从零开始一起学习SLAM | 三维空间刚体的旋转

刚体，顾名思义，是指本身不会在运动过程中产生形变的物体，如相机的运动就是刚体运动，运动过程中同一个向量的长度和夹角都不会发生变化。刚体变换也称为欧式变换。

视觉SLAM中使用的相机就是典型的刚体，相机一般通过人手持、机载（安装在机器人上）、车载（固定在车辆上）等方式在三维空间内运动，形式包括旋转、平移、缩放、切变等。其中，刚体在三维空间中最重要的运动形式就是旋转。那么刚体的旋转如何量化表达呢?

三维空间中刚体的旋转表示

三维空间中刚体的旋转总共有4种表示方法，高翔的十四讲中的第3讲比较详细的讲解了。本文提炼中最重要的内容，并加上实际使用过程中的经验总结进行了归纳。下面按照重要顺序分别进行介绍。

1 旋转矩阵

1、SLAM编程中使用比较频繁。需要重点掌握。

2、旋转矩阵不是一般矩阵，它有比较强的约束条件。旋转矩阵R具有**正交性，R和R的转置的乘积是单位阵，且行列式值为1。**

3、旋转矩阵R的逆矩阵表示了一个和R相反的旋转。

4、旋转矩阵R通常和平移向量t一起组成齐次的变换矩阵T，描述了欧氏坐标变换。**引入齐次坐标是为了可以方便的描述连续的欧氏变换**，这个在上一篇文章《从零开始一起学习SLAM | 为什么要用齐次坐标？》中有讲解。

5、冗余。用9个元素表示3个自由度的旋转，比较冗余。

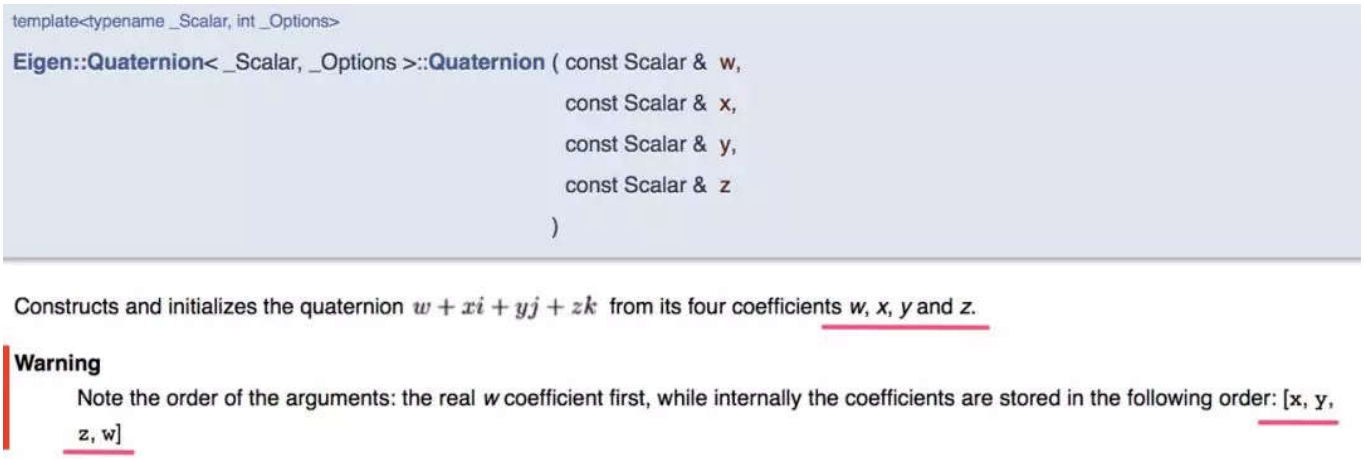
2 四元数

1、SLAM编程中使用频繁程度接近旋转矩阵。稍微有点抽象，不太直观，但是一定得掌

握。

2、四元数由一个实部和三个虚部组成，是一种**非常紧凑、没有奇异**的表达方式。

3、编程时候很多坑，必须注意。首先，一定要**注意**四元素定义中**实部虚部**和打印系数的**顺序**不同，很容易出错！



```
template<typename _Scalar, int _Options>
Eigen::Quaternion< _Scalar, _Options >::Quaternion ( const Scalar & w,
                                                     const Scalar & x,
                                                     const Scalar & y,
                                                     const Scalar & z
                                                   )
```

Constructs and initializes the quaternion $w + xi + yj + zk$ from its four coefficients *w*, *x*, *y* and *z*.

