \qquad (1.8)$$

1.3 IMU轴向偏差

一般情况下，加速度计的坐标系`AF`和陀螺仪的坐标系`GF`都不是正交的坐标系，但我们正常使用的时候都是默认测量量是在正交坐标系下的，所以就需要一个变换矩阵将测量量从非正交坐标系`AF\GF`下转到正交坐标系`BF`下，将其称之为机体坐标系

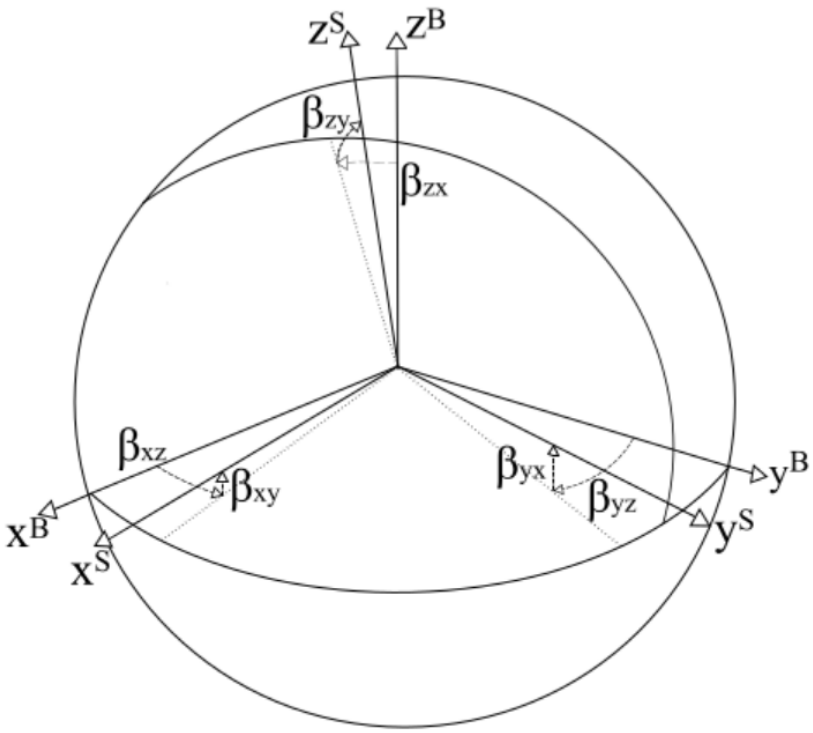


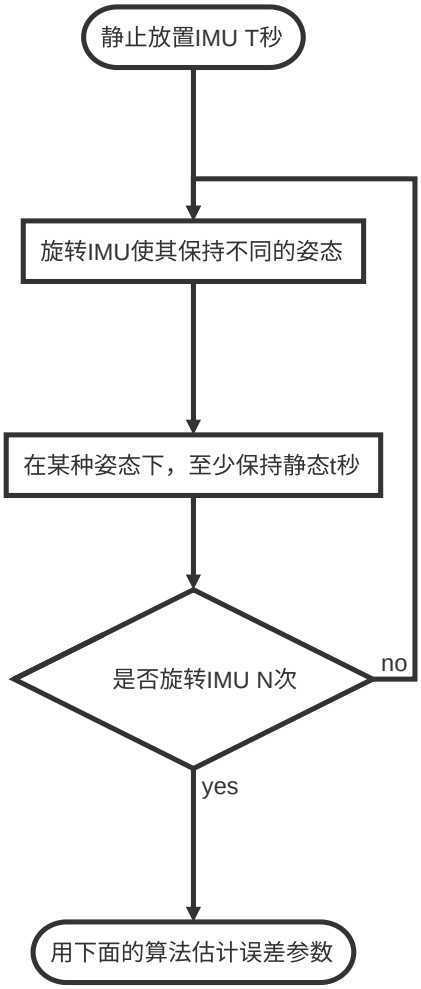
Fig. 2. Non-orthogonal sensor (accelerometers or gyroscopes) axes (x^S, y^S, z^S), and body frame axes (x^B, y^B, z^B).

所以这部分的误差就是一个变换矩阵，求取实际的旋转坐标系(AF/GF)到参考坐标系(BF)的转换。为了计算变换矩阵，将三个轴偏角进行进一步的分解，如图2所示。将每个轴向的偏差角沿着另外两个轴分解即可得到。

$$s^B = Ts^A, \qquad T = \begin{bmatrix} 1 & -\beta_{yz} & \beta_{zy} \\ \beta_{xz} & 1 & -\beta zx \\ \beta_{xy} & \beta_{yx} & 1 \end{bmatrix} \tag{1.9}$$

2 IMU 校准

IMU在校准过程中，加速度计和陀螺仪是分开校准的，一般是先校准加速度计，然后利用准确的加速度计信息再来校准陀螺仪。整个校准过程的流程如下图所示，



其中，初始化时间 T一般取50s，旋转后保持静态时间t 取 1~4s，旋转次数N 取 36~50 次。一般来说，这个旋转次数越多越好，至少旋转次数要大于要求解的参数的个数，这样才能避免奇异性。

2.1 静态检测

因为在加速度和陀螺仪的校准过程中，常常需要传感器处于静止状态，所以在校准的过程中也就需要判断传感器是否处于静止状态。判断依据也很简单，

$$\varsigma(t_w) = \sqrt{[var_{t_w}(a_x^t)]^2 + var_{t_w}(a_y^t)]^2 + var_{t_w}(a_z^t)]^2} \tag{2.1}$$

其中 $var_{tw}(a^t)$表示加速度a^t在时间段t_w内的方差。

判断的时候，只需将$\varsigma(t_w)$和阈值$\varsigma(T_init)$比较即可，小于则静态，反之动态。一般取T_{init}为$50s$。

2.2 加速度计校准

加速度计的校准过程一般都是将加速度计静止，然后根据测量值的二范数等于当地重力加速度这一规则校准的。一般会用上一小节提到的静态检测算法在采样序列上挑选出静态的测量片段，然后按照下述内容进行校准。

