Voxelized GICP for Fast and Accurate 3D Point Cloud Registration

Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno

National Institute of Advanced Industrial Science and Technology, Tsukuba, Japan,
(k.koide|yokotsuka-masashi|shuji.oishi|atsuhiko.banno)@aist.go.jp

Abstract. This paper proposes the voxelized generalized iterative closest point (VGICP) algorithm for fast and accurate three-dimensional point cloud registration. The proposed approach extends the generalized iterative closest point (GICP) approach with voxelization to avoid costly nearest neighbor search while retaining its accuracy. In contrast to the normal distributions transform (NDT), which calculates voxel distributions from point positions, we estimate voxel distributions by aggregating the distribution of each point in the voxel. The voxelization approach allows us to efficiently process the optimization in parallel, and the proposed algorithm can run at 30 Hz on a CPU and 120 Hz on a GPU. Through evaluations in simulated and real environments, we confirmed that the accuracy of the proposed algorithm is comparable to GICP, but is substantially faster than existing methods. This will enable the development of real-time 3D LIDAR applications that require extremely fast evaluations of the relative poses between LIDAR frames.

Keywords: point cloud, registration, GPU computing.

1 Introduction

Registration of three-dimensional (3D) point clouds is a crucial task for many 3D LIDAR applications, such as calibration, localization, mapping, and environment recognition. There are two popular point cloud registration methods for 3D LIDARs: Generalized Iterative Closest Point (GICP) [8] and the Normal Distributions Transform (NDT) [2, 4]. Whereas GICP extends the classical ICP algorithm [15] in a distribution-to-distribution comparison manner for accurate registration, the NDT takes advantage of the voxelization approach to avoid costly nearest neighbor search and improve processing speed. Both methods have their own weaknesses. Because GICP and other ICP-variants highly depend on nearest neighbor search, it is sometimes hard to run them in real time on a computer with limited computational power if the number of points is large. The NDT is typically very sensitive to the choice of the voxel resolution. The best voxel resolution depends on the environment and sensor properties, and if we do not choose an appropriate resolution, the registration accuracy of the NDT drastically drops.

In this paper, we propose the Voxelized GICP (VGICP) algorithm for fast and accurate 3D point cloud registration. The voxelization approach enables the

proposed algorithm to efficiently run in parallel, and our VGICP implementation can process a point cloud containing 15,000 points at 30 Hz on a CPU and 120 Hz on a GPU. By aggregating the distribution of all points in a voxel (multi-point distributions to a single-voxel distribution), we estimate the distributions of the voxels robustly. In contrast to the NDT, which estimates voxel distributions from point positions, this approach yields valid voxel distributions even when there are few points in a voxel, yielding an algorithm that is robust to changes in voxel resolution. The evaluation in simulated and real environments illustrates that the proposed VGICP algorithm shows registration accuracy that is comparable to that of GICP and outperforms other methods in terms of processing speed.

The contribution of this paper is three-fold. First, we propose a multi-point distribution aggregation approach to robustly estimate the distribution of a voxel from a smaller number of points. Second, we propose the VGICP algorithm, which is as accurate as GICP but substantially faster than existing methods. Third, the implementations are available from a public repository [1]. The repository contains the implementation of the proposed VGICP as well as a parallelized implementation of GICP.

2 Related Work

2.1 GICP

There are many variants of the classic ICP algorithm, such as Trimmed ICP [3] and Normal ICP [9]. GICP [8] is one of the most popular ICP variants. GICP extends the classical ICP algorithm in a distribution-to-distribution matching fashion. Although it is known for its good accuracy, this algorithm (and other ICP-variants) highly depend on nearest neighbor search to associate the closest points. Although an efficient KD-tree-based search is typically used, the nearest neighbor search often becomes a bottleneck so that the algorithm cannot run in real time when the number of points is large. Furthermore, a method based on a nearest neighbor search is not suitable for optimization on a GPU because it makes heavy use of conditional branches, which decreases the efficiency of the GPU computation significantly.