Warning

Note the order of the arguments: the real *w* coefficient first, while internally the coefficients are stored in the following order: [x, y, z, w]

其次，单位四元素才能描述旋转，所以四元素使用前必须**归一化**：q.normalize()。

3 旋转向量

1、用一个旋转轴n和旋转角θ来描述一个旋转，所以也称轴角。不过很明显，因为旋转角度有一定的周期性（360°一圈），所以这种表达方式具有奇异性。

2、从旋转向量到旋转矩阵的转换过程称为 罗德里格斯公式。这个推导比较麻烦，否则也不会有一个专属的名字了。OpenCV和MATLAB中都有专门的罗德里格斯函数。

3、旋转向量本身没什么出彩的，不过**旋转向量和旋转矩阵的转换关系，其实对应于李代数和李群的映射**，这对于后面理解李代数很有帮助。

4 欧拉角

1、把一次旋转分解成3次绕不同坐标轴的旋转，比如航空领域经常使用的“偏航-俯仰-滚转”（yaw，pitch，roll）就是一种欧拉角。该表达方式最大的优势就是直观。

2、欧拉角在SLAM中用的很少，原因是它的一个致命缺点：**万向锁**。也就是在俯仰角为±90°时，第一次和第3次旋转使用的是同一个坐标轴，会丢失一个自由度，引起奇异性。事实上，想要表达三维旋转，至少需要4个变量。

了解了四种旋转的表达方式，那么编程时如何使用呢？

矩阵线性代数运算库Eigen

事实上，上述几种旋转的表达方式在一个第三方库Eigen中已经定义好啦。Eigen是一个C++开源线性代数库，安装非常方便，Ubuntu下一行代码即可搞定：
sudo apt-get install libeigen3-dev

Eigen在SLAM编程中是必备基础，必须熟练编程。关于Eigen，主要有以下几点需要强调或注意。

1、Eigen库不同于一般的库，它**只有头文件**，**没有**.so和 .a那样的**二进制库文件**，所以在CMakeLists.txt里只需要添加头文件路径，并不需要使用 target_link_libraries 将程序链接到库上。

2、Eigen以**矩阵为基本数据单元**，在Eigen中，所有的矩阵和向量都是Matrix模板类的对象，Matrix一般使用3个参数：数据类型、行数、列数

Eigen::Matrix<typename Scalar, int rowsNum, int colsNum>

而向量只是一种特殊的矩阵（一行或者一列）。同时，Eigen通过typedef 预先定义好了很多内置类型，如下，我们可以看到底层仍然是Eigen::Matrix

typedef Eigen::Matrix<float, 4, 4> Matrix4f;
typedef Eigen::Matrix<float, 3, 1> Vector3f;

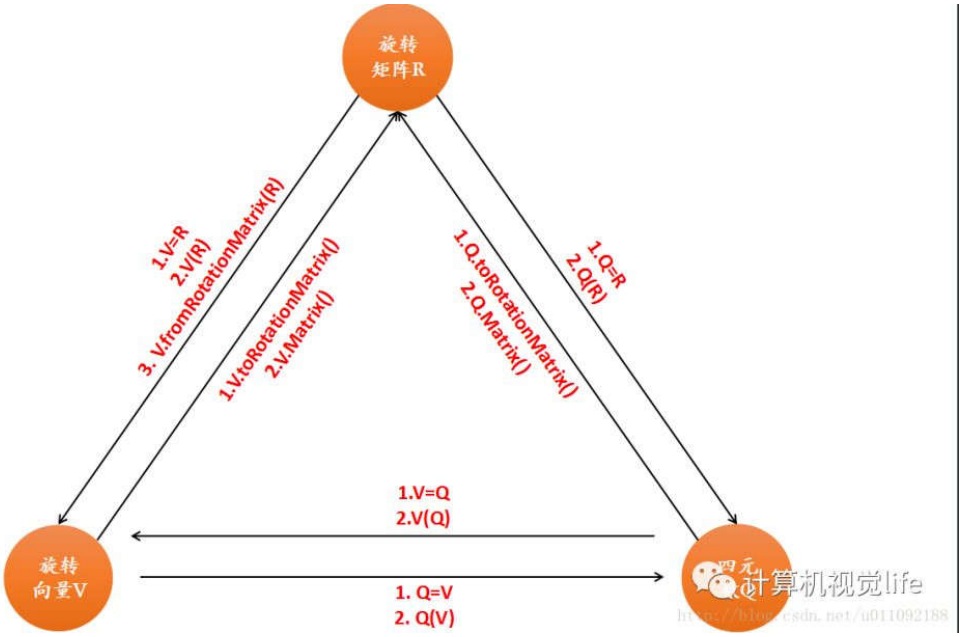
3、为了提高效率，对于已知大小的矩阵，使用时需要**指定矩阵的大小和类型**。如果不确定矩阵的大小，可以使用**动态矩阵**Eigen::Dynamic
Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> matrix_dynamic;

