前言

在导航中一般都具有两套坐标系，载体坐标系与世界坐标系，联系两套坐标系的是用旋转向量，但是旋转向量存在着奇异性，大多使用四元数，记录下四元数的运算及四元数微分方程。

四元数运算

四元数的表达形式为：

$$q = q_0 + q_1 i + q_2 i + q_3 i \tag{1}$$

其中ijk为三个虚部，三个虚部满足以下关系式：

$$\begin{cases} \mathrm{i}^2 = j2 = k^2 = -1 \\ \mathrm{i}j = k, ji = -k \\ \mathrm{j}k = i, kj = -i \\ \mathrm{k}i = j, ik = -j \end{cases} \tag{2}$$

四元数乘法

四元数乘法主要是用在旋转中，$q_k + 1 = \Delta q \otimes q_k$，$\otimes$为乘法符号。
如存在q_a,q_b乘法即为将q_a的每一项与 q_b的每项相乘，最后相加，虚部按照上式计算。
如：

$$q_a \otimes q_b = \\ \begin{cases} \mathrm{s}_a s_b - x_a x_b - y_a y_b - z_a z_b \\ + \left(s_a x_b + x_a s_b + y_a z_b - z_a y_b\right) i \\ + \left(s_a y_b - x_a z_b + y_a s_b + z_a x_b j\right. \\ + \left(s_a z_b + x_a y_b - y_a x_b + z_a s_b\right) k \end{cases}$$

里面有一项外积，所以两个四元数乘法不可交换。

四元数微分

四元数微分之前需要了解下四元数的三角式表达

$$\begin{aligned} &\|Q\| = q_0^2 + q_1^2 + q_2^2 + q_3^2 \\ &Q \text{ 的模 } |Q| = \sqrt{\|Q\|} \\ &\text{令 } \cos\frac{\theta}{2} = q_o / |Q| \\ &\text{令 } \sin\frac{\theta}{2} = \sqrt{\left(q_1^2 + q_2^2 + q_3^2\right)} / |Q| \\ &\hat{n} \text{ 为定向的单位向量 } : \hat{n} = \frac{q_1 i + q_2 j + q_3 k}{\sqrt{\left(q_1^2 + q_2^2 + q_3^2\right)}} \\ &Q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2} \cdot \hat{n} \end{aligned}$$

旋转四元数三角式为：

$$Q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2} \cdot \vec{n} \tag{3}$$

对式（3）两边求导可得：

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = -\frac{1}{2}\sin\frac{\theta}{2} \cdot \frac{\mathrm{d}\theta}{\mathrm{d}t} + \frac{1}{2}\cos\frac{\theta}{2} \cdot \frac{\mathrm{d}\theta}{\mathrm{d}t} \cdot \vec{n} + \sin\frac{\theta}{2} \cdot \frac{\mathrm{d}\vec{n}}{\mathrm{d}t} \tag{4}$$

因为旋转轴$\vec{n}$始终没有改变，则有：

$$\frac{\mathrm{d}\vec{n}}{\mathrm{d}t} = 0 \tag{5}$$

又因为：

$$\vec{n} \otimes \vec{n} = -1 \tag{6}$$

式中$\otimes$表示四元数乘法。整理可以得到：

$$\dot{Q} = \frac{\dot{\theta}}{2} \cdot \vec{n} \otimes \left(\cos\frac{\theta}{2} + \sin\frac{\theta}{2} \cdot \vec{n}\right) \tag{7}$$

而刚体绕瞬时转轴转过的θ角，其角速度为：

$$\omega^n = \dot{\theta} \cdot \vec{n} \tag{8}$$

式子（7）可以改写为：

$$\dot{\boldsymbol{Q}} = \frac{1}{2}\boldsymbol{\omega}^n \otimes \boldsymbol{Q} \tag{9}$$

由于角速度信息由陀螺仪和加速度计融合测得,而陀螺仪和加速度计位于机体坐标系上, 所以 ω^n 还需换算成 $\omega_{\circ}^b$ 根据四元数性质可知：

$$w^n = Q \otimes w^b \otimes Q^* \tag{10}$$

将式 (13)代人式 (12), 应用四元数乘法结合律得:

$$\dot{Q} = \frac{1}{2}\boldsymbol{Q} \otimes \omega^b \tag{11}$$

假设角速度为:

$$\omega^b = [\omega_x, \omega_y, \omega_z]^{\mathrm{T}} \tag{12}$$

则式 (11) 可写成矩阵形式：

$$\dot{Q} = \frac{1}{2}A(t)Q \tag{13}$$

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{14}$$

这个式子就是四元数运动学方程

求解四元数微分方程

可以看出式子（16）是关于Q的一阶齐次线性方程，其通解为：

$$Q(t) = e^{\frac{1}{2}\int_{t_0}^{t} A(r)dr} \cdot Q\left(t_0\right) \tag{15}$$

对该式子离散化可以得到：

$$Q(k+1) = \mathrm{e}^{\frac{1}{2}A(t)\Delta t} \cdot Q(k) \tag{16}$$

式中，假设在一个Δt采样时间间隔内，角速度为恒定的值，令：

$$\Delta\Theta = A(k)\Delta t = \begin{bmatrix} 0 & -\Delta\theta_x & -\Delta\theta_y & -\Delta\theta_z \\ \Delta\theta_x & 0 & \Delta\theta_z & -\Delta\theta_y \\ \Delta\theta_y & -\Delta\theta_z & 0 & \Delta\theta_x \\ \Delta\theta_z & \Delta\theta_y & -\Delta\theta_x & 0 \end{bmatrix} \tag{17}$$

