昵称： XXX已失联
园龄： 4年7个月
粉丝： 606
关注： 102
+加关注

< 2020年9月 >

日	一	二	三	四	五	六
30	31	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	1	2	3
4	5	6	7	8	9	10

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

V-rep(49)
机器人学(33)
python(27)
ROS(24)
计算机图形学(21)
自主移动机器人(21)
VTK(14)
物理引擎(13)
传感器(11)
PCL点云库(10)
更多

随笔档案 (166)

2020年8月(2)
2020年7月(1)
2020年5月(2)
2020年1月(1)
2019年12月(2)
2019年11月(1)
2019年3月(2)
2018年8月(1)
2018年7月(4)
2018年6月(9)
2018年5月(2)
2018年1月(9)
2017年12月(11)
2017年11月(7)
2017年10月(4)
2017年9月(7)
2017年8月(9)
2017年7月(5)
2017年6月(5)
2017年5月(4)

文章档案 (12)

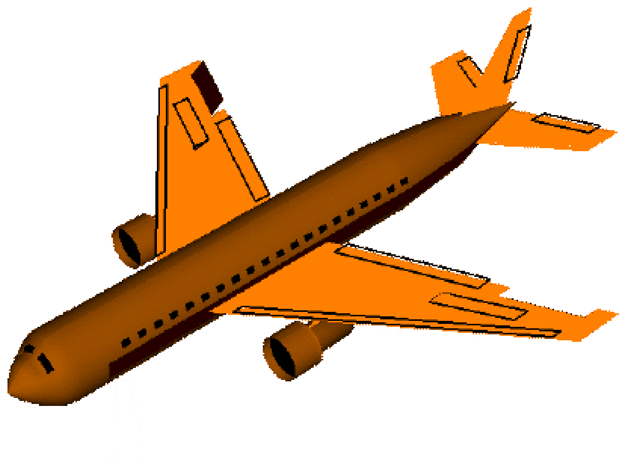
2018年8月(1)
2017年8月(2)
2017年7月(1)

四元数与欧拉角（RPY角）的相互转换

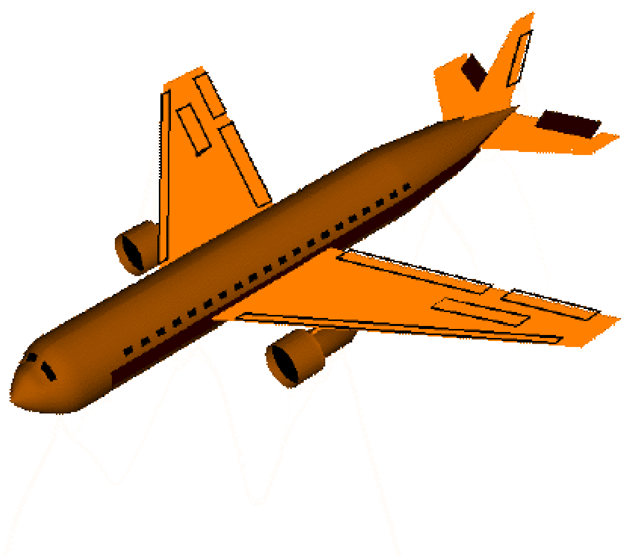
- **RPY角与Z-Y-X欧拉角**

描述坐标系{B}相对于参考坐标系{A}的姿态有两种方式。第一种是**绕固定（参考）坐标轴旋转**：假设开始两个坐标系重合，先将{B}绕{A}的X轴旋转γ，然后绕{A}的Y轴旋转β，最后绕{A}的Z轴旋转α，就能旋转到当前姿态。可以称其为X-Y-Z fixed angles或RPY角(Roll, Pitch, Yaw)。

Roll:横滚



Pitch: 俯仰



Yaw: 偏航（航向）

2017年6月(2)
2017年3月(1)
2017年2月(1)
2017年1月(1)
2016年12月(2)
2016年3月(1)

最新评论

1. Re:二连杆机械臂阻抗控制模拟（一）
请问博主，你是怎么推到出角加速度的 有没有公式，跪求详解，关于求角加速度的程序看不太懂

--懒阳阳

2. Re:PRM路径规划算法
您好，我正在做相关方向的论文，博主可以发一份给我吗?

邮箱：1475667786@qq.com

--1475667786

3. Re:ROS中利用V-rep进行地图构建仿真
博主你好，我想请教一下 我在保存激光雷达距离数据的时候一次只能导出一帧的数据，请问是不是只能这样导出，或者有没有办法把整个仿真过程的所有距离数据一次性导出？ 谢谢！ ...

--苏打不辣

4. Re:V-rep学习笔记：机器人逆运动学数值解法（The Jacobian Transpose Method）
@WangQi1024 软件手册里有写，支持6种编程语言c lua python matlab java octave...

