> 目前队里现有的资料里面有两份陀螺仪，一份是在f0上直接使用互补滤波解算，一份是倍哥移植的玺佬的EKF，我将两份代码和一些IMU基础的知识做一个简单的总结，对于KF的知识来说建议直接看倍哥总结的资料。

https://zhuanlan.zhihu.com/p/378581141?utm_psn=1721980571100770304
https://zhuanlan.zhihu.com/p/61765669?utm_psn=1721980691233992704
http://t.csdnimg.cn/tCj9c

飞控算法姿态解算
飞控算法-姿态解算之互补滤波_互补滤波姿态解算-CSDN博客
几种飞控的姿态解算算法 - 知乎 (zhihu.com)

# 基础知识

## 欧拉角转换为四元数

```
struct Quaternion
{
    double w, x, y, z;
};

Quaternion ToQuaternion(double yaw, double pitch, double roll) // yaw (Z), pitch (Y), roll (X)
{
    // Abbreviations for the various angular functions
    double cy = cos(yaw * 0.5);
    double sy = sin(yaw * 0.5);
    double cp = cos(pitch * 0.5);
    double sp = sin(pitch * 0.5);
    double cr = cos(roll * 0.5);
    double sr = sin(roll * 0.5);

    Quaternion q;
    q.w = cy * cp * cr + sy * sp * sr;
```

```
    q.x = cy * cp * sr - sy * sp * cr;
    q.y = sy * cp * sr + cy * sp * cr;
    q.z = sy * cp * cr - cy * sp * sr;

    return q;
```

# 四元数转欧拉角

```
// roll (x-axis rotation)
double sinr_cosp = 2 * (q.w * q.x + q.y * q.z);
double cosr_cosp = 1 - 2 * (q.x * q.x + q.y * q.y);
angles.roll = std::atan2(sinr_cosp, cosr_cosp);

// pitch (y-axis rotation)
double sinp = 2 * (q.w * q.y - q.z * q.x);
if (std::abs(sinp) >= 1)
    angles.pitch = std::copysign(M_PI / 2, sinp); // use 90 degrees if out of range
else
    angles.pitch = std::asin(sinp);

// yaw (z-axis rotation)
double siny_cosp = 2 * (q.w * q.z + q.x * q.y);
double cosy_cosp = 1 - 2 * (q.y * q.y + q.z * q.z);
angles.yaw = std::atan2(siny_cosp, cosy_cosp);
```

# 欧拉角和旋转矩阵

假设绕XYZ三个轴旋转的角度分别为 $\alpha\beta\gamma$ ，则三次旋转的旋转矩阵计算方法如下：

$$
\begin{aligned}
\mathrm{R_x}(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \\
\mathrm{R_y}(\beta) &= \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \\
\mathrm{R_z}(\gamma) &= \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

按照内旋方式，Z-Y-X旋转顺序（指先绕自身轴Z，再绕自身轴Y，最后绕自身轴X），可得旋转矩阵（内旋是右乘）：

$$\mathrm{R1} = \mathrm{R_z}(\gamma) * \mathrm{R_y}(\beta) * \mathrm{R_x}(\alpha)$$

按照外旋方式，X-Y-Z旋转顺序（指先绕固定轴X，再绕固定轴Y，最后绕固定轴Z），可得旋转矩阵（外旋是左乘）：

$$\mathrm{R2} = \mathrm{R_z}(\gamma) * \mathrm{R_y}(\beta) * \mathrm{R_x}(\alpha)$$