2.2 NDT

The NDT [2] takes a voxel-based association approach instead of an exact nearest neighbor search. This algorithm first splits an input point cloud into a set of voxels and fits a normal distribution to the points in each voxel. Then, it aligns another point cloud to the voxelized one by finding the transformation that maximizes the likelihood of the input points under the distributions of the voxels. Because the NDT avoids the costly nearest neighbor association, it is inherently much faster than ICP-variant algorithms. D2D-NDT (Distribution-to-Distribution NDT) [12] is an extension of the NDT that voxelizes both the

[1] `https://github.com/SMRT-AIST/fast_gicp`

source and target point clouds and calculate the distance between the distributions of the source and target voxels. Its comparison scheme is similar to that of GICP, and [5] suggests that D2D-NDT is superior to the classic NDT in terms of accuracy. However, the accuracy of the NDT and its variants depends on the choice of the voxel size. To obtain the best performance with the NDT, we need to carefully choose an appropriate voxel size depending on the sensor and environment properties. Some studies have proposed methods to make the NDT robust to hyper-parameter changes (e.g., multi-resolution [13] and trilinear voxel smoothing [4]). However, those extensions have a negative influence on the processing speed.

2.3 Feature-based registration

Feature-based registration methods first extract a number of representative features from input point clouds and then estimate the transformation from the feature correspondences. Many features have been proposed for point cloud registration, such as basic plane and edge features [11, 15], Fast Point Feature Histogram (FPFH) [6], and Signature of Histograms of OrienTations (SHOT) [7]. Because these features enable correspondences between point clouds to be robustly found, feature-based methods are inherently robust to initial pose errors (some do not even require an initial guess). However, because feature-based methods use only a limited number of features (which is usually much smaller than the number of input points), their accuracy is worse than that of point-based methods. Therefore, in a typical use case, a point-based fine registration is performed after a feature-based method. Feature-based and point-based registration methods are orthogonal, and they should be used to complement each other.

3 Proposed Method

In this section, we first explain the GICP algorithm and then extend it in a one-to-multiple distribution correspondence fashion to derive our VGICP algorithm.

3.1 GICP algorithm

We consider the estimation of the transformation $\mathbf{T}$, which aligns a set of points $\mathcal{A} = \{a_0, \cdots, a_N\}$ (source point cloud) with respect to another set of points $\mathcal{B} = \{b_0, \cdots, b_N\}$ (target point cloud). Following the classic ICP algorithm, we assume that the correspondences between $\mathcal{A}$ and $\mathcal{B}$ are given by nearest neighbor search: $b_i = \mathbf{T}a_i$. The GICP algorithm [8] models the surface from which a point was sampled as a Gaussian distribution: $a_i \sim \mathcal{N}(\hat{a}_i, C_i^A), b_i \sim \mathcal{N}(\hat{b}_i, C_i^B)$. Then, we define the transformation error as follows:

$$\hat{d_i} = \hat{b}_i - \mathbf{T}\hat{a}_i. \tag{1}$$

The distribution of d_i is given by the reproductive property of the Gaussian distribution as

$$d_i \sim \mathcal{N}(\hat{b}_i - \mathbf{T}\hat{a}_i, C_i^B + \mathbf{T}C_i^A\mathbf{T}^T) \tag{2}$$
$$= \mathcal{N}(0, C_i^B + \mathbf{T}C_i^A\mathbf{T}^T). \tag{3}$$

The GICP algorithm finds the transformation $\mathbf{T}$ that maximizes the log likelihood of Eq. (3) as follows:

$$\mathbf{T} = \arg\max_{\mathbf{T}} \sum_i \log(p(d_i)) \tag{4}$$
$$= \arg\min_{\mathbf{T}} \sum_i d_i^T (C_i^B + \mathbf{T}C_i^A\mathbf{T}^T)^{-1} d_i. \tag{5}$$

The covariance matrix of each point is typically estimated from its k neighbors (e.g., $k = 20$). Following the suggestion in [8], each covariance matrix is regularized by replacing its eigenvalues with $(1, 1, \epsilon)$. This regularization makes GICP work as a plane-to-plane ICP.

