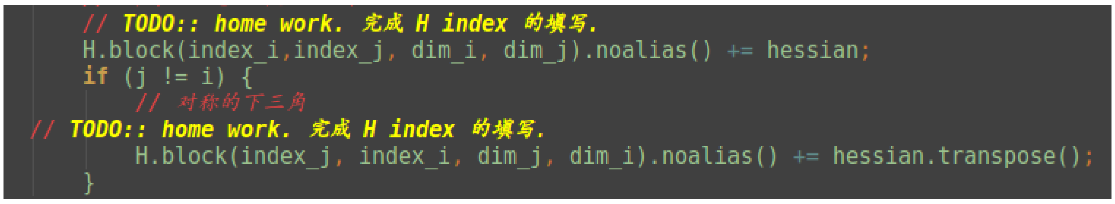
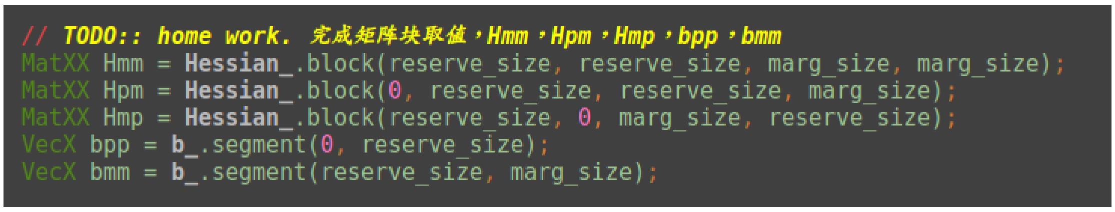
第一题 完成单目 Bundle Adjustment 求解器 problem.cc 中的部分代码

（1）Problem::MakeHessian()中信息矩阵 H 的计算

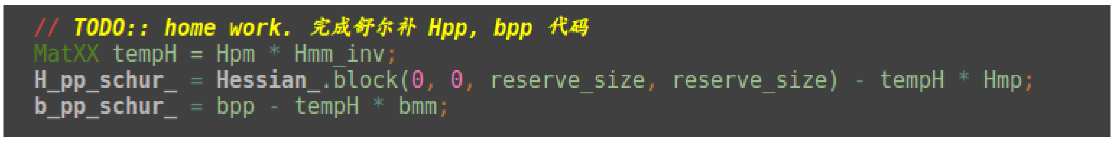


```
      // TODO:: home work. 完成 H index 的填写.
      H.block(index_i,index_j, dim_i, dim_j).noalias() += hessian;
      if (j != i) {
          // 对称的下三角
  // TODO:: home work. 完成 H index 的填写.
          H.block(index_j, index_i, dim_j, dim_i).noalias() += hessian.transpose();
      }
```

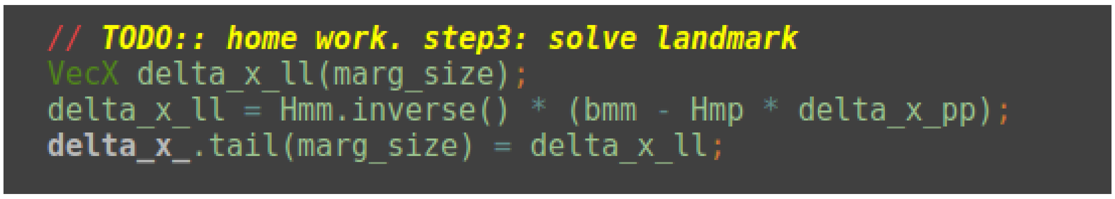
（2）Problem::SolveLinearSystem()中 SLAM 问题的求解



```
// TODO:: home work. 完成矩阵块取值，Hmm，Hpm，Hmp，bpp，bmm
MatXX Hmm = Hessian_.block(reserve_size, reserve_size, marg_size, marg_size);
MatXX Hpm = Hessian_.block(0, reserve_size, reserve_size, marg_size);
MatXX Hmp = Hessian_.block(reserve_size, 0, marg_size, reserve_size);
VecX bpp = b_.segment(0, reserve_size);
VecX bmm = b_.segment(reserve_size, marg_size);
```