Method A: 常规校准方式：

将变换矩阵ΓT 简单化，我们假设坐标系BF的x轴和坐标系AF的x轴重合，且BF的y轴处于AF的x轴和y轴的平面上。所以变换矩阵可以进一步写为，

$$T^a = \begin{bmatrix} 1 & -\alpha_{yz} & \alpha_{zy} \\ 0 & 1 & -\alpha_{zx} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

参数：

$$\theta^{acc} = [\alpha_{yz}, \alpha_{zy}, \alpha_{zx}, s_x^a, s_y^a, s_z^a, b_x^a, b_y^a, b_z^a] \tag{2.3}$$

状态方程：

$$a^O = h(a^S, \theta^{acc}) = T^a K^a (a^S + b^a) \tag{2.4}$$

因为在采集加速度计读数的时候取得时一个小窗口内的平均值，所以忽略了高斯白噪声。进而有了优化的代价函数。

$$L(\theta^{acc}) = \sum_{k=1}^{M} (||g||^2 - ||h(a_k^S, \theta^{acc})||^2)^2 \tag{2.5}$$

一般会有多组MM 组较为明显的、稳定的旋转量来放入代价函数中求解待求解参数，然后选取残差最小的一组所对应的参数即可。

Method B: 六面法校准加速度计：

$$A_t = \begin{bmatrix} A_x \\ A_y \\ A_Z \end{bmatrix} = R_{[3*3]} \begin{bmatrix} Scale_x & 0 & 0 \\ 0 & Scale_y & 0 \\ 0 & 0 & Scale_z \end{bmatrix} \begin{bmatrix} A_{mx} - offset_x \\ A_{my} - offset_y \\ A_{mz} - offset_z \end{bmatrix} \tag{2.6}$$

其中A_t表示最终的真实值，A_{m^*}表示测量值，$offset_{xyz}$表示测量中的零偏，$R[3*3]$表示旋转，$Scale_{xyz}$表示尺度缩放。将下式做变换得，

$$\begin{bmatrix} A_x \\ A_y \\ A_Z \end{bmatrix} = \begin{bmatrix} C_1 & C_2 & C_3 \\ C_4 & C_5 & C_6 \\ C_7 & C_8 & C_9 \end{bmatrix} \begin{bmatrix} A_{mx} \\ A_{my} \\ A_{mz} \end{bmatrix} + \begin{bmatrix} C_{off_x} \\ C_{off_y} \\ C_{off_z} \end{bmatrix} \tag{2.7}$$

其变为齐次坐标形式

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \begin{bmatrix} C_1 & C_2 & C_3 & C_{off_x} \\ C_4 & C_5 & C_6 & C_{off_y} \\ C_7 & C_8 & C_9 & C_{off_z} \end{bmatrix} \begin{bmatrix} A_{mx} \\ A_{my} \\ A_{mz} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} A_{mx} & A_{mx} & A_{mx} & 1 \end{bmatrix} \begin{bmatrix} C_1 & C_4 & C_7 \\ C_2 & C_5 & C_8 \\ C_3 & C_6 & C_9 \\ C_{off_x} & C_{off_y} & C_{off_z} \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \tag{2.8}$$

这时候就可以最小二乘求解上面的方程了。写作 $\beta = (X^T X)^{-1} X^T Y$,其中 β 为上面的 4∗3 矩阵， X 是左边的测量值， Y 是右边的真值。**关于真值我们可以把加速度计按三个轴向6个位置放置（ X 轴正面朝下： (g,0,0) .....)。**这样就对加速度计做了校准，这种方法也称为 **六面校准法。**

2.3 陀螺仪校准

陀螺仪的校准有两个部分：Allan方差(艾伦方差)校准Bias和优化方式求解尺度因子及轴偏差。在校准陀螺仪的时候，要使用到加速度的校准信息，所以加速度校准的好坏关系到整个IMU的校准效果。

2.3.1 Allan方差校准陀螺仪Bias

在IMU校准的最开始，要将IMU静止搁置时间 T_{init}(一般取50s)。Allan方差的求取可以参考文献[3]。
在Allan方差分析中，共有5个噪声参数：量化噪声、角度随机游走、零偏不稳定性、速度随机游走和速度爬升。

https://github.com/XinLiGH/GyroAllan

Allan方差分析陀螺仪偏差（完整）

1. allan 艾伦方差法： Allan方差法是20世纪60年代由美国国家标准局的David Allan提出的，它是一种基于时域的分析方法。Allan方差法的主要特点是能非常容易地对各种误差源及其对整个噪声统计特性的贡献进行细致的表征和辨识，而且具有便于计算、易于分离等优点。最初由David W. Allan开发，用于测量精密仪器的频率稳定性。 它还可用于识别固定陀螺仪测量中存在的各种噪声源。

   当Allan标准差的拟合多项式中的拟合系数是负值时，所得误差项的拟合结果随着相关时间的微小改变变化很大，拟合误差很大，可信度差。

2. allan方法

   (1)将陀螺仪静止放置时间T ,单个采样周期为 τ_0，共有N 组采样值。
   (2)计算单次采样输出角度 θ 和 平均因子 m， m 要尽量取得均匀。

$$\begin{aligned} \theta(n) &= \int^{\tau_0} \Omega(n) dt \\ m &= Rand(1, \frac{N-1}{2}) \end{aligned} \tag{2.9}$$

$$\sigma^2(\tau) = \frac{1}{2m^2(N-2m)} \sum_{j=1}^{N-2m} \left\{ \sum_{i=\mathrm{K}}^{j+m-1} (\bar{\Omega}_{\mathrm{K}+m}(\tau) - \bar{\Omega}_{\mathrm{K}}(\tau))^2 \right\} \tag{8}$$

(3)计算Allan方差，当 m 取不同的值的时候会有不同的Allan方差值。

$$\theta^2(\tau) = \frac{1}{2\tau^2(N-2m)} \sum_{k=1}^{N-2m} (\theta_{K+2m} - 2\theta_{K+m} + \theta_K) \tag{2.10}$$

其中，$\tau = m * \tau_0$.

