第3章作业

一. 公式推导

点到线雅克比推导:

点到线的距离 :

$$
d_{\varepsilon}=\frac{\left|\left(p_{i}^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)\right|}{\left|p_{a}-p_{b}\right|}
$$

线特征残差雅可比:

$$
J_{\varepsilon}=\frac{\alpha d_{\varepsilon}}{\alpha T}=\frac{\alpha d_{\varepsilon}}{\alpha p_{i}^{\prime}} \frac{\alpha p_{i}^{\prime}}{\alpha T}
$$

对平移的雅可比:

$$
\frac{\alpha p_{i}^{\prime}}{\alpha t}=I
$$

对旋转的雅可比:

$$
\frac{\alpha p_{i}^{\prime}}{\alpha R}=-\left(R p_{i}\right)_{\times}
$$

求等号右边第一项的偏导数, 令

$$
X=\left(p i^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)
$$

推导得:

$$
\begin{aligned}
\frac{\alpha d_{\varepsilon}}{\alpha p_{i}} & =\frac{\alpha d_{\varepsilon}}{\alpha|X|} \frac{\alpha|X|}{\alpha X} \frac{\alpha X}{\alpha p_{i}} \\
& =\frac{1}{\left|p_{a}-p_{b}\right|} \frac{\alpha|X|}{\alpha X} \frac{\alpha X}{\alpha p_{i}^{\prime}}=\frac{1}{\left|p_{a}-p_{b}\right|} \frac{X}{|X|} \frac{\alpha X}{\alpha p_{i}^{\prime}} \\
& =\frac{1}{\mid p_{a}-p_{b}} \mid \frac{\left(p_{i}^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)}{\left|\left(p_{i}^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)\right|}\left(-\left(p_{i}^{\prime}-p_{a}\right)_{\times}+\left(p i^{\prime}+p_{b}\right)_{\times}\right) \\
& =\frac{\left(p i^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)}{\left|\left(p_{i}^{\prime}-p_{b}\right) \times\left(p_{i}^{\prime}-p_{a}\right)\right|}=\frac{\left(p_{a}-p_{b}\right)_{\times}}{\left|p a-p_{b}\right|}
\end{aligned}
$$

其中,标量对矢量求导

$$
\frac{\alpha|X|}{\alpha X}=\frac{\alpha\left(s \operatorname{qrt}\left(X^{T} X\right)\right)}{\alpha\left(X^{T} X\right)} \frac{\alpha\left(X^{T} X\right)}{\alpha(X)}=\frac{X}{|X|}
$$

面到线雅可比推导:

点到面的距离:

$$
d_{H}=\left(p_{i}^{\prime}-p_{j}\right) \cdot \frac{\left(p_{l}-p_{j}\right) \times\left(p_{m}-p_{j}\right)}{\left|\left(p_{l}-p_{j}\right) \times\left(p_{m}-p_{j}\right)\right|}
$$

面特征残差雅可比:

$$J_H = \frac{\alpha d_H}{\alpha T} = \frac{\alpha d_H}{\alpha p_i'} \frac{\alpha p_i'}{\alpha T}$$

对平移和旋转的雅可比同上:

对平移的雅可比:

$$\frac{\alpha p_i'}{\alpha t} = I$$

对旋转的雅可比:

$$\frac{\alpha p_i'}{\alpha R} = -(Rp_i)_\times$$

求等号右边第一项的偏导数为:

$$\frac{\alpha d_H}{\alpha p_i'} = \frac{((p_l - p_j) \times (p_m - p_j))^T}{|(p_l - p_j) \times (p_m - p_j)|}$$

二.代码实现和评估

运行

代码运行命令:

```
roslaunch lidar_localization front_end.launch
```

evo工具运行命令:

evo_rpe:

```
evo_rpe kitti ground_truth.txt laser_odom.txt -r trans_part --delta 100 --plot --plot_mode xyz
```

evo_ape:

```
evo_ape kitti ground_truth.txt laser_odom.txt -r full --plot --plot_mode xyz
```

播放数据集命令:

```
rosbag  play kitti_lidar_only_2011_10_03_drive_0027_synced.bag
```

轨迹会自动保存下来，要注意的是，要做对比实验的话，要及时删除轨迹数据，不然会在txt文件中累加数据。

主要代码

主要运行了4个节点:

scan_registration_node：点云处理

aloam_laser_odometry_node：前端里程计

aloam_mapping_node：后端优化建图

evaluation_node：用于保存里程计

scan_registration_node：

ScanRegistration类的Update方法为主要方法，其函数分别：

FilterByRange：点云滤波

SortPointCloudByScan：求出线号（垂直角度）、水平角度、点云时间（用于去畸变），并存在intensity。

GetFeaturePoints：通过曲率计算得到分类后的点云特征。

aloam_laser_odometry_node：

主要是做帧间的关联，包括线特征关联和面特征关联，然后构建优化问题求解两帧之间的相对位姿。这里的关键是是求得残差关于待求变量的雅可比矩阵。

具体原理：

线特征关联：

在第k+1帧中找到一个线特征点（曲率最大），转换到k帧坐标系下（根据运动模型或者其他传感器猜测得到），然后在第k帧中找与k+1帧中特征点最近的一个点，然后再找相邻的一个点，组成一条线，那么第k+1帧的点到第k帧的线就有了，可以构建优化问题了。

面特征关联：

面特征也一样，先在第k帧搜索离第k+1帧面点（曲率最小）最近的一个点，然后在第k帧是找同根线的一个点和相邻线的一个点，三点构建平面，然后构建点面残差方差，进行优化求解。

aloam_mapping_node：

原始loam的帧和地图做匹配，地图已经是无序的点云了，是做线和平面拟合来构建残差。没有保存每个关键帧的位姿和点云，进行后端优化后的调整，这是比较不足的。

ceres优化核心代码：

aloam_laser_odometry_node和aloam_mapping_node两个节点的优化问题本质上是一样的，都是构建特征关联，前者是帧与帧的，后者是帧与地图的。所以问题求解的关键就在于如何用ceres构建优化问题并完成求解。

aloam_factor.hpp:课程自带

aloam_analytic_factor.hpp：解析式求导

```
#ifndef LIDAR_LOCALIZATION_MODELS_ALOAM_ANALYTIC_FACTOR_HPP_
#define LIDAR_LOCALIZATION_MODELS_ALOAM_ANALYTIC_FACTOR_HPP_
#include <eigen3/Eigen/Dense>
#include <sophus/so3.hpp>
#include <sophus/se3.hpp>

#include <ceres/ceres.h>
#include <ceres/rotation.h>

Eigen::Matrix<double,3,3> skew(Eigen::Matrix<double,3,1>& mat_in){
  //  反对称矩阵定义
    Eigen::Matrix<double,3,3> skew_mat;
    skew_mat.setZero();
    skew_mat(0,1) = -mat_in(2);
    skew_mat(0,2) =  mat_in(1);
    skew_mat(1,2) = -mat_in(0);
```

```
    skew_mat(1,0) =  mat_in(2);
    skew_mat(2,0) = -mat_in(1);
    skew_mat(2,1) =  mat_in(0);
    return skew_mat;
}

class  EdgeAnalyticCostFunction   :  public   ceres::SizedCostFunction<1, 4,  3>
{              // 优化参数维度：1     输入维度 ： q : 4   t : 3
public:
        double s;
        Eigen::Vector3d curr_point, last_point_a, last_point_b;
        EdgeAnalyticCostFunction(const Eigen::Vector3d curr_point_, const
Eigen::Vector3d last_point_a_,
                                       const Eigen::Vector3d last_point_b_,
const double s_  )
                : curr_point(curr_point_),  last_point_a(last_point_a_),
last_point_b(last_point_b_) , s(s_) {}

virtual  bool  Evaluate(double  const  *const  *parameters,
                                             double  *residuals,
                                             double  **jacobians) const
                        //   定义残差模型
{
        Eigen::Map<const  Eigen::Quaterniond>   q_last_curr(parameters[0]);
          //   存放 w  x y z
        Eigen::Map<const  Eigen::Vector3d>      t_last_curr(parameters[1]);
        Eigen::Vector3d  lp ;                       //   line point
        Eigen::Vector3d  lp_r ;
        lp_r =  q_last_curr*curr_point;
        lp      =   q_last_curr  * curr_point  + t_last_curr;   //   new point
        Eigen::Vector3d  nu =  (lp - last_point_a).cross(lp - last_point_b);
        Eigen::Vector3d  de = last_point_a  - last_point_b;

        residuals[0]  =  nu.norm()  /  de.norm();
 //  线残差

        //  归一单位化
        nu.normalize();

        if (jacobians !=  NULL)
        {
                if (jacobians[0]  !=  NULL)
                {
                        Eigen::Vector3d  re = last_point_b  -   last_point_a;
                        Eigen::Matrix3d  skew_re  =   skew(re);

                        //  J_so3_Rotation
                        Eigen::Matrix3d   skew_lp_r  =  skew(lp_r);
                        Eigen::Matrix3d    dp_by_dr;
                        dp_by_dr.block<3,3>(0,0)  =  -skew_lp_r;
                        Eigen::Map<Eigen::Matrix<double, 1, 4, Eigen::RowMajor>>
J_so3_r(jacobians[0]);
                        J_so3_r.setZero();
                        J_so3_r.block<1,3>(0,0)  =   nu.transpose()* skew_re *
dp_by_dr / (de.norm()*nu.norm());


                        //  J_so3_Translation
```

```
                        Eigen::Matrix3d  dp_by_dt;
                        (dp_by_dt.block<3,3>(0,0)).setIdentity();
                        Eigen::Map<Eigen::Matrix<double,  1,  3,
 Eigen::RowMajor>> J_so3_t(jacobians[1]);
                        J_so3_t.setZero();
                        J_so3_t.block<1,3>(0,0)  =   nu.transpose()  *  skew_re
/ (de.norm()*nu.norm());
                }
        }
        return  true;
}
};

class PlaneAnalyticCostFunction  :  public  ceres::SizedCostFunction<1, 4, 3>{
public:
    Eigen::Vector3d curr_point, last_point_j, last_point_l, last_point_m;
    Eigen::Vector3d ljm_norm;
    double s;

        PlaneAnalyticCostFunction(Eigen::Vector3d curr_point_, Eigen::Vector3d
last_point_j_,
                     Eigen::Vector3d last_point_l_, Eigen::Vector3d
last_point_m_, double s_)
        : curr_point(curr_point_), last_point_j(last_point_j_),
last_point_l(last_point_l_),last_point_m(last_point_m_), s(s_){}

        virtual  bool  Evaluate(double  const  *const  *parameters,
                                                        double  *residuals,
                                                         double
 **jacobians)const {      //   定义残差模型
                // 叉乘运算， j,l,m 三个但构成的平行四边面积(摸)和该面的单位法向量(方向)
                Eigen::Vector3d  ljm_norm = (last_point_j -
last_point_l).cross(last_point_j - last_point_m);
        ljm_norm.normalize();    //  单位法向量

                Eigen::Map<const Eigen::Quaterniond>
 q_last_curr(parameters[0]);
                Eigen::Map<const Eigen::Vector3d> t_last_curr(parameters[1]);

                Eigen::Vector3d  lp;      // “从当前阵的当前点” 经过转换矩阵转换到“上一
阵的同线束激光点”
                Eigen::Vector3d  lp_r = q_last_curr *  curr_point ;
            //  for compute jacobian o rotation  L: dp_dr
                lp = q_last_curr *  curr_point  +  t_last_curr;

                // 残差函数
                double  phi1 =  (lp - last_point_j ).dot(ljm_norm);
                residuals[0]  =   std::fabs(phi1);

                if(jacobians != NULL)
                {
                        if(jacobians[0] != NULL)
                        {
                                phi1 = phi1  /  residuals[0];
                                //  Rotation
                                Eigen::Matrix3d  skew_lp_r  = skew(lp_r);
                                Eigen::Matrix3d  dp_dr;
                                dp_dr.block<3,3>(0,0)  =  -skew_lp_r;
```

```
                                Eigen::Map<Eigen::Matrix<double, 1, 4,
Eigen::RowMajor>>  J_so3_r(jacobians[0]);
                                J_so3_r.setZero();
                                J_so3_r.block<1,3>(0,0) =  phi1 *
 ljm_norm.transpose() *  (dp_dr);

                                Eigen::Map<Eigen::Matrix<double, 1, 3,
Eigen::RowMajor>>  J_so3_t(jacobians[1]);
                                J_so3_t.block<1,3>(0,0)  = phi1 *
ljm_norm.transpose();




                        }
                }
                return  true;
        }

};


//  自定义旋转残差块
//参考博客   https://blog.csdn.net/jdy_lyy/article/details/119360492
class PoseSO3Parameterization  :   public  ceres::LocalParameterization {
                        //  自定义so3  旋转块
 public:
        PoseSO3Parameterization()  { }

        virtual ~PoseSO3Parameterization() { }

        virtual bool Plus(const double* x,
                        const double* delta,
                        double* x_plus_delta) const          //参数正切空间上的更新
函数
                {
                        Eigen::Map<const  Eigen::Quaterniond>   quater(x);
//   待更新的四元数
                        Eigen::Map<const  Eigen::Vector3d>     delta_so3(delta);
    //    delta 值,使用流形 so3 更新

                        Eigen::Quaterniond  delta_quater  =
Sophus::SO3d::exp(delta_so3).unit_quaternion();     //   so3 转换位 delta_p  四元数

                        Eigen::Map<Eigen::Quaterniond>
 quter_plus(x_plus_delta);    //    更新后的四元数

                        // 旋转更新公式
                        quter_plus =  (delta_quater*quater).normalized();


                        return  true;
                }

        virtual bool ComputeJacobian(const double* x, double* jacobian) const
 //  四元数对so3的偏导数
        {
                Eigen::Map<Eigen::Matrix<double, 4, 3, Eigen::RowMajor>>
j(jacobian);
```

```
                (j.topRows(3)).setIdentity();
                (j.bottomRows(1)).setZero();

                return true;
        }

        // virtual bool MultiplyByJacobian(const double* x,
        //                                   const int num_rows,
        //                                   const double* global_matrix,
        //                                   double* local_matrix) const;//一般不用

        virtual int GlobalSize() const  {return  4;} // 参数的实际维数
        virtual int LocalSize() const   {return  3;} // 正切空间上的参数维数
};


#endif
```

aloam_registration.cpp:

```
/*
 * @Description: LOAM scan registration, implementation
 * @Author: Ge Yao
 * @Date: 2021-05-04 14:53:21
 */

#include <chrono>

#include "glog/logging.h"

#include "lidar_localization/models/loam/aloam_factor.hpp"
#include "lidar_localization/models/loam/aloam_analytic_factor.hpp"
#include "lidar_localization/models/loam/aloam_registration.hpp"
#include "lidar_localization/global_defination/global_defination.h"

namespace lidar_localization {

CeresALOAMRegistration::CeresALOAMRegistration(const Eigen::Quaternionf &dq,
const Eigen::Vector3f &dt) {

    //
    // config optimizer:
    //
    std::string config_file_path = WORK_SPACE_PATH +
"/config/front_end/config.yaml";

    YAML::Node config_node = YAML::LoadFile(config_file_path);
    grade_way_name = config_node["grade_way"].as<std::string>();
    param_block_name = config_node["param_block"].as<std::string>();
    std::cout<<"grade_way:  "<<grade_way_name<<std::endl;
    std::cout<<"param_block:  "<<param_block_name<<std::endl;

    // 1. parameterization:
    if(param_block_name == "user_defined")  config_.q_parameterization_ptr =
 new  PoseSO3Parameterization() ;                           //  自定义旋转参数块
```

```cpp
    if(param_block_name == "ceres_defined")  config_.q_parameterization_ptr =
new ceres::EigenQuaternionParameterization();
    //   SE3 转换矩阵/位姿参数化，基类ceres派生 ，  构造顺序( x y z  w)

    // 2. loss function:
    // TODO: move param to config
    config_.loss_function_ptr =  new ceres::HuberLoss(0.10);

    // 3. solver:
    config_.options.linear_solver_type = ceres::DENSE_QR;
    // config_.options.use_explicit_schur_complement = true;
    // config_.options.trust_region_strategy_type = ceres::DOGLEG;
    // config_.options.use_nonmonotonic_steps = true;
    config_.options.num_threads = 2;
    config_.options.max_num_iterations = 50;
    config_.options.minimizer_progress_to_stdout = false;
    config_.options.max_solver_time_in_seconds = 0.10;

    //
    // config target variables:
    //
    param_.q[0] = dq.x(); param_.q[1] = dq.y(); param_.q[2] = dq.z();
param_.q[3] = dq.w();
    param_.t[0] = dt.x(); param_.t[1] = dt.y(); param_.t[2] = dt.z();
    problem_.AddParameterBlock(param_.q, 4, config_.q_parameterization_ptr);
    problem_.AddParameterBlock(param_.t, 3);
}

CeresALOAMRegistration::~CeresALOAMRegistration() {
}

/**
  * @brief  add residual block for edge constraint from lidar frontend
  * @param  source, source point
  * @param  target_x, target point x
  * @param  target_y, target point y
  * @param  ratio, interpolation ratio
  * @return void
  */
bool CeresALOAMRegistration::AddEdgeFactor(
    const Eigen::Vector3d &source,
    const Eigen::Vector3d &target_x, const Eigen::Vector3d &target_y,
    const double &ratio
) {
    /*自动求导*/
    if(grade_way_name == "autograde")
    {
        ceres::CostFunction *factor_edge = LidarEdgeFactor::Create(
        source,
        target_x, target_y,
        ratio
    );

    problem_.AddResidualBlock(
        factor_edge,
        config_.loss_function_ptr,
        param_.q, param_.t
    );
```

```
    }
    if(grade_way_name == "analytic_grade")
    {

       /*解析求导*/
        ceres::CostFunction *factor_analytic_edge =   new
EdgeAnalyticCostFunction(
                source,
                target_x, target_y,
                ratio
        );

        problem_.AddResidualBlock(
            factor_analytic_edge,                               //   约束边
cost_function
            config_.loss_function_ptr,        //   鲁棒核函数  lost_function
            param_.q, param_.t                      //  关联参数
        );
    }

    return true;
}

/**
  * @brief  add residual block for plane constraint from lidar frontend
  * @param  source, source point
  * @param  target_x, target point x
  * @param  target_y, target point y
  * @param  target_z, target point z
  * @param  ratio, interpolation ratio
  * @return void
  */
bool CeresALOAMRegistration::AddPlaneFactor(
    const Eigen::Vector3d &source,
    const Eigen::Vector3d &target_x, const Eigen::Vector3d &target_y, const
Eigen::Vector3d &target_z,
    const double &ratio
) {
    /*自动求导*/
    if(grade_way_name == "autograde")
    {
        ceres::CostFunction *factor_plane = LidarPlaneFactor::Create(
            source,
            target_x, target_y, target_z,
            ratio
        );

        problem_.AddResidualBlock(
            factor_plane,
            config_.loss_function_ptr,
            param_.q, param_.t
        );
    }
    if(grade_way_name == "analytic_grade")
    {
   /*解析求导*/
```

