

源代码修改

按思路提示修改：

按照思路提示对源代码做了如下修改：

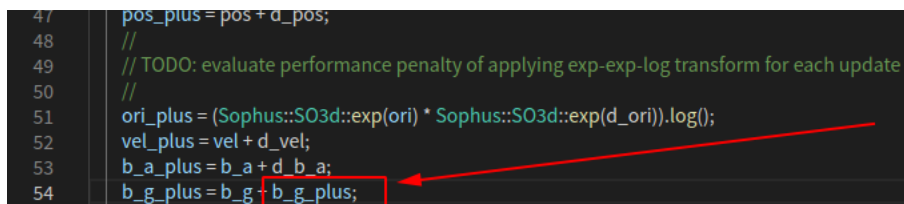
matching.yaml 文件中将 scan_context_path: 修改为： /workspace/assignments/09-sliding-window/src/lidar_localization/slam_data/scan_context

将 lidar_localization/slam_data 下的 map 和 scan_context 全部换成 07 代码框架下对应的文件（对于 scan_context，在编译前替换 lidar_localization/config 下的对应文件，编译后就不需要再替换，不过自己还没验证过）

自行修改：

对原程序 param_prvag.hpp 中的笔误做了修改

```
47 pos_plus = pos + d_pos;
48 //
49 // TODO: evaluate performance penalty of applying exp-exp-log transform for each update
50 //
51 ori_plus = (Sophus::SO3d::exp(ori) * Sophus::SO3d::exp(d_ori)).log();
52 vel_plus = vel + d_vel;
53 b_a_plus = b_a + d_b_a;
54 b_g_plus = b_g + b_g_plus;
```



边缘化相关

参考资料

VINS-MONO 边缘化策略，说明了边缘化因子中如何通过 H_{rr} 和 b_r 得到相应的雅可比矩阵和残差 https://blog.csdn.net/weixin_41394379/article/details/89975386?spm=1001.2014.3001.5506，关键部分如下图：

1. 舒尔补求解 $H_0^* = H_{11} - H_{12}H_{22}^{-1}H_{21}$, $b_0^* = b_{1,0} - H_{12}H_{22}^{-1}b_{2,0}$
2. 分解 H_0^* 求解 J_l 、 J_l^T 、 $(J_l^T)^+$ ，求解 $e_0 = (J_l^T)^+ b_0^*$ ，保存 J_l , e_0
3. 求解 δx 由 $H_0^* \delta x = b_0^*$ (ceres完成)
4. 使用 δx 更新 x , 计算当前状态 x 与 x_0 的差值 dx , 更新 e : $e_p = e_0 + J_l dx$ (costfunction定义residuals, 同时定义costfunction需要的jacobians, 把 J_l 由localsize转为globalsize)
ceres自动更新 b^* : $b^* = J_l^T e_p$;
5. 求解 δx 由 $H_0^* \delta x = b^*$ (ceres完成)
6. 循环4,5

按照视频和课件中的建议，看了 lio-mapping 的源码（还没有掌握好）

按照助教老师的建议，看了 vins-mono 的源码和相关的博客，感觉很多地方还是没有梳理清楚。相关资料如下：

VINS-Mono 代码解读

<https://blog.csdn.net/u012871872/article/details/78128087?locationNum=8&fps=1>

通过这个攻略重点学习 ceres 添加参数块，添加残差，边缘化，滑窗，优化等环节的逻辑顺序和相互关系

[从零写 VIO|第七节]——VINS-Mono 代码精简版代码详解——边缘化(内容|代码)

https://blog.csdn.net/weixin_40224537/article/details/106850694

通过这个攻略重点学习滑窗移动

思路

结合思路提示说明当前作业框架中边缘化相关思路的几个点：

核心思路是要使用 ceres 这个工具，即：要将边缘化因子构造成能够被 `problem.AddResidualBlock` 调用的类。

先对需要边缘化的状态量进行边缘化，得到边缘化因子，然后将该因子加入到 ceres 的残差块中，同时从各个量测量 buff 中去掉和要边缘化掉的量相关的因子。

课件中的 H^{rr} 分为剩余变量对应的 Hessian 矩阵 H^a_{rr} 和待边缘化的 Hessian 矩阵 H^b_{rr} ，代码中是先求解 H^b_{rr} ，然后去掉和要边缘化掉的量相关的因子，之后再添加残差块，自然就只剩下和要边缘化掉的量无关的因子，得到的也就是 H^a_{rr} 。

