姿态解算理解

1、姿态的描述方法

前几天在论坛里偶尔看到一个帖子，帖子的内容是问的为什么不用倾斜角表示姿态，我认为他说的倾斜角是指的斜面与斜面的夹角，或者说是物体与垂线的夹角吧，这种想法可能来源于我们日常生活的思维。

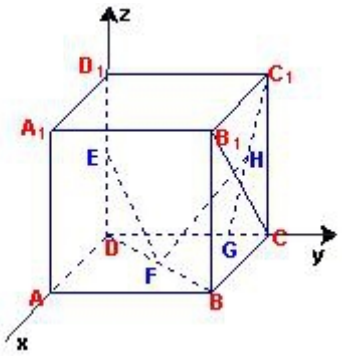


图 1　立方体

比如有一个立方体，我们放在水平面上的时候它的底面是和水平面平行的，但是当我们把立方体的一个脚垫起一个角度时，这样一来，立方体的一条棱与水平面的垂线就有了一定的夹角了。我们所说倾斜了多少多少度就是指的这个夹角，这是我们直观的反应。我认为这样直观的反应甚至比欧拉角还要来的直观，因为欧拉角是基于旋转的，肯定不会说这个立方体 X、Y 轴各旋转了多少度（假设 Z 旋转无效），我们可能也没那个概念，我们直观的反应就是它倾斜了一定度数。

但是我们在姿态解算的时候为啥不用的这种描述方法呢，个人认为是虽然方便我们直观的表达但不适合数学上的计算，还有就是我们仅仅知道这个倾斜角我们怎么施加控制量呢？高中物理学习物体运动的时候我们知道，物体的运动是合运动，我们可以把它的运动矢量正交分解为几个运动的合成（不正交也是可以的，但那不是在自找麻烦吗），同样道理，我们可以把刚体的旋转分解为三个轴上的旋转，这个旋转的角度就是欧拉角，如图 2。

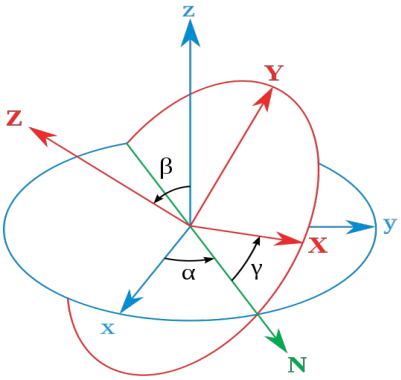


图 2　zxz 序规欧拉角

欧拉角

欧拉角的定义不仅仅和旋转角度有关系，还和旋转轴的旋转顺序有关系，任何一种旋转顺序都是合法的。根据定义，欧拉角有 12 种旋转顺序（维基），一个物体通过任意一个旋转顺序都可以达到同样的姿态，在各个学科里所以为了统一，航空航天领域规定 XYZ 为欧拉角的旋转顺序。

上面已经说了欧拉角的定义。欧拉角的定义也是很直观而且容易理解的，也利于我们的计算，因为我们用的惯性器件也是按照单个轴向运动来测量的。定义上的欧拉角还和我们所说的 Yaw、Pitch、Roll 不是一回事。因为定义上的欧拉角就是刚体绕三个轴的旋转角度，

欧拉旋转和外界的东西（参考系）是没有关系的。Yaw、Pitch、Roll 就是载体对于参考系来说的了，这意味着第一次的旋转不会影响第二、三次的转轴，因为选的参考系都是地球了（这也是产生 Gimbal Lock 的原因，下面会讲到）。

我们现在说的飞机都是在近地表附近飞行，所以我们习惯是拿地球作为参考系，我们的飞机总是在一点起飞在另一点降落。所以我们规定地理方位东、北、天为参考初始点，也就是说，我们的飞机头朝北水平放置时载体坐标系和参考坐标系是重合的，那么接下来我们绕飞机的 Z 轴旋转 30°，这个旋转的欧拉角就是我们所说的 Yaw，同样，绕飞机的 X 轴旋转 30°，我们得到 Pitch，绕飞机 Y 轴旋转得到 Roll，如图 3。

