四轴飞行器报告(高级篇)

姓　　名: 阿力木江　艾合买提江　高瞻

完成日期: 2014 年 12 月 29 日 星期一

报告内容

1. 姿态解算用到的常用数学方法和处理手段
2. 自动控制原理 PID 和系统建模

姿态解算用到的常用数学方法和处理手段

姿态的数学表示方法

姿态有多种数学表示方式，常见的是四元数，欧拉角，矩阵和轴角。他们各自有其自身的优点，在不同的领域使用不同的表示方式。在四轴飞行器中使用到了**四元数**和**欧拉角**。

四元数

四元数是由爱尔兰数学家威廉·卢云·哈密顿在 1843 年发现的数学概念。从明确地角度而言，四元数是复数的不可交换延伸。如把四元数的集合考虑成多维实数空间的话，四元数就代表着一个四维空间，相对于复数为二维空间。

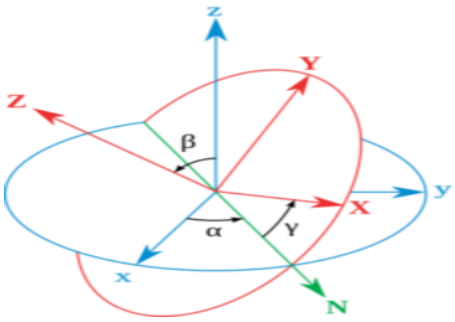
四元数大量用于电脑绘图（及相关的图像分析）上表示三维物件的旋转及方位。四元数亦见于控制论、信号处理、姿态控制、物理和轨道力学，都是用来表示旋转和方位。

相对于另几种旋转表示法（矩阵，欧拉角，轴角），四元数具有某些方面的优势，如速度更快、提供平滑插值、有效避免万向锁问题、存储空间较小等等。

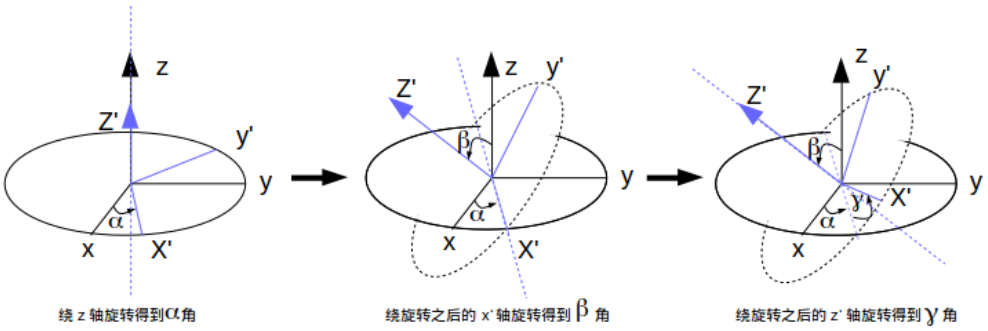
以上部分摘自维基百科-四元数。

欧拉角

莱昂哈德·欧拉用欧拉角来描述刚体在三维欧几里得空间的取向。对于在三维空间里的一个参考系，任何坐标系的取向，都可以用三个欧拉角来表现。参考系又称为实验室参考系，是静止不动的。而坐标系则固定于刚体，随着刚体的旋转而旋转。