--XXX已失联

5. Re:V-rep学习笔记：机器人逆运动学数值解法（The Jacobian Transpose Method）
请问博主，怎么在VREP中使用C或者python进行开发？是不是自己用C写好了动力学和运动学函数然后把关节运动的角度等参数传入在vrep中让模型进行运动呢？还是有其他方法？ ...

--WangQi1024

阅读排行榜

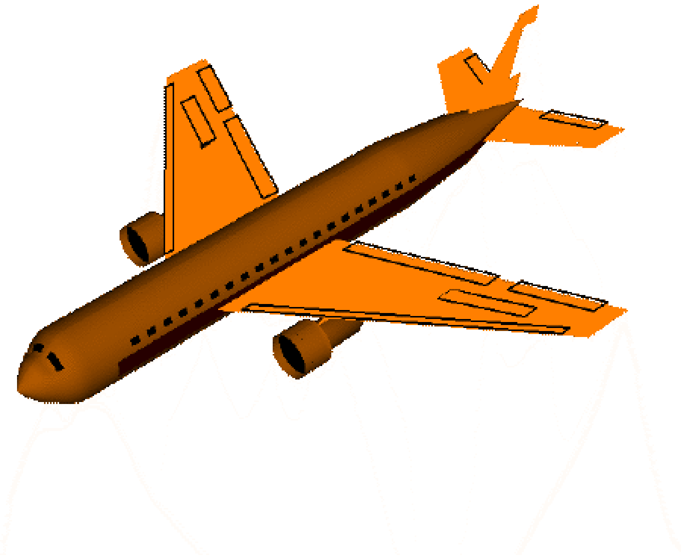
1. 四元数与欧拉角（RPY角）的相互转换(73125)
2. RRT路径规划算法(39088)
3. IIR数字滤波器的实现（C语言）(28229)
4. KNN算法与Kd树(27915)
5. 粒子群优化算法（Particle Swarm Optimization）(25868)

评论排行榜

1. Bug2算法的实现（RobotBASIC环境中仿真）(10)
2. PCL点云库：ICP算法(10)
3. V-rep学习笔记：转动关节1(8)
4. Solidworks 2016中导出URDF文件(8)
5. V-rep学习笔记：ROSInterface(8)

推荐排行榜

1. KNN算法与Kd树(17)
2. 四元数与欧拉角（RPY角）的相互转换(12)
3. RRT路径规划算法(9)
4. A*算法(9)
5. 粒子群优化算法（Particle Swarm Optimization）(8)



由于是绕固定坐标系旋转，则旋转矩阵为（$c\alpha$ is shorthand for $\cos\alpha$, $s\alpha$ is shorthand for $\sin\alpha$,and so on.）

$$R_{XYZ}(\gamma,\beta,\alpha)=R_Z(\alpha)R_Y(\beta)R_X(\gamma)=\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

另一种姿态描述方式是**绕自身坐标轴旋转**：假设开始两个坐标系重合，先将{B}绕自身的Z轴旋转α，然后绕Y轴旋转β，最后绕X轴旋转γ，就能旋转到当前姿态。称其为Z-Y-X欧拉角，由于是绕自身坐标轴进行旋转，则旋转矩阵为：

$$R_{Z'Y'X'}(\alpha,\beta,\gamma)=R_Z(\alpha)R_Y(\beta)R_X(\gamma)=\begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$$

可以发现这两种描述方式得到的旋转矩阵是一样的，即绕固定坐标轴X-Y-Z旋转(γ,β,α)和绕自身坐标轴Z-Y-X旋转(α,β,γ)的最终结果一样，只是描述的方法有差别而已。In gerenal: three rotations taken about fixed axes yield the same final orientation as the same three rotations taken in opposite order about the axes of the moving frame.

