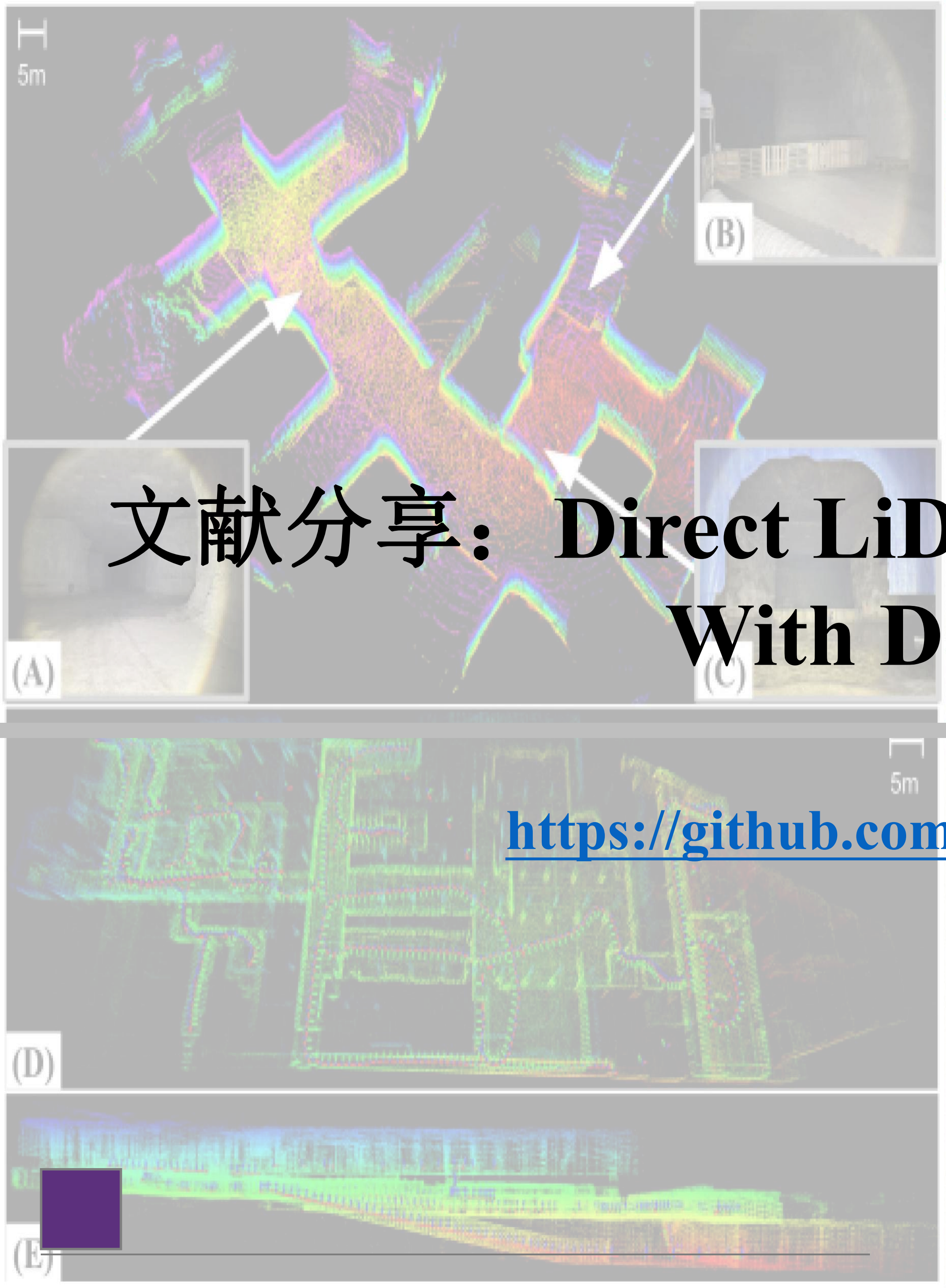
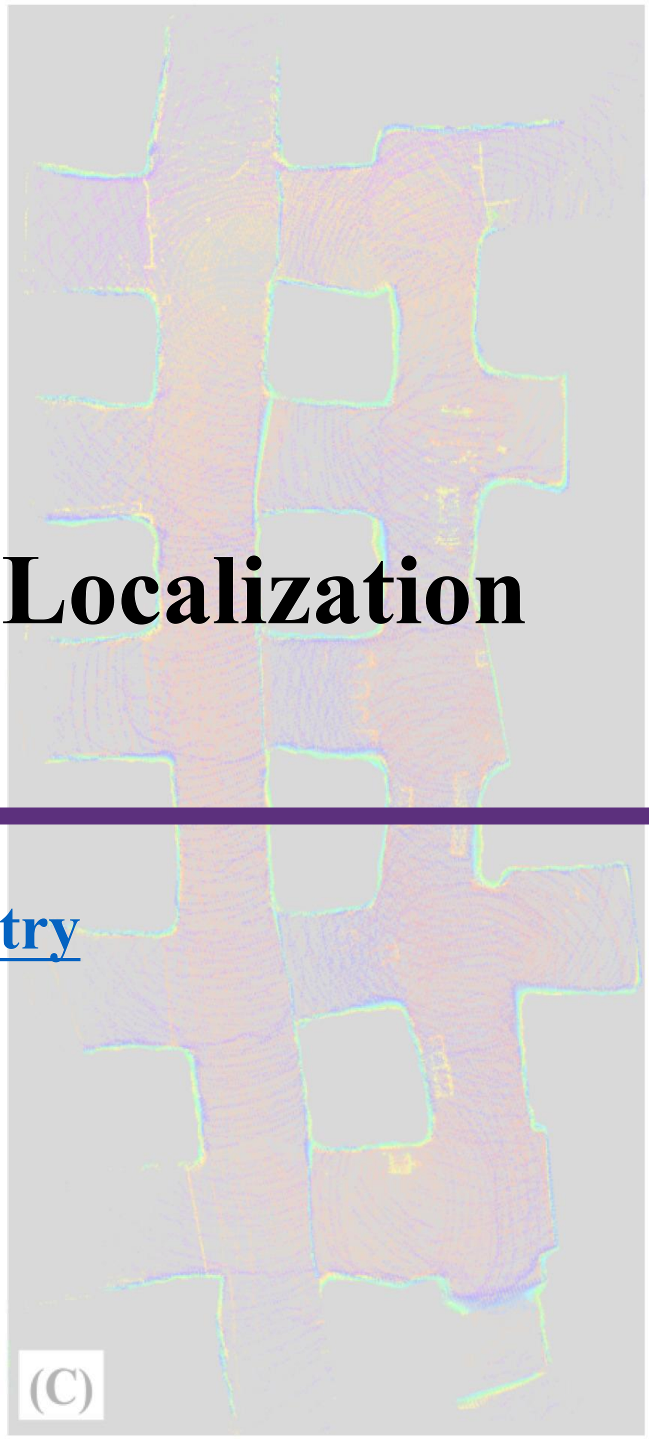
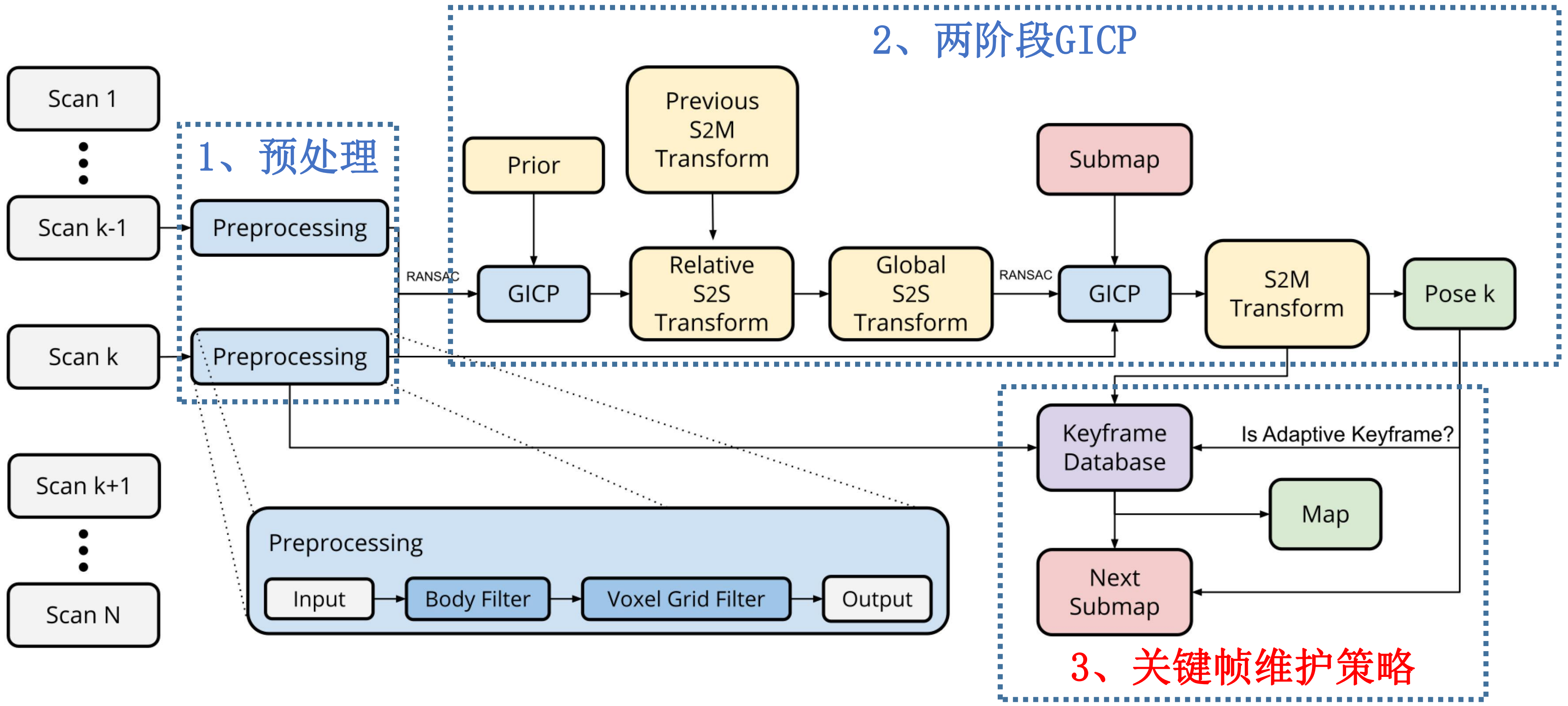