故R1=R2，具体不在此证明，记住即可。这个结论说明ZYX顺序的内旋等价于XYZ顺序的外旋

**Tait–Bryan angles**

$$X_1Z_2Y_3 = \begin{bmatrix} c_2c_3 & -s_2 & c_2s_3 \\ s_1s_3 + c_1c_3s_2 & c_1c_2 & c_1s_2s_3 - c_3s_1 \\ c_3s_1s_2 - c_1s_3 & c_2s_1 & c_1c_3 + s_1s_2s_3 \end{bmatrix}$$

$$X_1Y_2Z_3 = \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 & -c_2s_1 \\ s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$$

$$Y_1X_2Z_3 = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - c_3s_1 & c_1c_3s_2 + s_1s_3 & c_1c_2 \end{bmatrix}$$

$$Y_1Z_2X_3 = \begin{bmatrix} c_1c_2 & s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 \\ s_2 & c_2c_3 & -c_2s_3 \\ -c_2s_1 & c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 \end{bmatrix}$$

$$Z_1Y_2X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 \\ c_2s_1 & c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 \\ -s_2 & c_2s_3 & c_2c_3 \end{bmatrix}$$

$$Z_1X_2Y_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix}$$

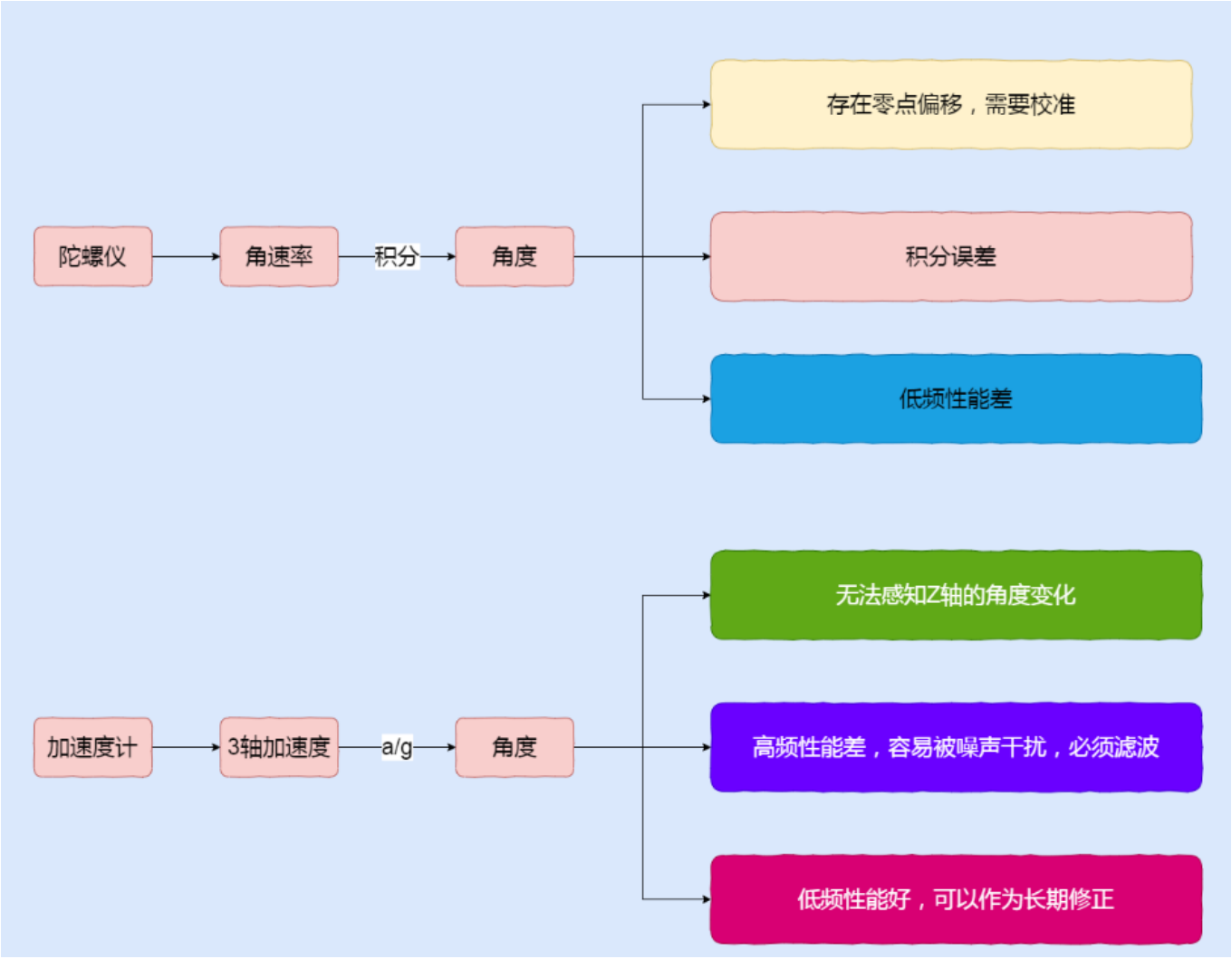


# 四元数

> 四元数的微分（四元数微分方程的推导和解算实现 - 知乎 (zhihu.com)）

# 老陀螺仪代码

## 传感器特性




```
        //计算加速度计测量角度
        pure_acc_pitch =  atan2((double)icm_accel[0], (double)icm_accel[2]) / 3.1415926f * 180.0f;//基于底盘坐标系的pitch轴
//      pure_acc_pitch = -atan2((double)icm_accel[0], sqrt(icm_accel[2]*icm_accel[2]+icm_accel[1]*icm_accel[1])) / 3.1415926f * 180.0f;//基于世界坐标系的pitch轴
        pure_acc_roll  =  -atan2((double)icm_accel[1], (double)icm_accel[2]) / 3.1415926f * 180.0f;
```

直接将加速度分解，用反三角函数直接算出pitch角度.

```
    pitch_w_cal =  /*y*/(icm_gyro[1]);//基于底盘坐标系的pitch轴
    roll_w_cal  = /*x->roll*/icm_gyro[0]*cos(combAngle_pitch*3.1415926f/180.0f)-icm_gyro[2]*sin(combAngle_pitch*3.1415926f/180.0f)*cos(combAngle_roll*3.1415926f/180.0f)+\