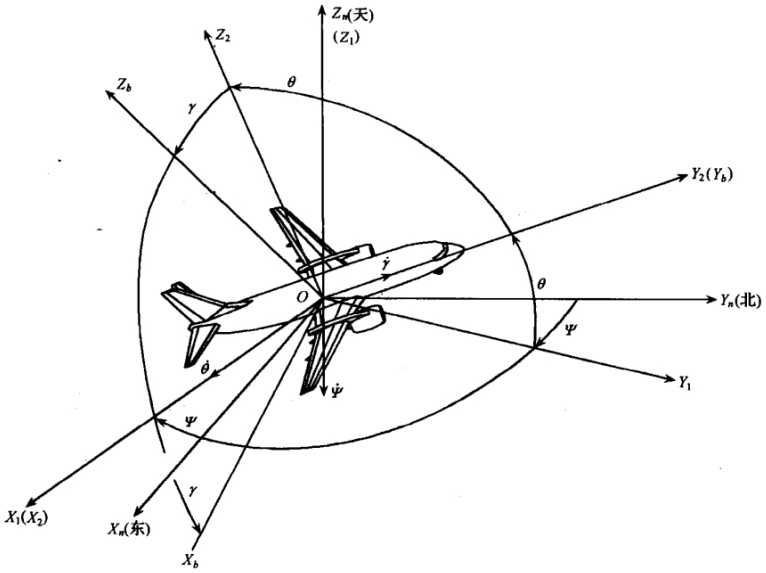
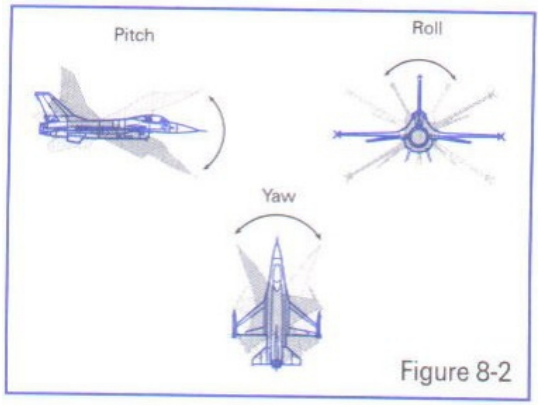


图 1.2.3　飞机空间角位置的确定

图 3

这和规定欧拉角的旋转顺序是一样的，所以说是和欧拉角是一一对应的，要注意欧拉角是基于飞机本身轴旋转得到的，但是得到的姿态却是对于参考坐标系而说的，要不然对我们来说也没有实际的意义，来个更直观的：



欧拉角是有很多优点的。但是也有致命的缺点，那就是 Gimbal Lock（万向节死锁），要理解 Gimbal Lock 所说的情况，这可能有点不好理解，让我们看个现实中的场景。

假如我们有一个望远镜和一个用来放望远镜的三脚架，（我们将）三脚架放在地面上，使支撑望远镜的三脚架的顶部是平行于地平面（参考平面）的，以便使得竖向的旋转轴（记为 x 轴）是完全地垂直于地平面的。现在，我们就可以将望远镜饶 x 轴旋转 360 度，从而观察（以望远镜为中心的）水平包围圈的所有方向。通常将正北朝向方位角度记为 0 度方位角。第二个坐标轴，即平行于地平面的横向的坐标轴（记为 y 轴）使得望远镜可以饶着它上下旋

转，通常将地平面朝向的仰角记为 0 度，这样，望远镜可以向上仰+90 度指向天顶，或者向下-90 度指向脚底。好了，万事俱备。现在，天空中（包括地面上）的每个点只需要唯一的一对 x 和 y 度数就可以确定。比如 x=90 度,y=45 度指向的点是位于正东方向的半天空上。现在，看看万向节死锁是怎么发生的。一次，我们探测到有一个飞行器贴地飞行，位于望远镜的正东方向（x=90 度，y=10 度），朝着我们直飞过来，我们跟踪它。飞行器飞行方向是保持 x 轴角度 90 度不变，而 y 向的角度在慢慢增大。随着飞行器的临近，y 轴角增长的越来越快且当 y 向的角度达到 90 度时（即将超越），突然它急转弯朝南飞去。这时，我们发现我们不能将望远镜朝向南方，因为此时 y 向已经是 90 度，造成我们失去跟踪目标。这就是万向节死锁！