文献分享：Direct LiDAR Odometry Fast Localization With Dense Point Clouds

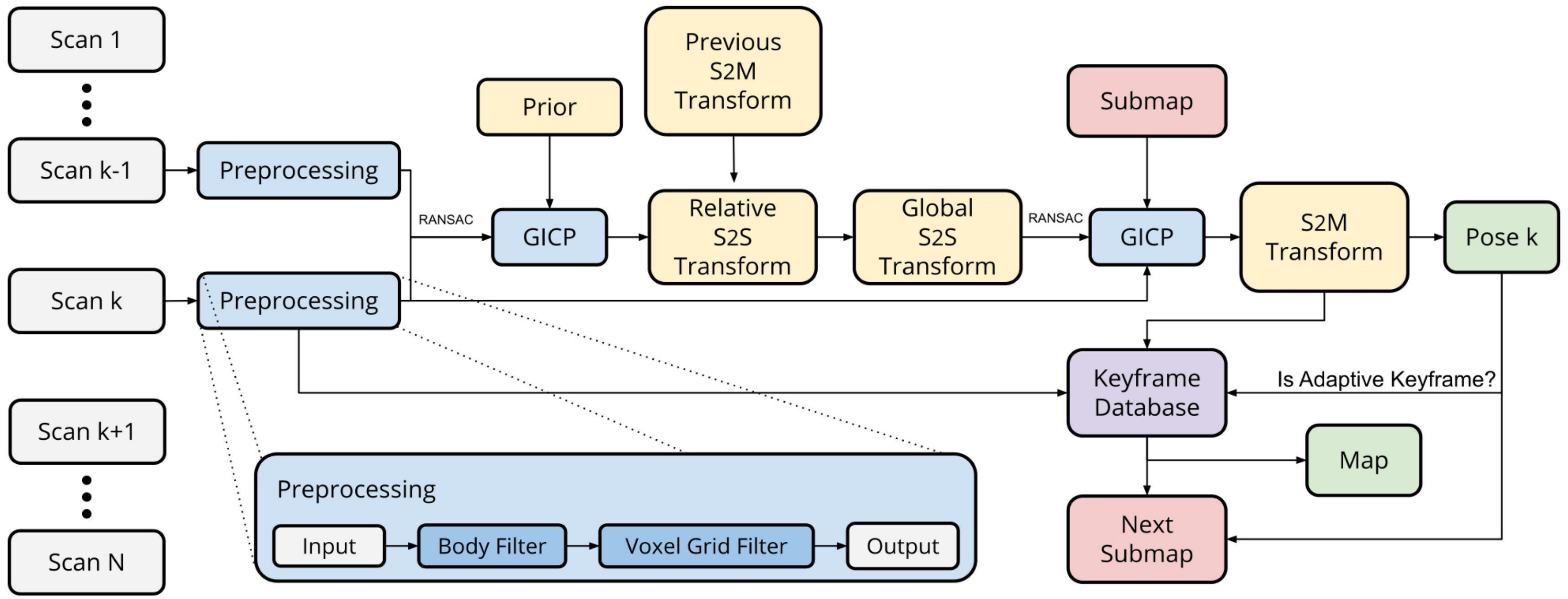
https://github.com/vectr-ucla/direct_lidar_odometry

Method



A. Notation
B. Preprocessing
C. Scan Matching Via Generalized-ICP
D. Optimization Prior
E. Fast Keyframe-Based Submapping
F. Algorithmic Implementation

C. Scan Matching Via Generalized-ICP



- 点云配准算法是**GICP**，点云匹配方面没有任何改进
- D部分是imu预积分项，无创新

1) Scan-to-Scan: In the first stage, the scan-to-scan matching objective is to compute a relative transform $\hat{\mathbf{X}}_k^{\mathcal{L}}$ between a source $\mathcal{P}_k^{\mathrm{s}}$ and a target $\mathcal{P}_k^{\mathrm{t}}$ (where $\mathcal{P}_k^{\mathrm{t}} = \mathcal{P}_{k-1}^{\mathrm{s}}$) captured in $\mathcal{L}$ where

$$\hat{\mathbf{X}}_k^{\mathcal{L}} = \underset{\mathbf{X}_k^{\mathcal{L}}}{\arg\min}\ \mathcal{E}\left(\mathbf{X}_k^{\mathcal{L}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{P}_k^{\mathrm{t}}\right) . \qquad (1)$$

The residual error $\mathcal{E}$ from GICP is defined as

$$\mathcal{E}\left(\mathbf{X}_k^{\mathcal{L}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{P}_k^{\mathrm{t}}\right) = \sum_i^N d_i^\top \left(\mathcal{C}_{k,i}^{\mathrm{t}} + \mathbf{X}_k^{\mathcal{L}}\mathcal{C}_{k,i}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{L}^\top}\right)^{-1} d_i \,, \qquad (2)$$

such that the overall objective for this stage is

$$\hat{\mathbf{X}}_k^{\mathcal{L}} = \underset{\mathbf{X}_k^{\mathcal{L}}}{\arg\min} \sum_i^N d_i^\top \left(\mathcal{C}_{k,i}^{\mathrm{t}} + \mathbf{X}_k^{\mathcal{L}}\mathcal{C}_{k,i}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{L}^\top}\right)^{-1} d_i \,, \qquad (3)$$

for N number of corresponding points between point clouds $\mathcal{P}_k^{\mathrm{s}}$ and $\mathcal{P}_k^{\mathrm{t}}$, where $d_i = p_i^{\mathrm{t}} - \mathbf{X}_k^{\mathcal{L}} p_i^{\mathrm{s}}$, $p_i^{\mathrm{s}} \in \mathcal{P}_k^{\mathrm{s}}, p_i^{\mathrm{t}} \in \mathcal{P}_k^{\mathrm{t}}, \forall i$, and $\mathcal{C}_{k,i}^{\mathrm{s}}$ and $\mathcal{C}_{k,i}^{\mathrm{t}}$ are the corresponding estimated covariance matrices

2) Scan-to-Map: After recovering an initial robot motion estimate, a secondary stage of scan-to-map matching is performed and follows a similar procedure to that of scan-to-scan. However, rather than computing a relative transform between two instantaneous point clouds, the objective here is to further refine the motion estimate from the previous step to be more globally-consistent by means of matching with a local submap. In other words, the task here is to compute an optimal transform $\hat{\mathbf{X}}_k^{\mathcal{W}}$ between the current source cloud $\mathcal{P}_k^{\mathrm{s}}$ and some derived submap $\mathcal{S}_k$ such that

$$\hat{\mathbf{X}}_k^{\mathcal{W}} = \underset{\mathbf{X}_k^{\mathcal{W}}}{\arg\min}\ \mathcal{E}\left(\mathbf{X}_k^{\mathcal{W}}\mathcal{P}_k^{\mathrm{s}}, \mathcal{S}_k\right) . \qquad (4)$$

After similarly defining the residual error $\mathcal{E}$ from GICP as in (2), the overall objective function for scan-to-map is

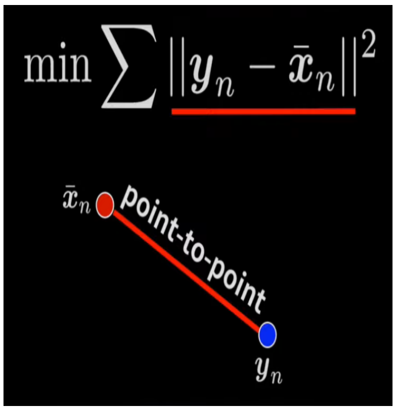
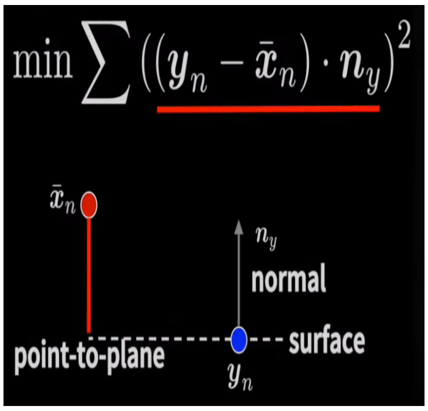
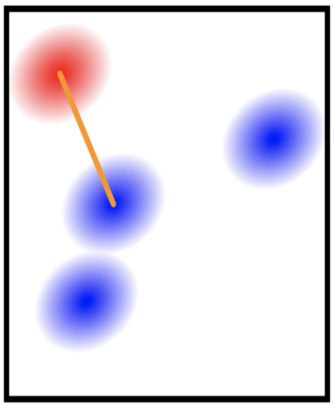
$$\hat{\mathbf{X}}_k^{\mathcal{W}} = \underset{\mathbf{X}_k^{\mathcal{W}}}{\arg\min} \sum_j^M d_j^\top \left(\mathcal{C}_{k,j}^{\mathcal{S}} + \mathbf{X}_k^{\mathcal{W}}\mathcal{C}_{k,j}^{\mathrm{s}}\mathbf{X}_k^{\mathcal{W}^\top}\right)^{-1} d_j \,, \qquad (5)$$

D. *Optimization Prior*

Eq. (3) describes the scan-to-scan nonlinear optimization problem and can be initialized with a prior to reduce the chances of converging into a sub-optimal local minima. This prior represents an initial guess of the relative motion between two LiDAR frames and can come from integrating angular velocity measurements from an inertial measurement unit (IMU). More specifically, angular velocity measurements $\hat{\omega}_k$ is defined as $\hat{\omega}_k = \omega_k + \mathbf{b}_k^\omega + \mathbf{n}_k^\omega$ measured in $\mathcal{B}$ with static bias $\mathbf{b}_k^\omega$ and zero white noise $\mathbf{n}_k^\omega$ for convenience. After calibrating for the bias, a relative rotational motion of the robot's body between

GICP 算法（ICP汇总）

ICP 算法介绍

POINT-TO-POINT ICP	POINT-TO-PLANE ICP	GICP
 $\min \sum \lVert y_n - \bar{x}_n \rVert^2$ $\bar{x}_n$ point-to-point y_n	 $\min \sum ((y_n - \bar{x}_n) \cdot n_y)^2$ $\bar{x}_n$ n_y normal point-to-plane surface y_n	
优化源点集与目标点集中对应点之间位置的偏差	优化源点集与目标点集中对应点平面的偏差	每个点都服从高斯分布，也就是说每对对应点不需要完美对应上， 尽可能提高对应的概率即可

GICP 原理：

假定我们找到了一对匹配点 $\boldsymbol{a}_i$ 和 $\boldsymbol{b}_i$ 的最优匹配为 $\boldsymbol{T}$，那么：

$$\hat{\boldsymbol{b}}_i = \boldsymbol{T}\hat{\boldsymbol{a}}_i \tag{1}$$

定义它们之间的距离残差为 $d_i^{(\boldsymbol{T})} = \hat{\boldsymbol{b}}_i - \boldsymbol{T}^*\hat{\boldsymbol{a}}_i$，假设 $\boldsymbol{a}_i$ 和 $\boldsymbol{b}_i$ 独立，则：

$$\begin{aligned} d_i^{(\mathbf{T})} &\sim \mathcal{N}\left(\hat{b}_i - (\mathbf{T})\hat{a}_i, C_i^B + (\mathbf{T})C_i^A(\mathbf{T})^T\right) \\ &= \mathcal{N}\left(0, C_i^B + (\mathbf{T})C_i^A(\mathbf{T})^T\right) \end{aligned} \tag{2}$$