式中, $\Delta\theta_x, \Delta\theta_y, \Delta\theta_z$ 为 x, y, z 轴在 Δt 采样时间间隔内的角 增量。
根据矩阵函数的幂级数表达式, 对式 (16) 进行展 开,得:

$$Q(k+1) = \left[I + \frac{\frac{1}{2}\Delta\Theta}{1!} + \frac{\left(\frac{1}{2}\Delta\Theta\right)^2}{2!} + \frac{\left(\frac{1}{2}\Delta\Theta\right)^3}{3!} + \cdots\right] Q(k)$$

对上式取有限项, 可得四元数的二阶近似算法:

$$Q(k+1) = \left[I\left(1 - \frac{\Delta\theta^2}{8}\right) + \frac{\Delta\Theta}{2}\right] \cdot Q(k) \tag{18}$$

$$\text{式中}, \Delta\theta^2 = \Delta\theta_x^2 + \Delta\theta_y^2 + \Delta\theta_z^2 \tag{19}$$

基于icm20609姿态更新计算的四元数算法

1. 数据采集

- 这里使用软件IIC来进行采集原始数据。
- 采样率是200HZ，程序中也是每5ms采集并更新一次姿态
- 陀螺仪的范围选择±2000，而对应的精度是16.4 LSB/(°/s)，而在四轴姿态计算中，我们通常要把角度换算成弧度。我们知道2Pi代表360度，那么1度换算成弧度就是： 2Pi/360=(2*3.1415926)/360=0.0174532=1/57.30。
- 加速度计选择的量程是±8G，精度为4096LSB/g

软件iic进行读取原始数据：

```
int8_t icm20609_accelAndGyro_read(accel_t *accel, gyro_t *gyro, int16_t temp)
{
  u8  reg_data[14];
  ICM20609_ReadData(ICM20609_ACCEL_XOUT_H_REG, (u8 *)&reg_data, 14);

  accel->x = (int16_t)(reg_data[0] << 8 | reg_data[1]);
  accel->y = (int16_t)(reg_data[2] << 8 | reg_data[3]);
  accel->z = (int16_t)(reg_data[4] << 8 | reg_data[5]);
  temp    = (int16_t)(reg_data[6] << 8 | reg_data[7]);
  gyro->x  = (int16_t)(reg_data[8] << 8 | reg_data[9]);
  gyro->y  = (int16_t)(reg_data[10] << 8 | reg_data[11]);
  gyro->z  = (int16_t)(reg_data[12] << 8 | reg_data[13]);
  return 0x01;
}
```

将原始数据进行转换并传给姿态解算的函数：

```
//get accel and gyro from iam20609
// 对accel一阶低通滤波(参考匿名)，对gyro转成弧度每秒(2000dps)
#define new_weight        0.35f
#define old_weight        0.65f
void IMU_getValues(float * values, accel_t * accelval, gyro_t * gyroval)
{
   static float lastaccel[3]= {0,0,0};
   int i;
   values[0] = ((float)accelval->x) * new_weight + lastaccel[0] * old_weight;
   values[1] = ((float)accelval->y) * new_weight + lastaccel[1] * old_weight;
   values[2] = ((float)accelval->z) * new_weight + lastaccel[2] * old_weight;
   for(i=0; i<3; i++)
   {
      lastaccel[i] = values[i];
   }

   values[3] = ((float)gyroval->x) * M_PI / 180 / 16.4f;
   values[4] = ((float)gyroval->y) * M_PI / 180 / 16.4f;
   values[5] = ((float)gyroval->z) * M_PI / 180 / 16.4f;

   //
}


```

2. 姿态解算

按照 原始数据->四元数->欧拉角的方式进行姿态解算并从匿名上位机中进行观察

```
#define delta_T     0.005f  //5ms计算一次

float I_ex, I_ey, I_ez;  // 误差积分
quaterInfo_t Q_info;  // 全局四元数//quaterInfo_ts是一个结构体，其中包括q0,q1,q2,q3
eulerianAngles_t eulerAngle; //欧拉角也是一个结构体，其中包括三个角度
float param_Kp = 50.0;   // 加速度计(磁力计)的收敛速率比例增益50
float param_Ki = 0.20;   //陀螺仪收敛速率的积分增益 0.2
```

```
/**
  * brief IMU_AHRSupdate_noMagnetic  姿态解算融合，是Crazepony和核心算法
  * 使用的是互补滤波算法，没有使用Kalman滤波算法
  * param float gx, float gy, float gz, float ax, float ay, float az
  *
  * return None
  */
static void IMU_AHRSupdate_noMagnetic(float gx, float gy, float gz, float ax, float ay, float az)
{
   float halfT = 0.5 * delta_T;
   float vx, vy, vz;   //当前的机体坐标系上的重力单位向量
   float ex, ey, ez;   //四元数计算值与加速度计测量值的误差
   float q0 = Q_info.q0;
   float q1 = Q_info.q1;
   float q2 = Q_info.q2;
   float q3 = Q_info.q3;
```

```
    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;
    float delta_2 = 0;
```

- 输入参数gx，gy，gz分别对应三个轴的角速度，单位是弧度/秒。
- 输入参数ax，ay，az分别对应三个轴的加速度数据，由于加速度的噪声较大，在读取的时候该数据是采用低通滤波。