为什么说不能将望远镜朝向南方呢，让我们看看坐标变化，从开始的（x=90 度，y=10 度）到（x=90 度，y=90 度），这个过程没有问题，望远镜慢慢转动跟踪飞行器。当飞行器到达（x=90 度，y=90 度）后，坐标突然变成（x=180 度，y=90 度）（因为朝南），x 由 90 突变成 180 度,所以望远镜需要饶垂直轴向 x 轴旋转 180-90=90 度以便追上飞行器,但此时，望远镜已经是平行于 x 轴，我们知道饶平行于自身的中轴线的的旋转改变不了朝向，就象拧螺丝一样，螺丝头的指向不变。所以望远镜的指向还是天顶。而后由于飞行器飞远，坐标变成（x=180 度，y<90 度）时，y 向角减小，望远镜只能又转回到正东指向，望'器'兴叹。这说明用 x,y 旋转角（又称欧拉角）来定向物体有时并不能按照你想像的那样工作，象上面的例子中从（x=90 度，y=10 度）到（x=90 度，y=90 度），按照欧拉角旋转确实可以正确地定向，但从（x=90 度，y=90 度）到（x=180 度，y=90 度），再到（x=180 度，y<90 度）,按照欧拉角旋转后的定向并非正确。我的理解是坐标值的变化和飞行器空间的位置变化一一对应，但是从（x=90 度，y=90 度）到（x=180 度，y=90 度），再到（x=180 度，y<90 度）这个变化，飞行器位置是连续的变化，但坐标值的变化却不是连续的（从 90 突变到 180），其原因在于（x=90 度，y=90 度）和（x=180 度，y=90 度）甚至和（x=任意度，y=90 度）这些不同的坐标值对应空间同一个位置，这种多个坐标值对应同一个位置的不一致性是造成死锁的根源。

上面是 2 维坐标系中的例子，同样，对于 3 维的也一样。比如有一个平行于 x 轴的向量，我们先将它饶 y 旋转直到它平行于 z 轴，这时，我们会发现任何饶 z 的旋转都改变不了向量的方向，即万向节死锁，所以说传统的欧拉角是不能做到全姿态解析的。（抄的）

方向余弦

一个向量在坐标系中的位置也可以用方向余弦表示，也就是这个向量分别到三个坐标轴的夹角余弦值，实际上就是这个向量到各个坐标轴的投影啦，角度范围是 0~π （维基）。所以推广到载体坐标系和参考坐标系当中，我们就有了载体坐标轴 xyz 分别与参考轴 XYZ 的方向余弦，这里就是所说的方向余弦矩阵了，它是由两组不同的标准正交基的基底向量之间的方向余弦所形成的 3x3 矩阵。方向余弦矩阵可以用来表达一组标准正交基与另一组标准正交基之间的关系。余弦矩阵的列表示载体坐标系中的单位矢量在参考坐标系中的投影。分量形式如下:

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

第 i 行、 j 列的元素表示参考坐标系 i 轴和载体坐标系 j 轴夹角的余弦。

其实方向余弦和欧拉角没有本质上的区别，因为这是用欧拉角表示的方向余弦。一个坐标系到另一个坐标系的变换，在《捷联惯性导航技术》、和《惯性导航》中都是有介绍的，

特别是《惯性导航》有推导的过程。

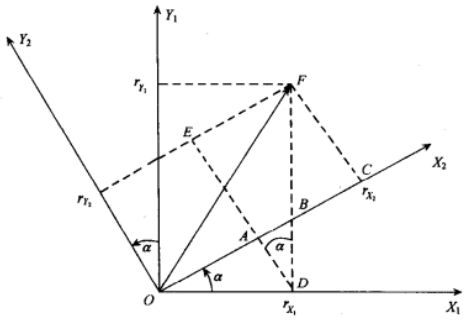


图 1.2.1 坐标系间的变换关系

推广到三轴的单次旋转，我们用矩阵表示为：

将上述三式写成矩阵形式：

$$\begin{bmatrix} r_{X_2} \\ r_{Y_2} \\ r_{Z_2} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{X_1} \\ r_{Y_1} \\ r_{Z_1} \end{bmatrix} \qquad (1.2.1)$$