GICP通过最大似然估计，找到置信最高的变换矩阵 $\boldsymbol{T}$：

$$\begin{aligned}\mathbf{T} &= \arg\max_{\mathbf{T}} \prod_i p\left(d_i^{(\mathbf{T})}\right) \\ &= \arg\max_{\mathbf{T}} \sum_i \log\left(p\left(d_i^{(\mathbf{T})}\right)\right) \\ &= \arg\max_{\mathbf{T}} \sum_i {d_i^{(\mathbf{T})}}^T \left(C_i^B + \mathbf{T} C_i^A \mathbf{T}^T\right)^{-1} d_i^{(\mathbf{T})}\end{aligned} \tag{3}$$

这是GICP中最核心的优化函数。$\left(C_i^B + \mathbf{T} C_i^A \mathbf{T}^T\right)^{-1}$ 是source点云与target点云对应点的局部协方差矩阵，这个项会影响对应残差在优化过程中的权重，加快收敛速度和精度。

此外:

- 点到点匹配 ➡ $C_i^B = I,\ C_i^A = 0$
- 点到面匹配 ➡ $C_i^B = (\boldsymbol{n}_i \boldsymbol{n}_i^T)^{-1},\ C_i^A = 0$
  - 其中 $\boldsymbol{n}_i \boldsymbol{n}_i^T$ 是一个正交投影矩阵 ($P = P^T, P^2 = P$) ，相当于把点与点之间的距离投影到法向量上，得到点到面的距离。
- 面到面匹配 ➡ GICP

附录

前置知识，多元高斯分布的概率密度函数：

$$\mathrm{f_x}(\mathrm{x_1}, \ldots, \mathrm{x_k}) = \frac{1}{\sqrt{(2\pi)^{\mathrm{k}} |\Sigma|}} \mathrm{e}^{-\frac{1}{2}(\mathrm{x}-\mu)^{\mathrm{T}} \Sigma^{-1} (\mathrm{x}-\mu)}, |\Sigma| \triangleq \det\Sigma \tag{4}$$

公式(3)目标函数的推导:

$$\begin{aligned}\mathrm{T} &= \arg\max_{\mathbf{T}} \prod_{\mathrm{i}} \mathrm{p}(d_{\mathrm{i}}^{(\mathbf{T})}) \\ &= \arg\max_{\mathbf{T}} \sum_{\mathrm{i}} \log(p(d_i^{(\mathbf{T})})) \\ &= \arg\max_{\mathbf{T}} \sum_{\mathrm{i}} \log(\frac{1}{\sqrt{(2\pi)^{\mathrm{k}} |\mathrm{C}_i^{\mathrm{B}} + \mathbf{T} \mathrm{C}_i^{\mathrm{A}} \mathbf{T}^{\mathrm{T}}|}}) \\ &- \frac{1}{2}(d_i^{(\mathbf{T})} - (\hat{b}_i - \mathbf{T}\hat{a}_i))^T (\mathrm{C}_{\mathrm{i}}^{\mathrm{B}} + \mathbf{T} \mathrm{C}_{\mathrm{i}}^{\mathrm{A}} \mathbf{T}^T)^{-1} (d_i^{(T)} - (\hat{b}_i - \mathbf{T}\hat{a}_i)) \\ &= \arg\max_{\mathbf{T}} \sum_{\mathrm{i}} \log(\frac{1}{\sqrt{(2\pi)^{\mathrm{k}} |\mathrm{C}_i^{\mathrm{B}} + \mathbf{T} \mathrm{C}_i^{\mathrm{A}} \mathbf{T}^{\mathrm{T}}|}}) \\ &- \frac{1}{2} d_i^{(\mathbf{T})^T} (\mathrm{C}_{\mathrm{i}}^{\mathrm{B}} + \mathbf{T} \mathrm{C}_{\mathrm{i}}^{\mathrm{A}} \mathbf{T}^T)^{-1} d_i^{(\mathbf{T})} \\ &= \arg\max_{\mathbf{T}} \sum_{\mathrm{i}} -\frac{1}{2} {d_i^{(\mathbf{T})}}^T (\mathrm{C}_{\mathrm{i}}^{\mathrm{B}} + \mathbf{T} \mathrm{C}_{\mathrm{i}}^{\mathrm{A}} \mathbf{T}^T)^{-1} d_i^{(\mathbf{T})} \\ &= \arg\min_{\mathbf{T}} \sum_{\mathrm{i}} {d_i^{(\mathbf{T})}}^{\mathrm{T}} (\mathrm{C}_{\mathrm{i}}^{\mathrm{B}} + \mathbf{T} \mathrm{C}_{\mathrm{i}}^{\mathrm{A}} \mathbf{T}^T)^{-1} d_i^{(\mathbf{T})}\end{aligned} \tag{5}$$

优化推导

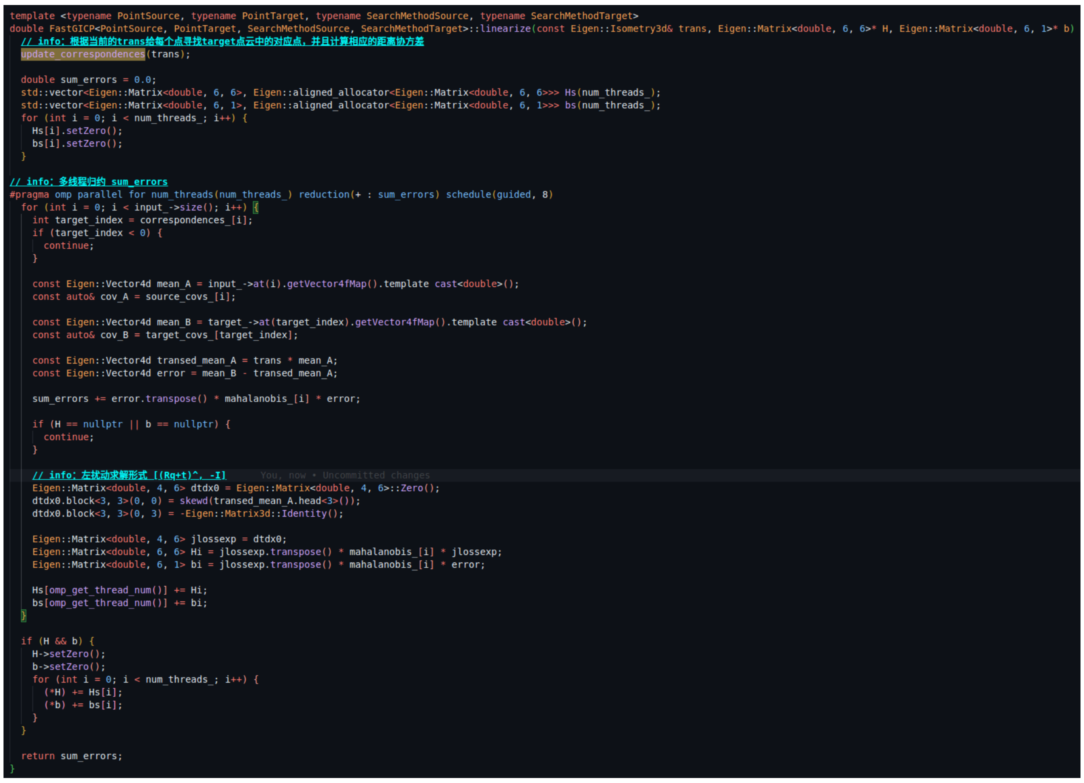
- 为了方便求导和加速运算，作者代码中将协方差权重部分 $(\mathrm{C}_i^B + \mathbf{T}\mathrm{C}_i^A\mathbf{T}^T)^{-1}$ 当作一个常数矩阵。

- 求导部分使用的是左扰动形式，并且是用的是 $SE(3)$ 进行推导：

$$\begin{aligned}
\frac{\partial(\mathbf{Tp})}{\partial\delta\boldsymbol{\xi}} &= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}} \frac{\exp(\delta\boldsymbol{\xi}^{\wedge})\exp(\boldsymbol{\xi}^{\wedge})\mathbf{p} - \exp(\boldsymbol{\xi}^{\wedge})\mathbf{p}}{\delta\boldsymbol{\xi}} \\
&= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}} \frac{(\mathbf{I}+\delta\boldsymbol{\xi}^{\wedge})\exp(\boldsymbol{\xi}^{\wedge})\mathbf{p} - \exp(\boldsymbol{\xi}^{\wedge})\mathbf{p}}{\delta\boldsymbol{\xi}} \\
&= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}} \frac{\delta\boldsymbol{\xi}^{\wedge}\exp(\boldsymbol{\xi}^{\wedge})\mathbf{p}}{\delta\boldsymbol{\xi}} \\
&= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}} \frac{\begin{bmatrix}\delta\boldsymbol{\phi}^{\wedge} & \delta\boldsymbol{\rho} \\ \mathbf{0}^T & 0\end{bmatrix}\begin{bmatrix}\mathbf{Rp}+\mathbf{t} \\ 1\end{bmatrix}}{\delta\boldsymbol{\xi}} \\
&= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}} \frac{\begin{bmatrix}\delta\boldsymbol{\phi}^{\wedge}(\mathbf{Rp}+\mathbf{t}) + \delta\boldsymbol{\rho} \\ \mathbf{0}^T\end{bmatrix}}{[\delta\boldsymbol{\rho}, \delta\boldsymbol{\phi}]^T} = \begin{bmatrix}\mathbf{I} & -(\mathbf{Rp}+\mathbf{t})^{\wedge} \\ \mathbf{0}^T & \mathbf{0}^T\end{bmatrix} \triangleq (\mathbf{Tp})^{\odot}
\end{aligned} \tag{6}$$