待解决问题

lio-mapping 中的边缘化时对应的残差维度是未知的，作业所对应的问题是已知的，根本原因还没有想清楚。

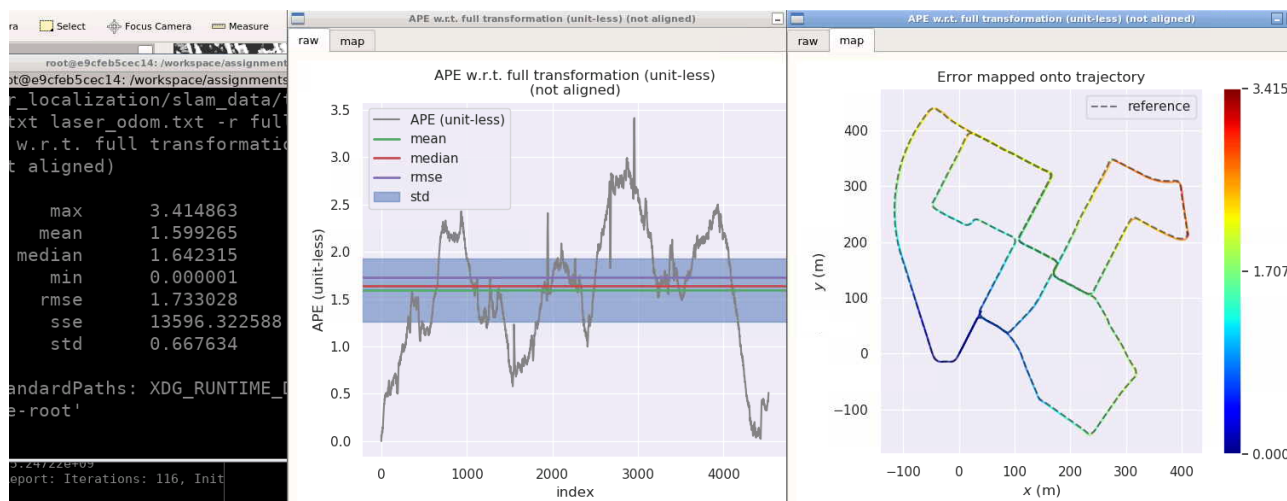
当前作业框架在边缘时忽略了之前边缘化得到的先验因子，只考虑 `map_matching_pose`，`RelativePose` 和 `IMUPreIntegration` 因子，自己目前还不能加入之前边缘化得到的先验因子（记得老师在视频中说，这算是两种方案）。

当前作业框架中没有找到与 VIO 课程中提到的 FEJ（First Estimated Jacobian）相关的点。

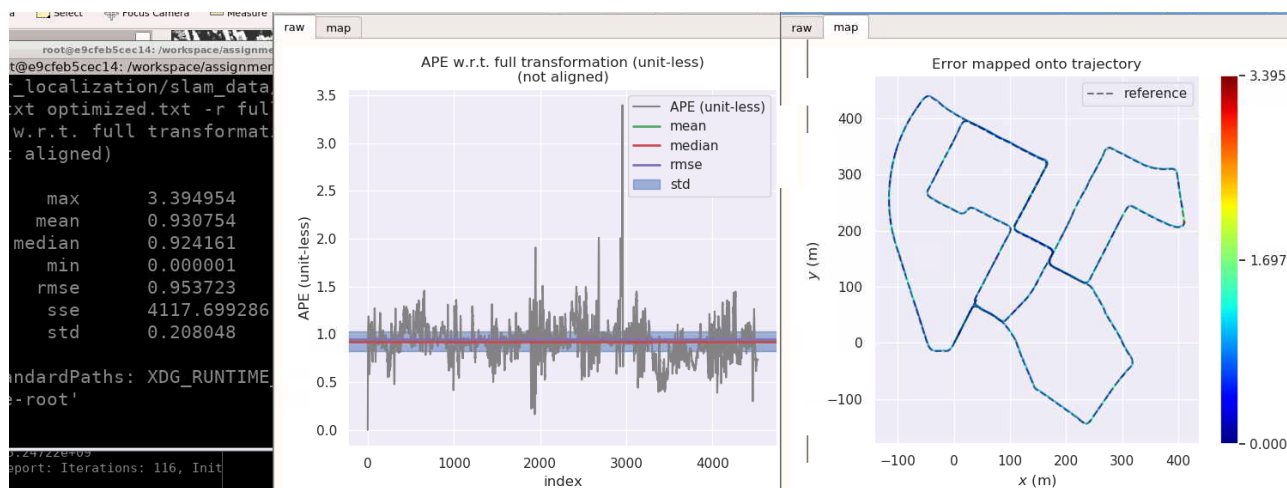
及格要求

在思路提示文件的帮助下，完成要求，可以看到优化后的性能得到了全面的提升。

laser:



optimized

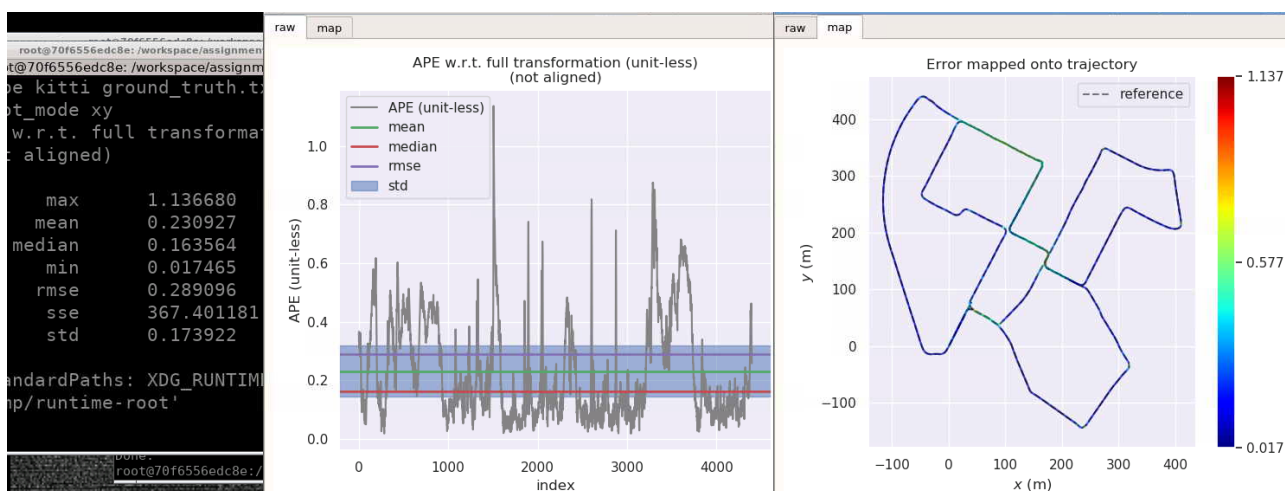


良好要求

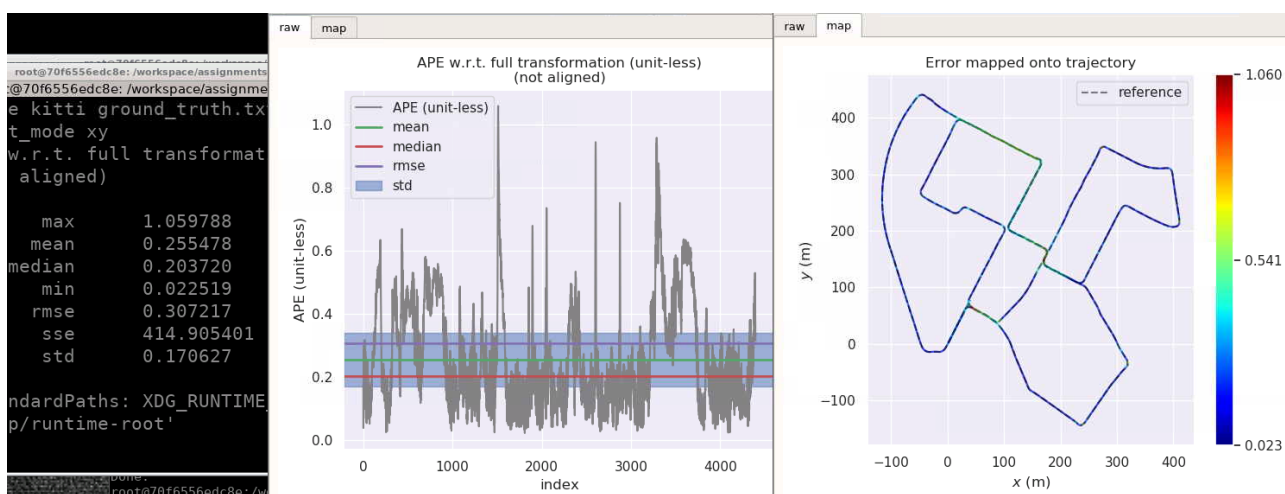
不能直接比较 EKF 滤波算法和图优化算法的仿真结果。因此，这里采用间接比较，即两种融合算法的仿真结果与对应融合前的结果作对比。