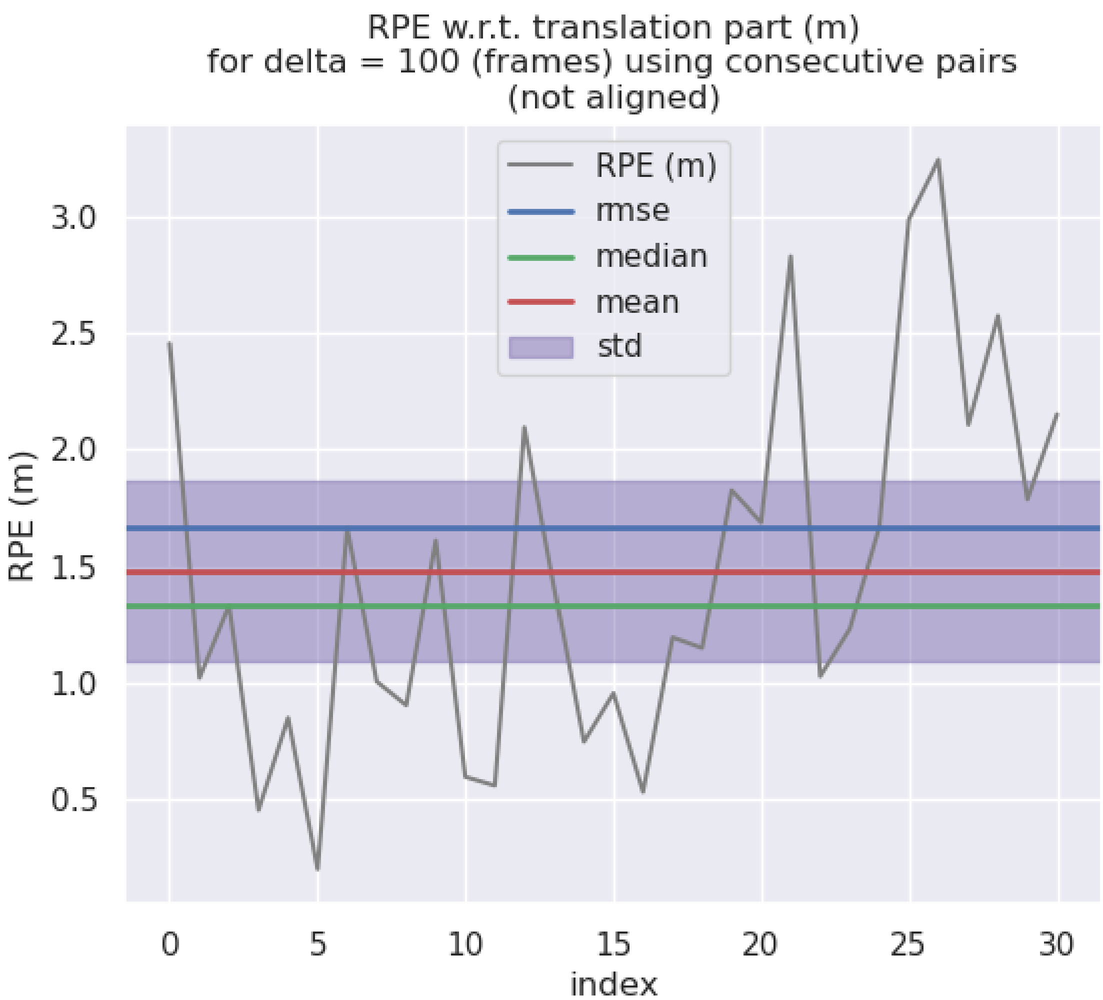
```
        ceres::CostFunction *factor_analytic_plane =new
PlaneAnalyticCostFunction(
            source,
            target_x, target_y, target_z,
            ratio
        );

        problem_.AddResidualBlock(
            factor_analytic_plane,
            config_.loss_function_ptr,
            param_.q, param_.t
        );
    }

    return true;
}

bool CeresALOAMRegistration::Optimize() {
    // solve:
    ceres::Solver::Summary summary;

    // time it:
    auto start = std::chrono::steady_clock::now();

    ceres::Solve(config_.options, &problem_, &summary);

    auto end = std::chrono::steady_clock::now();
    std::chrono::duration<double> time_used = end - start;

    // prompt:
    LOG(INFO) << "Time Used: " << time_used.count() << " seconds." << std::endl
                << "Cost Reduced: " << summary.initial_cost - summary.final_cost
<< std::endl
                << summary.BriefReport() << std::endl
                << std::endl;

    return true;
}

/**
  * @brief  get optimized relative pose
  * @return true if success false otherwise
  */
bool CeresALOAMRegistration::GetOptimizedRelativePose(Eigen::Quaternionf &dq,
Eigen::Vector3f &dt) {
    Eigen::Quaternionf q(param_.q[0], param_.q[1], param_.q[2], param_.q[3]);
    Eigen::Vector3f t(param_.t[0], param_.t[1], param_.t[2]);

    dq = q;
    dt = t;

    return true;
}

} // namespace graph_ptr_optimization
```

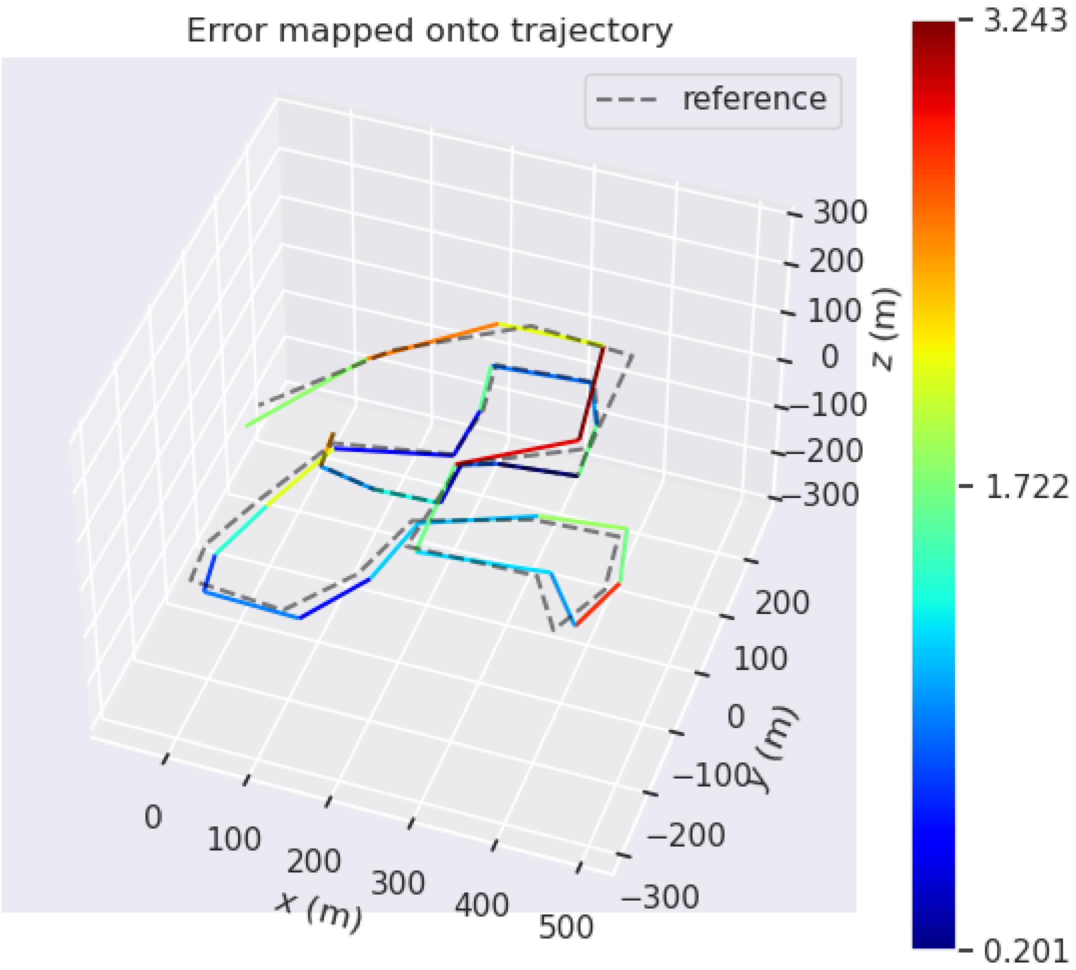
课程代码：ALOAM-自动求导+ceres自带参数块

rviz效果

evo评估

RPE



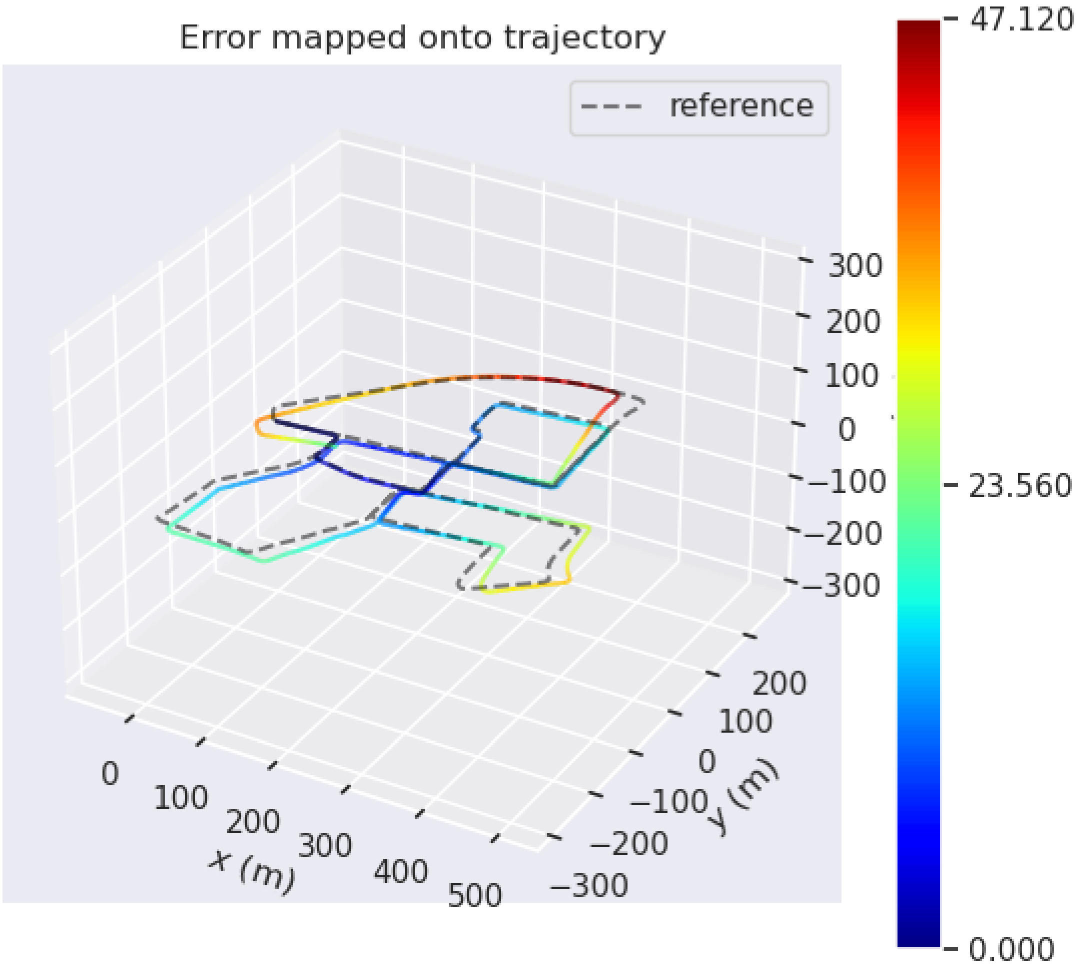


RPE指标：

```
  max    3.243457
  mean   1.479168
median   1.332409
   min   0.201085
  rmse   1.669851
   sse   86.440470
   std   0.774896
```

APE


APE w.r.t. full transformation (unit-less)
(not aligned)
APE (unit-less)
rmse
median
mean
std
APE (unit-less)
index
0
10
20
30
40
0
500
1000
1500
2000
2500
3000

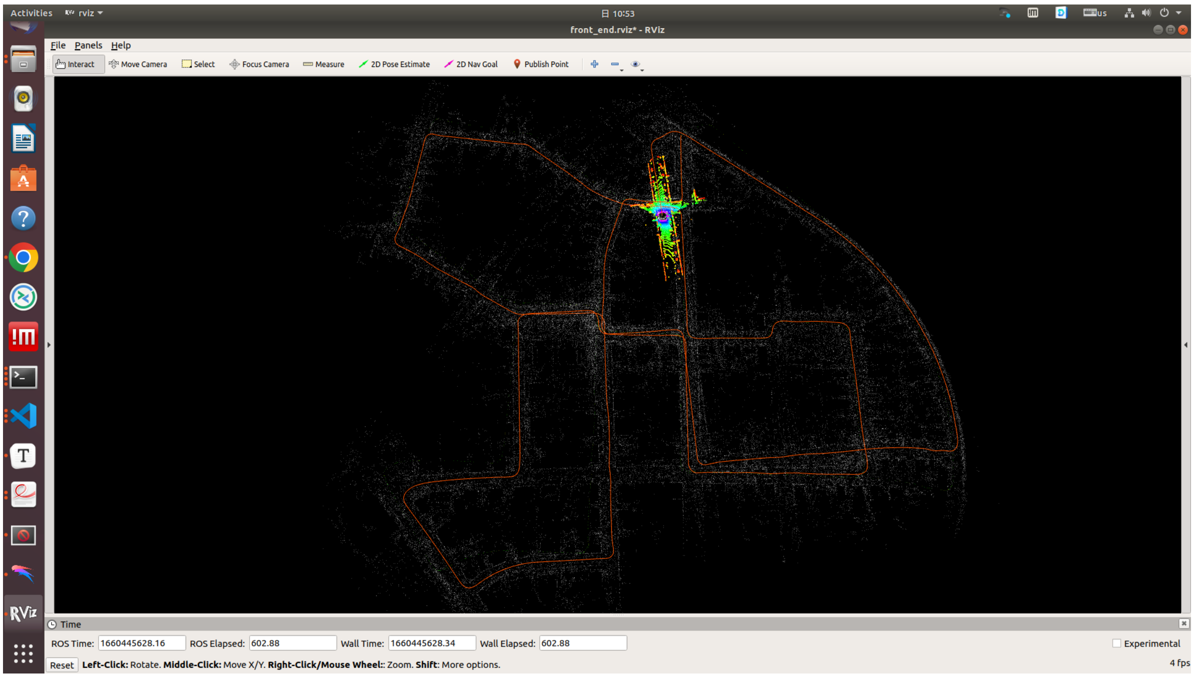

Error mapped onto trajectory
reference
47.120
23.560
0.000
x (m)
y (m)
0
100
200
300
400
500
−300
−200
−100
0
100
200
300

APE指标：

```
   max   47.119680
  mean   17.677912
median   15.351334
   min   0.000001
  rmse   20.849407
   sse   1377557.237589
   std   11.053922
```

