Received: 19 May 2021 | Accepted: 3 August 2021

DOI: 10.1002/rob.22032

RESEARCH ARTICLE

WILEY

Sparse point cloud registration and aggregation with mesh-based generalized iterative closest point

Matthew Young[1] | Chris Pretty[1] | Josh McCulloch[2] | Richard Green[2]

[1]Department of Mechanical Engineering, University of Canterbury, Christchurch, New Zealand

[2]Department of Computer Science and Software Engineering (CSSE), University of Canterbury, Christchurch, New Zealand

Correspondence
Matthew Young, Department of Mechanical Engineering, University of Canterbury, Upper Riccarton, Christchurch 8041, New Zealand.
Email: msy22@uclive.ac.nz

Funding information
Callaghan Innovation; Trimble Inc.

Abstract

Accurate registration is critical for robotic mapping and simultaneous localization and mapping (SLAM). Sparse or non-uniform point clouds can be very challenging to register, even in ideal environments. Previous research by Holz et al. has developed a mesh-based extension to the popular generalized iterative closest point (GICP) algorithm, which can accurately register sparse clouds where unmodified GICP would fail. This paper builds on that work by expanding the comparison between the two algorithms across multiple data sets at a greater range of distances. The results confirm that Mesh-GICP is more accurate, more precise, and faster. They also show that it can successfully register scans 4–17 times further apart than GICP. In two different experiments this paper uses Mesh-GICP to compare three different registration methods—pairwise, metascan, keyscan—in two different situations, one in a visual odometry (VO) style, and another in a mapping style. The results of these experiments show that the keyscan method is the most accurate of the three so long as there is sufficient overlap between the target and source clouds. Where there is unsufficient overlap, pairwise matching is more accurate.

KEYWORDS
GICP, PCL, registration, sparse point cloud

1 | INTRODUCTION

Simultaneous localization and mapping (SLAM) algorithms are a core component of most mobile robots, and they depend on sensors such as LiDAR and RGB-D cameras to provide visual data about the robot's environment. As their cost decreases, 3D LiDAR devices are increasingly becoming the sensor of choice for this purpose. SLAM algorithms vary in their methodology, but most state-of-the-art algorithms follow the same three-stage process:

(1) Coarse alignment of individual clouds using information from odometry or registration of the previous cloud.
(2) Fine alignment of adjacent or overlapping clouds using point cloud registration. Clouds are often registered to previous clouds—"pairwise," or grouped in a "metascan."
(3) Optimization of cloud placement by modelling the constraints between poses and minimizing them with a pose graph.

Of the three, Stage 2 is the most important. Poor cloud registration may adversely affect any state estimate (e.g., from a Kalman Filter) it is included in, and it may cause the pose graph to propagate any error to adjacent clouds along the robot's trajectory. Having the robot return to a previously visited position to "close the loop" can sometimes correct such gross errors, but this is not guaranteed. Especially severe failures in registration can distort the robot's trajectory so as to make it impossible for the SLAM algorithm to detect the loop closure in the first place (Wulf et al., 2008).

Achieving highly accurate point cloud registration can be a difficult task, even more so when the point clouds are irregular. Many popular 3D LiDAR designs have a small number of LiDAR emitter and receiver pairs, with limited vertical resolution. In addition, it is still

common for some researchers to use a two-dimensional (2D) LiDAR device in an actuated housing to create 3D point clouds (Holz & Behnke, 2016; Viejo & Cazorla, 2007; Xiao et al., 2012; Zhang & Singh, 2017). In both instances the result is a sparse or non-uniform density cloud, where points are grouped in vertical or horizontal rings.

This makes successful cloud registration difficult for a several of reasons. First, the point density may not be high enough in some parts of the cloud to accurately calculate the point normals or covariances, which are necessary for several registration methods. Second, the ringed nature of the clouds means that two consecutive clouds may have scanned different parts of the environment with little overlap, even if the clouds are relatively close. Third, incorrect point correspondence can lead to clouds being matched along scanlines, as illustrated in Holz and Behnke (2014). For these reasons, even state-of-the-art registration algorithms like generalized iterative closest point (GICP) (Segal et al., 2009) may routinely fail to accurately register sparse LiDAR scans.

Holz and Behnke (2014) proposed an extension of the GICP algorithm which computes the required covariance matrices from a mesh, rather than the underlying points. Provided the mesh parameters are well tuned, and there are enough distinctive features in the environment, this extended method can successfully register sparse scans where ordinary GICP fails.

This paper builds on the work of Holz and Behnke (2014, 2016); Razlaw et al. (2015) by expanding the comparison between GICP and MGICP. Specifically, it makes the following contributions: It thoroughly compares GICP versus MGICP at near to long ranges, using multiple data sets, and using multiple samples to quantify the precision as well as accuracy. In addition to expanding on the works of Holz and Behnke, this paper also experiments with three different methods of ordering the registration operations when registering and aggregating multiple clouds. The first two methods are commonly called "pairwise" and "metascan" registration. This paper uses both in addition to "keyscan" aggregation. These methods have a strong influence on the accuracy of the registration process, especially when acting alongside MGICP, as the results of this paper will show.

The paper is organized as follows: Section 2 covers existing research into the registration algorithms and methods that will be used. Section 3 contains the experimental setup, including descriptions of the hardware, software, error metric and notation used in this study. Section 4 describes the methodology and results before Section 5 discusses their significance.

2 | RELATED WORK

2.1 | Iterative closest point

The ICP family of algorithms are a popular choice for point cloud registration. First introduced by Besl and McKay (1992), the goal of ICP is align or "register" two point clouds. In the simplest form of this algorithm, two clouds A and B have m points in the sets $A = \{\mathbf{p}_{ia}, \ldots, \mathbf{p}_{ma}\}$ and $B = \{\mathbf{p}_{ib}, \ldots, \mathbf{p}_{mb}\}$, respectively. Where $\mathbf{p}_{ia}$ and $\mathbf{p}_{ib}$ are corresponding points. The "Point-to-Point" ICP algorithm then finds a transform $\mathbf{T}$ that aligns the clouds by minimizing an error function E shown in Equation (1).

$$E(\mathbf{T})_{ICP} = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{T}\mathbf{p}_{ia} - \mathbf{p}_{ib}\|^2. \tag{1}$$

ICP algorithm starts with an initial guess for each match and then iteratively improves the estimate of $\mathbf{T}$ by rematching pairs and recomputing the error function, eventually converging to an optimal solution.

A later variant of the minimization algorithm was created to instead minimize the distance between the points in one cloud, and the planes in the other. "Point-to-Plane" ICP (PICP) (Chen & Medioni, 1992) modifies the ICP error function as shown:

$$E(\mathbf{T})_{PICP} = \frac{1}{m}\sum_{i=1}^{m} \|(\mathbf{T}\mathbf{p}_{ia} - \mathbf{p}_{ib})\mathbf{n}_{iab}\|^2, \tag{2}$$

where $\mathbf{n}_{iab}$ is orthogonal to the modelled surface in B and runs between the surface and point $\mathbf{p}_{ia}$. This algorithm minimizes the error along the direction of the surface normals, while allowing the clouds to slide in directions orthogonal to the normals. GICP (Segal et al., 2009) extends this idea further, in that rather than minimizing point-to-point or point-to-plane distances, GICP's error function E finds the optimal transformation matrix $\mathbf{T}$ that minimizes the distance between corresponding covariances in cloud A and B:

$$E(\mathbf{T})_{GICP} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{d}_i^T(\mathbf{C}_i^B + \mathbf{T}\mathbf{C}_i^A\mathbf{T}^T)^{-1}\mathbf{d}_i, \tag{3}$$