- **Axis-Angle与四元数**

绕坐标轴的多次旋转可以等效为绕某一转轴旋转一定的角度。假设等效旋转轴方向向量为$\vec{K}=[k_x,k_y,k_z]^T$，等效旋转角为θ，则四元数$q=(x,y,z,w)$，其中：

$$\begin{aligned} x &= k_x \cdot sin\frac{\theta}{2} \\ y &= k_y \cdot sin\frac{\theta}{2} \\ z &= k_z \cdot sin\frac{\theta}{2} \\ w &= cos\frac{\theta}{2} \end{aligned}$$

且有$x^2+y^2+z^2+w^2=1$

即四元数存储了旋转轴和旋转角的信息，它能方便的描述刚体绕任意轴的旋转。

四元数转换为旋转矩阵：

$$R=\begin{bmatrix} 1-2y^2-2z^2 & 2(xy-zw) & 2(xz+yw) \\ 2(xy+zw) & 1-2x^2-2z^2 & 2(yz-xw) \\ 2(xz-yw) & 2(yz+xw) & 1-2x^2-2y^2 \end{bmatrix}$$

已知旋转矩阵为：

$$R=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

则对应的四元数为：

$$x = \frac{r_{32} - r_{23}}{4w}$$
$$y = \frac{r_{13} - r_{31}}{4w}$$
$$z = \frac{r_{21} - r_{12}}{4w}$$
$$w = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}}$$

- **四元数与欧拉角的相互转换**

  定义两个四元数：

  $q = a + \vec{u} = a + bi + cj + dk$
  $p = t + \vec{v} = t + xi + yj + zk$
  其中$\vec{u}$ 表示矢量$< b,c,d >$；而 $\vec{v}$ 表示矢量$< x,y,z >$

四元数加法：

跟复数、向量和矩阵一样，两个四元数之和需要将不同的元素加起来。

$$p + q = a + t + \vec{u} + \vec{v} = (a+t) + (b+x)i + (c+y)j + (d+z)k$$

加法遵循实数和复数的所有交换律和结合律。

四元数乘法：

四元数的乘法的意义类似于矩阵的乘法，可以表示旋转的合成。当有多次旋转操作时，使用四元数可以获得更高的计算效率。

$$pq = at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} + \vec{u}\times\vec{v}$$
$$pq = (at - bx - cy - dz) + (ax + bt + cz - dy)i + (ay - bz + ct + dx)j + (az + dt - cx + by)k$$

由于四元数乘法的非可换性，pq并不等于qp，qp乘积的向量部分是：

$$qp = at - \vec{u}\cdot\vec{v} + a\vec{v} + t\vec{u} - \vec{u}\times\vec{v}$$

Mathematica中有四元数相关的程序包**Quaternions Package**，需要先导入才能使用。下面计算了三个四元数的乘积：

```
<<Quaternions`      (* This loads the package *)
Quaternion[2, 1, 1, 3] ** Quaternion[2, 1, 1, 0] ** Quaternion[1, 1, 1, 1]    (* Be sure to u
```

计算结果为：Quaternion[-12, 4, 14, 2]

那么将Z-Y-X欧拉角（或RPY角：绕固定坐标系的X-Y-Z依次旋转α,β,γ角）转换为四元数：

$$q = \begin{bmatrix} \cos\frac{\gamma}{2} \\ 0 \\ 0 \\ \sin\frac{\gamma}{2} \end{bmatrix} \begin{bmatrix} \cos\frac{\beta}{2} \\ 0 \\ \sin\frac{\beta}{2} \\ 0 \end{bmatrix} \begin{bmatrix} \cos\frac{\alpha}{2} \\ \sin\frac{\alpha}{2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2}\cos\frac{\beta}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\sin\frac{\beta}{2}\sin\frac{\gamma}{2} \\ \sin\frac{\alpha}{2}\cos\frac{\beta}{2}\cos\frac{\gamma}{2} - \cos\frac{\alpha}{2}\sin\frac{\beta}{2}\sin\frac{\gamma}{2} \\ \cos\frac{\alpha}{2}\sin\frac{\beta}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\cos\frac{\beta}{2}\sin\frac{\gamma}{2} \\ \cos\frac{\alpha}{2}\cos\frac{\beta}{2}\sin\frac{\gamma}{2} - \sin\frac{\alpha}{2}\sin\frac{\beta}{2}\cos\frac{\gamma}{2} \end{bmatrix}$$

根据上面的公式可以求出逆解，即由四元数$q = (q_0, q_1, q_2, q_3)$或$q = (w, x, y, z)$到欧拉角的转换为：

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \arctan\frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0q_2 - q_1q_3)) \\ \arctan\frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

由于arctan和arcsin的取值范围在$\frac{-\pi}{2}$和$\frac{\pi}{2}$之间，只有180°，而绕某个轴旋转时范围是360°，因此要使用**atan2**函数代替arctan函数：

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} atan2(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0q_2 - q_1q_3)) \\ atan2(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