4、Eigen在数据类型方面“很傻很天真”。什么意思呢？就是使用Eigen时**操作数据类型必须完全一致，不能进行自动类型提升**。比如C++中，float类型加上double类型变量不会报错，编译器会自动将结果提升为double。但是在Eigen中float类型矩阵和double类型矩阵不能直接相加，必须统一为float或者double，否则会报错。

5、Eigen除了空间几何变换外，还提供了大量**矩阵分解、稀疏线性方程求解**等函数，非常方便。学习Eigen最好的方式就是官网：

http://eigen.tuxfamily.org/dox/
有非常多的示例参考。

上述四种旋转表达方式是可以相互转化的。在Eigen中它们之间的转化非常的方便。下图是我看的别人总结的旋转矩阵、四元素、旋转向量之间的相互转化图：



作 业

题目1：
已知旋转矩阵定义是沿着Z轴旋转45°。请按照该定义初始化旋转向量、旋转矩阵、四元数、欧拉角。请编程实现：

1、以上四种表达方式的相互转换关系并输出，并参考给出的结果验证是否正确。
2、假设平移向量为（1,2,3）,请输出旋转矩阵和该平移矩阵构成的欧式变换矩阵，并根据欧式变换矩阵提取旋转向量及平移向量。

本程序学习目标：
1、学习eigen中刚体旋转的四种表达方式，熟悉他们之间的相互转换关系
2、熟悉旋转平移和欧式变换矩阵的相互转换关系

以下是参考的编程框架：

```
/**************************
 * 题目：已知旋转矩阵定义是沿着Z轴旋转45°。
 * 按照该定义初始化旋转向量、旋转矩阵、四元数、欧拉角。请编程实现：
 * 1、以上四种表达方式的相互转换关系并输出，看看是否正确
 * 2、假设平移向量为（1,2,3）,请输出旋转矩阵和该平移矩阵构成的欧式变换矩阵，并根据欧式变换矩阵提取旋转向量及平移向量
 *
 * 本程序学习目标：
 * 学习eigen中刚体旋转的四种表达方式，熟悉他们之间的相互转换关系
 * 熟悉旋转平移和欧式变换矩阵的相互转换关系
 *
 * 作者：公众号：计算机视觉life。发布于公众号旗下知识星球：从零开始学习SLAM
```

```cpp
 * 时间: 2018.10
**************************/
#include <iostream>
#include <cmath>
#include <Eigen/Core>
#include <Eigen/Geometry>
using namespace std;

int main ( int argc, char** argv )
{
    // ----------  初始化 ----------//

    // 旋转向量（轴角）：沿Z轴旋转45°
    Eigen::AngleAxisd rotation_vector ( M_PI/4, Eigen::Vector3d ( 0,0,1 ) );
    cout<<"rotation_vector axis = \n" << rotation_vector.axis() <<"\n rotation_vector angle = "<< rotation_vector.angle()<<endl;

    //旋转矩阵：沿Z轴旋转45°
    Eigen::Matrix3d rotation_matrix = Eigen::Matrix3d::Identity();
    rotation_matrix <<  0.707, -0.707,      0,
            0.707,  0.707,      0,
            0,      0,          1;
    cout<<"rotation matrix =\n"<<rotation_matrix <<endl;

    // 四元数：沿Z轴旋转45°
    Eigen::Quaterniond quat = Eigen::Quaterniond(0, 0, 0.383, 0.924);
    cout<<"四元数输出方法1：quaternion = \n"<<quat.coeffs() <<endl;   // 请注意coeffs的顺序是(x,y,z,w),w为实部，前三者为虚部
    cout<<"四元数输出方法2：\n x = " << quat.x() << "\n y = " << quat.y() << "\n z = " << quat.z() << "\n w = " << quat.w() << endl;

    // 欧拉角：：沿Z轴旋转45°
    Eigen::Vector3d euler_angles = Eigen::Vector3d(M_PI/4, 0, 0);// ZYX顺序，即roll pitch yaw顺序
    cout<<"Euler: yaw pitch roll = "<<euler_angles.transpose()<<endl;


    //  相互转化关系
    // ----------  请在下面补充对应的代码 ----------
    // 旋转向量转化为其他形式
    cout<<"旋转向量转化为旋转矩阵方法1：rotation matrix =\n"<< <<endl;
    cout<<"旋转向量转化为旋转矩阵方法2：rotation matrix =\n"<< <<endl;

    quat =
    cout<<"旋转向量转化为四元数：quaternion =\n"<<  <<endl;  //coeffs的顺序是(x,y,z,w),w为实部，前三者为虚部


    // 旋转矩阵转化为其他形式

    cout<<"旋转矩阵转化为旋转向量：rotation_vector axis = \n" <<  <<"\n rotation_vector angle = "<< <<endl;
    //注意：fromRotationMatrix 参数只适用于旋转向量，不适用于四元数

    rotation_vector =
    cout<<"旋转矩阵直接给旋转向量赋值初始化：rotation_vector axis = \n" <<  <<"\n rotation_vector angle = "<< <<endl;

    euler_angles =
    cout<<"旋转矩阵转化为欧拉角：yaw pitch roll = "<< <<endl;

    quat =
    cout<<"旋转矩阵转化为四元数：quaternion =\n"<<  <<endl;

    // 四元数转化为其他形式
    rotation_vector =
    cout<<"四元数转化为旋转向量：rotation_vector axis = \n" <<  <<"\n rotation_vector angle = "<< r<<endl;

    rotation_matrix =
    cout<<"四元数转化为旋转矩阵方法1：rotation matrix =\n"<< <<endl;

    rotation_matrix =
    cout<<"四元数转化为旋转矩阵方法2：rotation matrix =\n"<< <<endl;


    //  欧氏变换矩阵

    // 欧氏变换矩阵使用 Eigen::Isometry，仿射变换用 Eigen::Affine3d，射影变换用 Eigen::Projective3d
    Eigen::Isometry3d T = Eigen::Isometry3d::Identity();                // 名称为3d，实质上是4 x 4的矩阵

    cout << "Transform matrix = \n" <<  <<endl;                // 变换矩阵需要用成员函数转一下输出

    cout << "欧氏变化矩阵提取旋转矩阵：rotation_matrix = \n" <<  << endl;
    cout << "欧氏变化矩阵提取平移向量：translation = \n" <<  << endl;

    // ----------  结束 ----------
    return 0;
}
```