这里要说一下矩阵的含义，C^2_1 表示坐标系 1 到坐标系 2 的变换矩阵，那么，

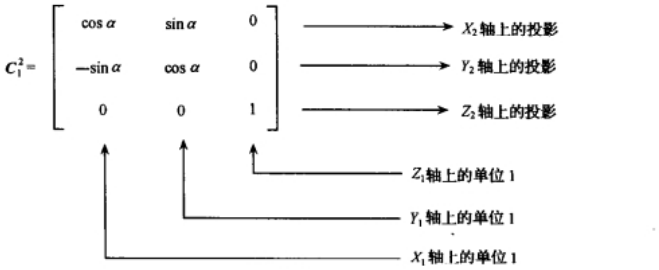


图 1.2.2 坐标系 1 和坐标系 2 之间的投影关系

这是绕 Z 轴的基本旋转，我们可以看到 X2、Y2 投影为 0，Z2 投影为 1，这是为什么呢？自己想想。。。咱们看下面的单位 1，这是什么？所以说这就是上面说的坐标系 2 到坐标系 1 轴的三个方向余弦！对吧。。。这在大家熟知的 imu.c 里面就有直接的计算。那么单独旋转各个轴，我们得到：

$$\boldsymbol{C}_n^1 = \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{C}_1^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \quad \boldsymbol{C}_2^b = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix}$$

实际上，两坐标系任何复杂的角位置关系都可以看做有限次基本旋转的组合，变换矩阵等于基本旋转确定的变换矩阵的连乘，要是不知道矩阵和矩阵乘法，那就看线性代数吧，连乘的基本顺序依据基本旋转的顺序向右排列。之所以有顺序是因为矩阵有“左乘”和“右乘”之分。那么我们得到：

$$\boldsymbol{C}_n^b = \boldsymbol{C}_2^b \boldsymbol{C}_1^2 \boldsymbol{C}_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

按上面推论，此矩阵的下面三个就是三轴对应的方向余弦，γ、θ、ψ就是欧拉角，现在明白了欧拉角和方向余弦矩阵的关系了吧。

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 Rowan Hamilton,1805-1865）在 1843 年爱尔兰发现的数学概念（百度百科）。

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（抄的）。

1、四元数的表示方式：

（1）矢量式

$$\boldsymbol{Q} = q_0 + \boldsymbol{q} \tag{9.2.3}$$

其中，q_0 称四元数 $\boldsymbol{Q}$ 的标量部分，$\boldsymbol{q}$ 称四元数 $\boldsymbol{Q}$ 的矢量部分。对照式(9.2.1)，可看出 $\boldsymbol{q}$ 是三维空间中的一个向量。

（2）复数式

$$\boldsymbol{Q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} \tag{9.2.4}$$

可视为一个超复数，$\boldsymbol{Q}$ 的共轭复数记为

$$\boldsymbol{Q}^* = q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k} \tag{9.2.5}$$

$\boldsymbol{Q}^*$ 称为 $\boldsymbol{Q}$ 的共轭四元数。

（3）三角式

$$\boldsymbol{Q} = \cos\frac{\theta}{2} + \boldsymbol{u}\sin\frac{\theta}{2} \tag{9.2.6}$$

式中，θ 为实数，$\boldsymbol{u}$ 为单位向量。

（4）指数式

$$\boldsymbol{Q} = \mathrm{e}^{\boldsymbol{u}\frac{\theta}{2}} \tag{9.2.7}$$

θ 和 $\boldsymbol{u}$ 同上。

（5）矩阵式

$$\boldsymbol{Q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{9.2.8}$$

2、四元数的计算：

关于四元数的基本运算，加减乘除，大家从别的地方看看吧，维基百科 http://zh.wikipedia.org/wiki/%E5%9B%9B%E5%85%83%E6%95%B0