    icm_gyro[1]*sin(combAngle_pitch*3.1415926f/180.0f)*sin(combAngle_roll*3.1415926f/180.0f);

    yaw_w_cal= (icm_gyro[2]) *cos(combAngle_pitch*3.1415926f/180.0f)* cos(combAngle_roll*3.1415926f/180.0f) + (icm_gyro[0]) * (sin(combAngle_pitch*3.1415926f/180.0f)) - \
               (icm_gyro[1]) * sin(combAngle_roll*3.1415926f/180.0f)*cos(combAngle_pitch*3.1415926f/180.0f);
```

欧拉角对应的角速度产生的时候也要有顺序。

$$Y_1 Z_2 X_3 = \begin{bmatrix} c_1 c_2 & s_1 s_3 - c_1 c_3 s_2 & c_3 s_1 + c_1 s_2 s_3 \\ s_2 & c_2 c_3 & -c_2 s_3 \\ -c_2 s_1 & c_1 s_3 + c_3 s_1 s_2 & c_1 c_3 - s_1 s_2 s_3 \end{bmatrix}$$

跟这张图一样先转p。

```
        pitch_w_cal =  /*y*/(icm_gyro[1]);//基于底盘坐标系的pitch轴
        roll_w_cal  =
icm_gyro[0]*cos(combAngle_pitch*3.1415926f/180.0f)-
icm_gyro[2]*sin(combAngle_pitch*3.1415926f/180.0f)*cos(combAngle_roll*
3.1415926f/180.0f)+icm_gyro[1]*sin(combAngle_pitch*3.1415926f/180.0f)*
sin(combAngle_roll*3.1415926f/180.0f);

        yaw_w_cal= (icm_gyro[2])
*cos(combAngle_pitch*3.1415926f/180.0f)*
cos(combAngle_roll*3.1415926f/180.0f) + (icm_gyro[0]) *
(sin(combAngle_pitch*3.1415926f/180.0f)) -
(icm_gyro[1])*sin(combAngle_roll*3.1415926f/180.0f)*cos(combAngle_pitc
h*3.1415926f/180.0f);
```

```
k_filter_pitch_roll = -k_com*fabsf(acc_norm-acc_inti_norm ) +0.02f ;//+ k_com_w*fabsf(pitch_w_acc);//自适应参数
k_filter_pitch_roll = LIMIT_MAX_MIN(0.02f,0.0f,k_filter_pitch_roll);
//fabsf(acc_norm-acc_inti_norm)越大，说明加速度越大，则k_filter_pitch_roll越小，combAngle_pitch越趋向于使用陀螺仪数据
//反之取静态量
combAngle_pitch = (combAngle_pitch +pitch_w_cal*T/3.1415926f * 180.0f)*(1 - k_filter_pitch_roll) + pure_acc_pitch*k_filter_pitch_roll;
combAngle_roll  = (combAngle_roll  +roll_w_cal*T/3.1415926f * 180.0f)*(1 - k_filter_pitch_roll) + pure_acc_roll *k_filter_pitch_roll;
```