当电子罗盘数据有效的时候，需要融合电子罗盘的数据。这里没有使用磁力计，所以只进行加速度数据融合。

```
    //对加速度数据进行归一化 得到单位加速度
    float norm = invSqrt(ax*ax + ay*ay + az*az);
    ax = ax * norm;
    ay = ay * norm;
    az = az * norm;
```

对加速度数据进行归一化，得到单位加速度。这里介绍一种快速计算1/sqrt(x)的函数：

```
float invSqrt(float x) {
    float halfx = 0.5f * x;
    float y = x;
    long i = *(long*)&y;
    i = 0x5f3759df - (i>>1);
    y = *(float*)&i;
    y = y * (1.5f - (halfx * y * y));
    return y;
}
```

```
    vx = 2*(q1q3 - q0q2);
    vy = 2*(q0q1 + q2q3);
    vz = q0q0 - q1q1 - q2q2 + q3q3;
```

把飞行器上次计算得到的姿态（四元数）换算成“方向余弦矩阵”中的第三列的三个元素。根据余弦矩阵和欧拉角的定义，地理坐标系的重力向量，转到机体坐标系，正好是这三个元素。所以这里的vx、vy、vz，其实就是用上一次飞行器机体姿态（四元数）换算出来的在当前的机体坐标系上的重力单位向量。

推导：

实际上，我们也可以通过另一种方式来验证四元数与姿态矩阵间的关系，如果将 r^* 和 r^b 看做零标量的四元数，那么 我们定义一个四元数 $\mathbf{q_b}$, 用来表示从导航坐标系 n 和载体坐标系 b 之间的放转变换：

$$
\begin{aligned}
\mathrm{r}^* &= \boldsymbol{q}_b^n \otimes \boldsymbol{r}^b \otimes \left(q_b^n\right)^* \\
&= M\left(q_b^n\right) r^b M\left(\left(q_b^n\right)^*\right)
\end{aligned}
$$

$$
\mathrm{M}(\mathrm{P}_b^*)^* \mathrm{M}\left((\mathrm{P}_b^n)^*\right) = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} * \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix}
$$

代入求得，

$$
\begin{bmatrix} 0 \\ r_x^n \\ r_y^n \\ r_z^n \end{bmatrix} = \begin{bmatrix} q_0^2+q_1^2+q_2^2+q_3^2 & 0 & 0 & 0 \\ 0 & q_0^2+q_1^2-q_2^2-q_3^2 & 2\left(q_1 q_2 - q_0 q_3\right) & 2\left(q_1 q_3 + q_0 q_2\right) \\ 0 & 2\left(q_1 q_2 + q_0 q_3\right) & {q_0}^2-{q_1}^2+{q_2}^2-{q_3}^2 & 2\left(q_2 q_3 - q_0 q_1\right) \\ 0 & 2\left(q_1 q_3 - q_0 q_2\right) & 2\left(q_2 q_3 + q_0 q_1\right) & {q_0}^2-{q_1}^2-{q_2}^2+{q_3}^2 \end{bmatrix} * \begin{bmatrix} 0 \\ r_x^b \\ r_y^b \\ r_z^b \end{bmatrix} \quad (20)
$$

上式矩阵中右下角的 3^*3 方块即为 C_b^n ，便可以得到：

$$
\begin{bmatrix} r_x^n \\ r_y^n \\ r_z^n \end{bmatrix} = \begin{bmatrix} {q_0}^2+{q_1}^2-{q_2}^2-{q_3}^2 & 2\left(q_1 q_2 - q_0 q_3\right) & 2\left(q_1 q_3 + q_0 q_2\right) \\ 2\left(q_1 q_2 + q_0 q_3\right) & {q_0}^2-{q_1}^2+{q_2}^2-{q_3}^2 & 2\left(q_2 q_3 - q_0 q_1\right) \\ 2\left(q_1 q_3 - q_0 q_2\right) & 2\left(q_2 q_3 + q_0 q_1\right) & {q_0}^2-{q_1}^2-{q_2}^2+{q_3}^2 \end{bmatrix} * \begin{bmatrix} r_x^b \\ r_y^b \\ r_z^b \end{bmatrix} = \mathbf{C}_b^n * \begin{bmatrix} r_x^b \\ r_y^b \\ r_z^b \end{bmatrix} \quad (21)
$$

```
ex = ay * vz - az * vy;
ey = az * vx - ax * vz;
ez = ax * vy - ay * vx;
```

在机体坐标系上，加速度计测出来的重力向量是$[a_x, a_y, a_z]$。由上次解算的姿态推算出的重力向量是$[v_x, v_y, v_z]$。它们之间的误差向量，就是上次姿态和加速度计测出来的姿态之间的误差。

向量间的误差，可以用向量积(也叫外积、叉乘)来表示，$[e_x, e_y, e_z]$就是两个重力向量的叉积。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。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。

所以上面一段代码含义就是获得叉乘误差。

```
//用叉乘误差来做PI修正陀螺零偏，
//通过调节 param_Kp，param_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。
I_ex += delta_T * ex;  // integral error scaled by Ki
I_ey += delta_T * ey;
I_ez += delta_T * ez;

gx = gx+ param_Kp*ex + param_Ki*I_ex;
gy = gy+ param_Kp*ey + param_Ki*I_ey;
gz = gz+ param_Kp*ez + param_Ki*I_ez;
```

上面一段代码，叉乘误差进行积分。用叉乘误差来做PI修正陀螺零偏，通过调节param_Kp，param_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。