where $\mathbf{C}_i^B$ and $\mathbf{C}_i^A$ are the covariance matrices of corresponding points $\mathbf{p}_{ia}$ and $\mathbf{p}_{ib}$, and $\mathbf{d}_i = \mathbf{p}_{ib} - \mathbf{T}\mathbf{p}_{ia}$ is a vector of their point-to-point distances. Note that here $\mathbf{d}_i^T$ and $\mathbf{T}^T$ are transposes. GICP methodology assumes that each point is sampling a locally planar surface, and that each point $\mathbf{p}_{ia}$ or $\mathbf{p}_{ib}$ only provides a constraint along their normals $\mathbf{n}_{ia}$ or $\mathbf{n}_{ib}$, and has a high degree of uncertainty as to where exactly on the plane it is. To model this uncertainty, GICP computes the covariance matrices as follows:

$$\mathbf{C}_i^A = \mathbf{R}_{\mathbf{n}_{ia}}\begin{bmatrix}\epsilon & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\mathbf{R}_{\mathbf{n}_{ia}}^T, \mathbf{C}_i^B = \mathbf{R}_{\mathbf{n}_{ib}}\begin{bmatrix}\epsilon & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\mathbf{R}_{\mathbf{n}_{ib}}^T, \tag{4}$$

where ϵ is a small constant that represents the uncertainty along the normals $\mathbf{n}_{ia}$ and $\mathbf{n}_{ib}$. Matrices $\mathbf{R}_{\mathbf{n}_{ia}}^T$ and $\mathbf{R}_{\mathbf{n}_{ia}}^T$ are rotation matrices that transform the basis vector $e_1 \rightarrow \mathbf{n}_i$. The ICP algorithm, particularly the original and point-to-plane variants, relies on accurate correspondence between point pairs. Improperly selected pairs can cause the minimization algorithm to converge to an incorrect alignment. One of the reasons GICP is more accurate is because by matching covariance matrices rather than points or planes, more

information about the local region is included, and so the covariance matrices are less sensitive to suboptimal pair choices (Segal et al., 2009).

Note that the names of these minimization algorithms are typically used to refer to the entire registration process, whereas in execution there are many important steps in the registration process, which all involve the selection, weighting and filtering of corresponding point pairs (Donoso et al., 2017a, 2017b; Rusinkiewicz & Levoy, 2001).

Other studies have iterated further on the ICP algorithm. Servos and Waslander (2014) uses the GICP algorithm as described above, but incorporates LiDAR intensity and colour information from an RGB camera into the point covariance matrices $\mathbf{C}_i^A$ and $\mathbf{C}_i^B$. Korn et al. (2014) also use colour information, but weights it and uses it in the distance calculation between potential point pairs. The intuition is that points of different colour have a greater "colour distance" and should be deprioritized by a nearest-neighbour search algorithm.

2.2 | Sparse cloud registration with Mesh-GICP

For reasons discussed in the introduction, ICP algorithms can struggle to accurately register sparse point clouds. The fundamental problem is that a sparse or non-uniform cloud does not adequately represent the underlying topography. Many solutions attempt to bypass this problem by fitting planes or other geometric features to the available points. For example, LiDAR Odometry and Mapping (LOAM) uses K-D trees to detect planar surfaces and edges in each newly acquired cloud (Zhang & Singh, 2017). Other methods use: a Hough-based voting scheme (Grant et al., 2013), a region-growing algorithm (Georgiev et al., 2011; Pathak et al., 2010; Xiao et al., 2012) or principal component analysis (Viejo & Cazorla, 2007) to identify and register planes rather than the raw cloud.

Because of their dependence on planes, these methods are limited to highly structured environments where a sufficient number of planes can be extracted. Instead, the works of Holz and Behnke (2014, 2016) simplify this process by organizing the raw cloud and connecting adjacent points to form a mesh. The mesh is then used to approximate the covariances $\mathbf{C}_i^B$ and $\mathbf{C}_i^A$ (see Equations 3 and 4) rather than the raw points. Compared to alternative solutions, this method is more accurate (Razlaw et al., 2015) and has a very low computational cost (Holz & Behnke, 2014). The first key part to this methodology is the creation of organized point clouds. An unorganized cloud is stored as an unsorted, 1D list of points. So to find the nearest neighbours of any given point, a search tree must be constructed. An organized cloud however, is stored as a 2D array, where the ordering of points in the array is the same as their spacial ordering in the cloud. That is, the nearest four neighbours of any given point are the four points directly adjacent to it in its row and column of the array. In addition, laser pulses that do not return from the terrain to the LiDAR are still stored in an organized cloud as null or not-a-number (NaN) points. This is opposed to an unorganized cloud where such failed returns are typically not recorded at all. This makes nearest neighbour operations for organized clouds trivial, and guarantees the correct spatial relationship between points.

This is particularly important for creating meshes from sparse point clouds with a non-uniform point density. For example, in Figure 1, the points of the Velodyne HDL-32E scan are concentrated into rings. So when a K-D tree is used to execute a nearest neighbour search on each point, the closest points all reside in the same ring, resulting in a incomplete mesh where the triangle vertices all reside in the same ring (Figure 1a). This mesh was created using PCL's Greedy Projection Triangulation algorithm (Marton et al., 2009) with the parameters listed in the appendix in Table A1.

By contrast, the mesh generated from an organized mesh in Figure 1b is significantly more accurate, and was created faster, simply because the triangles could be formed by connecting each point with it's nearest row and column neighbour. Some additional filtering steps are taken to remove occluded points and points that are not included in any mesh triangle (Holz & Behnke, 2016).

The second key part of this methodology is the creation of approximate covariances from the mesh, rather than the raw point cloud. First, the normal $\mathbf{n}_i$ of each point $\mathbf{p}_i$ is calculated as the weighted normal of the surrounding N_T faces, with each surface normal weighted proportional to the area of the triangle it corresponds to (Holz & Behnke, 2016), summarized by Equation (5) and reproduced directly from Holz and Behnke:

$$\mathbf{n}_i = \frac{\sum_{j=0}^{N_T} (\mathbf{p}_{j,a} - \mathbf{p}_{j,b}) \times (\mathbf{p}_{j,a} - \mathbf{p}_{j,c})}{\left\| \sum_{j=0}^{N_T} (\mathbf{p}_{j,a} - \mathbf{p}_{j,b}) \times (\mathbf{p}_{j,a} - \mathbf{p}_{j,c}) \right\|}, \tag{5}$$

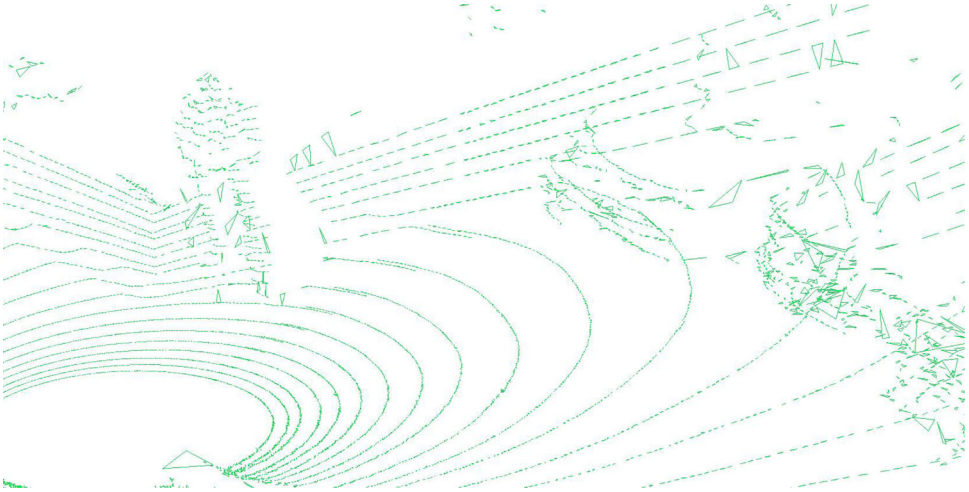
where $\mathbf{p}_{j,a}$, $\mathbf{p}_{j,b}$, and $\mathbf{p}_{j,c}$ are three neighbouring vertices that form triangle j. The covariance matrices $\mathbf{C}_i^A$ and $\mathbf{C}_i^B$ from Equation (3) are then calculated as per the normal GICP methodology (Equation 4)