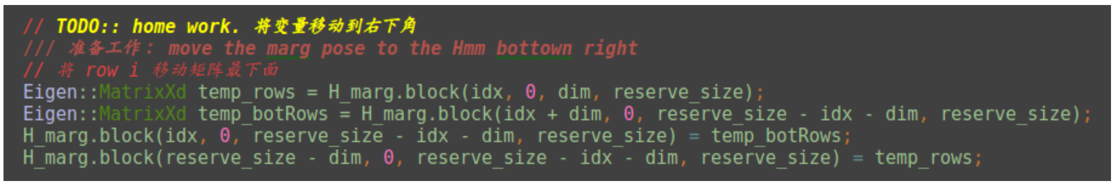
```
// TODO:: home work. 完成舒尔补 Hpp, bpp 代码
MatXX tempH = Hpm * Hmm_inv;
H_pp_schur_ = Hessian_.block(0, 0, reserve_size, reserve_size) - tempH * Hmp;
b_pp_schur_ = bpp - tempH * bmm;
```



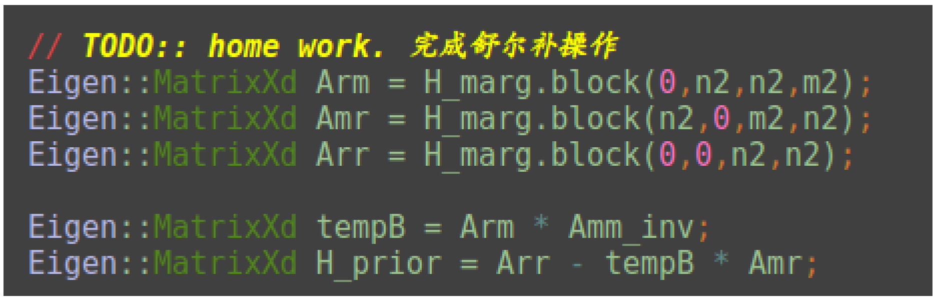
```
// TODO:: home work. step3: solve landmark
VecX delta_x_ll(marg_size);
delta_x_ll = Hmm.inverse() * (bmm - Hmp * delta_x_pp);
delta_x_.tail(marg_size) = delta_x_ll;
```