到此为止，使用加速度计数据对陀螺仪数据的修正已经完成，这就是姿态解算中的深度融合。

下面就是四元数微分方程，使用修正后的陀螺仪数据对时间积分，得到飞行器的当前姿态（四元数表示）。然后进行四元数的单位化处理。

```
//四元数微分方程，其中halfT为测量周期的1/2，gx gy gz为陀螺仪角速度，以下都是已知量，这里使用了一阶龙哥库塔求解四元数微分方程
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;

//    delta_2=(2*halfT*gx)*(2*halfT*gx)+(2*halfT*gy)*(2*halfT*gy)+(2*halfT*gz)*(2*halfT*gz);
    // 整合四元数率  四元数微分方程 四元数更新算法，二阶毕卡法
//    q0 = (1-delta_2/8)*q0 + (-q1*gx - q2*gy - q3*gz)*halfT;
//    q1 = (1-delta_2/8)*q1 + (q0*gx + q2*gz - q3*gy)*halfT;
//    q2 = (1-delta_2/8)*q2 + (q0*gy - q1*gz + q3*gx)*halfT;
//    q3 = (1-delta_2/8)*q3 + (q0*gz + q1*gy - q2*gx)*halfT;

    // normalise quaternion
    norm = invSqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
    Q_info.q0 = q0 * norm;
    Q_info.q1 = q1 * norm;
    Q_info.q2 = q2 * norm;
    Q_info.q3 = q3 * norm;
```

最后通过四元数计转换欧拉角
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

```
void IMU_quaterToEulerianAngles()
{
    float q0 = Q_info.q0;
    float q1 = Q_info.q1;
    float q2 = Q_info.q2;
    float q3 = Q_info.q3;
    eulerAngle.pitch = asin(-2*q1*q3 + 2*q0*q2) * 180/M_PI; // pitch
    eulerAngle.roll = atan2(2*q2*q3 + 2*q0*q1, -2*q1*q1 - 2*q2*q2 + 1) * 180/M_PI; // roll
    eulerAngle.yaw = atan2(2*q1*q2 + 2*q0*q3, -2*q2*q2 - 2*q3*q3 + 1) * 180/M_PI; // yaw

/*  可以不用作姿态限度的限制
    if(eulerAngle.roll>90 || eulerAngle.roll<-90)
    {
        if(eulerAngle.pitch > 0)
        {
            eulerAngle.pitch = 180-eulerAngle.pitch;
        }
        if(eulerAngle.pitch < 0)
        {
            eulerAngle.pitch = -(180+eulerAngle.pitch);
        }
    }
    if(eulerAngle.yaw > 180)
    {
        eulerAngle.yaw -=360;
    }
    else if(eulerAngle.yaw <-180)
    {
        eulerAngle.yaw +=360;
    }
    */
}
```

这里贴一下融合磁力计的四元数更新算法：

```
static void IMU_AHRSupdate_withMagnetic(float gx, float gy, float gz, float ax, float ay, float az,
float mx, float my, float mz)
{
  float norm;
  float halfT = 0.5 * delta_T;
  float hx, hy, hz;
  float bx, bz;
  float vx, vy, vz;
  float wx, wy, wz;
  float ex, ey, ez;
  float delta_2 = 0;
  float q0 = Q_info.q0;
  float q1 = Q_info.q1;
  float q2 = Q_info.q2;
  float q3 = Q_info.q3;
  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;

  norm = invSqrt(ax*ax + ay*ay + az*az);
  ax = ax * norm;
  ay = ay * norm;
  az = az * norm;

  norm = invSqrt(mx*mx + my*my + mz*mz);
  mx = mx * norm;
  my = my * norm;
  mz = mz * norm;

  // compute reference direction of flux 通量的计算参考方向
  hx = 2*mx*(0.5f - q2q2 - q3q3) + 2*my*(q1q2 - q0q3) + 2*mz*(q1q3 + q0q2);
  hy = 2*mx*(q1q2 + q0q3) + 2*my*(0.5f - q1q1 - q3q3) + 2*mz*(q2q3 - q0q1);
  hz = 2*mx*(q1q3 - q0q2) + 2*my*(q2q3 + q0q1) + 2*mz*(0.5f - q1q1 - q2q2);
  bx = sqrt((hx*hx) + (hy*hy));
  bz = hz;

  // estimated direction of gravity and flux (v and w)
  vx = 2*(q1q3 - q0q2);                 // 估计方向的重力
  vy = 2*(q0q1 + q2q3);
  vz = q0q0 - q1q1 - q2q2 + q3q3;
  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

  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);
```

```
56
57    if(ex != 0.0f && ey != 0.0f && ez != 0.0f)
58    {
59
60      I_ex += delta_T * ex;   // integral error scaled by Ki
61      I_ey += delta_T * ey;
62      I_ez += delta_T * ez;
63      //exdev=(ex[1]-ex[0]) / halfT;
64      //eydev=(ey[1]-ey[0]) / halfT;
65      // ezdev=(ez[1]-ez[0]) / halfT;
66      // adjusted gyroscope measurements
67
68      gx = gx+ param_Kp*ex + param_Ki*I_ex;//+ Kd*exdev;
69      gy = gy+ param_Kp*ey + param_Ki*I_ey;//+ Kd*eydev;
70      gz = gz+ param_Kp*ez + param_Ki*I_ez;//+ Kd*ezdev;
71    }
72    delta_2=(2*halfT*gx)*(2*halfT*gx)+(2*halfT*gy)*(2*halfT*gy)+(2*halfT*gz)*(2*halfT*gz);
73    // 整合四元数率  四元数微分方程 四元数更新算法，二阶毕卡法
74    q0 = (1-delta_2/8)*q0 + (-q1*gx - q2*gy - q3*gz)*halfT;
75    q1 = (1-delta_2/8)*q1 + (q0*gx + q2*gz - q3*gy)*halfT;
76    q2 = (1-delta_2/8)*q2 + (q0*gy - q1*gz + q3*gx)*halfT;
77    q3 = (1-delta_2/8)*q3 + (q0*gz + q1*gy - q2*gx)*halfT;
78    // normalise quaternion
79    norm = invSqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
80    Q_info.q0 = q0 * norm;
81    Q_info.q1 = q1 * norm;
82    Q_info.q2 = q2 * norm;
83    Q_info.q3 = q3 * norm;
84  }
```