以上部分摘自维基百科-欧拉角。下面我们通过图例来看看欧拉角是如何产生的，并且分别对应哪个角度。



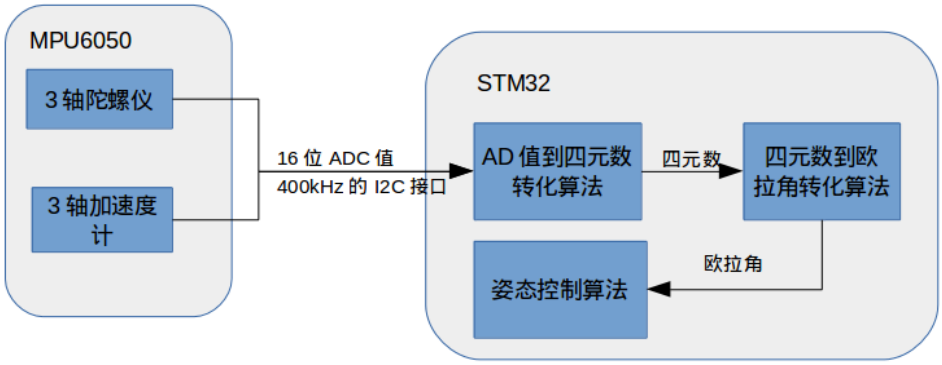
姿态解算为什么要用四元数和欧拉角

姿态解算的核心在于旋转，一般旋转有 4 种表示方式：矩阵表示、欧拉角表示、轴角表示和四元数表示。矩阵表示适合变换向量，欧拉角最直观，轴角表示则适合几何推导，而在组合旋转方面，四元数表示最佳。因为姿态解算需要频繁组合旋转和用旋转变换向量，所以采用四元数保存组合姿态、辅以矩阵来变换向量的方案。

总结来说，在飞行器中，姿态解算中使用四元数来保存飞行器的姿态，包括旋转和方位。在获得四元数之后，会将其转化为欧拉角，然后输入到姿态控制算法中。

姿态控制算法的输入参数必须要是欧拉角。AD 值是指 MPU6050 的陀螺仪和加速度值，3 个维度的陀螺仪值和 3 个维度的加速度值，每个值为 16 位精度。AD 值必须先转化为四元数，然后通过四元数转化为欧拉角。这个四元数可能是软解，主控芯片（STM32）读取到 AD 值，用软件从 AD 值算得，也可能是通过 MPU6050 中的 DMP 硬解，主控芯片（STM32）直接读取到四元数。具体参考《MPU60x0 的四元数生成方式介绍》。

下面就是四元数软解过程，可以由下面这个框图表示：



扩展阅读-四元数的运算

下面介绍一下四元数，然后给出几种旋转表示的转换。这些运算在飞行器的代码中都会遇到。

四元数可以理解为一个实数和一个向量的组合，也可以理解为四维的向量。这里用一个圈表示 q 是一个四元数（很可能不是规范的表示方式）。

$$\mathring{q}=\{w,\vec{v}\}=[w\ \ x\ \ y\ \ z]^T$$

四元数的长度（模）与普通向量相似。

$$|\mathring{q}|=\sqrt{w^2+x^2+y^2+z^2}$$

下面是对四元数的单位化，单位化的四元数可以表示一个旋转。

$$\hat{\mathring{q}}=\frac{\mathring{q}}{|\mathring{q}|}=\left[\frac{w}{|\mathring{q}|}\ \ \frac{x}{|\mathring{q}|}\ \ \frac{y}{|\mathring{q}|}\ \ \frac{z}{|\mathring{q}|}\right]^T$$

四元数相乘，旋转的组合就靠它了。

$$\mathring{q}_0=\mathring{q}_1\cdot\mathring{q}_2$$

$$\begin{cases}w_0=w_1w_2-x_1x_2-y_1y_2-z_1z_2\\x_0=w_1x_2+x_1w_2+y_1z_2-z_1y_2\\y_0=w_1y_2-x_1z_2+y_1w_2+z_1x_2\\z_0=w_1z_2+x_1y_2-y_1x_2+z_1w_2\end{cases}$$

旋转的“轴角表示”转“四元数表示”。这里创造一个运算 q(w,θ)，用于把绕单位向量w 转 θ 角的旋转表示为四元数。

$$\{\hat{w},\theta\}\rightarrow\mathring{\mathrm{q}}(\hat{w},\theta)=\{\cos(\frac{\theta}{2}),\hat{w}\cdot\sin(\frac{\theta}{2})\}$$

通过 q(w,θ)，引伸出一个更方便的运算 q(f,t)。有时需要把向量 f 的方向转到向量 t 的方向，这个运算就是生成表示对应旋转的四元数的（后面会用到）。

$$\mathring{q}(\vec{f},\vec{t})=\mathring{q}\left(\frac{\vec{f}\times\vec{t}}{|\vec{f}\times\vec{t}|},\operatorname{atan2}(|\vec{f}\times\vec{t}|,\vec{f}\cdot\vec{t})\right)$$