下面是第 8 章作业中，量测量为 POSE+VEL，且加入运动约束时，使用 EKF 算法融合前后的仿真结果，可以看到融合后的仿真结果并没有全面优于融合前的仿真结果。结合及格要求中的仿真结果可知，图优化算法结果的精度优于 EKF 算法结果的精度。

laser:



fused:



优秀要求

结论：

窗口长度过小，不能对历史帧的状态量进行有效优化，从而降低融合算法整体性能。

窗口长度过大，优化时计算量增大，算法实时性变差，IMU 预积分的时间间隔变大，积分误差变大，从而降低了融合算法所使用的量测因子的精度，最终导致融合算法的精度下降。

要结合硬件计算资源，选择大小适中的 sliding window size。

待提高：

还没有做设定优化算法时间约束的工作，导致滑动窗口增大后，仿真结果只能体现在实时性能上，无法体现在精度上。如果设定了优化算法的时间消耗限制以保证实时性，那么精度必然会下降。

无法合理解释：

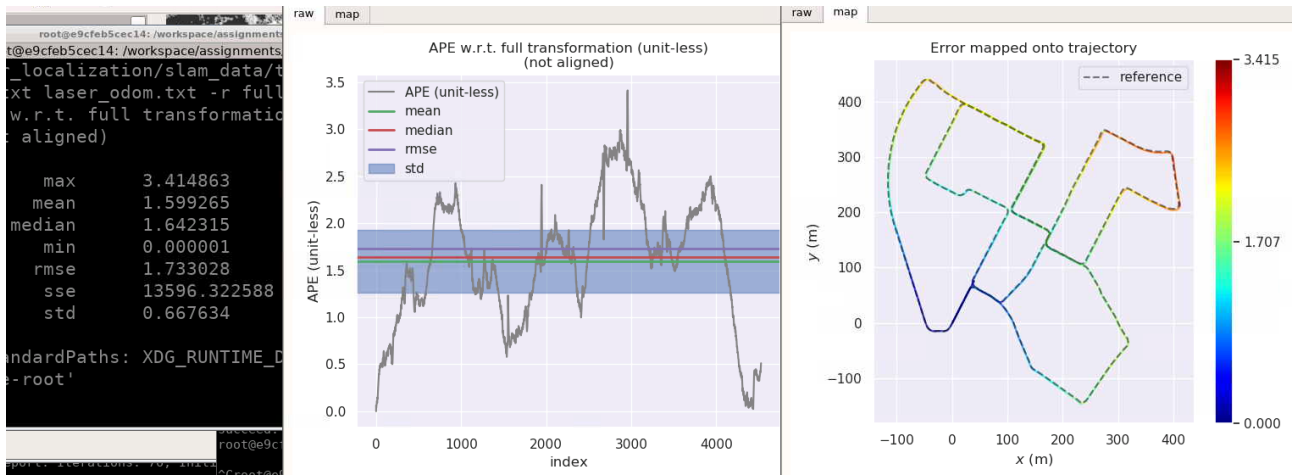
sliding window size 为 3 时的仿真结果性能优于 20 时的性能。暂时无法清楚解释。

