

自动驾驶与机器人中的 SLAM技术

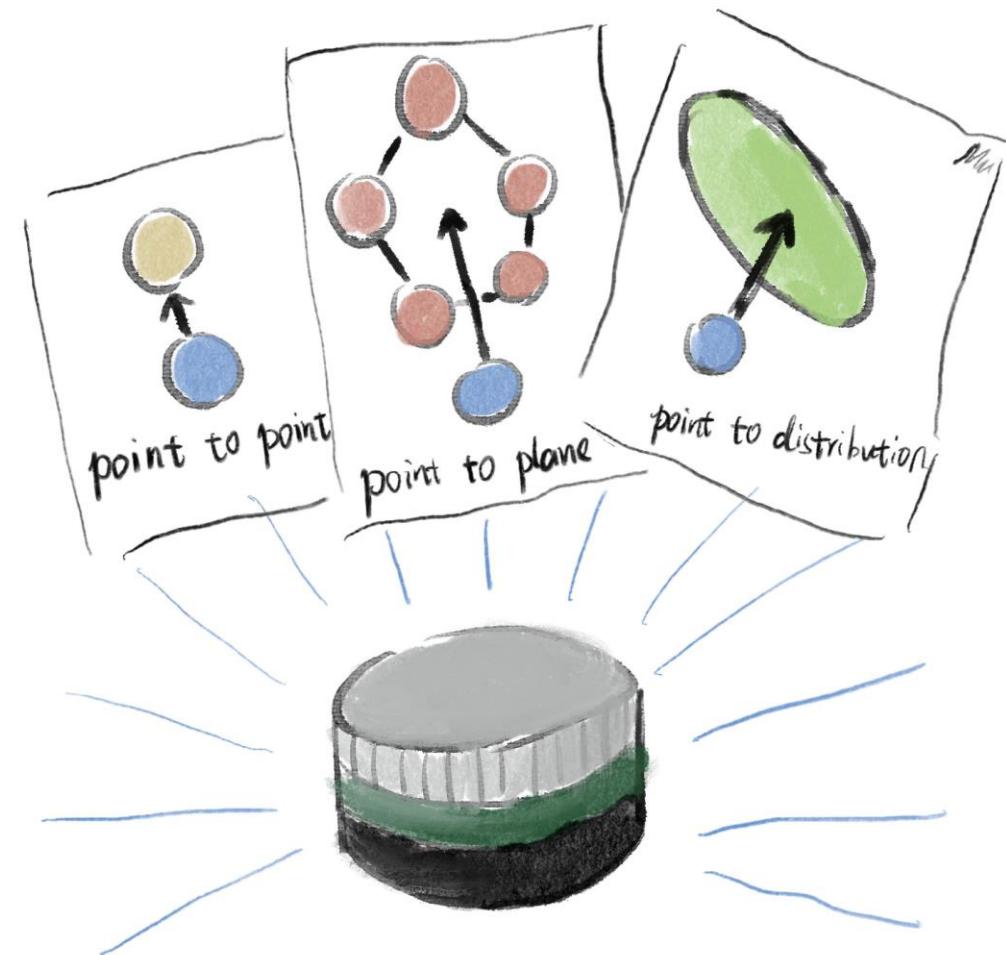
3D激光SLAM





Contents

- 多线激光的Scan Matching
- 直接法LO/特征法LO
- 松耦合LIO及工程问题



配准方法可以使用各种不同的
误差类型。

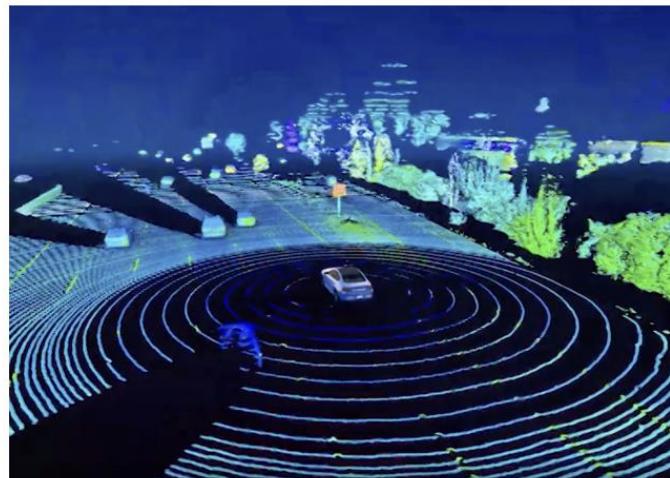


多线激光的Scan Matching



多线激光的Scan Matching

- 多线激光是自动驾驶主要使用的传感器
- 与单线激光类似，它可以一次探测多条点上的测距信息
- 常见的包括：16线、32线、64线、80线、128线



多线激光雷达存在众多品牌



大部分自动驾驶车辆使用顶部多线雷达方案



多线激光的Scan Matching

□ 点云的Scan Matching

- 与单线类似，单次扫描数据也称为一个scan
- 多线点云通常使用3D pose和3D点

待配准的点云: $S_1 = \{\mathbf{p}_1, \dots, \mathbf{p}_m\}$ $S_2 = \{\mathbf{q}_1, \dots, \mathbf{q}_n\}$

对一组匹配点: $\mathbf{p}_i \in S_1, \mathbf{q}_j \in S_2$

它们构成残差: $\mathbf{p}_i = \mathbf{R}\mathbf{q}_j + \mathbf{t}.$ 点到点残差形式



多线激光的Scan Matching

□ ICP: 交替求解数据关联问题和位姿估计问题

点到点ICP:

1. 设初始的位姿估计为 $\mathbf{R}_0, \mathbf{t}_0$ 。
2. 从初始位姿估计开始迭代。设第 k 次迭代时位姿估计为 $\mathbf{R}_k, \mathbf{t}_k$ 。
3. 在 $\mathbf{R}_k, \mathbf{t}_k$ 估计下，按照最近邻方式寻找匹配点。记匹配之后的点对为 $(\mathbf{p}_i, \mathbf{q}_i)$ 。
4. 计算本次迭代的结果：

$$\mathbf{R}_{k+1}, \mathbf{t}_{k+1} = \arg \min_{\mathbf{R}, \mathbf{t}} \sum_i \|\mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t})\|_2^2. \quad \text{这一步有闭式解, 当然也可以用优化}$$

5. 判断解是否收敛，若不收敛则返回 3，收敛则退出。



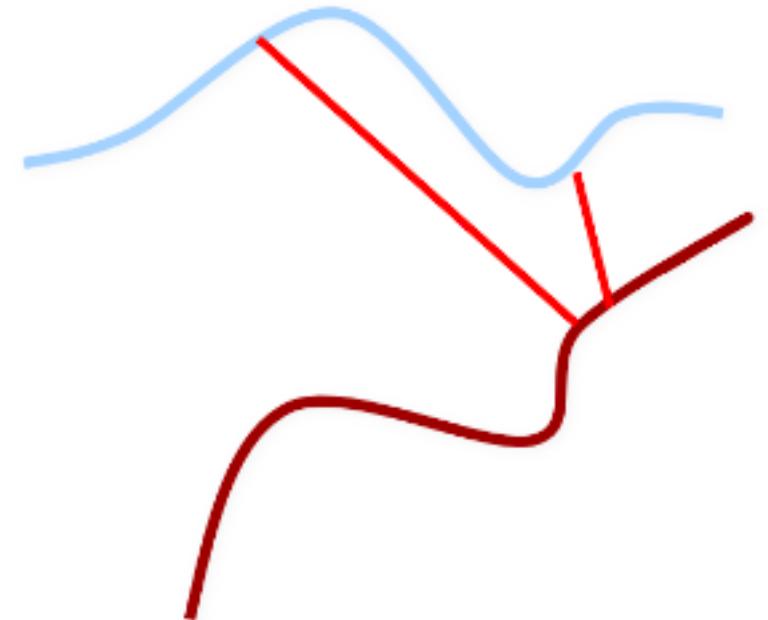
多线激光的Scan Matching

□ ICP的注解

1. 交替求解是对问题的简化。原则上， S_1 中某个点可能和 S_2 中任意一个点呈配对关系，但ICP只确认距离最近的点。
2. 位姿求解可以用闭式方法或者优化方法。如用优化，则还能引入核函数、权重信息。闭式方法在《十四讲》第7讲有介绍。
3. 点到点的残差是最简单的形式，也可以利用其他的形式。

残差: $e_i = p_i - Rq_i - t.$

雅可比: $\frac{\partial e_i}{\partial R} = Rq^\wedge, \quad \frac{\partial e_i}{\partial t} = -I.$