详解互补滤波四元数中向量叉积与陀螺仪角速度补偿问题（Mahony算法）

一、归一化与坐标转换

很多做四轴的同学对互补滤波四元数姿态解算代码中的向量叉积和陀螺仪积分补偿问题有疑问
先放一段互补滤波和四元数姿态解算的代码：

```
1  /**
2   * 6DOF 互补滤波姿态估计(via Mahony)
3   * @param[in] halfT:状态估计周期的一半
4   */
5  const float Kp = 3.5, Ki = 0.05;
6  float exInt, eyInt, ezInt;
7  float q0 = 1.0f, q1 = 0.0f, q2 = 0.0f, q3 = 0.0f; // roll,pitch,yaw 都为 0 时对应的四元数值。
8  void IMUupdate(float gx, float gy, float gz, float ax,float ay, float az)
9  {
10   float norm;
11   float vx, vy, vz;
12   float ex, ey, ez;
13
14   float q0q0 = q0*q0;
15   float q0q1 = q0*q1;
16   float q0q2 = q0*q2;
```

```
    float q1q1 = q1*q1;
    float q1q3 = q1*q3;
    float q2q2 = q2*q2;
    float q2q3 = q2*q3;
    float q3q3 = q3*q3;

    if(ax*ay*az==0)
      return;

    // 第一步：对加速度数据进行归一化
    norm = sqrt(ax*ax + ay*ay + az*az);
    ax = ax / norm;
    ay = ay / norm;
    az = az / norm;

    // 第二步：DCM矩阵旋转
    vx = 2*(q1q3 - q0q2);
    vy = 2*(q0q1 + q2q3);
    vz = q0q0 - q1q1 - q2q2 + q3q3 ;

    // 第三步：在机体坐标系下做向量叉积得到补偿数据
    ex = ay*vz - az*vy ;
    ey = az*vx - ax*vz ;
    ez = ax*vy - ay*vx ;

    // 第四步：对误差进行PI计算，补偿角速度
    exInt = exInt + ex * Ki;
    eyInt = eyInt + ey * Ki;
    ezInt = ezInt + ez * Ki;

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

    // 第五步：按照四元数微分公式进行四元数更新
    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;

    norm = sqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
    q0 = q0/norm;
    q1 = q1/norm;
    q2 = q2/norm;
    q3 = q3/norm;

    roll =  atan2f(2*q2*q3 + 2*q0*q1, -2*q1*q1 - 2*q2*q2 + 1)*57.3;
    pitch =  asinf(2*q1*q3 - 2*q0*q2)*57.3;
    yaw  =  -atan2f(2*q1*q2 + 2*q0*q3, -2*q2*q2 -2*q3*q3 + 1)*57.3;
}
```

这段代码中 $[a_x, a_y, a_z]$是加速度计测量得到的重力向量, 然后对其归一化, 但是为什么要对它归一化处理呢? 不是画蛇添足, 而是后面大有用途!

$[\mathrm{Vx}, \mathrm{Vy}, \mathrm{Vz}]$ 是世界坐标系重力分向量 $[0, 0, 1]$ 经过DCM旋转矩阵计算得到的机体坐标系中的重力向量, 即:

$$\begin{bmatrix} vx \\ vy \\ vz \end{bmatrix} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2*(q_1q_2-q_0q_3) & 2*(q_1q_3+q_0q_2) \\ 2*(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2*(q_2q_3-q_0q_1) \\ 2*(q_1q_3-q_0q_2) & 2*(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (22)$$