Gaussian-Newton 优化

- 代码实现 **FastGICP::linearize** 函数：通过多线程对H矩阵和b矩阵进行计算　此外，这是个虚函数，每个子类需要自己进行实现



```cpp
template <typename PointSource, typename PointTarget, typename SearchMethodSource, typename SearchMethodTarget>
double FastGICP<PointSource, PointTarget, SearchMethodSource, SearchMethodTarget>::linearize(const Eigen::Isometry3d& trans, Eigen::Matrix<double, 6, 6>* H, Eigen::Matrix<double, 6, 1>* b)
  // info: 根据当前的trans给每个点寻找target点云中的对应点，并且计算相应的距离协方差
  update_correspondences(trans);

  double sum_errors = 0.0;
  std::vector<Eigen::Matrix<double, 6, 6>, Eigen::aligned_allocator<Eigen::Matrix<double, 6, 6>>> Hs(num_threads_);
  std::vector<Eigen::Matrix<double, 6, 1>, Eigen::aligned_allocator<Eigen::Matrix<double, 6, 1>>> bs(num_threads_);
  for (int i = 0; i < num_threads_; i++) {
    Hs[i].setZero();
    bs[i].setZero();
  }

// info: 多线程归约 sum_errors
#pragma omp parallel for num_threads(num_threads_) reduction(+ : sum_errors) schedule(guided, 8)
  for (int i = 0; i < input_->size(); i++) {
    int target_index = correspondences_[i];
    if (target_index < 0) {
      continue;
    }

    const Eigen::Vector4d mean_A = input_->at(i).getVector4fMap().template cast<double>();
    const auto& cov_A = source_covs_[i];

    const Eigen::Vector4d mean_B = target_->at(target_index).getVector4fMap().template cast<double>();
    const auto& cov_B = target_covs_[target_index];

    const Eigen::Vector4d transed_mean_A = trans * mean_A;
    const Eigen::Vector4d error = mean_B - transed_mean_A;

    sum_errors += error.transpose() * mahalanobis_[i] * error;

    if (H == nullptr || b == nullptr) {
      continue;
    }

    // info: 左扰动求解形式 [(Rq+t)^, -I]        You, now • Uncommitted changes
    Eigen::Matrix<double, 4, 6> dtdx0 = Eigen::Matrix<double, 4, 6>::Zero();
    dtdx0.block<3, 3>(0, 0) = skewd(transed_mean_A.head<3>());
    dtdx0.block<3, 3>(0, 3) = -Eigen::Matrix3d::Identity();

    Eigen::Matrix<double, 4, 6> jlossexp = dtdx0;
    Eigen::Matrix<double, 6, 6> Hi = jlossexp.transpose() * mahalanobis_[i] * jlossexp;
    Eigen::Matrix<double, 6, 1> bi = jlossexp.transpose() * mahalanobis_[i] * error;

    Hs[omp_get_thread_num()] += Hi;
    bs[omp_get_thread_num()] += bi;
  }

  if (H && b) {
    H->setZero();
    b->setZero();
    for (int i = 0; i < num_threads_; i++) {
      (*H) += Hs[i];
      (*b) += bs[i];
    }
  }

  return sum_errors;
}
```

E. Fast Keyframe-Based Submapping（主要创新点）

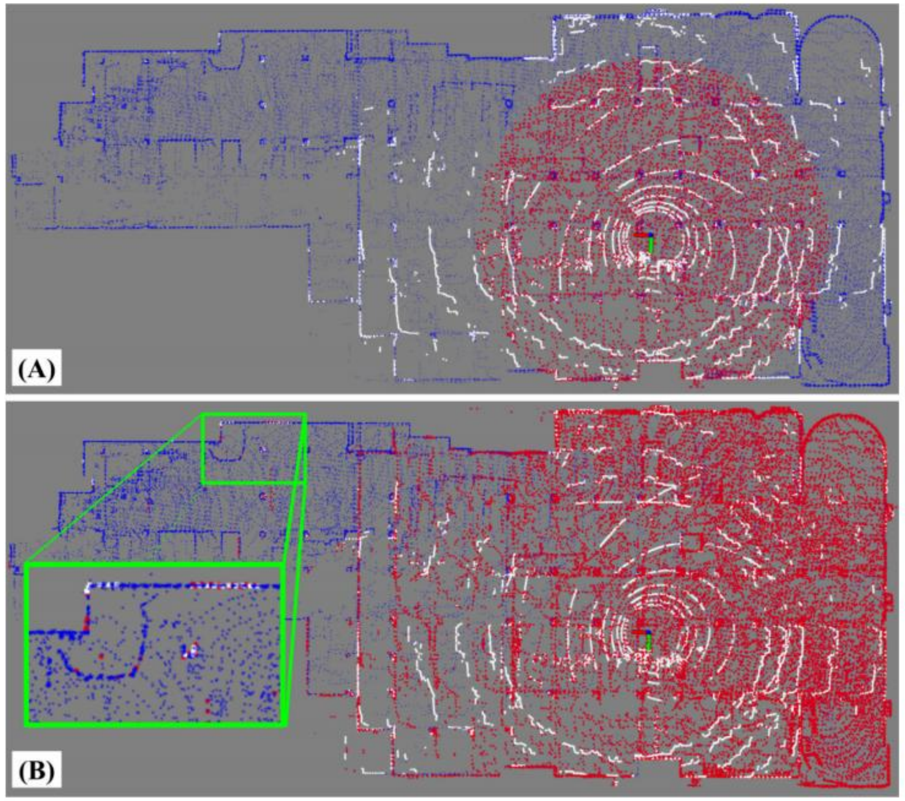


Fig. 3. **Keyframe-based submapping.** A comparison between the different submapping approaches, visualizing the current scan (white), the derived submap (red), and the full map (blue). (A) A common radius-based submapping approach of $r = 20$ m retrieved in point cloud-space. (B) Our keyframe-based submapping approach, which concatenates a subset of keyed scans and helps anchor even the most distant points in the current scan (green box) during the scan-to-map stage.

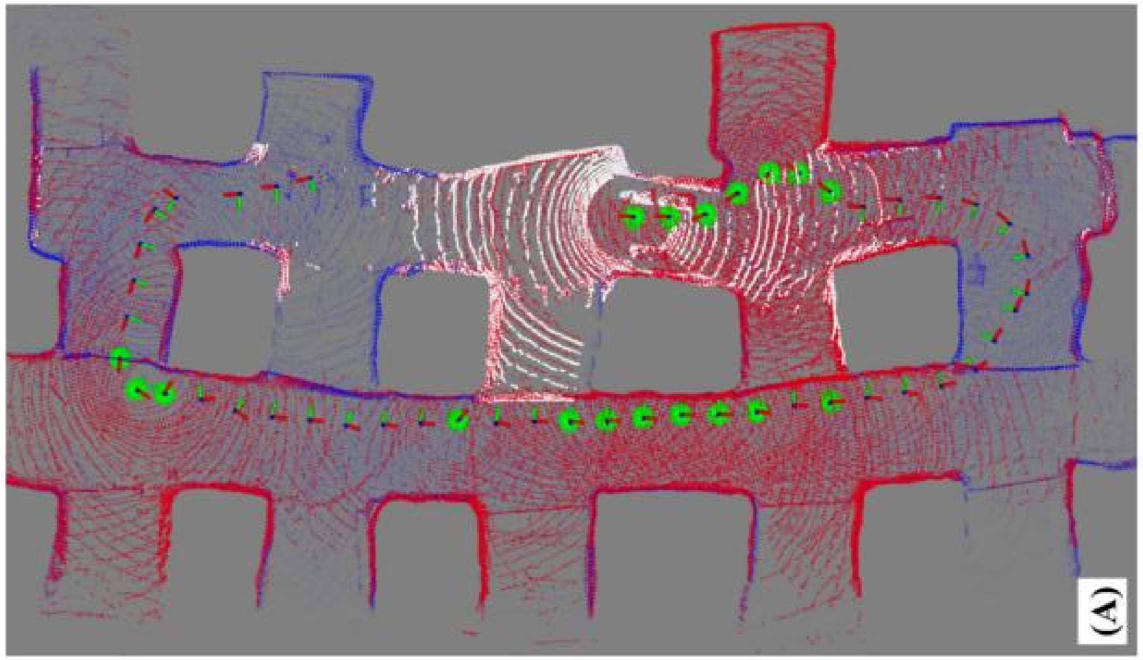
- 通过从关键帧的子集将相应的点云连接起来，生成submap，而不是直接检索机器人当前位置一定半径内的局部点云。

- 优点：
  - 计算复杂度降低；
  - 可选择性强；
  - 和current scan有更大的overlap；
  - 对于GICP来说计算协方差（用了近似）更快；

E. Fast Keyframe-Based Submapping（主要创新点）

1) Keyframe Selection Via kNN and Convex Hull: To construct the submap $\mathcal{S}_k$, we concatenate the corresponding point clouds from a selected subset of environmental keyframes. Let $\mathcal{K}_k$ be the set of all keyframe point clouds such that $\mathcal{S}_k \subseteq \mathcal{K}_k$. We define submap $\mathcal{S}_k$ as the concatenation of K nearest neighbor keyframe scans $\mathcal{Q}_k$ and L nearest neighbor convex hull scans $\mathcal{H}_k$ such that $\mathcal{S}_k = \mathcal{Q}_k \oplus \mathcal{H}_k$, where the indices which specify the convex hull are defined by the set of keyframes which make up the intersection of all convex sets containing the keyframes which compose $\mathcal{K}_k$.

- k近邻
- pcl convex hull
  - contain boundary map points
  - increase the overlap with more distant points in the scan



E. Fast Keyframe-Based Submapping（主要创新点）

2) *Adaptive Keyframing:* The location of keyframes affects the derived submap and can subsequently influence accuracy and robustness of the odometry. Keyframe nodes are commonly dropped using fixed thresholds (e.g., every 1 m or 10° of translational or rotational change) [4], [6], [13], but the optimal position can be highly dependent on a surrounding environment's structure. More specifically, in large-scale settings, features captured by the point cloud scan are much more prominent and can be depended on for longer periods of time. Conversely, for narrow or small-scale environments, a smaller threshold is necessary to continually capture the small-scale features (i.e., tight corners) in the submap for better localization. Thus, we choose to scale the translational threshold for new keyframes according to the "spaciousness" in the instantaneous point cloud scan, defined as $m_k = \alpha m_{k-1} + \beta M_k$, where M_k is the median Euclidean point distance from the origin to each point in the preprocessed point cloud, $\alpha = 0.95$, $\beta = 0.05$, and m_k is the smoothed signal used to scale the keyframe threshold th_k at time k such that

$$th_k = \begin{cases} 10\text{m} & \text{if } m_k > 20\text{m} \\ 5\text{m} & \text{if } m_k > 10\text{m} \,\&\, m_k \leq 20\text{m} \\ 1\text{m} & \text{if } m_k > 5\text{m} \,\&\, m_k \leq 10\text{m} \\ 0.5\text{m} & \text{if } m_k \leq 5\text{m} \end{cases} \tag{6}$$