要注意的是，四元数的元的平方定义为-1，这点等同于复数，四元数不同元相乘得到另外一个，这点有点像向量叉乘。可见，四元数相乘不是虚数相乘也不是向量相乘，这一运算特点是由四元数的性质决定的。四元数由一个实数和三个虚数构成，所以是一个四维空间的向量，但是它的三个虚数又有三维空间的性质，因此，三维空间中的一个矢量，可以看作一个实部为 0 的四元数，这个四元数是这个三维空间的一个矢量在四维空间里的“映像”，或者叫做这个矢量的“四元数映像”。这样，我们就把三维空间和一个四维空间联系起来，用四维空间中四元数的性质和运算规律来研究三维空间中刚体定点转动的问题。

四元数的乘法不可逆性，这里说的是四元数的外积，定义两个四元数:

$$q = a + \vec{u} = a + bi + cj + dk$$
$$p = t + \vec{v} = t + xi + yj + zk$$

它们相乘得到：

$pq = at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} + \vec{v}\times\vec{u}$（看明白这个！既得到了平行又得到了垂直分量）

$$pq = (at-bx-cy-dz)+(bt+ax+dy-cz)i+(ct+ay+bz-dx)j+(dt+za+cx-by)k$$

四元数乘法的非可换性，pq 并不等于 qp。我们发现，两个四元数相乘，我们不仅仅的得到内机，而且还得到外积！这是由四元数运算性质决定的。qp 乘积的向量部分是:

$qp = at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} - \vec{v}\times\vec{u}$，四元数乘法不可逆。

下面要说四元数是如何表示旋转的，这个确实有点不好理解，我看到的资料有的说的还是有点出入的，或许只有自己理解了才能明白吧。“一个单位四元数可以表示一个旋转”，这句话的信息量确实够大的。它基于的思路是:一个坐标系到另一个坐标系的变换可以通过绕一个定义在参考坐标系中的矢量 μ 的单次转动来实现，四元数提供了这种数学描述。我们直观的想一下，我们可以把刚体仅仅旋转一次就能达到任意的姿态，前提是确定了旋转轴线和旋转的角度，大家想一想，感觉也是可以实现的吧。

设 Q = a + bi + cj +dk ，我们可以写成[w，v],其中 w=a，v=bi + cj +dk。那么，v 是矢量，表示三维空间里的旋转轴。w 标量，表示旋转角度。所以，一个四元数可以表示一个完整的旋转。但要注意只有单位四元数才可以表示旋转，至于为什么，那是因为这就是四元数表示旋转的约束条件。

就如上面说的我们利用角度和旋转轴构造了一个四元数，下面就是要满足的这个关系：

$$\boldsymbol{q} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \cos(\mu/2) \\ (\mu_x/\mu)\sin(\mu/2) \\ (\mu_y/\mu)\sin(\mu/2) \\ (\mu_z/\mu)\sin(\mu/2) \end{bmatrix} \qquad (3.53)$$

式中：μ_x、μ_y、μ_z 是角矢量 $\boldsymbol{\mu}$ 的分量，μ 是其大小。

定义 μ 的大小和方向是使参考坐标系绕 μ 转动一个角度 μ ，就能与载体坐标系重合。这可能不好理解，为什么说绕矢量 μ 还转动 μ，(μ_x/μ)就相当于

方向余弦，这里并不矛盾。个人认为，也可以旋转载体坐标系和参考坐标系重合。这样定义也是一样的：

通过旋转轴和绕该轴旋转的角度可以构造一个四元数：

$$w = \cos(\alpha/2)$$
$$x = \sin(\alpha/2)\cos(\beta_x)$$
$$y = \sin(\alpha/2)\cos(\beta_y)$$
$$z = \sin(\alpha/2)\cos(\beta_z)$$

其中 α 是绕旋转轴旋转的角度，$\cos(\beta_x)、\cos(\beta_y)\ \cos(\beta_z)$ 为旋转轴在 x, y, z 方向的分量，就是方向余弦（由此确定了旋转轴，为啥？）。

3、利用四元数进行矢量变换

那么利用四元数代表旋转是如何实现的，在载体系定义的一个矢量 r^b 可以直接利用四元数将其在参考系中表示为 r^n。首先定义一个四元数 $r^{b'}$，它的虚部等于 r^b 的相应分量，标量分量为零：

$$\boldsymbol{r}^b = \mathrm{i}x + \mathrm{j}y + \mathrm{k}z$$
$$\boldsymbol{r}^{b'} = 0 + \mathrm{i}x + \mathrm{j}y + \mathrm{k}z$$