然后是“四元数表示”转“矩阵表示”。再次创造运算，用 R(q)表示四元数 q 对应的矩阵（后面用到）。

$$\mathring{q}\to R(\mathring{q})=\begin{bmatrix}1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy\\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx\\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2\end{bmatrix}$$

多个旋转的组合可以用四元数的乘法来实现。

$$R(\mathring{q}_0)\cdot R(\mathring{q}_1)=R(\mathring{q}_0\cdot\mathring{q}_1)$$

“四元数表示”转“欧拉角表示”。用于显示。

$$\mathring{q}\to\{\psi,\theta,\varphi\}$$

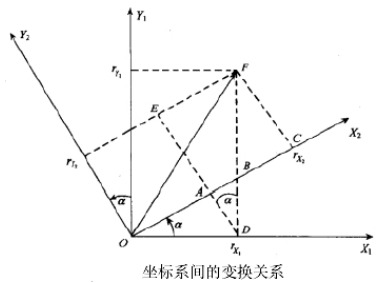
$$\begin{cases}\psi=\operatorname{atan2}(2wz+2xy,1-2y^2-2z^2)\\ \theta=\arcsin(2wy-2zx)\\ \varphi=\operatorname{atan2}(2wx+2yz,1-2x^2-2y^2)\end{cases}$$

$$\begin{cases}\psi & : \text{偏航角(yaw),z轴}\\ \theta & : \text{俯仰角(pitch),y轴}\\ \varphi & : \text{滚转角(roll),x轴}\end{cases}$$

软件姿态解算

使用 MPU6050 硬件 DMP 解算姿态是非常简单的，下面介绍由三轴陀螺仪和加速度计的值来使用四元数软件解算姿态的方法。

我们先来看看如何用欧拉角描述一次平面旋转(坐标变换)：



坐标系间的变换关系

设坐标系绕旋转 α 角后得到坐标系,在空间中有一个矢量在坐标系中的投影为,在内的投影为由于旋转绕进行，所以 Z 坐标未变，即有。

$$
\begin{aligned}
r_{x2} &= OA + AB + BC \\
&= OD\cos\alpha + BD\sin\alpha + BF\sin\alpha \\
&= r_{x1}\cos\alpha + r_{y1}\sin\alpha \\
r_{y2} &= DE - AD \\
&= DF\cos\alpha - OD\sin\alpha \\
&= r_{y1}\cos\alpha - OD\sin\alpha \\
&= r_{y1}\cos\alpha - r_{x1}\sin\alpha \\
r_{z2} &= r_{z1}
\end{aligned}
$$

转换成矩阵形式表示为：

$$
\begin{bmatrix} r_{x2} \\ r_{y1} \\ r_{z1} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix}
$$

整理一下：

$$
r^1 = \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix} \qquad r^2 = \begin{bmatrix} r_{x2} \\ r_{y2} \\ r_{z2} \end{bmatrix} \qquad \text{旋转阵 } C_1^2 = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

所以从旋转到可以写成

上面仅仅是绕一根轴的旋转，如果三维空间中的欧拉角旋转要转三次:

$$
O-X_nY_nZ_n \xrightarrow{\text{绕-}Z_n\text{轴旋转}\,\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\,\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\,\gamma} O-X_bY_bZ_b
$$

$$
\begin{aligned}
C_n^b = C_2^b C_1^2 C_n^1 &= \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\gamma\cos\psi + \sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi + \sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi - \cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi - \cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}
\end{aligned}
$$

