第三章作业代码及笔记

1.Ceres 入门笔记

初级入门可参考高翔slam14讲，ceres 曲线拟合。

解析求导需要对雅克比进行赋值，因为ceres默认的更新形式是加减方式，而so3的四元数更新方式不同，所以需要自定义旋转参数块（顾名思义：自定义delta_q更新运算法则）

自定义旋转参数块：LocalParameterization参数化

参考博客：

Ceres详解（一） Problem类

Ceres（二）LocalParameterization参数化

优化库——ceres（二）深入探索ceres::Problem

对于四元数或者旋转矩阵这种使用过参数化表示旋转的方式，它们是不支持广义的加法（因为使用普通的加法就会打破其 constraint，比如旋转矩阵加旋转矩阵得到的就不再是旋转矩阵），所以我们在使用ceres对其进行迭代更新的时候就需要自定义其更新方式了，具体的做法是实现一个参数本地化的子类，需要继承于LocalParameterization，LocalParameterization是纯虚类，所以我们继承的时候要把所有的纯虚函数都实现一遍才能使用该类生成对象。

除了不支持广义加法要自定义参数本地化的子类外，如果你要对优化变量做一些限制也可以如法炮制，比如ceres中slam2d example中对角度范围进行了限制.

LocalParameterization 自定义实现

这里以四元数为例子，解释如何实现参数本地化，需要注意的是，QuaternionParameterization中表示四元数中四个量在内存中的存储顺序是[w, x, y, z]，而Eigen内部四元数在内存中的存储顺序是[x, y, z, w]，但是其构造顺序是[w, x, y, z]（不要被这个假象给迷惑），所以就要使用另一种参数本地化类，即EigenQuaternionParameterization，下面就以QuaternionParameterization为例子说明，如下：

```
class CERES_EXPORT QuaternionParameterization : public LocalParameterization {
 public:
  virtual ~QuaternionParameterization() {}
  virtual bool Plus(const double* x,
                    const double* delta,
                    double* x_plus_delta) const;
  virtual bool ComputeJacobian(const double* x,
                               double* jacobian) const;
  virtual int GlobalSize() const { return 4; }
  virtual int LocalSize() const { return 3; }
};
```

1.GlobalSize()

表示参数 x xx 的自由度（可能有冗余），比如四元数的自由度是4，旋转矩阵so3的自由度是3

2.LocalSize()

表示 Δx 所在的正切空间（tangent space）的自由度，那么这个自由度是多少呢？下面进行解释，

正切空间是流形（manifold）中概念，对流形感兴趣的可以参考[2]，参考论文[3]，我们可以这么理解manifold：

> A manifold is a mathematical space that is not necessarily Euclidean on a global scale, but can be seen as Euclidean on a local scale

上面的意思是说，SO3 空间是属于非欧式空间的，但是没关系，我们只要保证旋转的局部是欧式空间，就可以使用流形进行优化了. 比如用四元数表示的3D旋转是属于非欧式空间的，那么我们取四元数的向量部分作为优化过程中的微小增量（因为是小量，所以不存在奇异性）. 为什么使用四元数的向量部分？这部分可以参考[4]或者之前写的关于四元数的博客. 这样一来，向量部分就位于欧式空间了，也就得到了正切空间自由度是3.

3.Plus()

自定义优化变量更新函数

4.ComputeJacobian()

个人理解，是 定义待更新变量 和 待更新变量的delta 的雅克比

4、ComputeJacobian()

参考[2]中的公式24，使用链式法则我们知道 $\tilde{\mathbf{J}}_{ij}$ 由两部分构成，第一部分 $\frac{\partial \mathbf{e}_{ij}(\check{\mathbf{x}})}{\partial \check{\mathbf{x}}_i}$ 是对原始过参数化的优化变量（比如，四元数）的导数，这个很容易求得，直接借助ceres的 AutoDiffCostFunction() 计算即可，或者自己计算雅可比矩阵，实现一个costfunction，关键是第二部分是如何求得的呢?