仿真结果：

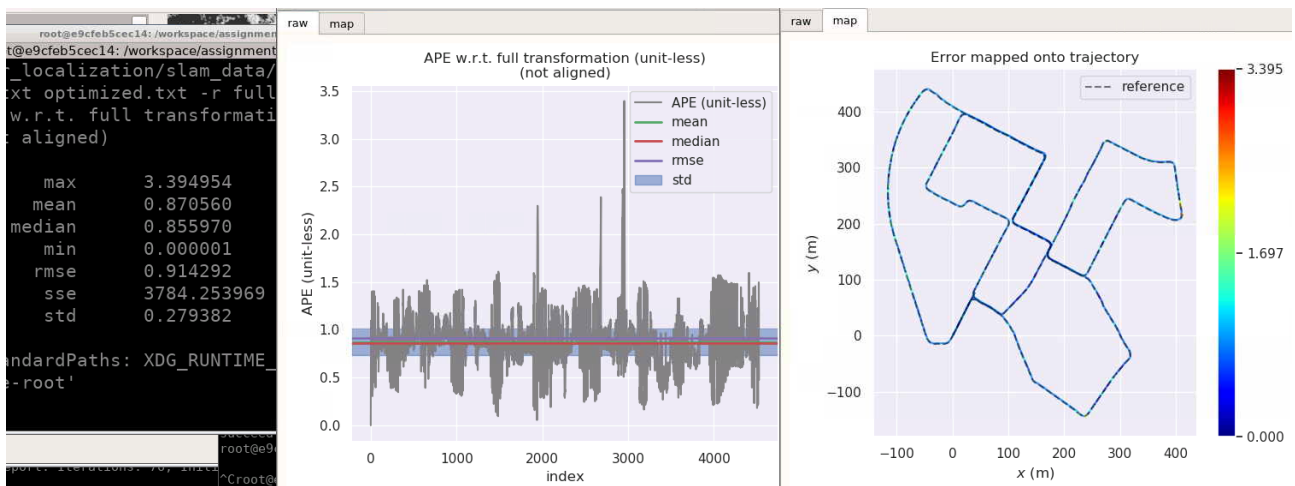
sliding window size : 3

暂时无法解释为什么性能这么好……

laser:



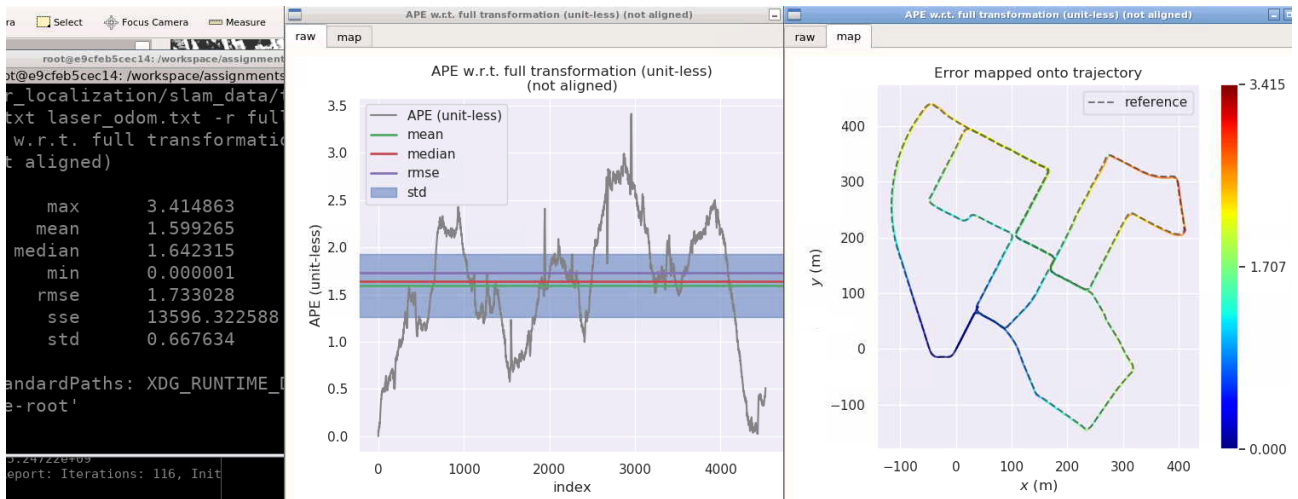
optimized:



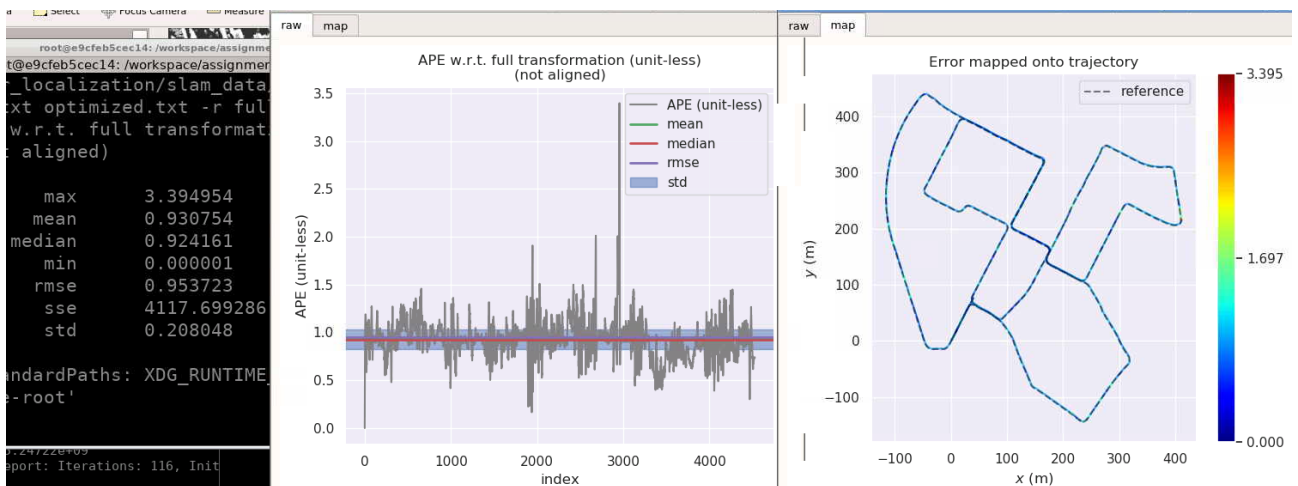
sliding window size : 20

与及格要求相同

laser:

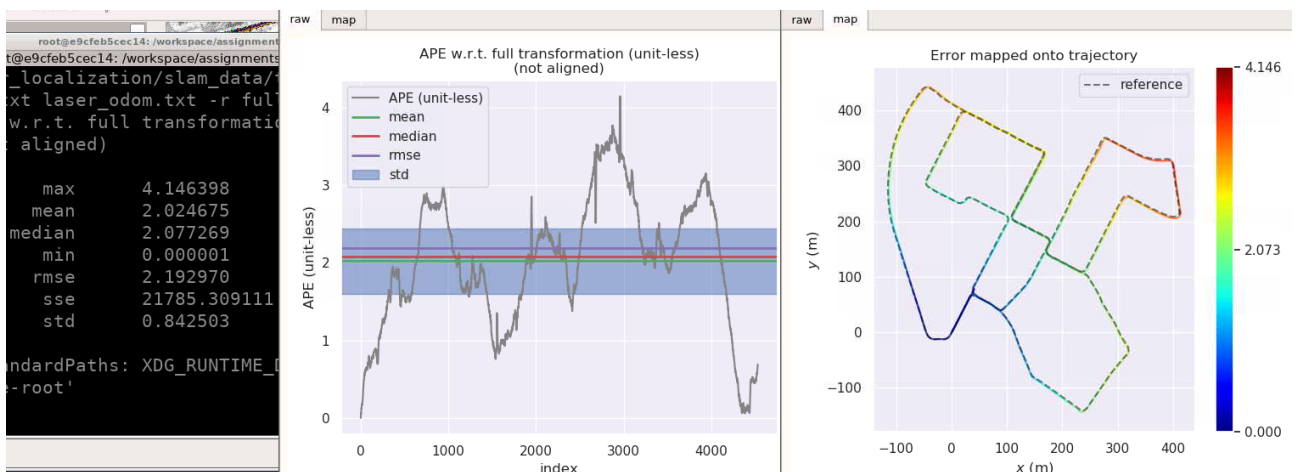


optimized

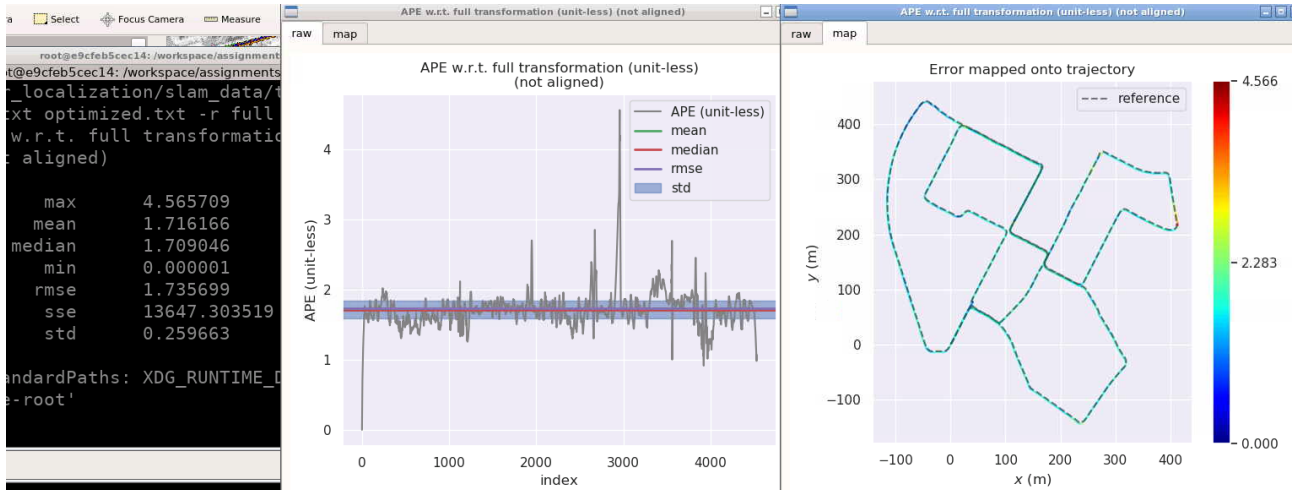


sliding window size : 100

laser:



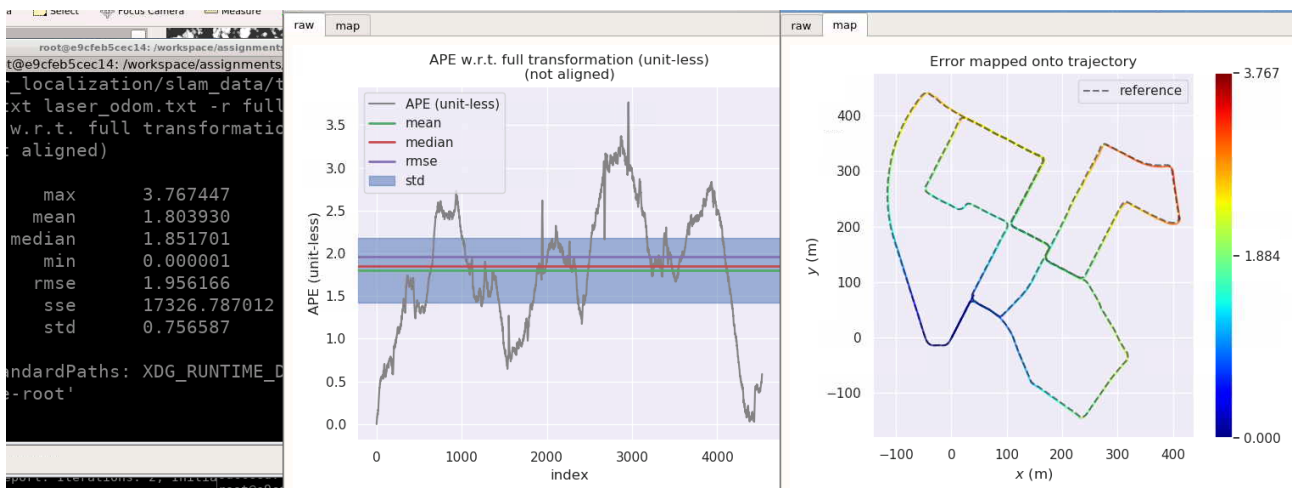
optimized:



sliding window size : 200

虽然在结果上 optimized 的精度优于 laser 的精度，但是这是在优化算法消耗极大计算资源的前提下的结果，在 PC 机上，laser 的仿真结果可以与 rosbag 播放的速度同步，但是 potimized 的结果延迟相当大。还没有做设定优化算法时间约束的工作。

laser:



optimized:

APE w.r.t. full transformation (unit-less)
(not aligned)

Legend:

- APE (unit-less)
- mean
- median
- rmse
- std

The plot displays the APE (unit-less) over an index from 0 to 4500. The y-axis ranges from 0.0 to 4.0. The x-axis ranges from 0 to 4500. The APE (unit-less) line (black) shows significant fluctuations, with a major peak around index 2800 reaching approximately 3.8. The mean (green), median (red), rmse (purple), and std (blue) lines are relatively stable, with the std line showing a slight downward trend after index 3500.