(4)一般在绘制Allan方差曲线的时候使用的是Allan方差的平方根，所以将式(2.10)中的计算结果去平方根即可。

3. matlab 伪代码

```
function [T,sigma] = allan(omega,fs,pts)
[N,M]  = size(omega);                % figure out how big the output data set is
n      = 2.^(0:floor(log2(N/2)))';   % determine largest bin size
maxN = n(end);
endLogInc = log10(maxN);
m = unique(ceil(logspace(0,endLogInc,pts)))';  % create log spaced vector average factor
t0     = 1/fs;                               % t0 = sample interval
T      = m*t0;                               % T = length of time for each cluster
theta  = cumsum(omega)/fs;      % integration of samples over time to obtain output angle θ
sigma2 = zeros(length(T),M);    % array of dimensions (cluster periods) X (#variables)
for i=1:length(m)               % loop over the various cluster sizes
  for k=1:N-2*m(i)            % implements the summation in the AV equation
    sigma2(i,:) = sigma2(i,:) + (theta(k+2*m(i),:) - 2*theta(k+m(i),:) + theta(k,:)).^2;
  end
end
sigma2 = sigma2./repmat((2*T.^2.*(N-2*m)),1,M);
sigma  = sqrt(sigma2)
```

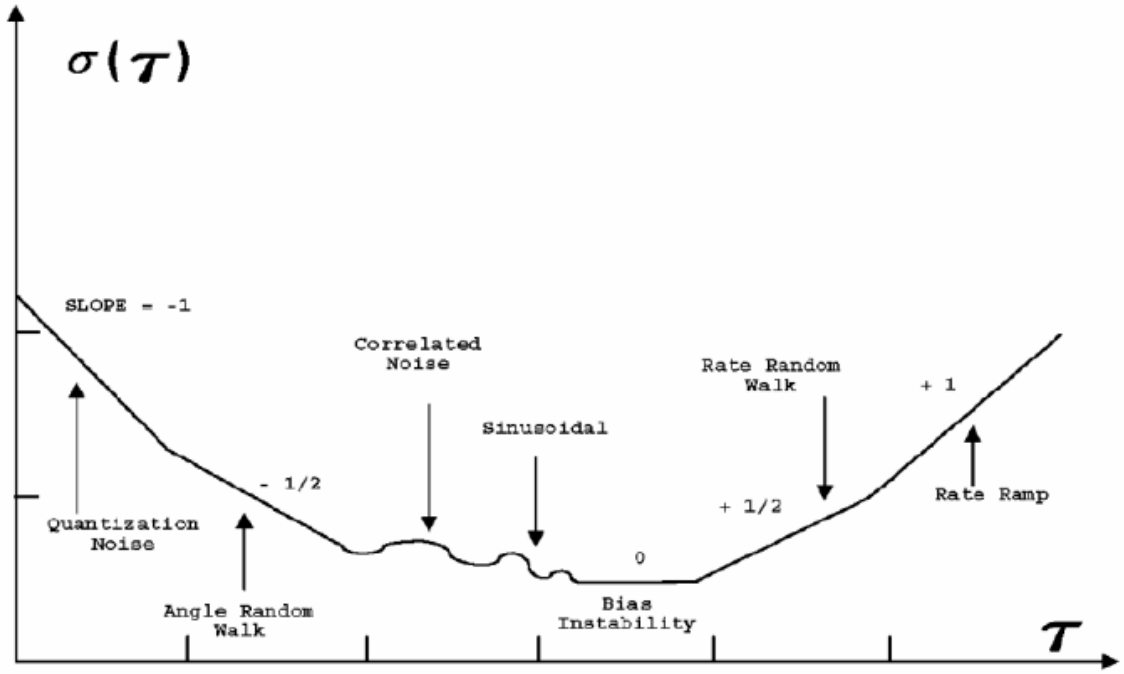
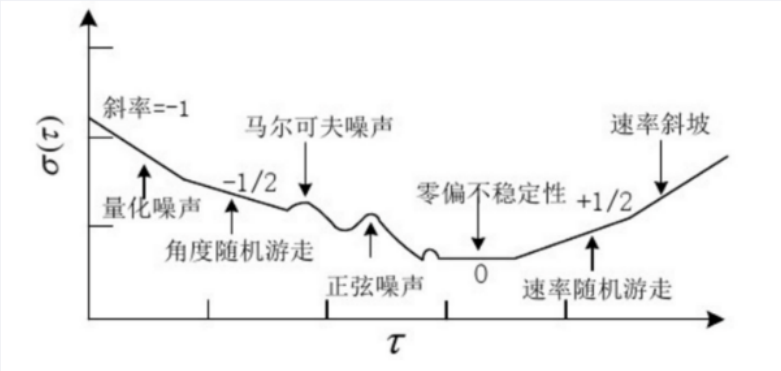


Figure 2. Different Noise processes on an Allan Variance Plot

下面是具体的matlab程序： 此处需要注意的是，数据量要足够多，否则会反映不出来噪声特性。我使用了大概120000组数据，采样时间是5ms。（对于数据格式的要求）
具体可以参考下图：



