题目一：补充直接线性方法的里程计标定模块代码；

代码：

*//TODO:*

*//求解得到两帧数据之间的位姿差*

*//即求解当前位姿　在　上一时刻　坐标系中的坐标*

Eigen::Vector3d cal\_delta\_distence(Eigen::Vector3d *odom\_pose*)

{

    Eigen::Vector3d d\_pos; *//return value*

    now\_pos = odom\_pose;

*//TODO:*

    Eigen::Matrix3d TOB\_now;

    TOB\_now << cos(now\_pos(2)), -sin(now\_pos(2)), now\_pos(0),

        sin(now\_pos(2)), cos(now\_pos(2)), now\_pos(1),

        0, 0, 1;

    Eigen::Matrix3d TOB\_last;

    TOB\_last << cos(last\_pos(2)), -sin(last\_pos(2)), last\_pos(0),

        sin(last\_pos(2)), cos(last\_pos(2)), last\_pos(1),

        0, 0, 1;

    Eigen::Matrix3d TBO\_last = TOB\_last.inverse();

    Eigen::Matrix3d Tlast\_now = TBO\_last \* TOB\_now;

    d\_pos[0] = Tlast\_now(0, 2);

    d\_pos[1] = Tlast\_now(1, 2);

    d\_pos[2] = atan2(Tlast\_now(1, 0), Tlast\_now(0, 0));

*//end of TODO:*

    return d\_pos;

}

*TODO:*

*构建最小二乘需要的超定方程组*

*Ax = b*

*\*/*

bool OdomCalib::Add\_Data(Eigen::Vector3d *Odom*, Eigen::Vector3d *scan*)

{

    if (now\_len < INT\_MAX) {

*//TODO: 构建超定方程组*

        Eigen::Matrix<double, 1, 3> odom\_data;

        odom\_data(0, 0) = Odom[0];

        odom\_data(0, 1) = Odom[1];

        odom\_data(0, 2) = Odom[2];

        for (size\_t i = 0; i < 3; i++) {

            A.block<1, 3>(now\_len \* 3 + i, 3 \* i) = odom\_data;

        }

        b.block<3, 1>(now\_len \* 3, 0) = scan;

*//end of TODO*

        now\_len++;

        return true;

    } else {

        return false;

    }

}

*/\**

*\* TODO:*

*\* 求解线性最小二乘Ax=b*

*\* 返回得到的矫正矩阵*

*\*/*

Eigen::Matrix3d OdomCalib::Solve()

{

    Eigen::Matrix3d correct\_matrix;

*//TODO: 求解线性最小二乘*

    Eigen::Matrix<double, 9, 1> params;

*// params = (A.transpose() \* A).inverse() \* A.transpose() \* b;*

    params = A.bdcSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(b);

    std::cout << params.transpose() << std::endl;

    for (size\_t i = 0; i < 3; i++) {

        correct\_matrix.block<1, 3>(i, 0) = params.block<3, 1>(i \* 3, 0).transpose();

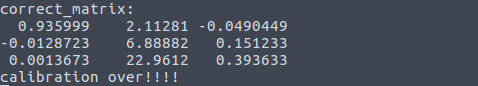
    }

*//end of TODO*

    return correct\_matrix;

}

终端输出结果：

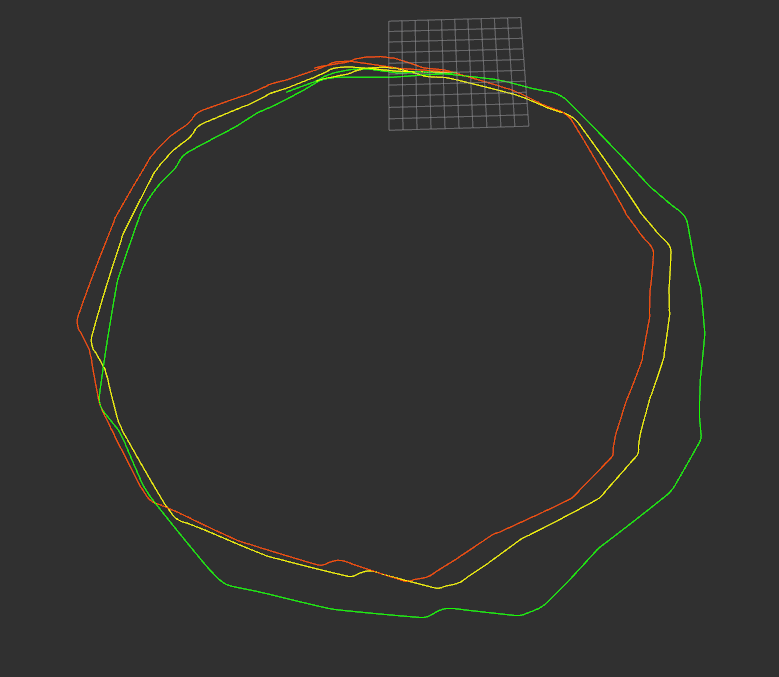


RVIZ轨迹效果：

红色：激光雷达轨迹

绿色：里程计轨迹

黄色：矫正后的轨迹



题目二：补充基于模型方法的里程计标定模块代码

代码：

*// 填充A, b矩阵*

*//TODO: (3~5 lines)*

            A(id\_s, 0) = w\_Lt;