对于tan(θ) = y / x：

θ = ATan(y / x)求出的θ取值范围是[-PI/2, PI/2]；

θ = ATan2(y, x)求出的θ取值范围是[-PI,　PI]。

- 当 (x, y) 在第一象限, 0 < θ < PI/2
- 当 (x, y) 在第二象限 PI/2 < θ≤PI
- 当 (x, y) 在第三象限, -PI < θ < -PI/2
- 当 (x, y) 在第四象限, -PI/2 < θ < 0

将四元数转换为欧拉角可以参考下面的代码。需要注意**欧拉角有12种旋转次序**，而上面推导的公式是按照Z-Y-X顺序进行的，所以有时会在网上看到不同的转换公式（因为对应着不同的旋转次序），在使用时一定要注意旋转次序是什么。比如ADAMS软件里就默认Body 3-1-3次序，即Z-X-Z欧拉角，而VREP中则按照X-Y-Z欧拉角旋转。

```
enum RotSeq{zyx, zyz, zxy, zxz, yxz, yxy, yzx, yzy, xyz, xyx, xzy,xzx};
```

```
// COMPILE: g++ -o quat2EulerTest quat2EulerTest.cpp
#include <iostream>
#include <cmath>

using namespace std;

///////////////////////////////
// Quaternion struct
// Simple incomplete quaternion struct for demo purpose
///////////////////////////////
struct Quaternion{
  Quaternion():x(0), y(0), z(0), w(1){};
  Quaternion(double x, double y, double z, double w):x(x), y(y), z(z), w(w){};

  void normalize(){
    double norm = std::sqrt(x*x + y*y + z*z + w*w);
    x /= norm;
    y /= norm;
    z /= norm;
    w /= norm;
  }

  double norm(){
    return std::sqrt(x*x + y*y + z*z + w*w);
  }

  double x;
  double y;
  double z;
  double w;

};

///////////////////////////////
// Quaternion to Euler
///////////////////////////////
enum RotSeq{zyx, zyz, zxy, zxz, yxz, yxy, yzx, yzy, xyz, xyx, xzy,xzx};

void twoaxisrot(double r11, double r12, double r21, double r31, double r32, double res[]){
  res[0] = atan2( r11, r12 );
  res[1] = acos ( r21 );
  res[2] = atan2( r31, r32 );
}

void threeaxisrot(double r11, double r12, double r21, double r31, double r32, double res[]){
  res[0] = atan2( r31, r32 );
  res[1] = asin ( r21 );
  res[2] = atan2( r11, r12 );
}

void quaternion2Euler(const Quaternion& q, double res[], RotSeq rotSeq)
{
    switch(rotSeq){
    case zyx:
      threeaxisrot( 2*(q.x*q.y + q.w*q.z),
                     q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                    -2*(q.x*q.z - q.w*q.y),
                     2*(q.y*q.z + q.w*q.x),
                     q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
```

```
                  res);
      break;

    case zyz:
      twoaxisrot( 2*(q.y*q.z - q.w*q.x),
                   2*(q.x*q.z + q.w*q.y),
                   q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
                   2*(q.y*q.z + q.w*q.x),
                  -2*(q.x*q.z - q.w*q.y),
                  res);
      break;

    case zxy:
      threeaxisrot( -2*(q.x*q.y - q.w*q.z),
                      q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                      2*(q.y*q.z + q.w*q.x),
                     -2*(q.x*q.z - q.w*q.y),
                      q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
                      res);
      break;

    case zxz:
      twoaxisrot( 2*(q.x*q.z + q.w*q.y),
                  -2*(q.y*q.z - q.w*q.x),
                   q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
                   2*(q.x*q.z - q.w*q.y),
                   2*(q.y*q.z + q.w*q.x),
                   res);
      break;

    case yxz:
      threeaxisrot( 2*(q.x*q.z + q.w*q.y),
                     q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
                    -2*(q.y*q.z - q.w*q.x),
                     2*(q.x*q.y + q.w*q.z),
                     q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                     res);
      break;

    case yxy:
      twoaxisrot( 2*(q.x*q.y - q.w*q.z),
                   2*(q.y*q.z + q.w*q.x),
                   q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                   2*(q.x*q.y + q.w*q.z),
                  -2*(q.y*q.z - q.w*q.x),
                  res);
      break;

    case yzx:
      threeaxisrot( -2*(q.x*q.z - q.w*q.y),