第二题 完成滑动窗口算法测试函数

完成 Problem::TestMarginalize() 中的代码



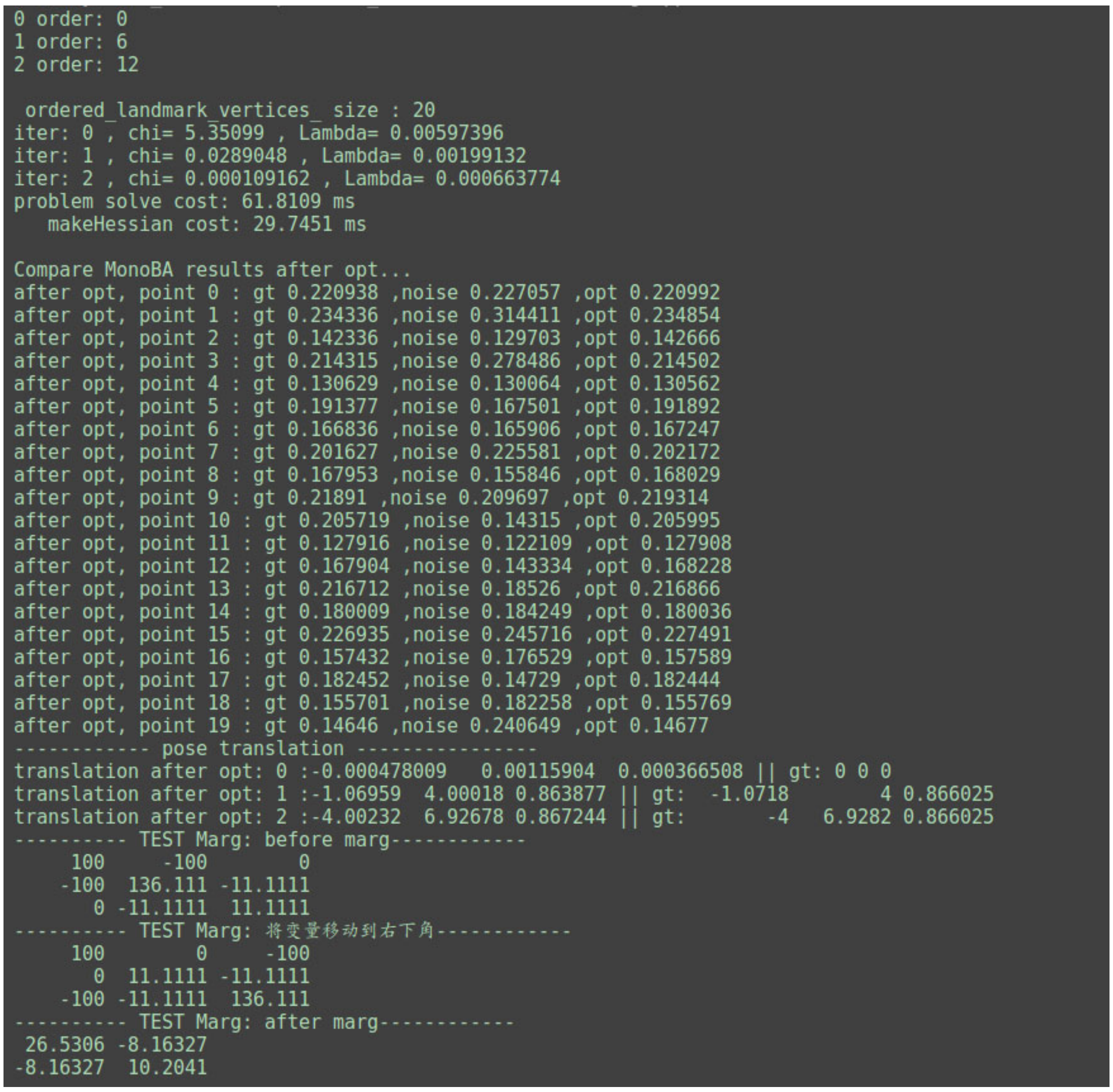
```
// TODO:: home work. 将变量移动到右下角
/// 准备工作： move the marg pose to the Hmm bottown right
// 将 row i 移动矩阵最下面
Eigen::MatrixXd temp_rows = H_marg.block(idx, 0, dim, reserve_size);
Eigen::MatrixXd temp_botRows = H_marg.block(idx + dim, 0, reserve_size - idx - dim, reserve_size);
H_marg.block(idx, 0, reserve_size - idx - dim, reserve_size) = temp_botRows;
H_marg.block(reserve_size - dim, 0, reserve_size - idx - dim, reserve_size) = temp_rows;
```



```
// TODO:: home work. 完成舒尔补操作
Eigen::MatrixXd Arm = H_marg.block(0,n2,n2,m2);
Eigen::MatrixXd Amr = H_marg.block(n2,0,m2,n2);
Eigen::MatrixXd Arr = H_marg.block(0,0,n2,n2);

Eigen::MatrixXd tempB = Arm * Amm_inv;
Eigen::MatrixXd H_prior = Arr - tempB * Amr;
```

编译后运行可执行程序 TestMonoBA，结果如下图所示：



```
0 order: 0
1 order: 6
2 order: 12

 ordered_landmark_vertices_ size : 20
iter: 0 , chi= 5.35099 , Lambda= 0.00597396
iter: 1 , chi= 0.0289048 , Lambda= 0.00199132
iter: 2 , chi= 0.000109162 , Lambda= 0.000663774
problem solve cost: 61.8109 ms
   makeHessian cost: 29.7451 ms

Compare MonoBA results after opt...
after opt, point 0 : gt 0.220938 ,noise 0.227057 ,opt 0.220992
after opt, point 1 : gt 0.234336 ,noise 0.314411 ,opt 0.234854
after opt, point 2 : gt 0.142336 ,noise 0.129703 ,opt 0.142666
after opt, point 3 : gt 0.214315 ,noise 0.278486 ,opt 0.214502
after opt, point 4 : gt 0.130629 ,noise 0.130064 ,opt 0.130562
after opt, point 5 : gt 0.191377 ,noise 0.167501 ,opt 0.191892
after opt, point 6 : gt 0.166836 ,noise 0.165906 ,opt 0.167247
after opt, point 7 : gt 0.201627 ,noise 0.225581 ,opt 0.202172
after opt, point 8 : gt 0.167953 ,noise 0.155846 ,opt 0.168029
after opt, point 9 : gt 0.21891 ,noise 0.209697 ,opt 0.219314
after opt, point 10 : gt 0.205719 ,noise 0.14315 ,opt 0.205995
after opt, point 11 : gt 0.127916 ,noise 0.122109 ,opt 0.127908
after opt, point 12 : gt 0.167904 ,noise 0.143334 ,opt 0.168228
after opt, point 13 : gt 0.216712 ,noise 0.18526 ,opt 0.216866
after opt, point 14 : gt 0.180009 ,noise 0.184249 ,opt 0.180036
after opt, point 15 : gt 0.226935 ,noise 0.245716 ,opt 0.227491
after opt, point 16 : gt 0.157432 ,noise 0.176529 ,opt 0.157589
after opt, point 17 : gt 0.182452 ,noise 0.14729 ,opt 0.182444
after opt, point 18 : gt 0.155701 ,noise 0.182258 ,opt 0.155769
after opt, point 19 : gt 0.14646 ,noise 0.240649 ,opt 0.14677
------------ pose translation ----------------
translation after opt: 0 :-0.000478009   0.00115904  0.000366508 || gt: 0 0 0
translation after opt: 1 :-1.06959  4.00018 0.863877 || gt:  -1.0718        4 0.866025
translation after opt: 2 :-4.00232  6.92678 0.867244 || gt:       -4   6.9282 0.866025
---------- TEST Marg: before marg------------
     100     -100        0
    -100  136.111 -11.1111
       0 -11.1111  11.1111
---------- TEST Marg: 将变量移动到右下角------------
     100        0     -100
       0  11.1111 -11.1111
    -100 -11.1111  136.111
---------- TEST Marg: after marg------------
 26.5306 -8.16327
-8.16327  10.2041
```