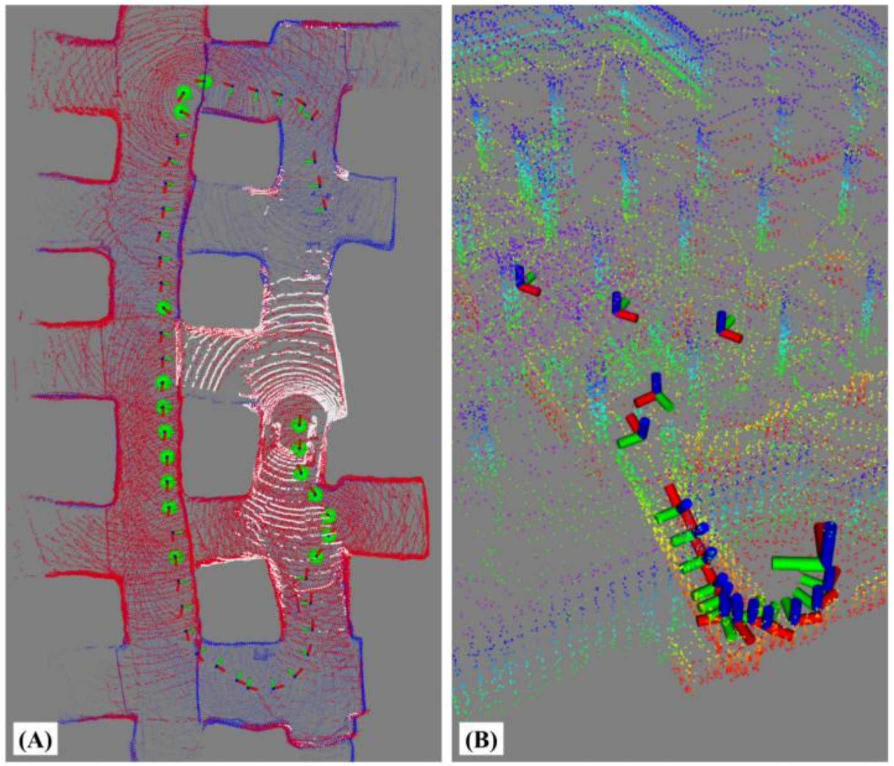
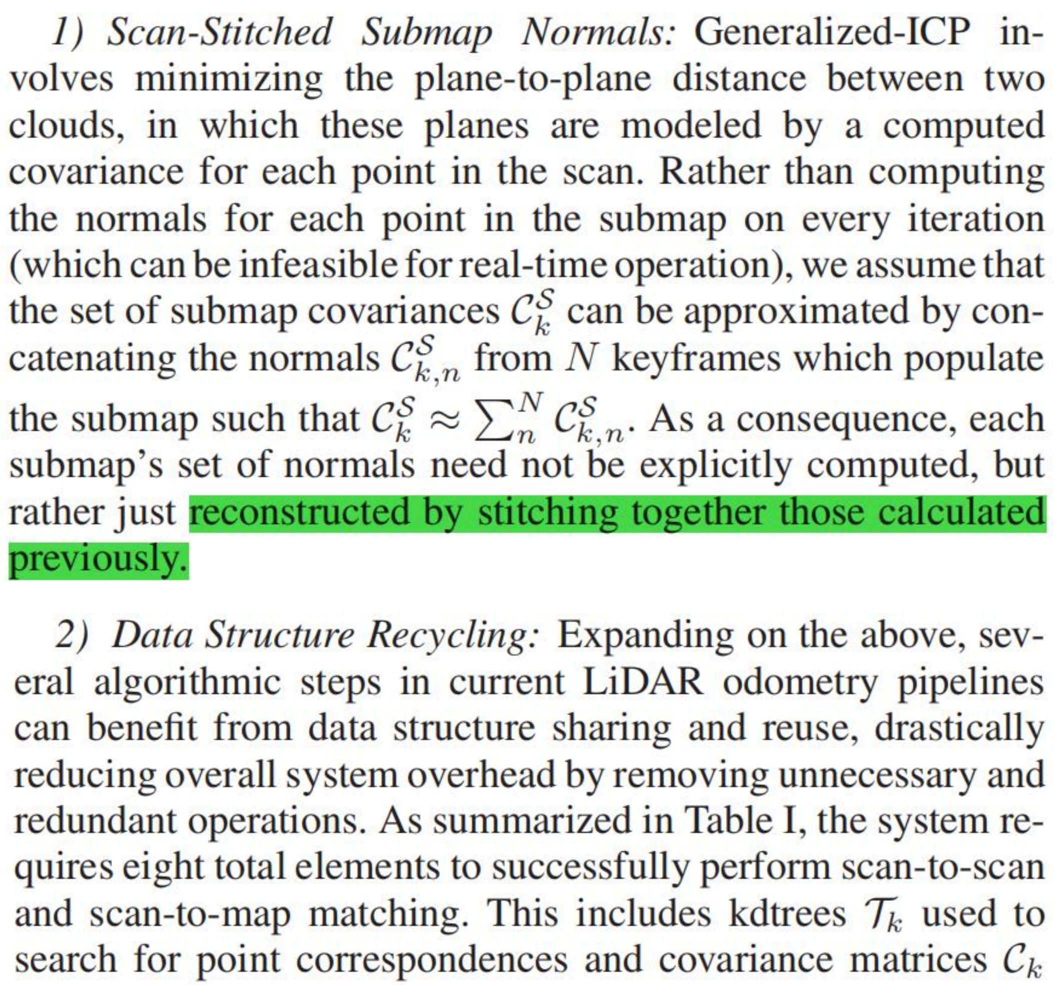


Fig. 4. **Keyframe selection and adaptive thresholds.** (A) Our method's submap (red) is generated by concatenating the scans from a subset of keyframes (green spheres), which consists of K nearest neighbor keyframes and those that construct the convex hull of the keyframe set. (B) An illustration of adaptive keyframing. In this scenario, the threshold decreases when traversing down a narrow ramp to better capture small-scale details.

- **2）自适应关键帧选取：**应对狭窄区间，保留更多的信息

F. Algorithmic Implementation



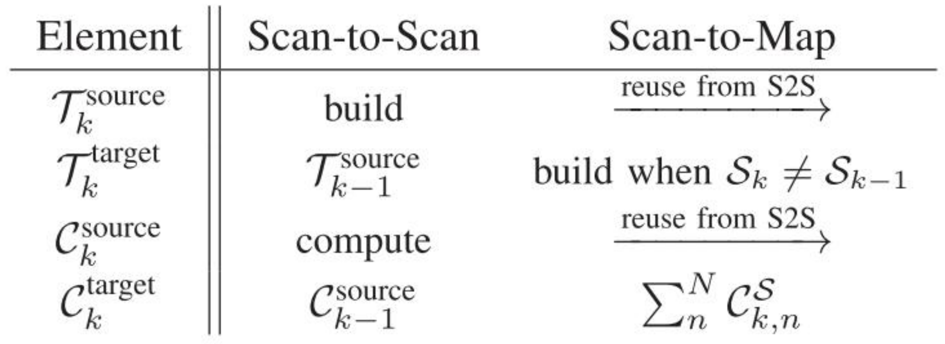
1) Scan-Stitched Submap Normals: Generalized-ICP involves minimizing the plane-to-plane distance between two clouds, in which these planes are modeled by a computed covariance for each point in the scan. Rather than computing the normals for each point in the submap on every iteration (which can be infeasible for real-time operation), we assume that the set of submap covariances $\mathcal{C}_k^S$ can be approximated by concatenating the normals $\mathcal{C}_{k,n}^S$ from N keyframes which populate the submap such that $\mathcal{C}_k^S \approx \sum_n^N \mathcal{C}_{k,n}^S$. As a consequence, each submap's set of normals need not be explicitly computed, but rather just reconstructed by stitching together those calculated previously.

2) Data Structure Recycling: Expanding on the above, several algorithmic steps in current LiDAR odometry pipelines can benefit from data structure sharing and reuse, drastically reducing overall system overhead by removing unnecessary and redundant operations. As summarized in Table I, the system requires eight total elements to successfully perform scan-to-scan and scan-to-map matching. This includes kdtrees $\mathcal{T}_k$ used to search for point correspondences and covariance matrices $\mathcal{C}_k$

- 1）Submap 法向量近似计算：submap的协方差矩阵直接由每个关键帧的协方差进行近似，节省计算时间和开销

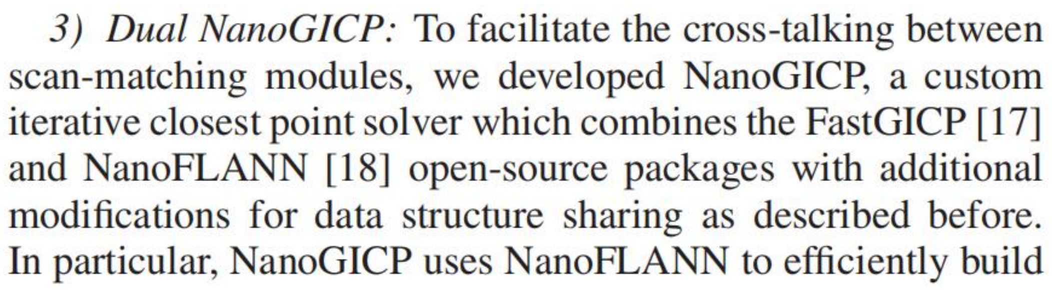
- 2）数据的重复利用：
  - 建树2次（本来需要4次）
  - 协方差计算只需要1次（本来需要4次）

TABLE I
SUMMARY OF DATA STRUCTURE RECYCLING



Element	Scan-to-Scan	Scan-to-Map
$\mathcal{T}_k^{\text{source}}$	build	$\xrightarrow{\text{reuse from S2S}}$
$\mathcal{T}_k^{\text{target}}$	$\mathcal{T}_{k-1}^{\text{source}}$	build when $\mathcal{S}_k \neq \mathcal{S}_{k-1}$
$\mathcal{C}_k^{\text{source}}$	compute	$\xrightarrow{\text{reuse from S2S}}$
$\mathcal{C}_k^{\text{target}}$	$\mathcal{C}_{k-1}^{\text{source}}$	$\sum_n^N \mathcal{C}_{k,n}^S$