多线激光的Scan Matching

□ 点到面ICP

- 提取多个最近邻后，对最近邻进行平面拟合
- 平面方程与参数：

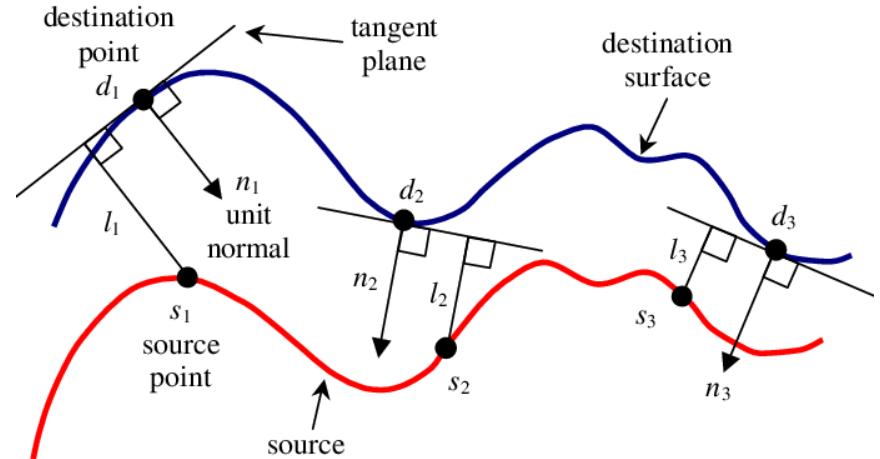
$$\mathbf{n}^\top \mathbf{p} + d = 0,$$

- 平面外一点 \mathbf{q}_1 到平面的距离：

$$e_i = \mathbf{n}^\top (\mathbf{R}\mathbf{q}_i + \mathbf{t}) + d. \quad \text{1维误差, 有向}$$

- 误差对旋转和平移的导数：

$$\frac{\partial e}{\partial \mathbf{R}} = -\mathbf{n}^\top \mathbf{R}\mathbf{q}_i^\wedge, \quad \frac{\partial e}{\partial \mathbf{t}} = \mathbf{n}.$$





多线激光的Scan Matching

□ 点到线ICP

- 直线参数: $\mathbf{p} = \mathbf{d}\tau + \mathbf{p}_0$, \mathbf{t} 用来表示平移了, 这里换个符号

误差取点到直线距离: $\mathbf{e}_i = \mathbf{d} \times (\mathbf{R}\mathbf{q}_i + \mathbf{t} - \mathbf{p}_0)$, 或 $\mathbf{e}_i = \mathbf{d}^\wedge(\mathbf{R}\mathbf{q}_i + \mathbf{t} - \mathbf{p}_0)$.

它关于 \mathbf{R}, \mathbf{t} 的导数为: $\frac{\partial \mathbf{e}_i}{\partial \mathbf{R}} = -\mathbf{d}^\wedge \mathbf{R} \mathbf{q}_i^\wedge$, $\frac{\partial \mathbf{e}_i}{\partial \mathbf{t}} = \mathbf{d}^\wedge$.

类似地, 也可以定义其他各种点到几何形状的ICP, 但线性形状最为简单直接。

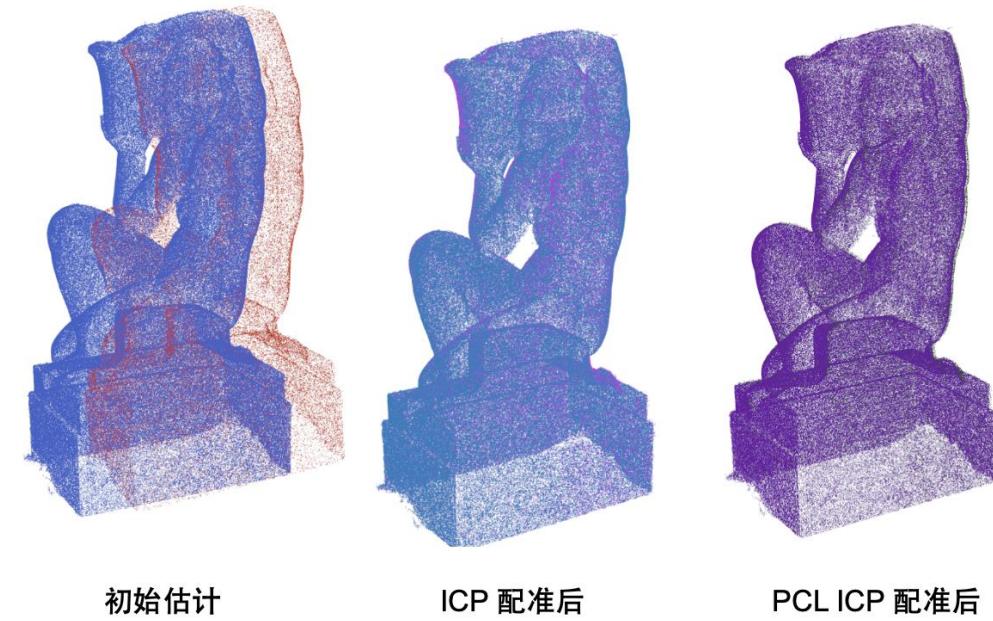


多线激光的Scan Matching

□ 代码实现与程序说明

- 使用EPFL雕像数据集进行测试，给定真实位姿并施加一定噪声
- 观察各配准算法的位姿收敛精度和计算时间
- 使用第5章的Kdtree作为最近邻结构

```
I0130 16:46:40.795749 84155 icp_3d.cc:13] aligning with point to point
I0130 16:46:40.805476 84155 icp_3d.cc:96] iter 0 total res: 11.523, eff: 44455, mean res:
0.000259205, dnx: 0.0271729
I0130 16:46:40.805512 84155 icp_3d.cc:101] iter 0 pose error: 0.0689234
I0130 16:46:40.811969 84155 icp_3d.cc:96] iter 1 total res: 6.15052, eff: 44515, mean res:
0.000138167, dnx: 0.020881
I0130 16:46:40.811990 84155 icp_3d.cc:101] iter 1 pose error: 0.0482068
I0130 16:46:40.817767 84155 icp_3d.cc:96] iter 2 total res: 3.09131, eff: 44515, mean res: 6.94443e
-05, dnx: 0.0151463
I0130 16:46:40.817786 84155 icp_3d.cc:101] iter 2 pose error: 0.0331819
I0130 16:46:40.823176 84155 icp_3d.cc:96] iter 3 total res: 1.53192, eff: 44515, mean res: 3.44137e
-05, dnx: 0.0106271
I0130 16:46:40.828301 84155 test_icp.cc:54] icp p2p align success, pose: 0.0265454 0-0.01074
0-0.02348 00.999314, -0.0688936 0-0.103293 0.00503732
I0130 16:46:40.878885 84155 sys_utils.h:32] 方法 ICP P2P 平均调用时间/次数: 163.421/1 毫秒.
...
I0130 16:46:42.158504 84155 test_icp.cc:140] pose from icp pcl: 0000.029222 -0.00915236 0-0.0195184
0000.999341, -0.0636124 -0.0567144 0.00273507
I0130 16:46:42.161834 84155 test_icp.cc:146] ICP PCL pose error: 0.262063
I0130 16:46:42.162148 84155 sys_utils.h:32] 方法 ICP PCL 平均调用时间/次数: 895.009/1 毫秒.
```





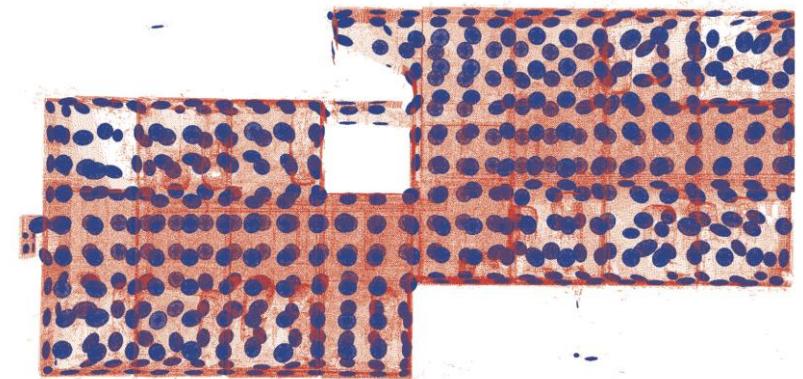
多线激光的Scan Matching

□ NDT

我们可以将点配准到它附近的单个点或某种形状，显然形状信息比单个点更丰富。因此，也可以将单个点配准到点云的统计信息，其中最简单的就是高斯分布。

NDT:

1. 把目标点云按一定分辨率分成若干体素。
2. 计算每个体素中的点云高斯分布。设第 k 个体素中的均值为 μ_k ，方差为 Σ_k 。
3. 配准时，先计算每个点落在哪个体素中，然后建立该点与该体素的 μ_k, Σ_k 构成的残差。
4. 利用 G-N 或 L-M 方法对估计位姿进行迭代，然后更新位姿估计。



每个体素中点云以高斯分布来表示



多线激光的Scan Matching

□ NDT配准

- 某个点与体素的均值和协方差作残差: $\mathbf{e}_i = \mathbf{R}\mathbf{q}_i + \mathbf{t} - \boldsymbol{\mu}_i$,
- 协方差控制最小二乘的权重:

$$(\mathbf{R}, \mathbf{t})^* = \arg \min_{\mathbf{R}, \mathbf{t}} \sum_i (\mathbf{e}_i^\top \boldsymbol{\Sigma}_i^{-1} \mathbf{e}_i).$$

等价于最大似然估计: $(\mathbf{R}, \mathbf{t})^* = \arg \max_{\mathbf{R}, \mathbf{t}} \prod_i P(\mathbf{R}\mathbf{q}_i + \mathbf{t})$

即: 配准后点云应该服从该统计分布。

它的高斯牛顿解法:

$$\sum_i (\mathbf{J}_i^\top \boldsymbol{\Sigma}_i^{-1} \mathbf{J}_i) \Delta \mathbf{x} = - \sum_i \mathbf{J}_i^\top \boldsymbol{\Sigma}_i^{-1} \mathbf{e}_i,$$

雅可比矩阵: $\frac{\partial \mathbf{e}_i}{\partial \mathbf{R}} = -\mathbf{R}\mathbf{q}_i^\wedge, \quad \frac{\partial \mathbf{e}_i}{\partial \mathbf{t}} = \mathbf{I}.$

注: 本书推导的NDT比原始论文简单很多。



多线激光的Scan Matching

□ NDT实现以及注意事项

- 体素也存在最近邻关系，可以用KD树或者体素最近邻实现；
- 注意体素可能存在欠采样或者呈退化分布，导致协方差矩阵不可逆或者逆矩阵存在数值极大的分量；
- 体素大小会明显影响配准效果。

□ 实现当中要点

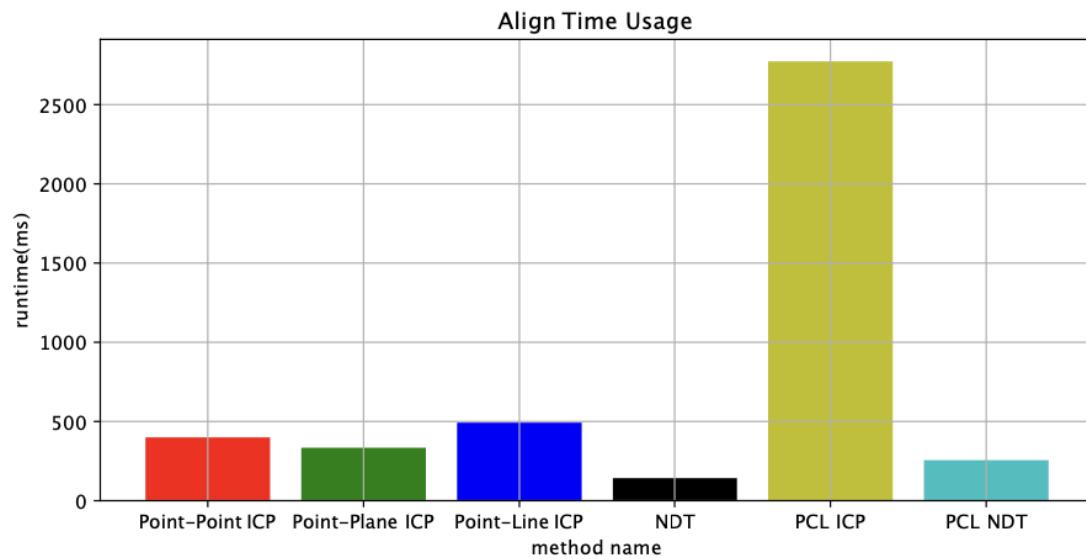
- 体素适当取大一些，稀疏体素即可（PCL默认稠密体素）；
- 限制每个体素中的最小点数；
- 可以用采样点数加权，采样充分的体素权重更大；
- 对协方差矩阵求特征值分解，限制最小特征值（例如，特征值小于0.1的角上取0.1），相当于保留栅格内的不确定性；
- 用Robust Kernel效果会更好（安排习题验证）。



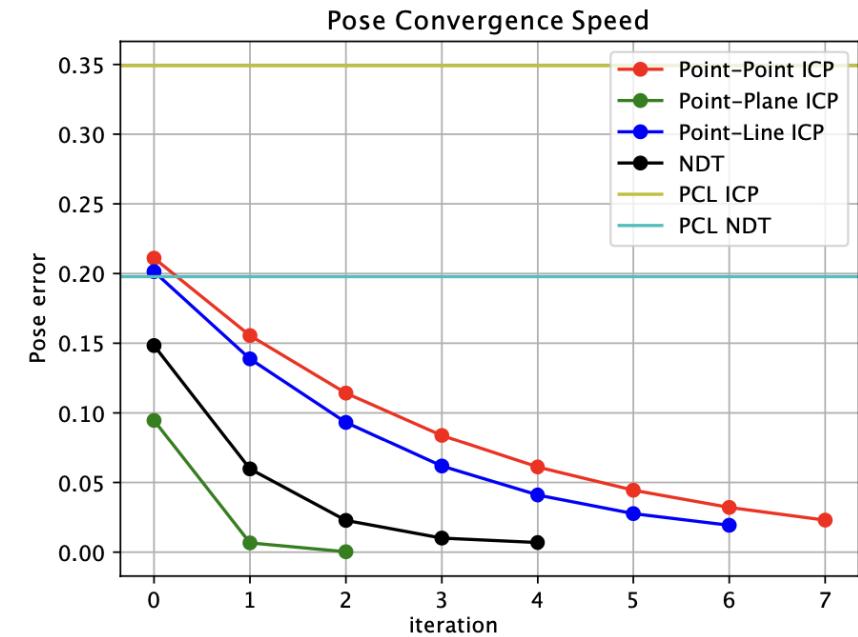
多线激光的Scan Matching

□ NDT的实现以及各种配准算法的对比

- 由于本书使用并发编程（并发最近邻+残差计算），大部分算法要比PCL版本快很多
- NDT最快，但位姿收敛精度不如点面ICP



性能对比



有真值Pose时，可对比每一步迭代的Pose精度



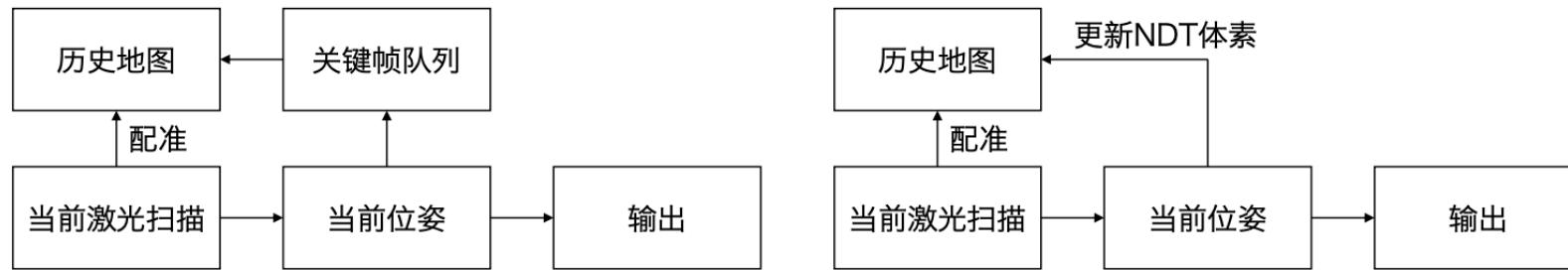
直接法LO/间接法LO



直接法LO/间接法LO

利用帧间的相对配准，就可以实现激光里程计（Lidar odometry）

- 比较粗暴的方式是直接将若干帧的点云累计起来，形成**局部地图**，再将新的点云配准到此地图上；
- 该方法的问题是：局部地图的最近邻结构需要不断更新，默认情况下，每添加一个点云，就需要重新构建Kd树或重新估计体素结构。