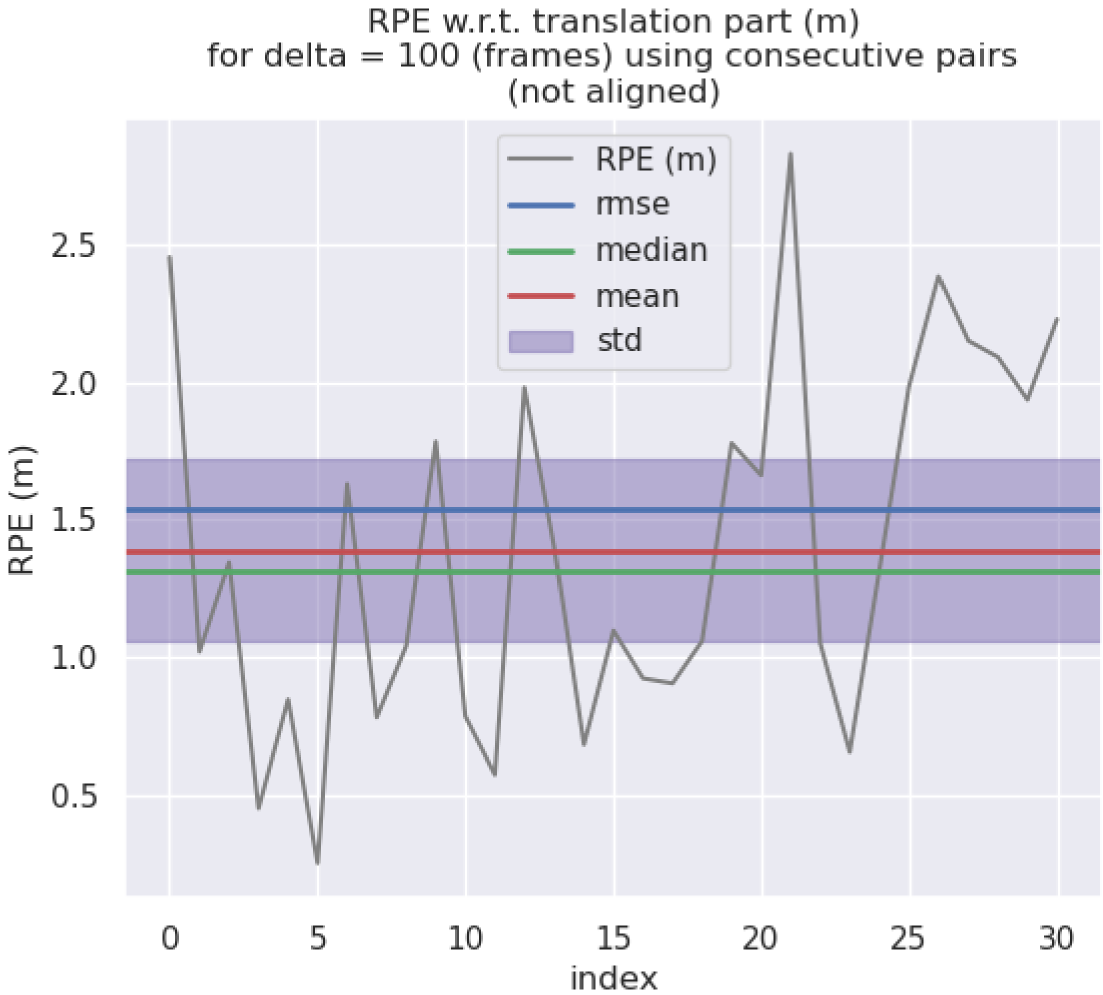
ALOAM-自动求导+自定义参数块

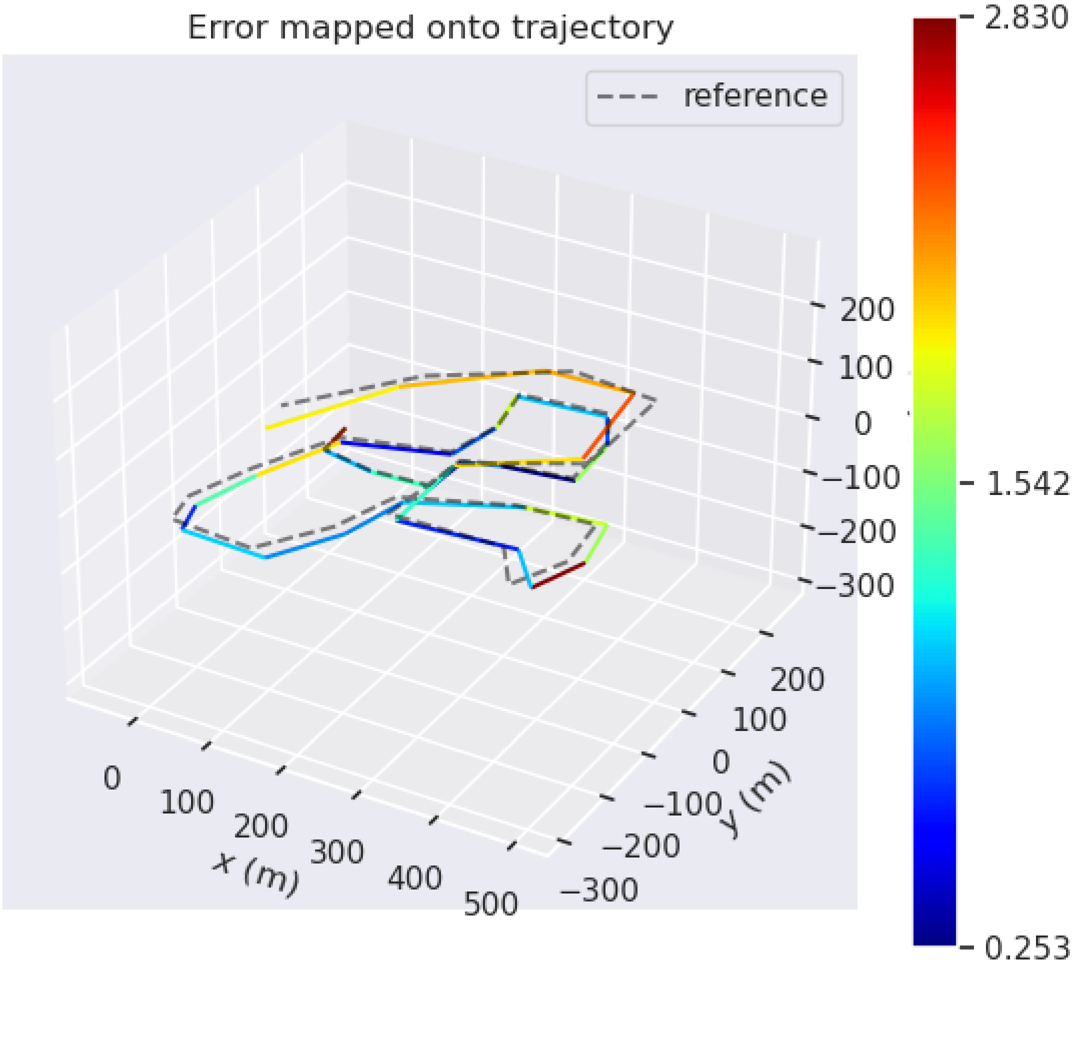
rviz效果



evo评估

RPE

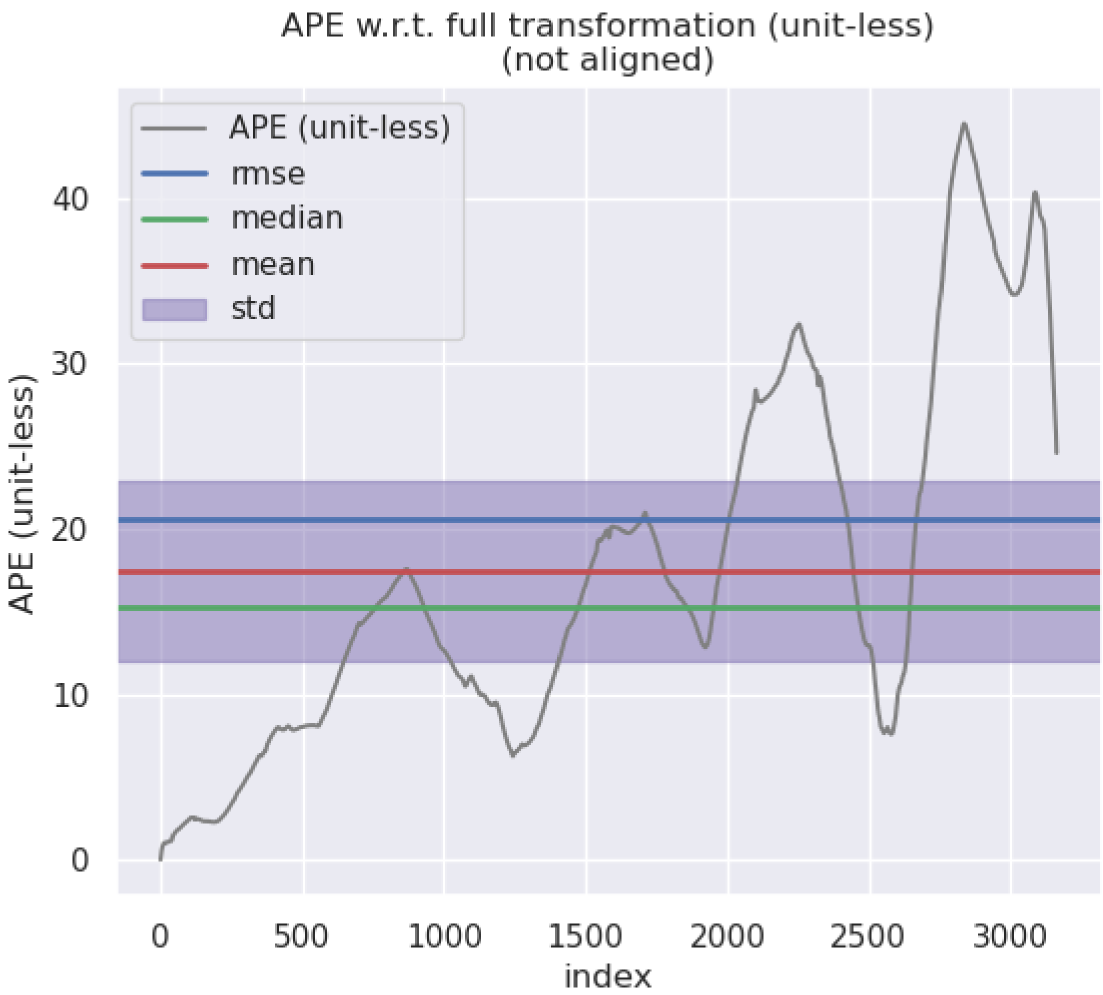

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30
index

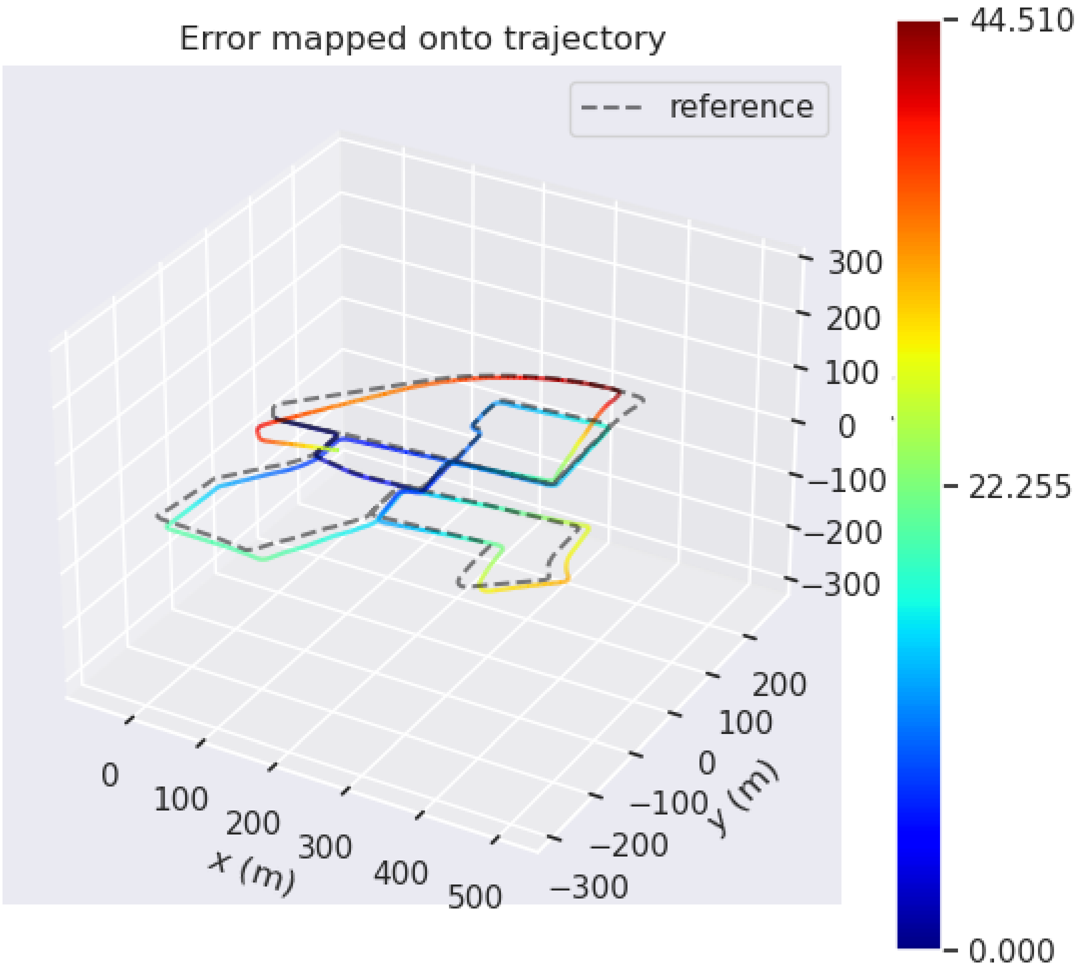

Error mapped onto trajectory
reference
2.830
1.542
0.253
200
100
0
−100
−200
−300
x (m)
y (m)
500
400
300

RPE指标：

```
   max    2.830145
   mean   1.390351
 median   1.315126
    min   0.253277
   rmse   1.537500
    sse   73.281047
    std   0.656375
```

APE



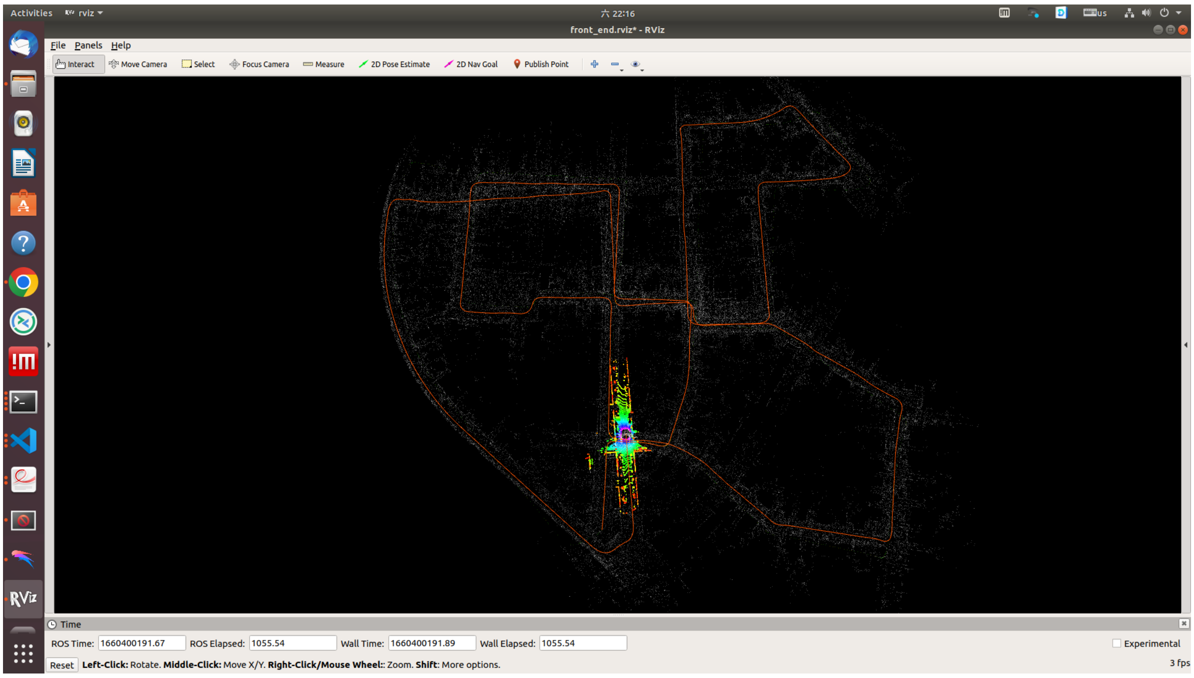


APE指标：

```
   max  44.510189
  mean  17.500540
median  15.274735
   min  0.000001
  rmse  20.628949
   sse  1346025.807956
   std  10.921750
```

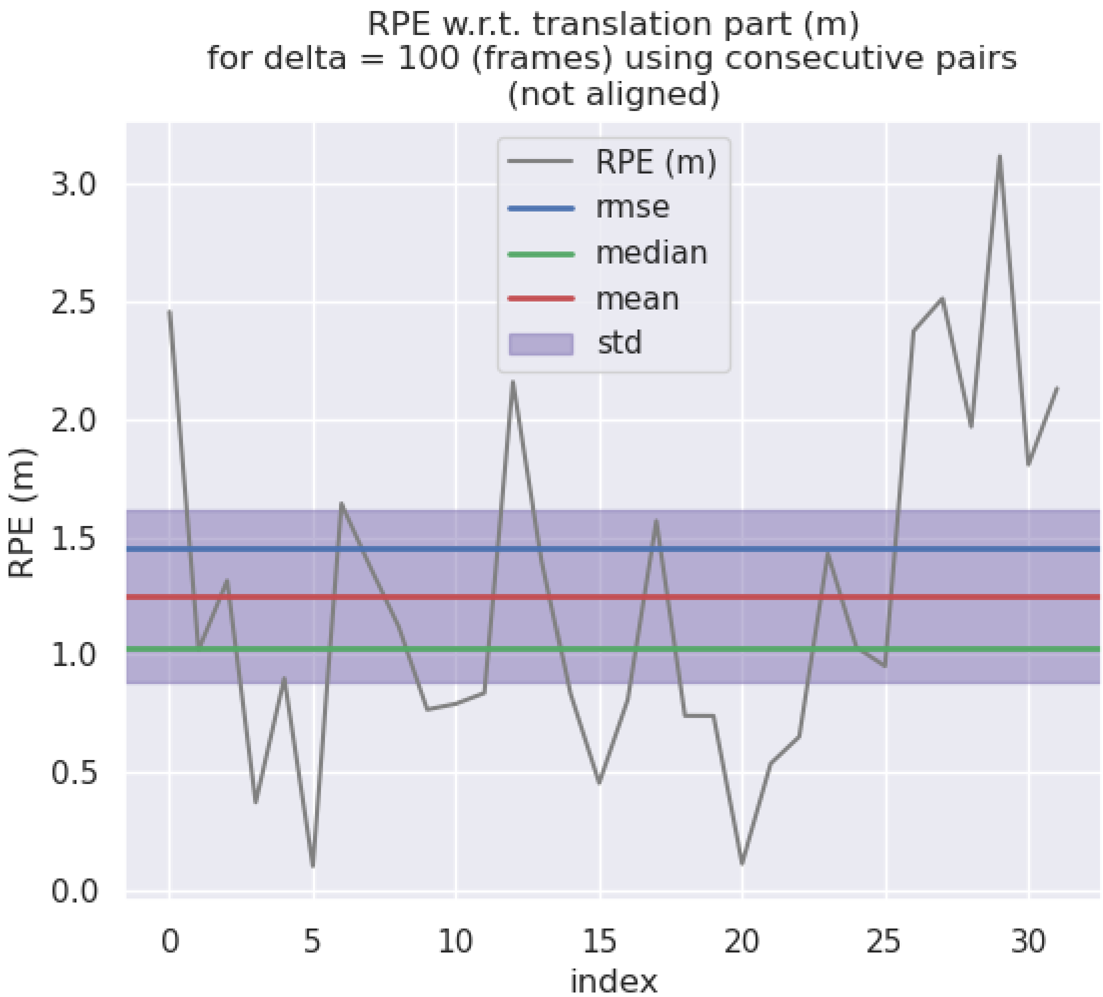
ALOAM-解析式求导+自定义参数块

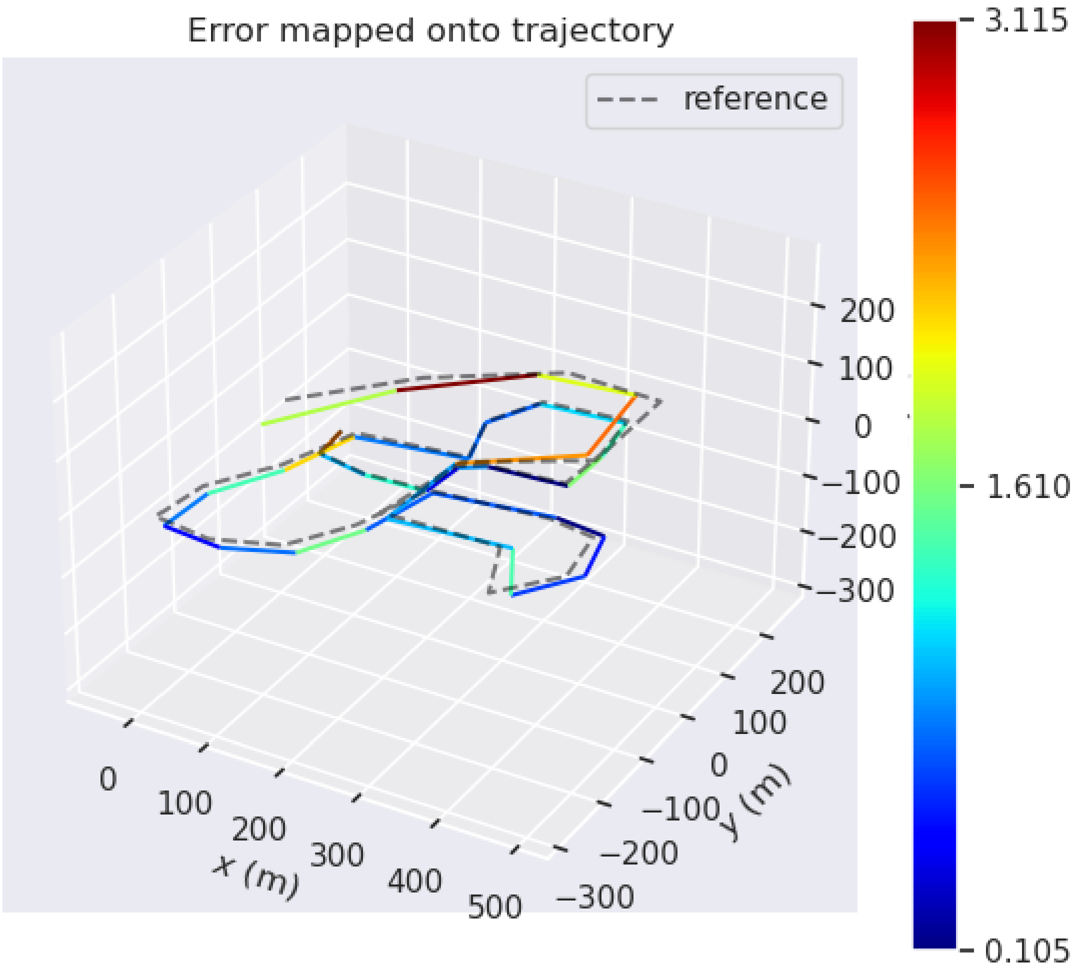
rviz效果


Activities
front_end.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 1660400191.67
ROS Elapsed: 1055.54
Wall Time: 1660400191.89
Wall Elapsed: 1055.54
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
3 fps