不同段的Allan方差曲线代表了不同的误差参数，我们要求解的零偏噪声(Bias)对应的曲线段的斜率就是0。

4. 计算各部分的噪声参数

基于上述的角度均值计算，计算方差，依据下面的allan方差模型进行参数标定。

在保证测试环境稳定定情况下，可以认为各噪声源是独立的，那么计算出的Allan方差就是各部分误差的平方和，

$$\begin{aligned}\sigma_{total}^2(\tau) &= \sigma_Q^2(\tau)+\sigma_N^2(\tau)+\sigma_B^2(\tau)+\sigma_K^2(\tau)+\sigma_R^2(\tau)\\ &= \frac{3Q^2}{\tau^2}+\frac{N^2}{\tau}+\frac{2B^2}{\pi}ln2+\frac{K^2\tau}{3}+\frac{R^2\tau^2}{2}\end{aligned} \tag{2.10}$$

$$\sigma_{total}^2(\tau)=\sum_{i=-2}^{2}C_i\tau^i \tag{2.11}$$

根据第一步中得到的 τ 和 Allan方差值，依据最小二乘法进行拟合，进而得到 Ci 的值，最终可以得到相应的误差系数，

$$\begin{cases} Q=\frac{\sqrt{C_{-2}}}{3600\sqrt{3}}(^\circ) \\ N=\frac{\sqrt{C_{-1}}}{60}(^\circ)\sqrt{h} \\ B=\frac{\sqrt{C_0}}{0.664}(^\circ)h \\ K=60\sqrt{3C_1}[(^\circ)\cdot h^{-1}]/\sqrt{h} \\ R=3600\sqrt{2C_2}[(^\circ)\cdot h^{-1}]/h \end{cases} \tag{2.12}$$

以上就是通过Allan方差分析得到陀螺仪Bias的过程，一般要采集好几个小时的数据。如果仅需要，零偏参数，则在初始放置的50s左右的时间就足够了。

2.3.2 优化方式求解尺度因子及轴偏差

陀螺仪的剩余参数在校准的时候，挑选的是加速度计校准过程中的静态采样片段夹杂的动态片段。刚好可以使用校准过的静态片段测量值的平均值作为初始的重力向量和旋转完成后的实际重力向量。

在已知陀螺仪的Bias之后，还需要校准的参数有：

$$\theta^{gyro}=[\gamma_{yz},\gamma_{zy},\gamma_{xz},\gamma_{zx},\gamma_{xy},\gamma_{yx},s_x^g,s_y^g,s_z^g] \tag{2.13}$$

其他参数的校准的思路也比较简单，即“陀螺仪任意转动，积分得到的角度和加速度的测量值求得的角度做比较即可”这里把加速度的测量值当做了参考值，所以前面加速度的校准是非常重要的。

设由加速度计测量值得到的一个初始的加速度向量 $u_{a,k-1}$， n个陀螺仪测量值 w_i，则可以得到陀螺仪积分得到的旋转矩阵旋转之后新的重力向量。$u_{g,k}$，整个过程描述为，

$$u_{g,k}=\Psi[w_i^S,u_{a,k-1}] \tag{2.14}$$

按照上面的描述，进一步得到代价函数:

$$L(\theta^{gyro})=\sum_{k=2}^{M}||u_{a,k}-u_{g,k}||^2 \tag{2.15}$$

其中$u_{a,k}$为旋转结束之后，由加速度的测量值得到的实际的重力向量。

在式(2.14)中会涉及到离散时间的陀螺仪积分问题，这里选择使用四阶龙格库塔法(Runge-Kutta, RK4n)。具体的积分过程描述如下。

定义四元数的导数(局部干扰得到)

$$f(q,t) = \dot{q} = \frac{1}{2}\Omega(w(t))q$$

RK4的具体过程为:

$$q_{k+1} = qk + \Delta t \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.16a}$$

$$k_i = f(q^{(i)}, t_k + c_i \Delta t) \tag{2.16b}$$

$$q^{(i)} = q_k \tag{2.16c}$$

$$q^{(i)} = q_k + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j \tag{2.16d}$$

$$c_1 = 0, \quad c_2 = \frac{1}{2}, \quad c_3 = \frac{1}{2}, \quad c_4 = 1$$

$$a_{21} = \frac{1}{2}, \quad a_{31} = 0, \quad a_{41} = 0,$$

$$a_{32} = \frac{1}{2}, \quad a_{42} = 0, \quad a_{43} = 1$$

```
template <typename _T> inline void imu_tk::quatIntegrationStepRK4(
                const Eigen::Matrix< _T, 4, 1> &quat,
                const Eigen::Matrix< _T, 3, 1> &omega0,
                const Eigen::Matrix< _T, 3, 1> &omega1,
                const _T &dt, Eigen::Matrix< _T, 4, 1> &quat_res )
{
  Eigen::Matrix< _T, 3, 1> omega01 = _T(0.5)*( omega0 + omega1 );
  Eigen::Matrix< _T, 4, 1> k1, k2, k3, k4, tmp_q;
  Eigen::Matrix< _T, 4, 4> omega_skew;

  // First Runge-Kutta coefficient
  computeOmegaSkew( omega0, omega_skew );
  k1 = _T(0.5)*omega_skew*quat;
  // Second Runge-Kutta coefficient
  tmp_q = quat + _T(0.5)*dt*k1;
  computeOmegaSkew( omega01, omega_skew );
  k2 = _T(0.5)*omega_skew*tmp_q;
  // Third Runge-Kutta coefficient (same omega skew as second coeff.)
  tmp_q = quat + _T(0.5)*dt*k2;
  k3 = _T(0.5)*omega_skew*tmp_q;
  // Forth Runge-Kutta coefficient
  tmp_q = quat + dt*k3;
  computeOmegaSkew( omega1, omega_skew );
  k4 = _T(0.5)*omega_skew*tmp_q;
  _T mult1 = _T(1.0)/_T(6.0), mult2 = _T(1.0)/_T(3.0);
  quat_res = quat + dt*(mult1*k1 + mult2*k2 + mult2*k3 + mult1*k4);
  normalizeQuaternion(quat_res);
}
```

3. imu_tk

参考： https://www.cnblogs.com/feifanrensheng/p/10439057.html
https://zhuanlan.zhihu.com/p/22319718?refer=zimmon

Noise大家通常使用Allan方差进行估计可以得到较为可信的结果，这里不赘述了。内参数标定比较方便的一个工具就是imu_tk。

3.1 理论分析

首先分步介绍算法流程：

1. 读入数据，将时间单位转化为秒
2. 设置初始参数和标定算法的控制参数
3. 开始标定
3.1 标定加速度计
首先调用initiInterval函数，返回前50s(默认是30s)的数据的index