However there are several limitations to the results of Holz and Behnke (2014, 2016) and Razlaw et al. (2015). They do not provide registration results when the distance between the source and target cloud is large. They report the result of only one registration per data point, rather than taking multiple samples. They do not show the rotation error from registration, only the translation error. And they present the results of multiple scan pairs, but only from one data set. Given the significant improvement in computational speed and accuracy of MGICP over GICP, this is an algorithm that warrants further analysis. It should be noted that there are other studies that have also sought to address the problem of registering sparse point clouds with ICP. In particular, Bouaziz et al. (2013) and later Mavridis et al. (2015) use sparsity-inducing norms and an additional optimizer to minimize the effect of outlying points. However these methods are designed and optimized for 3D point clouds of small objects, rather than scans of large environments.

2.3 | Point cloud aggregation

Point cloud registration is typically referred to by the minimization algorithm that is used, for example, "ICP registration." The implicit assumption that one single point cloud is being registered to another single point cloud is often made when referring to registration in this way. But when registering multiple point clouds in sequence, the order in which they are

(a) Mesh created from an unorganized cloud. Note that most triangles are formed between adjacent points in the same LiDAR ring.



(b) Mesh created from an organized cloud. Triangles are formed by connecting points that are adjacent in the 2D array that stores the cloud.

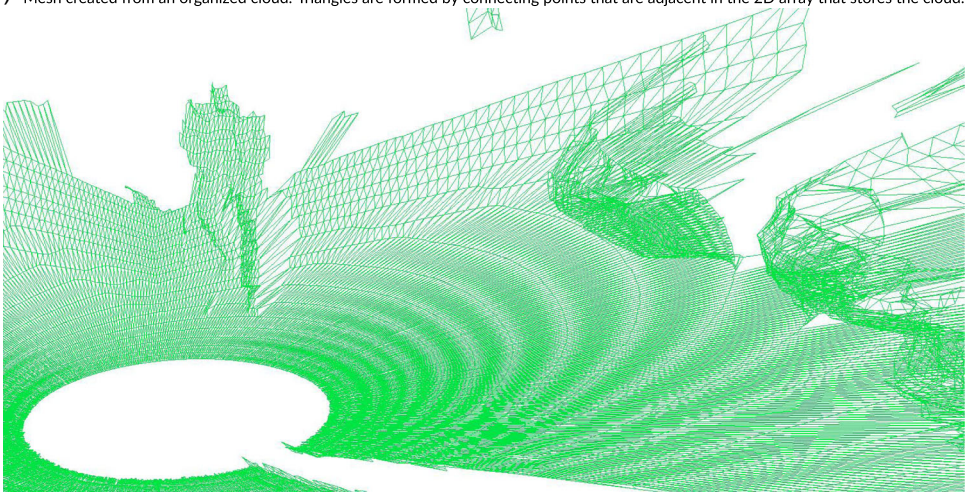


FIGURE 1 Difference between unorganized and organized clouds in mesh creation. The Velodyne HDL-32E point cloud scan is of a pedestrian sitting on a low retaining wall next to two cars. (a) Mesh created from an unorganized cloud. Note that most triangles are formed between adjacent points in the same LiDAR ring; (b) Mesh created from an organized cloud. Triangles are formed by connecting points that are adjacent in the 2D array that stores the cloud

registered and then aggregated plays a vital role in the accuracy of the result. Two of the most common aggregation methods are called "pairwise" and "metascan." Pairwise registers each new cloud to the previously registered cloud, while metascan accumulates registered scans into one monolithic scan, which is the target of all registration operations. These procedures for ordering and grouping registered clouds can more broadly be called "aggregation" methods, to avoid confusing them with the minimization algorithm (e.g., GICP).

In addition to pairwise and metascan aggregation, this paper also uses an additional aggregation method called "keyscan." In this method, one cloud in the sequence is chosen as the target for *all* registration operations. So every other cloud in the sequence is registered to the "key" cloud or scan. The reason for including this method is that it maintains the organized status of each cloud, thus enabling the use of MGICP, while avoiding the compounding of registration errors which can occur with pairwise aggregation.

With reference to Figure 2, the three aggregation methods which will be used can be more formally described as follows:

(1) *Pairwise*: Each newly acquired cloud A_i is registered to the previous cloud $A_i \Rightarrow A_{i-1}$.
(2) *Metascan*: (also called "incremental") Every new cloud A_i is first registered and then concatenated to a larger meta-cloud $A_i \Rightarrow M$, which consists of all previously registered clouds $M = \{A_j | j \in 1 \ldots i-1\}, i > 1$.
(3) *Keyscan*: In each set of clouds, one is chosen as the 'key' scan A_k and all other clouds in the set are registered to it $A_i \Rightarrow A_k, \{A_i, A_k | i, k \in n \ldots m\}, i \neq k$.

Where each arrow denotes the order of registration, pointing from source to target. Of the three, pairwise is the most commonly used aggregation method as it integrates well with graph-based optimization

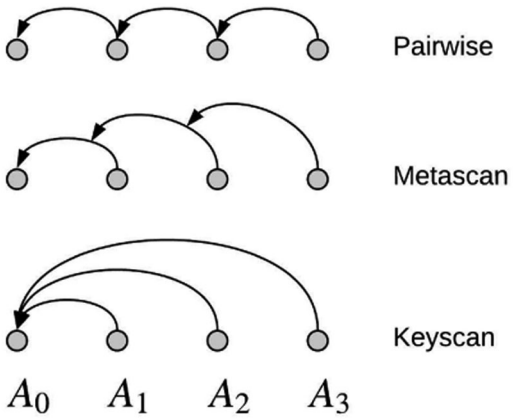


FIGURE 2 Order of registration for the pairwise, metascan and keyscan aggregation methods

algorithms. The typical result of the metascan and keyscan methods is a large, aggregated point cloud, and as such these methods are more commonly used where submaps are employed. In such cases, if the original poses of the clouds are not stored before they are added to the metascan, then any alignment errors cannot be undone at a later time. Most research that conducts point cloud registration simply chooses one of these methods arbitrarily, or based on the design of the rest of the system. As such, point cloud aggregation methods are an under-researched area, because there are few papers that directly compare these methods.

Wulf et al. (2008) compare pairwise and metascan aggregation methods using point-to-point ICP with and without the popular 6D Lu and Milos (LUM) (Borrmann et al., 2008) graph optimization algorithm. Their results show that the metascan methods consistently outperform their pairwise equivalents in terms of position and orientation error, but take considerably longer to compute Wulf et al. (2008). For this study, registration was performed on 3D point clouds, but the position and rotation errors were expressed using 2D metrics.