其实老版的IMU代码和网上常见的网上常见的互补滤波算法的写法和表达不太一样，但是核心思路是一样的，都是传感器的数据融合，在低加速度的情况之下更加相信加速度计的数据，在高加速度、高速等情况下，更加相信三轴陀螺仪。

# 倍哥代码学习

```
/*   F, number with * represent vals to be set
 0      1*     2*     3*     4      5
 6*     7      8*     9*    10     11
12*    13*    14     15*    16     17
18*    19*    20*    21     22     23
24     25     26     27     28     29
30     31     32     33     34     35
*/
QEKF_INS.dt = dt;

QEKF_INS.Gyro[0] = gx - QEKF_INS.GyroBias[0];
QEKF_INS.Gyro[1] = gy - QEKF_INS.GyroBias[1];
QEKF_INS.Gyro[2] = gz - QEKF_INS.GyroBias[2];

// set F
halfgxdt = 0.5f * QEKF_INS.Gyro[0] * dt;
halfgydt = 0.5f * QEKF_INS.Gyro[1] * dt;
halfgzdt = 0.5f * QEKF_INS.Gyro[2] * dt;

// 此部分设定状态转移矩阵F的左上角部分 4x4子矩阵,即0.5(Ohm-Ohm^bias)*deltaT,右下角有一个2x2单位阵已经初始
// 注意在predict步F的右上角是4x2的零矩阵,因此每次predict的时候都会调用memcpy用单位阵覆盖前一轮线性化后的
memcpy(QEKF_INS.IMU_QuaternionEKF.F_data, IMU_QuaternionEKF_F, sizeof(IMU_QuaternionEKF_F));

QEKF_INS.IMU_QuaternionEKF.F_data[1] = -halfgxdt;
QEKF_INS.IMU_QuaternionEKF.F_data[2] = -halfgydt;
QEKF_INS.IMU_QuaternionEKF.F_data[3] = -halfgzdt;

QEKF_INS.IMU_QuaternionEKF.F_data[6] = halfgxdt;
QEKF_INS.IMU_QuaternionEKF.F_data[8] = halfgzdt;
QEKF_INS.IMU_QuaternionEKF.F_data[9] = -halfgydt;

QEKF_INS.IMU_QuaternionEKF.F_data[12] = halfgydt;
QEKF_INS.IMU_QuaternionEKF.F_data[13] = -halfgzdt;
QEKF_INS.IMU_QuaternionEKF.F_data[15] = halfgxdt;

QEKF_INS.IMU_QuaternionEKF.F_data[18] = halfgzdt;
QEKF_INS.IMU_QuaternionEKF.F_data[19] = halfgydt;
QEKF_INS.IMU_QuaternionEKF.F_data[20] = -halfgxdt;
```

四元数的微分（[四元数微分方程的推导和解算实现 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/45404840)）

$$
\begin{aligned}
\boldsymbol{F}_k &= \frac{\partial f(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} \\
&= \begin{bmatrix} \boldsymbol{I}_4 + \frac{1}{2}\left(\boldsymbol{\Omega}_k - \boldsymbol{\Omega}_k^{bias}\right)\Delta t & \boldsymbol{O}_k \\ \boldsymbol{0}_{2\times 4} & \boldsymbol{I}_2 \end{bmatrix}
\end{aligned}
$$

