

- 一、标定模块代码补充
  - 1.1 直接线性方法
  - 1.2 基于模型方法
- 二、线性方程组 $Ax=b$ 的求解方法
- 三、设计里程计与激光雷达外参标定方法

## 一、 标定模块代码补充

### 1.1 直接线性方法

1) 求解得到两帧数据之间的位姿差

```
1 //TODO:
2 Eigen::Matrix3d T_last_now;
3 double c,s;
4 c = cos(now_pos(2));
5 s = sin(now_pos(2));
6 T_last_now << c, -s, 0,
7               s,  c, 0,
8               0,  0, 1;
9 d_pos = T_last_now.transpose() * (now_pos - last_pos);
10 //end of TODO:
```

2) 构建超定方程组

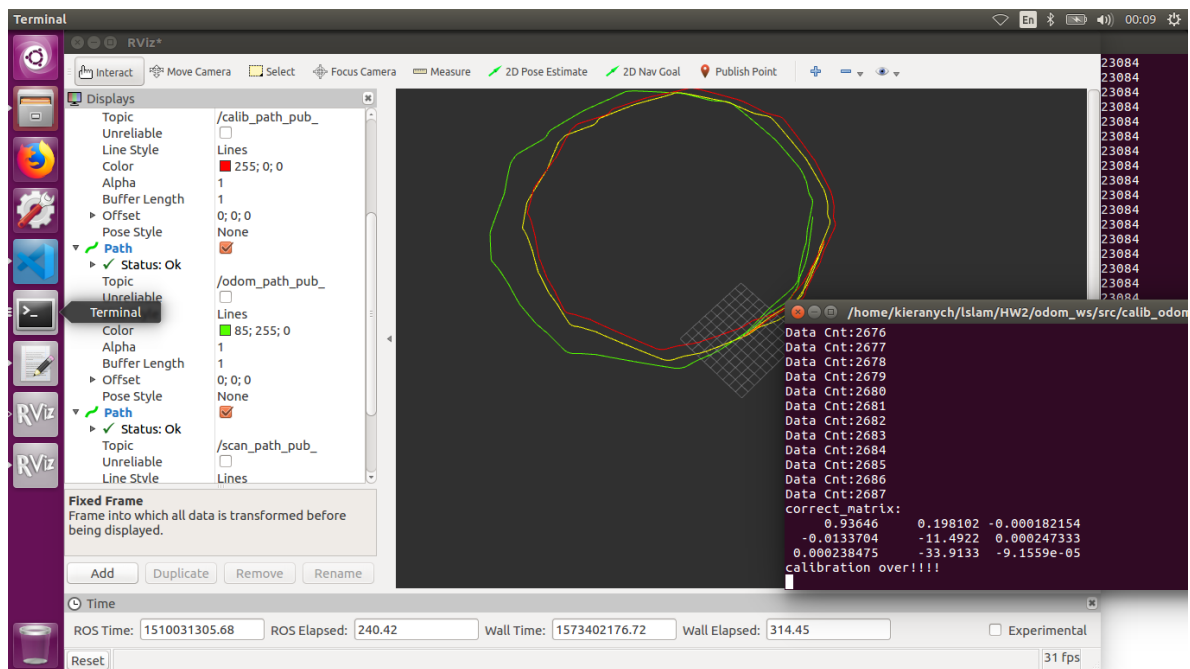
```
1 //TODO: 构建超定方程组
2 Eigen::Matrix<double,3,9> A_tmp;
3 Eigen::Matrix<double,3,1> b_tmp;
4 A_tmp.setZero();
5 b_tmp.setZero();
6
7 b_tmp = scan;
8
9 for(int i = 0; i < 3; i++){
10     Eigen::Matrix<double,1,9> tmp;
11     tmp.setZero();
12     tmp.block<1,3>(0,i*3) = odom.transpose();
13     A_tmp.block<1,9>(i,0) = tmp;
14 }
15 A.block<3,9>(now_len*3,0) = A_tmp;//J^T * J
16 b.block<3,1>(now_len*3,0) = b_tmp;//J^T * B
17 //end of TODO
```