计算初始的init_static_interval的方差，定义为norm_th
For循环：th_mult=2:10
{利用大小为101的滑动窗口搜索静止区间：如果该滑动窗口内的加速度计读数的方差小于th_mult*norm_th，则认为是静止区间
提取出静止区间内的加速度计读数。如果某个区间的大小小于初始设置的interval_n_samples_（默认是100）则去除该静止区间；
注意，如果初始参数中acc_use_means_为true，则在静止区间内只取所有读数的平均值作为static_sample，且其时间戳为静止区间的时间戳的中值。否则保存所有的静止区间内的sample。如果提取出的静止区间个数小于初始设置的min_num_intervals_（默认是12），则认为采集的数据不足以标定imu，则程序退出。

构造目标函数：g-unbiasnorm(acc_samples)，其中前者为初始设置的重力加速度值，后者为去除bias以后的加速度计读数的norm。因为如果imu静止，则其加速度计的读数的模应当等于重力加速度的值。
利用ceres最小化目标函数得到加速度计的九个参数。并利用标定得到的参数将加速度计的raw_data进行修正。
}
th_mult在2~10时最小的估计误差对应的参数为最准的加速度计标定参数，同时保存该参数对应的static_interval。

3.2 标定陀螺仪

根据加速度计的标定结果，提取static_sample。
根据初始的50s的陀螺仪读数，估计陀螺仪的bias。
利用上步得到的bias矫正陀螺仪读数。
根据提取的static_interval找到运动区间的start_index和end_index。
构造目标函数：integrate_R’*g_start-g_end。其中g_start和g_end均是归一化后的向量。因为在imu运动区间内，两帧加速度计的读数之间应当是两帧间imu的旋转，也就是imu的陀螺仪积分后得到的结果。

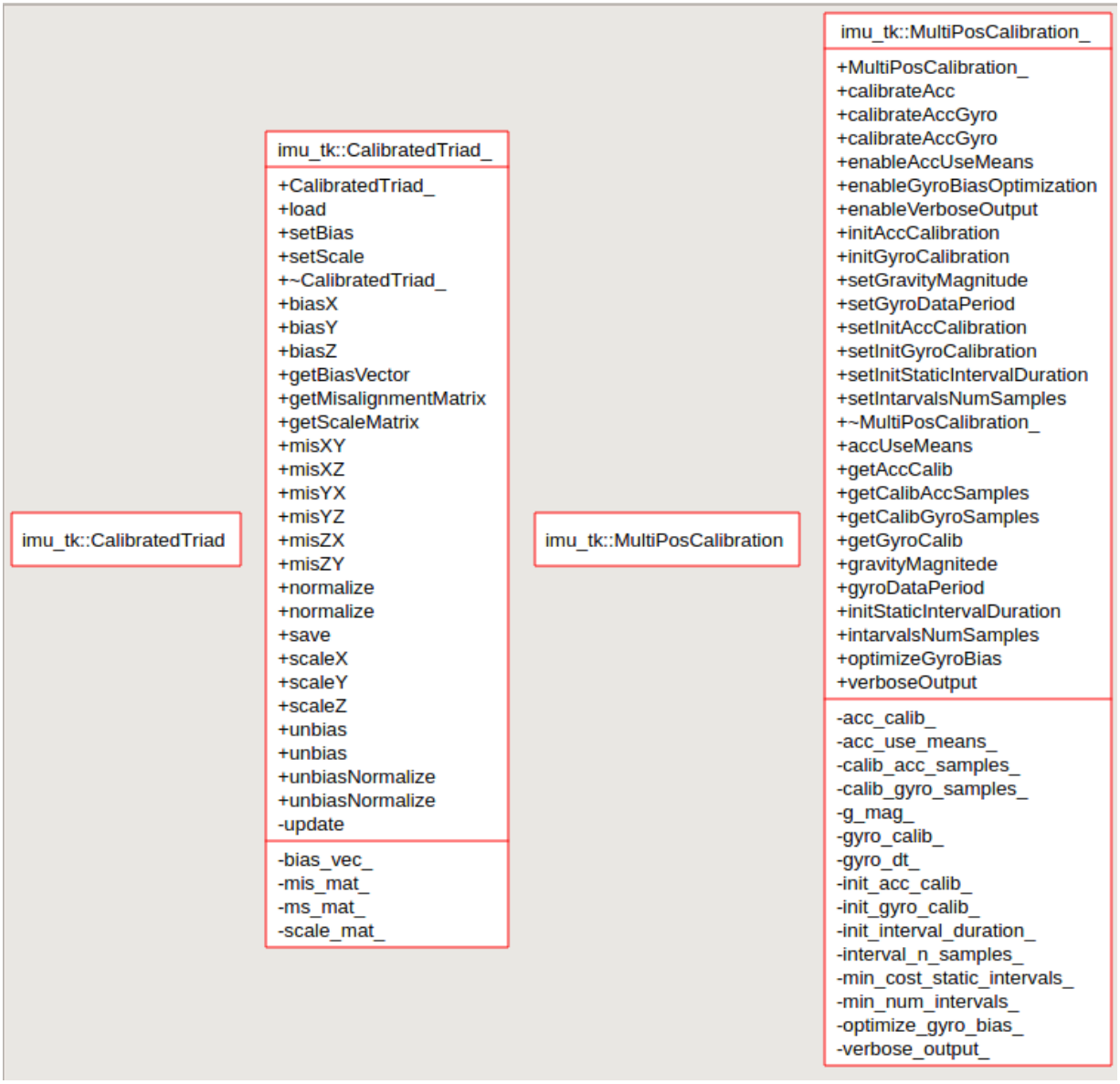
利用ceres最小化目标函数得到陀螺仪的十二个参数。注意，如果初始optimize_gyro_bias_为true，则在矫正陀螺仪读数后仍然需要标定bias参数，否则返回初始读数估计得到的bias。gyro_dt_如果为-1，则利用两帧陀螺仪的timestamp进行积分，否则利用gryo_dt给定的时间间隔进行积分。