参考系中的 $r^{n'}$ 表示为 $\boldsymbol{r}^{n'} = \boldsymbol{q}\boldsymbol{r}^{b'}\boldsymbol{q}^*$，其中 $q = a + bi + cj + dk$，q^*的复共轭。因此有：

$$\begin{aligned}\boldsymbol{r}^{n'} &= (a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d)(0 + \mathrm{i}x + \mathrm{j}y + \mathrm{k}z)(a - \mathrm{i}b - \mathrm{j}c - \mathrm{k}d) = \\ &0 + \{(a^2 + b^2 - c^2 - d^2)x + 2(bc - ad)y + 2(bd + ac)z\}\mathrm{i} + \\ &\{2(bc + ad)x + (a^2 - b^2 + c^2 - d^2)y + 2(cd - ab)z\}\mathrm{j} + \\ &\{2(bd - ac)x + 2(cd + ab)y + (a^2 - b^2 - c^2 + d^2)z\}\mathrm{k}\end{aligned}$$

写成矩阵式：

$$\boldsymbol{C} = \begin{bmatrix} (a^2 + b^2 - c^2 - d^2) & 2(bc - ad) & 2(bd + ac) \\ 2(bc + ad) & (a^2 - b^2 + c^2 - d^2) & 2(cd - ab) \\ 2(bd - ac) & 2(cd + ab) & (a^2 - b^2 - c^2 + d^2) \end{bmatrix} \quad (3.59)$$

这个矩阵就对应方向余弦矩阵了，从而对应欧拉角了。

肯定还有没回过神的，上面仅仅说明了一个“结果”：$\boldsymbol{r}^{n'} = \boldsymbol{q}\boldsymbol{r}^{b'}\boldsymbol{q}^*$。要直观说清楚四元数如何工作的确实有点抽象。我们把三维空间里的点用实部为0的四元数表示成 $q = bi + cj + dk$，我们可以理解为虚部就是表示我们生活的三维空间，那么我们把 q 左乘一个标准四元数后，我们能得到什么？

为了更清楚地看到两个四元数乘积到底是什么样子，我们把上节用到的向量空间的观点

拿过来，四元数的全体构成的集合 F 是实数域上的四维向量空间，可以把四元数 q = a + bi + cj +dk 看成四维实数元组（a，b，c，d）。然后用另外一个四元数 Q 左乘 q，这个运算就相当于 q 在四维空间 F 上的线性变换，这就如同两个虚数相乘（1+i）*（2+i）=1+3i，（2+i）在复数坐标系上的一个线性变换——大小和方向都发送了变化。如果 Q 是个纯实数（虚部为0的四元数），那么 Q*q 就是 q 简单的伸缩比例，方向什么的没有发生变化；如果 Q 是个虚部不为0的四元数，同样道理 q 不仅仅是做了伸缩，而且方向也发生了旋转，不仅如此，所有的向量长度都伸缩相同的倍数。根据线性代数理论，一个等距线性变换要么是单纯的旋转，要么是单纯的对称变换，要么是二者的复合。而四维空间上这样的线性变换必有两个垂直的二维不变子空间（就是$pq = at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} + \vec{v}\times\vec{u}$后三相），也就是说，可以在四维空间中找到两个相垂直的平面，在每个平面上的向量经过变换之后还是在这个平面上。

4、欧拉角、方向余弦、四元数的关系

上面已经说的很清楚了，看一下：

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

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

$$\begin{bmatrix} (a^2 + b^2 - c^2 - d^2) & 2(bc - ad) & 2(bd + ac) \\ 2(bc + ad) & (a^2 - b^2 + c^2 - d^2) & 2(cd - ab) \\ 2(bd - ac) & 2(cd + ab) & (a^2 - b^2 - c^2 + d^2) \end{bmatrix} \quad (3.63)$$

通过比较上述等式的各个元素，四元数可以直接用欧拉角或方向余弦表示。同样地，欧拉角也可以用方向余弦或四元数表示。

用方向余弦表示四元数：

对于小角度位移，四元数参数可以用下面的关系式推导：

$$\begin{aligned} a &= \frac{1}{2}(1 + c_{11} + c_{22} + c_{33})^{1/2} \\ b &= \frac{1}{4a}(c_{32} - c_{23}) \\ c &= \frac{1}{4a}(c_{13} - c_{31}) \\ d &= \frac{1}{4a}(c_{21} - c_{12}) \end{aligned} \quad (3.64)$$