Razlaw et al. (2015) also compares point cloud aggregation methods. Specifically pairwise, incremental and incremental with multiresolution surfel maps, using the ICP, GICP, MGICP, normal distributions transform (NDT), and Surfel registration algorithms. Their study however does not present the results or data from this experiment, stating only that the incremental surfel aggregation method achieved the best results (Razlaw et al., 2015).

3 | EXPERIMENTAL SETUP

The remainder of this paper covers three distinct experiments:

1 Compares GICP with MGICP at close to long range (0–25 m), beyond the ranges shown in Holz and Behnke (2014, 2016) and Razlaw et al. (2015).
2 Compares the three described aggregation methods (keyscan, pairwise, and metascan) with both the standard GICP minimization algorithm and the mesh-based MGICP, at ranges of 0–25 m. In this experiment, the initial placement of each scan is determined by the previous registration result, simulating a visual odometry (VO) or SLAM-style system (see Figure 5a).
3 Is identical to Experiment 2, except that the initial placement of scans is determined by adding a random amount of error to the true pose of each scan, simulating a mapping-style situation where the placement of each scan is often determined by GNSS data, which contains a nonzero amount of error in its position estimate (see Figure 5b).

Each experiment is described in greater detail in their dedicated sections (see Sections 4.1–4.3). All experiments use the same three data sets, called Garage, Carpark, and Forest. Each data set consists of a number of Velodyne HDL-32 scans, their respective poses and one ground-truth point cloud. How these Velodyne scans are collected is explained in Section 3.1. The ground-truth cloud of each area was collected using a Trimble SX10 Total Station, which is used to find the true locations of each Velodyne scan as explained in Section 3.2. All experiments also use the same error metrics, which are all discussed in Section 3.3. It should be emphasized that point cloud registration typically encompasses an entire workflow including prefiltering, selection of corresponding point pairs, filtering and weighting of pairs, minimization, and then often a back-end involving pose optimization. However for the purposes of this paper, the focus is on the registration minimization algorithms (GICP vs. MGICP) and aggregation methods (metascan, pairwise, and keyscan). So all additional steps are ignored, tuning is minimal, and most algorithms use the default parameters. Parameters for the most salient algorithms are given in the appendix of this paper. It must also be noted that the testing of registration to a range of 25 m is deliberate, to test the accuracy of the algorithms beyond the ranges given in Holz and Behnke (2014, 2016) and Razlaw et al. (2015), and to test the algorithms beyond the point of failure.

3.1 | Hardware and data collection

Testing MGICP and the various aggregation methods required multiple data sets that fulfilled the following criteria:

(1) A set of 3D point clouds that could be easily organized.
(2) Data that provided the ground-truth position of each point cloud, or a method to obtain this information.
(3) Point clouds collected sequentially whilst moving through an environment (to create significant distance between the first and last cloud).
(4) Point clouds collected in a range of environments from highly planar to highly unstructured.

Unfortunately, while there are many public data sets available online, and many which are dedicated to testing point cloud registration, all failed to meet one or more of the above criteria.

For example, RGB-D SLAM data set and benchmark used by Sturm et al. (2012) provides an extensive range of ground-truthed and organized RGB-D data sets. But the clouds themselves are short range, and due to the photogrammetric generation of the distance measurements, less accurate than LiDAR. This makes them less desirable than the clouds produced by the robotic platform described below. Other LiDAR-based data sets, such as the well-known KITTI data set (Geiger et al., 2013), or the data set described by Pomerleau et al. (2012), do not include NaN or Null points to represent LiDAR pulses that did not return to the sensor. Such points are vital for ensuring that each point lies in its correct order within the row, such that each column of data contains points collected in the same packet. In addition, the ring number of each point is not often not provided, further increasing the difficulty of the problem.

If you have access to a Velodyne LiDAR, the "velodyne_pointcloud" driver package (O'Quin *et al*., 2019) provides a Velodyne point cloud type that includes the ring number for each point, along with the expected XYZ information. This driver provides point clouds with points in packet order (i.e., column by column), which makes it straight-forward to identify missing LiDAR pulses that did not return to the sensor, and populate the correct row/column positions with NaN points. This is a significantly easier problem to solve than retroactively organizing existing data sets. For these reasons, it was decided that the best way to collect data sets fulfilling these requirements was to use a Velodyne HDL-32 LiDAR and robotic platform that was already available for this study. The robot in question is shown in Figure 3. It consists of a Clearpath Robotics "Jackal" UGV that has been upgraded with a MicroStrain 3DM-GX5-25 IMU, Trimble Zephyr 2 GNSS antenna, Trimble R7 + Net R9 GNSS RTK system and a Velodyne HDL-32e. All sensors have been mounted on a custom-built frame. The Velodyne HDL-32e has been mounted at a 10° angle, so that its forward field of view is 0° to −40° from the horizontal, and its rear field of view ranges from +20° to −20°.

The Velodyne HDL-32e is configured to spin at 600 RPM, producing clouds at 10 Hz. Each cloud contains approximately 6000 points, spread evenly between 32 individual scan lines. By necessity, the Zephyr 2 GNSS antenna has been mounted high on the robot, and the strut it is mounted atop creates a persistent LiDAR shadow to the left of the robot.

FIGURE 3 Robotic platform used for data collection in this study

The robot is controlled by the robot operating system (specifically ROS Indigo). Data was stored in a ROS "bag" file and processed offline on a Dell Precision M3800 laptop. The M3800 has a Intel Core i7-4712HQ 2.30 GHz processor and used an Ubuntu 14.04 operating system. Each experiment was written in C++11 and compiled with CMake version 3.12.1 using GCC and G++ versions 4.8.4. All cloud processing, including registration, was done with the open-source Point Cloud Library (PCL) Rusu and Cousins (2011), version 1.8.1.

Each data set was collected by slowly driving the robot through each environment, and then saving a series of sequential scans as individual point cloud files. Three different environments were selected for different reasons. The Carpark data set provides an almost entirely planar environment, well suited to any form of registration. The Forest data set by comparison is almost entirely unstructured, as the majority of scan points have returned from grass, bushes, and trees. The Garage data set contains a mixture of planar and unstructured features. These data sets were specifically chosen to provide a range of environments from easy to challenging for the registration algorithms. These data sets are shown in Figure 4. The Garage and Carpark data sets consist of 24 clouds, and the Forest data set of 22. Each cloud contains between 67,200 and 67,800 points.

Note that in each data set, no moving objects were present in the scene, and the scans were collected with the robot moving at a slow speed (approx. 0.25 m/s). This was done to ensure that the scans did not experience significant warping due to the movement of the robot, and so that moving objects in the scene did not interfere with registration by creating poor point correspondences.

3.2 | Ground-truth point cloud creation

The error metrics and methodology of this study rely on knowing the "ground-truth" position of every cloud in each data set. To calculate this, every cloud in each of the three data sets was registered to a highly accurate "reference map" collected using a Trimble SX10 Total Station, which is accurate to 2.5 mm at 100 m (Trimble Inc, n.d.). This is the method proposed in Sprickerhof et al. (2009), and similar to what is used in Wulf et al. (2008). Every cloud has been aligned with the reference map using a combination of manual positioning and PICP in CloudCompare.

This process converts the coordinates of each cloud to be relative to the reference frame that the SX10 cloud is expressed in, which is an arbitrary local reference frame. But as discussed in our previous work (Young et al., 2019), registration conducted when the clouds are far from the origin can cause numerical loss of significance, which can significantly degrade the performance of the algorithm. To avoid this, every cloud is demeaned such that the first cloud in the sequence becomes the origin of the coordinate system. That is, the coordinates of every cloud after the first is transformed to be relative to the first. Figure 4 shows the data sets, with the Velodyne scans coloured red, and the ground-truth SX10 scan coloured grey.