简化的Lidar odometry框架



直接法LO/间接法LO

□ 直接法LO的实现

```
void DirectNDTLO::AddCloud(CloudPtr scan, SE3& pose) {
    if (local_map_ == nullptr) {
        // 第一个帧，直接加入local map
        local_map_.reset(new PointCloudType);
        *local_map_ += *scan;
        pose = SE3();
        last_kf_pose_ = pose;
    }

    if (options_.use_pcl_ndt_) {
        ndt_pcl_.setInputTarget(local_map_);
    } else {
        ndt_.SetTarget(local_map_);
    }

    return;
}
```

```
// 计算scan相对于local map的位姿
pose = AlignWithLocalMap(scan);
CloudPtr scan_world(new PointCloudType);
pcl::transformPointCloud(*scan, *scan_world, pose.matrix().cast<float>());

if (IsKeyframe(pose)) {
    last_kf_pose_ = pose;

    // 重建local map
    scans_in_local_map_.emplace_back(scan_world);
    if (scans_in_local_map_.size() > options_.num_kfs_in_local_map_) {
        scans_in_local_map_.pop_front();
    }
}

local_map_.reset(new PointCloudType);
for (auto& scan : scans_in_local_map_) {
```

与Local Map配准

```
local_map_.reset(new PointCloudType);
for (auto& scan : scans_in_local_map_) {
```

每次都重新建立local map
并设为NDT的目标点云 (NDT的体素会重新构建)

278

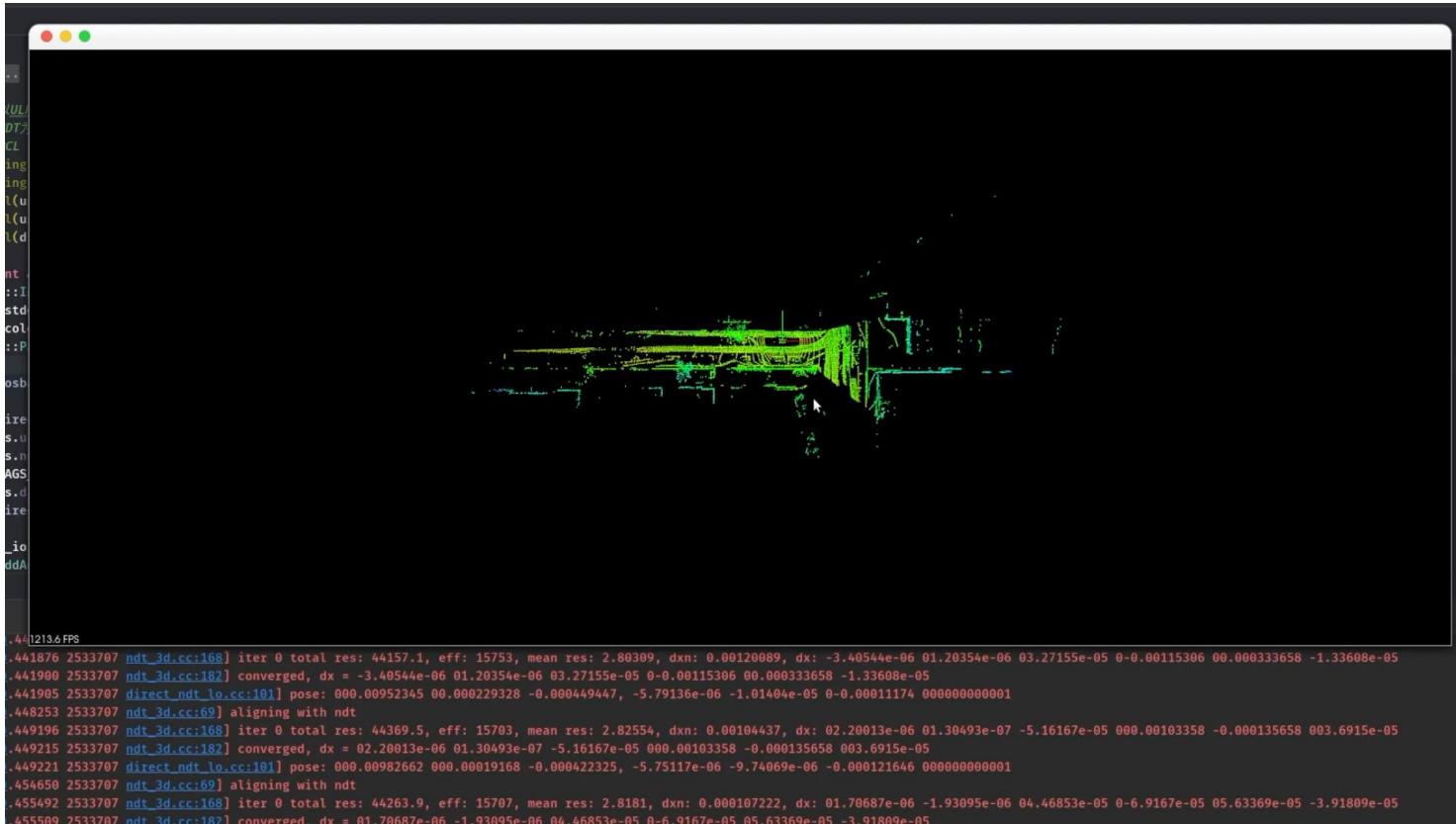
```
*local_map_ += *scan;
}

if (options_.use_pcl_ndt_) {
    ndt_pcl_.setInputTarget(local_map_);
} else {
    ndt_.SetTarget(local_map_);
}
}
```



直接法LO/间接法LO

□ NDT LO的实际测试效果



ULHK数据集

使用本章NDT时 (NEARBY1) :

- 带上PCL可视化, 每帧约35ms
- 不带可视化, 每帧约15ms

使用PCL NDT时:

- 带可视化, 约109ms
- 不带可视化, 约88ms



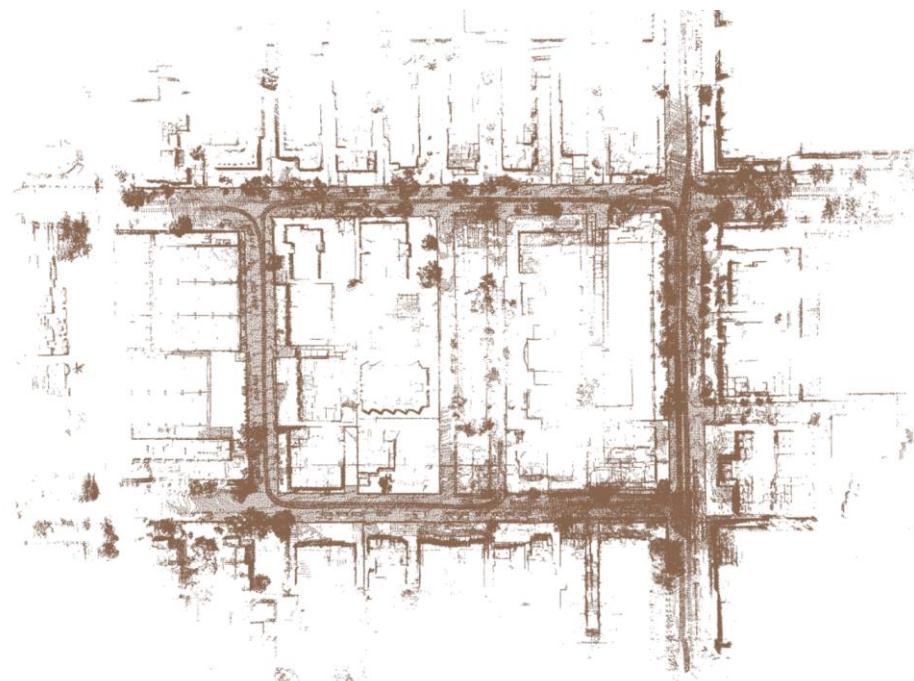
直接法LO/间接法LO

□ 对上述LO的改进：

- 不要直接拼接关键帧点云（需要重新计算整个local map）
- 不要重新构建NDT体素（大部分是已经有的），也不要重新构建Kdtree
- 一次性，更优雅地解决上面两个问题

Incremental NDT LO：

- 增量式维护NDT体素
- 当新点云插入时，更新体素分布和内部的高斯分布
- 使用哈希方法维护体素的稀疏分布
- 使用LRU cache删除已经远离车辆的体素
- 利用体素求取最近邻，不使用Kdtree
- 该部分实际就是faster-lio中的ivox