这个旋转矩阵在秦永元的惯性导航书中有详细解释，这里就看出来了，旋转得到的正是机体坐标系归一化后的重力向量。

有人可能纳闷为什么要进行旋转？这就涉及到世界坐标系和机体坐标系的概念了，我从网上摘了几张图进行简单讲解。

给出一个"小球-盒子"模型，将重力加速度看成一个实体黑色"小球"，加速计就看成"盒子"。当物体失重时，盒子不受重力。如图1。

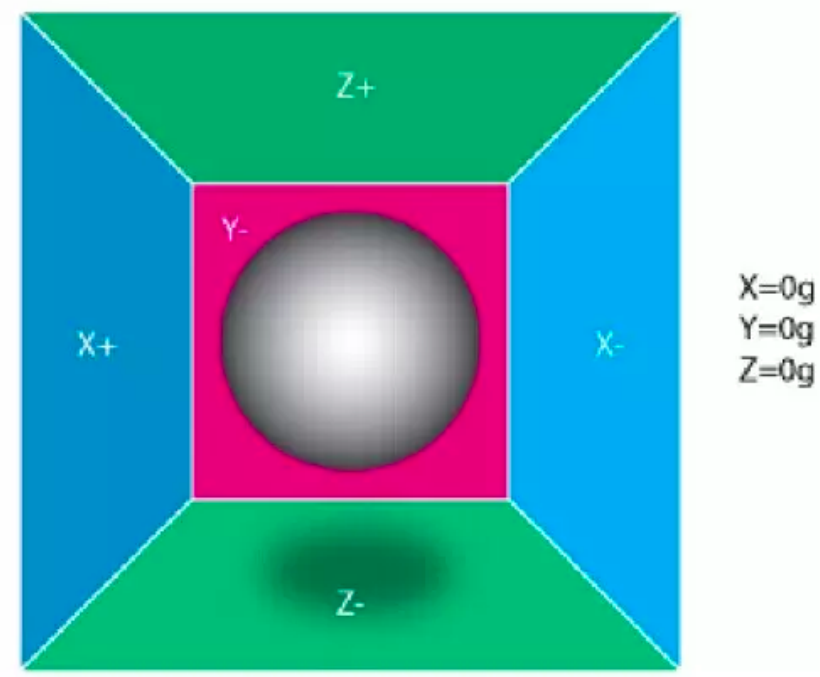
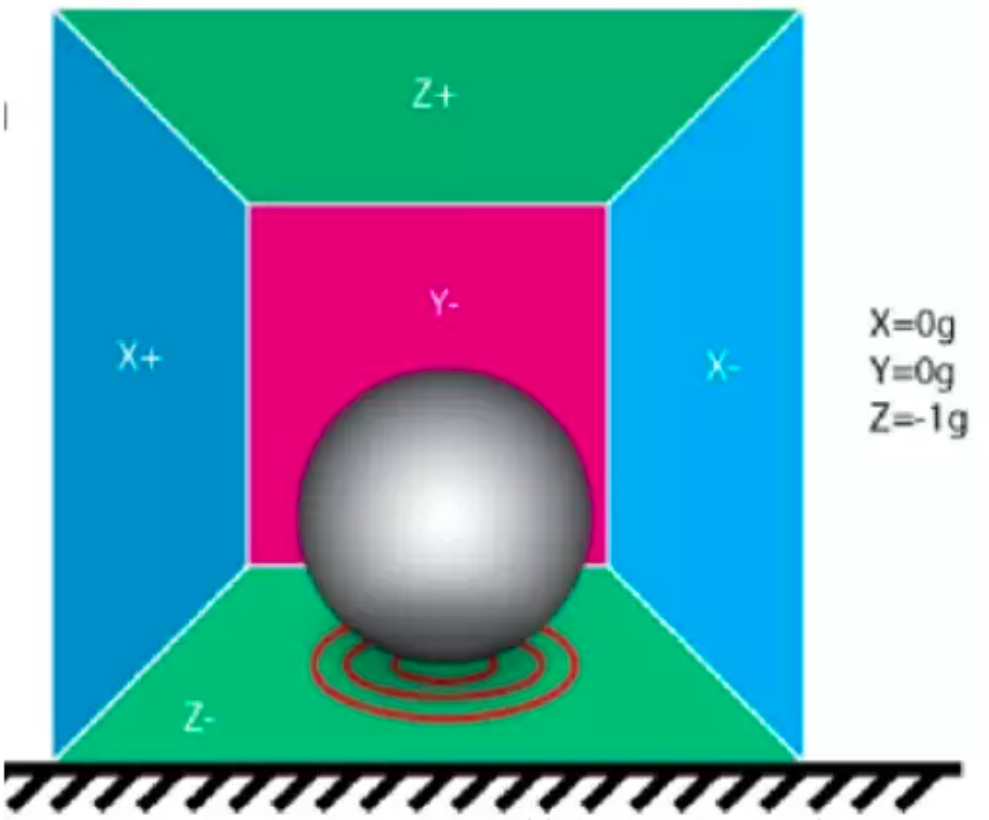