上面得到了一个表示旋转的方向余弦矩阵。

不过要想用欧拉角解算姿态，其实我们套用欧拉角微分方程就行了：

$$\begin{bmatrix} \dot{\gamma} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos\theta} \begin{bmatrix} \cos\theta & \sin\gamma\sin\theta & \cos\gamma\sin\theta \\ 0 & \cos\theta\cos\gamma & -\sin\gamma\cos\theta \\ 0 & \sin\gamma & \cos\gamma\cos\theta \end{bmatrix}^{-1} \bullet \begin{bmatrix} \omega_{EbX}^{b} \\ \omega_{EbY}^{b} \\ \omega_{EbZ}^{b} \end{bmatrix}$$

上式中左侧,,是本次更新后的欧拉角,对应 row,pit,yaw。右侧，是上个周期测算出来的角度，，，三个角速度由直接安装在四轴飞行器的三轴陀螺仪在这个周期转动的角度，单位为弧度，计算间隔时 T *陀螺角速度，比如0.02 秒*0.01 弧度/秒=0.0002 弧度。间因此求解这个微分方程就能解算出当前的欧拉角。

前面介绍了什么是欧拉角，而且欧拉角微分方程解算姿态关系简单明了，概念直观容易理解，那么我们为什么不用欧拉角来表示旋转而要引入四元数呢？

一方面是因为欧拉角微分方程中包含了大量的三角运算，这给实时解算带来了一定的困难。而且当俯仰角为 90 度时方程式会出现神奇的“GimbalLock”。所以欧拉角方法只适用于水平姿态变化不大的情况，而不适用于全姿态飞行器的姿态确定。

四元数法只求解四个未知量的线性微分方程组，计算量小，易于操作，是比较实用的工程方法。

我们知道在平面(x,y)中的旋转可以用复数来表示，同样的三维中的旋转可以用单位四元数来描述。我们来定义一个四元数：

$$\mathbf{q} = a + \overrightarrow{u} = q_0 + q_1 i + q_2 j + q_3 k$$

我们可以把它写成,其中,。那么是矢量，表示三维空间中的旋转轴。w 是标量，表示旋转角度。那么就是绕轴旋转 w 度，所以一个四元数可以表示一个完整的旋转。只有单位四元数才可以表示旋转，至于为什么，因为这就是四元数表示旋转的约束条件。

而刚才用欧拉角描述的方向余弦矩阵用四元数描述则为：

$$C_n^b = \begin{bmatrix} q_1^2 + q_0^2 - q_3^2 - q_2^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_2^2 - q_3^2 + q_0^2 - q_1^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_3^2 - q_2^2 - q_1^2 + q_0^2 \end{bmatrix}$$

所以在软件解算中，我们要首先把加速度计采集到的值(三维向量)转化为单位向量,即向量除以模，传入参数是陀螺仪 x,y,z 值和加速度计 x,y,z 值：

```
void IMUupdate(float gx, float gy, float gz, float ax, float ay, float az) {

float norm;

float vx, vy, vz;
```

```
float ex, ey, ez;



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

ax = ax / norm;

ay = ay / norm;

az = az / norm;
```

下面把四元数换算成方向余弦中的第三行的三个元素。刚好 vx,vy,vz 其实就是上一次的欧拉角（四元数）的机体坐标参考系换算出来的重力的单位向量。

```
// estimated direction of gravity

vx = 2*(q1*q3 - q0*q2);

vy = 2*(q0*q1 + q2*q3);

vz = q0*q0 - q1*q1 - q2*q2 + q3*q3;
```

axyz 是机体坐标参照系上，加速度计测出来的重力向量，也就是实际测出来的重力向量。

axyz 是测量得到的重力向量，vxyz 是陀螺积分后的姿态来推算出的重力向量，它们都是机体坐标参照系上的重力向量。

那它们之间的误差向量，就是陀螺积分后的姿态和加计测出来的姿态之间的误差。

向量间的误差，可以用向量叉积（也叫向量外积、叉乘）来表示，exyz 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。（你可以自己拿东西想象一下）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。

```
// integral error scaled integral gain

exInt = exInt + ex*Ki;

eyInt = eyInt + ey*Ki;

ezInt = ezInt + ez*Ki;
```