注意事项：

1. 标定时，首先需要将imu静止一段时间，根据程序可知，至少需要静止50s以上。
2. 由于程序中检测静止区间的滑动窗口大小为101，所以每次静止时间需要超过100帧数据
3. 由于程序中检测静止区间时，需要至少end_index开始的滑动窗口内的方差大于2倍的静止方差，所以每两次静止区间之间的运动时间不能太短，且最好是有明显的加速或减速运动。最好运动时间超过100帧。
4. 由于程序中需要检测到的静止区间数大于12，且论文中提到静止区间为30+~40+次时，精度较好。所以需要有大概30多次的静止区间。
5. 静止区间内尽量保证imu是静止不动的。初始的1分钟中内尤其要保持imu静止，以得到较好的norm_th的估计和gyro_bias的估计。

3.2 代码说明

主要是数据集中提取静态的数据样本集， 先标定加速度，再依据静态数据的转换，利用四阶龙格库塔积分计算陀螺积分得到旋转矩阵，加速度在旋转矩阵作用下比对。




```
vector< TriadData > acc_data, gyro_data;

MultiPosCalibration mp_calib;
mp_calib.calibrateAccGyro(acc_data, gyro_data );
```

```
calibrateAcc( acc_samples )
1. 采集开始的静止区间
DataInterval::initialInterval
2. 方差
Eigen::Matrix<_T, 3, 1> acc_variance = dataVariance
_T norm_th = acc_variance.norm();

3. 利用ceres标定9个参数：
    std::vector< double > acc_calib_params(9);

    staticIntervalsDetector ( acc_samples, th_mult*norm_th, static_intervals );
    extractIntervalsSamples ( acc_samples, static_intervals,
                              static_samples, extracted_intervals,
                              interval_n_samples_, acc_use_means_ );

  mp_calib.setGravityMagnitude(9.81744);

MultiPosAccResidual
MultiPosGyroResidual
```

MultiPosAccResidual

```
/** @brief This object contains the calibration parameters (misalignment, scale factors, ...)
 *         of a generic orthogonal sensor triad (accelerometers, gyroscopes, etc.)
 *
 * Triad model:
 *
 * -Misalignment matrix:
 *
 * general case:
 *
 *     [    1     -mis_yz   mis_zy  ]
 * T = [  mis_xz     1     -mis_zx  ]
 *     [ -mis_xy   mis_yx     1     ]
 *
 * "body" frame spacial case:
 *
 *     [  1     -mis_yz   mis_zy  ]
 * T = [  0        1     -mis_zx  ]
 *     [  0        0        1     ]
 *
 * Scale matrix:
 *
 *     [  s_x      0        0  ]
 * K = [   0      s_y       0  ]
 *     [   0       0       s_z ]
 *
 * Bias vector:
 *
 *     [ b_x ]
 * B = [ b_y ]
 *     [ b_z ]
 *
 * Given a raw sensor reading X (e.g., the acceleration ), the calibrated "unbiased" reading X' is
obtained
 *
 * X' = T*K*(X - B)
 *
 * with B the bias (variable) + offset (constant, possibbly 0), or, equivalently:
 *
 * X' = T*K*X - B'
 *
 * with B' = T*K*B
 *
 * Without knowing the value of the bias (and with offset == 0), the calibrated reading X'' is
simply:
 *
 * X'' = T*K*X
*/
```

4. GyroALLAN 艾伦方差法标定陀螺仪

```
clc;    %清空命令行窗口
clear;  %清空工作区

data = dlmread('data.dat');          %从文本中读取数据，单位：deg/s，速率：100Hz
data = data(1:720000, 3:5)*3600;     %截取两个小时的数据，把 deg/s 转为 deg/h
[A, B] = allan(data, 100, 100);      %求Allan标准差，用100个点来描述

loglog(A, B, 'o');                   %画双对数坐标图
xlabel('time:sec');                  %添加x轴标签
ylabel('Sigma:deg/h');               %添加y轴标签
```

```
legend('X axis','Y axis','Z axis'); %添加标注
grid on;                            %添加网格线
hold on;                            %使图像不被覆盖

C(1, :) = nihe(sqrt(A'), B(:, 1)', 2)';   %拟合
C(2, :) = nihe(sqrt(A'), B(:, 2)', 2)';
C(3, :) = nihe(sqrt(A'), B(:, 3)', 2)';

Q = abs(C(:, 1)) / sqrt(3);         %量化噪声，单位：arcsec
N = abs(C(:, 2)) / 60;              %角度随机游走，单位：deg/h^0.5
Bs = abs(C(:, 3)) / 0.6643;         %零偏不稳定性，单位：deg/h
K = abs(C(:, 4)) * sqrt(3) * 60;    %角速率游走，单位：deg/h/h^0.5
R = abs(C(:, 5)) * sqrt(2) * 3600;  %速率斜坡，单位：deg/h/h

fprintf('量化噪声      X轴：%f Y轴：%f Z轴：%f  单位：arcsec\n', Q(1), Q(2), Q(3));
fprintf('角度随机游走  X轴：%f Y轴：%f Z轴：%f  单位：deg/h^0.5\n', N(1), N(2), N(3));
fprintf('零偏不稳定性  X轴：%f Y轴：%f Z轴：%f  单位：deg/h\n', Bs(1), Bs(2), Bs(3));
fprintf('角速率游走    X轴：%f Y轴：%f Z轴：%f  单位：deg/h/h^0.5\n', K(1), K(2), K(3));
fprintf('速率斜坡      X轴：%f Y轴：%f Z轴：%f  单位：deg/h/h\n', R(1), R(2), R(3));

D(:, 1) = C(1, 1)*sqrt(A).^(-2) + C(1, 2)*sqrt(A).^(-1) + C(1, 3)*sqrt(A).^(0) + C(1,
4)*sqrt(A).^(1) + C(1, 5)*sqrt(A).^(2);    %生成拟合函数
D(:, 2) = C(2, 1)*sqrt(A).^(-2) + C(2, 2)*sqrt(A).^(-1) + C(2, 3)*sqrt(A).^(0) + C(2,
4)*sqrt(A).^(1) + C(2, 5)*sqrt(A).^(2);
D(:, 3) = C(3, 1)*sqrt(A).^(-2) + C(3, 2)*sqrt(A).^(-1) + C(3, 3)*sqrt(A).^(0) + C(3,
4)*sqrt(A).^(1) + C(3, 5)*sqrt(A).^(2);

loglog(A, D);   %画双对数坐标图
```

```
function [T,sigma] = allan(omega,fs,pts)
[N,M] = size(omega);              % figure out how big the output data set is
n = 2.^(0:floor(log2(N/2)))';     % determine largest bin size
maxN = n(end);
endLogInc = log10(maxN);
m = unique(ceil(logspace(0,endLogInc,pts)))';    %logspace生成从10的0次方到10的endLogInc次方之间按对数等分的pts个元素， ceil向上取整， unique数组中的唯一值
t0 = 1/fs;                                       % t0 = sample interval
T = m*t0;                                        % T = length of time for each cluster
theta = cumsum(omega)/fs;       % 累加和  /fs 补T中的fs; 缩小整体累积位数
sigma2 = zeros(length(T),M);    % array of dimensions (cluster periods) X (#variables)
for i=1:length(m)               % loop over the various cluster sizes
    for k=1:N-2*m(i)            % implements the summation in the AV equation
        sigma2(i,:) = sigma2(i,:) + (theta(k+2*m(i),:) - 2*theta(k+m(i),:) + theta(k,:)).^2;
    end
end
sigma2 = sigma2./repmat((2*T.^2.*(N-2*m)),1,M);%重复数组副本 创建一个所有元素的值均为 xx 的 1×M 矩阵
sigma = sqrt(sigma2);
```

```
function C=nihe(tau,sig,M)
X=tau';Y=sig';
B=zeros(1,2*M+1);
F=zeros(length(X),2*M+1);

for i=1:2*M+1
    kk=i-M-1;
    F(:,i)=X.^kk;
end

A=F'*F;
B=F'*Y;
C=A\B;
```

