

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：

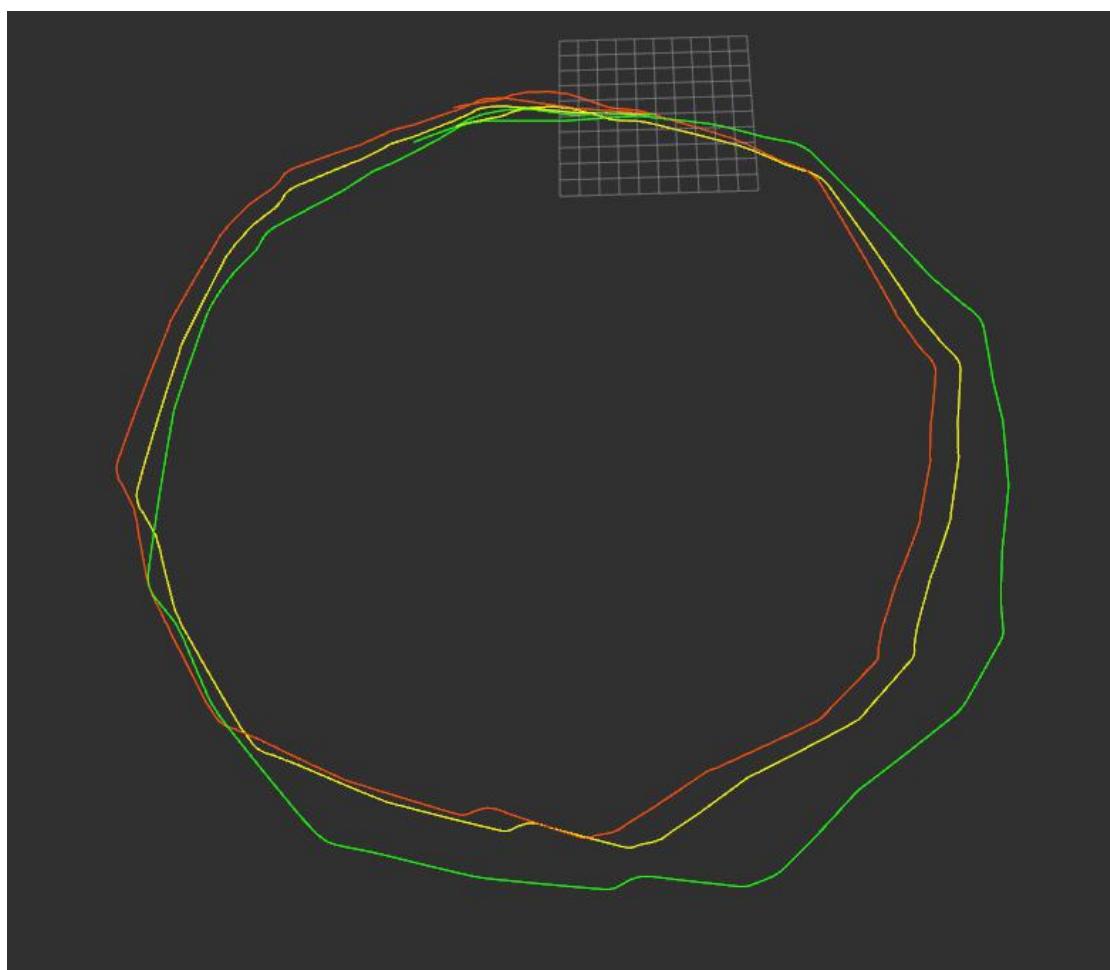
```
correct_matrix:  
 0.935999  2.11281 -0.0490449  
-0.0128723  6.88882  0.151233  
 0.0013673  22.9612   0.393633  
calibration over!!!!
```

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：

```
sunm@sunm-Legion:~/work/code/shenlan-2d/2/HW2/odom_calib$ ./odom_calib
J21: -0.163886
J22: 0.170575
b: 0.59796
r_L: 0.0979974
r_R: 0.101997
参考答案：轮间距b为0.6m左右，两轮半径为0.1m左右
```

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： $Ax=b \rightarrow A'Ax=A'b \rightarrow x=1/(A'A)^*A'^*b$

' 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 $L = (l_x, l_y, l\alpha) \in SE(2)$

对于 $SE(2)$ 的计算，规定如下：

其中 \oplus 表示 $SE(2)$ 加法：

$$\begin{pmatrix} a_x \\ a_y \\ a_\theta \end{pmatrix} \oplus \begin{pmatrix} b_x \\ b_y \\ b_\theta \end{pmatrix} = \begin{pmatrix} a_x + b_x \cos(a_\theta) - b_y \sin(a_\theta) \\ a_y + b_x \sin(a_\theta) + b_y \cos(a_\theta) \\ a_\theta + b_\theta \end{pmatrix}$$

\ominus 表示求逆：

$$\ominus \begin{pmatrix} a_x \\ a_y \\ a_\theta \end{pmatrix} = \begin{pmatrix} -a_x \cos(a_\theta) - a_y \sin(a_\theta) \\ +a_x \sin(a_\theta) - a_y \cos(a_\theta) \\ -a_\theta \end{pmatrix}$$

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

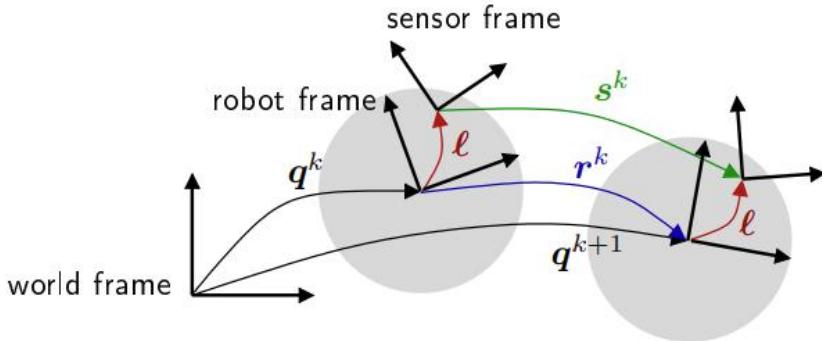
世界坐标系下，里程计坐标系 k 时刻的位置为 $q_{k,k+1}$ 时刻为 q_{k+1}

那么有：

$$s^k = \ominus (q^k \oplus \ell) \oplus (q^{k+1} \oplus \ell) \quad (1)$$

$$r^k = \ominus q^k \oplus q^{k+1} = r^k(r_L, r_R, b), \quad (2)$$

如下图所示：



将 (1) 拆开并带入 (2) 式

得到：

$$s^k = \ominus \ell \oplus r^k(r_L, r_R, b) \oplus \ell$$

其中内参 r_L, r_R, b 已经在内参标定中得到。

上面的 s^k 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 s_k'

所以可以使用最小二乘法来优化下面的目标函数：

$$\mathcal{J} = -\frac{1}{2} \sum_{k=1}^n \left\| \hat{s}^k - \ominus \ell \oplus r^k(r_L, r_R, b) \oplus \ell \right\|_{\Sigma_k^{-1}}^2$$