题目2：

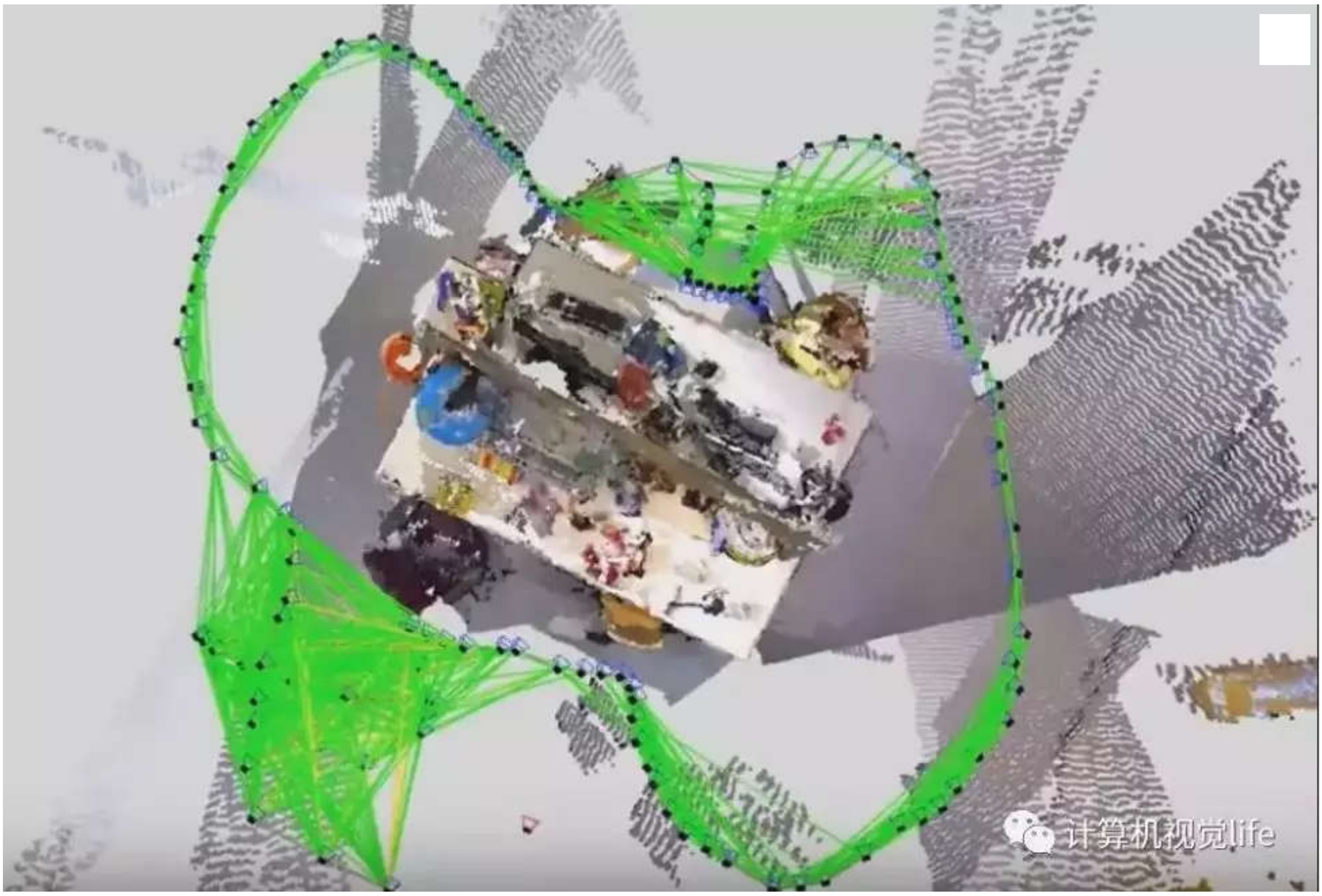
我们知道单位四元数q可以表达旋转。一个三维空间点可以用虚四元数p表示，用四元数 q 旋转点 p 的结果p'为:

$$p' = qpq^{-1}$$

证明：此时 p' 必定为虚四元数（实部为零）

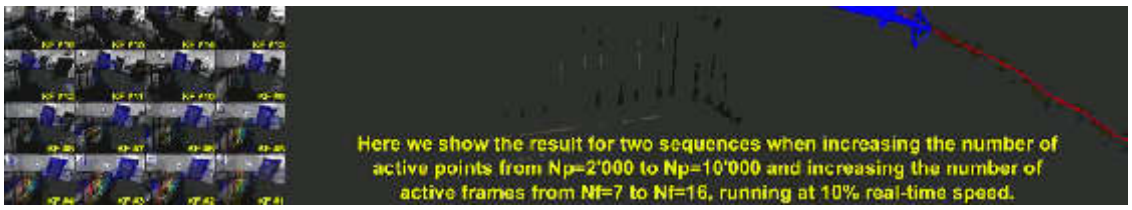
证明：此时 p 必定为虚四元数（实部为零）。