$$\tilde{\mathbf{J}}_{ij} = \frac{\partial \mathbf{e}_{ij}(\check{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \Delta\tilde{\mathbf{x}}}\bigg|_{\Delta\tilde{\mathbf{x}}=0} = \frac{\partial \mathbf{e}_{ij}(\check{\mathbf{x}})}{\partial \check{\mathbf{x}}_i} \cdot \frac{\check{\mathbf{x}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\bigg|_{\Delta\tilde{\mathbf{x}}=0}$$

现在求解 $\frac{\partial \check{\mathbf{x}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_i}\Big|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=0}$，以四元数为例：

$$\frac{\partial \check{\mathbf{q}}_i \boxplus \boldsymbol{\Delta}\tilde{\mathbf{q}}_i}{\partial \boldsymbol{\Delta}\tilde{\mathbf{q}}_i} = \frac{\partial \check{\mathbf{q}}_i \otimes e^{\boldsymbol{\Delta}\tilde{\mathbf{q}}_i}}{\partial \boldsymbol{\Delta}\tilde{\mathbf{q}}_i} \approx \frac{\partial \check{\mathbf{q}}_i \otimes \begin{bmatrix} 1 \\ \boldsymbol{\Delta}\tilde{\mathbf{q}}_i \end{bmatrix}}{\partial \boldsymbol{\Delta}\tilde{\mathbf{q}}_i} = \frac{\partial [\check{\mathbf{q}}_i]_L \begin{bmatrix} 1 \\ \boldsymbol{\Delta}\tilde{\mathbf{q}}_i \end{bmatrix}}{\partial \boldsymbol{\Delta}\tilde{\mathbf{q}}_i}$$

注意上面符号的变化，先从 ⊞ 变成四元数乘法 ⊗，最后变成普通的乘法，进一步得到，

$$\begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix} \frac{\partial \begin{bmatrix} 1 \\ \boldsymbol{\Delta}\tilde{\mathbf{q}}_i \end{bmatrix}}{\partial \boldsymbol{\Delta}\tilde{\mathbf{q}}_i} = \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix}$$

最后得到雅可比矩阵是4*3维的，代码里使用一个size为12的数组存储.

5.Problem::AddParameterBlock()

自定义参数块后，需要通过 Problem::AddParameterBlock 自定义参数快加载

```
void Problem::AddParameterBlock(double *values, int size, LocalParameterization
*local_parameterization)
```

CostFunction 定义

重点参考博客：

Ceres详解（一） Problem类

Ceres详解（二） CostFunction

2.基于线面特征解析求导的实现

原工程移植了 aloam的自动求导，这里实现解析求导，公式推导如上“基于线面特征位姿优化/公式推导”所示

FILE: lidar_localization/include/lidar_localization/models/loam/aloam_analytic_factor.hpp

定义CostFunction

解析求导-点线匹配 CostFunction

```
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);
            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;
}
};
```

解析求导-点面匹配 CostFunction

```
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;
    }
};
```

Eigen 反对称矩阵函数定义

```
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;
}
```

定义旋转参数块

使用ceres自带的四元数更新参数块

通过观察自动求导的源码，可以发现，在自动求导中，使用了ceres的一个自定义四元数参数块

FILE: lidar_localization/src/models/loam/aloam_registration.cpp

使用EigenQuaternionParameterization 参数块

```
config_.q_parameterization_ptr = new ceres::EigenQuaternionParameterization();
```

加载参数块

```
problem_.AddParameterBlock(param_.q, 4, config_.q_parameterization_ptr);       // 加载自定义旋转参数块
```

自己定义四元数更新参数块

FILE: lidar_localization/include/lidar_localization/models/loam/aloam_analytic_factor.hpp

参考博客：

优化库——ceres（二）深入探索ceres::Problem

[ceres-solver] From QuaternionParameterization to LocalParameterization

这里需要注意的是自定义时，输入的 q :四元数 4维 ， 更新以 so3的形式更新 3 维 ， 所以GlobalSize：4 LocalSize：3 具体体现在 ComputeJacobian() 函数中

```

