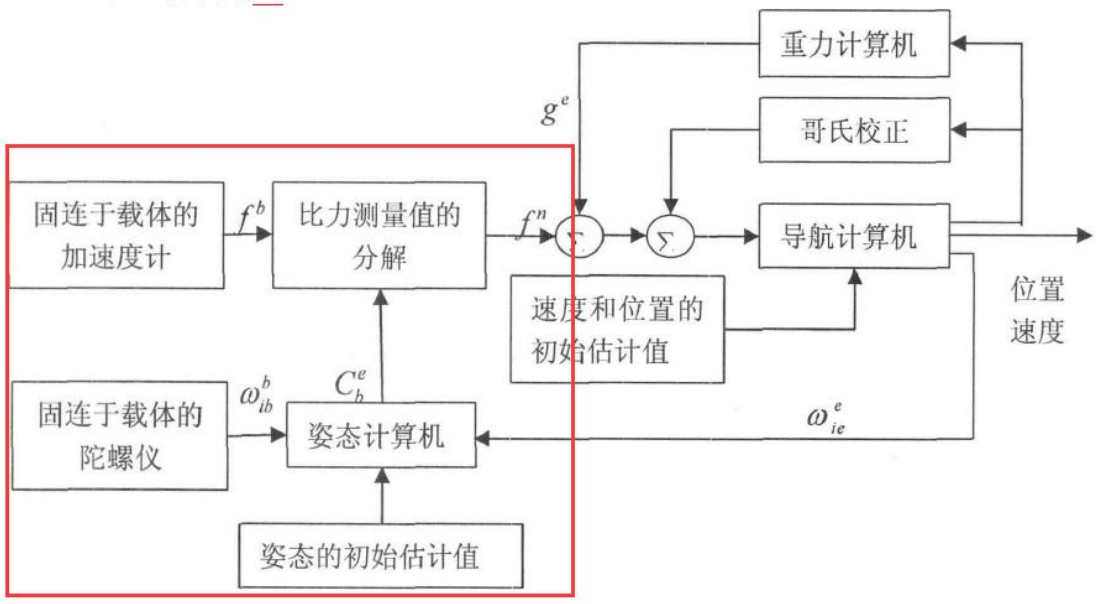
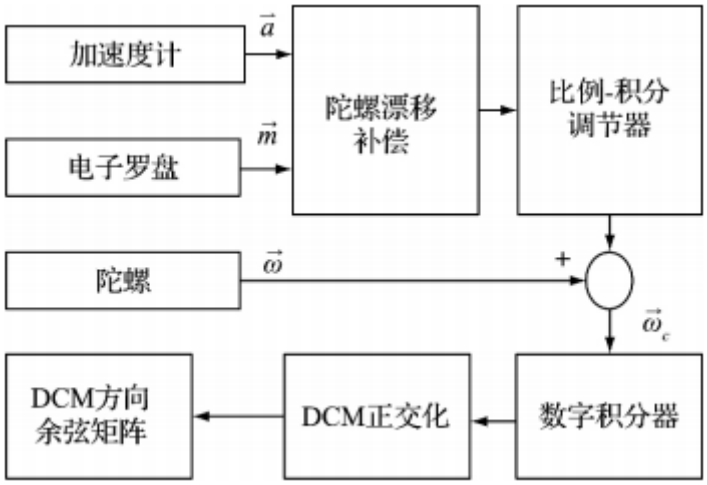
Pixhawk_源码分析

本篇主要是搞姿态解算这一部分。





姿态表示的方法有很多种，比如欧拉角、四元数、DCM，各有的各的优势，比较常用的就是四元数，方便计算。下面的介绍的方法也是基于四元数的。

国内的科研水平太 low 了，堂堂清*大学发表的文章竟然直接翻译的国外的论文，但是

其精明之处在于参考文献不提及被翻译的那篇论文；妙，实在是妙不可言啊。可是、难道就没有 peer review 么~~~~~再或者参与 peer review 的正是所谓的“专家”么~~~

姿态解算方法的比较：

算法	优点	缺点	PS
欧拉角法（3 参数）	1、通过欧拉角微分方程直接解算出 pitch、roll、yaw 2、概念直观，易于理解 3、解算结果永远是正交的，无需再次正交化处理	1、方程中有三角函数的运算，接超越函数有一定的困难 2、当俯仰角接近 90°时出现退化现象	1、适应于水平姿态角变化不大的情况 2、不适用于全姿态运载体
方向余弦法（9 参数）	1、对姿态矩阵微分方程的求解，避免了欧拉角法中出现的退化现象 2、可以全姿态运行	1、参数量过多，计算量大，实时性不好	很少在工程中使用
四元数法（4 参数）	1、直接求解四元数微分方程 2、只需要求解四个参数，计算量小 3、算法简单，易于操作	漂移比较严重	可以在实现过程中修正漂移，应用比较广泛