            A(id\_s, 1) = w\_Rt;

            b[id\_s] = s\_th;

*//end of TODO*

*// 进行最小二乘求解*

    Eigen::Vector2d J21J22;

*//TODO: (1~2 lines)*

    J21J22 = A.bdcSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(b);

*//end of TODO*

*// 填充C, S矩阵*

*//TODO: (4~5 lines)*

            C[id\_s \* 2] = cx;

            C[id\_s \* 2 + 1] = cy;

            S[id\_s \* 2] = s\_x;

            S[id\_s \* 2 + 1] = s\_y;

*//end of TODO*

*//TODO: (3~5 lines)*

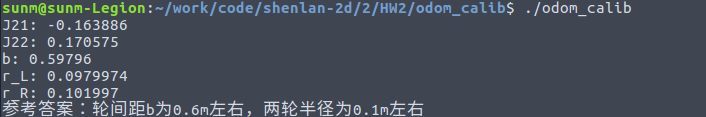
    b\_wheel = C.colPivHouseholderQr().solve(S)[0];

    r\_L = -J21 \* b\_wheel;

    r\_R = J22 \* b\_wheel;

*//end of TODO*

终端输出结果：



可见输出结果与答案很接近

题目三： 通过互联网总结学习线性方程组 Ax=b 的求解方法，回答以下问题：（2 分）

（1）对于该类问题，你都知道哪几种求解方法？

（2）各方法的优缺点有哪些？分别在什么条件下较常被使用？

（1）

直接求解

LU分解法，QR分解法，SVD（奇异值分解）、特征值分解

（2）

直接求解对于小矩阵可以，但是对于维度高的矩阵，运算起来效率很低，在大矩阵计算的时候不使用。

LU方法： Ax=b -> A'Ax=A'b -> x= 1/(A'A)\*A'\*b

'符号代表转制。在这种算法下，就可以用到LU分解（具体的说是LU分解的特殊情况 Cholesky factorization），把A'A分解后求逆然后进行计算，但是这种算法的缺点很明显：首先，A'A有时候不可逆，不能得出结果，其次，即便可逆，A'A数值稳定性不好，会造成误差。

SVD比QR数值稳定性更好，但是速度更慢

总结：LU需要A可逆的条件，QR速度快但是没有SVD稳定，SVD是目前求解最小二乘最好的矩阵分解法。一般在时间效率允许的情况下，选择SVD分解

4. 简答题，开放性答案：设计里程计与激光雷达外参标定方法。（2 分）

题目四： 我们一般把传感器内自身要调节的参数称为内参，比如前面作业中里程计模型的两轮间距与两个轮子的半径。把传感器之间的信息称为外参，比如里程计与激光雷达之间的时间延迟，位姿变换等。请你选用直接线性方法或基于模型的方法，设计一套激光雷达与里程计外参的标定方法，并回答以下问题：

（1）你设计的方法是否存在某些假设？基于这些假设下的标定观测值和预测值分别是什么？

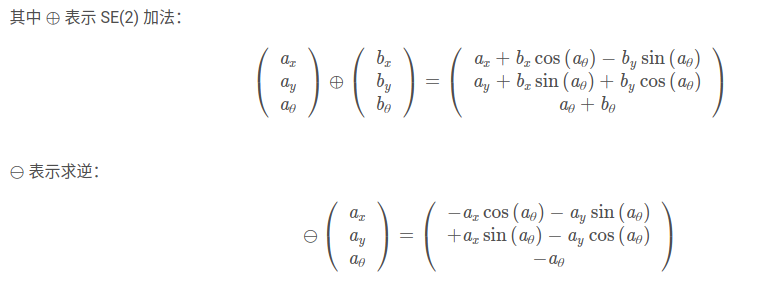
（2）如何构建你的最小二乘方程组求解该外参？  
解答：

（1）

在假设已经标定好了里程计内参的情况下，标定里程计和激光雷达之间的外参

设定激光雷达坐标行想对于机器人坐标系的外参数为L=（lx,ly,lα）∈SE(2)

对于SE（2）的计算，规定如下：



设定k时刻与k+1时刻，里程计坐标系位姿变化量为rk，激光雷达坐标系位姿变化量为sk

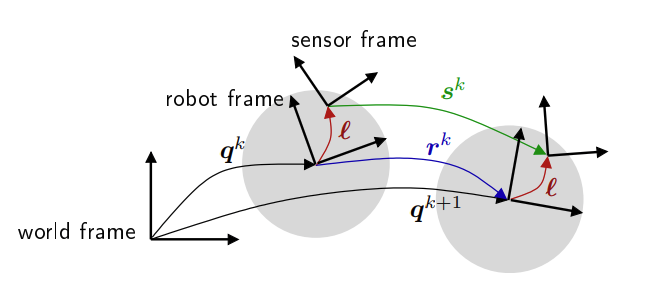
世界坐标系下，里程计坐标系k时刻的位置为qk,k+1时刻为qk+1

那么有：



 （2）

如下图所示：



将（1）拆开并带入（2）式

得到：



其中内参rL,rR,b已经在内参标定中得到。

上面的sk是标定预测值，即是通过里程计和设定的外参计算得到的

而激光雷达自身可以通过匹配得到k时刻到k+1时刻的转换位姿sk’

所以可以使用最小二乘法来优化下面的目标函数：