3) 求解线性最小二乘 $Ax=b$

```

1 //TODO: 求解线性最小二乘
2 Eigen::Matrix<double,9,1> x;
3 Eigen::MatrixXd A_tmp;
4 Eigen::VectorXd b_tmp;
5 A_tmp = A.transpose() * A;
6 b_tmp = A.transpose() * b;
7 // 使用qr求解尝试效果
8 x = A_tmp.colPivHouseholderQr().solve(b_tmp); //Cholesky decomposition
   (A^TA)*x=A^Tb
9
10 correct_matrix << x(0),x(1),x(2),
11                  x(3),x(4),x(5),
12                  x(6),x(7),x(8);
13 //end of TODO

```



## 1.2 基于模型方法

1) 填充A, b矩阵

```
1 // 填充A, b矩阵
2 //TODO: (3~5 lines)
3 A(id_s,0) = w_Lt;
4 A(id_s,1) = w_Rt;
5 b(id_s,0) = s_th;
6 //end of TODO
```

## 2) J21J22求解

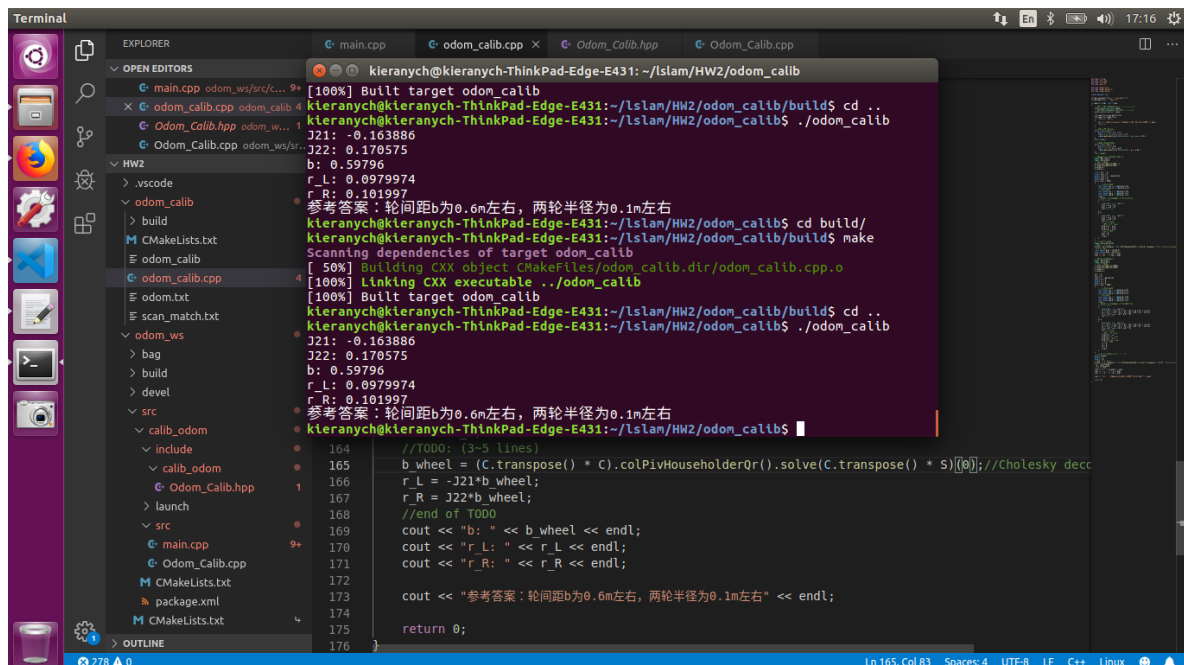
```
1 //TODO: (1~2 lines)
2 J21J22 = (A.transpose() * A).colPivHouseholderQr().solve(A.transpose() *
3 b); //Cholesky decomposition
4 //end of TODO
```

### 3) 填充C, S矩阵

```
1 // 填充C, S矩阵
2 //TODO: (4~5 lines)
3 C(id_s*2) = cx;
4 C(id_s*2 + 1) = cy;
5 S(id_s*2) = s_x;
6 S(id_s*2 + 1) = s_y;
7 //end of TODO
```