开篇

下面是德国人写的一份 AHRS 姿态解算源码，很有参考价值（理论基础是 DCM IMU:Theory）。

```
/*
* AHRS
* Copyright 2010 S.O.H. Madgwick
*/
// AHRS.c
// S.O.H. Madgwick
// 25th August 2010
// Quaternion implementation of the 'DCM filter' [Mayhony et al]. Incorporates the magnetic distortion
// compensation algorithms from my filter [Madgwick] which eliminates the need for a reference
// direction of flux (bx bz) to be predefined and limits the effect of magnetic distortions to yaw
// axis only.
// User must define 'halfT' as the (sample period / 2), and the filter gains 'Kp' and 'Ki'.
// Global variables 'q0', 'q1', 'q2', 'q3' are the quaternion elements representing the estimated
// orientation. See my report for an overview of the use of quaternions in this application.
// User must call 'AHRSupdate()' every sample period and parse calibrated gyroscope ('gx', 'gy', 'gz'),
// accelerometer ('ax', 'ay', 'ay') and magnetometer ('mx', 'my', 'mz') data. Gyroscope units are
// radians/second, accelerometer and magnetometer units are irrelevant as the vector is
```

normalised.

```
#include "stm32f10x.h"
#include "AHRS.h"
#include "Positioning.h"
#include <math.h>
#include <stdio.h>
/* Private define ------------------------------------------------------------*/
#define Kp 2.0f                         // proportional gain governs rate of convergence to accelerometer/magnetometer
#define Ki 0.005f                       // integral gain governs rate of convergence of gyroscope biases
#define halfT 0.0025f                   // half the sample period : 0.005s/2=0.0025s 用于求解四元数微分方程时计算角增量
#define ACCEL_1G 1000                   //the acceleration of gravity is: 1000 mg
/* Private variables ---------------------------------------------------------*/
static float q0 = 1, q1 = 0, q2 = 0, q3 = 0;    // quaternion elements representing the estimated orientation
static float exInt = 0, eyInt = 0, ezInt = 0;   // scaled integral error
/* Public variables ----------------------------------------------------------*/
EulerAngle_Type EulerAngle;             //unit: radian
u8 InitEulerAngle_Finished = 0;
float Magnetoresistor_mGauss_X = 0, Magnetoresistor_mGauss_Y = 0, Magnetoresistor_mGauss_Z = 0;//unit: milli-Gauss
float Accelerate_mg_X, Accelerate_mg_Y, Accelerate_mg_Z;//unit: mg
float AngularRate_dps_X, AngularRate_dps_Y, AngularRate_dps_Z;//unit: dps: degree per second
int16_t Magnetoresistor_X, Magnetoresistor_Y, Magnetoresistor_Z;
uint16_t Accelerate_X = 0, Accelerate_Y = 0, Accelerate_Z = 0;
uint16_t AngularRate_X = 0, AngularRate_Y = 0, AngularRate_Z = 0;
u8 Quaternion_Calibration_ok = 0;
/* Private macro -------------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/*******************************************************************************
* Function Name  : AHRSupdate
* Description    : None
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
//陀螺仪、加速度计、磁力计数据融合
void AHRSupdate(float gx, float gy, float gz, float ax, float ay, float az, float mx, float my, float mz) {
        float norm;
```

```
float hx, hy, hz, bx, bz;
float vx, vy, vz, wx, wy, wz; //v*当前姿态计算得来的重力在三轴上的分量
float ex, ey, ez;

// auxiliary variables to reduce number of repeated operations
float q0q0 = q0*q0;
float q0q1 = q0*q1;
float q0q2 = q0*q2;
float q0q3 = q0*q3;
float q1q1 = q1*q1;
float q1q2 = q1*q2;
float q1q3 = q1*q3;
float q2q2 = q2*q2;
float q2q3 = q2*q3;
float q3q3 = q3*q3;

// normalise the measurements
norm = sqrt(ax*ax + ay*ay + az*az);
ax = ax / norm;
ay = ay / norm;
az = az / norm;
norm = sqrt(mx*mx + my*my + mz*mz);
mx = mx / norm;
my = my / norm;
mz = mz / norm;

// compute reference direction of magnetic field
hx = 2*mx*(0.5 - q2q2 - q3q3) + 2*my*(q1q2 - q0q3) + 2*mz*(q1q3 + q0q2);
hy = 2*mx*(q1q2 + q0q3) + 2*my*(0.5 - q1q1 - q3q3) + 2*mz*(q2q3 - q0q1);
hz = 2*mx*(q1q3 - q0q2) + 2*my*(q2q3 + q0q1) + 2*mz*(0.5 - q1q1 - q2q2);
bx = sqrt((hx*hx) + (hy*hy));
bz = hz;

// estimated direction of gravity and magnetic field (v and w)
//参考坐标 n 系转化到载体坐标 b 系的用四元数表示的方向余弦矩阵第三列即是。
//处理后的重力分量
vx = 2*(q1q3 - q0q2);
vy = 2*(q0q1 + q2q3);
vz = q0q0 - q1q1 - q2q2 + q3q3;
//处理后的 mag
wx = 2*bx*(0.5 - q2q2 - q3q3) + 2*bz*(q1q3 - q0q2);
wy = 2*bx*(q1q2 - q0q3) + 2*bz*(q0q1 + q2q3);
wz = 2*bx*(q0q2 + q1q3) + 2*bz*(0.5 - q1q1 - q2q2);
```

