Turtorial1_Eigen库的安装和使用

0.环境安装

虚拟机or双系统 安装 ubuntu18.04 or ubuntu20.04 （不建议安装高版本，可能bug较多）

1.Eigen库的介绍

Eigen是一个C++语言中的开源的模板库，支持线性代数的运算，包括向量运算，矩阵运算，数值分析等相关算法。

2.linux下Eigen库的安装

- Terminal 软件源安装

```
# 直接在apt软件包中安装
sudo apt-get install libeigen3-dev
# 这种安装方式有一个缺点，因为apt包更新比较慢，安装的版本可能不是最新版，那么之后的一些依赖于eigen的库可能没有办法使用(如Sophus库要求必须选用3.3以上版本的eigen包)，可以使用指令来查看apt包中eigen的版本
apt show libeigen3-dev
```

- 源码安装（需提前安装cmake、gcc、g++ [参考链接]）

在[官网](Eigen)中下载指定安装包源码，之后提取压缩包，进入提取出来的文件夹，在终端中打开，编译安装到/usr/include路径（默认）

```
mkdir build
cd build
cmake ..
make -j8 #-j8表示用8核加速编译
sudo make install
```

- 安装多版本eigen库
- CMake Tutorial

```
# 选择特定版本的eigen
```

```
cd Eigen3.3.7
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr/include/eigen337 ..     # 编译预选项，表示安装到该路径下
make
sudo make install
# CMakelist.txt 中修改
find_package(Eigen3 REQUIRED) -> # 修改为指定版本的头文件路径
include_directories("/usr/include/eigen337/include/eigen3")
```

3.基于eigen库的旋转矩阵、旋转向量、四元数的演示（参考VSLAM14 p63内容）

```
#include <iostream>
#include <cmath>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Geometry>
using namespace Eigen;
using namespace std;

int main() {

    // Eigen/Geometry 提供了各种旋转和平移的方式
    // 3D 旋转矩阵 可直接使用 Matrix3d 或 Matrix3f
    Matrix3d rotation_matrix = Matrix3d::Identity(); // 初始化单位矩阵
    cout << "Identity Matrix = \n" << rotation_matrix << endl;

    // rotation vector -> AngleAxis (角轴)，底层不是Matrix ，但运算可以当成矩阵运算（有运算符重载）
    // 表示沿 Z 轴旋转 45 度
    AngleAxisd rotation_vector(M_PI / 4, Vector3d(0,0,1));
    // Convert rotation_vector to rotation_matrix
    cout.precision(3);
    cout << "rotation_matrix = \n" << rotation_vector.matrix() << endl;

    // or assignment directly
    rotation_matrix = rotation_vector.toRotationMatrix();
    cout << "to_Rotation_Matrix = \n" << rotation_matrix << endl;

```

```cpp
    // 用 AngleAxis 可以进行坐标变换
    Vector3d v(1,0,0); // 初始向量
    Vector3d v_rotated = rotation_vector * v;
    cout << "(1,0,0) after rotation (by angle axis) : " << v_rotated.transpose() << endl;

    // or using rotation_matrix
    v_rotated = rotation_matrix * v;
    cout << "(1,0,0) after rotation (by rotation_matrix) : " << v_rotated.transpose() << endl;


    // 欧拉角（一般不用）0 : 可以将旋转矩阵直接转换为欧拉角
    // ZYX顺序，即 roll pitch yaw 滚转角 俯仰角 偏航角
    Vector3d euler_angles = rotation_matrix.eulerAngles(2,1,0);
    cout << "yaw pitch roll = " << euler_angles.transpose() << endl;

    // 欧式变换矩阵 Eigen::Isometry
    Isometry3d T = Isometry3d::Identity();
    cout << "Transform Matrix : \n" << T.matrix() << endl;
    /**
     * 或者使用旋转矩阵初始化 T.rotate(rotation_matrix);
     * 或者使用四元数初始化  T.rotate(q);
     * 或者使用旋转矩阵初始化 T.rotate(rotation_vector);
     */
    T.rotate(rotation_vector);


    // 初始化平移向量 t
    T.pretranslate(Vector3d(1,3,4));
    cout << "Transform matrix = \n" << T.matrix() << endl;

    // use Transform Matrix 进行坐标变换
    Vector3d v_transformed = T * v;
    cout << "v tranformed = " << v_transformed.transpose() << endl;

    /**
     * 放射变换&射影变换 -> Eigen::Affine3d & Eigen::Projective3d
     */

    // 四元数，可把AngleAxis赋值给四元数，反之亦然
    Quaterniond q = Quaterniond(rotation_vector);
```

```
    cout << "quaternion from rotation vector = " << q.coeffs().transpose() << endl;
    // coeffs 的顺序是 (x,y,z,w) w 是实部，前三者为虚部
    // use rotation_matrix
    q = Quaterniond (rotation_matrix);
    cout << "quaternion from rotation matrix = " << q.coeffs().transpose() << endl;

    // 使用四元数旋转向量
    v_rotated = q * v; // result = qvq^{-1}
    cout << "(1,0,0) after rotation = " << v_rotated.transpose() << endl;

    // 常规乘法 result = qvq^{-1} -> 取出虚部
    Quaterniond result = q * Quaterniond(0,1,0,0) * q.inverse();
    cout << "should be equal to " << result.coeffs().transpose() << endl;

    return 0;
}

# 输出
Identity Matrix =
1 0 0
0 1 0
0 0 1
rotation_matrix =
 0.707 -0.707      0
 0.707  0.707      0
     0      0      1
to_Rotation_Matrix =
 0.707 -0.707      0
 0.707  0.707      0
     0      0      1
(1,0,0) after rotation (by angle axis) : 0.707 0.707     0
(1,0,0) after rotation (by rotation_matrix) : 0.707 0.707     0
yaw pitch roll = 0.785    -0     0
Transform Matrix :
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Transform matrix =
 0.707 -0.707      0      1
```

```
 0.707  0.707      0      3
     0      0      1      4
     0      0      0      1
v tranformed = 1.71 3.71    4
quaternion from rotation vector =     0     0 0.383 0.924
quaternion from rotation matrix =     0     0 0.383 0.924
(1,0,0) after rotation = 0.707 0.707     0
should be equal to 0.707 0.707     0     0
```