3.2 Voxelized GICP algorithm

To derive the voxelized GICP algorithm, we first extend Eq. (1) so that it calculates the distances between a_i and its neighbor points $\{b_j | \|a_i - b_j\| < r\}$ as follows:

$$\hat{d}_i' = \sum_j \left(\hat{b}_j - \mathbf{T}\hat{a}_i\right). \tag{6}$$

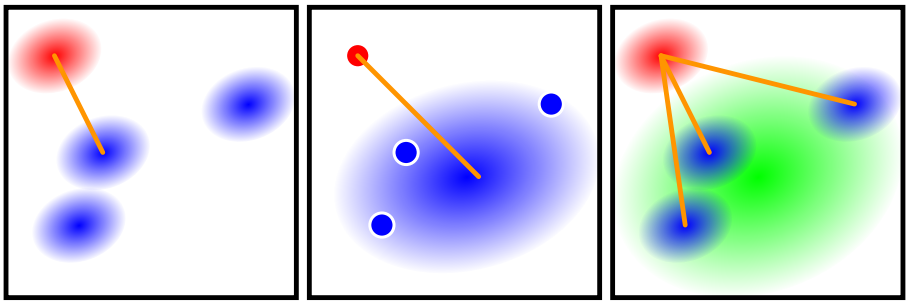
This equation can be interpreted as smoothing the target point distributions. Then, similar to Eq. (3), the distribution of d_i' is given by

$$\hat{d}_i' \sim (\mu^{d_i}, C^{d_i}), \tag{7}$$
$$\mu^{d_i} = \sum_j (\hat{b}_j - \mathbf{T}\hat{a}_i) = 0, \tag{8}$$
$$C^{d_i} = \sum_j (C_j^B + \mathbf{T}C_i^A\mathbf{T}^T). \tag{9}$$

We estimate the transformation $\mathbf{T}$ that maximizes the log likelihood of Eq. (7) as follows:

$$\mathbf{T} = \arg\min_T \sum_i \left(\sum_j (b_j - \mathbf{T}a_i)\right)^T \left(\sum_j \left(C_j^B + \mathbf{T}C_i^A\mathbf{T}^T\right)\right)^{-1} \left(\sum_j (b_j - \mathbf{T}a_i)\right). \tag{10}$$

To efficiently calculate the above equation, we modify it to



(a) GICP (b) NDT (c) VGICP

Fig. 1: Correspondence models for distance calculation in (a) GICP, (b) the NDT, and (c) VGICP. The red circles indicate a source point and the blue circles indicate target points. (GICP: nearest distribution-to-distribution, NDT: voxel-based point-to-distribution, VGICP: voxel-based distribution-to-multi-distribution.) The VGICP model yields a valid distribution even when a voxel contains only a few points.

$$\mathbf{T} = \arg\min_{T} \sum_{i} N_i \left(\frac{\sum b_j}{N_i} - \mathbf{T} a_i \right)^T \left(\frac{\sum C_j^B}{N_i} + \mathbf{T} C_i^A \mathbf{T}^T \right)^{-1} \left(\frac{\sum b_j}{N_i} - \mathbf{T} a_i \right), \tag{11}$$

where N_i is the number of neighbor points. Eq. (11) suggests that we can efficiently compute the objective function by substituting the mean of the distributions of the points (b_j and C^j) around a_i for b_i and C_i^B in Eq. (5) and weighting the function by N_i. We can naturally adapt this equation to voxel-based calculation by storing $b'_i = \frac{\sum b_j}{N_i}$ and $C'_i = \frac{\sum C_j^B}{N_i}$ in each voxel.

Fig. 1 illustrates the correspondence models used in GICP, the NDT, and our VGICP. GICP employs the nearest distribution-to-distribution correspondence model, which is reasonable, but relies on costly nearest neighbor search. For fast registration, the NDT uses the point-to-voxel-distribution correspondence model. However, we need at least four points (more than ten in practice) to calculate a 3D covariance matrix. If the number of points in the voxel is low, the covariance matrix becomes corrupted. Our VGICP exploits single-to-multiple distributions in the voxel correspondences to deal with the case in which only a few points fall within a voxel. Because it calculates a voxel distribution from point distributions, it yields a proper covariance matrix even when the voxel contains only one point.

3.3 Implementation

Algorithm 1 describes the registration procedure of VGICP in detail. As stated above, the VGICP algorithm does not require costly nearest neighbor search