                      q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                      2*(q.x*q.y + q.w*q.z),
                     -2*(q.y*q.z - q.w*q.x),
                      q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                      res);
      break;

    case yzy:
      twoaxisrot( 2*(q.y*q.z + q.w*q.x),
                  -2*(q.x*q.y - q.w*q.z),
                   q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                   2*(q.y*q.z - q.w*q.x),
                   2*(q.x*q.y + q.w*q.z),
                   res);
      break;

    case xyz:
      threeaxisrot( -2*(q.y*q.z - q.w*q.x),
                    q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z,
                    2*(q.x*q.z + q.w*q.y),
                   -2*(q.x*q.y - q.w*q.z),
                    q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                    res);
      break;

    case xyx:
      twoaxisrot( 2*(q.x*q.y + q.w*q.z),
```

```
                  -2*(q.x*q.z - q.w*q.y),
                   q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                   2*(q.x*q.y - q.w*q.z),
                   2*(q.x*q.z + q.w*q.y),
                   res);
      break;

    case xzy:
      threeaxisrot( 2*(q.y*q.z + q.w*q.x),
                     q.w*q.w - q.x*q.x + q.y*q.y - q.z*q.z,
                    -2*(q.x*q.y - q.w*q.z),
                     2*(q.x*q.z + q.w*q.y),
                     q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                     res);
      break;

    case xzx:
      twoaxisrot( 2*(q.x*q.z - q.w*q.y),
                   2*(q.x*q.y + q.w*q.z),
                   q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z,
                   2*(q.x*q.z + q.w*q.y),
                  -2*(q.x*q.y - q.w*q.z),
                  res);
      break;
    default:
      std::cout << "Unknown rotation sequence" << std::endl;
      break;
   }
}

///////////////////////////////
// Helper functions
///////////////////////////////
Quaternion operator*(Quaternion& q1, Quaternion& q2){
  Quaternion q;
  q.w = q1.w*q2.w - q1.x*q2.x - q1.y*q2.y - q1.z*q2.z;
  q.x = q1.w*q2.x + q1.x*q2.w + q1.y*q2.z - q1.z*q2.y;
  q.y = q1.w*q2.y - q1.x*q2.z + q1.y*q2.w + q1.z*q2.x;
  q.z = q1.w*q2.z + q1.x*q2.y - q1.y*q2.x + q1.z*q2.w;
  return q;
}

ostream& operator <<(std::ostream& stream, const Quaternion& q) {
  cout << q.w << " "<< showpos << q.x << "i " << q.y << "j " << q.z << "k";
  cout << noshowpos;
}

double rad2deg(double rad){
  return rad*180.0/M_PI;
}

///////////////////////////////
// Main
///////////////////////////////
int main(){

  Quaternion q; // x,y,z,w
  Quaternion qx45(sin(M_PI/8), 0,0, cos(M_PI/8) );
  Quaternion qy45(0, sin(M_PI/8), 0, cos(M_PI/8));
  Quaternion qz45(0, 0, sin(M_PI/8), cos(M_PI/8));
  Quaternion qx90(sin(M_PI/4), 0,0, cos(M_PI/4) );
  Quaternion qy90(0, sin(M_PI/4), 0, cos(M_PI/4));
  Quaternion qz90(0, 0, sin(M_PI/4), cos(M_PI/4));

  double res[3];

  q = qz45*qx45;
  q.normalize();
  quaternion2Euler(q, res, zyx);
  cout << "Rotation sequence: X->Y->Z" << endl;
  cout << "x45 -> z45" << endl;
  cout << "q: " << q << endl;
  cout << "x: " << rad2deg(res[0]) << " y: " << rad2deg(res[1]) << " z: " << rad2deg(res[2])

  q = qz90*qx90;
  q.normalize();
  quaternion2Euler(q, res, zyx);
```

```
    cout << "Rotation sequence: X->Y->Z" << endl;
    cout << "x90 -> z90" << endl;
    cout << "q: " << q << endl;
    cout << "x: " << rad2deg(res[0]) << " y: " << rad2deg(res[1]) << " z: " << rad2deg(res[2])

    q = qx90*qz90;
    q.normalize();
    quaternion2Euler(q, res, xyz);
    cout << "Rotation sequence: Z->Y->X" << endl;
    cout << "z90 -> x90" << endl;
    cout << "q: " << q << endl;
    cout << "x: " << rad2deg(res[0]) << " y: " << rad2deg(res[1]) << " z: " << rad2deg(res[2])
}
```

