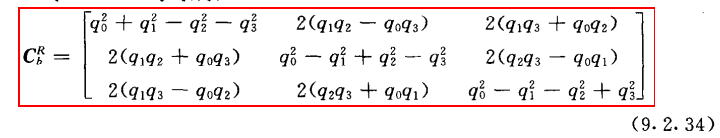
二、四元数与姿态阵间的关系

从上面我们知道了四元数的定义，可这四个数和我们要求的三个姿态角有什么关系呢？下面是详细的推导，同样摘自《惯性导航》-秦永元P292-297。

|  |
| --- |
| 四元数与姿态阵间的关系——摘自《惯性导航》-秦永元P292-297 |

呃，粘了这么多其实就是为了想知道推导过程小伙伴好理解，真正有用的就是（9.2.34）这个公式。▲这个公式把四元数转换成了方向余弦矩阵中的几个元素，再用这几个元素转换为欧拉角。就求解除了姿态！



先从四元数q0~q3转成方向余弦矩阵：

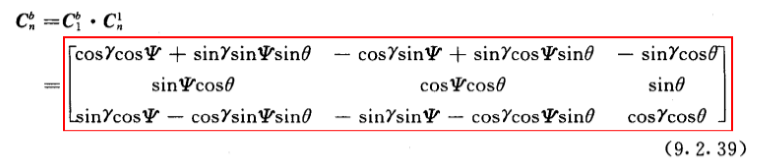
再从方向余弦矩阵转换为欧拉角：

好的，公式原理都讲清楚了，下面来看一下代码：

|  |
| --- |
| //四元数转欧拉角，这里四元数是q1~q4 和公式里q0~q3相对应。  void Quaternion::to\_euler(float \*roll, float \*pitch, float \*yaw)  {  if (roll) {  \*roll = degrees(atan2f(2.0f\*(q1\*q2 + q3\*q4),1 - 2.0f\*(q2\*q2 + q3\*q3)));  //\*roll = degrees(atan2f(-2.0f\*(q2\*q4 - q1\*q3),1 - 2.0f\*(q2\*q2 + q3\*q3)));  }  if (pitch) {  // 使用safe\_asin()来处理pitch接近90/-90时的奇点  \*pitch = degrees(safe\_asin(2.0f\*(q1\*q3 - q2\*q4)));  //\*pitch = degrees(safe\_asin(2.0f\*(q3\*q4 + q1\*q2)));  }  if (yaw) {  \*yaw = degrees(atan2f(2.0f\*(q2\*q3 - q1\*q4), 2.0f\*(q1\*q1 + q2\*q2) - 1));  //\*yaw = degrees(atan2f(2.0f\*(q2\*q3 - q1\*q4), 2.0f\*(q1\*q1 + q3\*q3) - 1));  }  } |

对比余弦矩阵转换为欧拉角的公式很容易理解了吧，注意一下，红色是原版的代码，和公式还是有出入的，自己细心观察一下吧。被注释了的黑色代码是我根据上面的公式写的。但黑色的求解出来的欧拉角反映出来的姿态是不对的，具体表现为俯仰（pitch）和横滚（roll）是相反的，为啥根据公式写的是不对的？群里的小伙伴给与了我热心的解答。

这个错误主要是由于方向余弦矩阵的旋转顺序不一样，也就是公式（9.2.39）不一样，这是由于旋转的先后顺序不同引起的，具体大家参考《惯性导航》绪论来看就能明白，因为这一点小弟还有点混乱，就点到这为止。



以上就是四元数求解欧拉角的方法。通过公式可以看到，要求欧拉角需要求得四元数的方向余弦矩阵；要求得四元数方向余弦矩阵，需要求得四元数q0~q3，那么如何求得q0~q3？接下来详细介绍。