1. 补充直接线性方法的里程计标定模块代码；（6 分）

    1. Main.cpp，第 358 行中的 cal_delta_distance()函数,该函数的功能为给定两个里程计位姿,计算这两个位姿之间的位姿差。

```
//TODO:
// std::cout << "last_pos: " << last_pos << std::endl;
// std::cout << "now_pos: " << now_pos << std::endl;
Eigen::Matrix3d T_last, T_now, T_last_now;
T_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;

T_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;

T_last_now = T_last.inverse() * T_now;
d_pos << T_last_now(0, 2),
         T_last_now(1, 2),
         atan2(T_last_now(1, 0), T_last_now(0, 0));
// std::cout << "d_pos: " << d_pos << std::endl;
//end of TODO:
```

    2. Odom_Calib.cpp，第 28 行 Add_Data()函数，该函数的功能为构建超定方程组 Ax=b，具体参考PPT。

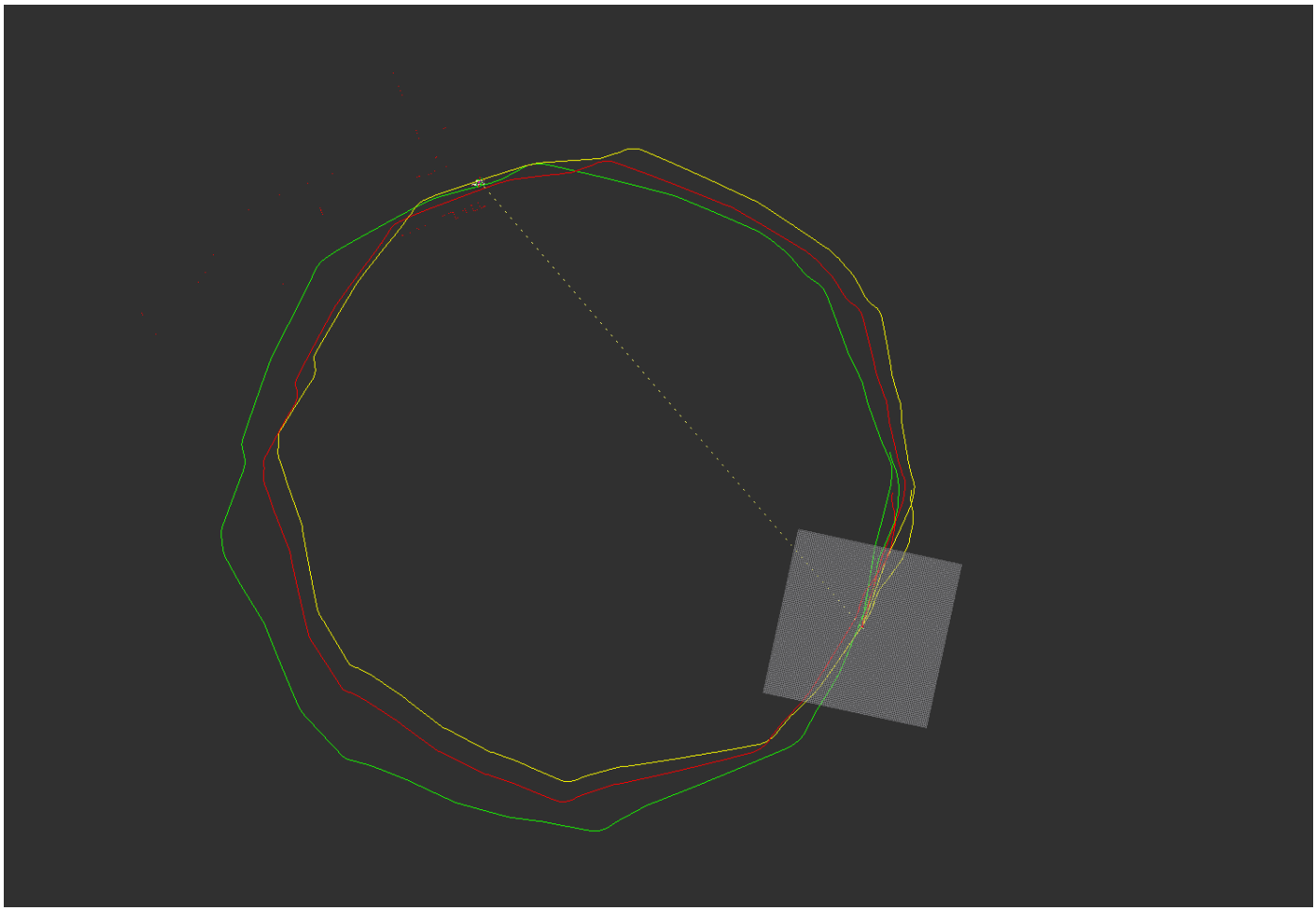
```
    //TODO: 构建超定方程组
// std::cout << "1" << std::endl;
// A << Odom(0), Odom(1), Odom(2), 0, 0, 0, 0, 0, 0,
//      0, 0, 0, Odom(0), Odom(1), Odom(2), 0, 0, 0,
//      0, 0, 0, 0, 0, 0, Odom(0), Odom(1), Odom(2);
A(now_len % data_len * 3, 0) =  Odom(0);
A(now_len % data_len * 3, 1) =  Odom(1);
A(now_len % data_len * 3, 2) =  Odom(2);
A(now_len % data_len * 3 + 1, 3) =  Odom(0);
A(now_len % data_len * 3 + 1, 4) =  Odom(1);
A(now_len % data_len * 3 + 1, 5) =  Odom(2);
A(now_len % data_len * 3 + 2, 6) =  Odom(0);
A(now_len % data_len * 3 + 2, 7) =  Odom(1);
A(now_len % data_len * 3 + 2, 8) =  Odom(2);

// b << scan(0),
//      scan(1),
//      scan(2);
b(now_len % data_len * 3) = scan(0);
b(now_len % data_len * 3 + 1) = scan(1);
b(now_len % data_len * 3 + 2) = scan(2);
// std::cout << "1" << std::endl;
//end of TODO
```