Algorithm 1 VGICP algorithm

Point clouds : $\mathcal{A} = \{a_0, \cdots, a_N\}$, $\mathcal{B} = \{b_0, \cdots, b_M\}$
Covariances : $\mathcal{C}^A = \{C_0^A, \cdots, C_N^A\}$, $\mathcal{C}^B = \{C_0^B, \cdots, C_M^B\}$
Initial guess: $\tilde{\mathbf{T}}$
procedure VGICP($\mathcal{A}, \mathcal{B}, \mathcal{C}^A, \mathcal{C}^B, \tilde{\mathbf{T}}$)
    $\mathbf{T} \leftarrow \tilde{\mathbf{T}}$
    $\mathcal{V} \leftarrow$ VOXELIZATION($\mathcal{B}, \mathcal{C}^B$) ▷ Voxelization of target points $\mathcal{B}$
    while $\mathbf{T}$ is not converged **do**
        $e = [], J = []$
        for $i \in \{0, \cdots, N\}$ **do**
            voxel_index $\leftarrow$ convert_to_voxel_index(a_i)
            if voxel_index $\notin \mathcal{V}$ **then** ▷ Point did not fall in a voxel
                continue
            $e_i, J_i \leftarrow \text{Cost}(\mathbf{T}, a_i, C_i^A, v.\mu, v.C, v.N)$ ▷ Objective defined by eq. 11
            $e \leftarrow e \cup e_i, J \leftarrow J \cup J_i$
        $\delta\mathbf{T} \leftarrow -(J^T J)^{-1} J^T e$ ▷ Gauss-Newton update
        $\mathbf{T} \leftarrow \mathbf{T} \boxplus \delta\mathbf{T}$
    return $\mathbf{T}$
procedure VOXELIZATION($\mathcal{B}, \mathcal{C}^B$)
    $\mathcal{V} \leftarrow []$
    for $j \in \{0, \cdots, M\}$ **do**
        voxel_index = convert_to_voxel_index (b_j)
        if voxel_index $\notin \mathcal{V}$ **then**
            Insert $(\mu = 0, C = 0, N = 0)$ to $\mathcal{V}$[voxel_index]
        $\mathcal{V}$[voxel_index].$\mu \leftarrow \mathcal{V}$[voxel_index].$\mu + b_j$
        $\mathcal{V}$[voxel_index].$C \leftarrow \mathcal{V}$[voxel_index].$C + C_j^B$
        $\mathcal{V}$[voxel_index].$N \leftarrow \mathcal{V}$[voxel_index].$N + 1$
    for $v \in \mathcal{V}$ **do**
        $v.\mu \leftarrow v.\mu / v.N$
        $v.C \leftarrow v.C / v.N$
    return $\mathcal{V}$

during optimization, and it can hence leverage CPU and GPU parallel processing. For pose optimization, we choose the Gauss–Newton optimizer because it converges quickly and requires no hyper-parameters, unlike quasi-Newton methods.

We implemented three versions of the VGICP algorithm: single-threaded, multi-threaded, and GPU processing. All versions first estimate the covariance matrix of each point using a KD-tree-based nearest neighbor search [1]. This covariance estimation is parallelized in the multi-threaded and GPU processing versions. We also implemented a GPU-based brute force nearest neighbor search. As the baseline, we implemented the GICP algorithm in addition to VGICP. The GICP implementation is also parallelized by CPU multi-threading, is but not implemented on the GPU because it relies on KD-tree nearest neighbor search, which is not suitable for the GPU.

4 Experiment

4.1 Simulated environment

We evaluated the proposed VGICP algorithm in a simulated environment. For this evaluation, we developed a realistic LIDAR data simulator. To generate realistic LIDAR data, the simulator first renders an omnidirectional depth image and then performs ray casting to generate Velodyne-like rotating LIDAR data. Fig. 2 shows examples of point clouds generated by our simulator and Microsoft AirSim [10]. Whereas our simulator generates realistic point clouds, Airsim and other popular simulators generate LIDAR data from collision models, and thus their point clouds are not realistic.

With this simulator, we generated a sequence of LIDAR point clouds $(\mathcal{P}_0, \cdots, \mathcal{P}_N)$ and corresponding sensor poses $(\hat{R}_0, \cdots, \hat{R}_N$ and $\hat{p}_0, \cdots, \hat{p}_N)$. We used the parameters of Velodyne's HDL-32e sensor to simulate a real sensor model. We applied a scan matching method between consecutive frames and estimated the sensor pose (R_t, p_t) at time t by accumulating the scan matching results (i.e., the scan matching odometry). Following [16], we calculated the absolute trajectory error (ATE) and relative error (RE) to evaluate the accuracy of the scan matching methods.