提升题

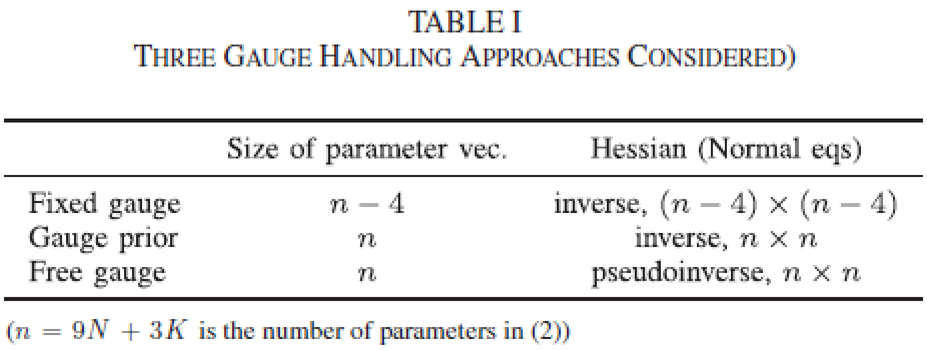


TABLE I
THREE GAUGE HANDLING APPROACHES CONSIDERED)

	Size of parameter vec.	Hessian (Normal eqs)
Fixed gauge	$n-4$	inverse, $(n-4)\times(n-4)$
Gauge prior	n	inverse, $n\times n$
Free gauge	n	pseudoinverse, $n\times n$

($n = 9N + 3K$ is the number of parameters in (2))

方法一：gauge fixation

将不可观的状态固定为给定值，在没有不可观察状态的较小参数空间中进行优化，因此 Hessian 是可逆的。求解器（Gauss-Newton 或 Levenberg-Marquardt-LM）的迭代期间更新方向变量（即旋转）的标准方法是使用局部坐标，其中，在第 q 次迭代时，更新是：