图1 失重状态

当物体放在水平面地面上时，仅Z轴受重力。如图2。



当物体右倾斜放45°放在地上时， X轴和Z轴受重力的分力。如图3。

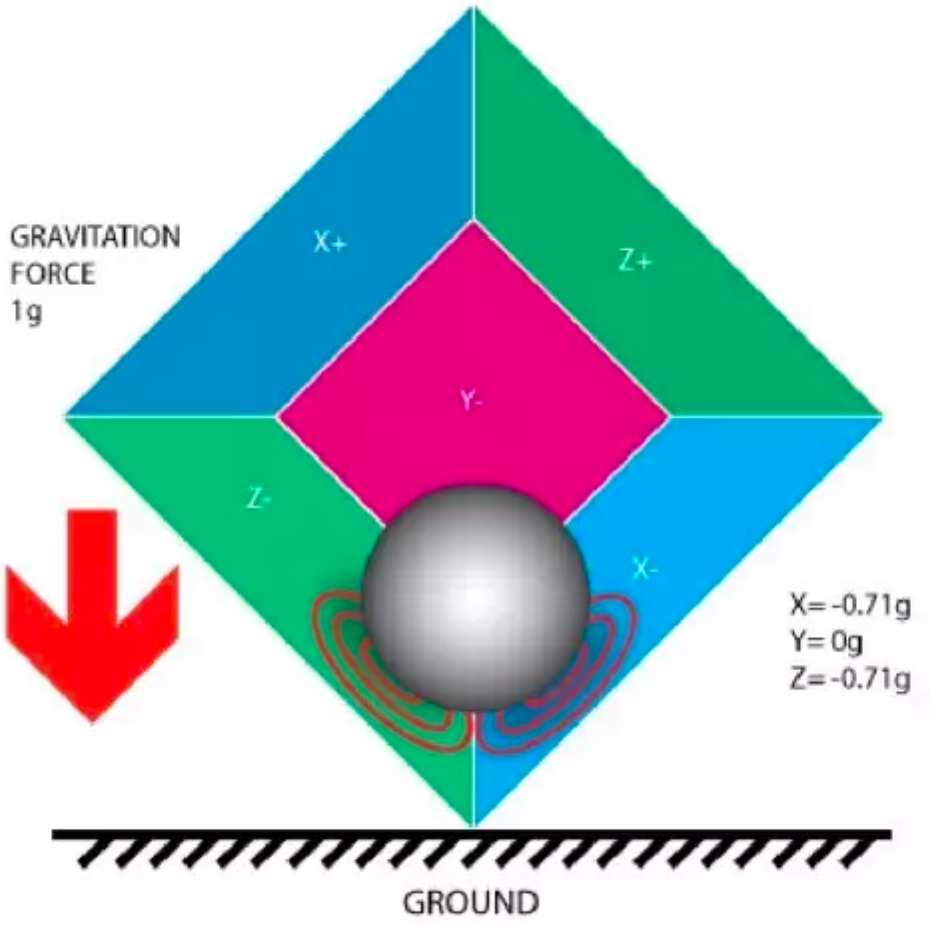


图3 当物体右倾斜放45°放在地上时

- 图1中盒子模型中的小球处于失重状态，坐标系与世界坐标系重合。
- 图2中盒子坐标系仍与世界坐标系重合，描述为$[x, y, z]$,不同的是小球收到重力作用，盒子模型坐标系中重力向量为$[0, 0, 1]$,世界坐标系中也是$[0, 0, 1]$.
- 图3中盒子在世界坐标系中与地面夹角为45度，但是盒子坐标系仍是$[x, y, z]$，重力向量变为了$[-0.71, 0, -0.71]$，注意这个向量是盒子模型坐标系的向量值，转换到世界坐标系仍是$[0, 0, 1]$. 所以无论盒子怎么旋转重力向量在世界坐标系中都是$[0, 0, 1]$,但是我们有加速度计读到的值是基于盒子模型坐标系的，也就是我们所说的机体坐标系。现在就明白了，刚才为什么要用四元数将$[0, 0, 1]$乘一个旋转矩阵，我们之后关于加速度计和陀螺仪的计算都是基于机体坐标系的。

二、叉乘运算

重点来了，我们得到了加速度计测得的$[a_x, a_y, a_z]$，还有机体坐标系标准的重力向量，下面我们就要对它们做叉积运算：
首先我们先来一点向量叉积的预备知识

1. 矢量的点乘与数量积
   1.1 数量积的定义
   两个矢量 $\vec{a}$、$\vec{b}$ 的点乘 $\vec{a} \cdot \vec{b}$ 称为两个矢量的数量积, 也称为内积。数量积是一个数量, 或叫标量。
   1.2 点乘的坐标表示
   若 $\vec{a} = (x_1, y_1, z_1)^T, \vec{b} = (x_2, y_2, z_2)^T$, 则其数量积为$\vec{a} \cdot \vec{b} = (\vec{a})^T \vec{b} = x_1 x_2 + y_1 y_2 + z_1 z_2$
   2 矢量的叉乘与向量积

2. 1 向量积的定义

两个矢量 $\vec{a}$、$\vec{b}$ 的叉乘 $\overrightarrow{\mathbf{a}} \times \vec{b}$ 称为两个矢量的向量积, 也称为外积。

向量积是一个向量, 或叫矢量。其大小为 $|\vec{a}||\vec{b}| \sin\theta$, 其中 θ 为 $\vec{a}$、$\vec{b}$ 的夹角; 方向与 与 $\vec{a}$、$\vec{b}$ 所在平面垂直, 且 $\vec{a}$、$\vec{b}$、$\overrightarrow{\mathbf{a}} \times \vec{b}$ 构成右手系。

2.2 叉乘的坐标表示

若 $\vec{a} = (x_1, y_1, z_1)^T, \vec{b} = (x_2, y_2, z_2)^T$, 则其向量积可用坐标表示为

$$\overrightarrow{\mathbf{a}} \times \overrightarrow{\mathbf{b}} = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_2 \\ y_2 \\ z_2 \end{bmatrix} = (\overrightarrow{\mathbf{a}} \times) \cdot \overrightarrow{\mathbf{b}} \tag{23}$$

$$\text{其中 } \overrightarrow{\mathbf{a}} \times \text{ 称为向量 } \vec{a} \text{ 的叉乘反对称阵, 其定义为} \tag{24}$$

$$\overrightarrow{\mathbf{a}} \times = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \tag{25}$$

可以看出, 通过叉乘反对称阵的引入, 矢量的向量积可以通过矩阵乘法表示, 形式非常 简洁。不用求出两个矢量的夹角就可以求出其向量积, 所以在理论计算中具有很高的应用价 值。