```

```
// 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
};
```

3.调用

FILE: lidar_localization/src/models/loam/aloam_registration.cpp

解析求导

为了方便调用，沿用自动求导的调用方式

定义宏

```
#define autograde          // 使用自动求导 or 解析求导
```

点线匹配

```
bool CeresALOAMRegistration::AddEdgeFactor(
  const Eigen::Vector3d &source,
  const Eigen::Vector3d &target_x, const Eigen::Vector3d &target_y,
  const double &ratio
){

  /*自动求导*/
 #ifdef  autograde
  ceres::CostFunction *factor_edge = LidarEdgeFactor::Create(            // 创建误差项
    source,
    target_x, target_y,
    ratio
  );

  problem_.AddResidualBlock(
    factor_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
  );

  #else
  /*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
  );

#endif
  return true;
}
```

点面匹配

```
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*/
  #ifdef  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
    );

#else
  /*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
    );

#endif

    return true;
}
```

自定义旋转参数块

定义宏

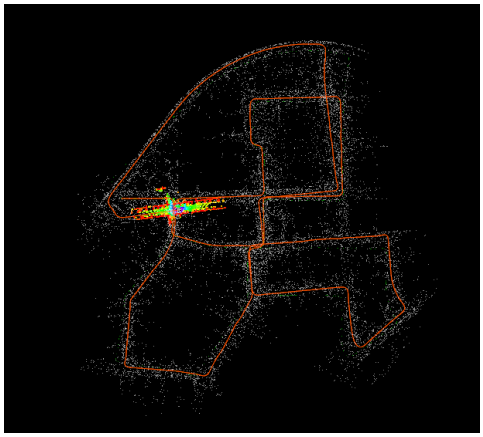
```
#define  maunual_block_loder              // 使用ceres 自带EigenQuaternionParameterization  参数块   or 自定义 PoseSO3Parameterization 旋转参数块
```

```
#ifdef  maunual_block_loder
    config_.q_parameterization_ptr =  new  PoseSO3Parameterization() ;              // 自定义旋转参数块
#else
    config_.q_parameterization_ptr = new ceres::EigenQuaternionParameterization();       //  SE3 转换矩阵/位姿参数化，
#endif
```

4.evo 评价

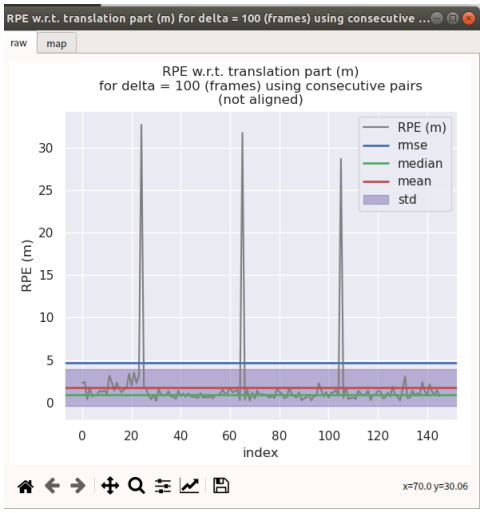
自动求导 + Ceres 自带四元数参数块

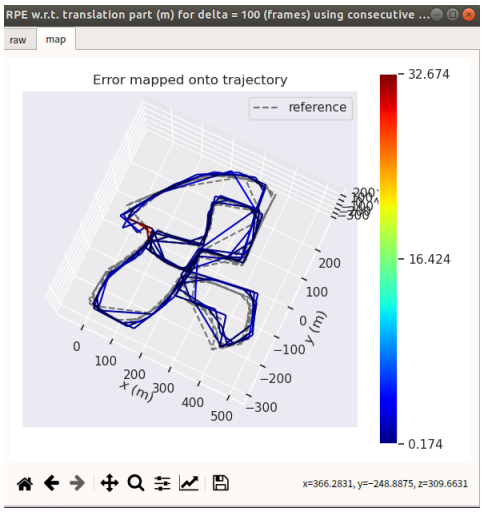
```
whole mapping time 179.438701 ms +++++
```