// error is sum of cross product between reference direction of fields and direction measured by sensors 体现在加速计补偿和磁力计补偿，因为仅仅依靠加速计补偿没法修正 Z 轴的变差，所以还需要通过磁力计来修正 Z 轴。（公式 28）。《四元数解算姿态完全解析及资料汇总》的作者把这部分理解错了，不是什么叉积的差，而叉积的计算就是这样的。计算方法是公式 10。

```
ex = (ay*vz - az*vy) + (my*wz - mz*wy);
ey = (az*vx - ax*vz) + (mz*wx - mx*wz);
ez = (ax*vy - ay*vx) + (mx*wy - my*wx);

// integral error scaled integral gain
exInt = exInt + ex*Ki* (1.0f / sampleFreq);
eyInt = eyInt + ey*Ki* (1.0f / sampleFreq);
ezInt = ezInt + ez*Ki* (1.0f / sampleFreq);
// adjusted gyroscope measurements
```

//将误差 PI 后补偿到陀螺仪，即补偿零点漂移。通过调节 Kp、Ki 两个参数，可以控制加速度计修正陀螺仪积分姿态的速度。（公式 16 和公式 29）

```
gx = gx + Kp*ex + exInt;
gy = gy + Kp*ey + eyInt;
gz = gz + Kp*ez + ezInt;

// integrate quaternion rate and normalize
```

//一阶龙格库塔法更新四元数

```
q0 = q0 + (-q1*gx - q2*gy - q3*gz)*halfT;
q1 = q1 + (q0*gx + q2*gz - q3*gy)*halfT;
q2 = q2 + (q0*gy - q1*gz + q3*gx)*halfT;
q3 = q3 + (q0*gz + q1*gy - q2*gx)*halfT;

// normalise quaternion
norm = sqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
q0 = q0 / norm;
q1 = q1 / norm;
q2 = q2 / norm;
q3 = q3 / norm;
}
```

1、 陀螺仪和加速计（特性分析）

1）陀螺仪

Gyro sensitivity、 operating range、offset、drift、calibrationand saturation must be taken into account in the implementation of DCM。

灵敏度

测量角速度，具有高动态特性，它是一个间接测量角度的器件其中一个关键参数就是 gyro sensitivity（其单位是 millivolts per degree per second，把转速转换到电压值），测量范围越小气灵敏度越好。也就是说测量的是角度的导数，即角速度，要将角速度对时间积分才能得到角度。陀螺仪就是内部有一个陀螺，它的轴由于陀螺效应始终与初始方向平行，这样就可以通过与初始方向的偏差计算出旋转方向和角度。

偏移

偏移就是在陀螺没有转动的时候却又输出，这个输出量的大小和供电电压以及温度有关，该偏移可以在陀螺仪上电时通过一小段时间的测量来修正。

漂移

它是由于在时间的积累下偏移和噪声相互影响的结果，例如有一个偏置（offset）0.1dps 加在上面，于是测量出来是 0.1dps，积分一秒之后，得到的角度是 0.1 度，1 分钟之后是 6 度，还能忍受，一小时之后是 360 度，转了一圈，也就是说，陀螺仪在短时间内有很大的参考价值。

白噪声

电信号的测量中，一定会带有白噪声，陀螺仪数据的测量也不例外。所以获得的陀螺仪数据中也会带有白噪声，而且这种白噪声会随着积分而累加。

积分误差

对陀螺仪角速度的积分是离散的，长时间的积分会出现漂移的情况。所以要考虑积分误差的问题。

溢出

就是转速超过了其测量的最大转速范围。关于这个问题的解决办法，在 DCM IMU:Theory 中给出来三种解决办法，不写了。

2）加速度计

加速度计的低频特性好，可以测量低速的静态加速度。在无人机上就是对重力加速度 g 的测量和分析。 当把加速度计拿在手上随意转动时，看的是重力加速度在三个轴上的分量值。加速度计在自由落体时，其输出为 0。为什么会这样呢？这里涉及到加速度计的设计原理：加速度计测量加速度是通过比力来测量(比力方程)，而不是通过加速度。

加速度计仅仅测量的是重力加速度,而重力加速度与刚才所说的 R 坐标系（Earth Frame）是固连的,通过这种关系,可以得到加速度计所在平面与地面的角度关系。

加速度计仅仅测量的是重力加速度，3 轴加速度计输出重力加速度在加速度计所在机体坐标系 3 个轴上的分量大小。重力加速度的方向和大小是固定的。通过这种关系，可以得到加速度计所在平面与地面的角度关系。加速度计若是绕着重力加速度的轴转动，则测量值不会改变，也就是说加速度计无法感知这种水平旋转。

Accelerometers are used for roll-pitch drift correction because they have zero drift