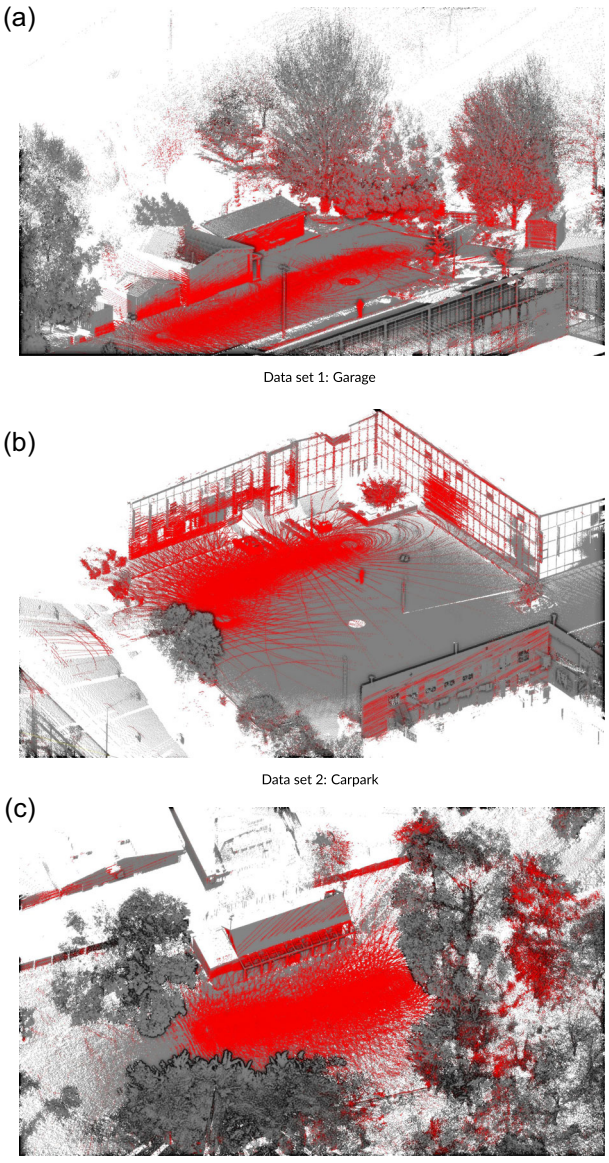


FIGURE 4 Data sets used in this study. Grey points come from a Trimble SX10 Scanning Total Station, red points come from one of several ground-truth HDL-32 scans that has been registered to the SX10 cloud; (a) Data set 1: Garage; (b) Data set 2: Carpark; (c) Data set 3: Forest

3.3 | Error metrics

Each data set used in this study consists of n clouds and their associated poses $\{A_i, \mathbf{P}_i | i \in 1...n\}$, where each pose marks the position and orientation of the sensor (HDL-32) within its respective cloud. Poses are represented by a 4×4 transformation matrix consisting of rotation $\mathbf{R}_i$ and translation $\mathbf{t}_i$. Since the clouds are in their ground-truth positions, the poses are denoted $\mathbf{P}_i^{tru}$.

With reference to Figure 5, most robotic systems model the robots trajectory as a series of single poses ($\mathbf{P}_i$, $\mathbf{P}_{i+1}$, $\mathbf{P}_{i+2}$, ...). In reality, any given pose $\mathbf{P}_i$ can have many different values for its position and orientation. This is why many systems model the pose probabilistically as a mean state with an accompanying covariance matrix to represent the distribution of possible states. In a full SLAM system, the initial placement of the next cloud (denoted $\mathbf{P}_i^{est}$) is based on the prior registration result as well as other sensor data. If the system uses VO, then only the prior registration result is used. In both cases the estimate can drift progressively further from its true value $\mathbf{P}_i^{tru}$. This model is illustrated in Figure 5a.

Alternatively, when SLAM or mapping applications use absolute positioning systems such as GNSS, the error from previous registration does not accumulate. In this case, it is more appropriate to model each estimated pose as an error-added copy of its true position $\mathbf{P}_i^{err}$, as shown in Figure 5b.

Depending on the experiment and methodology used, the estimated or error-added cloud is registered to the previous cloud. The result is a cloud and pose that have been transformed by the output of Equation (3), for example, $\mathbf{P}_i^{reg} = \mathbf{T}_{GICP}\mathbf{P}_i^{err}$ or $\mathbf{P}_i^{reg} = \mathbf{T}_{GICP}\mathbf{P}_i^{est}$.

In all experiments in this paper, the registration error is defined as the difference between a clouds true and registered position. Specifically the transform that expresses the registered pose relative to the true pose, which is defined as follows:

$$\mathbf{E}_i = \begin{bmatrix} \mathbf{R}_i^E & \mathbf{t}_i^E \\ 000 & 1 \end{bmatrix} = (\mathbf{P}_i^{tru})^{-1}\mathbf{P}_i^{reg}. \tag{6}$$

The translation component of this matrix is then just the Euclidean distance between them, which is the same translation error metric used in other studies (Pomerleau et al., 2013; Sprickerhof et al., 2009; Wulf et al., 2008):

$$e_i^t = |\mathbf{t}_i^{tru} - \mathbf{t}_i^{reg}| = |\mathbf{t}_i^E|. \tag{7}$$

When quantifying the rotation error, we use the metric proposed by Pomerleau et al. (2013), which is the angle expressed by an axis-angle representation of the error transforms rotation matrix:

$$e_i^r = arccos\left(\frac{trace(\mathbf{R}_i^E) - 1}{2}\right). \tag{8}$$

Also like Pomerleau et al. (2013), we calculate the median value where multiple data points are collected. The median is more robust to outliers than alternative metrics like the mean or root-mean-square. And as the results in Sections 4.1 show, the unmodified GICP algorithm is prone to producing outliers when given a poor guess for the initial alignment.

The error metrics above quantify how accurate any given registration operation is. But they do not indicate how long that method remains accurate in the case where different aggregation methods are compared. The simplest way to quantify this is to set a threshold, and identify when the positional accuracy of a method exceeds it.

For this study, we set the positional accuracy threshold at 0.25 m, as this is the lowest horizontal accuracy of the R7 GNSS system (when in differential mode). We set the rotational threshold at 1.5° as this is the lowest stated heading accuracy of the

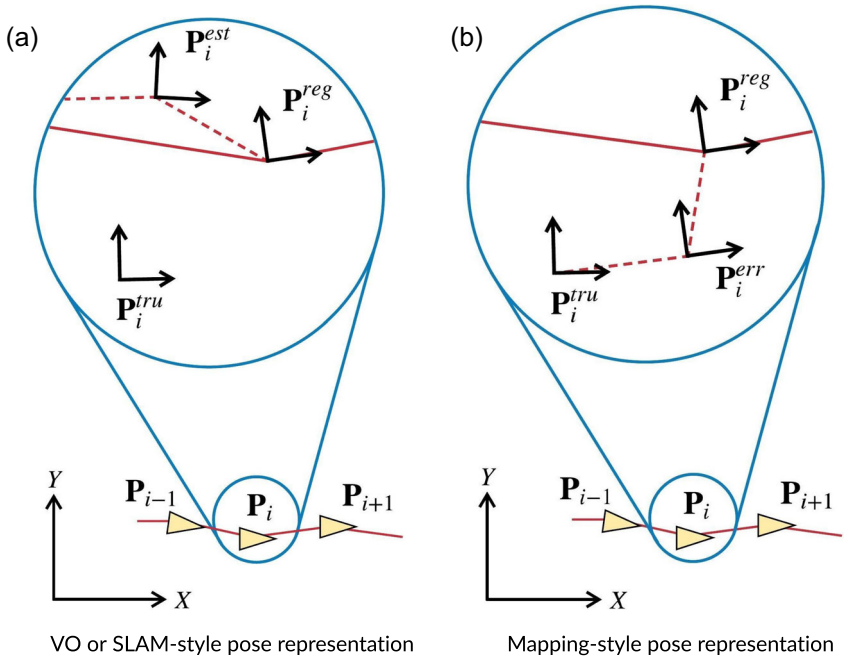


FIGURE 5 Two-dimensional example of reference frame notation used in this study. Solid red lines represent the final trajectory, while dashed-red lines represent the links between related poses. Each pose is given relative to a common coordinate system; (a) VO or SLAM-style pose representation; (b) Mapping-style pose representation. SLAM, Simultaneous localization and mapping; VO, visual odometry