相关阅读

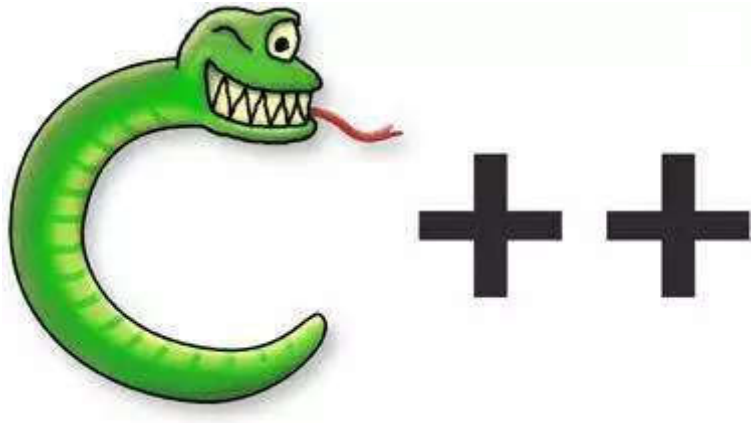


从零开始一起学习SLAM | 为什么要学SLAM?

从零开始一起学习SLAM | 学习SLAM到底需要学什么?



从零开始一起学习SLAM | SLAM有什么用?



从零开始一起学习SLAM | C++新特性要不要学?

从零开始一起学习SLAM | 为什么要用齐次坐标?

零基础小白，如何入门计算机视觉?