evo评估

RPE

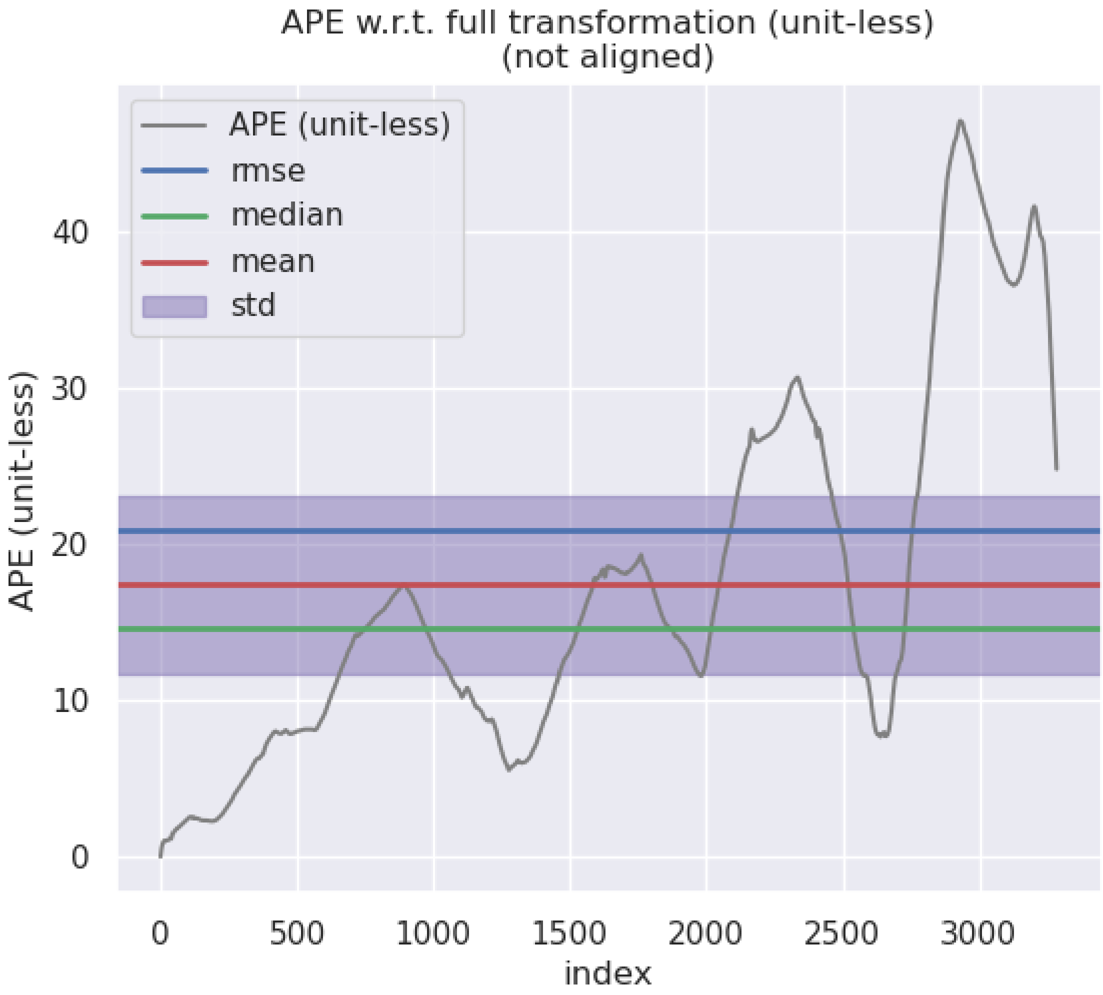

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
RPE (m)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30
index

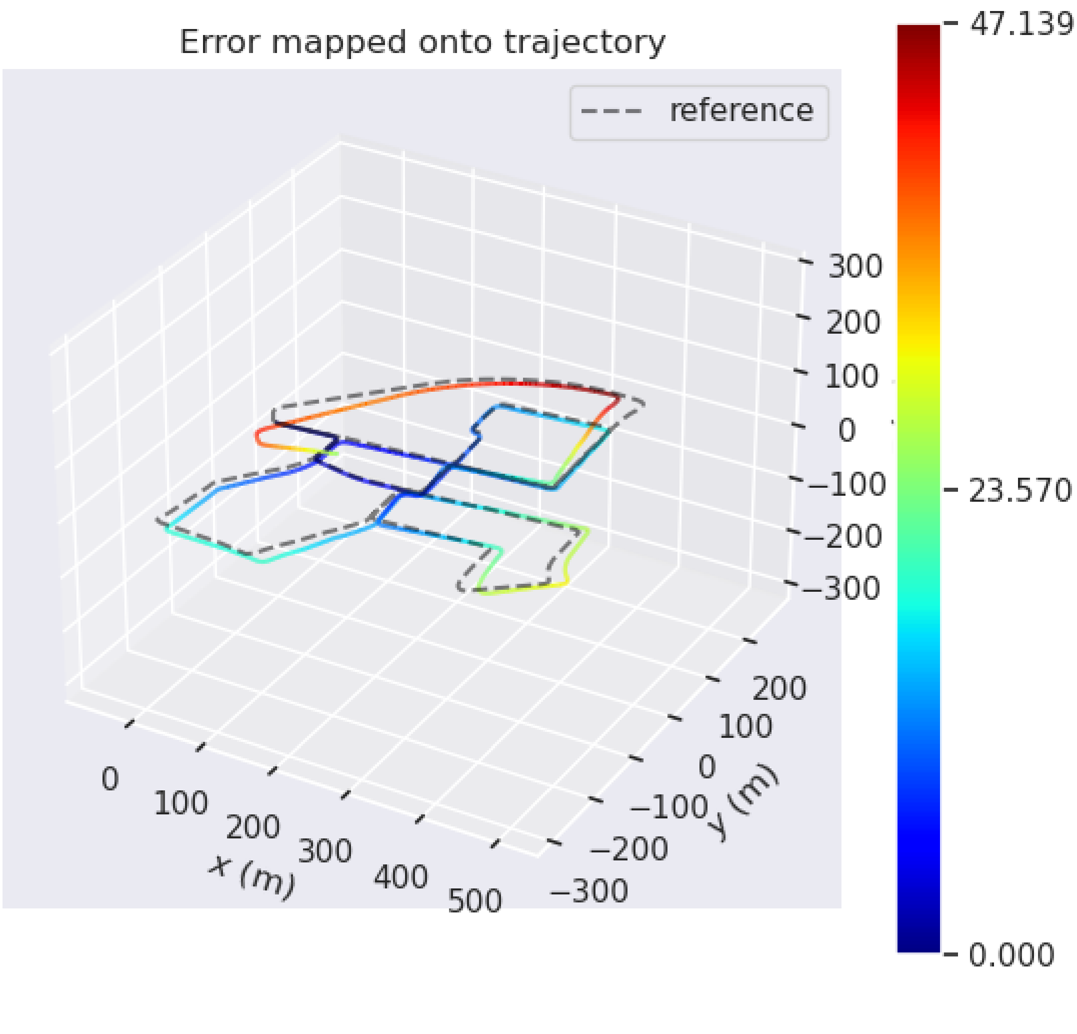


RPE指标：

```
   max   3.115124
  mean   1.251383
median   1.026564
   min   0.104845
  rmse   1.450670
   sse   67.342221
   std   0.733816
```

APE


APE w.r.t. full transformation (unit-less)
(not aligned)
APE (unit-less)
rmse
median
mean
std
APE (unit-less)
index
0
10
20
30
40
0
500
1000
1500
2000
2500
3000

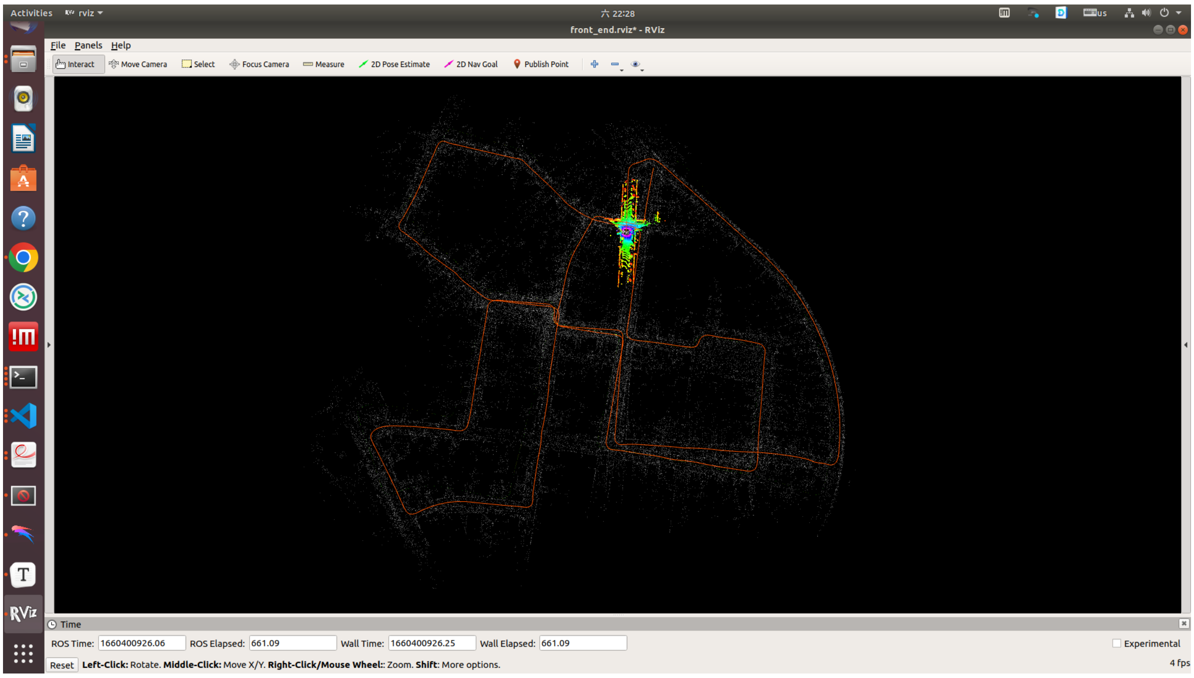

Error mapped onto trajectory
reference
x (m)
y (m)
0
100
200
300
400
500
−300
−200
−100
0
100
200
300
47.139
23.570
0.000

APE指标：

```
   max	47.139245
  mean	17.395350
median	14.574086
   min	0.000001
  rmse	20.862871
   sse	1429827.033418
   std	11.517862
```

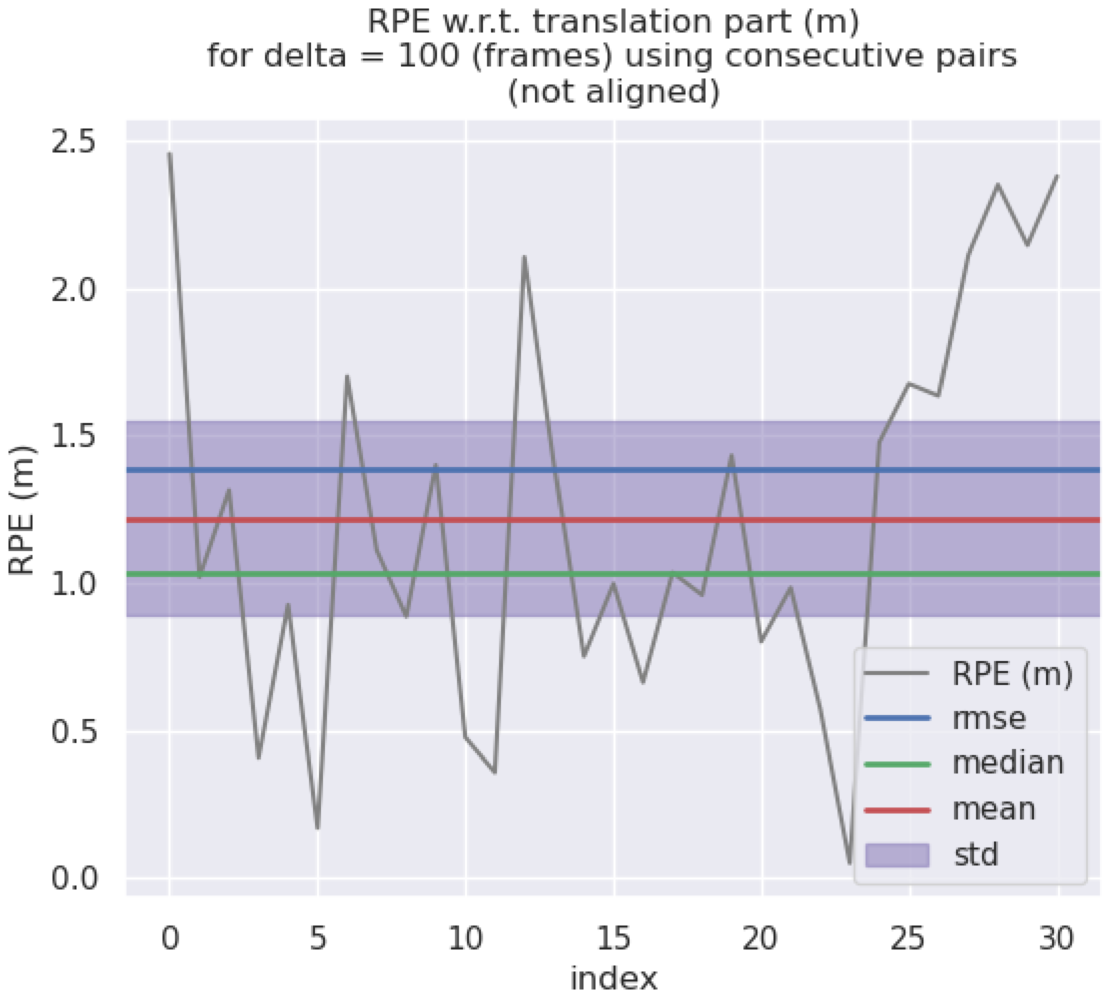
ALOAM-解析式求导+ceres自带参数块

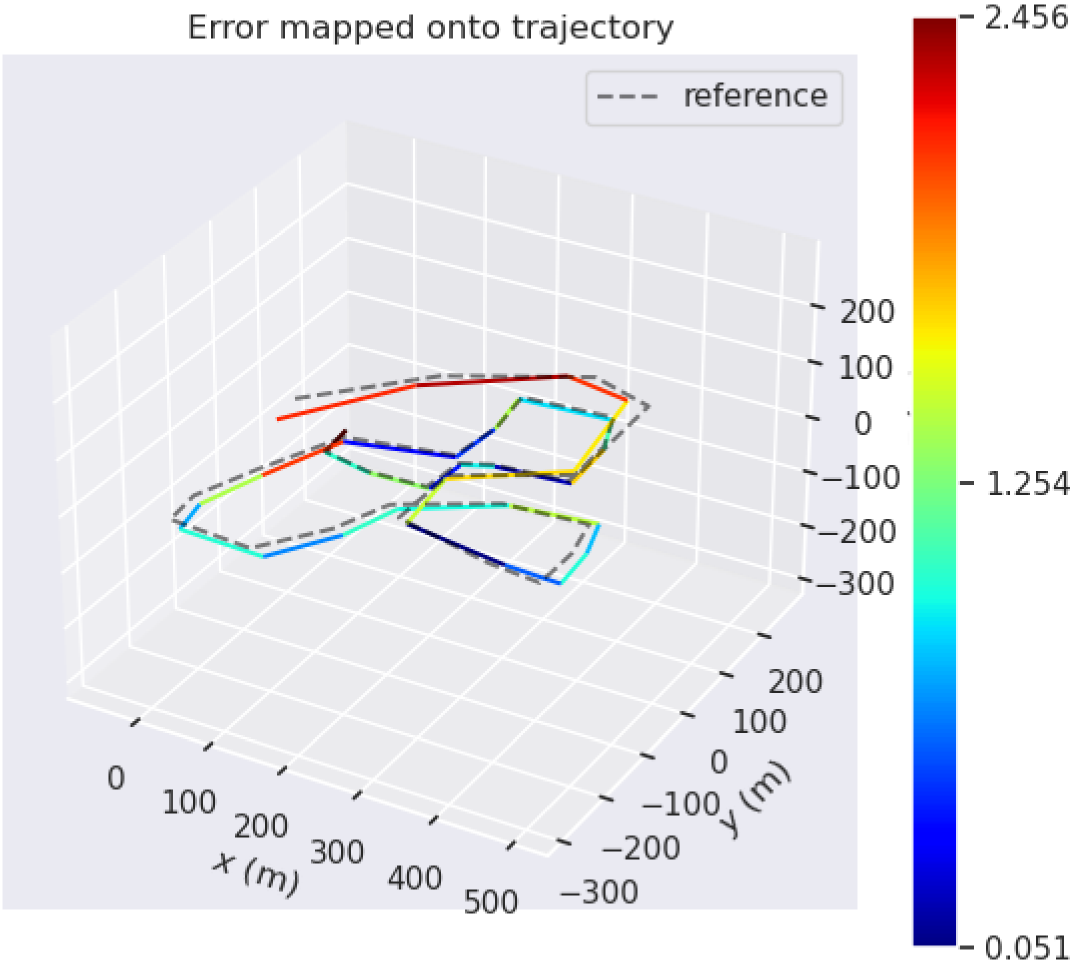
rviz效果



evo评估

RPE

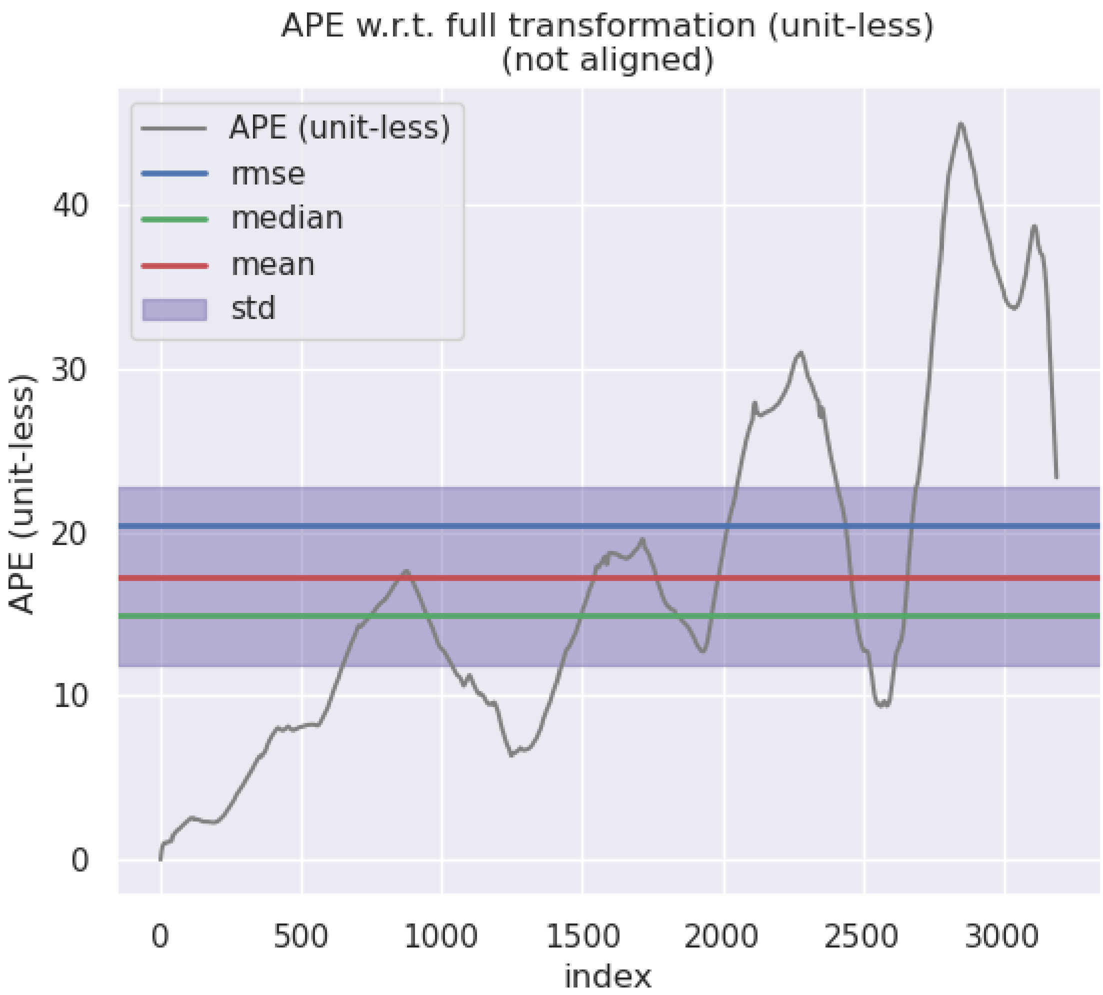

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
index
RPE (m)
rmse
median
mean
std
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30