$$\mathrm{R}^{q+1} = \mathrm{Exp}(\delta\phi^{q})\mathrm{R}^{q}$$

对于第一个相机位姿

$$R_0 = \mathrm{Exp}(\Delta\phi_0)R_0^0$$

方法二：gauge prior

将不可观的状态设置先验，通过额外的惩罚（产生可逆的 Hessian）来增加目标函数。 求解器的更新方式与方法一相同。

方法三：free gauge

将不可观的状态在优化期间自由演化。

在处理奇异的 Hessian 时，使用违逆或添加阻尼（Levenberg-Marquardt 算法）

- **仿真比较**

➢ 仿真设置：

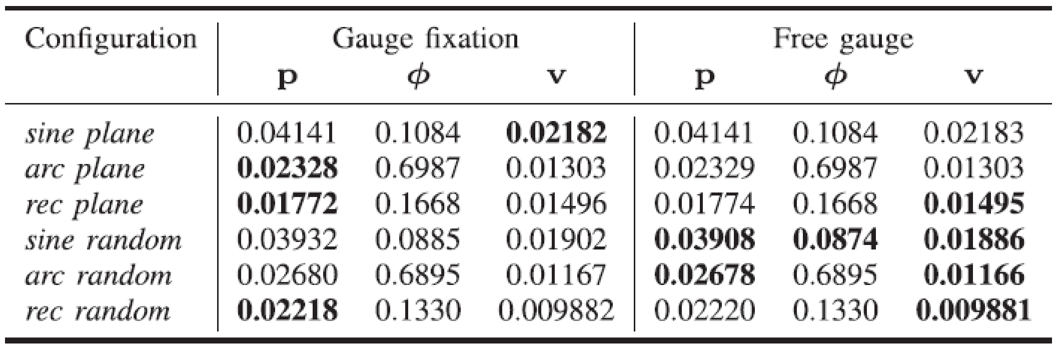
（1）使用三个 6-DoF 轨迹进行实验，即正弦形状，弧形和矩形。3D 点大致分布在多个平面上并且是随机的，并且 3D 点沿着轨迹随机生成；

（2）使用 B 样条拟合轨迹，然后采样加速度和角速度，加上高斯噪声和偏差以生成 IMU 数据；

（3）通过针孔相机模型投影 3D 点以获得相应的图像坐标，然后加高斯噪声来作为相机数据。

➢ 仿真结果：

TABLE II
RMSE ON DIFFERENT TRAJECTORIES AND 3D POINTS CONFIGURATIONS



Configuration	Gauge fixation			Free gauge		
	$\mathbf{p}$	$\boldsymbol{\phi}$	$\mathbf{v}$	$\mathbf{p}$	$\boldsymbol{\phi}$	$\mathbf{v}$
sine plane	0.04141	0.1084	**0.02182**	0.04141	0.1084	0.02183
arc plane	**0.02328**	0.6987	0.01303	0.02329	0.6987	0.01303
rec plane	**0.01772**	0.1668	0.01496	0.01774	0.1668	**0.01495**
sine random	0.03932	0.0885	0.01902	**0.03908**	**0.0874**	**0.01886**
arc random	0.02680	0.6895	0.01167	**0.02678**	0.6895	**0.01166**
rec random	**0.02218**	0.1330	0.009882	0.02220	0.1330	**0.009881**

The smallest errors (e.g., $\mathbf{p}$ gauge fixation vs. $\mathbf{p}$ free gauge) are highlighted. Position, rotation and velocity RMSE are measured in m, deg and m/s, respectively.

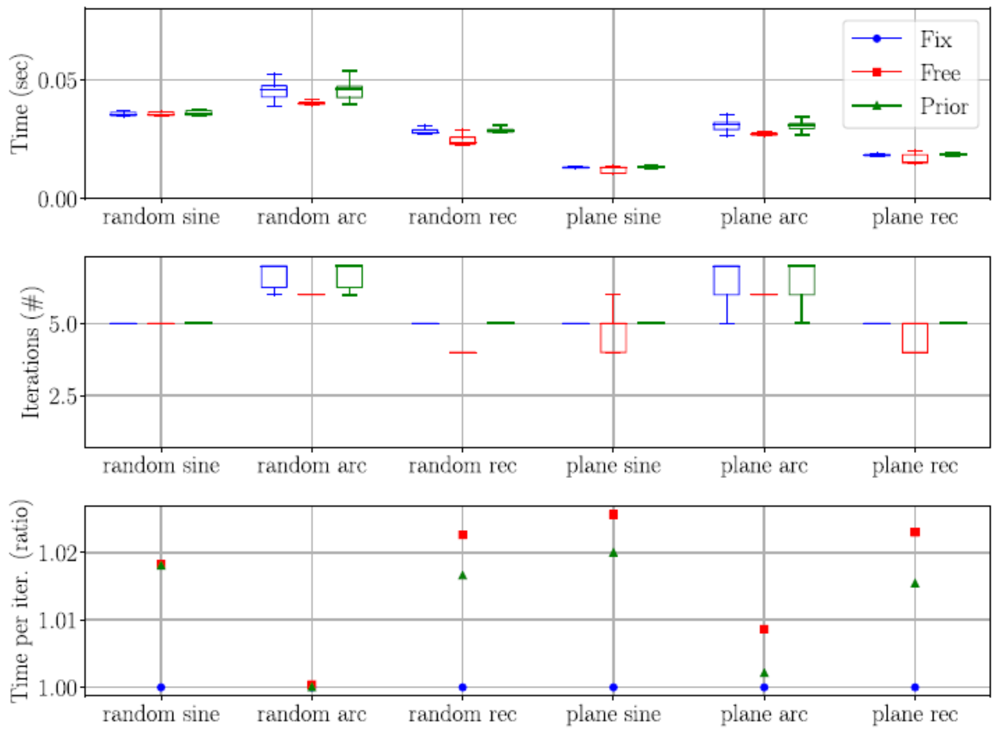


Fig. 7. Number of iterations, total convergence time and time per iteration for all configurations. The time per iteration is the ratio with respect to the gauge fixation approach (in blue), which takes least time per iteration.