用叉积误差来做 PI 修正陀螺零偏

```
// integral error scaled integral gain

exInt = exInt + ex*Ki;
```

```
eyInt = eyInt + ey*Ki;

ezInt = ezInt + ez*Ki;



// adjusted gyroscope measurements

gx = gx + Kp*ex + exInt;

gy = gy + Kp*ey + eyInt;

gz = gz + Kp*ez + ezInt;
```

四元数微分方程，其中 T 为测量周期，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：

$$O-X_nY_nZ_n \xrightarrow{\text{绕}-Z_n\text{轴旋转}\,\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\,\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\,\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^b C_1^2 C_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma\cos\psi+\sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi+\sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi-\cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi-\cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}$$

```
// integrate quaternion rate and normalise

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;
```

最后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：

$$O-X_nY_nZ_n \xrightarrow{\text{绕}-Z_n\text{轴旋转}\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^b C_1^2 C_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma\cos\psi+\sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi+\sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi-\cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi-\cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}$$

所以有：

```
Q_ANGLE.Yaw = atan2(2 * q1 * q2 + 2 * q0 * q3, -2 * q2*q2 - 2 * q3* q3 + 1)* 57.3; // yaw

Q_ANGLE.Y  = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; // pitch

Q_ANGLE.X = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; // roll
```

硬件姿态解算

四轴的姿态解算无疑是最繁琐的步骤没有之一，但是自从 MPU6050 出现了硬件 DMP 的时候，大妈都能完成姿态解算了！

飞行器使用了 MPU6050 自带的硬件四元数单元，可以通过 IIC 直接读取四元数，省却了软件解算繁琐的算法步骤，非常方便易用。

这里还是要首先介绍下四元数，四元数要说的实在太多，因为它的优点很多，利用起来很方便，但是理解起来就有点蹩脚了。我们百度四元数，一开始看到的就是四元数来历，还有就是四元数的基本计算。对于来历，还是想说一下，四元数（Quaternions）是由威廉·卢云·哈密尔顿(William Row anHamilton,1805-1865）在 1843 年爱尔兰发现的数学概念。

将实数域扩充到复数域，并用复数来表示平面向量，用复数的加、乘运算表示平面向量的合成、伸缩和旋，这就是我们熟知的复数的二维空间含义，所以人们会继续猜想，利用三维复数不就可以表达三维空间的变换了吗，历史上有很多数学家试图寻找过三维的复数，但后来证明这样的三维复数是不存在的。有关这个结论的证明，我没有查到更明确的版本，据《古今数学思想》中的一个理由，三维空间中的伸缩旋转变换需要四个变量来决定：两个变量决定轴的方向，一个变量决定旋转角度，一个变量决定伸缩比例。这样，只有三个变量的三维复数无法满足这样的要求。但是历史上得到的应该是比这个更强的结论，即使不考虑空间旋转，只从代数角度来说，三维的复数域作为普通复数域的扩张域是不存在的。并且，据《古今数学思想》叙述，即使像哈密尔顿后来引入四元数那样，牺牲乘法交换律，这样的三维复数也得不到。经过一些年的努力之后， Hamilton 发现自己被迫应作两个让步，第一个是他的新数包含四个分量，而第二个是他必须牺牲乘法交换律。（《古今数学思想》第三册 177 页）但是四元数用作旋转的作用明显，简化了运算，而且避免了 Gimbal Lock，四元数是最简单的超复数，我们不能把四元数简单的理解为 3D 空间的矢量，它是 4 维空间中的的矢量，也是非常不容易想像的。

我们知道在平面(x,y)中的旋转可以用复数来表示，同样的三维中的旋转可以用单位四元数来描述。我们来定义一个四元数：

$$\mathbf{q} = a + \overrightarrow{u} = q_0 + q_1 i + q_2 j + q_3 k$$

我们可以把它写成$[w, \mathbf{v}]$，其中$\mathbf{v} = q_1 i + q_2 j + q_3 k$，$w = a = q_0$。那么 V 是矢量，表示三维空间中的旋转轴。w 是标量，表示旋转角度。那么$[w, \mathbf{v}]$就是绕轴 V 旋转 w 度，所以一个四元数可以表示一个完整的旋转。只有单位四元数才可以表示旋转，至于为什么，因为这就是四元数表示旋转的约束条件。

