第一问：

线、面残差雅克比推导过程如下图所示：

作业答案做法

线特征残差：

$$d_\varepsilon = \frac{|(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)|}{|p_a - p_b|}, \quad \tilde{p}_i = Rp_i + t.$$

设 $A = \dfrac{(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)}{|p_a - p_b|}$

$$\frac{\partial d_\varepsilon}{\partial T^T} = \frac{\partial d_\varepsilon}{\partial A^T}^{1\times 3} \frac{\partial A}{\partial \tilde{p}_i^T}^{3\times 3} \frac{\partial \tilde{p}_i}{\partial T^T}^{3\times 6} \qquad T = \begin{bmatrix} t \\ \delta\theta \end{bmatrix}$$

$$\frac{\partial d_\varepsilon}{\partial A^T} = \frac{\partial \sqrt{A^T A}}{\partial A^T} = \frac{1}{2}\frac{1}{\sqrt{A^T A}} \cdot 2A^T = \frac{A^T}{d_\varepsilon} = \frac{\left((\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)\right)^T}{|(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)|}$$

$$\frac{\partial A}{\partial \tilde{p}_i^T} = \frac{\partial \dfrac{(\tilde{p}_i - p_b) \times (\tilde{p}_i - p_a)}{|p_a - p_b|}}{\partial \tilde{p}_i^T} = \frac{1}{|p_a - p_b|} \cdot \left(\frac{\partial (\tilde{p}_i - p_b)^\wedge}{\partial \tilde{p}_i^T} (\tilde{p}_i - p_a) + (\tilde{p}_i - p_b)^\wedge \frac{\partial (\tilde{p}_i - p_a)}{\partial \tilde{p}_i^T} \right)$$

$$= \frac{1}{|p_a - p_b|} \left(-(\tilde{p}_i - p_a)^\wedge + (\tilde{p}_i - p_b)^\wedge \right) = \frac{(p_a - p_b)^\wedge}{|p_a - p_b|}.$$

对旋转求导：$\dfrac{\partial d_\varepsilon}{\partial \delta\theta^T} = \dfrac{\partial d_\varepsilon}{\partial A^T}^{1\times 3} \dfrac{\partial A}{\partial \tilde{p}_i^T}^{3\times 3} \cdot \dfrac{\partial \tilde{p}_i}{\partial \delta\theta}^{3\times 3}$

对平移求导：$\dfrac{\partial d_\varepsilon}{\partial t} = \dfrac{\partial d_\varepsilon}{\partial A^T}^{1\times 3} \cdot \dfrac{\partial A}{\partial \tilde{p}_i^T}^{3\times 3} \cdot \underbrace{\dfrac{\partial \tilde{p}_i}{\partial t}^{3\times 3}}_{= I}$

$$\frac{\partial \tilde{p}_i}{\partial \delta\theta} = \frac{\partial (\exp(\delta\theta) R p_i + t)}{\partial \delta\theta} = \frac{(I + \delta\theta^\wedge) R p_i}{\partial \delta\theta} = -(Rp_i)^\wedge$$

面特征残差：

$$d_H = \left| (\tilde{p}_i - p_j)^T \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|} \right| = |a|. \qquad a = (\tilde{p}_i - p_j)^T \frac{(p_l - p_j) \times (p_m - p_j)}{|(p_l - p_j) \times (p_m - p_j)|}$$

$$\frac{\partial d_H}{\partial T^T} = \frac{\partial d_H}{\partial a^T}^{1\times 1} \cdot \frac{\partial a}{\partial \tilde{p}_i^T}^{1\times 3} \cdot \frac{\partial \tilde{p}_i}{\partial T^T}^{3\times 6}$$

$$\frac{\partial d_H}{\partial a^T} = \frac{|a|}{a} = \pm 1$$

$$\frac{\partial a}{\partial \tilde{p}_i^T} = \frac{\left((p_l - p_j) \times (p_m - p_j)\right)^T}{|(p_l - p_j) \times (p_m - p_j)|}$$

$$\frac{\partial \tilde{p}_i}{\partial \delta\theta} = -(Rp_i)^\wedge$$

对旋转求导：$\dfrac{\partial d_H}{\partial \delta\theta^T} = \dfrac{\partial d_H}{\partial A^T}^{1\times 3} \cdot \dfrac{\partial A}{\partial \tilde{p}_i^T}^{3\times 3} \cdot \dfrac{\partial \tilde{p}_i}{\partial \delta\theta}^{3\times 3}$