The ATE is defined as follows:

$$\mathrm{ATE}_{\mathrm{rot}} = \left(\frac{1}{N} \sum_i \| \angle \left(\Delta R_i \right) \|^2 \right)^{\frac{1}{2}}, \tag{12}$$

$$\mathrm{ATE}_{\mathrm{pos}} = \left(\frac{1}{N} \sum_i \| \Delta p_i \|^2 \right)^{\frac{1}{2}}, \tag{13}$$

where $\Delta R_i = R_i (\hat{R}_i)^{-1}$ and $\Delta p_i = p_i - \Delta R_i \hat{p}_i$. Note that the estimated sensor trajectories are aligned with respect to the ground truth using [14]. The RE is defined as follows:

$$\mathrm{RE}_{\mathrm{rot}} = \angle \left(\delta R_k \right) = \angle \left(R_e \hat{R}_e^T \right), \tag{14}$$

$$\mathrm{RE}_{\mathrm{pos}} = \| p_e - \delta R_k \hat{p}_e \|_2, \tag{15}$$

where $R_e = R_t^{-1} R_{t+N}$ and $p_e = p_t - p_{t+N}$. Parameters t and $t + N$ define the size of the window used to evaluate the RE. We set the window size to 1, 5, and 25 m (sensor travel distance) and evaluated the RE using a sliding window.

Fig. 3 shows the REs of different registration methods, and the absolute trajectory errors are shown in Table 1. We evaluated our VGICP and GICP implementations as well as the ICP, GICP, and NDT implementations in Point Cloud Library (PCL). For VGICP and the NDT, we tested several voxel resolutions to evaluate the stability of their performance with respect to changes in hyper-parameter values.

The accuracy of the NDT heavily relies on the choice of voxel resolution. We can see that the registration accuracy of the NDT largely drops for voxel

(a) Our LIDAR simulator.

(b) Microsoft Airsim [10].

Fig. 2: Examples of point clouds generated by our simulator and Microsoft AirSim. AirSim generates point clouds from collision models, and thus objects in the LIDAR data have shapes that are too simplified (see the points on the trees, cars, and buildings), and objects without collision models (pedestrians) do not appear in the point clouds. Our simulator performs raycasting on an omnidirectional depth image to generate realistic point clouds.

resolutions of 1, 2, and 6 m compared to the best one. In this evaluation, we observed the best accuracy with the NDT when the voxel resolution is 4.0 m. This value is, however, too large to capture the fine details of the environment, and thus the accuracy of the NDT is worse than that of GICP.

The GICP-based algorithms show superior accuracy compared to the classic ICP algorithm. Our VGICP and GICP implementations yield slightly better accuracy than does the PCL implementation of GICP. This might be due to the

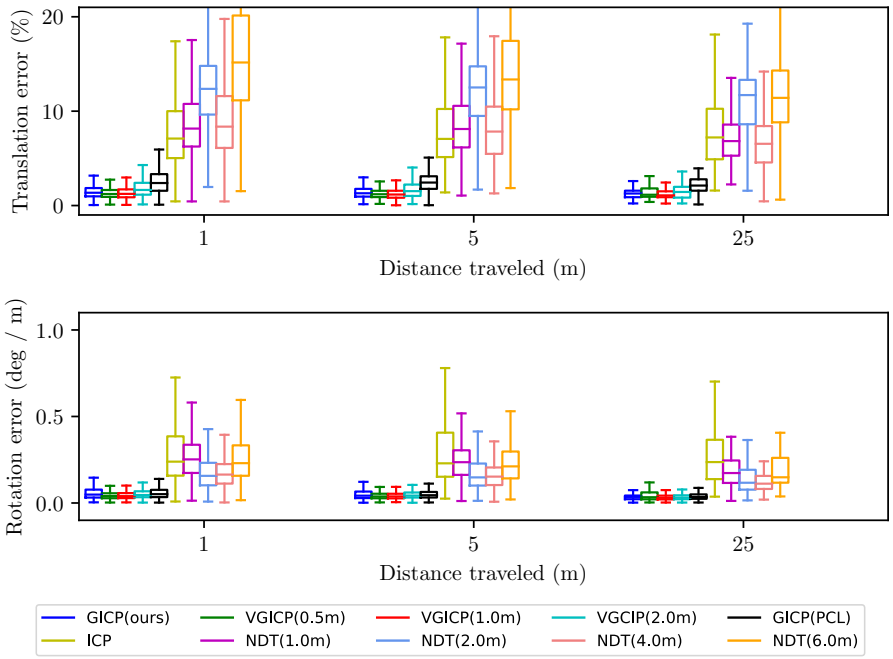


Fig. 3: REs of different registration methods.