evo_rpe

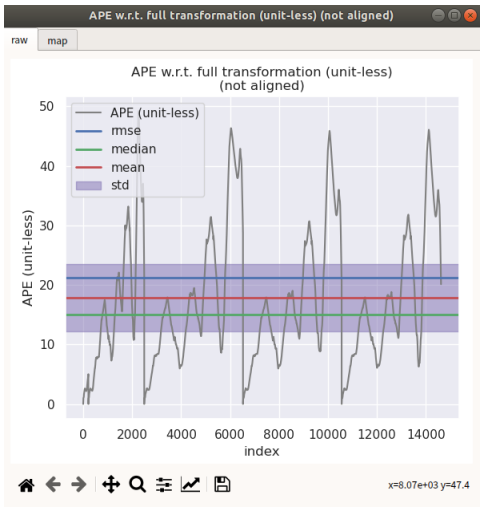
```
evo_rpe kitti ground_truth.txt laser_odom.txt -r trans_part --delta 100 --plot --plot_mode xyz
       max 32.674307
      mean 1.727705
    median 0.949400
       min 0.174067
      rmse 4.633799
       sse 3134.926234
       std 4.299667
```

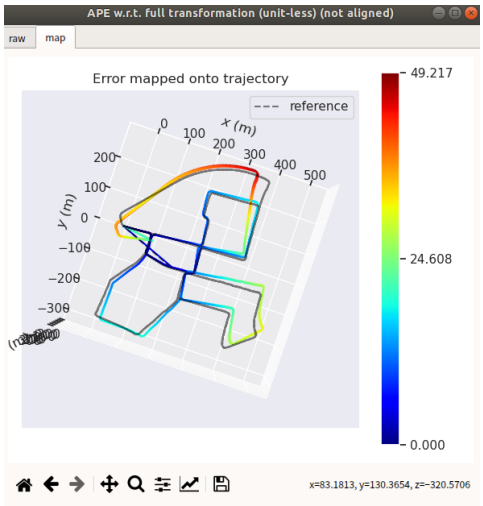




evo_ape

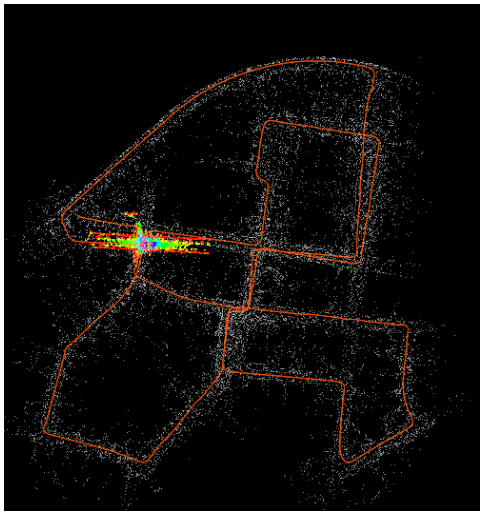
```
evo_ape kitti ground_truth.txt laser_odom.txt -r full --plot --plot_mode xyz
      max 49.216907
     mean 17.845561
   median 15.060993
      min 0.000000
     rmse 21.164576
      sse 6550216.305710
      std 11.378720
```