5. 姿态解算

5.1 互补滤波

以陀螺仪积分计算角度为测量值，加速度计为观测值； 依据向量叉乘来描述两向量的角度偏差，假定重力加速度静态为世界坐标系下（0,0,1），依据上一时刻的姿态，旋转转换到集体坐标系V， 将基体坐标系V与当前加速度计的测量值acc作叉乘得到偏差； 该偏差补偿陀螺仪的测量值更新陀螺测量值； 依据龙格库塔法计算新的姿态值；

```
bool ImuNode::UpdateAHRSByCrossDletaFilter(const Eigen::Vector3d &acc, const Eigen::Vector3d &gyro,
const double data_time_sec)
{
    LOG(INFO)<<"acc:"<<acc[0]<<"  "<< acc[1]<<"  "<<acc[2];
    LOG(INFO)<<"gyro:"<<gyro[0]<<"  "<< gyro[1]<<"  "<<gyro[2];
    Eigen::Vector3d acc_norm = acc.normalized();
    LOG(INFO)<<"acc_norm:"<<acc_norm[0]<<"  "<< acc_norm[1]<<"  "<<acc_norm[2];
    //calc direction of gravity after rotation : last_R * [0 0 1]'
    Eigen::Vector3d last_base_direction;
    last_base_direction[0] = 2 * (state_quaternion_[1] * state_quaternion_[3] - state_quaternion_[0]
* state_quaternion_[2]);
    last_base_direction[1] = 2 * (state_quaternion_[0] * state_quaternion_[1] + state_quaternion_[2]
* state_quaternion_[3]);
    last_base_direction[2] = state_quaternion_[0] * state_quaternion_[0] - state_quaternion_[1] *
state_quaternion_[1] - state_quaternion_[2] * state_quaternion_[2] + state_quaternion_[3] *
state_quaternion_[3];
    /* //QuaternionToScaledRotation
        R(0, 0) = aa + bb - cc - dd; R(0, 1) = T(2) * (bc - ad);  R(0, 2) = T(2) * (ac + bd);
        R(1, 0) = T(2) * (ad + bc);  R(1, 1) = aa - bb + cc - dd; R(1, 2) = T(2) * (cd - ab);
        R(2, 0) = T(2) * (bd - ac);  R(2, 1) = T(2) * (ab + cd);  R(2, 2) = aa - bb - cc + dd;
    */

    // last_base_direction[0] = 2 * (state_quaternion_[0] * state_quaternion_[2] +
state_quaternion_[1] * state_quaternion_[3]);
    // last_base_direction[1] = 2 * (state_quaternion_[2] * state_quaternion_[3] -
state_quaternion_[0] * state_quaternion_[1]);
    // last_base_direction[2] = state_quaternion_[0] * state_quaternion_[0] - state_quaternion_[1] *
state_quaternion_[1] - state_quaternion_[2] * state_quaternion_[2] + state_quaternion_[3] *
state_quaternion_[3];
    LOG(INFO)<<"last_base_direction:"<<last_base_direction[0]<<"  "<< last_base_direction[1]<<"  "
<<last_base_direction[2];
    // calc erro by cross product
    Eigen::Vector3d cross_error;
    cross_error[0] = acc_norm[1] * last_base_direction[2] - acc_norm[2] * last_base_direction[1];
    cross_error[1] = acc_norm[2] * last_base_direction[0] - acc_norm[0] * last_base_direction[2];
    cross_error[2] = acc_norm[0] * last_base_direction[1] - acc_norm[1] * last_base_direction[0];
    LOG(INFO)<<"cross_error:"<<cross_error[0]<<"  "<< cross_error[1]<<"  "<<cross_error[2];
    // integral error with scale
    integral_cross_error_[0] += kKi * cross_error[0];
    integral_cross_error_[1] += kKi * cross_error[1];
    integral_cross_error_[2] += kKi * cross_error[2];

    //adjusted gyro measurements
    Eigen::Vector3d gyro_adjust;
    gyro_adjust[0] = gyro[0] + kKp * cross_error[0] + integral_cross_error_[0];
    gyro_adjust[1] = gyro[1] + kKp * cross_error[1] + integral_cross_error_[1];
    gyro_adjust[2] = gyro[2] + kKp * cross_error[2] + integral_cross_error_[2];
    LOG(INFO)<<"gyro_adjust:"<<gyro_adjust[0]<<"  "<< gyro_adjust[1]<<"  "<<gyro_adjust[2];

    //update quaternion By RK1
    double half_dT = (data_time_sec - last_time_sec_) / 2.0;
    Eigen::Vector4d updated_state_quaternion_;
    updated_state_quaternion_[0] = state_quaternion_[0] - (state_quaternion_[1] * gyro_adjust[0] +
state_quaternion_[2] * gyro_adjust[1] +
                                                           state_quaternion_[3] * gyro_adjust[2]) *
                                                              half_dT;
```

```
    updated_state_quaternion_[1] = state_quaternion_[1] + (state_quaternion_[0] * gyro_adjust[0] +
state_quaternion_[2] * gyro_adjust[0] -
                                                           state_quaternion_[3] * gyro_adjust[1]) *
                                                              half_dT;
    updated_state_quaternion_[2] = state_quaternion_[2] + (state_quaternion_[0] * gyro_adjust[1] -
state_quaternion_[1] * gyro_adjust[2] +
                                                           state_quaternion_[3] * gyro_adjust[0]) *
                                                              half_dT;
    updated_state_quaternion_[3] = state_quaternion_[3] + (state_quaternion_[0] * gyro_adjust[2] +
state_quaternion_[1] * gyro_adjust[1] -
                                                           state_quaternion_[2] * gyro_adjust[0]) *
                                                              half_dT;
    //normalized the updated state quaternion
    state_quaternion_ = updated_state_quaternion_.normalized();
    last_time_sec_ = data_time_sec;
    LOG(INFO)<<"state_quaternion_:"<<state_quaternion_[0]<<"  "<< state_quaternion_[1]<<"  "
<<state_quaternion_[2]<<std::endl;
    return true;
}
```