考虑方向余弦元素的相对大小，谢伯特提出了由方向余弦求解四元数的更全面的算法[2]。

用欧拉角直接表示四元数：

$$a = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2}$$

$$b = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2}$$

$$c = \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2}$$

$$d = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2}$$

用方向余弦表示欧拉角：

可以按下述方法直接由方向余弦推导出欧拉角。当 0 ！= 90 。时，欧拉角可由下式计算:

$$\phi = \arctan\left[\frac{c_{32}}{c_{33}}\right]$$

$$\theta = \arcsin[-c_{31}]$$

$$\psi = \arctan\left[\frac{c_{21}}{c_{11}}\right]$$

这个公式是在 imu.c 里有直接应用的。

5、传感器

关于传感器，价格贵的肯定性能就好点。还有就是传感器的安装方向，个人喜好 X 模式，飞行更灵活，适合航拍。

6、数据融合

数据融合也算是姿态解算的核心了，好的解算算法可以更好的补偿传感器误差，目前就研究了论坛里的 imu.c 滤波思想，感觉很巧妙。

```
// Description:
//
// Quaternion implementation of the 'DCM filter' [Mayhony et al].
//
// 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 'IMUupdate()' every sample period and parse calibrated gyroscope ('gx', 'gy', 'gz')
// and accelerometer ('ax', 'ay', 'ay') data.  Gyroscope units are radians/second, accelerometer
// units are irrelevant as the vector is normalised.
```

```
//
//=====================================================================
====================================

#define Kp 10.0f //proportional gain governs rate of convergence to
accelerometer/magnetometer
#define Ki 0.008f // integral gain governs rate of convergence of gyroscope biases
#define halfT 0.001f // half the sample period 采样周期的一半

float q0 = 1, q1 = 0, q2 = 0, q3 = 0; // quaternion elements representing the estimated
orientation
float exInt = 0, eyInt = 0, ezInt = 0; // scaled integral error

void IMUupdate(float gx, float gy, float gz, float ax, float ay, float az)
{
  float norm;
  float vx, vy, vz;// wx, wy, wz;
  float ex, ey, ez;

//先把这些用得到的值算好
  float q0q0 = q0*q0;
  float q0q1 = q0*q1;
  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;


//normalise the measurements
  norm = sqrt(ax*ax + ay*ay + az*az); //加计数据归一化,把加计的三维向量转换
为单位向量。因为是单位矢量到参考性的投影，所以要把 acc 单位化
  ax = ax / norm; //其实归一化改变的只是这三
个向量的长度，也就是只
  ay = ay / norm; //改变了相同的倍数，方向并
没改变，也是为了与单位
  az = az / norm; //四元数对应。
```

//estimated direction of gravity and flux (v and w)，估计重力方向和流量/变迁

//这是把四元数换算成方向余弦中的第三行的三个元素，根据余弦矩阵和欧拉角的定义，地里坐标系的 Z 轴重力向量，

$$\begin{bmatrix} (a^2+b^2-c^2-d^2) & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & (a^2-b^2+c^2-d^2) & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & (a^2-b^2-c^2+d^2) \end{bmatrix} \quad (3.63)$$

//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。(是带有误差的姿态。)

```
vx = 2*(q1q3 - q0q2);                    //参考系 Z 轴与载体系 x 轴之间方向余弦向量
vy = 2*(q0q1 + q2q3);                    //参考系 Z 轴与载体系 y 轴之间方向余弦向量
vz = q0q0 - q1q1 - q2q2 + q3q3 ;         //参考系 Z 轴与载体系 z 轴之间方向余弦向量
```

//error is sum of cross product between reference direction of fields and direction measured by sensors

//ax，ay，az 是机体坐标参考系上，加计测出来的重力向量，也就是实际测出来的重力向量（这明显就是重力在三个轴

//上的“分担”)。那他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的叉积（外积）来表示，ex，ey，ez 就是两重力方向的叉积，注意这个