解析求导 + Ceres 自带四元数参数块

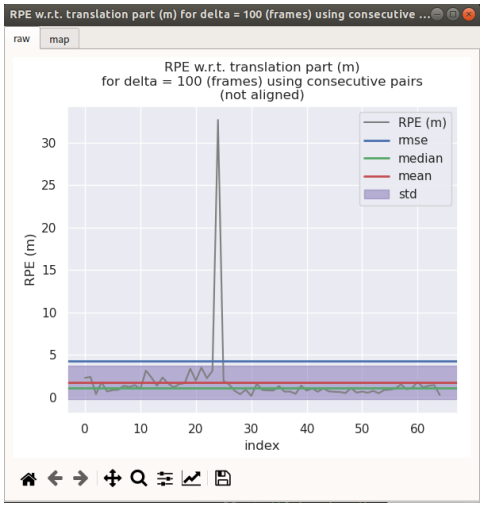
whole mapping time 174.593543 ms +++++

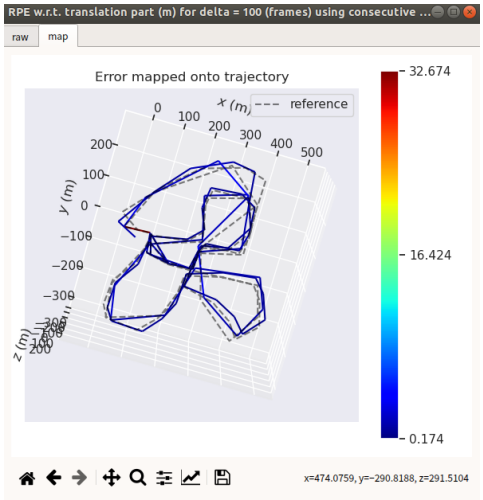


evo_rpe

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

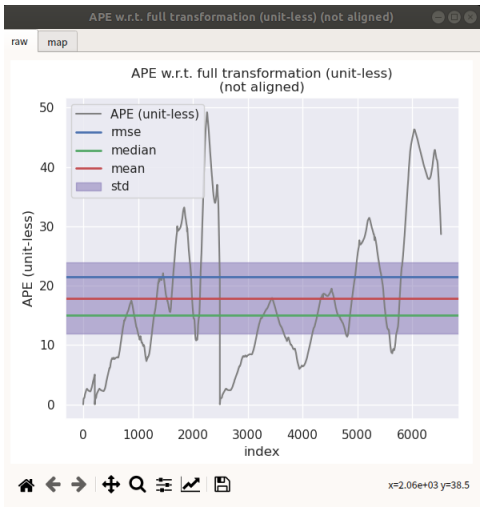
max	32.674307
mean	1.752770
median	1.073263
min	0.174067
rmse	4.309392
sse	1207.105821
std	3.936833

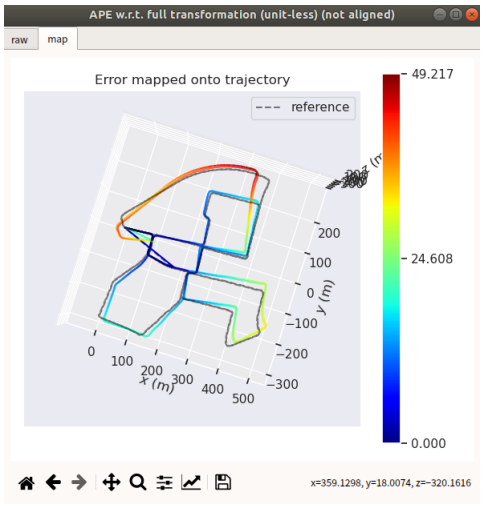




evo_ape

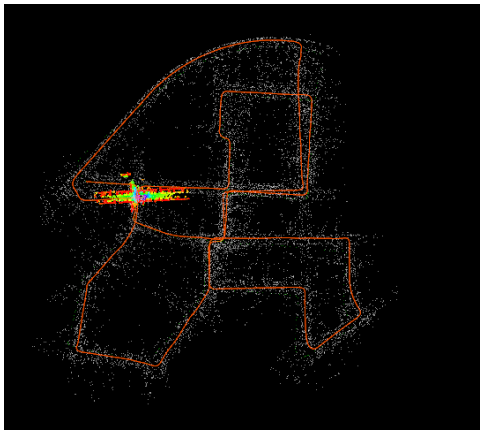
```
evo_ape kitti ground_truth.txt laser_odom.txt -r full --plot --plot_mode xyz
    max 49.216907
   mean 17.908280
 median 15.086003
    min 0.000000
   rmse 21.509767
    sse 3021235.643653
    std 11.914847
```