所以大家可以理解为，单位四元数能够表示旋转。我们给出一组单位四元数和欧拉角的转换关系，通过这个关系来将采集到的四元数转化成欧拉角，原理将在软件解算中给出，这里直接使用就可以了：

$$\theta = -\sin^{-1} 2*(q_1 q_2 - q_0 q_3)$$
$$\psi = tg^{-1}\left(\frac{2(q_1 q_2 + q_0 q_3)}{q_1^2 + q_0^2 - q_3^2 - q_2^2}\right)$$
$$\gamma = tg^{-1}\left(\frac{2(q_1 q_3 + q_0 q_1)}{q_3^2 + q_2^2 - q_1^2 + q_0^2}\right)$$

所以在四轴飞行器中，传感器读取到四元数，首先应先将它归一化成单位四元数：

```
norm = dmpinvSqrt(q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3]);

q[0] = q[0] * norm;

q[1] = q[1] * norm;

q[2] = q[2] * norm;

q[3] = q[3] * norm;
```

归一化后根据四元数和欧拉角转换公式把四元数转化为欧拉角，OK,硬件姿态解算完成！

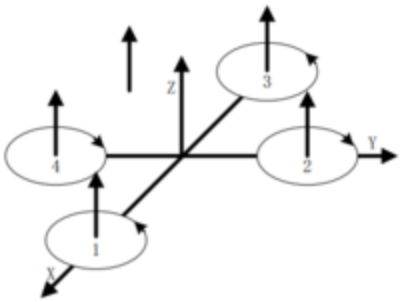
```
DMP_DATA.dmp_roll = (atan2(2.0*(q[0]*q[1] + q[2]*q[3]), 1 - 2.0*(q[1]*q[1] + q[2]*q[2])))* 180/
M_PI;

// we let safe_asin() handle the singularities near 90/-90 in pitch

DMP_DATA.dmp_pitch = dmpsafe_asin(2.0*(q[0]*q[2] - q[3]*q[1]))* 180/M_PI;



DMP_DATA.dmp_yaw = -atan2(2.0*(q[0]*q[3] + q[1]*q[2]), 1 - 2.0*(q[2]*q[2] + q[3]*q[3]))* 180/M_
PI;
```

PID 控制算法

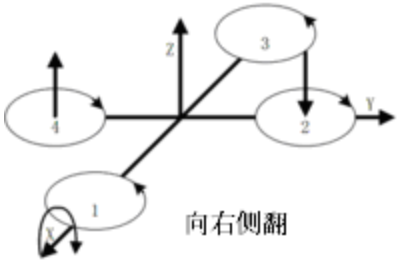
四轴如何起飞的原理

四轴飞行器的螺旋桨与空气发生相对运动，产生了向上的升力，当升力大于四轴的重力时四轴就可以起飞了。

四轴飞行器飞行过程中如何保持水平：由于四个电机转向相同，四轴会发生旋转。我们控制四轴电机 1 和电机 3 同向，电机 2 电机 4 反向，刚好抵消反扭矩，巧妙的实现了平衡：

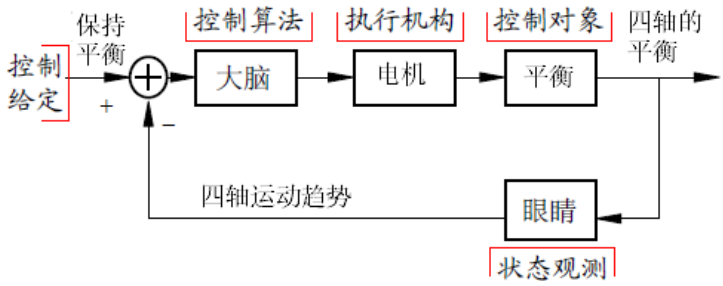


实际上由于电机和螺旋桨本身制造的差异我们无法做到四个电机转速完全相同，如果我们控制同样的转速很有可能飞行器起飞之后就侧翻了。



由于电机的不平衡，在人眼的观察下发现飞机向右侧翻，我们控制右侧电机 1 电机 2 提高转速增加升力，飞机归于平衡。由于 飞机是一个动态系统，在接下来我们会一直重复：