上面的代码存在一个问题，即奇异性没有考虑。下面看一种特殊的情况（参考Maths - Conversion Quaternion to Euler）：假设一架飞机绕Y轴旋转了90°（俯仰角pitch=90），机头垂直向上，此时如何计算航向角和横滚角？

这时会发生自由度丢失的情况，即Yaw和Roll会变为一个自由度。此时再使用上面的公式根据四元数计算欧拉角会出现问题：

$\arcsin(2(q_0q_2 - q_1q_3))$的定义域为$[-1,1]$，因此$(q_0q_2 - q_1q_3) \in [-0.5, 0.5]$，当$q_0q_2 - q_1q_3 = 0.5$时（在程序中浮点数不能直接进行等于判断，要使用合理的阈值），俯仰角β为90°，将其带入正向公式计算出四元数(q_0, q_1, q_2, q_3)，然后可以发现逆向公式中atan2函数中的参数全部为0，即出现了$\frac{0}{0}$的情况！无法计算。

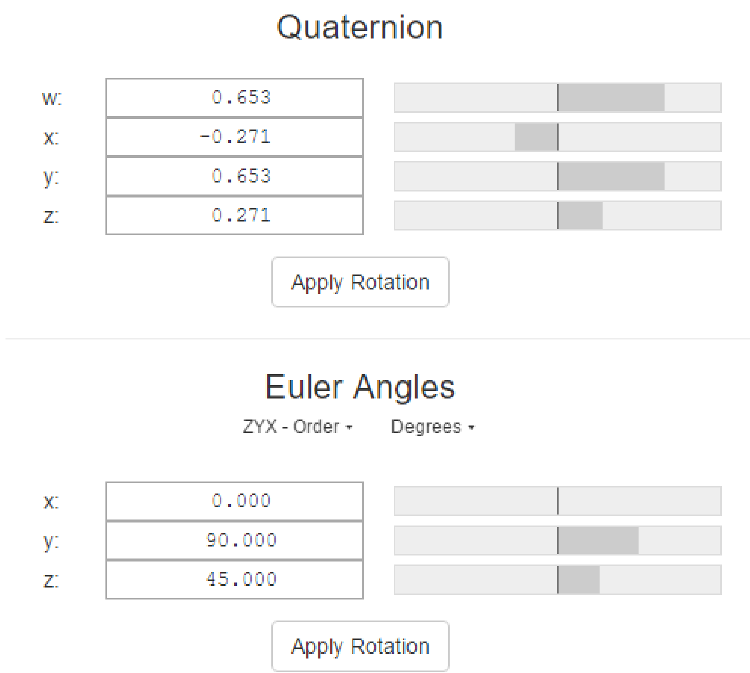
$\beta = \pi/2$时，$\sin\frac{\beta}{2} = \cos\frac{\beta}{2} = 0.707$，将其带入公式中有

$$q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} \begin{bmatrix} 0.707(\cos\frac{\alpha}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\sin\frac{\gamma}{2}) \\ 0.707(\sin\frac{\alpha}{2}\cos\frac{\gamma}{2} - \cos\frac{\alpha}{2}\sin\frac{\gamma}{2}) \\ 0.707(\cos\frac{\alpha}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\sin\frac{\gamma}{2}) \\ 0.707(\cos\frac{\alpha}{2}\sin\frac{\gamma}{2} - \sin\frac{\alpha}{2}\cos\frac{\gamma}{2}) \end{bmatrix} = \begin{bmatrix} 0.707\cos\frac{\alpha-\gamma}{2} \\ 0.707\sin\frac{\alpha-\gamma}{2} \\ 0.707\cos\frac{\alpha-\gamma}{2} \\ 0.707\sin\frac{\alpha-\gamma}{2} \end{bmatrix}$$

则$\frac{x}{w} = \frac{z}{y} = \tan\frac{\alpha-\gamma}{2}$，于是有

$$\alpha - \gamma = 2 \cdot atan2(x, w)$$

通常令$\alpha = 0$，这时$\gamma = -2 \cdot atan2(x, w)$。可以进行验证：当四元数为(w,x,y,z)=(0.653,-0.271,0.653,0.271)时，根据这些规则计算出来的ZYX欧拉角为α=0°，β=90°，γ=45°



当俯仰角为-90°，即机头竖直向下时的情况也与之类似，可以推导出奇异姿态时的计算公式。比较完整的四元数转欧拉角（Z-Y-X order）的代码如下：

```
CameraSpacePoint QuaternionToEuler(Vector4 q) // Z-Y-X Euler angles
{
    CameraSpacePoint euler = { 0 };
    const double Epsilon = 0.0009765625f;
    const double Threshold = 0.5f - Epsilon;

    double TEST = q.w*q.y - q.x*q.z;

    if (TEST < -Threshold || TEST > Threshold) // 奇异姿态,俯仰角为±90°
    {
        int sign = Sign(TEST);

        euler.Z = -2 * sign * (double)atan2(q.x, q.w); // yaw

        euler.Y = sign * (PI / 2.0); // pitch

        euler.X = 0; // roll

    }
    else
    {
        euler.X = atan2(2 * (q.y*q.z + q.w*q.x), q.w*q.w - q.x*q.x - q.y*q.y + q.z*q.z);
        euler.Y = asin(-2 * (q.x*q.z - q.w*q.y));
        euler.Z = atan2(2 * (q.x*q.y + q.w*q.z), q.w*q.w + q.x*q.x - q.y*q.y - q.z*q.z);
    }

    return euler;
}
```