对平移求导：$\dfrac{\partial d_H}{\partial t^T} = \dfrac{\partial d_H}{\partial A^T}^{1\times 3} \cdot \dfrac{\partial A}{\partial \tilde{p}_i^T}^{3\times 3} \cdot \dfrac{\partial \tilde{p}_i}{\partial t}^{3\times 3}$

第二问：ceres 解析求导

本方案采取 aloam.launch 中涉及的代码。

在 03-lidar-odometry-advanced/src/lidar_localization/include/lidar_localization/models/loam 目录下新建 aloam_analytic_factor.hpp 文件，用于解析求导。

文件主要包括线特征的 CostFunction 类和面特征的 CostFunction 类，分别为 EdgeAnalyticCostFunction 和 SizedCostFunction，这两类继承 ceres::SizedCostFunction，而 ceres::SizedCostFunction 继承自 CostFunction。以面特征 CostFunction 为例，在构建 ceres problem 时，通过以下代码添加进 problem 中。

```
ceres::CostFunction *cost_function = new PlaneAnalyticCostFunction(curr_point, last_point_a, last_point_b, last_point_c, s);
problem.AddResidualBlock(cost_function, loss_function, para_q, para_t);
```

线特征的 costfunction 如下所示。

```
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_b).cross(lp - last_point_a);
        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::Matrix3d  skew_de  =   skew(de);

```

```
            // 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_de * 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_de / (de.norm()*nu.norm());
        }
    }
    return  true;
}
};
```

面特征的 costfunction 如下所示：

```
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_l - last_point_j).cross(last_point_m - last_point_j);
        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;
    }

};
```

以上 costfunction 用到将向量转换成反对称矩阵的 skew 函数，skew 函数定义如下：

```
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;
}
```

修改 aloam_scan_scan_registration_node.cpp 两处添加线、面 costfunction 的代码，注释部分为源代码。

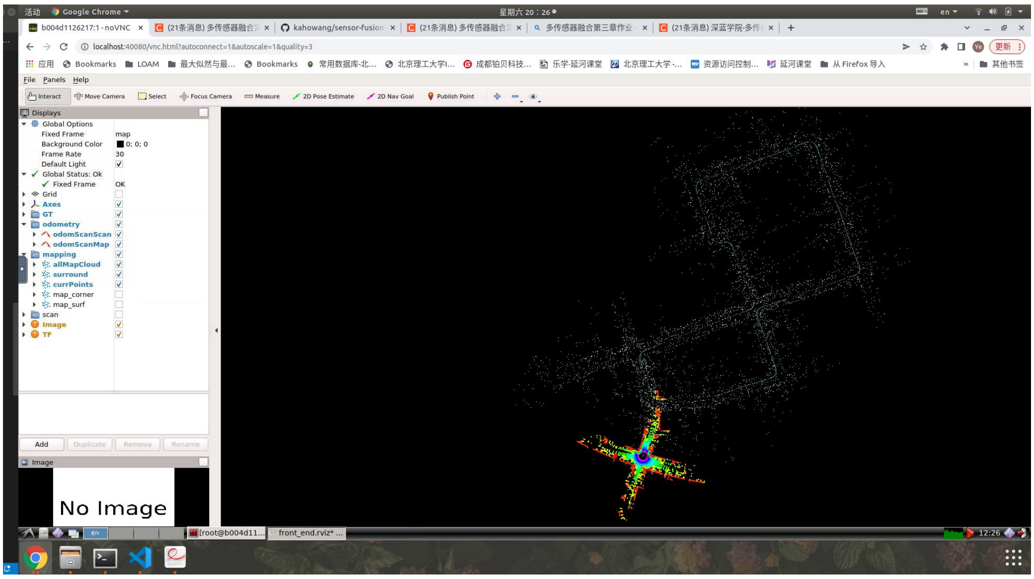
添加线特征 costfunction

```
// ceres::CostFunction *cost_function = LidarEdgeFactor::Create(curr_point, last_point_a, last_point_b, s);
ceres::CostFunction *cost_function = new EdgeAnalyticCostFunction(curr_point, last_point_a, last_point_b, s);
problem.AddResidualBlock(cost_function, loss_function, para_q, para_t);
```