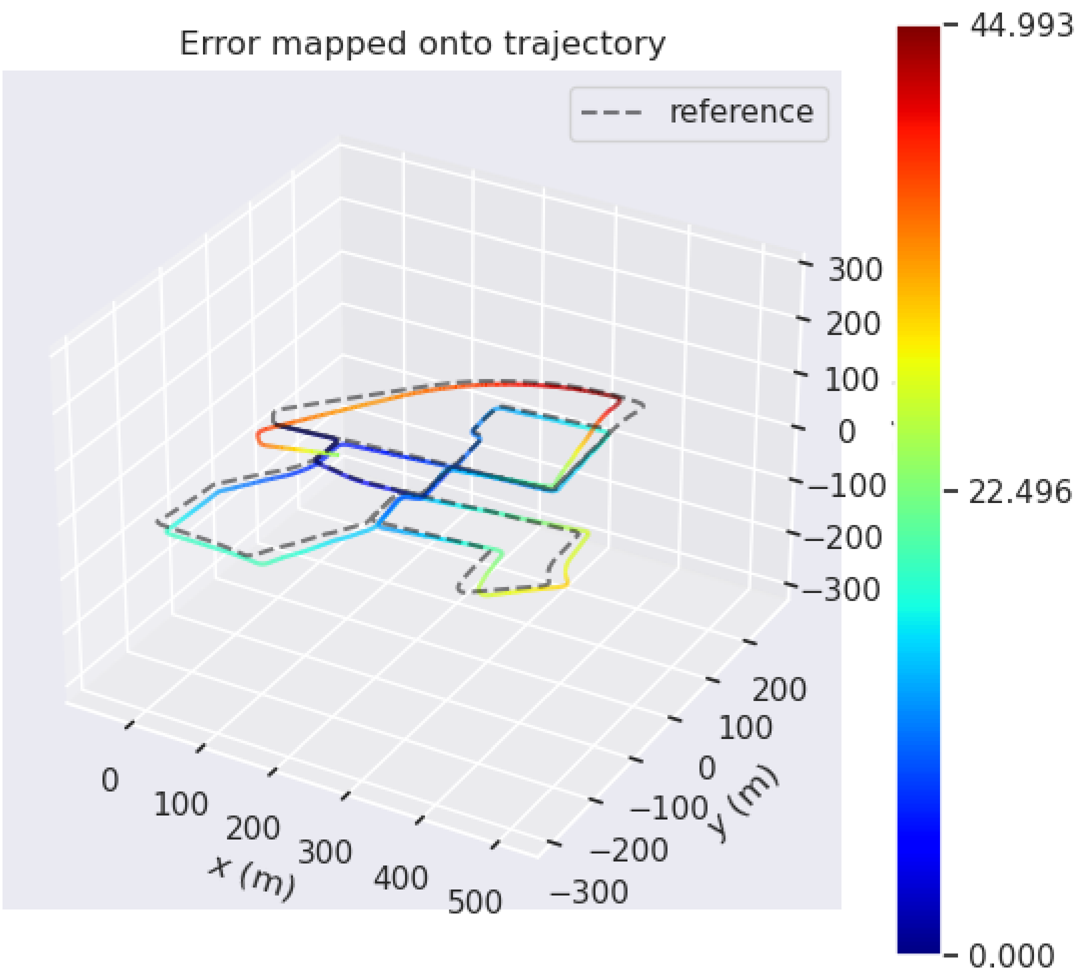

Error mapped onto trajectory
reference
x (m)
y (m)
0
100
200
300
400
500
200
100
0
−100
−200
−300
200
100
0
−100
−200
−300
2.456
1.254
0.051

RPE指标：

```
   max   2.455913
  mean   1.218680
median   1.036104
   min   0.051391
  rmse   1.384630
   sse   59.433238
   std   0.657283
```

APE





APE指标：

```
   max  44.992544
  mean  17.325534
median  14.906351
   min  0.000001
  rmse  20.423035
   sse  1330550.182920
   std  10.813244
```

三.实车部署

实车硬件如下：

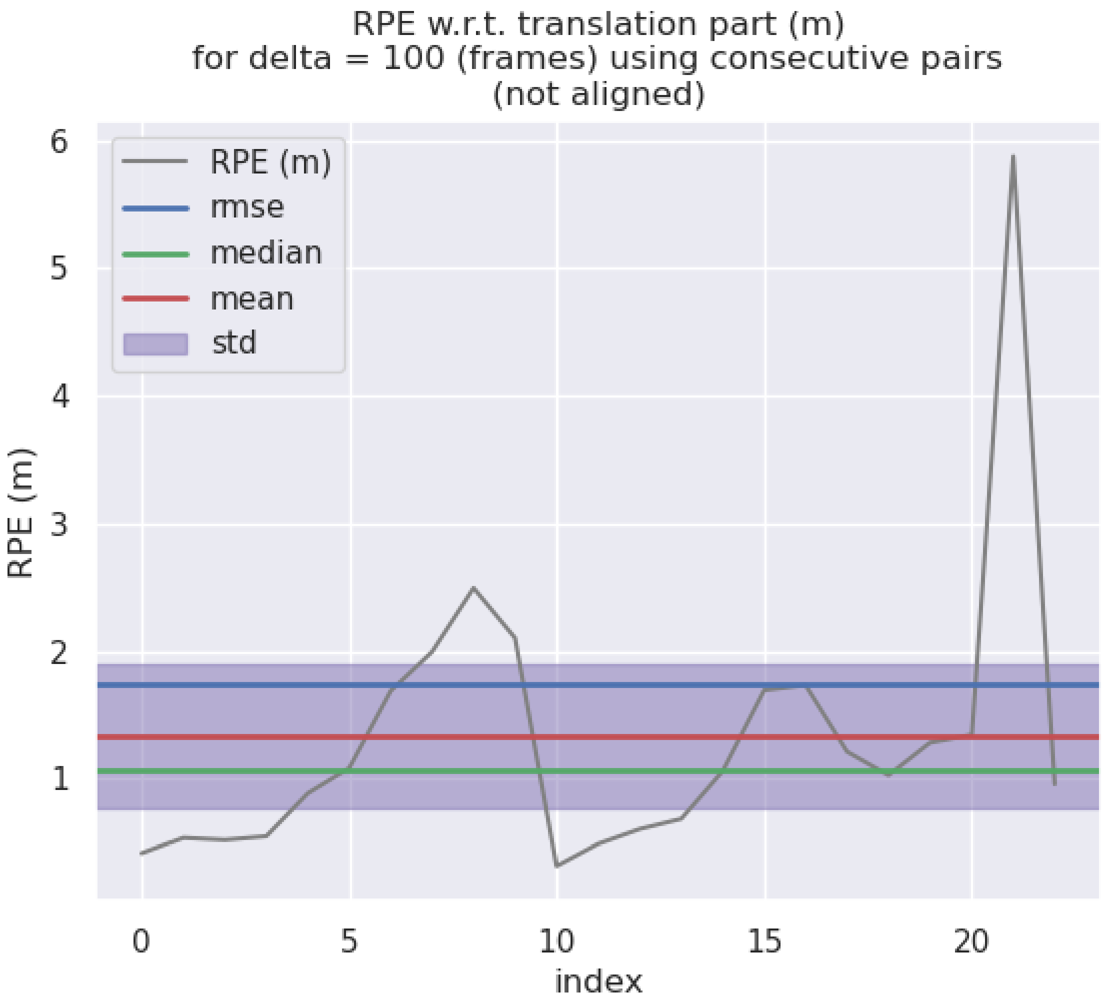
1. 松灵Scout2，车速为1.5m/s
2. 速腾16线雷达
3. SBG-ellipse-N 九轴惯导+单天线RTK

课程代码：ALOAM-自动求导+ceres自带参数块

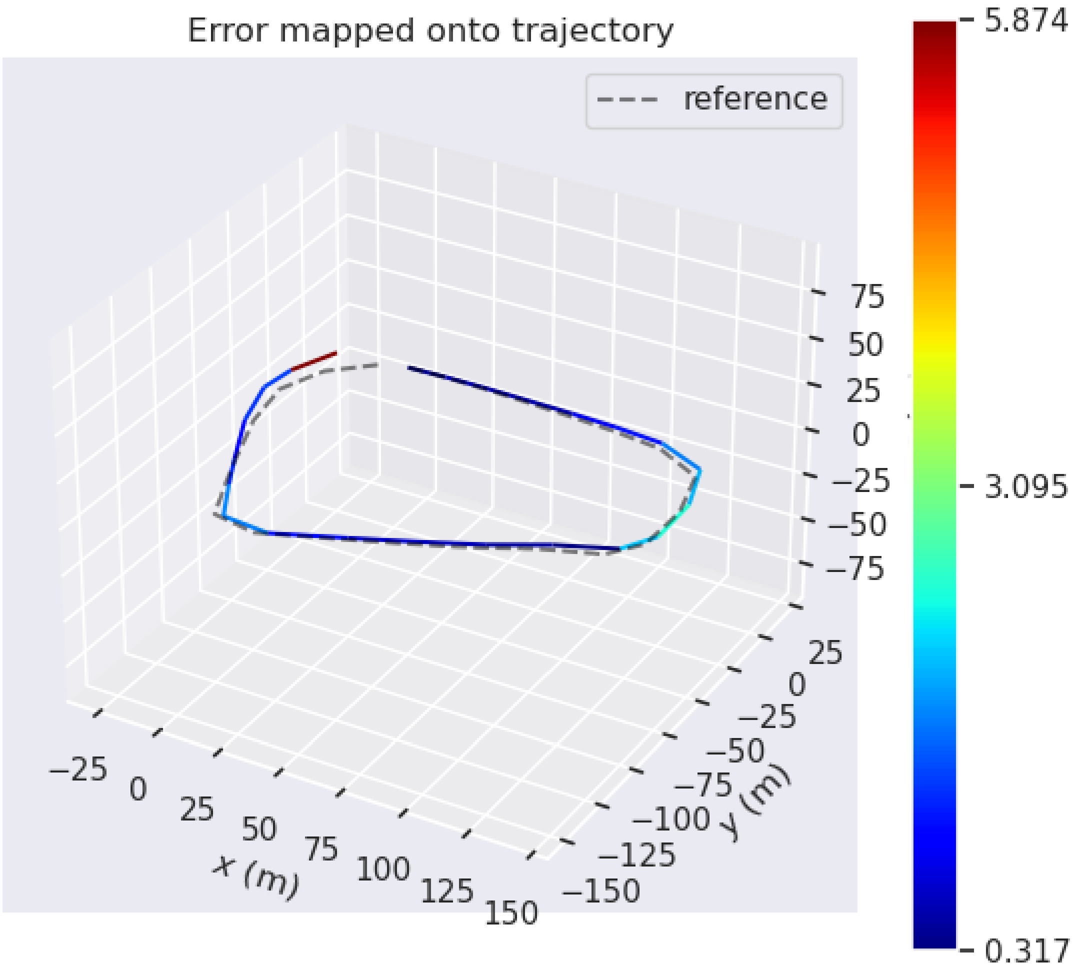
rviz效果

evo评估

RPE



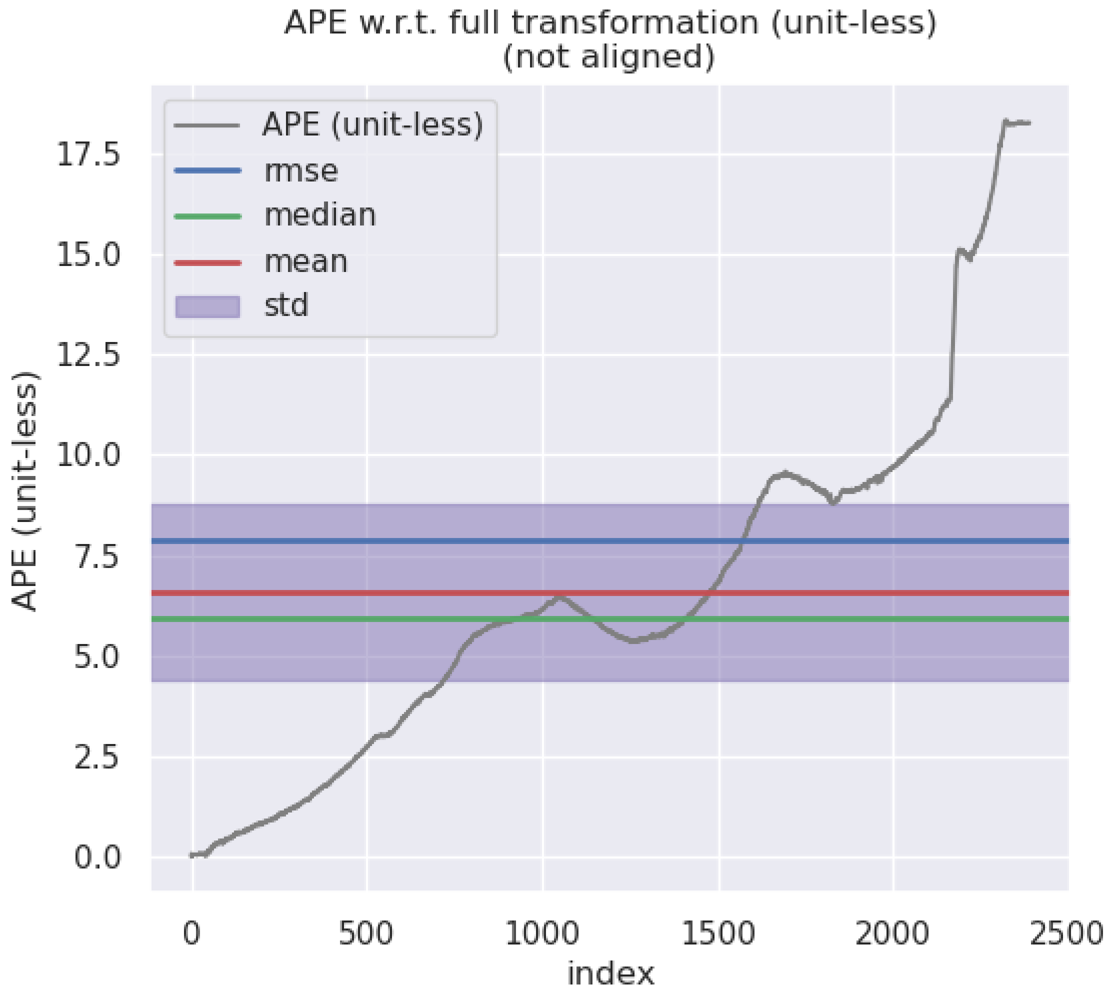
RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
RPE (m)
index
0
5
10
15
20
1
2
3
4
5
6

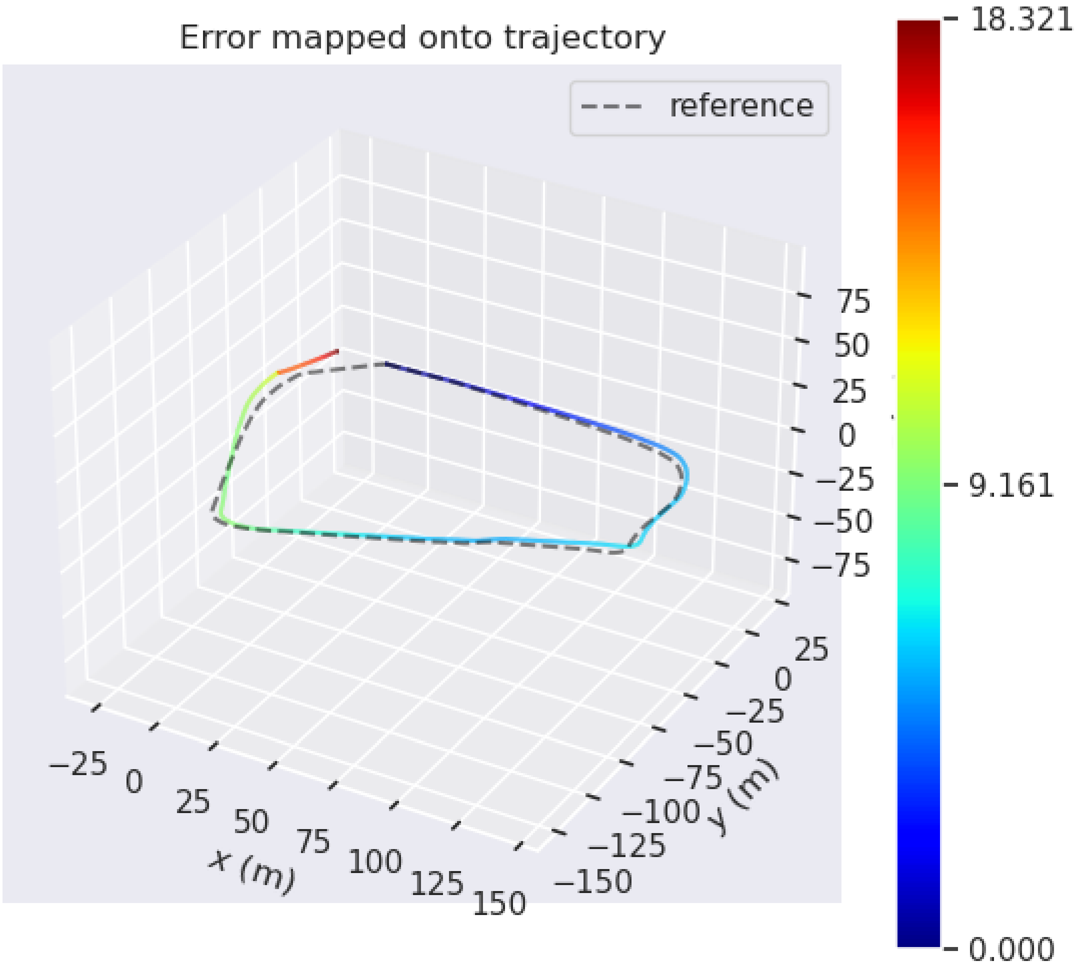


RPE指标：

```
     max	5.873776
    mean	1.327785
  median	1.058711
     min	0.316742
    rmse	1.744557
     sse	70.000057
     std	1.131578
```