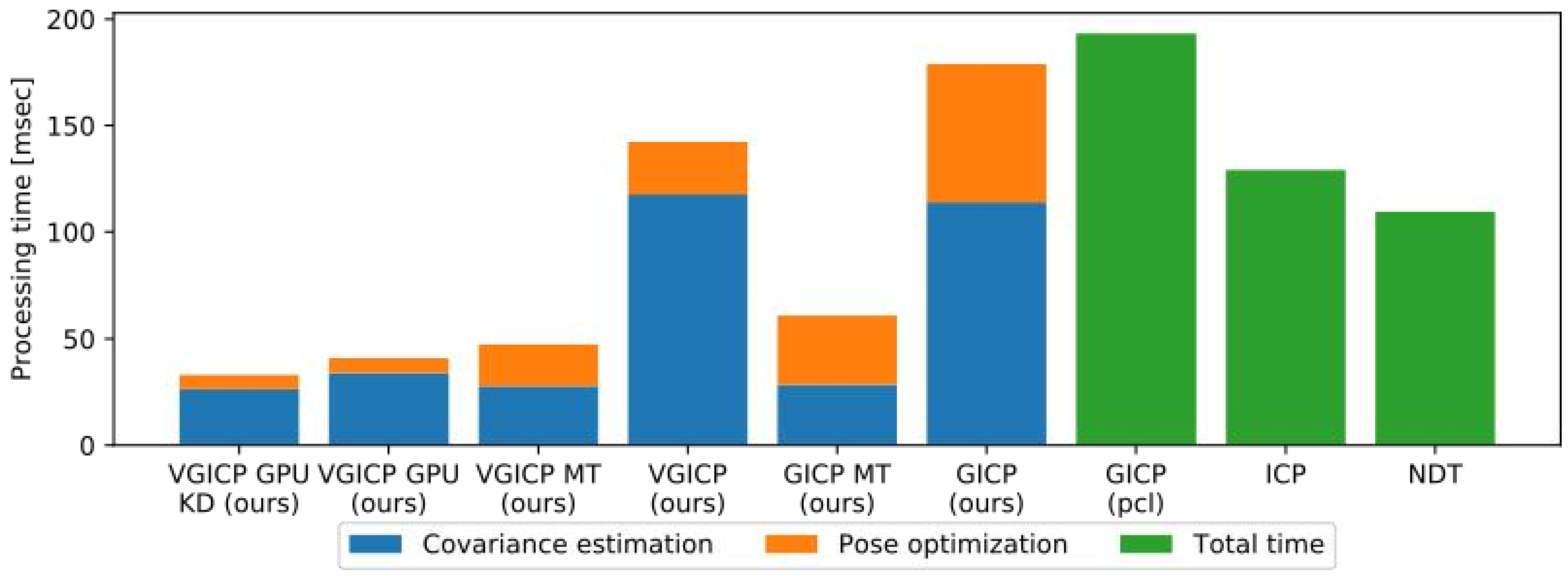
F. Algorithmic Implementation



3) Dual NanoGICP: To facilitate the cross-talking between scan-matching modules, we developed NanoGICP, a custom iterative closest point solver which combines the FastGICP [17] and NanoFLANN [18] open-source packages with additional modifications for data structure sharing as described before. In particular, NanoGICP uses NanoFLANN to efficiently build

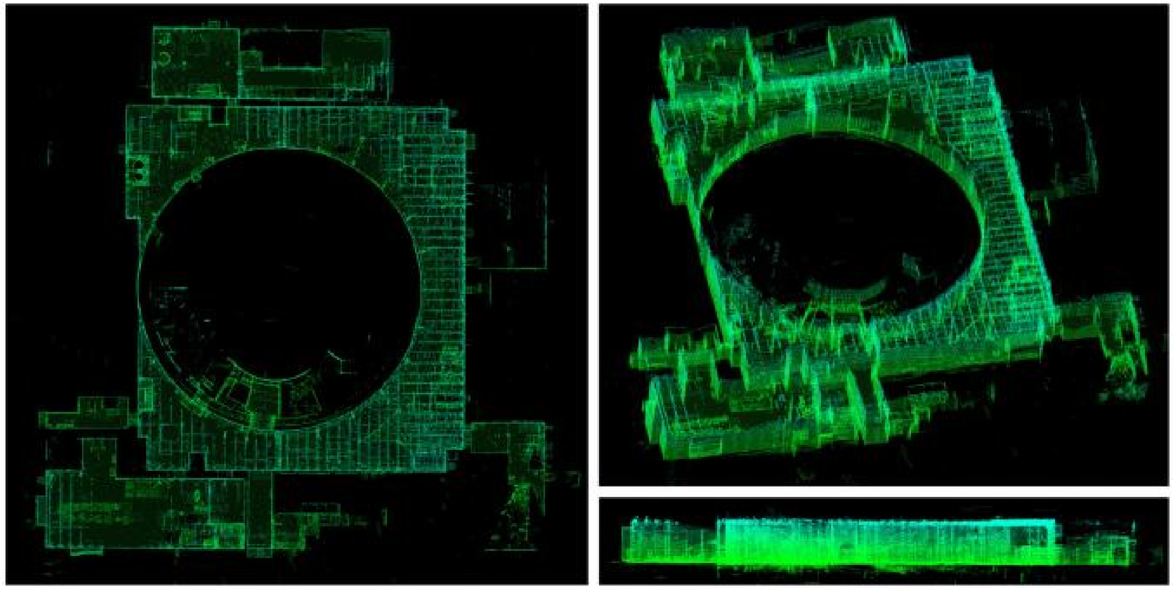
- 3）作者结合了FastGICP：https://github.com/SMRT-AIST/fast_gicp和 nanoflann：https://github.com/jlblancoc/nanoflann

  nanoflann算法对fastann进行了改进，效率以及内存使用等方面都进行了优化，而且代码十分轻量级且开源



实验

- 数据集： Alpha Course dataset fromthe Urban circuit of the DARPA Subterranean Challenge



1) Keyframe-Based Submapping: We compared the absolute pose error (APE), processing time, and CPU load across three submapping schemes, including: radius-based ($r = 10$ m),

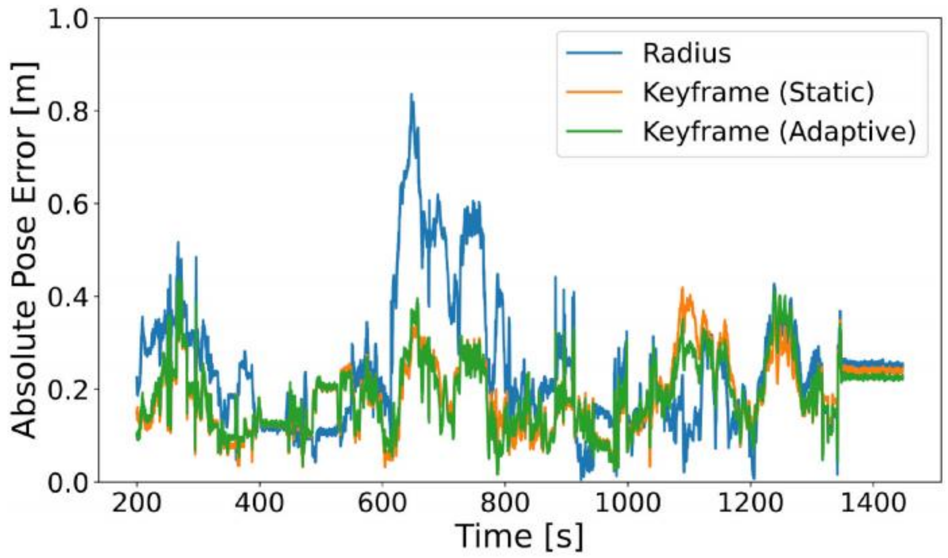


Fig. 6. **Error comparison.** The absolute pose error plotted across a 1200 s window of movement, showing the difference between radius and keyframe submapping schemes. Keyframe-based approaches do not have the range restriction that radius-based approaches inherently contain, which directly translates to a lower error in odometry due to more perceptive submapping. Note that adaptive keyframing primarily helps with reliability and robustness to changes in environmental dimension (Fig. 9).

- 这部分实验是针对submap的选取方式：和Radius的submap方式对比更快、drift更小

实验

2) *Data Structure Recycling:* To evaluate the effectiveness of data reusage, we measured and compared the processing time and CPU usage between different recycling schemes via a box plot (Fig. 8) and percentage of dropped scans over the dataset (Table II). In a naive system which explicitly calculates each kdtree and cloud covariance, computation time exceeded LiDAR rate (10 Hz for Velodyne) with a high average of 69.8 ms per scan and nearly 10% of scans dropped due to high processing time. Recycling kdtrees but not covariances provides a slight improvement in processing time and CPU percentage, while recycling covariances but not kdtrees provides a more prominent performance boost; this is reasonable since our covariance recycling scheme is more aggressive than kdtree reusage. Finally, using the full scheme as detailed in Table I significantly decreases both metrics, with an average processing time of 21.9 ms and 9.5% CPU load, which prevents any LiDAR frames from dropping .

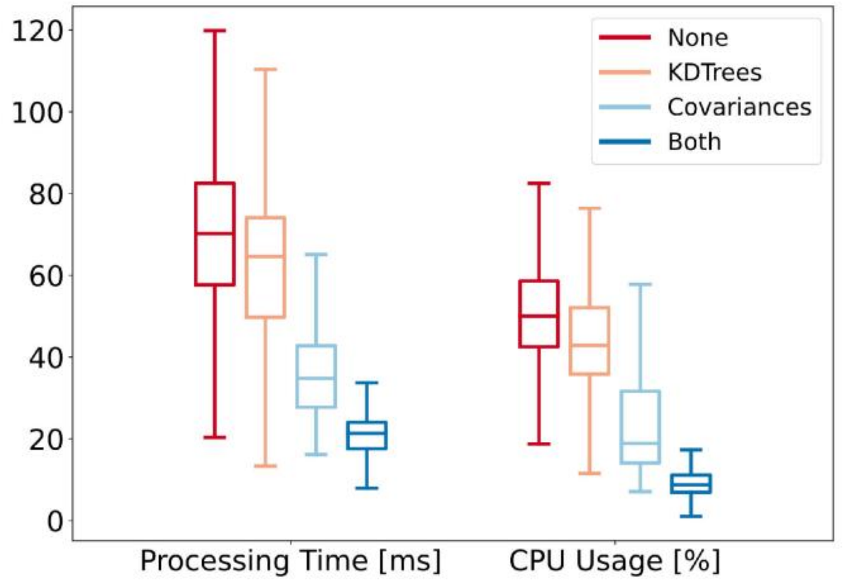


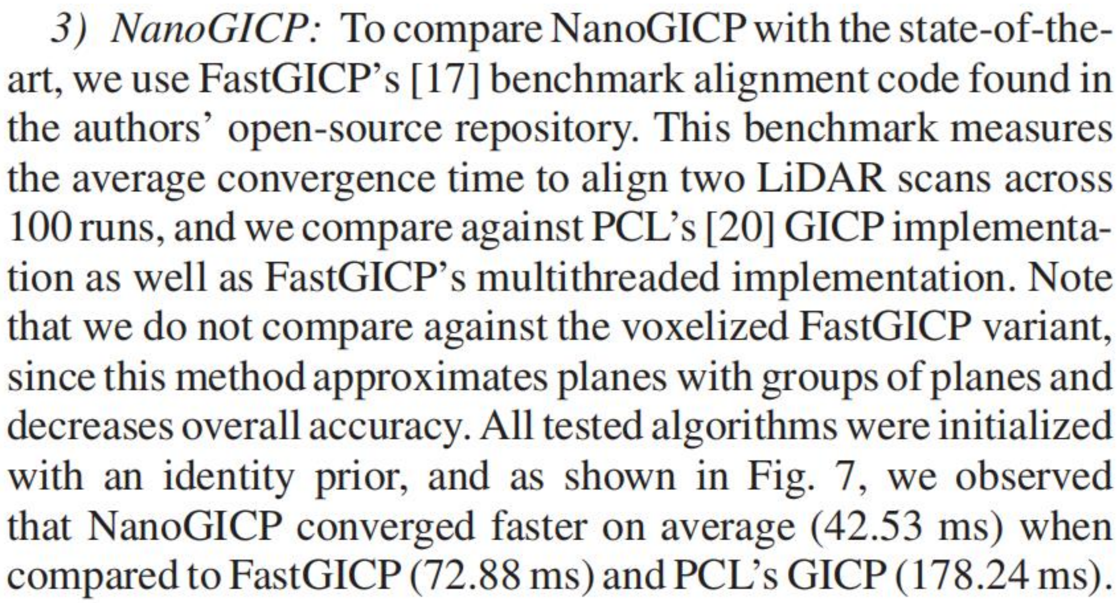
Fig. 8. **Ablation study of data recycling schemes.** Box plots of the processing time and CPU usage for four different data recycling schemes, ranging from no data structure reuse to partial reuse and full reuse.