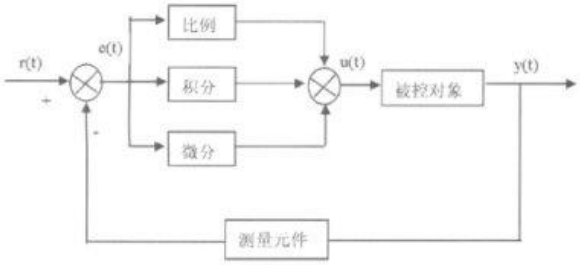
观察->大脑计算->控制->观察->大脑计算->控制 这个过程。



但事实上这是不可能的，因为人无法长时间精确的同时控制四个电机。我们需要一个自动反馈系统替代人操作来完成飞机的自稳定，我们人只需要控制飞机的方向和高度就可以了。这个系统中反馈由姿态传感器替代眼睛，而大脑则由单片机来替代。这时候该运用 PID 控制系统

什么是 PID?

PID 控制器由偏差的比例（P）、积分（I）和微分（D）来对被控对象进行控制，是应用最为广泛的一种自动控制器。



PID控制原理图

- 比例（P）控制

比例控制是一种最简单的控制方式。其控制器的输出与输入误差信号成比例关系。当仅有比例控制时系统输出存在稳态误差（Steady-state error）。

- 积分（I）控制

在积分控制中，控制器的输出与输入误差信号的积分成正比关系。对一个自动控制系统，如果在进入稳态后存在稳态误差，则称这个控制系统是有稳态误差的 或简称有差系统（System w ith Steady-state Error）。为了消除稳态误差，在控制器中必须引入“积分项”。积分项对误差取决于时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积 分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到等于零。因此，比例+积分(PI)控制器，可以使系统在进入稳态后无稳 态误差。 积分项输出：

$$y=\frac{1}{T_I}\int e(t)dt \qquad (4\text{-}2)$$

- 微分（D）控制

微分调节就是偏差值的变化率。使用微分环节能够实现系统的超前控制。如果输入偏差值线性变化，则在调节器输出侧叠加一个恒定的调节量。大部分控制系统不需要调节微分时间。因为只有时间滞后的系统才需要附加这个参数。如果画蛇添足加上这个参数反而会使系统的控制受到影响。微分项输出：

$$y=T_D\frac{de(t)}{dt} \qquad (4\text{-}4)$$

综上所述得到一个一条公式，这个就是模拟 PID:

$$y = K_p\left[e(t) + \frac{1}{T_I}\int e(t)dt + T_D\frac{de(t)}{dt}\right]$$

而 PID 中又可以只使用 PI 项构成比例-积分控制器，使用 PD 项构成比例-微分控制器。在飞行器四轴飞行器中我们使用了增量式 PD 控制，以 ROLL 方向角度控制为例：

- 测得 ROLL 轴向偏差：

```
偏差=目标期望角度-传感器实测角度

DIF_ANGLE.X = EXP_ANGLE.X - Q_ANGLE.Roll;
```