//叉积向量仍是在机体坐标系上的，而陀螺的积分误差也是在机体坐标系上的，而且叉积的大小和陀螺的积分误差成正

//比，正好用来修正陀螺，这是因为，由于陀螺是对机体直接积分的，所以对陀螺的误差修正会直接会直接体现在对机

//体坐标系的修正，就是对机体修正。说白了，就是用加计测量标定陀螺积分。

//这里误差没说清楚，不是指向量差。这个叉积误差是指将带有误差的加计向量转动到与重力向量重合，需要绕什么轴，

//转多少角度。逆向推理一下，这个叉积在机体三轴上的投影，就是加计和重力之间的角度误差。也就是说，如果陀螺

//按这个叉积误差的轴，转动叉积误差的角度（也就是转动三轴投影的角度）那就能把加计和重力向量的误差消除掉。

//（具体可看向量叉积的定义）如果完全按叉积误差转过去，那就是完全信任加计。如果一点也不转，那就是完全信任

//陀螺。那么把这个叉积的三轴乘以 X%，加到陀螺的积分角度上去，就是这个 x%互补

系数的互补算法了。

//ps：实际上叉积的 length 是两向量夹角的正弦，而且必须在±90 度以内，并不完全与误差角度成线性正比。如果转成三

//轴夹角，按欧拉角的转动顺序分解到三轴上去，会很麻烦。这里的叉积算法把 sin 误差当成角度误差，并无视欧拉角的

//转动顺序，在误差较小的时候，并不会有影响。因为这个修正对误差来说，是收敛的，正常情况下误差只会越来越小。

//为了解释方便，所以上面就把叉积等同于角度误差了。

```
ex = ay*vz - az*vy ;                    //上面说了，vx,vy,vz 就是方向余弦，
ey = az*vx - ax*vz ;
ez = ax*vy - ay*vx ;
```

```
//对误差进行积分
exInt = exInt + ex * Ki;
eyInt = eyInt + ey * Ki;
ezInt = ezInt + ez * Ki;
```

```
// adjusted gyroscope measurements
gx = gx + Kp*ex + exInt;                //将误差 PI 后补偿到陀螺仪，即补偿零点漂移
gy = gy + Kp*ey + eyInt;                //修正陀螺输出
gz = gz + Kp*ez + ezInt;                //这里的 gz 由于没有观测者进行矫正会产生漂移，表现出来的就是积分自增或自减
```

```
// integrate quaternion rate and normalise          //四元素的微分方程
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;
```

这是四元数的微分方程，用于更新四元数，

$$\dot{a} = -0.5(b\omega_x + c\omega_y + d\omega_z)$$

$$\dot{b} = 0.5(a\omega_x - d\omega_y + c\omega_z)$$

$$\dot{c} = 0.5(d\omega_x + a\omega_y - b\omega_z)$$

$$\dot{d} = -0.5(c\omega_x - b\omega_y - a\omega_z)$$

所以说 halfT 定义为 0.001，因为我们的姿态解算的周期是 2ms，这里有个 0.5 系数。

```
    // normalise quaternion
    //由于误差的引入，使计算的变换四元数的模不再等于 1，变换四元数失去规范性，因此在利用更新四元数
    //计算欧拉角时必须要对四元数规范化处理。
      norm = sqrt(q0*q0 + q1*q1 + q2*q2 + q3*q3);
      q0 = q0 / norm;
      q1 = q1 / norm;
      q2 = q2 / norm;
      q3 = q3 / norm;
    //由计算四元数计算欧拉角
    // Q_ANGLE.Z = atan2(2 * q1 * q2 + 2 * q0 * q3, -2 * q2*q2 - 2 * q3* q3 + 1)* Rad; // yaw
      Q_ANGLE.Y    = asin(-2 * q1 * q3 + 2 * q0* q2)* Rad; // pitch
      Q_ANGLE.X = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1) * Rad; // roll
      if(Q_ANGLE.X>90||Q_ANGLE.X<-90)
      {
      if(Q_ANGLE.Y>0)
        Q_ANGLE.Y=180-Q_ANGLE.Y;
      if(Q_ANGLE.Y<0)
        Q_ANGLE.Y=-(180+Q_ANGLE.Y);
      }
    }
```