3. Odom_Calib.cpp，第 48 行 Solve()函数，该函数的功能为对 2 中构建的超定方程组进行求解。

```
//TODO: 求解线性最小二乘
Eigen::VectorXd x(9);
x.setZero();
x = A.colPivHouseholderQr().solve(b);
// x = (A.transpose() * A).inverse() * A.transpose() * b;
correct_matrix << x(0), x(1), x(2),
                  x(3), x(4), x(5),
                  x(6), x(7), x(8);
//end of TODO
```

最终运行结果为：



```
Data Cnt:2518
Data Cnt:2519
Data Cnt:2520
Data Cnt:2521
Data Cnt:2522
Data Cnt:2523
Data Cnt:2524
Data Cnt:2525
Data Cnt:2526
Data Cnt:2527
Data Cnt:2528
correct_matrix:
  0.936047    2.11783 -0.0491109
-0.0129435     6.8227   0.152785
0.00128118    22.8207   0.397173
calibration over!!!!
Data Cnt:2529
```

2. 补充基于模型方法的里程计标定模块代码；（2 分）

```
    // 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.colPivHouseholderQr().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)
    // std::cout << C.colPivHouseholderQr().solve(S) << std::endl;
    Eigen::VectorXd b_tmp = C.colPivHouseholderQr().solve(S);
    b_wheel = b_tmp(0);
    r_L = -J21 * b_wheel;
    r_R = J22 * b_wheel;
    //end of TODO
```

最终运行结果为:

```
In file included from /usr/include/c++/5/bits/stl_function.h:1128:0,
                 from /usr/include/c++/5/string:48,
                 from /home/txcom-ubuntu64/HW2/odom_calib/odom_calib.cpp:1:
/usr/include/c++/5/backward/binders.h:143:11: note: declared here
     class binder2nd
           ^
Linking CXX executable odom_calib
[100%] Built target odom_calib
txcom-ubuntu64@ubuntu:~/HW2/odom_calib$ ./odom_calib
J21: -0.163886
J22: 0.170575
0.59796
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
```

3. 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?

这里只考虑 Ax=b 为超定方程组，所以求最小二乘解。一般有三种求解方法：SVD分解、QR分解、正规矩阵(normal equation)。

    2. 各方法的优缺点有哪些？分别在什么条件下较常被使用?

在这三种方程中，SVD分解通常准确率最高，但是速度最慢，对准确性高的情况下使用该求解方法；正规矩阵速度最快，但是准确率最低，如果矩阵 A 是病态的，那么这不是一个好方法，因为 A^TA 是 A 的平方，这意为着使用正规方程所造成的损失是使用其他方法所造成损失的两倍，存在病态矩阵的情况下不使用该方法；QR分解位于两者之间，Eigen中QR分解类有三个：HouseholderQR（没有枢转，快速但不稳定），ColPivHouseholderQR（列枢转，因此有一些慢但是更准确），FullPivHouseholderQR（完全旋转，最慢，但是最稳定），我一般会使用ColPivHouseholderQR方法。

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）
   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?

使用模型法标定激光雷达与里程计的相对角度（yaw），假设激光雷达与车体相对位置的x、y值为0，仅有一个角度偏差（yaw）；且里程计的内参已经标定完成。标定观测值为激光雷达的匹配值，预测值为里程计的积分值。

    2. 如何构建你的最小二乘方程组求解该外参?

设 r_x r_y 为里程计积分值， s_x s_y 为激光雷达的匹配值,激光雷达与里程计的相对角度（yaw）为 θ 。

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} r_x \\ r_y \end{bmatrix}$$

得到：

$$\begin{bmatrix} r_y & r_x \\ -r_x & r_y \end{bmatrix} \cdot \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix} = \begin{bmatrix} s_x \\ s_y \end{bmatrix}$$

由n组数据，构建最小二程方程组：

$$\begin{bmatrix} r_{y0} & r_{x0} \\ -r_{x0} & r_{y0} \\ \vdots & \vdots \\ r_{yn} & r_{xn} \\ -r_{xn} & r_{yn} \end{bmatrix} \cdot \begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix} = \begin{bmatrix} s_{x0} \\ s_{y0} \\ \vdots \\ s_{xn} \\ s_{yn} \end{bmatrix}$$

$$Ax = b$$

已知 $\sin\theta$ $\cos\theta$，因此：$\theta = atan2(\sin\theta, \cos\theta)$。