直接法LO/间接法LO

□ 增量体素结构

```
/// 体素内置结构
struct VoxelData {
    VoxelData() {}
    VoxelData(const Vec3d& pt) {
        pts_.emplace_back(pt);
        num_pts_ = 1;
    }

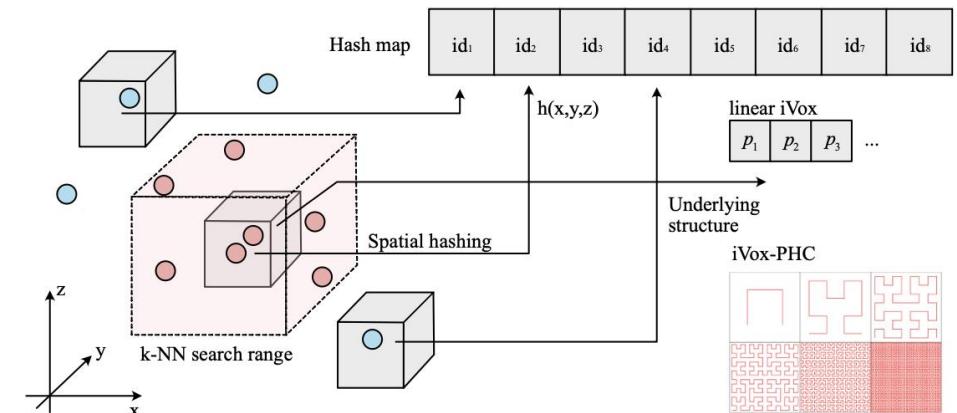
    void AddPoint(const Vec3d& pt) {
        pts_.emplace_back(pt);
        if (!ndt_estimated_) {
            num_pts_++;
        }
    }

    std::vector<Vec3d> pts_;           // 内部点, 多于一定数量之后再估计均值和协方差
    Vec3d mu_ = Vec3d::Zero();          // 均值
    Mat3d sigma_ = Mat3d::Zero();       // 协方差
    Mat3d info_ = Mat3d::Zero();        // 协方差之逆

    bool ndt_estimated_ = false; // NDT是否已经估计
    int num_pts_ = 0;              // 总共的点数, 用于更新估计
};
```

实际数据和它的索引

```
using KeyAndData = std::pair<KeyType, VoxelData>; // 预定义
std::list<KeyAndData> data_;                         // 真实数据, 会缓存, 也会清理
std::unordered_map<KeyType, std::list<KeyAndData>::iterator, hash_vec<3>> grids_; // 栅格数据, 存储真实数据的迭代器
std::vector<KeyType> nearby_grids_;                  // 附近的栅格
```





直接法LO/间接法LO

□ 增量更新一个高斯分布

- 原先有 m 个点，参数为 $\boldsymbol{\mu}_H, \boldsymbol{\Sigma}_H$ ；增加 n 个点，参数为 $\boldsymbol{\mu}_A, \boldsymbol{\Sigma}_A$ ；
- 更新之后，均值为：

$$\boldsymbol{\mu} = \frac{\mathbf{x}_1 + \dots + \mathbf{x}_m + \mathbf{y}_1 + \dots + \mathbf{y}_n}{m+n} = \frac{m\boldsymbol{\mu}_H + n\boldsymbol{\mu}_A}{m+n}.$$

- 方差定义为：

$$\boldsymbol{\Sigma} = \frac{1}{m+n} \left(\sum_{i=1}^m (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top + \sum_{j=1}^n (\mathbf{y}_j - \boldsymbol{\mu})(\mathbf{y}_j - \boldsymbol{\mu})^\top \right).$$



直接法LO/间接法LO

□ 左侧部分：

$$\boldsymbol{\Sigma} = \frac{1}{m+n} \left(\sum_{i=1}^m (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top + \sum_{j=1}^n (\mathbf{y}_j - \boldsymbol{\mu})(\mathbf{y}_j - \boldsymbol{\mu})^\top \right).$$

$$\begin{aligned} \sum_{i=1}^m (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top &= \sum_{i=1}^m [\mathbf{x}_i - \boldsymbol{\mu}_H + (\boldsymbol{\mu}_H - \boldsymbol{\mu})] [\mathbf{x}_i - \boldsymbol{\mu}_H + (\boldsymbol{\mu}_H - \boldsymbol{\mu})]^\top, \\ &= \sum_{i=1}^m \left(\underbrace{(\mathbf{x}_i - \boldsymbol{\mu}_H)(\mathbf{x}_i - \boldsymbol{\mu}_H)^\top}_{=\boldsymbol{\Sigma}_H} + (\mathbf{x}_i - \boldsymbol{\mu}_H)(\boldsymbol{\mu}_H - \boldsymbol{\mu})^\top \right. \\ &\quad \left. + (\boldsymbol{\mu}_H - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu}_H)^\top + (\boldsymbol{\mu}_H - \boldsymbol{\mu})(\boldsymbol{\mu}_H - \boldsymbol{\mu})^\top \right). \end{aligned}$$

易见中间两项为零

于是左侧为：

$$\sum_{i=1}^m (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top = m(\boldsymbol{\Sigma}_H + (\boldsymbol{\mu}_H - \boldsymbol{\mu})(\boldsymbol{\mu}_H - \boldsymbol{\mu})^\top).$$

同理，右侧为：

$$\sum_{j=1}^n (\mathbf{y}_j - \boldsymbol{\mu})(\mathbf{y}_j - \boldsymbol{\mu})^\top = n(\boldsymbol{\Sigma}_A + (\boldsymbol{\mu}_A - \boldsymbol{\mu})(\boldsymbol{\mu}_A - \boldsymbol{\mu})^\top).$$

合并后： $\boldsymbol{\Sigma} = \frac{m(\boldsymbol{\Sigma}_H + (\boldsymbol{\mu}_H - \boldsymbol{\mu})(\boldsymbol{\mu}_H - \boldsymbol{\mu})^\top) + n(\boldsymbol{\Sigma}_A + (\boldsymbol{\mu}_A - \boldsymbol{\mu})(\boldsymbol{\mu}_A - \boldsymbol{\mu})^\top)}{m+n}.$ 即NDT增量估计



直接法LO/间接法LO

□ 增量更新的实现：

```
void IncNdt3d::UpdateVoxel(VoxelData& v) {
    if (v.ndt_estimated_ && v.num_pts_ > options_.max_pts_in_voxel_) {
        return;
    }

    if (!v.ndt_estimated_ && v.pts_.size() > options_.min_pts_in_voxel_) {
        // 新增的 voxel
        math::ComputeMeanAndCov(v.pts_, v.mu_, v.sigma_, [this](const Vec3d& p) { return p; });
        v.info_ = (v.sigma_ + Mat3d::Identity() * 1e-3).inverse(); // 避免矩阵不可逆的情况
        v.ndt_estimated_ = true;
        v.pts_.clear();
    } else if (v.ndt_estimated_ && v.pts_.size() > options_.min_pts_in_voxel_) {
        // 已经估计，而且还有新来的点
        Vec3d cur_mu, new_mu;
        Mat3d cur_var, new_var;
        math::ComputeMeanAndCov(v.pts_, cur_mu, cur_var, [this](const Vec3d& p) { return p; });
        math::UpdateMeanAndCov(v.num_pts_, v.pts_.size(), v.mu_, v.sigma_, cur_mu, cur_var, new_mu,
                               new_var);

        v.mu_ = new_mu;
        v.sigma_ = new_var;
        v.num_pts_ += v.pts_.size();
        v.pts_.clear();

        // check info
        Eigen::JacobiSVD svd(v.sigma_, Eigen::ComputeFullU | Eigen::ComputeFullV);
        Vec3d lambda = svd.singularValues();
        if (lambda[1] < lambda[0] * 1e-3) {
            lambda[1] = lambda[0] * 1e-3;
        }

        if (lambda[2] < lambda[0] * 1e-3) {
            lambda[2] = lambda[0] * 1e-3;
        }

        Mat3d inv_lambda = Vec3d(1.0 / lambda[0], 1.0 / lambda[1], 1.0 / lambda[2]).asDiagonal();
        v.info_ = svd.matrixV() * inv_lambda * svd.matrixU().transpose();
    }
}
```