有关陀螺仪和加速度计和关系，姿态解算融合的原理，再把下面这个比喻放到这里一遍。

机体好似一条船，姿态就是航向（船头的方位），重力是灯塔，陀螺（角速度积分）是舵手，加速度计是瞭望手。舵手负责估计和把稳航向，他相信自己，本来船向北开的，就一定会一直往北开，觉得转了 90 度弯，那就会往东开。当然如果舵手很牛逼，也许能估计很准确，维持很长时间。不过只信任舵手，肯定会迷路，所以一般都有瞭望手来观察误差。

瞭望手根据地图灯塔方位和船的当前航向，算出灯塔理论上应该在船的 X 方位。然而看到实际灯塔在船的 Y 方位，那肯定船的当前航向有偏差了，偏差就是 ERR=X-Y。舵手收到瞭望手给的 ERR 报告，觉得可靠，那就听个 90%ERR，觉得天气不好、地图误差大，那就听个 10%ERR，根据这个来纠正估算航向。

3）磁传感器

可以测量磁场，在没有其他磁场的情况下，仅仅测量的是地球的磁场，而地磁也是和 R 坐标系固连的，通过这种关系，可以得到平面 A 和地平面的关系。(平面 A：和磁场方向垂直的平面)，同样的，若是沿着磁场方向的轴旋转，测量值不会改变，无法感知这种旋转。

综合考虑

加速度计和磁传感器都是极易受外部干扰的传感器，都只能得到 2 维的角度关系，但是测量值随时间的变化相对较小，结合加速度计和磁传感器可以得到 3 维的角度关系。陀螺仪可以积分得到三维的角度关系，动态性能好，受外部干扰小，但测量值随时间变化比较大。

可以看出，它们优缺点互补，结合起来才能有好的效果。

4）关于数据融合

现在有了三个传感器，都能在一定程度上测量角度关系，但是究竟相信谁？根据刚才的分析，应该是在短时间内更加相信陀螺仪，隔三差五的问问加速度计和磁传感器，角度飘了多少了？有一点必须非常明确，陀螺仪才是主角，加速度计和磁传感器仅仅是跑龙套的。其实加速度计无法对航向角进行修正，修正航向角需要磁力计。

2、 关于用加速计和磁力计求姿态

假设现在已经得到正确的姿态四元数 Q，那么可以利用四元数旋转将参考坐标系和体坐标系下的向量互相转换，将 Eh 和 Eg 转换到体坐标系下(BEh=Q×Eh×Q* ，BEg=Q×Eg×Q*，BEh、BEg 是参考坐标系的，Eh、Eg 由 Q 旋转到体坐标系下)。如果这个世界是理想的，美好的，那么 Bg=Beg、Bh=BEh 也就是｛Q×Eh×Q* - Bh = 0}、｛Q×Eg×Q* - Bg = 0}这两个方程成立，联立这两个方程就可以解得姿态四元数 Q，不过很遗憾,现实是残酷的,由于各种误差的存在，这个方程组无解，因此我们只能找到最优解，找最优解的最小方差问题有许多方法可以采用。如[梯度下降][高斯牛顿]。

加速度计和磁传感器经过高斯牛顿迭代得到姿态误差速率，陀螺仪直接通过四元数微分方程得到姿态四元数速率，两个加起来积分得到姿态四元数。

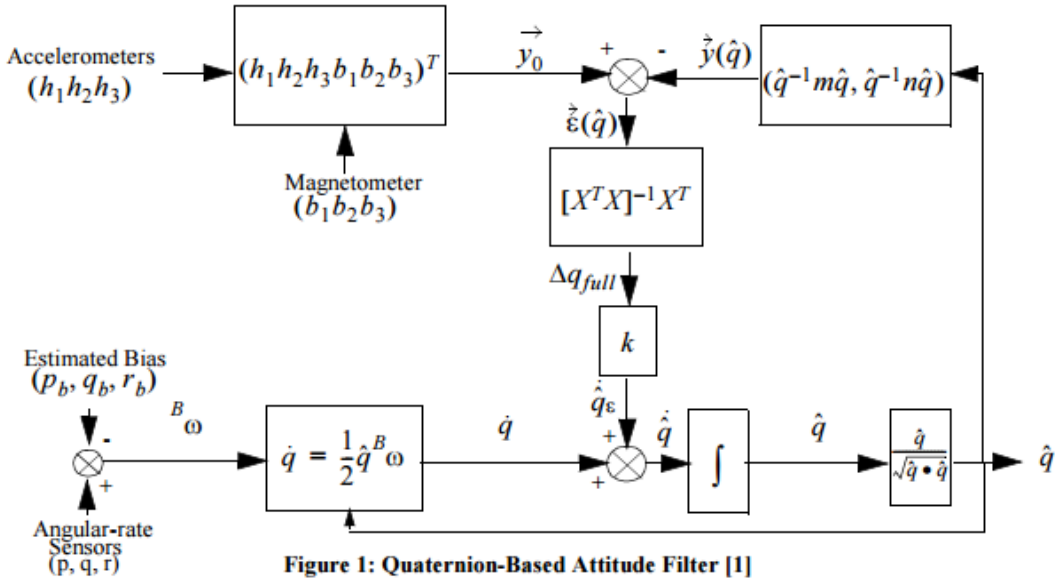
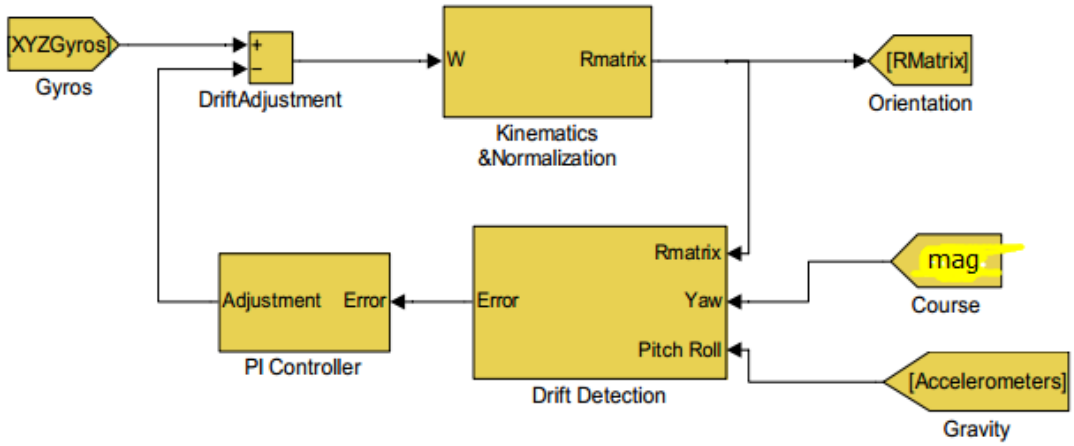