Table 1: Absolute trajectory errors.

	[m]	[deg]
GICP(ours)	**0.562**	**1.696**
VGICP(1.0m)	**0.624**	**2.777**
GICP(PCL)	**0.954**	**1.724**
ICP	75.349	32.171
NDT(4.0m)	7.952	23.971

Bold indicates the top three results.

choice of the optimizer (our implementations use Gauss–Newton, which can be faster and more accurate than the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm used in the PCL version of GICP). We note that our VGICP algorithm shows consistent results over a wide range of voxel resolutions thanks to the proposed voxelization approach, which yields a valid distribution even when the number of points in a voxel is low. This result illustrates that the proposed VGICP algorithm shows comparable accuracy with respect to GICP and is robust to hyper-parameter changes.

Fig. 4 shows the average processing times of the registration methods. All methods were run on an Intel Core i9-9900K and NVIDIA Geforce RTX2080Ti. For our GICP and VGICP implementations, we evaluated both the single and

X

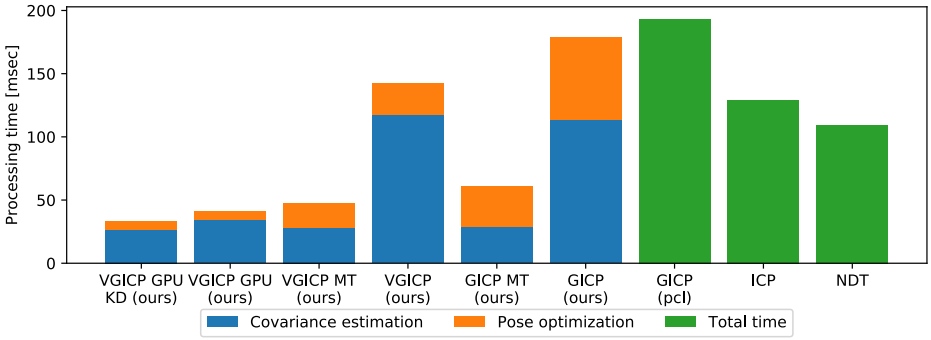


Fig. 4: Average processing time per scan.

multi-threaded versions. We also evaluated the GPU version of our VGICP algorithm. The results show that our single-thread implementations (GICP: 189 msec, VGICP: 156 msec) are faster than the reference GICP implementation in PCL (201 msec). Because the preprocessing part (the estimation of the covariance matrix of each point) is the same for both GICP and VGICP, they spend the same amount of time for preprocessing. The following pose optimization part is greatly sped up by the voxelization approach. This suggests that in scenarios in which the target point cloud is not frequently changed (e.g., keyframe-based odometry estimation), we can obtain more benefits from the high efficiency of the VGICP algorithm. With multi-threading, our implementations of GICP and VGICP are further sped up, taking only 68 msec and 50 msec, respectively. Our GPU-implementation of VGICP is extremely fast and takes only 6 msec to optimize. It is worth mentioning that the GPU-based brute-force nearest neighbor search is slower than the CPU-based parallel KD-tree in this evaluation. However, it would be a reasonable choice for computers with a low-specification CPU and a high-performance GPU such as the NVIDIA Jetson.

4.2 Real environment

We recorded eight LIDAR sequences in the environment shown in Fig. 5 using a Velodyne HDL-32e sensor. The length of each trajectory is about 120 [m], and the number of points in each frame is about 15,000. Similar to the evaluation in Sec. 4.1, we estimated the sensor trajectory by applying registration algorithms between consecutive frames. For our implementations, we reused the calculated covariance matrices of an input point cloud in the registration of the next frame to reduce the preprocessing cost. For each sequence, we aligned the last frame with respect to the first frame using GICP and used the result as ground truth.

Table 2 shows the pose estimation errors at the last frames (i.e., the accumulated registration errors). ICP and the NDT obtain worse results than the GICP-based algorithms. The NDT obtains the best accuracy (2.717 m and 0.164 deg error) when the voxel resolution is set to 1.0 m. However, the accuracy dras-

Fig. 5: Experimental environment.