- **数据集实验**

使用 EuRoC MAV 数据集的两个序列：Machine Hall 1(MH1)和 Vicon Room 1(VI1)，使用了 SVO 提供优化问题中参数的初始化。

➢ 实验结果

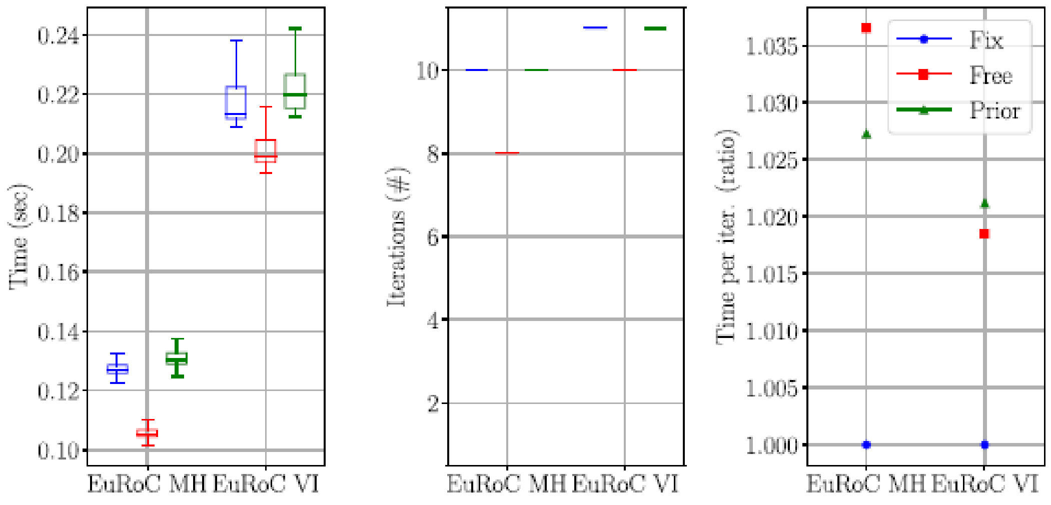


Fig. 11. Computational cost of the three different methods for handling gauge freedom on two sequences from the EuRoC dataset. The time per iteration is the ratio with respect to the gauge fixation approach.

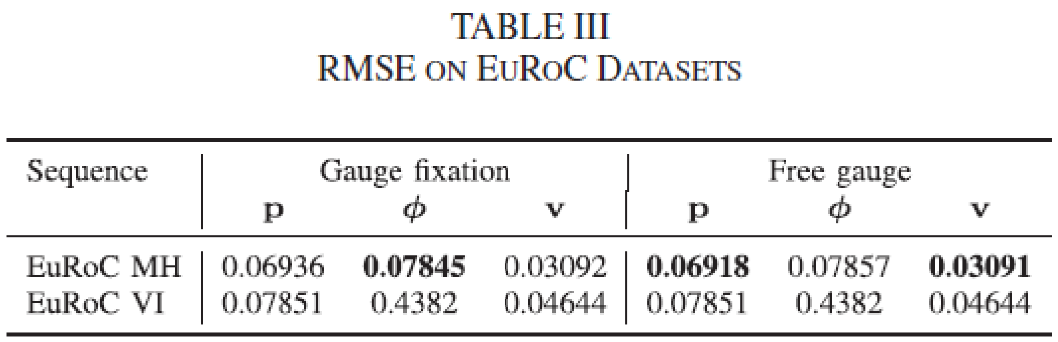


TABLE III
RMSE on EuRoC Datasets

Sequence	Gauge fixation			Free gauge		
	p	ϕ	v	p	ϕ	v
EuRoC MH	0.06936	**0.07845**	0.03092	**0.06918**	0.07857	**0.03091**
EuRoC VI	0.07851	0.4382	0.04644	0.07851	0.4382	0.04644

- **结论**

（1）仿真和实际数据集实验都表明三种方法的准确性和计算时间相似，其中方法三略快；

（2）方法协方差估计表面上看与其他方法完全不同，但实际上与其他方法密切相关。