TABLE II
DROPPED LIDAR SCANS PER RECYCLING SCHEME

	None	KDTrees	Covariances	Both
% Scans	9.37%	4.51%	0.00%	0.00%

- 这部分的实验是针对 F. *Algorithmic Implementation*小节：针对是否重新利用kdtree和协方差进行了四组实验，效果是肯定有的，速度快了很多
- **但是**，没有定位方面的对比，感觉定位可能还是会稍微下降的，毕竟做了近似。

实验



3) NanoGICP: To compare NanoGICP with the state-of-the-art, we use FastGICP's [17] benchmark alignment code found in the authors' open-source repository. This benchmark measures the average convergence time to align two LiDAR scans across 100 runs, and we compare against PCL's [20] GICP implementation as well as FastGICP's multithreaded implementation. Note that we do not compare against the voxelized FastGICP variant, since this method approximates planes with groups of planes and decreases overall accuracy. All tested algorithms were initialized with an identity prior, and as shown in Fig. 7, we observed that NanoGICP converged faster on average (42.53 ms) when compared to FastGICP (72.88 ms) and PCL's GICP (178.24 ms).

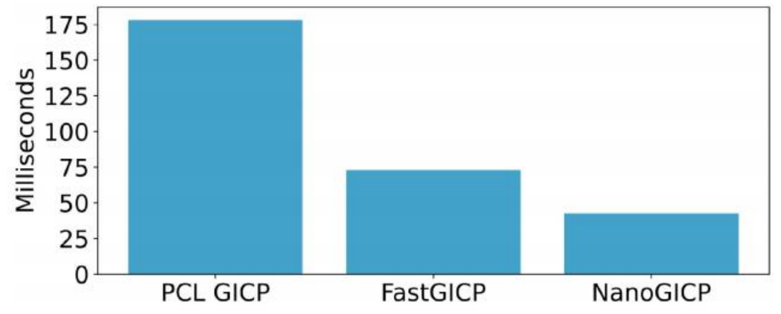
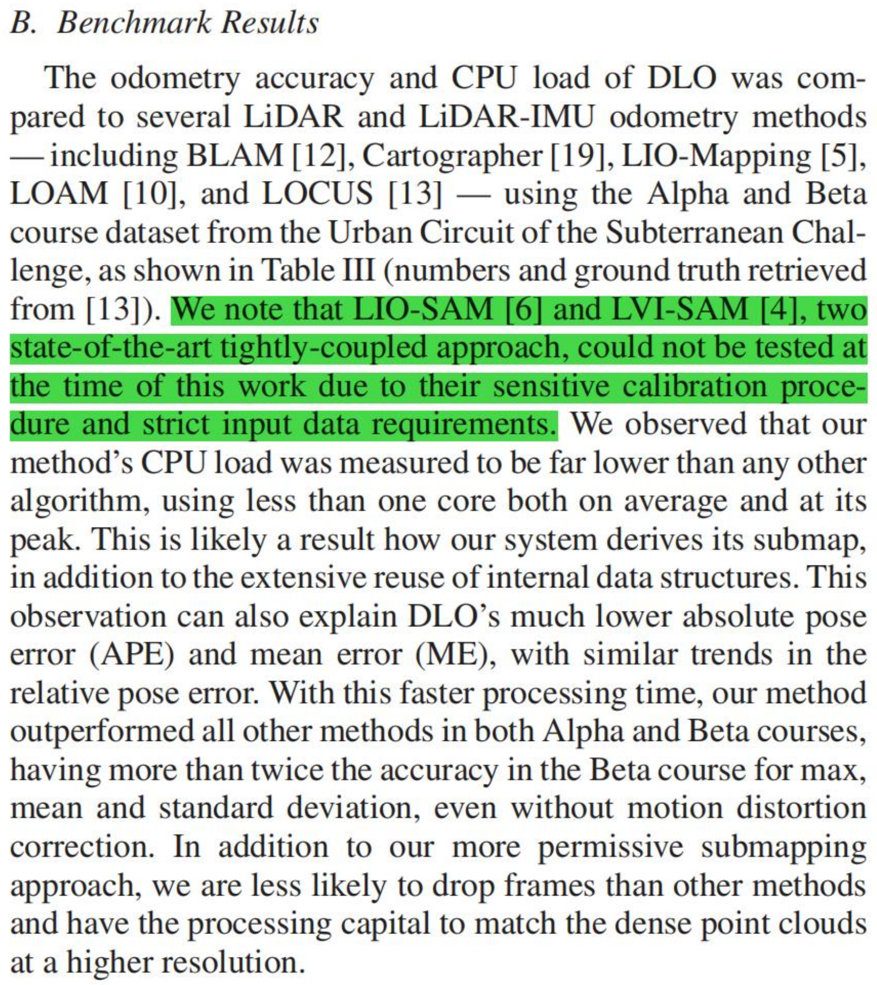


Fig. 7. **Average convergence time.** A comparison of average convergence times across 100 benchmark alignments for each algorithm, including our NanoGICP solver and two other open-source GICP packages.

- 这部分的实验是针对他们自己重构的GICP算法*NanoGICP*，和PCL、FastGICP（包括多线程版本）进行了对比，收敛速度更快。

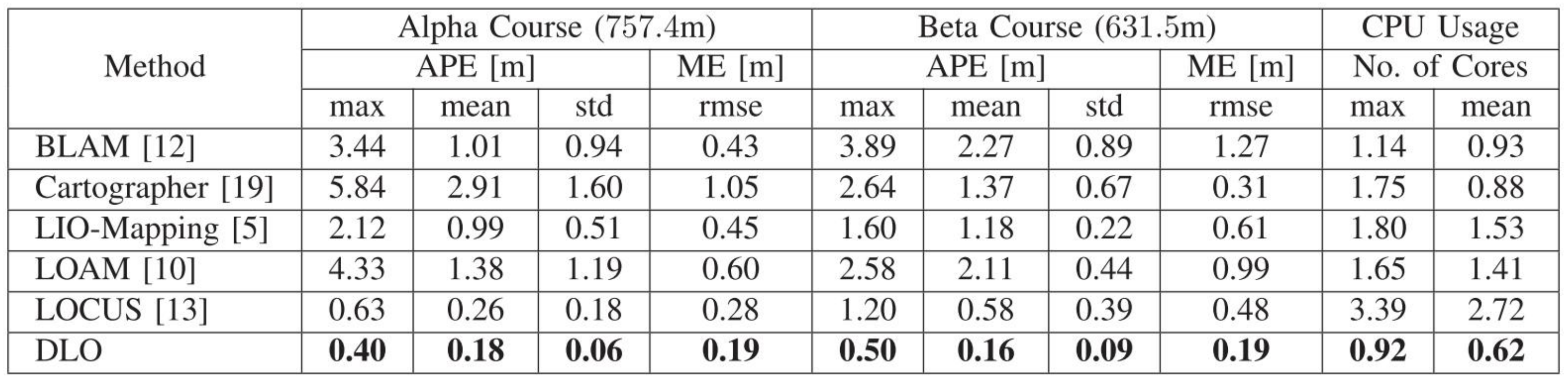
实验



B. Benchmark Results

The odometry accuracy and CPU load of DLO was compared to several LiDAR and LiDAR-IMU odometry methods — including BLAM [12], Cartographer [19], LIO-Mapping [5], LOAM [10], and LOCUS [13] — using the Alpha and Beta course dataset from the Urban Circuit of the Subterranean Challenge, as shown in Table III (numbers and ground truth retrieved from [13]). We note that LIO-SAM [6] and LVI-SAM [4], two state-of-the-art tightly-coupled approach, could not be tested at the time of this work due to their sensitive calibration procedure and strict input data requirements. We observed that our method's CPU load was measured to be far lower than any other algorithm, using less than one core both on average and at its peak. This is likely a result how our system derives its submap, in addition to the extensive reuse of internal data structures. This observation can also explain DLO's much lower absolute pose error (APE) and mean error (ME), with similar trends in the relative pose error. With this faster processing time, our method outperformed all other methods in both Alpha and Beta courses, having more than twice the accuracy in the Beta course for max, mean and standard deviation, even without motion distortion correction. In addition to our more permissive submapping approach, we are less likely to drop frames than other methods and have the processing capital to match the dense point clouds at a higher resolution.