APE


APE w.r.t. full transformation (unit-less)
(not aligned)
APE (unit-less)
rmse
median
mean
std
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
APE (unit-less)
0
500
1000
1500
2000
2500
index

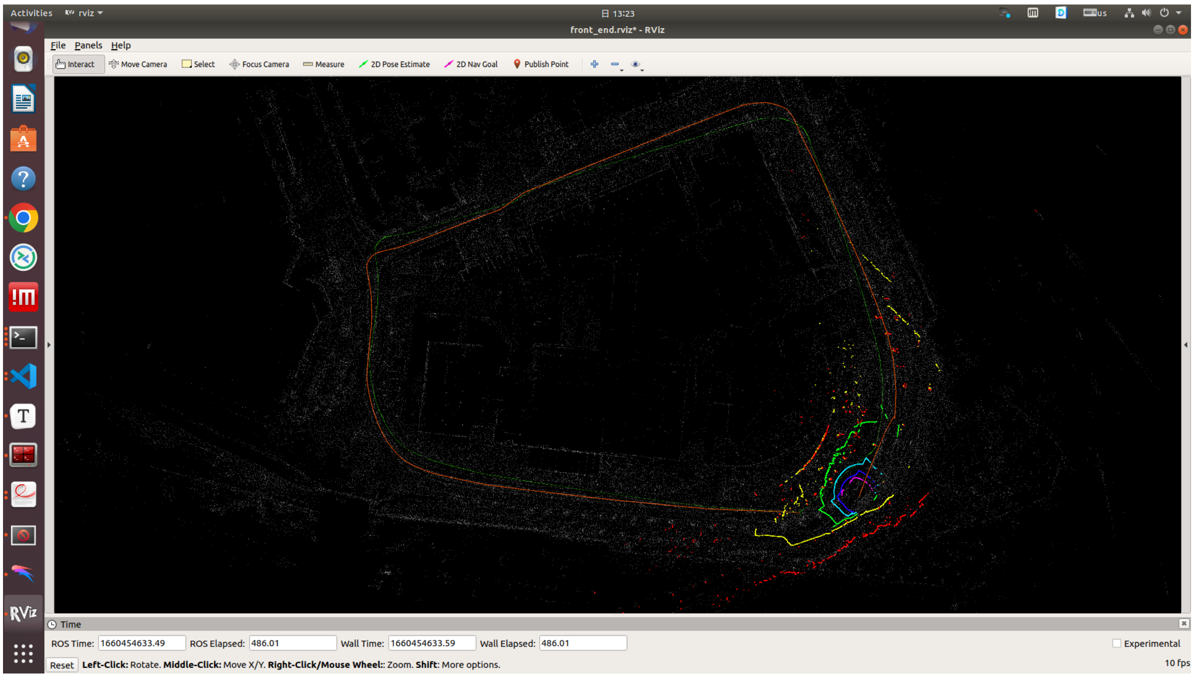

Error mapped onto trajectory
reference
75
50
25
0
−25
−50
−75
25
0
−25
−50
−75
−100
−125
−150
y (m)
−25
0
25
50
75
100
125
150
x (m)
18.321
9.161
0.000

APE指标：

```
    max  18.321411
   mean  6.576407
 median  5.921852
    min  0.000000
   rmse  7.900457
    sse  149114.723115
    std  4.378137
```

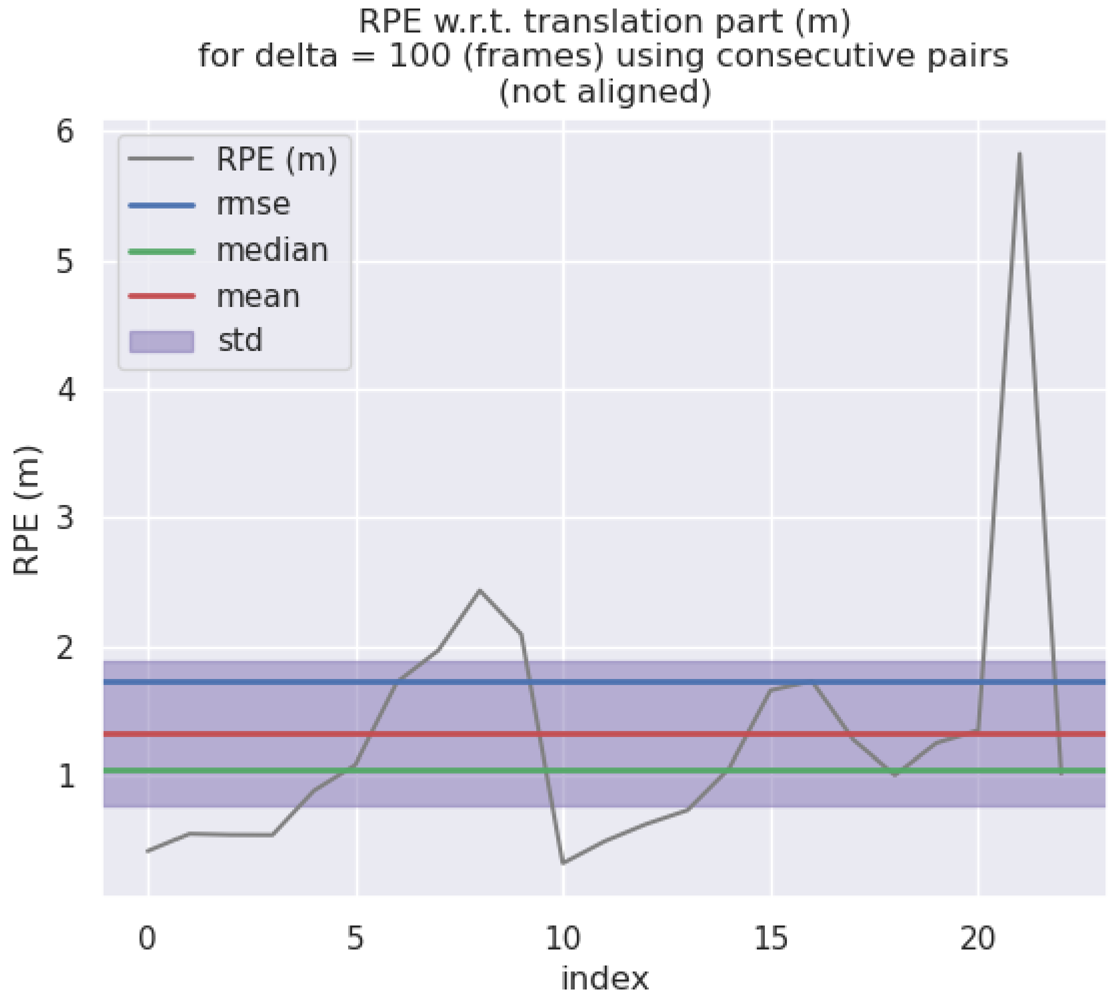
ALOAM-自动求导+自定义参数块

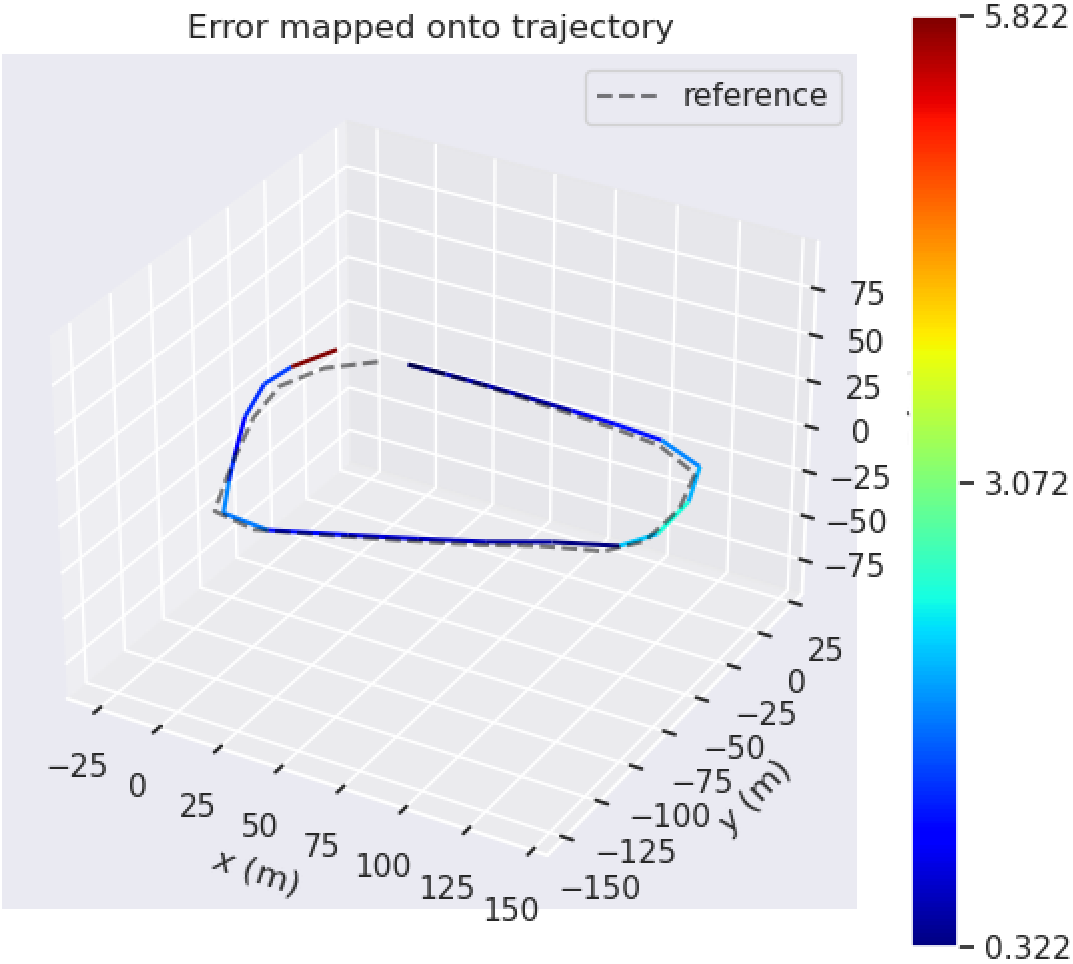
rviz效果



evo评估

RPE

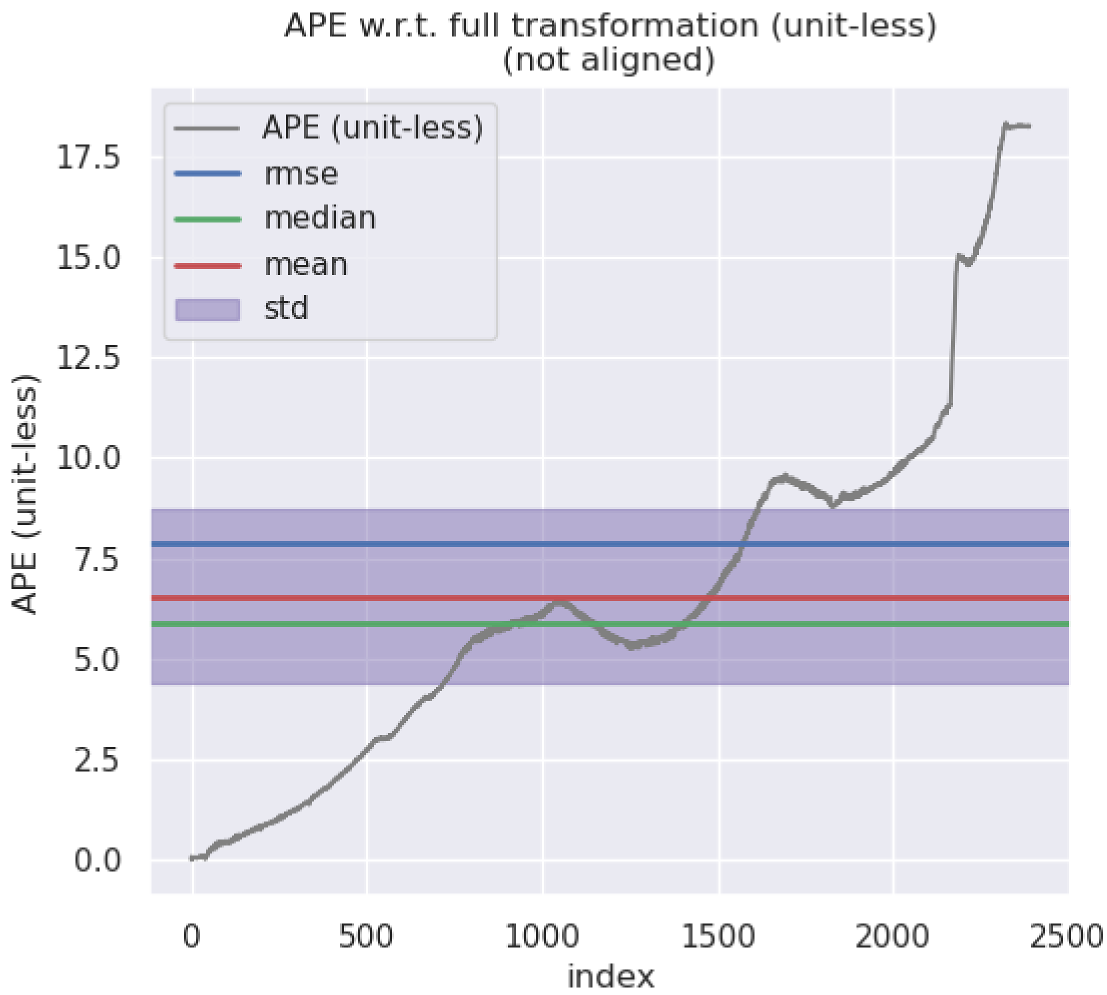

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
6
5
4
3
2
1
RPE (m)
0
5
10
15
20
index

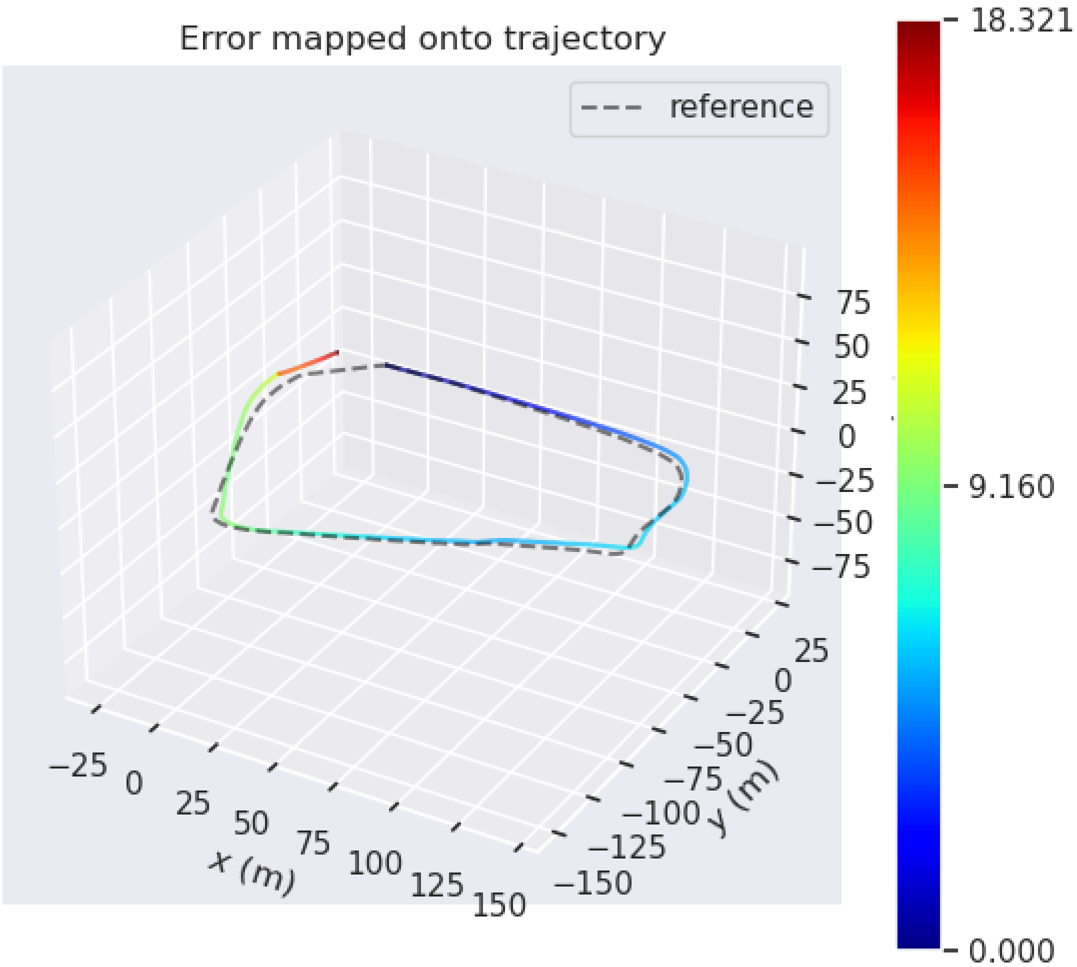

Error mapped onto trajectory
reference
5.822
3.072
0.322
75
50
25
0
−25
−50
−75
25
0
−25
−50
−75
−100
−125
−150
y (m)
−25
0
25
50
75
100
125
150
x (m)

RPE指标：

```
   max  5.821638
  mean  1.329391
median  1.055417
   min  0.322271
  rmse  1.735479
   sse  69.273390
   std  1.115619
```