**注意**：此处的·表示四元数乘法；$w$为角速度的纯四元数表示，$w = \begin{bmatrix} 0 & w_x & w_y & w_z \end{bmatrix}^T$；

由于实际工程中我们都是通过固连在机体上的陀螺仪等传感器来获知机体角速度

$$w^b$$

它与地理坐标系下的角速度表示有如下关系

$$w = q(t) w^b q(t)^{-1}$$

带入上式即可得到姿态解算过程中常用的四元数的微分形式

$$\dot{q}(t) = \frac{1}{2} q(t) w^b$$

可以看出，通过一次四元数乘法运算便可得到四元数的微分。
上式可以写成如下的矩阵形式：

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

或者：

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

```
// set Q R,过程噪声和观测噪声矩阵
QEKF_INS.IMU_QuaternionEKF.Q_data[0] = QEKF_INS.Q1 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.Q_data[7] = QEKF_INS.Q1 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.Q_data[14] = QEKF_INS.Q1 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.Q_data[21] = QEKF_INS.Q1 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.Q_data[28] = QEKF_INS.Q2 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.Q_data[35] = QEKF_INS.Q2 * QEKF_INS.dt;
QEKF_INS.IMU_QuaternionEKF.R_data[0] = QEKF_INS.R;
QEKF_INS.IMU_QuaternionEKF.R_data[4] = QEKF_INS.R;
QEKF_INS.IMU_QuaternionEKF.R_data[8] = QEKF_INS.R;
```

**给举证对角线赋值**

## 2.4 量测模型

根据 $\boldsymbol{g} = \boldsymbol{C}_n^b [0 \quad 0 \quad 1]^T$，在无运动加速度情况下有：

旋转矩阵的使用，将对应的加速的转换到z轴上面就是重力加速度了

坐标变换矩阵$\boldsymbol{C}_b^n$：

$$\boldsymbol{C}_b^n = \begin{bmatrix} 1-2\left(q_2^2+q_3^2\right) & 2\left(q_1q_2-q_0q_3\right) & 2\left(q_1q_3+q_0q_2\right) \\ 2\left(q_1q_2+q_0q_3\right) & 1-2\left(q_1^2+q_3^2\right) & 2\left(q_2q_3-q_0q_1\right) \\ 2\left(q_1q_3-q_0q_2\right) & 2\left(q_2q_3+q_0q_1\right) & 1-2\left(q_1^2+q_2^2\right) \end{bmatrix}$$

## atan(y/x)函数

atan(**y/x**)函数用以求取y/x的反正切（注意atan只有一个输入参数），返回值的单位为弧度，返回值的取值范围为 $[-\pi/2, \pi/2]$ 。

- 当y=x=0时返回值为NaN;
- 当y/x为0时返回值为0;
- 当y/x为正时返回值在 $(0, \pi/2]$ 之间;
- 当y/x为负时返回值在 $[-\pi/2, 0)$ 之间;

## atan2(y,x)函数

atan2(**y,x**)函数用以求取y/x的反正切（注意atan2有两个输入参数），返回值的单位为弧度，返回值的取值范围为 $(-\pi, \pi]$ 。

- 当y=x=0时返回值为0;
- 当y=0,x>0时返回值为0;
- 当x>0且y>0时返回值在 $(0, \pi/2)$ 之间;
- 当x=0且y>0时返回值为 $\pi/2$ ;
- 当x<0且y>0时返回值在 $(\pi/2, \pi)$ 之间;
- 当y=0,x<0时返回值为 $\pi$ ;
- 当x<0且y<0时返回值在 $(-\pi, -\pi/2)$ 之间;
- 当x=0且y<0时返回值为 $-\pi/2$ ;
- 当x>0且y<0时返回值在 $(-\pi/2, 0)$ 之间;