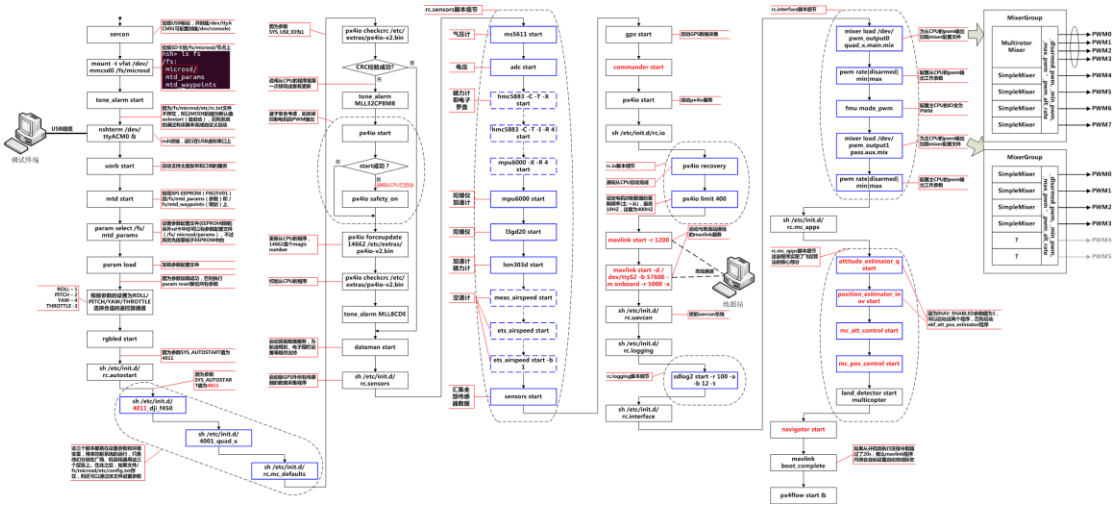
Figure 1: Quaternion-Based Attitude Filter [1]

代码中实现 gyro drift 补偿的理论框图，整个流程图如下。

Figure 1



通过 PI 反馈控制器修正 gyro 的漂移，以达到修正旋转向量的误差；把 PI 控制器的输出反馈到 gyro。Mahony 的算法也是基于这个流程。



3、 关于对 mahony 算法的理解（数据融合算法__开篇处的算法）

首先，东北天坐标系是 n 系（地理坐标系，参考坐标系），载体坐标系是 b 系，就是我们飞行器的坐标系。对于四元数法的姿态解算，需要求取的就是四元数的值；方向余弦矩阵（用于表示 n 系和 b 系的相对关系）中的元素本来应该是三角函数，这里由于使用四元数法，所以矩阵中的元素就成了四元数。所以姿态解算的任务就是求解由四元数构成的方向余弦矩阵 nCb（nCb 表示从 b 系到 n 的转换矩阵，同理，bCn 表示从 n 系到 b 的转换矩阵，它们的关系是转置）。

显然，上述矩阵是有误差存在的。对于一个确定的向量 n，用不同的坐标系表示时，他们所表示的大小和方向一定是相同的。但是由于这两个坐标系的转换矩阵存在误差，那么当一个向量经过这么一个有误差存在的旋转矩阵变换后，在另一个坐标系中肯定和理论值是有偏差的，我们通过这个偏差来修正这个旋转矩阵。这个旋转矩阵的元素是四元数，这就是说修正的就是四元数，于是乎姿态就这样被修正了，这才是**姿态解算的原理**。