Table 2: Processing speed and transformation errors

Method	FPS (CPU/GPU)	Translation[m]	Rotation[deg]
VGICP (0.5m)	**28.7/116**	**0.852** ± 0.289	**0.049** ± 0.013
VGICP (1.0m)	**30.4/120**	**1.177** ± 0.456	**0.048** ± 0.026
GICP (ours)	**20.5**	**0.893** ± 0.210	**0.045** ± 0.023
GICP (PCL)	5.2	1.316 ± 0.310	0.051 ± 0.022
NDT (0.5m)	8.3	7.119 ± 5.640	0.443 ± 0.331
NDT (1.0m)	10.3	2.717 ± 2.645	0.164 ± 0.088
NDT (2.0m)	10.4	3.290 ± 1.420	0.116 ± 0.066
NDT (4.0m)	9.3	5.590 ± 2.177	0.149 ± 0.077
ICP	11.0	10.192 ± 4.365	0.371 ± 0.106

Bold indicates the top three results.

tically drops when the voxel resolution is too small or too large (7.119 m and 0.443 deg for 0.5 m resolution; 5.590 m and 0.149 deg for 4.0 m resolution).

The PCL-based GICP obtains a better result than do ICP and the NDT (1.316 m and 0.051 deg). However, because it heavily relies on costly nearest neighbor search, it is the slowest of the evaluated methods (5.2 fps).

Our GICP implementation is substantially sped up by multi-threading and the fast Gauss–Newton optimizer (20.5 fps). It also obtains a higher accuracy than the PCL-based GICP (0.893 m and 0.045 deg). The VGICP algorithm has almost the same accuracy as our GICP implementation (0.852 m and 0.049 deg) when the voxel resolution is set to 0.5 m. Although the accuracy of VGICP slightly drops (1.177 m and 0.048 deg) when the voxel resolution is changed to 1.0 m, the result is still comparable to that of GICP. Our VGICP algorithm leverages CPU multi-threading, which results in the best processing speed (about 30 fps). Furthermore, it becomes extremely fast on the GPU (about 120 fps).

5 Conclusion and Future Work

In this study, we proposed the voxelized GICP algorithm. The proposed VGICP is as accurate as GICP because it utilizes the voxel-based association approach. The evaluation results in the simulated and real environments show that the proposed method shows a superior processing speed (30fps on a CPU and 120 fps on a GPU) and is robust to voxel resolution changes.

We plan to evaluate and improve the convergence of the propose VGICP algorithm because it employs the voxelization approach which may affect the registration result when the initial guess is not close to the true pose. We also plan to develop an IMU-LIDAR joint optimization-based localization method with VGICP, which requires an extremely fast evaluation of the relative poses between keyframes.

Acknowledgement

This work was partly supported by JSPS KAKENHI(Grant Number 18K18072) and a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

We would like to thank H. Hamagaki for his efforts on the development the LIDAR simulator.

References

1. Bentley, J.L.: Multidimensional binary search trees used for associative searching. vol. 18, pp. 509–517. ACM (Sep 1975)
2. Biber, P., Strasser, W.: The normal distributions transform: a new approach to laser scan matching. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE (Oct 2003)
3. Chetverikov, D., Svirko, D., Stepanov, D., Krsek, P.: The trimmed iterative closest point algorithm. In: Object recognition supported by user interaction for service robots. IEEE (Aug 2002)
4. Magnusson, M., Nuchter, A., Lorken, C., Lilienthal, A., Hertzberg, J.: Evaluation of 3d registration reliability and speed - a comparison of ICP and NDT. In: IEEE International Conference on Robotics and Automation. IEEE (May 2009)
5. Magnusson, M., Vaskevicius, N., Stoyanov, T., Pathak, K., Birk, A.: Beyond points: Evaluating recent 3d scan-matching algorithms. In: IEEE International Conference on Robotics and Automation. IEEE (May 2015)
6. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (FPFH) for 3d registration. In: IEEE International Conference on Robotics and Automation. IEEE (May 2009)
7. Salti, S., Tombari, F., Stefano, L.D.: SHOT: Unique signatures of histograms for surface and texture description. vol. 125, pp. 251–264. Elsevier (Aug 2014)
8. Segal, A., Haehnel, D., Thrun, S.: Generalized-ICP. In: Robotics: Science and Systems. Robotics: Science and Systems Foundation (Jun 2009)