数学部分

src/common/math_utils.h

```
template <typename S, int D>
void UpdateMeanAndCov(int hist_m, int curr_n, const Eigen::Matrix<S, D, 1>& hist_mean,
                      const Eigen::Matrix<S, D, D>& hist_var, const Eigen::Matrix<S, D, 1>& curr_mean,
                      const Eigen::Matrix<S, D, D>& curr_var, Eigen::Matrix<S, D, 1>& new_mean,
                      Eigen::Matrix<S, D, D>& new_var) {
    new_mean = (hist_m * hist_mean + curr_n * curr_mean) / (hist_m + curr_n);
    new_var = (hist_m * (hist_var + (hist_mean - new_mean) * (hist_mean - new_mean).template transpose
                         ()) + curr_n * (curr_var + (curr_mean - new_mean) * (curr_mean - new_mean).template transpose
                         ()))/(hist_m + curr_n);
}
```

维护奇异值



直接法LO/间接法LO

□ 增加NDT LO的实现

- 现在不需要维护local map了，因为已经在NDT内部完成了



相比PCL版本的NDT LO，性能可以提升10倍以上

性能指标

- 默认参数，打开可视化，每帧约23.3ms
- 关闭可视化，每帧约6ms



直接法LO/间接法LO

□ 特征法LO

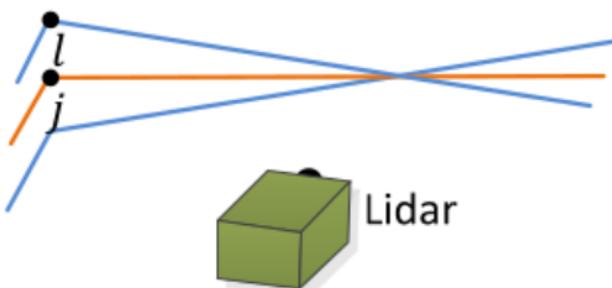
- 先对点云形状提取特征，再根据不同特征使用不同的配准方法；
- 典型的特征：地面点、立面点、天花板、柱状物；
- 常见的特征法LO：LOAM系列（ALOAM、FLOAM、LeGO-LOAM等）；
- 特征法可以显著降低要配准的点数，但提特征所需要的时间也不可忽略。



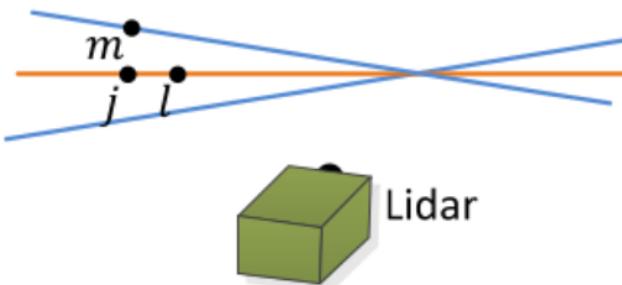
直接法LO/间接法LO

□ 点云特征大部分基于线束曲率来提取，或者通过range image来提取

- 例如LOAM：沿线束曲率较大的是角点，曲率较小的是平面点
- 通过分区分阈值手段，让特征分布更加均匀
- 对线点，使用点线ICP；对面点，使用点面ICP



(a)



(b)

- 特征提取方法通常是非常工程化的计算，我们不准备展开介绍
- 其本身也没有很多数学根据，大部分只有实践意义



直接法LO/间接法LO

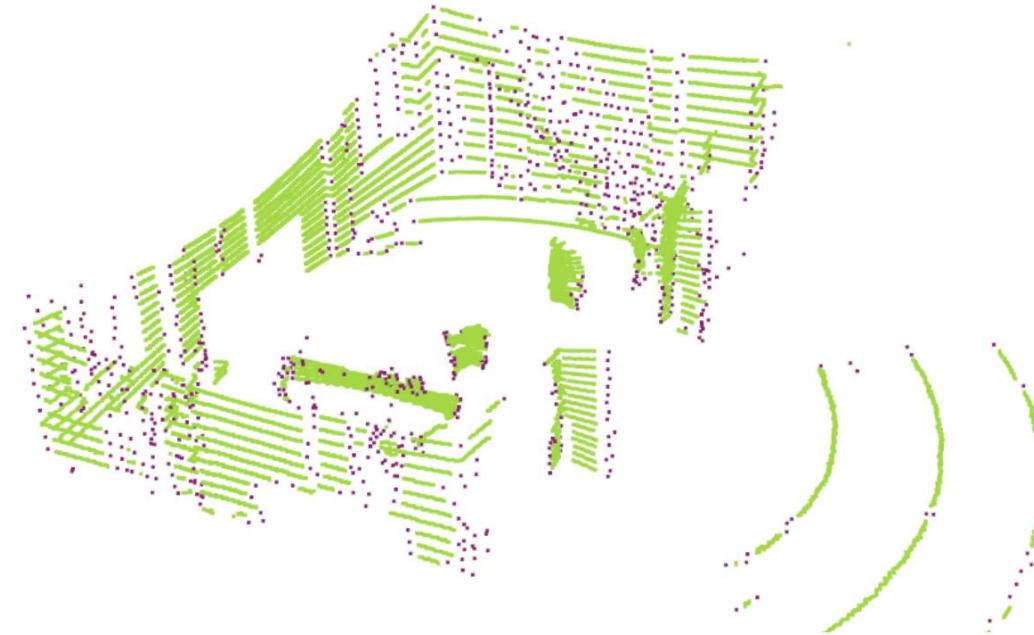
□ 特征提取的实现：

```
for (int j = 5; j < (int)scans_in_each_line[i]->points.size() - 5; j++) {
    // 两头留一定余量，采样周围10个点取平均值
    double diffX = scans_in_each_line[i]->points[j - 5].x + scans_in_each_line[i]->points[j - 4].x +
        scans_in_each_line[i]->points[j - 3].x + scans_in_each_line[i]->points[j - 2].x +
        scans_in_each_line[i]->points[j - 1].x - 10 * scans_in_each_line[i]->points[j].x +
        scans_in_each_line[i]->points[j + 1].x + scans_in_each_line[i]->points[j + 2].x +
        scans_in_each_line[i]->points[j + 3].x + scans_in_each_line[i]->points[j + 4].x +
        scans_in_each_line[i]->points[j + 5].x;
    double diffY = scans_in_each_line[i]->points[j - 5].y + scans_in_each_line[i]->points[j - 4].y +
        scans_in_each_line[i]->points[j - 3].y + scans_in_each_line[i]->points[j - 2].y +
        scans_in_each_line[i]->points[j - 1].y - 10 * scans_in_each_line[i]->points[j].y +
        scans_in_each_line[i]->points[j + 1].y + scans_in_each_line[i]->points[j + 2].y +
        scans_in_each_line[i]->points[j + 3].y + scans_in_each_line[i]->points[j + 4].y +
        scans_in_each_line[i]->points[j + 5].y;
    double diffZ = scans_in_each_line[i]->points[j - 5].z + scans_in_each_line[i]->points[j - 4].z +
        scans_in_each_line[i]->points[j - 3].z + scans_in_each_line[i]->points[j - 2].z +
        scans_in_each_line[i]->points[j - 1].z - 10 * scans_in_each_line[i]->points[j].z +
        scans_in_each_line[i]->points[j + 1].z + scans_in_each_line[i]->points[j + 2].z +
        scans_in_each_line[i]->points[j + 3].z + scans_in_each_line[i]->points[j + 4].z +
        scans_in_each_line[i]->points[j + 5].z;
    IdAndValue distance(j, diffX * diffX + diffY * diffY + diffZ * diffZ);
    cloud_curvature.push_back(distance);
}

// 对每个区间选取特征，把360度分为6个区间
for (int j = 0; j < 6; j++) {
    int sector_length = (int)(total_points / 6);
    int sector_start = sector_length * j;
    int sector_end = sector_length * (j + 1) - 1;
    if (j == 5) {
        sector_end = total_points - 1;
    }

    std::vector<IdAndValue> sub_cloud_curvature(cloud_curvature.begin() + sector_start,
    cloud_curvature.begin() + sector_end);

    ExtractFromSector(scans_in_each_line[i], sub_cloud_curvature, pc_out_edge, pc_out_surf);
}
```