TABLE III
COMPARISON ON BENCHMARK DATASETS

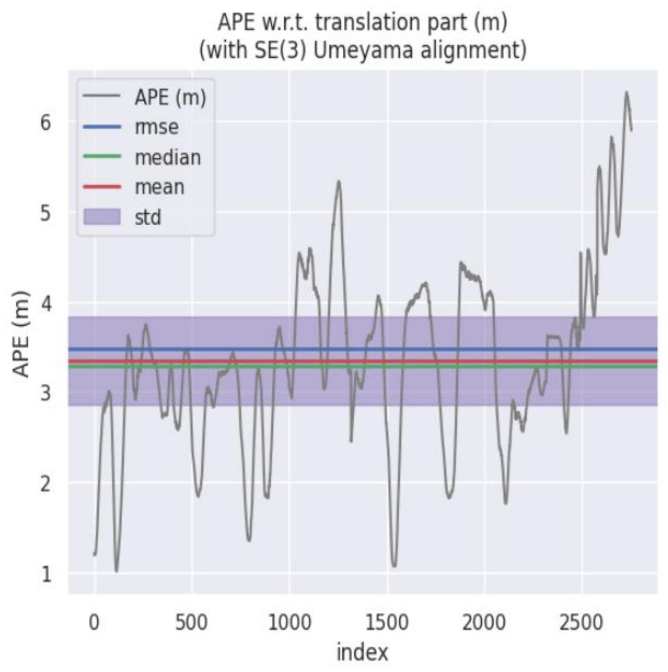
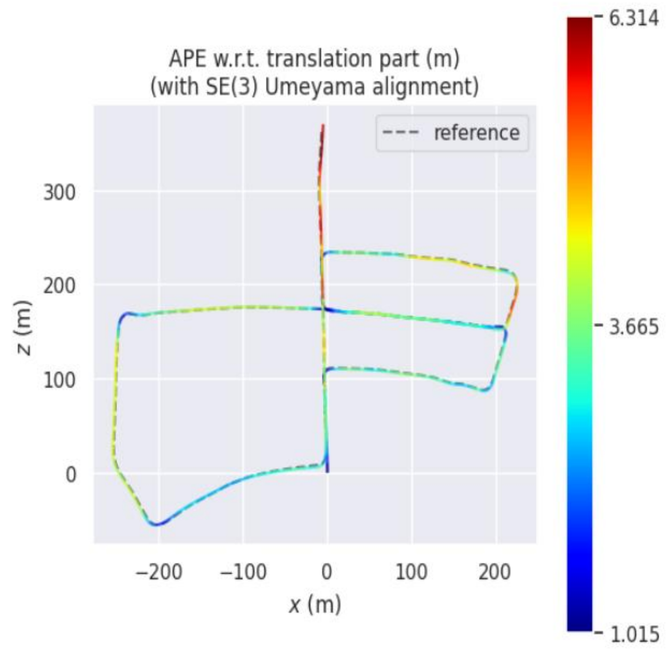
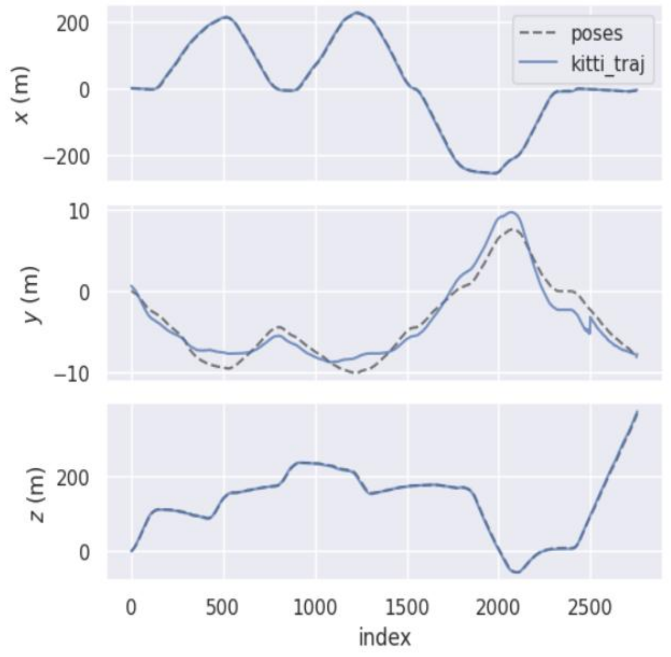


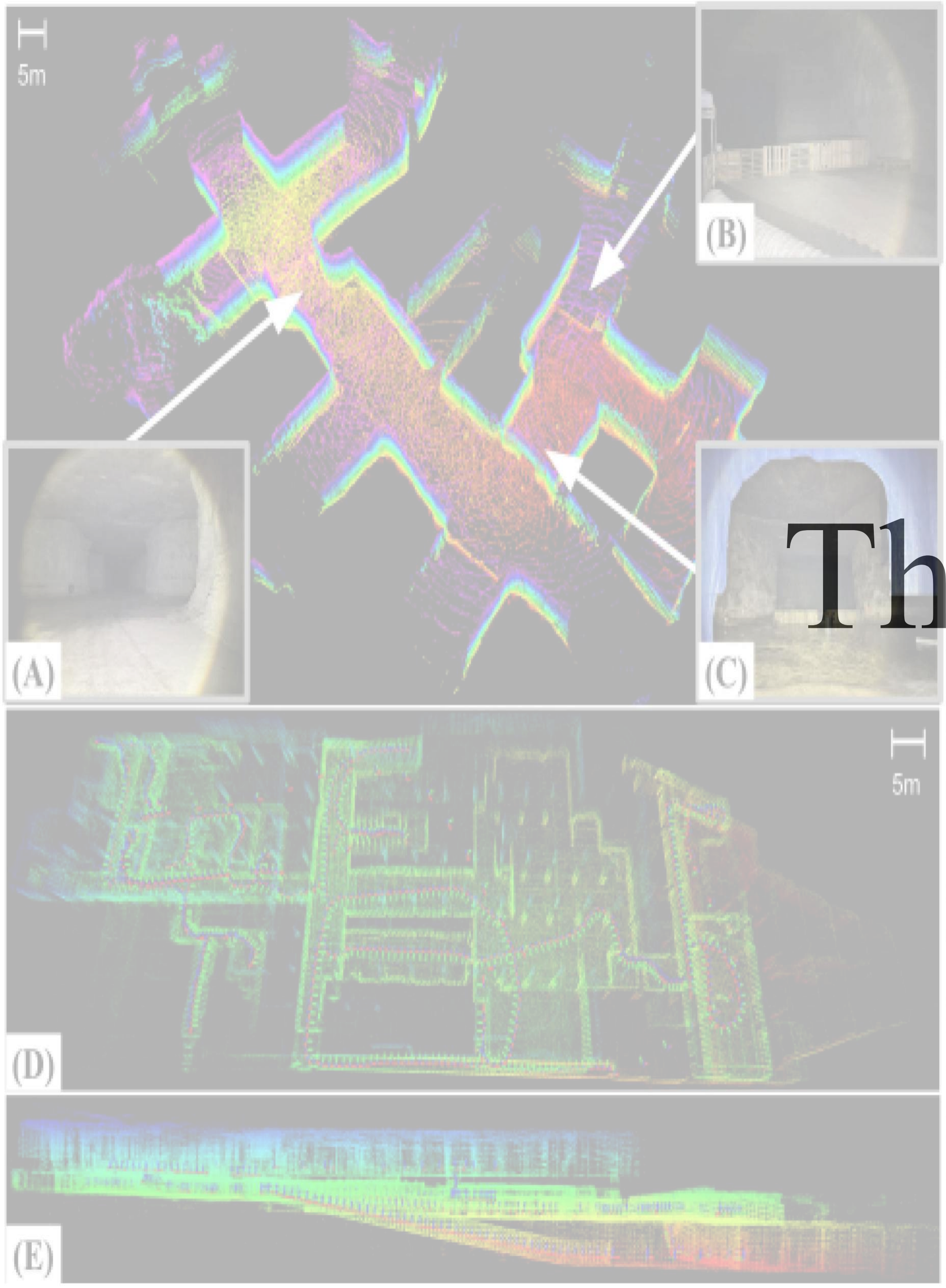
Method	Alpha Course (757.4m)				Beta Course (631.5m)				CPU Usage	
	APE [m]			ME [m]	APE [m]			ME [m]	No. of Cores	
	max	mean	std	rmse	max	mean	std	rmse	max	mean
BLAM [12]	3.44	1.01	0.94	0.43	3.89	2.27	0.89	1.27	1.14	0.93
Cartographer [19]	5.84	2.91	1.60	1.05	2.64	1.37	0.67	0.31	1.75	0.88
LIO-Mapping [5]	2.12	0.99	0.51	0.45	1.60	1.18	0.22	0.61	1.80	1.53
LOAM [10]	4.33	1.38	1.19	0.60	2.58	2.11	0.44	0.99	1.65	1.41
LOCUS [13]	0.63	0.26	0.18	0.28	1.20	0.58	0.39	0.48	3.39	2.72
DLO	**0.40**	**0.18**	**0.06**	**0.19**	**0.50**	**0.16**	**0.09**	**0.19**	**0.92**	**0.62**

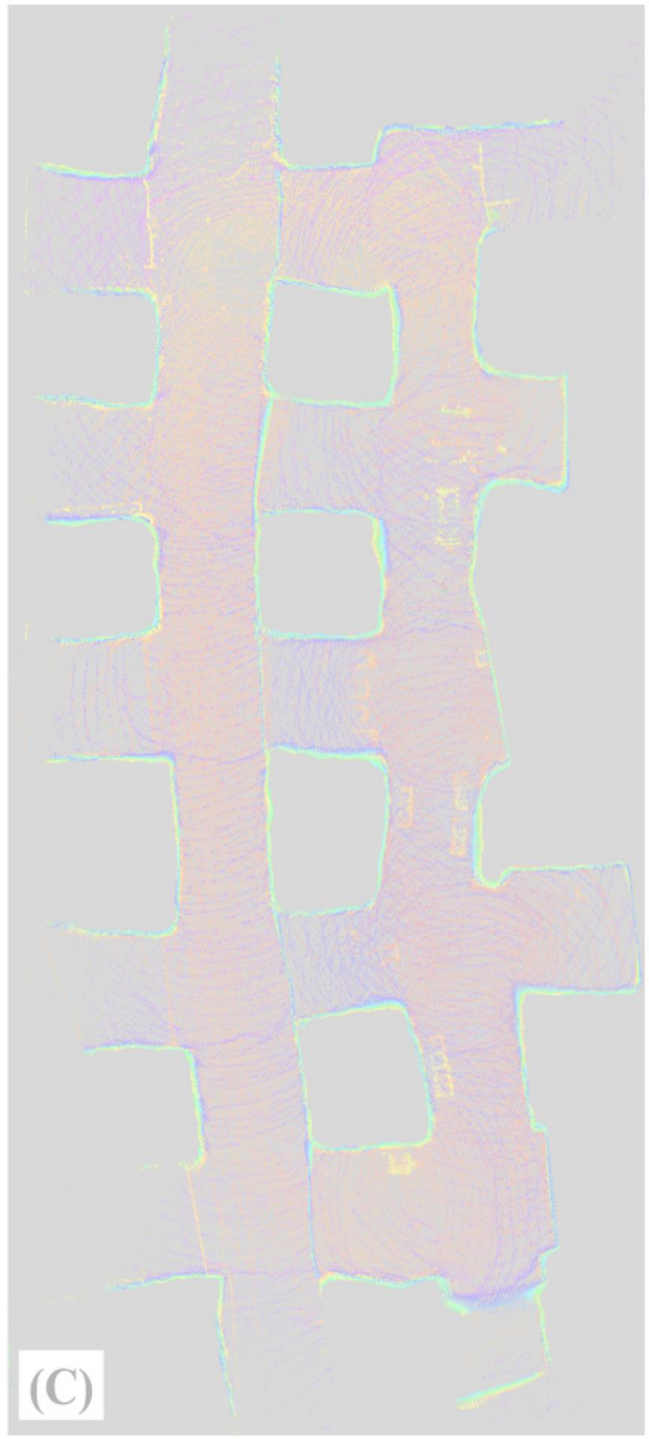
- 这部分的实验是针对完整的SLAM\Odometry系统进行对比；
- LIO-SAM和LVI-SAM因其输入数据的苛刻性没有纳入到对比中；
- 实验结果：绝对位姿误差、平均误差、实时性和CPU使用率都是最好的

总结

- 我在KITTI上进行了测试，xy方向的效果还挺不错，但是z轴drift有点大
- 官方提到加上imu可以有效抑制（待尝试）

APE	Map	XYZ(y是主观z轴)
		





Thank you!