APE



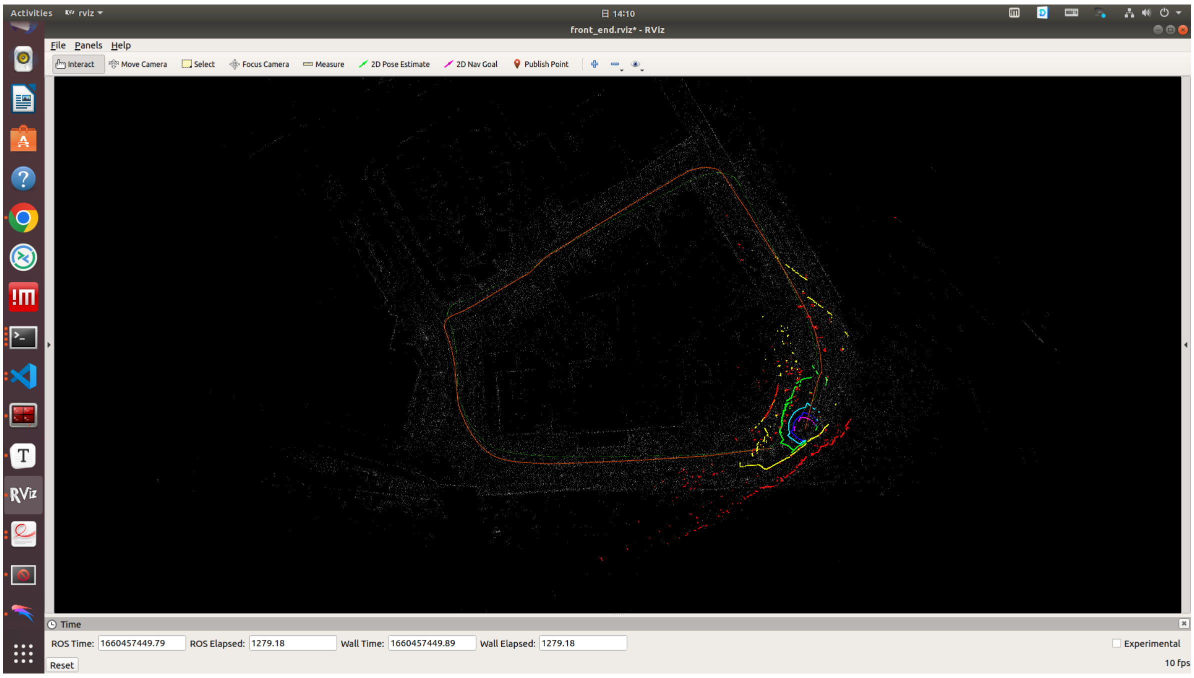


APE指标：

```
   max  18.320923
  mean  6.553765
median  5.908130
   min  0.000000
  rmse  7.872068
   sse  148045.044908
   std  4.360921
```

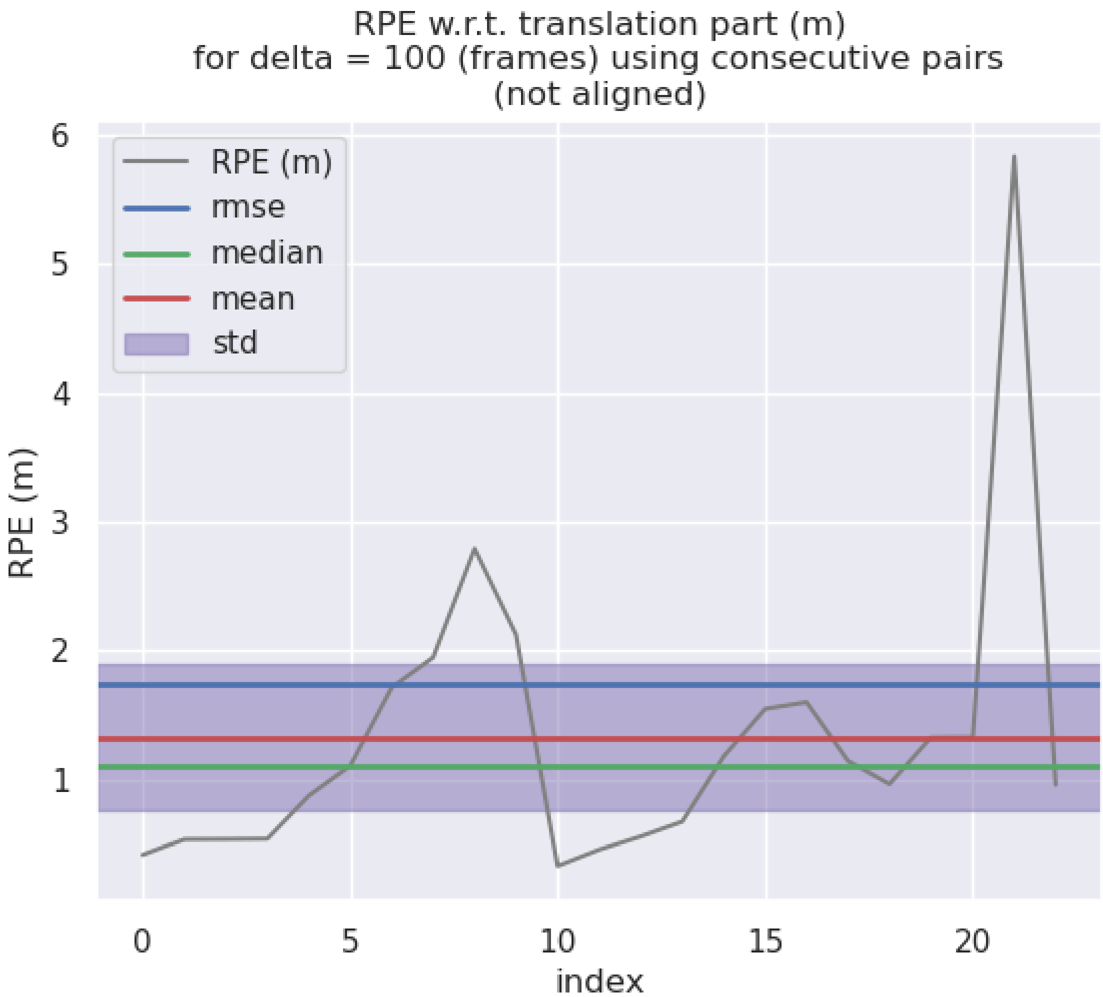
ALOAM-解析式求导+自定义参数块

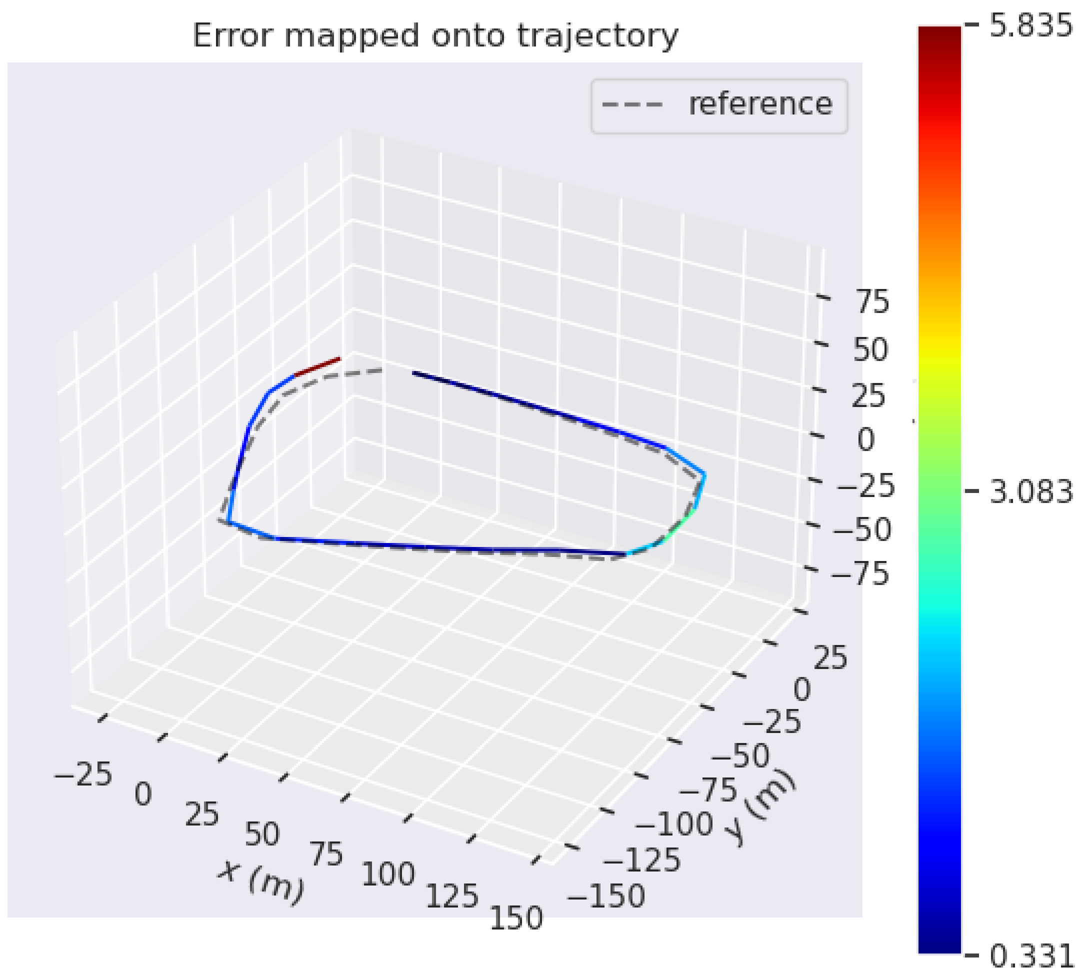
rviz效果



evo评估

RPE

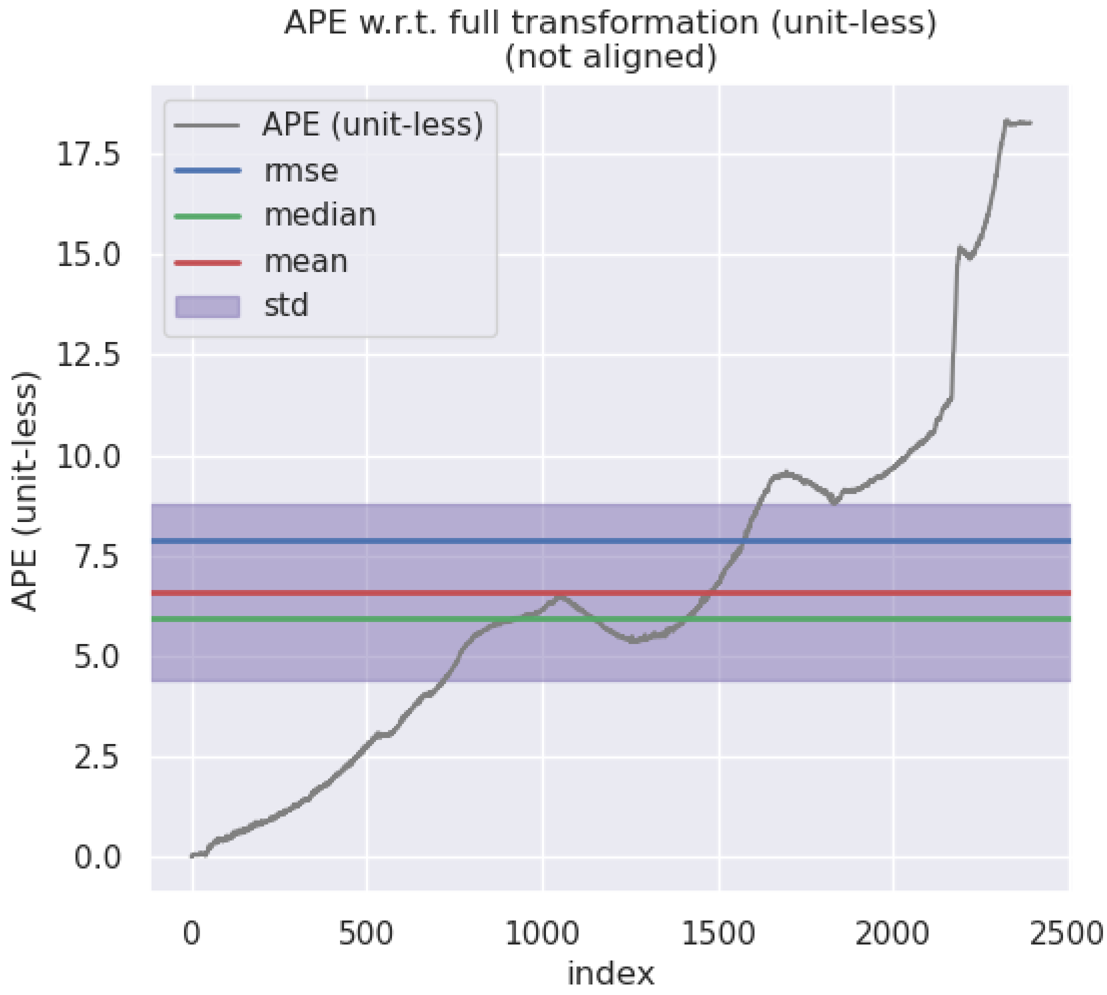

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
RPE (m)
index

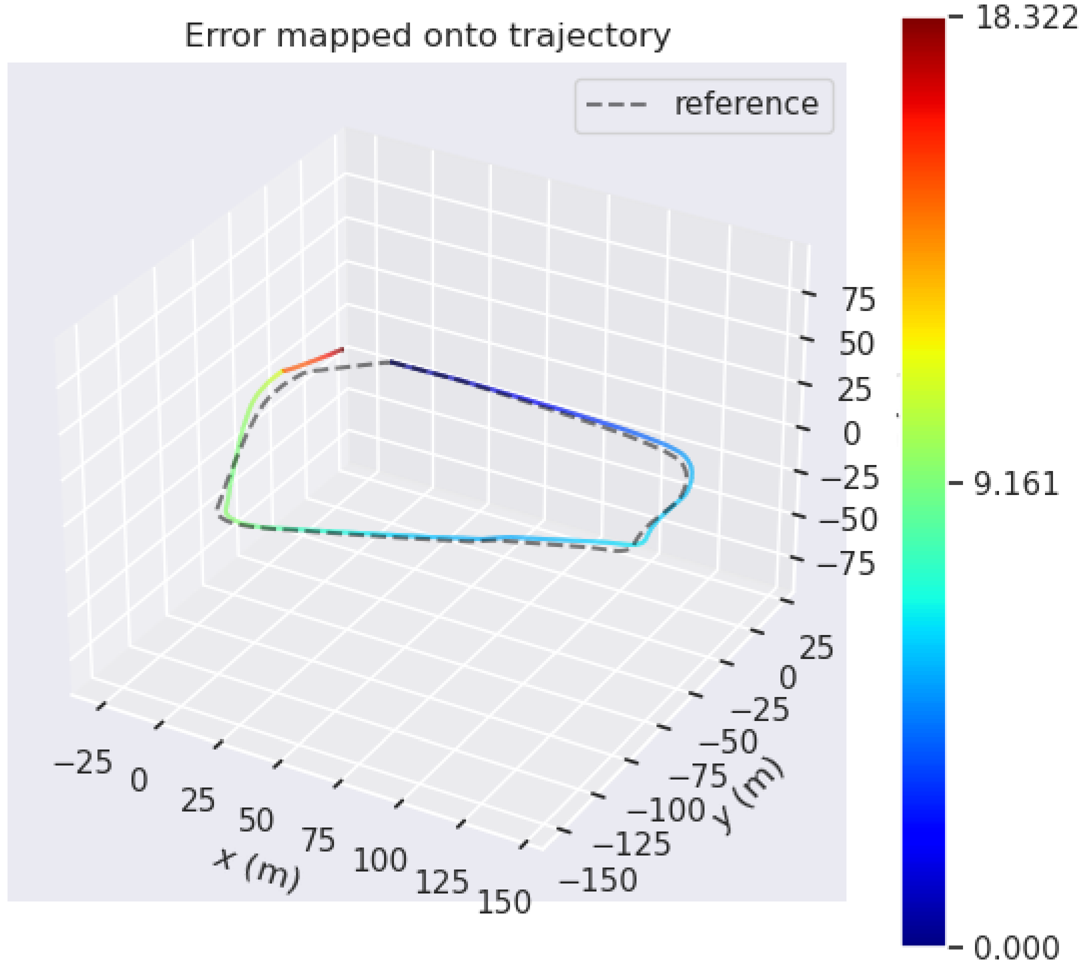


RPE指标：

```
   max  5.834810
  mean  1.325900
median  1.106933
   min  0.330803
  rmse  1.747511
   sse  70.237245
   std  1.138325
```

APE


APE w.r.t. full transformation (unit-less)
(not aligned)
APE (unit-less)
rmse
median
mean
std
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
500
1000
1500
2000
2500
index

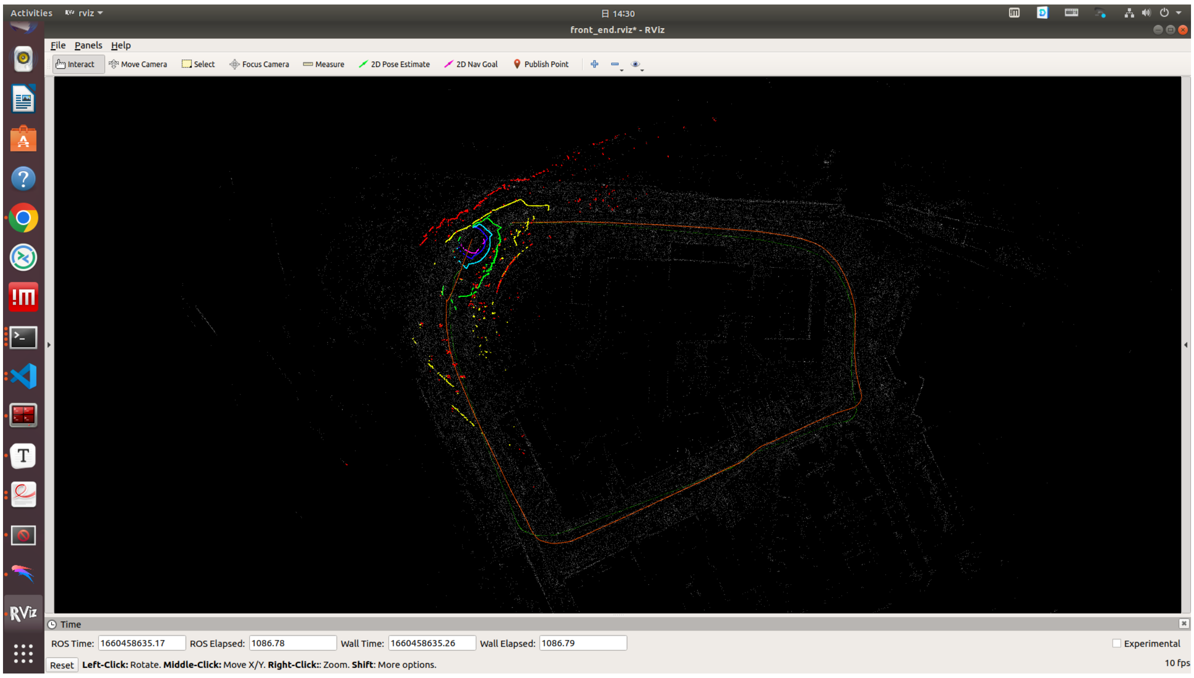

Error mapped onto trajectory
reference
75
50
25
0
−25
−50
−75
−25
0
25
50
75
100
125
150
x (m)
25
0
−25
−50
−75
−100
−125
−150
y (m)
18.322
9.161
0.000

