**eigen库使用方法**

首先，使用eigen库需要头文件包含：核心功能通常需要包含<Eigen/Core>头文件，它提供了基础的线性代数运算和数组操作，如矩阵和向量的定义与基本运算；对于几何变换相关的功能，需包含<Eigen/Geometry>头文件，其中包括平移、缩放、2D/3D 旋转变换、四元数、轴旋转变换等操作所需的类和函数。

而在矩阵与向量基础中

1.定义与初始化：

可以使用Matrix<数据类型, 行数, 列数>的模板形式定义矩阵，例如Eigen::Matrix<double, 3, 3> matrix3d;定义了一个3x3的双精度矩阵。Eigen 还提供了许多typedef定义的内置类型，如MatrixXd表示动态大小的双精度矩阵，Vector3d表示3x1的双精度向量。矩阵可以使用<<操作符进行赋值，如matrix3d << 1, 2, 3, 4, 5, 6, 7, 8, 9;按行依次为矩阵元素赋值.

2.矩阵运算 ：

常见的四则运算可直接通过+、-、\*、/等符号实现，但需注意操作数的类型和尺寸需匹配，Eigen 不支持自动类型转化 。求矩阵的转置使用transpose()函数，如matrix.transpose()。需要注意的是，该函数返回一个代理对象，若执行a = a.transpose()可能会出现别名问题，在 debug 模式且 assertions 打开时可自动检测到这种问题 。计算矩阵的迹使用trace()函数，行列式使用determinant()函数，逆矩阵使用inverse()函数。

3.访问元素与块操作 ：

可以使用()操作符访问矩阵和向量的元素，如matrix(i, j)表示访问矩阵matrix的第i行第j列元素，向量vector(i)表示访问向量vector的第i个元素。对于矩阵的块操作，block()函数可用于提取矩阵的子块，如matrix.block(i, j, rows, cols)表示提取从第i行第j列开始，行数为rows，列数为cols的子块。对于向量，segment()函数可提取连续的一段元素，如vector.segment(i, size)表示提取从第i个元素开始，长度为size的子向量。

**四元数、欧拉角、旋转矩阵、旋转向量之间的转换**

1.旋转向量 ：

使用Eigen::AngleAxisd类来表示旋转向量，通过指定旋转角度和旋转轴来初始化，如Eigen::AngleAxisd rotation\_vector(alpha, Eigen::Vector3d(x, y, z));，其中alpha是旋转角，(x, y, z)是旋转轴。

旋转向量转旋转矩阵可通过rotation\_vector.matrix()或rotation\_vector.toRotationMatrix()实现，得到Eigen::Matrix3d类型的旋转矩阵。

旋转向量转欧拉角使用rotation\_vector.matrix().eulerAngles(2, 1, 0)，按照 Z-Y-X 的顺序，即 roll-pitch-yaw 的顺序获取Eigen::Vector3d类型的欧拉角向量。

旋转向量转四元数可直接使用Eigen::Quaterniond quaternion(rotation\_vector)或Eigen::Quaterniond quaternion = rotation\_vector。

2.旋转矩阵 ：

可以使用Eigen::Matrix3d并通过赋值的方式初始化旋转矩阵，如Eigen::Matrix3d rotation\_matrix << x\_00, x\_01, x\_02, x\_10, x\_11, x\_12, x\_20, x\_21, x\_22;。旋转矩阵转旋转向量可以使用Eigen::AngleAxisd rotation\_vector(rotation\_matrix)或rotation\_vector = rotation\_matrix或rotation\_vector.fromRotationMatrix(rotation\_matrix)。旋转矩阵转欧拉角通过rotation\_matrix.eulerAngles(2, 1, 0)实现，按照 Z-Y-X 顺序转换为欧拉角向量。旋转矩阵转四元数使用Eigen::Quaterniond quaternion(rotation\_matrix)或Eigen::Quaterniond quaternion = rotation\_matrix。

3.欧拉角 ：

使用Eigen::Vector3d来初始化欧拉角，如Eigen::Vector3d eulerAngle(yaw, pitch, roll)，其中yaw、pitch、roll分别是绕 Z 轴、Y 轴、X 轴的旋转角度。欧拉角转旋转向量需先分别创建绕 X、Y、Z 轴的旋转向量，然后将它们相乘得到最终的旋转向量，如：

Eigen::AngleAxisd rollAngle(AngleAxisd(eulerAngle(2), Eigen::Vector3d::UnitX()));

Eigen::AngleAxisd pitchAngle(AngleAxisd(eulerAngle(1), Eigen::Vector3d::UnitY()));

Eigen::AngleAxisd yawAngle(AngleAxisd(eulerAngle(0), Eigen::Vector3d::UnitZ()));

Eigen::AngleAxisd rotation\_vector;

rotation\_vector = yawAngle \* pitchAngle \* rollAngle;

欧拉角转旋转矩阵与转旋转向量类似，先创建绕各轴的旋转向量，然后相乘得到旋转矩阵。欧拉角转四元数同样先创建绕各轴的旋转向量，相乘后得到四元数。

4.四元数 ：

使用Eigen::Quaterniond类来初始化四元数，可以通过直接赋值的方式，如Eigen::Quaterniond quaternion(w, x, y, z)，其中w是实部，(x, y, z)是虚部。四元数转旋转向量使用Eigen::AngleAxisd rotation\_vector(quaternion)或rotation\_vector = quaternion。四元数转旋转矩阵通过quaternion.matrix()或quaternion.toRotationMatrix()实现，得到Eigen::Matrix3d类型的旋转矩阵。四元数转欧拉角使用quaternion.matrix().eulerAngles(2, 1, 0)，按照 Z-Y-X 顺序转换为欧拉角向量。

1. 欧氏变换矩阵：使用Eigen::Isometry3d来表示三维空间中的刚体变换，它是一个4×4的矩阵。可以通过Isometry3d::Identity()创建单位变换矩阵，然后使用rotate()函数进行旋转，pretranslate()函数进行平移，如Eigen::Isometry3d T = Eigen::Isometry3d::Identity(); T.rotate(rotation\_vector); T.pretranslate(Eigen::Vector3d(1, 3, 4));

转换代码及输出：#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iostream>

using namespace std;

#include <Eigen/Core>

#include <Eigen/Geometry>

using namespace Eigen;

int main(int argc, char \*\*argv) {

// 3D旋转的变换矩阵使用Eigen::Isometry3d

Isometry3d T = Isometry3d::Identity();

T.rotate(AngleAxisd(M\_PI/4, Vector3d(0, 1, 0))); // 沿y轴旋转45°

T.pretranslate(Vector3d(1, 3, 4)); // 把平移向量设成(1,3,4)

cout.precision(3);

Vector3d v(1, 0, 0);

Vector3d v\_rotated = T \* v; // 用变换矩阵进行坐标变换

cout << "Transform matrix = " << T.matrix() << endl;

cout << "(1,0,0) after rotation = " << v\_rotated.transpose() << endl;

// 四元数

Quaterniond q = Quaterniond(T.rotation());

cout << "quaternion from rotation matrix = " << q.coeffs().transpose() << endl;

// 使用四元数旋转一个向量，使用重载的乘法即可

v\_rotated = q \* v; // 注意数学上是qvq^{-1}

cout << "(1,0,0) after rotation = " << v\_rotated.transpose() << endl;

// 用常规向量乘法表示，则计算如下

cout << "should be equal to " << (q \* Quaterniond(0, 1, 0, 0) \* q.inverse()).coeffs().transpose() << endl;

return 0;

}