姿态解算求的是四元数，是通过修正旋转矩阵中的四元数来达到姿态解算的目的，而不要以为通过加速度计和地磁计来修正姿态，加速度计和地磁计只是测量工具和载体，通过这两个器件表征旋转矩阵的误差存在，然后通过算法修正误差，修正四元数，修正姿态。

加速度计修正 pitch_roll

加速度计可以修正 pitch_roll，但是我们必须要考虑离心加速度（centrifugal acceleration），离心加速度就等于旋转率向量（即 gyro vector）和速度向量的叉积（没有原因，平均得来的并且还相当准确，速度可以用 GPS 获取）。我们假设飞机方向和 X 轴平行。

$$\mathbf{A}_{centrifugal} = \boldsymbol{\omega}_{gyro} \times \mathbf{V}$$

$$\mathbf{V} = \begin{bmatrix} velocity \\ 0 \\ 0 \end{bmatrix} \qquad \text{Eqn. 25}$$

在机体上测得的重力的为：

$$\mathbf{g}_{reference} = \mathbf{Accelerometer} + \boldsymbol{\omega}_{gyro} \times \mathbf{V}$$

$$\mathbf{Accelerometer} = \begin{bmatrix} Accelerometer_x \\ Accelerometer_y \\ Accelerometer_z \end{bmatrix} \qquad \text{Eqn. 26}$$

Pitch_roll 的旋转修正向量是由 DCM 的 Z 行与归一化以后的重力参考向量的叉积。

$$\mathbf{RollPitchCorrectionPlane} = \begin{bmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{bmatrix} \times \mathbf{g}_{reference} \qquad \text{Eqn. 27}$$

在 n 系中，加速度计输出为$[0,0,1]^T$，经过 bCn（用四元数表示的转换矩阵）转换之后到 b 系中的值为$[vx,vy,vz]^T$；在 b 系中，加速度计的测量值为$[ax,ay,az]^T$，现在$[vx,vy,vz]^T$和$[ax,ay,az]^T$均表示在 b 系中的竖直向下的向量，由此，我们来做向量积（叉积），得到误差$[ex,ey,ez]^T$，利用这个误差来修正 bCn 矩阵，于是四元数就在这样一个过程中被修正了。但是，由于加速度计无法感知 z 轴上的旋转运动，所以还需要用地磁计来进一步补偿。

介绍一下$[vx,vy,vz]^T$的计算过程：

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

重力加速度归一化：$g_n = [0,0,1]^T$

转换到 b 系之后的三轴分量为：$g_b = [vx,vy,vz]^T = C_n^b.g_n = C_n^b.[0,0,1]^T$

最后计算结果为：$g_b = [2(q_1q_3-q_0q_2),2(q_2q_3+q_0q_1),q_0^2-q_1^2-q_2^2+q_3^2]^T$；即用四元数表示的 DCM 的第三列元素。

然后就是$[ex,ey,ez]^T$的求取过程，这个主要就靠向量的叉积，例如两个向量 A 和 B，它们的叉积计算公式为：

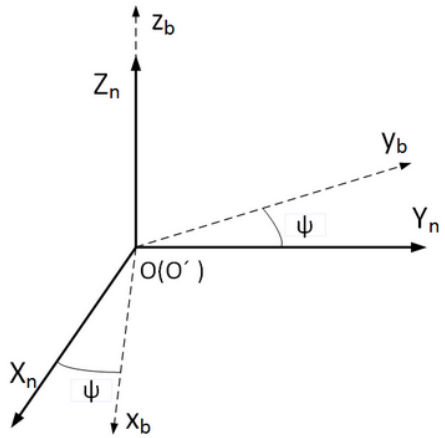
$$(\mathbf{A}\times\mathbf{B})_x = A_yB_z - A_zB_y$$

$$(\mathbf{A}\times\mathbf{B})_y = A_zB_x - A_xB_z \qquad \text{Eqn. 10}$$

$$(\mathbf{A}\times\mathbf{B})_z = A_xB_y - A_yB_x$$

加速度计在静止时测量的是重力加速度，是有大小和方向的；同理，地磁计同样测量的是地球磁场的大小和方向，只不过这个方向不再是竖直向下，而是与 x 轴（或者 y 轴）呈一个角度，与 z 轴呈一个角度。记作$[bx, by, bz]^T$，假设 x 轴对准北边，所以 by=0，即$[bx, 0, bz]^T$。倘若知道 bx 和 bz 的精确值，那么就可以采用和加速度计一样的修正方法来修正。只不过在加速度计中，在 n 系中的参考向量是$[0,0,1]^T$，变成了地磁计的$[bx, 0, bz]^T$。如果我们知道 bx 和 bz 的精确值，那么就可以摆脱掉加速度计的补偿，直接用地磁计和陀螺仪进行姿态解算，但是你看过谁只用陀螺仪和地磁计进行姿态解算吗？没有，因为没人会去测量当地的地磁场相对于东北天坐标的夹角，也就是 bx 和 bz。那么现在怎么办？