在DirectXMath Library中有许多与刚体姿态变换相关的函数可以直接调用：

- 四元数乘法：XMQuaternionMultiply method --Computes the product of two quaternions.
- 旋转矩阵转四元数：XMQuaternionRotationMatrix method --Computes a rotation quaternion from a rotation matrix.

- 四元数转旋转矩阵：XMMatrixRotationQuaternion method -- Builds a rotation matrix from a quaternion.
- 欧拉角转四元数：XMQuaternionRotationRollPitchYaw method --Computes a rotation quaternion based on the pitch, yaw, and roll (Euler angles).
- 四元数转Axis-Angle：XMQuaternionToAxisAngle method --Computes an axis and angle of rotation about that axis for a given quaternion.
- 欧拉角转旋转矩阵：XMMatrixRotationRollPitchYaw method --Builds a rotation matrix based on a given pitch, yaw, and roll (Euler angles).
- Axis-Angle转旋转矩阵：XMMatrixRotationAxis method --Builds a matrix that rotates around an arbitrary axis.
- 构造绕X/Y/Z轴的旋转矩阵：XMMatrixRotationX method --Builds a matrix that rotates around the x-axis.(Angles are measured clockwise when looking along the rotation axis toward the origin)

下面的代码中坐标系绕X轴旋转90°（注意这里不是按照右手定则的方向，而是沿着坐标轴向原点看过去以顺时针方式旋转，因此与传统的右手定则刚好方向相反），来进行变换：

```
#include "stdafx.h"
#include<iostream>

#include <DirectXMath.h>
using namespace DirectX;

#define PI 3.1415926

int _tmain(int argc, _TCHAR* argv[])
{
    //------------------Computes the product of two quaternions.
    XMVECTOR q1 = XMVectorSet(1, 1, 3, 2);
    XMVECTOR q2 = XMVectorSet(1, 1, 0, 2);
    XMVECTOR q3 = XMVectorSet(1, 1, 1, 1);

    XMVECTOR result = XMQuaternionMultiply(XMQuaternionMultiply(q3, q2), q1);   // Returns tl

    std::cout << "Quaternion Multiply:" << std::endl;
    std::cout << XMVectorGetX(result) << "," << XMVectorGetY(result) << "," << XMVectorGetZ(


    //-----------------Computes a rotation quaternion based on the pitch, yaw, and roll (Eu
    float pitch = 90.0 * PI / 180.0; // Angle of rotation around the x-axis, in radians.
    float yaw = 0;                   // Angle of rotation around the y-axis, in radians.
    float roll = 0;                   // Angle of rotation around the z - axis, in radians

    result = XMQuaternionRotationRollPitchYaw(pitch, yaw, roll);

    std::cout << "RPY/Euler angles to Quaternion:" << std::endl;
    std::cout << XMVectorGetX(result) << "," << XMVectorGetY(result) << "," << XMVectorGetZ(


    //----------------Computes a rotation quaternion from a rotation matrix.
    float matrix[16] = { 1, 0, 0, 0,   0, 0, 1, 0,   0, -1, 0, 0,   0, 0, 0, 1 };
    XMMATRIX trans(matrix); // Initializes a new instance of the XMMATRIX structure from a s

    result = XMQuaternionRotationMatrix(trans); // This function only uses the upper 3x3 por

    std::cout << "Matrix to Quaternion:" << std::endl;
    std::cout << XMVectorGetX(result) << "," << XMVectorGetY(result) << "," << XMVectorGetZ(


    //----------------Builds a rotation matrix from a quaternion.
    trans = XMMatrixRotationQuaternion(result);

    XMFLOAT3X3 fView;
    XMStoreFloat3x3(&fView, trans); // Stores an XMMATRIX in an XMFLOAT3X3
    std::cout << "Quaternion to Matrix:" << std::endl;
    std::cout << fView._11 << "," << fView._12 << "," << fView._13 << std::endl
```

```
        << fView._21 << "," << fView._22 << "," << fView._23 << std::endl
        << fView._31 << "," << fView._32 << "," << fView._33 << std::endl << std::endl;



    //-----------------Computes an axis and angle of rotation about that axis for a given qua
    float Angle = 0;
    XMVECTOR Axis;

    XMQuaternionToAxisAngle(&Axis, &Angle, result);

    Axis = XMVector3Normalize(Axis); // Returns the normalized version of a 3D vector
    std::cout << "Quaternion to Axis-Angle:" << std::endl;
    std::cout << "Axis: " << XMVectorGetX(Axis) << "," << XMVectorGetY(Axis) << "," << XMVect
    std::cout << "Angle: " << Angle*180.0 / PI << std::endl << std::endl;



    //-----------------Builds a matrix that rotates around an arbitrary axis.
    Angle = 90.0 * PI / 180.0;

    trans = XMMatrixRotationAxis(Axis, Angle);

    XMStoreFloat3x3(&fView, trans); // Stores an XMMATRIX in an XMFLOAT3X3
    std::cout << "Axis-Angle to Matrix:" << std::endl;
    std::cout << fView._11 << "," << fView._12 << "," << fView._13 << std::endl
              << fView._21 << "," << fView._22 << "," << fView._23 << std::endl
              << fView._31 << "," << fView._32 << "," << fView._33 << std::endl << std::endl;



    //-----------------Builds a rotation matrix based on a given pitch, yaw, and roll(Euler a
    trans = XMMatrixRotationRollPitchYaw(pitch, yaw, roll);

    XMStoreFloat3x3(&fView, trans); // Stores an XMMATRIX in an XMFLOAT3X3
    std::cout << "RPY/Euler angles to Matrix:" << std::endl;
    std::cout << fView._11 << "," << fView._12 << "," << fView._13 << std::endl
        << fView._21 << "," << fView._22 << "," << fView._23 << std::endl
        << fView._31 << "," << fView._32 << "," << fView._33 << std::endl << std::endl;


    //-----------------Builds a matrix that rotates around the x - axis.
    trans = XMMatrixRotationX(Angle); // Angles are measured clockwise when looking along the

    XMStoreFloat3x3(&fView, trans); // Stores an XMMATRIX in an XMFLOAT3X3
    std::cout << "Builds a matrix that rotates around the x-axis.:" << std::endl;
    std::cout << fView._11 << "," << fView._12 << "," << fView._13 << std::endl
        << fView._21 << "," << fView._22 << "," << fView._23 << std::endl
        << fView._31 << "," << fView._32 << "," << fView._33 << std::endl << std::endl;

    return 0;
}
```