添加面特征 costfunction

```
1. // ceres::CostFunction *cost_function = LidarPlaneFactor::Create(curr_point, last_point_a, last_point_b, last_point_c, s);
2. ceres::CostFunction *cost_function = new PlaneAnalyticCostFunction(curr_point, last_point_a, last_point_b, last_point_c, s);
3. problem.AddResidualBlock(cost_function, loss_function, para_q, para_t);
```

至此，第二问解析求导代码修改完成。运行截图如下：

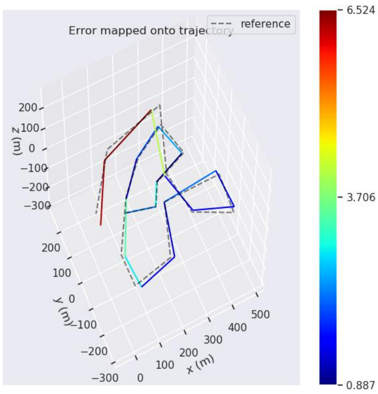


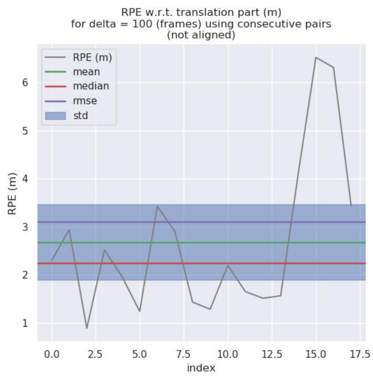
第三问：evo 精度评估

精度评估结果如下：

evo_rpe kitti ground_truth.txt laser_odom.txt -r trans_part -d 100 -p --plot_mode xyz

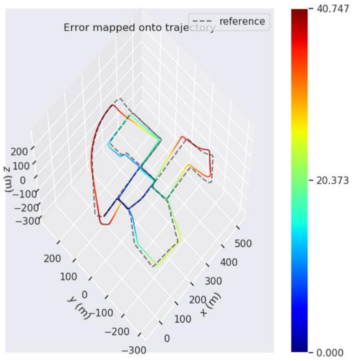
(疑问：为啥 evo_rpe -d 100 画出的图这么奇怪，如下面第一张图)

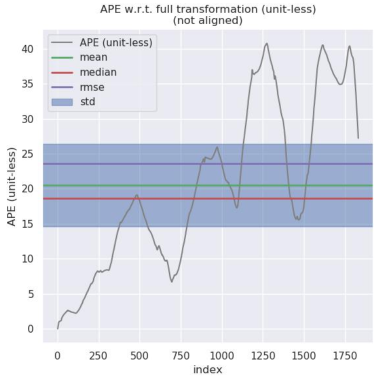





```
   max      6.524446
  mean      2.676345
median      2.243878
   min      0.887179
  rmse      3.105633
   sse      173.609171
   std      1.575478
```

evo_ape kitti ground_truth.txt laser_odom.txt -r full -p --plot_mode xyz





max	40.746721
mean	20.511087
median	18.603328
min	0.000002
rmse	23.641750
sse	1024522.965405
std	11.757024