由上面知识我们得到了下面的计算矩阵：

$$\begin{bmatrix} ex \\ ey \\ ez \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az \end{bmatrix} \times \begin{bmatrix} vx \\ vy \\ vz \end{bmatrix} = \begin{bmatrix} 0 & -az & ay \\ az & 0 & -ax \\ -ay & ax & 0 \end{bmatrix} \begin{bmatrix} vx \\ vy \\ vz \end{bmatrix} \tag{26}$$

```
ex = ay*vz - az*vy ;
ey = az*vx - ax*vz ;
ez = ax*vy - ay*vx ;
```

得到的这个向量叉积就是姿态误差向量, Mahony论文中是这样说的： the error e is the relative rotation between the measured inertial direction $\bar{v}$ and the predicted direction $\hat{v}$. 翻译过来就 是：误差e就是测量得到的 $\bar{v}$ 和预测得到的 $\hat{v}$ 之间的相对旋转。这里的 $\bar{v}$ 就是 $[a_x, a_y, a_z]$, $\hat{v}$ 就是 $[v_x, a_y, v_z]$ 。

怎么理解这句话呢?

通过整个程序的流程我们知道欧拉角是用更新后的四元数解得的, 设是t时刻更新后的四元数为 $Q(t)$, 假定在 $t + T$ 时刻（下一时刻） $Q(t + T) = Q(t)$,即用上一时刻的四元数预测当前时刻的四元数仍为 $Q(t)$ 。然 后将加速度计测得的值归一化后与预测四元数表示的重力向量做叉积运算。

归一化后的加速度计值与上一时刻预测得到的四元数做叉积运算，只要机体存在旋转也就是说只要机体姿态发生偏移，叉积结果必然不为0，那么该“误差”值就会一直做PI运算对陀螺仪进行补偿，直到机体停止旋转。若停止旋转后叉积值仍不为0，就说明我们预测的值不正确，同样会进行PI互补滤波重新对当前时刻进行预测，直到预测值收敛到加速度计值。

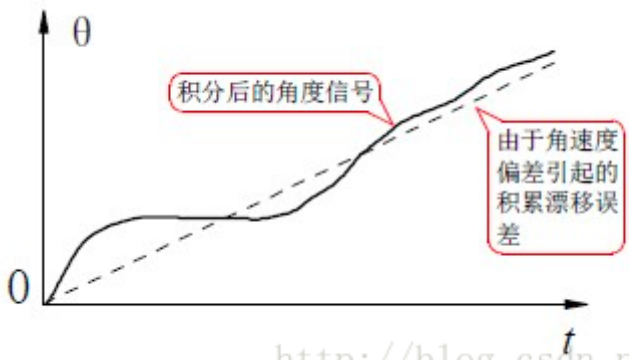
总结为二点：

1、机体做旋转运动时，叉积结果就是当前时刻加速度计结果相对于上一时刻重力向量的旋转和测量结果误差的叠加。

2、机体静止时，没有旋转；叉积结果若不为0，说明预测值有误差，因而就会一直做PI运算，直到结果收敛到加速度计归一化值。

三、分析为什么叉积运算结果能对陀螺仪角速度进行补偿?

1. 分析公式
   两个向量的叉积得到的结果是两个向量的模与他们之间夹角正弦的乘积a×v=|a||v|sinθ
   ,加速度计测量得到的重力向量和预测得到的机体重力向量（前面已经详细解释了这个是由世界坐标系下的[0,01]旋转得到的）已经经过单位化，因而他们的模是1，也就是说它们向量的叉积结果仅与sinθ有关，当角度很小时，叉积结果可以近似于角度成正比。
2. 物理意义
   这个进行叉积得到的误差结果近似等于当期时刻相对于上一时刻旋转的角度，这个角度即是旋转角度和预测误差角度的叠加，总称为向量误差，角度越大，误差越大。这里是不是很神奇，我们用加速度计的值间接得出了误差变化的角度相关值。
3. 陀螺仪误差由来



陀螺仪由于本身精度问题，测量的角速度存在误差，在积分过程中这个误差会一直累加，我们要做的就是去消除或是补偿这个误差，因为加速度计长期的测量值是准确的，所以可以用加速度计来进行修正。

4. 如何修正
   `void IMUupdate(float gx, float gy, float gz, float ax,float ay, float az){}`
   如何找到一个另一个角速度量纲的值来修正陀螺仪的角速度值呢？这里明明只有陀螺仪可以测量角速度！！！
   这时候前面提到的向量叉积得到的误差向量就帮上大忙了，这个误差向量不就是反映出了角度变化量吗。算法巧妙的将加速度相关量转化为角度相关量，因而可以用这个角度值乘一个系数来修正陀螺仪的角速度，因为在偏差角度很小的情况下，我们可以将陀螺仪角速度误差和加速度计求得的角度差看做正比的关系，也就说明陀螺仪积分误差和向量叉积存在正比关系。

$$\omega = \omega + kp * \theta acc \tag{27}$$

最后得到PI滤波计算公式：

```
exInt = exInt + ex * Ki;
eyInt = eyInt + ey * Ki;
ezInt = ezInt + ez * Ki;

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

其中：

1. 比例增益kp控制收敛到加速度计速率
2. 积分增益ki控制陀螺仪偏差的收敛速率
   有人会问，为什么不能直接将预测的重力向量和测量得到的重力向量直接相减呢?上面已经解释了用叉积运算时有相应的物理意义的，二直接相减得到的向量值与角度无关，是不能用来补偿角速度的。