结果如下图所示:

```
RPY/Euler angles to Quaternion:
0.707107,0,0,0.707107

Matrix to Quaternion:
0.707107,0,0,0.707107

Quaternion to Matrix:
1,0,0
0,5.96046e-008,1
0,-1,5.96046e-008

Quaternion to Axis-Angle:
Axis: 1,0,0
Angle: 90

Axis-Angle to Matrix:
1,0,0
0,5.96046e-008,1
0,-1,5.96046e-008

RPY/Euler angles to Matrix:
1,0,0
0,1.78814e-007,1
0,-1,1.78814e-007

Builds a matrix that rotates around the x-axis.:
1,0,0
0,5.96046e-008,1
0,-1,5.96046e-008
```

参考：

quaternions.online

DirectXMath Library Quaternion Functions

Convert quaternion to euler rotations

Conversion between quaternions and Euler angles

Maths - Conversion Quaternion to Euler

Coordinate Transformations in Robotics—MATLAB

Introduction to Robotics - Mechanics and Control. Chapter 2 Spatial descriptions and transformations

评论列表

#1楼 2019-03-14 10:20 飞控渣渣灰 回复 引用

6666666666+

支持(0) 反对(0)

#2楼 2019-04-18 21:11 尚修能的技术博客 回复 引用

博主，您的文章写的很好，谢谢您

支持(1) 反对(0)

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]

相关博文：
· 旋转矩阵、欧拉角、四元数理论及其转换关系
· 四元数与欧拉角之间的转换
· 四元数和欧拉角，轴角对之间的相互转化
· 学习笔记—四元数与欧拉角之间的转换
· 三维旋转：旋转矩阵，欧拉角，四元数
» 更多推荐...

最新 IT 新闻:
· 飞利浦要出售家电业务：格力九阳考虑竞购
· 华为发布Freebuds Pro无线耳机：全球首发动态降噪 1099元起
· 美国禁令即将生效：三星显示器申请许可证 以求向华为供货
· 1TB 799元！长江存储致钛SC001 SATA固态硬盘图赏
· 不用打针：中国造的全球第一个鼻喷新冠疫苗进入临床试验
» 更多新闻...