3DM-GX5-25 IMU. These thresholds then indicate the point beyond which it would be more accurate to use sensor data to localize the clouds, rather than registration. A position error threshold of 0.25 m is also the same as the "strict" threshold used in Holz and Behnke (2014).

4 | EXPERIMENTS

4.1 | Experiment 1: GICP versus MGICP

To compare the accuracy of GICP versus MGICP, each algorithm is used to sequentially register each cloud in each data set using the keyscan aggregation method. That is, for every cloud A_i in a set of n clouds $\{A_i | i \in 2...n\}$, A_i is registered to the first cloud in the set A_k, $k = 1$. But as all the clouds are already correctly aligned with each other, uniform random error is added first. So for each cloud pair, a copy of the source cloud A_i is made. Then error is added to it *and* it's corresponding pose (now $\mathbf{P}_i^{err}$). The error-added cloud is then registered to generate $\mathbf{P}_i^{reg}$ as per Figure 5a. Following the methodology in Segal et al. (2009), the error is set to a uniformly distributed translation and rotation with bounds of ±1.5 m and ±15°, respectively.

The translation and rotation error of the registered result are computed as per Section 3.3. This process is repeated 50 times for each cloud, with a freshly generated error each time. The distribution of the results form the violins shown in Figure 6. Simply put, the width of each violin represents the distribution of those 50 results at each distance. So the wider a section of a violin is, the greater the distribution of results at that point. The long, thin end to most violins shows the most extreme outlying points. In this way, violin plots are more visually descriptive than a simple median with error bars, and they can also represent bimodal distributions. The median translation and rotation error for each set of cloud pair results is also calculated and plotted as a line.

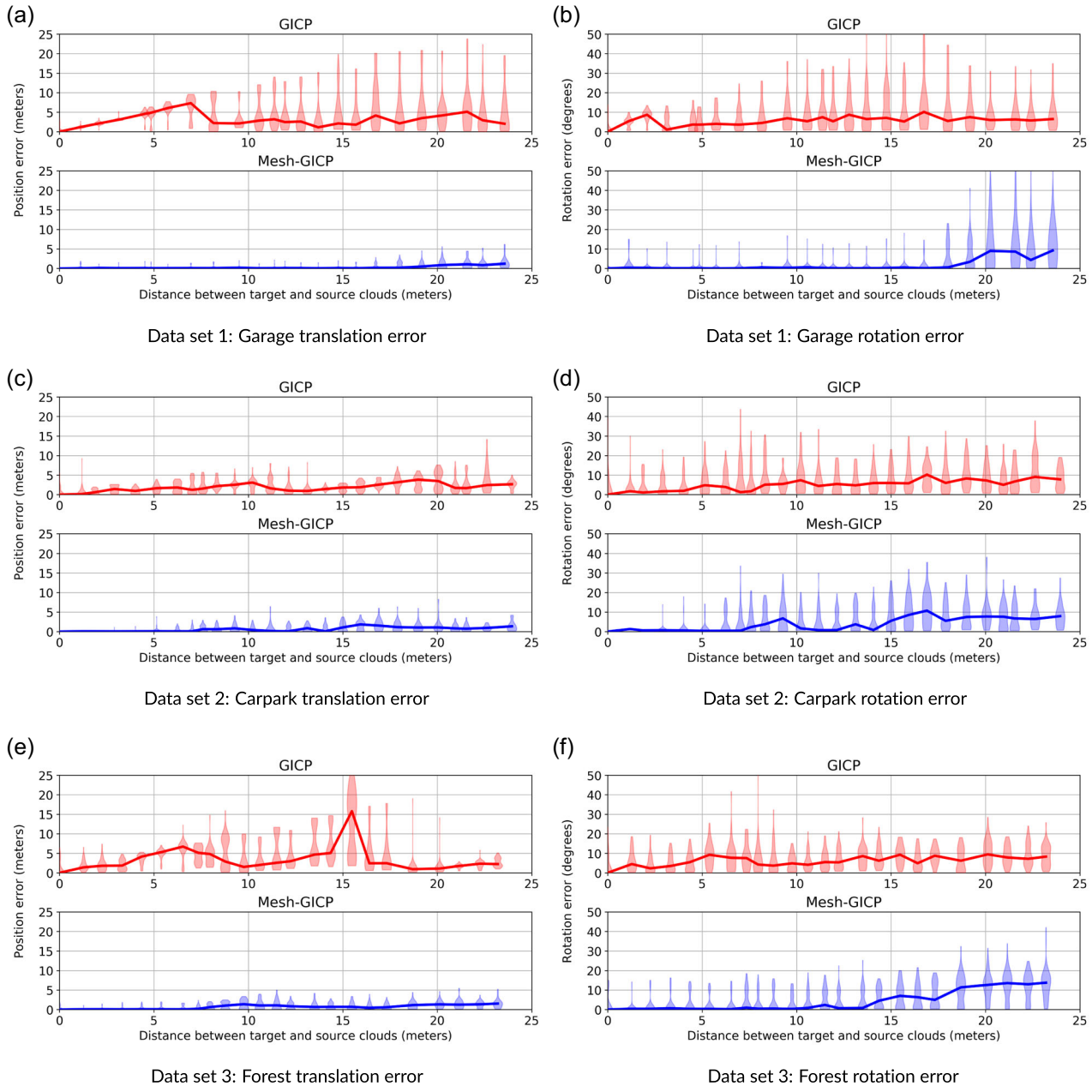
4.2 | Experiment 2: Aggregation methods with MGICP and VO-style initial cloud placement

To compare the effectiveness of each aggregation method, MGICP alone is used. Having already described each aggregation method in Section 2.3, we now apply the ^{tru}, ^{err}, ^{reg} notation to more formally describe how each method is implemented in this test:

(1) Pairwise: $A_i^{est} \Rightarrow A_{i-1}^{reg}$.
(2) Metascan: $A_i^{est} \Rightarrow M = \left\{A_j^{reg} | j \in 1...i-1\right\}, j > 1$.
(3) Keyscan: $A_i^{est} \Rightarrow A_k^{tru}, \{A_i^{est} | i \in 2...n\}, k = 1$.

This follows the unconstrained VO-style pose system, where the initial placement of each pose is determined by the previous registration. This allows the registered cloud to deviate from its true position without bound. Each initial registration of the first cloud is always to its ground-truth version. That is, $A_1^{err} \Rightarrow A_1^{tru}$ and again, as in the previous section, for each cloud the error is calculated between $\mathbf{P}_i^{err}$ and $\mathbf{P}_i^{tru}$. The results of this test are shown in Figure 7.