前面已经讲了，姿态解算就是求解旋转矩阵，这个矩阵的作用就是将 b 系和 n 正确的转化直到重合。现在我们假设 nCb 旋转矩阵是经过加速度计校正后的矩阵，当某个确定的向量（b 系中）经过这个矩阵旋转之后（到 n 系），这两个坐标系在 XOY 平面上重合（参考 DCM IMU:Theory 的 Drift cancellation 部分），只是在 z 轴旋转上会存在一个偏航角的误差。下图表示的是经过 nCb 旋转之后的 b 系和 n 系的相对关系。可以明显发现加速度计可以把 b 系通过四元数法从任意角度拉到与 n 系水平的位置上，此时只剩下一个偏航角误差。这也是为什么加速度计无法修正偏航的原因。



现在我们反过来从 b 系推往 n 系：设地磁计在 b 系中的输出为$[mx, my, mz]^T$，经过 nCb1 旋转之后得到$[hx, hy, hz]^T$（n 系）。在这个 XOY 平面上（n 系），$[bx, 0, bz]^T$的投影为 sqrt(bx*bx)，$[hx, hy, hz]^T$的投影为 sqrt((hx*hx) + (hy*hy))。显然，地磁计在 XOY 平面上（n 系）的向量的大小必定相同，所以有 bx= sqrt((hx*hx) + (hy*hy))。而对于 bz 的处理，我们不做变动，令 bz=hz 即可。经过这样处理之后的$[bx, 0, bz]^T$，经过 bCn1 旋转回转到 b 系中，得到$[wx, wy, wz]^T$，这个值再和 b 系中的地磁计输出$[mx, my, mz]^T$做向量积求误差，再次修正 bCn1（或者 nCb1），得到 bCn2（或者 nCb2）。这样就完成了一次地磁计的补偿。

将加速度计没能做到的 z 轴上的旋转修正，通过地磁计在 XOY 平面上的地磁力相同原理，得到了修正。于是乎，pitch 和 roll 通过加速度计修正，然后在这个基础之上（该地磁计补偿方法必须依靠加速度计修正提供一致的 XOY 平面，才会有 bx= sqrt((hx*hx) + (hy*hy))等式成立），yaw 通过地磁计来补偿，最终得到了没有偏差的实时姿态（也就是由四元数组成的旋转矩阵）。

对 Gyro 修正补偿以后为：

$$\boldsymbol{\omega}(t) = \boldsymbol{\omega}_{gyro}(t) + \boldsymbol{\omega}_{correction}(t)$$
$$\boldsymbol{\omega}_{gyro}(t) = \text{three axis gyro measurements} \qquad \text{Eqn. 16}$$
$$\boldsymbol{\omega}_{correction}(t) = \text{gyro correction}$$

并把通过 PI 反馈控制器修正过的 gyro vector 作为公式 17 的输入更新 DCM：

$$\mathbf{R}(t+dt) = \mathbf{R}(t)\begin{bmatrix} 1 & -d\theta_z & d\theta_y \\ d\theta_z & 1 & -d\theta_x \\ -d\theta_y & d\theta_x & 1 \end{bmatrix} \qquad \text{Eqn. 17}$$
$$d\theta_x = \omega_x dt$$
$$d\theta_y = \omega_y dt$$
$$d\theta_z = \omega_z dt$$

总结一下修正过程：

获取总的修正量。

$$\mathbf{TotalCorrection} = W_{RP}\mathbf{RollPitchCorrectionPlane} + W_Y\mathbf{YawCorrectionPlane} \qquad \text{Eqn. 28}$$

通过 PI 控制器修正。

$$\boldsymbol{\omega}_{PCorrection} = K_P\mathbf{TotalCorrection}$$
$$\boldsymbol{\omega}_{ICorrection} = \boldsymbol{\omega}_{ICorrection} + K_I dt\mathbf{TotalCorrection} \qquad \text{Eqn. 29}$$
$$\boldsymbol{\omega}_{correction} = \boldsymbol{\omega}_{PCorrection} + \boldsymbol{\omega}_{ICorrection}$$

4、 DCM 更新（重要）

姿态解算过程中不仅需要修正陀螺仪的各种 errors，还需要实时的更新的 DCM。

上周一直没能理解的一个问题，在 AP_AHRS_DCM.cpp 中的 matrix_update(delta_t)，就是实时更新 DCM 矩阵的，源码如下，这一部分研究了很久，需要的基础知识比较多。

```
// update the DCM matrix using only the gyros
Void AP_AHRS_DCM::matrix_update(float _G_Dt)
{
    _omega.zero();
    // average across first two healthy gyros. This reduces noise on
    // systems with more than one gyro. We don't use the 3rd gyro
    // unless another is unhealthy as 3rd gyro on PH2 has a lot more
    // noise
    uint8_t healthy_count = 0;
    Vector3f delta_angle;
    for (uint8_t i=0; i<_ins.get_gyro_count(); i++) {
        if (_ins.get_gyro_health(i) && healthy_count < 2) {
            Vector3f dangle;
            if (_ins.get_delta_angle(i, dangle)) {
                healthy_count++;
```

```
                delta_angle += dangle;
            }
        }
    }
    if (healthy_count > 1) {
        delta_angle /= healthy_count;
    }
    if (_G_Dt > 0) {
        _omega = delta_angle / _G_Dt;
        _omega += _omega_I;
        _dcm_matrix.rotate((_omega + _omega_P + _omega_yaw_P) * _G_Dt);
    }
}
```