参考文献

1 [Tedaldi D, Pretto A, Menegatti E. A robust and easy to implement method for IMU calibration without external equipments[J]. 2014:3042-3049.
2 https://github.com/ethz-asl/kalibr
3 Allan Variance: Noise Analysis for Gyroscopes
4 [厉宽宽, 陈允芳, 程敏,等. MEMS-IMU随机误差的Allan方差分析[J]. 全球定位系统, 2016,41(6):102-106.
5 https://bitbucket.org/alberto_pretto/imu_tk

https://bitbucket.org/alberto_pretto/imu_tk

https://github.com/gaowenliang/imu_utils

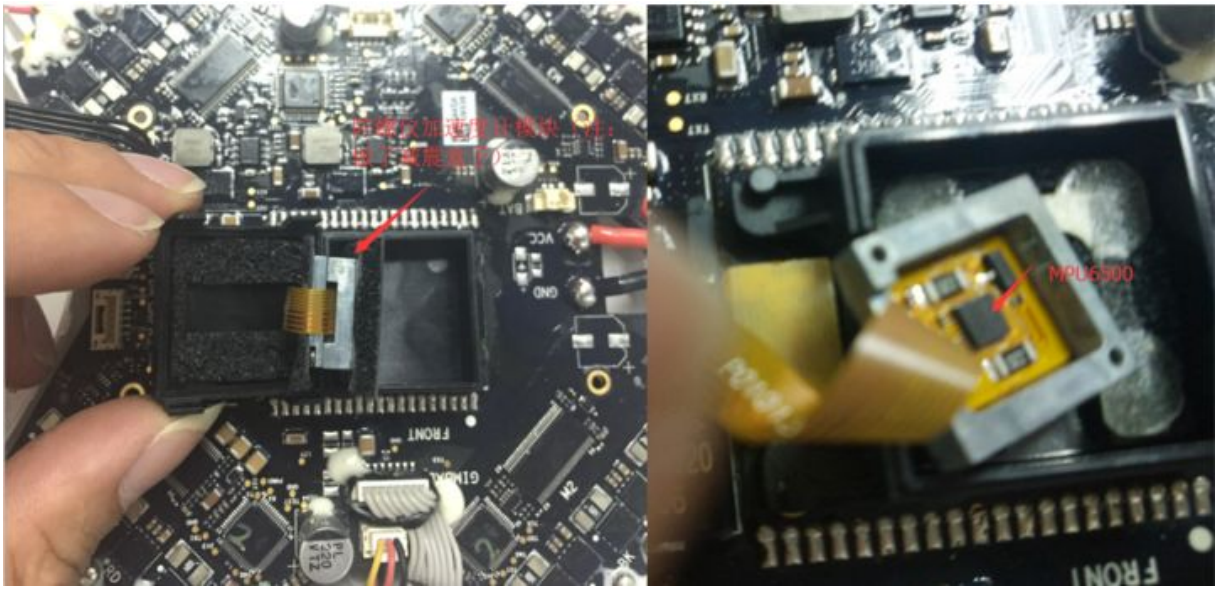
```
git clone https://github.com/robcn/IMU_Calibration.git
git clone https://github.com/robcn/imu_tk.git
git clone https://github.com/robcn/GyroAllan.git
git clone https://github.com/robcn/IMUCalibration-Gesture.git

git clone https://github.com/robcn/ai-imu-dr.git
```

从零开始的 IMU 状态模型推导

陀螺仪的数据处理

目前解决这个问题的方法有两种：一种是**传感器数据保持恒温输出**，如下图的DJI的拆机图，IMU模块有个大电阻进行加热，使得传感器的工作温度在一恒定值。


MPU6500
FRONT