Note that point cloud registration is an entirely deterministic operation. So as this test does not add randomly generated error, the results for each registration operation are identical. This is why Figure 7 shows only the position and rotation error lines and not a full violin graph.



Data set 1: Garage translation error

Data set 1: Garage rotation error

Data set 2: Carpark translation error

Data set 2: Carpark rotation error

Data set 3: Forest translation error

Data set 3: Forest rotation error

FIGURE 6 Experiment 1 results: Violin plots showing the GICP versus MGICP error. Each violin shows the distribution of 50 registration operations. GICP results are red, MGICP are blue. Plotted lines indicate the calculated median error; (a) Data set 1: Garage translation error; (b) Data set 1: Garage rotation error; (c) Data set 2: Carpark translation error; (d) Data set 2: Carpark rotation error; (e) Data set 3: Forest translation error; (f) Data set 3: Forest rotation error. GICP, generalized iterative closest point; MGICP, mesh generalized iterative closest point

4.3 | Experiment 3: Aggregation methods with MGICP and mapping-style initial cloud placement

The third experiment is similar to the second, except that it uses mapping-style poses. So that instead of relying on the previous registration, the initial pose of each cloud is determined by adding random error to the ground-truth cloud. Again the aggregation methods are listed here for clarity:

(1) Pairwise: $A_i^{err} \Rightarrow A_{i-1}^{reg}$.
(2) Metascan: $A_i^{err} \Rightarrow M = \left\{A_j^{reg} | j \in 1...i-1\right\}, j > 1$.
(3) Keyscan: $A_i^{err} \Rightarrow A_k^{tru}, \{A_i^{err} | i \in 2...n\}, k = 1$.

Because the error can be determined randomly, multiple samples were collected. Each sample consisted of one complete registration trajectory, that is, one complete sequence of registration operations from cloud A_0 to cloud A_n. Every creation of pose $\mathbf{P}_i^{err}$ was done with

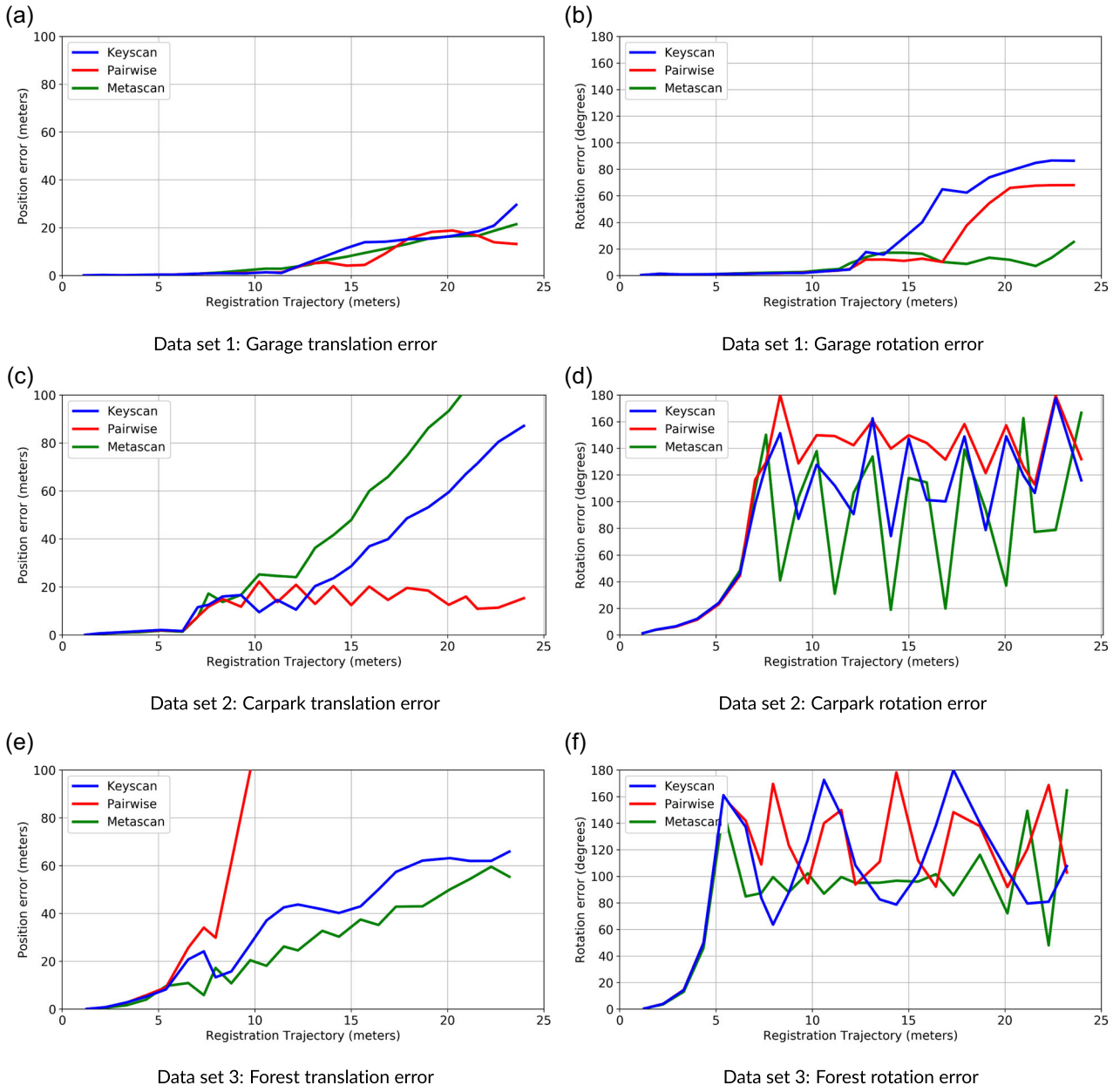


FIGURE 7 Experiment 2 results: Aggregation error when using VO-style estimated initial positions. Note that no violins are shown as this is an entirely deterministic test; (a) Data set 1: Garage translation error; (b) Data set 1: Garage rotation error; (c) Data set 2: Carpark translation error; (d) Data set 2: Carpark rotation error; (e) Data set 3: Forest translation error; (f) Data set 3: Forest rotation error. VO, visual odometry

a freshly generated uniform random error. As with Experiment 1, 50 samples were collected for each aggregation method, and used to generate a violin plot, where each violin represents the error distribution for that source cloud (Figure 8).

5 | DISCUSSION

5.1 | GICP versus MGICP accuracy

The results of the GICP versus MGICP experiment expand on the work of Holz and Behnke (2014, 2016) and Razlaw et al. (2015) by showing that MGICP is superior to GICP at a wide range of distances between the source and target cloud. MGICP outperforms GICP in both position and rotation accuracy. By collecting multiple samples with different error, the results also clearly highlight that MGICP is also significantly more robust to error. As the results in Figure 6 show, not only are the median translation and rotation errors lower for MCIP, but the distributions of the errors are also much tighter and closer to the median. Ordinary GICP by comparison can have a wide distribution, sometimes with clusters of results away from the median.