APE指标：

```
   max	18.321558
  mean	6.580434
median	5.925364
   min	0.000000
  rmse	7.900475
   sse	149302.681339
   std	4.372116
```

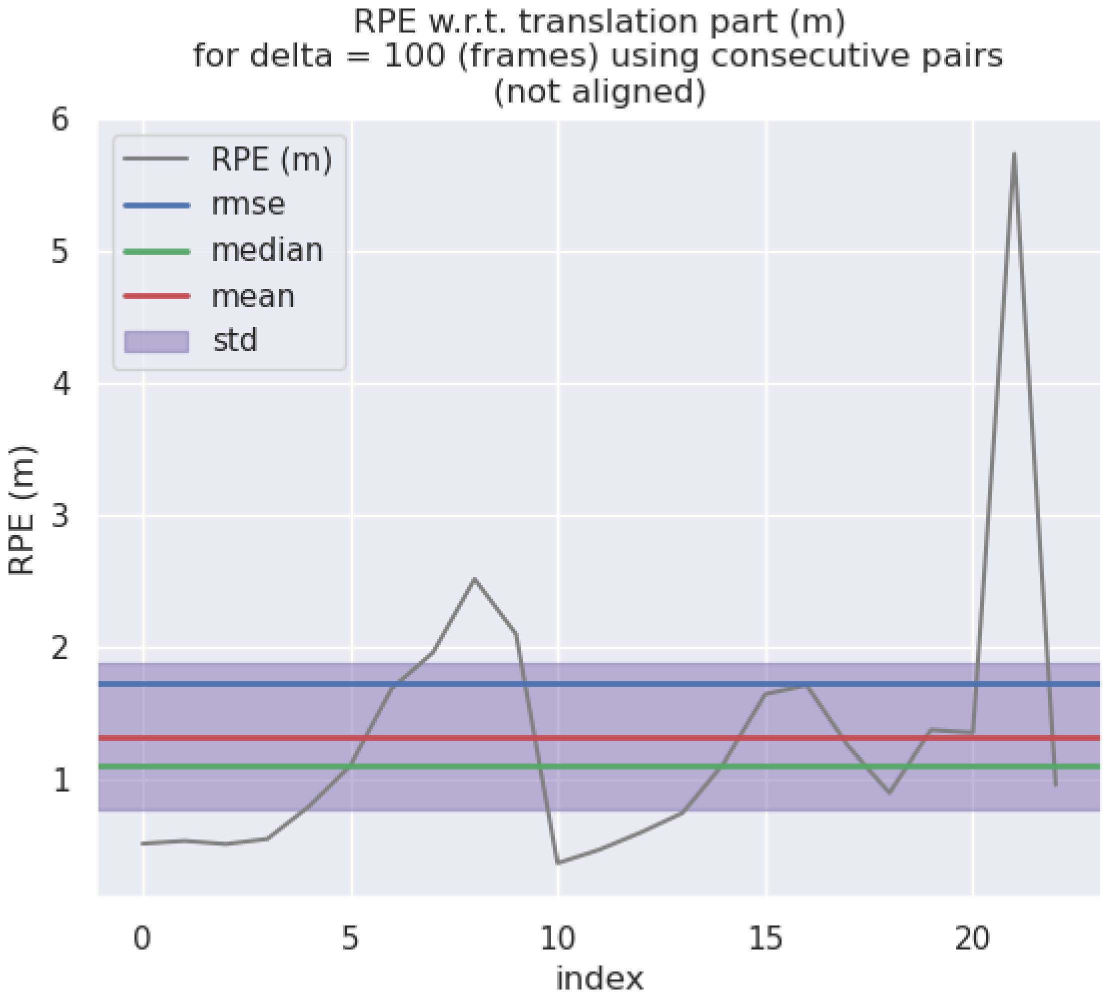
ALOAM-解析式求导+ceres自带参数块

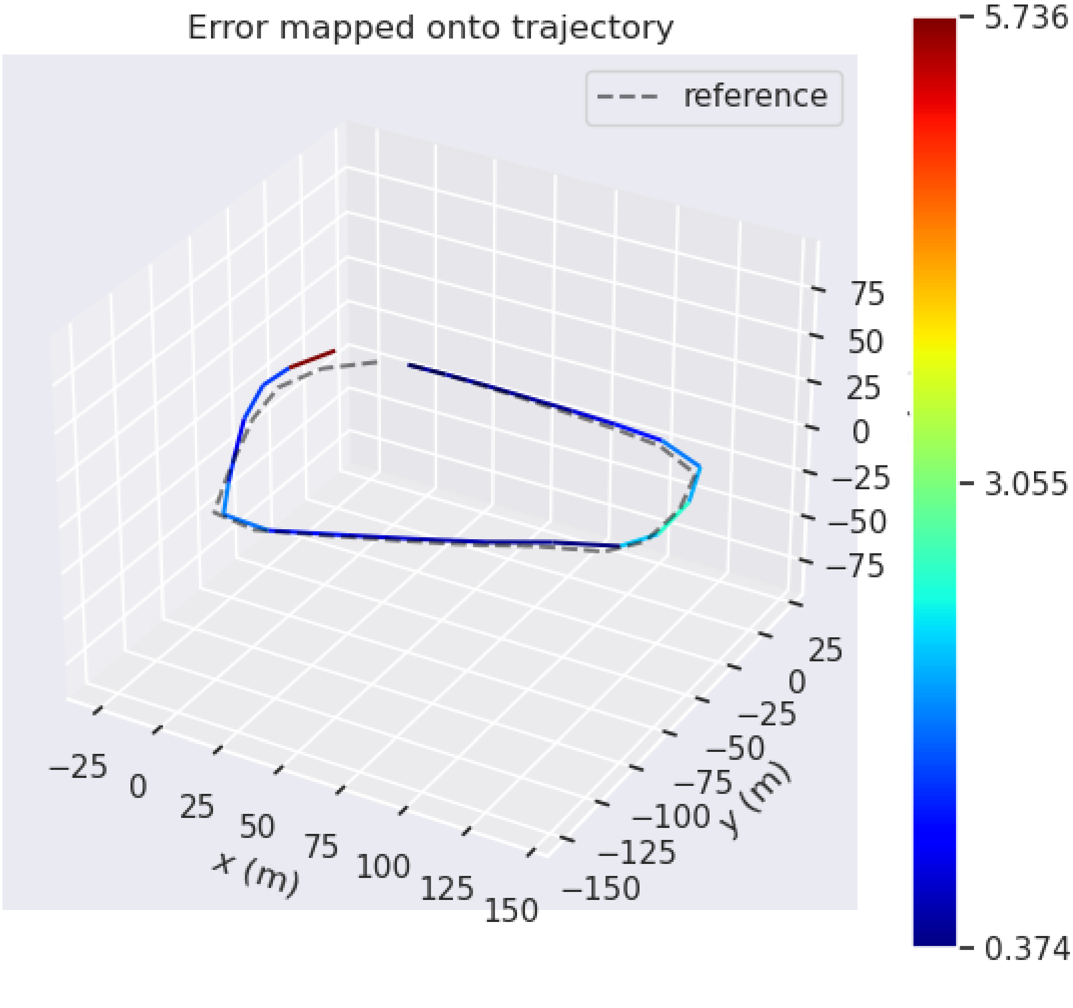
rviz效果



evo评估

RPE

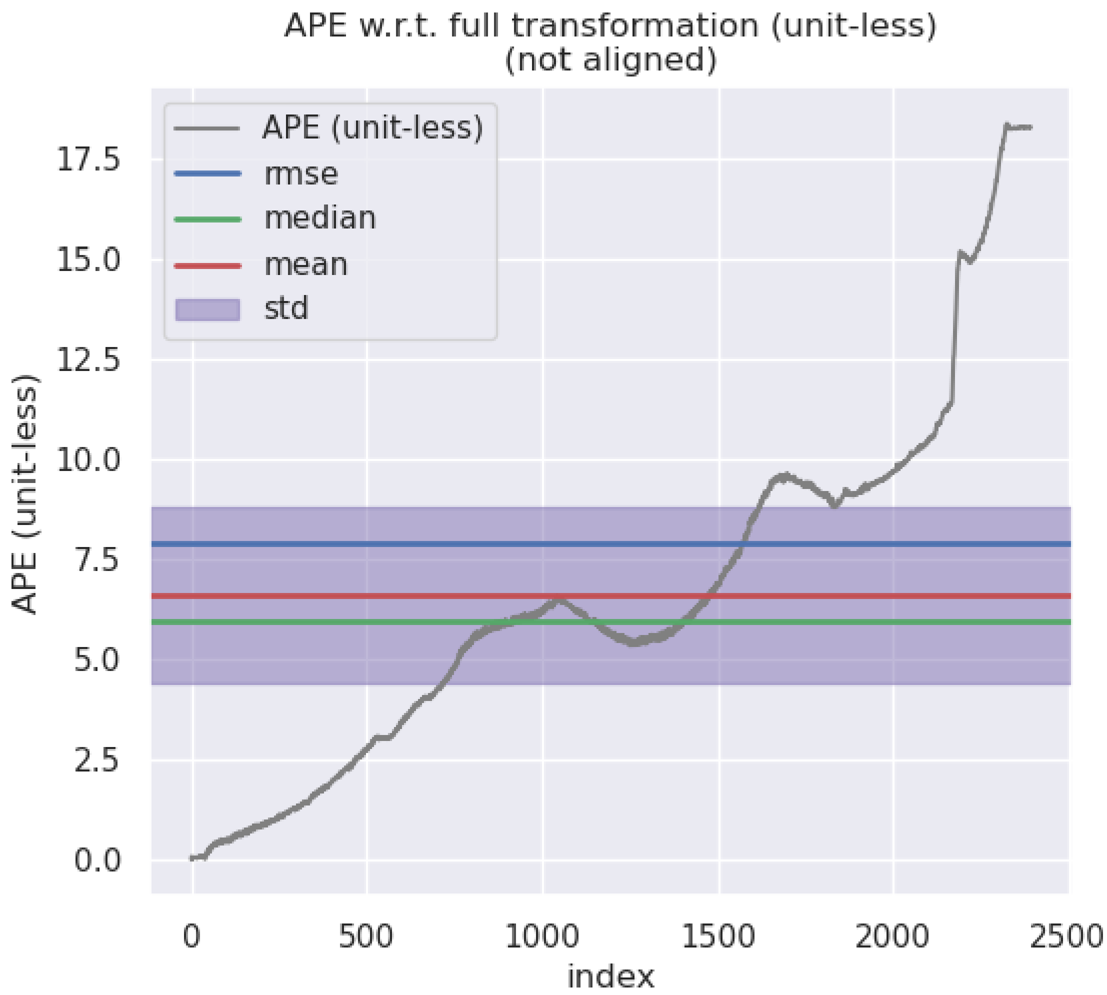

RPE w.r.t. translation part (m)
for delta = 100 (frames) using consecutive pairs
(not aligned)
RPE (m)
rmse
median
mean
std
RPE (m)
index
0
5
10
15
20
1
2
3
4
5
6

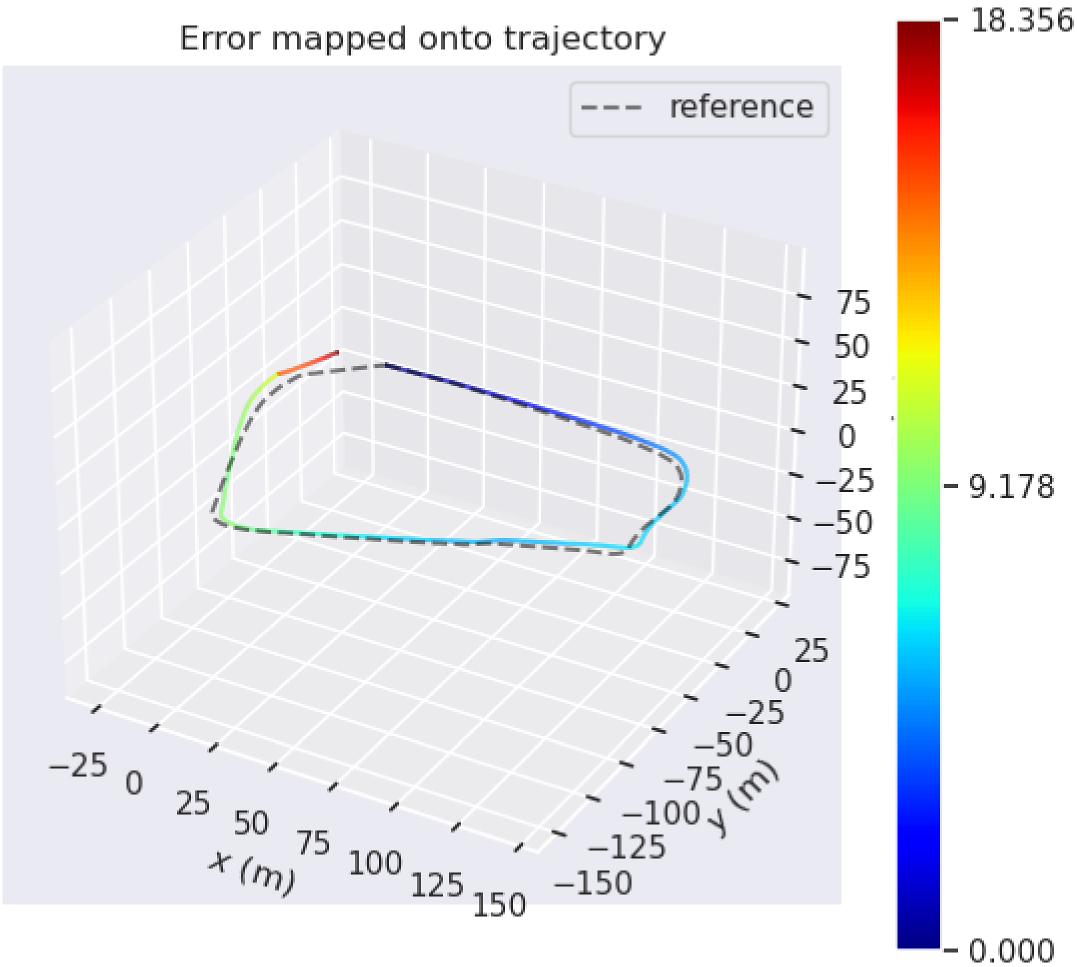

Error mapped onto trajectory
reference
x (m)
y (m)
5.736
3.055
0.374

RPE指标：

```
   max  5.735796
  mean  1.331910
median  1.110271
   min  0.374149
  rmse  1.728747
   sse  68.737045
   std  1.102081
```

APE





APE指标：

```
     max	18.355989
    mean	6.605808
  median	5.948474
     min	0.000000
    rmse	7.916706
     sse	149979.431575
     std	4.363201
```

四.总结和思考

总结

kitti数据集下A-LOAM算法不同情况下的指标

A-LOAM算法	RPE(rmse)	RPE(median)	APE(rmse)	APE(median)	单核CPU占用率（map/odo/scan)	建图总时间（最后一帧）t/ms
analytic_grade-ceres_defined	1.384630	1.036104	20.423035	14.906351	90%/25%/30%	198.495697
analytic_grade-user_defined	1.450670	1.026564	20.862871	14.574086	90%/20%/30%	169.763989
autograde-ceres_defined	1.669851	1.332409	20.849407	15.351334	100%/30%/30%	224.249608
autograde-user_defined	1.537500	1.315126	20.628949	15.274735	90%/25%/30%	210.316974

自录数据集下A-LOAM算法不同情况下的指标

A-LOAM算法	RPE(rmse)	RPE(median)	APE(rmse)	APE(median)	单核CPU占用率约（map/odo/scan)	建图最后一帧总时间约（t/ms)
analytic_grade-ceres_defined	1.728747	1.110271	7.916706	5.948474	25%/2%/3%	44.107110
analytic_grade-user_defined	1.747511	1.106933	7.872068	5.908130	25%/<2%/3%	48.435103
autograde-ceres_defined	1.744557	1.058711	7.900457	5.921852	40%/2%/3%	51.040462
autograde-user_defined	1.735479	1.055417	7.872068	5.908130	35%/<2%/5%	45.912901

注:

1. 单核CPU占用率中有三个线程，map/odo/scan分别对应是aloam_mapping/aloam_laser_odo/scan_registration
2. 建图最后一帧总时间是原工程代码中自带的，仅可作为参考，评估算法耗时更合理的是所有建图耗时累加进行比较。
3. analytic_grade为解析式求导，autograde为自动求导；ceres_defined为ceres自带参数块，user_defined为自己定义的参数块。
4. EVO指标中挑选了个人认为比较主要的两个指标，分别为rmse和median。
5. 单核CPU占用率是通过top命令查看三个线程的占用率，粗略一看得到的。

第二章的指标：

kitti数据集下各前端匹配算法的指标

算法	RPE(rmse)	RPE(median)	APE(rmse)	APE(median)	单核CPU占用率（约)	匹配代码总时间（t/s)
PCL-ICP	100.470429	30.754721	545.844611	407.135850	>100%	404.749952
PCL-NDT	0.938290	0.716354	27.903072	13.975223	>100%	342.512242
ICP-SVD	0.954655	0.852509	15.359713	14.084914	>100%	358.358025
NDT-CPU	0.886698	0.720926	26.474762	19.032020	>100%	342.512242
SICP	71.603350	35.070945	414.013381	381.383080	>100%	3620.398145

自录数据集下各前端匹配算法的指标

算法	RPE(rmse)	RPE(median)	APE(rmse)	APE(median)	单核CPU占用率（约）	匹配代码总时间（t/s）
PCL-ICP	3.782357	2.255728	19.595615	4.767711	35%	≈47
PCL-NDT	1.293570	1.130785	5.101803	4.644881	40%	≈55
ICP-SVD	1.695362	1.092998	9.578515	6.830639	30%	≈55
NDT-CPU	1.283679	1.168267	3.685613	4.060760	30%	≈61
SICP	12.381528	1.866251	45.598697	38.405363	85%	≈350

从指标和截图得出这次实验的结论：

1. 对比第二章在kitti数据集中能够有效果（**指的是全局不飘**）的前端算法，分别为PCL-NDT、ICP-SVD、NDT-CPU，a-loam的evo精度都比较好，原因是aloam加了后端优化，里程计精度更高。
2. 从evo指标来看，ALOAM的四种情况下的指标基本一致；从建图效果来看也是基本一致。
3. 从单核CPU占用率和建图总时间可以粗略得到效率为：analytic_grade-user_defined>analytic_grade-ceres_defined>autograde-user_defined>autograde-ceres_defined
4. 从两个数据集中的使用来看，自采数据集的条件更加好（小车运行速度慢、雷达点云数量少、转弯少、场景规模也小），效率和精度都更加好。

思考和疑问

1. ALOAM将前后端多线程化的思路，能够大大提高单核计算效率，提高实时性，这也是后面SLAM的一个通用思路。
2. 为什么在docker中编译第三章的作业会出现缺少 fmt 库的错误，自己配置的本地环境可以编译。