### 4) b\_wheel 求解

```
1 //TODO: (3~5 lines)
2 b_wheel = (C.transpose() * C).colPivHouseholderQR().solve(C.transpose() * S)
3 (0); //Cholesky decomposition
4 r_L = -J21*b_wheel;
5 r_R = J22*b_wheel;
```



```
Terminal
kieranych@kieranych-ThinkPad-Edge-E431: ~/Islam/HW2/odom_calib
[100%] Built target odom_calib
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/HW2/odom_calib/build$ cd ..
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/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
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/HW2/odom_calib/build$ make
Scanning dependencies of target odom_calib
[ 50%] Building CXX object CMakeFiles/odom_calib.dir/odom_calib.cpp.o
[100%] Linking CXX executable ../odom_calib
[100%] Built target odom_calib
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/HW2/odom_calib/build$ cd ..
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/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
kieranych@kieranych-ThinkPad-Edge-E431:~/Islam/HW2/odom_calib$

164 //TODO: (3~5 lines)
165 b_wheel = (C.transpose() * C).colPivHouseholderQR().solve(C.transpose() * S)[0]; //Cholesky decc
166 r_L = -J21*b_wheel;
167 r_R = J22*b_wheel;
168 //end of TODO
169 cout << "b: " << b_wheel << endl;
170 cout << "r_L: " << r_L << endl;
171 cout << "r_R: " << r_R << endl;
172
173 cout << "参考答案: 轮间距b为0.6m左右, 两轮半径为0.1m左右" << endl;
174
175 return 0;
176 }
```

## 二、线性方程组 $Ax=b$ 的求解方法

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) 对于该类问题, 我找到有2种求解方法, 分别如下:

- 直接法: 若在计算过程中没有舍入误差, 经过有限步算术运算, 可求得方程的精确解的方法。
- 迭代法: 用某种极限过程去逐步逼近线性方程组精确解的方法。

2) 其中,

对于直接法, 通过 $x = A^{-1}b$ , 获得方程解, 但是对较大的的矩阵, 一般这方法都不是好的选择。因为求 $A^{-1}$ 的过程中, 会做许多不必要的计算。而且当A近于奇异时, 很难解出来。

而对于迭代法, 适用于大而稀疏的矩阵, LU分解后可采用用Gaussian消去法, 还有就是对正定的对称矩阵, 也可以采用共轭梯度法, 收敛速度非常快。

### 三、设计里程计与激光雷达外参标定方法

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？

#### 1) 手眼标定法

里程计与激光雷达，因为两者的测量之间没有直接的对应，所以需要使用手眼标定的方法对外参的初值进行求解。假设在  $t_i$  时刻里程计的位置姿态为  $T_i^o$ ，激光雷达里程计的位置为  $T_i^l$ ，则经典的手眼标定问题为求解  $T_l^o$ ，使得：

$$T_{i,i+1}^o T_l^o = T_l^o T_{i,i+1}^l$$

其中  $T_{i,i+1}^o = T_{i+1}^o (T_i^o)^{-1}$ ， $T_{i,i+1}^l = T_{i+1}^l (T_i^l)^{-1}$  是两个传感器的相对运动。

这里的标定观测值是  $T_i^o$  和  $T_i^l$ ，预测值是  $T_l^o$ 。

#### 2) 外参求解

由于车辆在近似平面内运动，将问题简化为二维的手眼标定问题，则有以下式子成立：

$$(R_o - I)t = Rt_l - t_o,$$

其中  $R_o$  和  $t_o$  分别是组合惯导相对运动的旋转和平移部分， $t_l$  是激光雷达相对运动的平移部分， $R$  和  $t$  是外参的旋转和平移。令

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

则有，

$$Rt_l - (R_o - I)t = t_o$$

一次相对运动能构造两个约束，当有三个以上不同位置朝向的运动时，方程满秩，可线性求解。为了保证初值求解以及第二步优化过程中对外参构成足够的约束，算法要求机器人以8字形状的轨迹行驶。