- 比例项的计算：

```
比例项输出 = 比例系数 P * 偏差

Proportion =  PID_Motor.P * DIF_ANGLE.X;
```

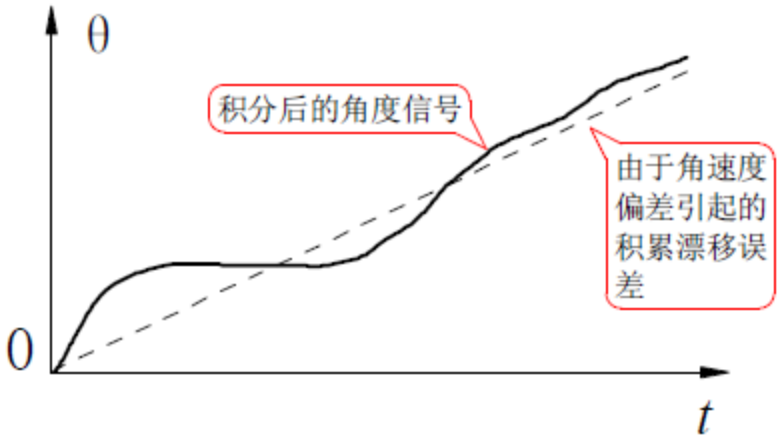
- 微分项计算： 由于陀螺仪测得的是 ROLL 轴向旋转角速率，角速率积分就是角度，那么角度微分即角速率，所以微分量刚好是陀螺仪测得的值。

```
微分输出=微分系数 D*角速率

DifferentialCoefficient = PID_Motor.D * DMP_DATA.GYROx;
```

- 整合结果总输出为：

```
ROLL 方向总控制量=比例项输出+微分量输出
```

ROLL 和 PIT 轴向按照以上公式计算 PID 输出，但 YAW 轴比较特殊，因为偏航角法线方向刚好和地球重力平行，这个方向的角度无法由加速度计直接测得，需要增加一个电子罗盘来替代加速度计。如果不使用罗盘的话，我们可以单纯的通过角速度积分来测得偏航角，缺点是由于积分环节中存在积分漂移，偏航角随着时间的推移会偏差越来越大。我们不使用罗盘就没有比例项，只仅使用微分环节来控制。

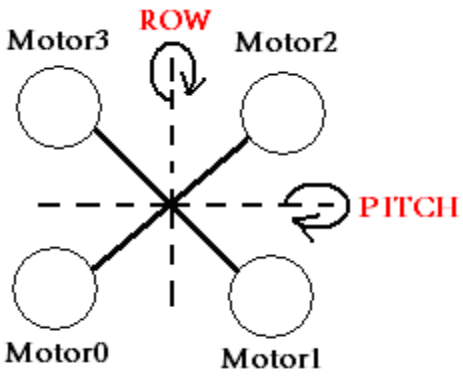


角速度积分漂移现象

- YAW 轴输出：

```
微分输出=微分系数 D*角速率


YAW 方向控制量 = PID_YAW.D * DMP_DATA.GYROz;
```

- 电机的输出： 油门控制 Throttle 是电机输出的基准值，增加油门即可提高四轴高度。最后整合 ROLL/PIT/YAW 三轴输出量进行电机控制：




```
Motor[2] = (int16_t)(Thr - Pitch - Roll - Yaw );    //M3

Motor[0] = (int16_t)(Thr + Pitch + Roll - Yaw );    //M1

Motor[3] = (int16_t)(Thr - Pitch + Roll + Yaw );    //M4

Motor[1] = (int16_t)(Thr + Pitch - Roll + Yaw );    //M2
```

如图四轴绕 ROW 轴向右下倾斜 5 度，那么电机 1 电机 2 应该提高升力，电机 3 电机 0 减小升力恢复平衡状态，所以有以下规则：

- Roll 方向旋转，则电机 1 电机 2 同侧出力，电机 0 电机 3 反向出力
- Pitch 方向旋转，则电机 2 电机 3 同侧出力，电机 0 电机 1 反向出力
- Yaw 方向旋转，则电机 1 电机 3 同侧出力，电机 0 电机 2 反向出力

参考:

软件架构部分:小马论坛,匿名论坛, 圆点博士论坛,crazyfile 论坛

控制 PID 部分:中山大学四轴俱乐部,crazyfile 论坛

硬件参考:众多大神匿名提供的资料和解决方案,还有许多热心帮助的爱好者.

参考书籍:《实时操作系统 CSD 方法》《UC/OS-Ⅲ 基于 STM32 移植》《由入门到精通吃透 PID》《高频电子线路设计》

参考文献：《基于非线性滤波的初始对准研究》《卡尔曼滤波器介绍》《捷联式惯性导航系统》《四旋翼飞行器的动力学建模及 pid 控制》《四旋翼几种控制方法研究》