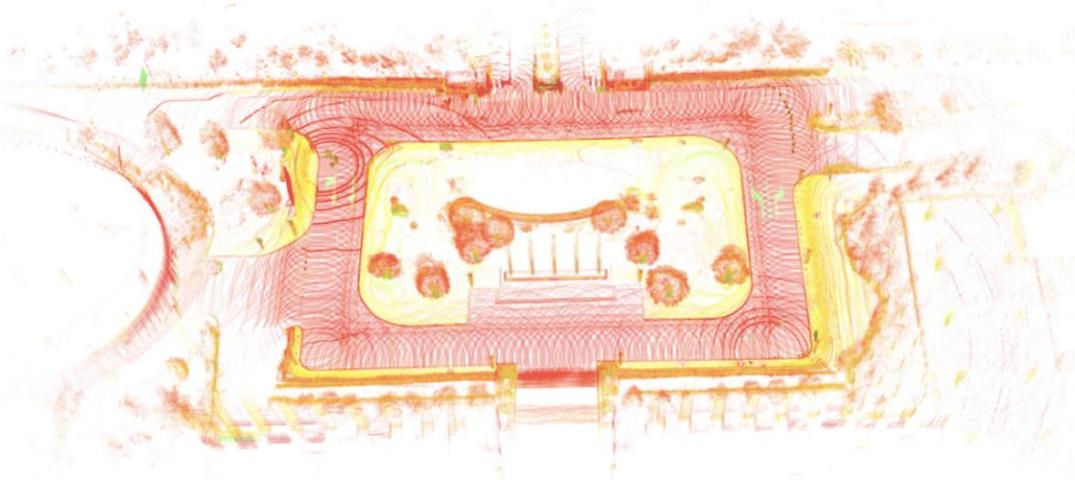
一个scan中的平面点和角点



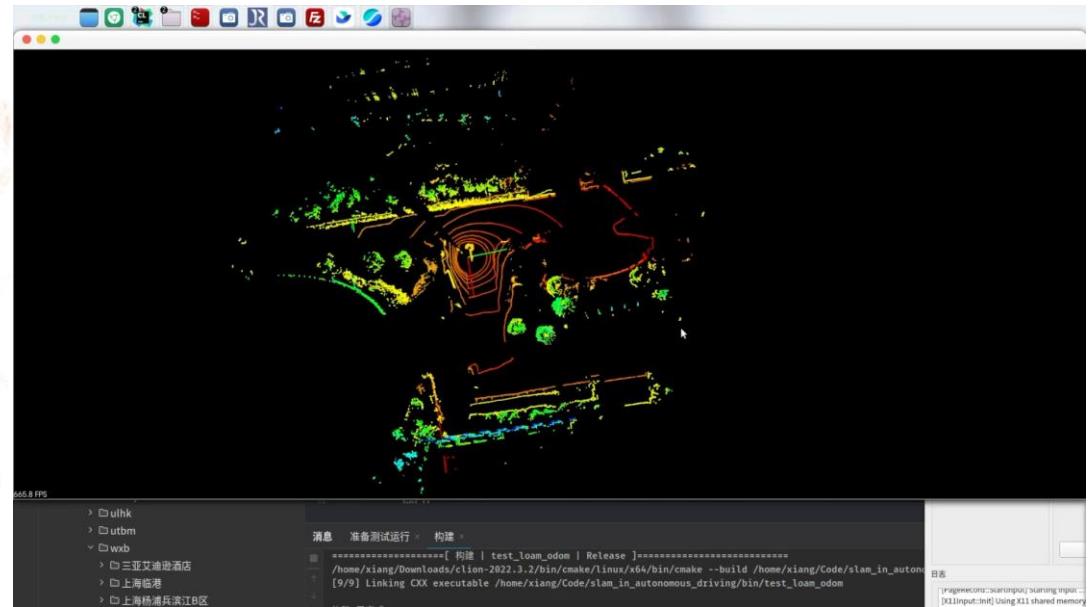
直接法LO/间接法LO

□ 基于特征法LO的实现（Loam-like LO）

- 由于需要用到激光线束信息，对不同类型的激光就没法兼容



性能指标：关闭可视化时，约18ms/帧，和直接法NDT相当





松耦合LIO及其工程问题



松耦合LIO及其工程问题

□ 纯激光的LO的问题

- 和单线一样，纯激光容易受环境结构、scan内容影响，可能产生退化；
- 比较简单的做法是让LO与IMU融合，形成LIO系统或者LINS系统。

□ 本节演示一个松耦合LIO案例

- 利用第3章介绍的ESKF处理IMU数据；
- 利用增量NDT LO来处理点云；
- LO的估计位姿作为ESKF的位姿观测来源；
- 处理激光和IMU的坐标系关系；
- 利用IMU和ESKF对点云去畸变。



松耦合LIO及其工程问题

□ 在LIO系统中，现在有三个主要的坐标系

- 世界系 W , IMU系 I , 激光系 L
- 通常要在IMU系下处理IMU读数, 所以比较简单的做法是将激光转到IMU系下;
- 设激光点为 \mathbf{p}_L , 那么可以通过外参变换到IMU系中:

$$\mathbf{p}_I = \mathbf{T}_{IL}\mathbf{p}_L = \mathbf{R}_{IL}\mathbf{p}_L + \mathbf{t}_{IL}.$$

- 我们将这一步列入预处理环节, 拿到的点云就是IMU系下的。



松耦合LIO及其工程问题

□ 松耦合LIO的运动与观测方程

运动方程保持不变：

$$\mathbf{F} = \begin{bmatrix} \mathbf{I} & \mathbf{I}\Delta t & 0 & 0 & 0 & 0 \\ 0 & \mathbf{I} & -\mathbf{R}(\tilde{\mathbf{a}} - \mathbf{b}_a)^\wedge\Delta t & 0 & -\mathbf{R}\Delta t & \mathbf{I}\Delta t \\ 0 & 0 & \text{Exp}(-(\tilde{\boldsymbol{\omega}} - \mathbf{b}_g)\Delta t) & -\mathbf{I}\Delta t & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{I} \end{bmatrix}$$

LO直接观测 \mathbf{R}, \mathbf{p}

同GNSS观测，对旋转也直接视为观测误差状态：

$$\delta\boldsymbol{\theta} = \text{Log}(\mathbf{R}^\top \mathbf{R}_{LO}). \quad \mathbf{r}_{\mathbf{R}} = -\text{Log}(\mathbf{R}^\top \mathbf{R}_{LO}).$$

此时旋转和平移的雅可比均为恒等阵。



松耦合LIO及其工程问题

□ 数据同步

- 我们将两个scan之间的IMU数据收集起来，形成一个测量数据结构

```
/// IMU 数据与雷达同步
struct MeasureGroup {
    MeasureGroup() { this->lidar_.reset(new FullPointCloudType()); }

    double lidar_begin_time_ = 0; // 雷达包的起始时间
    double lidar_end_time_ = 0; // 雷达的终止时间
    FullCloudPtr lidar_ = nullptr; // 雷达点云
    std::deque<IMUPtr> imu_; // 上一时刻到现在的IMU读数
};

/**
 * 将激光数据和IMU数据同步
*/
class MessageSync {
public:
    using Callback = std::function<void(const MeasureGroup &)>;
    MessageSync(Callback cb) : callback_(cb), conv_(new CloudConvert) {}

    /// 初始化
    void Init(const std::string &yaml);
```

通过lambda函数，实现同步之后的回调



松耦合LIO及其工程问题

□ 利用ESKF实现运动补偿（去畸变）

- 运动补偿即考虑单次Scan过程中车辆的运动
- 运动补偿需要查询Scan的起始和结束位姿，而高频IMU就可以提供这个观测源
- 记雷达得到的某个点为 $\mathbf{p}_t = (x, y, z)^\top, t \in (0, t_s)$, 其中 t 为扫描时间
- 设该时刻IMU位姿为 $\mathbf{T}(t)_{WI}$, 结束时刻为: $\mathbf{T}(t_s)_{WI}$
- 那么利用位姿变换, 就可以将该点变换到结束时刻的坐标系下

$$\mathbf{p}' = \mathbf{T}_{LI}\mathbf{T}(t_s)_{IW}\mathbf{T}(t)_{WI}\mathbf{T}_{IL}\mathbf{p}_t.$$