首先就是通过陀螺仪获取两次 gyro_vector，然后取平均以降低误差；然后就是比较专业的算法实现_dcm_matrix.rotate((_omega + _omega_P + _omega_yaw_P) * _G_Dt)了。进入到 matrix3.cpp 中有 **rotate()**函数，该函数就是实现 DCM 更新的算法实现，算法主要是用陀螺仪的输出值与 DCM 矩阵的乘积再加回到 DCM 中去，处理过程中使用了离散化的概念，即 dcm（k+1）=dcm（k）+增量，因为公式里有求导，必须离散化后才能计算机处理，感谢青龙的指导。

```
// apply an additional rotation from a body frame gyro vector
// to a rotation matrix.
template <typename T>
void Matrix3<T>::rotate(const Vector3<T> &g)
{
    Matrix3<T> temp_matrix;
    temp_matrix.a.x = a.y * g.z - a.z * g.y;
    temp_matrix.a.y = a.z * g.x - a.x * g.z;
    temp_matrix.a.z = a.x * g.y - a.y * g.x;
    temp_matrix.b.x = b.y * g.z - b.z * g.y;
    temp_matrix.b.y = b.z * g.x - b.x * g.z;
    temp_matrix.b.z = b.x * g.y - b.y * g.x;
    temp_matrix.c.x = c.y * g.z - c.z * g.y;
    temp_matrix.c.y = c.z * g.x - c.x * g.z;
    temp_matrix.c.z = c.x * g.y - c.y * g.x;

    (*this) += temp_matrix;//增加累加到原始数据上
}
```

主要分析一下该算法的实现，该部分如果直接参考 DCM IMU:Theory 中的 Computing direction cosines from gyro signals 部分介绍的公式 17（15 页）有些过于抽象，所以下面介绍一下推导部分。在载体坐标系中定义矢量r^b，可以通过该矢量左乘方向余弦矩阵C_b^n转换到参考系中： $r^n = C_b^n.r^b$。

其中$C_b^n = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$

方向余弦矩阵随时间的传递的推导过程。C_b^n随时间的变化率(标红表示更新后的)如下：

$$\mathbf{C_b^n} = \lim_{\Delta t \to 0} C_b^n / \Delta t = \lim_{\Delta t \to 0} [C_b^n(t+\Delta t) - C_b^n(t)] / \Delta t$$

$C_b^n(t+\Delta t)$可以写成如下两个矩阵相乘的形式：$C_b^n(t+\Delta t) = C_b^n(t)A(t)$

式中 A(t)是一个联系 b 系从 t 时刻到$t+\Delta t$时刻的方向余弦矩阵。对于小角度转动，A(t)可以表示为：A(t)=[I+ΔA]，其中 I 是 3*3 的单位矩阵。且

$$\Delta A = \begin{bmatrix} 0 & -\Delta\theta_z & \Delta\theta_y \\ \Delta\theta_z & 0 & \Delta\theta_x \\ -\Delta\theta_y & \Delta\theta_x & 0 \end{bmatrix}$$

式中$\theta_x\theta_y\theta_z$分别表示 b 系绕其横滚轴、俯仰轴和偏航轴在Δt时间间隔内转动的小角度。在Δt趋近于 0 时，小角度近似是有效的。替换$\boldsymbol{C_b^n}$公式后得到：

$$\boldsymbol{C_b^n} = C_b^n \lim_{\Delta t \to 0} \Delta A / \Delta t$$

在Δt趋近于 0 时，ΔA/Δt是角速度矢量$\boldsymbol{W_{nb}^b} = [w_x, w_y, w_z]^T$的斜对称形式，表示 b 系相对于 n 系在载体轴系的转动角速率。即：

$$\boldsymbol{C_b^n} = C_b^n \Omega \quad \text{其中} \quad \Omega = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

上式可以在捷联惯性导航系统的计算机中解算，以便跟踪载体相对于参考系的姿态，它的分量形式如下。即使用 gyro_vector 更新以后的 DCM：

$$c_{11} = c_{12}w_z - c_{13}w_y, \quad c_{12} = c_{13}w_x - c_{11}w_x, \quad c_{13} = c_{11}w_y - c_{12}w_x$$
$$c_{21} = c_{22}w_z - c_{23}w_y, \quad c_{22} = c_{23}w_x - c_{21}w_x, \quad c_{23} = c_{21}w_y - c_{22}w_x$$
$$c_{31} = c_{32}w_z - c_{33}w_y, \quad c_{32} = c_{33}w_x - c_{31}w_x, \quad c_{33} = c_{31}w_y - c_{32}w_x$$