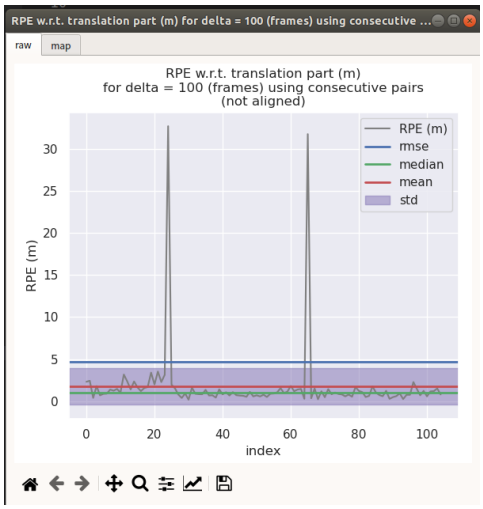
解析求导 + 自定义参数块

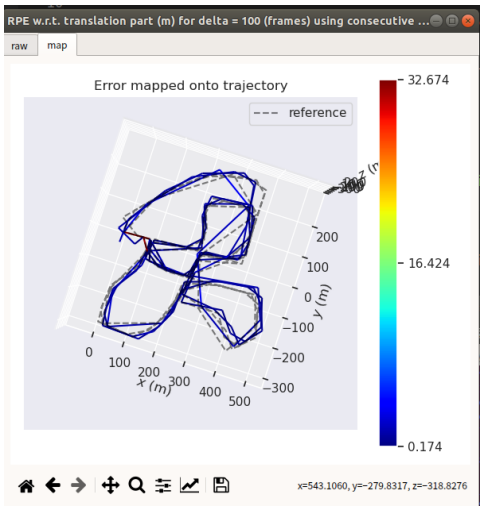
whole mapping time 170.378546 ms +++++



evo_rpe

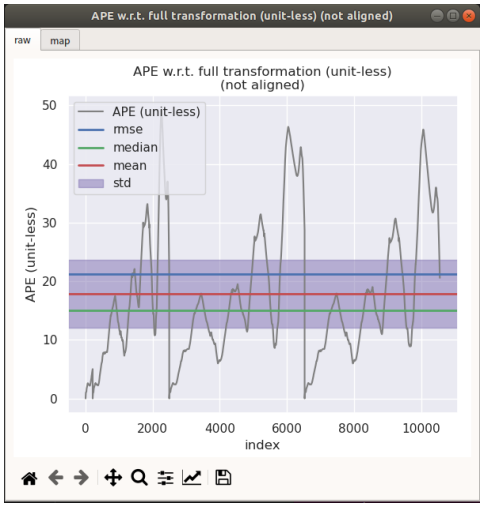
```
evo_rpe kitti ground_truth.txt laser_odom.txt -r trans_part --delta 100 --plot --plot_mode xyz
max 32.674307
 mean 1.727784
median 1.003377
  min 0.174067
 rmse 4.634043
  sse 2254.806880
  std 4.299897
```

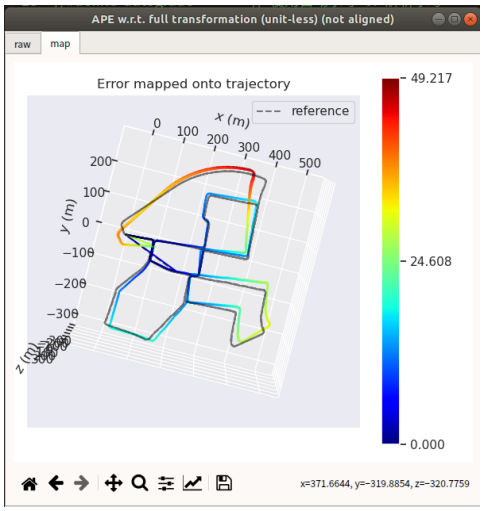




evo_ape

```
 evo_ape kitti ground_truth.txt laser_odom.txt -r full --plot --plot_mode xyz
 max  49.216907
 mean 17.860616
median 15.072742
  min 0.000000
 rmse 21.251952
  sse 4769827.822403
  std 11.517112
```





总结：

图中结果可以看出，缺乏回环矫正的情况下在拐弯的情况下由于实现流程和缺乏回环导致误差会偏大，解析求求导和自动求导的结果相似。总体来看轨迹的趋势还是正确的，算法应该没有问题，但是参数调整还有改进的余地。