松耦合LIO及其工程问题

□ 去畸变代码：

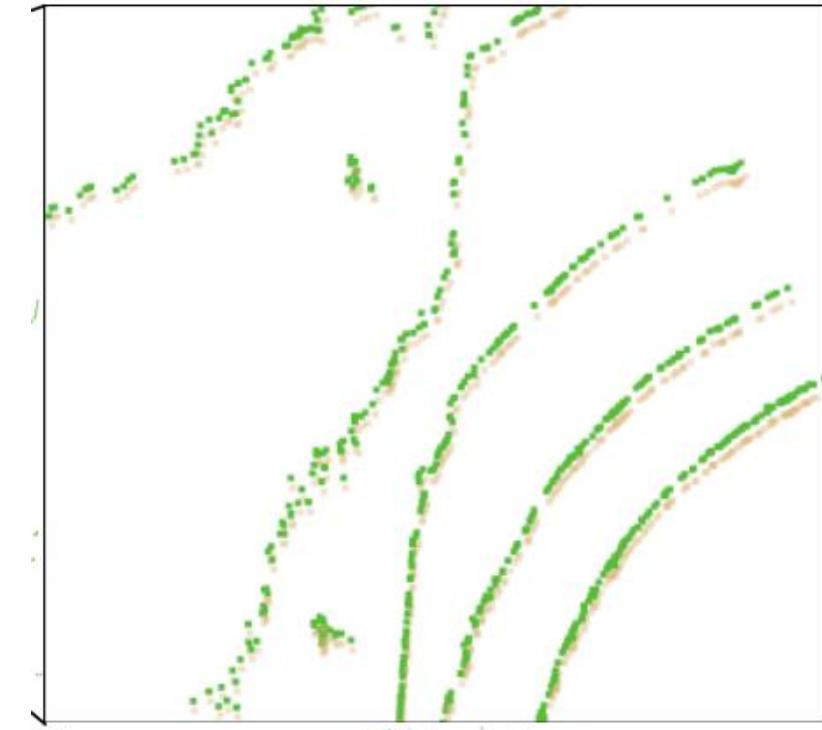
```
src/ch7/loosely_coupled_lio/loosly_lio.cc
void LooselyLIO::Undistort() {
    auto cloud = measures_.lidar_;
    auto imu_state = eskf_.GetNominalState(); // 最后时刻的状态
    SE3 T_end = SE3(imu_state.R_, imu_state.p_);

    /// 将所有点转到最后时刻状态上
    std::for_each(std::execution::par_unseq, cloud->points.begin(), cloud->points.end(), [&](auto &pt)
    {
        SE3 Ti = T_end;
        NavStated match;

        // 根据pt.time查找时间, pt.time是该点测量时间与雷达开始时间之差, 单位为毫秒
        math::PoseInterp<NavStated>(
            measures_.lidar_begin_time_ + pt.time * 1e-3, imu_states_, [](<const NavStated &s) { return s.
                timestamp_; }, [](<const NavStated &s) { return s.GetSE3(); }, Ti, match);

        Vec3d pi = ToVec3d(pt);
        Vec3d p_compensate = TIL_.inverse() * T_end.inverse() * Ti * TIL_ * pi;

        pt.x = p_compensate(0);
        pt.y = p_compensate(1);
        pt.z = p_compensate(2);
    });
    scan_undistort_ = cloud;
}
```



去畸变效果图



松耦合LIO及其工程问题

□ LooselyLIO的主要代码部分：

```
/// 从EKF中获取预测pose，放入LO，获取LO位姿，最后合入EKF
SE3 pose_predict = eskf_.GetNominalSE3();
inc_ndt_lo_->AddCloud(current_scan_filter, pose_predict, true);

pose_of_lo_ = pose_predict;
eskf_.ObserveSE3(pose_of_lo_, 1e-2, 1e-2);
```

由于LooselyLIO框架层面还对点云进行了降采样，整体速度会比增量NDT LO更快
UTBM、NCLT等32线数据集可以达到2ms/帧左右



松耦合LIO及其工程问题

□ LooslyLIO的实现





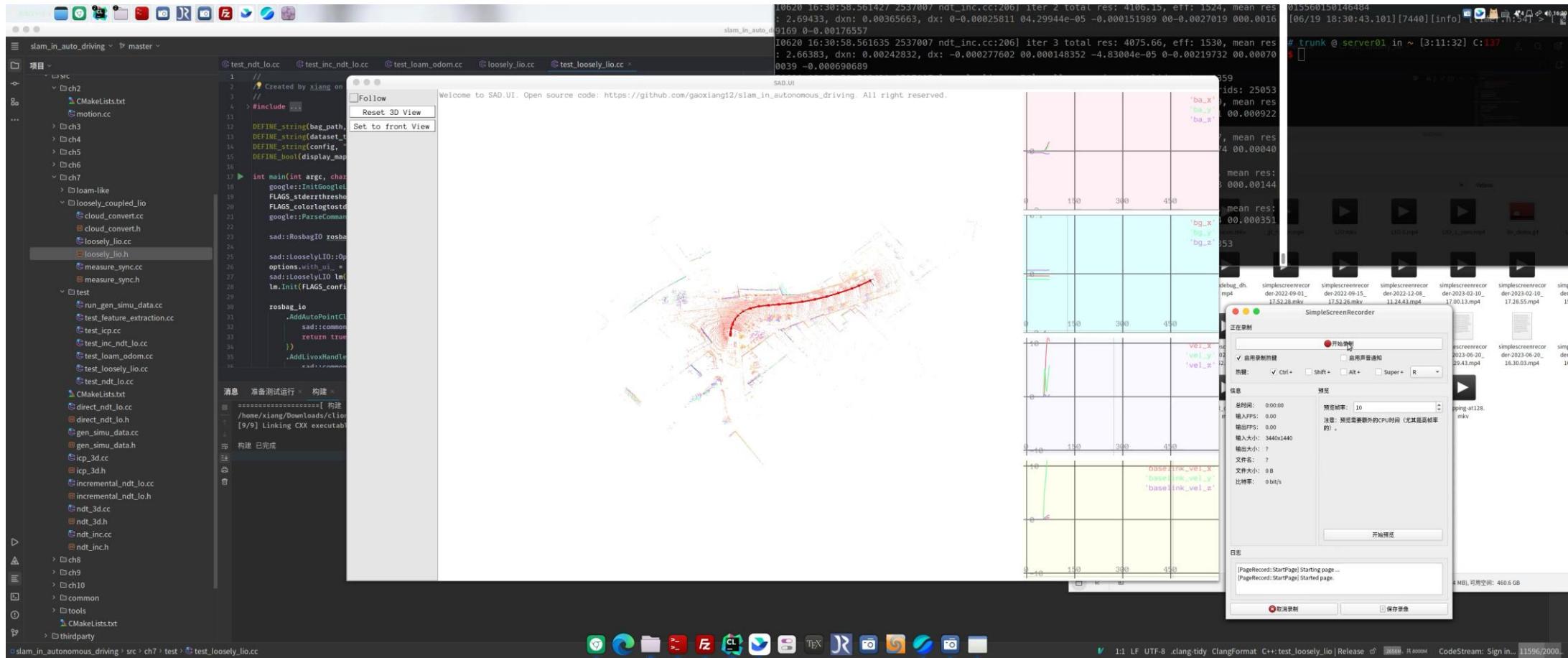
松耦合LIO及其工程问题

The screenshot shows a developer's environment with multiple windows open:

- Terminal:** Shows log output from a build process, specifically related to NDT-LO demo. It includes timestamped messages like "I0620 16:30:10.592562 2536844 loosely_lo.lo.cc:54] call meas, imu: 10, lidar pts: 3481" and error messages such as "Failed to find match for field 'time'".
- Code Editor:** Displays the source code for `test_loosely_lo.cc`. The code implements a main function that initializes Google logging, sets color threshold, parses command-line flags, and handles ROS bag input. It also includes logic for processing sensor messages and calling back functions.
- File Browser:** Shows the project structure under `slam_in_autonomous_driving`, including sub-directories like `src`, `ch2`, `ch3`, etc., and various C++ files and header files.
- Screen Recording Application:** A window titled "SimpleScreenRecorder" is active, showing a preview of the screen being recorded. It includes controls for starting and stopping the recording, and settings for frame rate, audio capture, and keyboard/mouse capture.
- System Tray:** Shows various system icons, including a battery icon, network status, and system notifications.
- Bottom Status Bar:** Provides system information such as CPU load (1.1), memory usage (1.6 GB / 460.6 GB), and a code stream sign-in status.



松耦合LIO及其工程问题





松耦合LIO及其工程问题

□ 小结

- 本章实现3D点云配准中的几种重要方法：ICP、PL-ICP、NDT
- 利用它们构建了LO
- 改进了增量NDT LO的表现
- 提特征的Loam-Like LO
- 在增量NDT LO基础上加入IMU的LooselyLIO

- 下章介绍紧耦合LIO系统



习题

1. 为三类ICP和NDT配准添加一个Cauchy核的robust loss，讲述原理并代码实现。
2. 将第1题的robust loss引入IncNDTLO和LooselyLIO，给出实现和运行效果。
3. 从概率层面解释NDT残差和协方差矩阵的关系，说明为什么NDT协方差矩阵可以用于最小二乘。
4. *为LOAM like LO设计一个地面点云提取流程，并单独为这些点使用点面ICP。

感谢聆听！
Thanks for Listening