In addition, the outlying GICP errors are much more extreme, as evident by the maximum height of each violin of GICP data. These outliers indicate that even when the target and source cloud are close together,

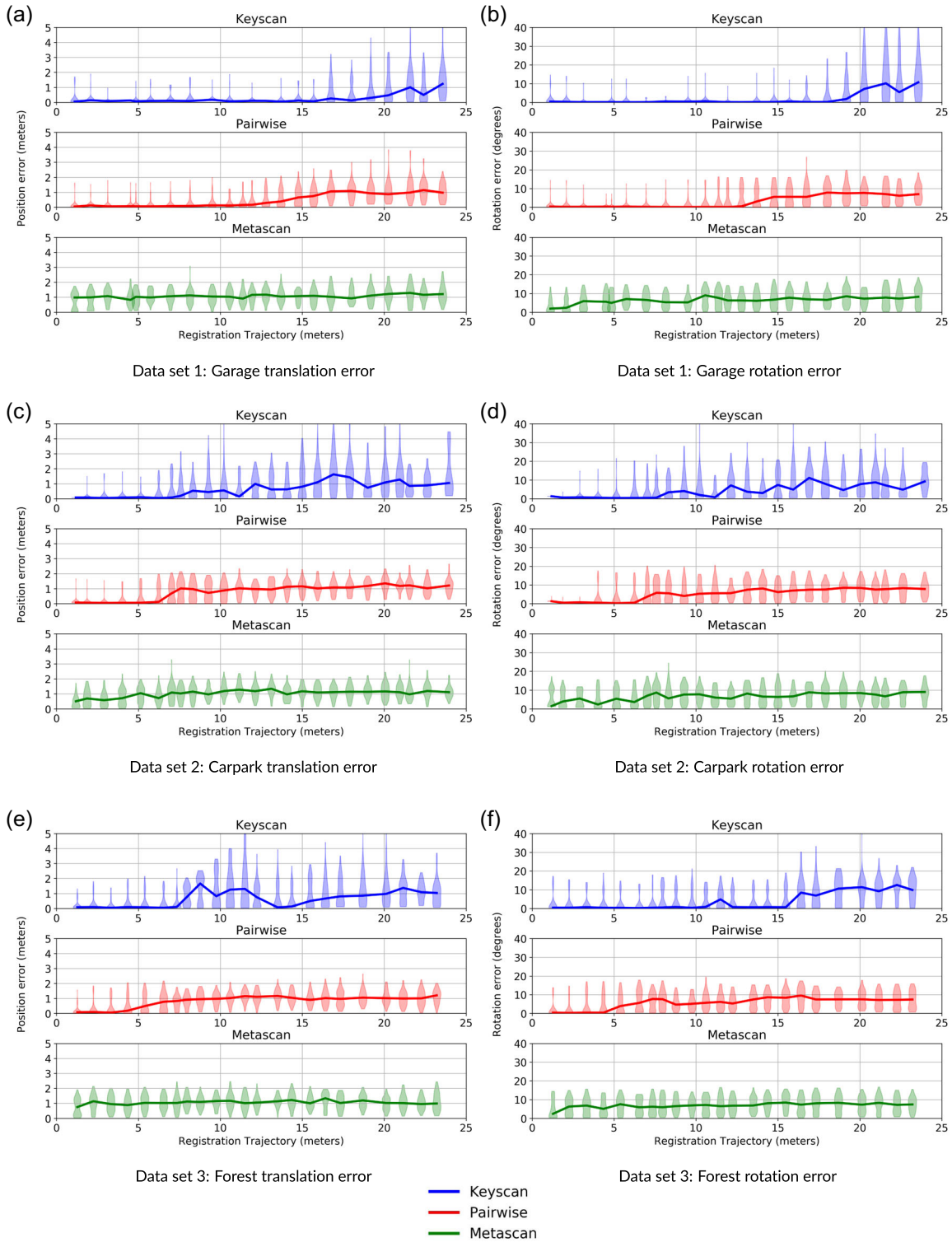


FIGURE 8 Experiment 3: Aggregation error when using mapping-style estimated initial positions. Each violin shows the distribution of 50 registration operations. Plotted lines indicate the calculated median error; (a) Data set 1: Garage translation error; (b) Data set 1: Garage rotation error; (c) Data set 2: Carpark translation error; (d) Data set 2: Carpark rotation error; (e) Data set 3: Forest translation error; (f) Data set 3: Forest rotation error

ordinary GICP can leave the source cloud further from the target than its initial error-added position. Although in most VO or SLAM systems, registered clouds will likely be close together, the distance between source and target clouds is important for the keyscan aggregation method.

The median distances at which the positional accuracy threshold (0.25 m) and rotational accuracy threshold (1.5°) are exceeded by each method are shown in Table 1. They clearly show that that MGICP remains accurate at distances approximately 4–17 times

TABLE 1 Median trajectory length at which the position and rotation thresholds are exceeded for Experiment 1

	Position (m)			Rotation (°)		
Data set	Garage	Carpark	Forest	Garage	Carpark	Forest
Mesh-GICP	19.18	7.59	7.96	19.18	7.59	11.50
GICP	1.12	1.87	1.28	1.12	1.19	1.28

Note: Position error threshold: 0.25 m. Rotation error threshold: 1.5°.

greater than ordinary GICP, which loses significant accuracy after only 1 m between the target and source cloud.

5.2 | GICP versus MGICP computation time

As already discussed in Section 2.2, the inherent nature of organized clouds makes certain point cloud processing operations significantly faster. In the context of this paper, the MGICP methodology can generate covariances directly from a mesh that was created by connecting adjacent points in the same row or column. The ordinary GICP method however uses a K-D tree to find the nearest neighbours of every point, before then computing covariances. This is a significantly slower process, as illustrated by Table 2, which shows the computation time take to complete experiment 1.

There are some caveats that must be added to this table. The first is obviously these times will be specific to the computer and development environment in which the experiment was run. In particular, the significant processing times of the GICP method may reflect more on the limitations of the hardware than the GICP algorithm.

The second caveat is that the computation time of any one registration operation will depend on the complexity of the scene, the amount of overlap between the registering clouds, and their initial offset from each other. So while the absolute values may not have a large degree of significance, the *relative* computation time of MGICP highlights how much faster it is a registration method, taking approximately 9% of the computation time taken by ordinary GICP.

5.3 | Aggregation method accuracy

When using VO-style initial pose estimation, the biggest source of error does not come from the aggregation methods themselves, but the simple fact that any registration error accumulates. In Wulf et al. the position errors can be reduced when the LUM pose graph recognizes a closed loop and adjusts the trajectory. But in this experiment, the registration error accumulates before any definitive differences between aggregation methods can be determined, which limits its ability to show differences between aggregation methods. This is a problem for other comparisons, such as that shown in Wulf et al., and is the primary motivation for conducting Experiment 3 where registration error cannot accumulate. When using the mapping-style initial placement of each cloud, the differences between aggregation methods become more apparent. Metascan point cloud aggregation performs the worst in both position and rotation accuracy because only the source cloud is organized, and has covariances computed from a mesh. The target cloud (the metascan) is unorganized and any registration errors accumulate when each cloud is added to the metascan. This is also why the distribution of errors are more widely spread. Table 3 reflects this with the metascan method never achieving an accuracy below the thresholds.

Pairwise aggregation retains the benefit of having both source and target covariances derived from an organized mesh. But because the registered position of each cloud is influenced by the position of the previous cloud, a significant failure in the registration process compromises the accuracy of every subsequent operation. This is evident from the abrupt increases in error in the median line of the pairwise results in Figure 8.

Keyscan aggregation is the most accurate aggregation for mapping because it makes full use of the MGICP algorithm and keeps each registration operation independent. The limitation is that as distance between source and target cloud increases, the amount of overlap decreases. And beyond a certain distance, registration will obviously fail completely. Table 3 shows that this distance can be as low as 8 m or as high as 19 m, depending on the data set. This is why keyscan aggregation is not used in SLAM. But in mapping applications where absolute position information is available, keyscan aggregation would be the superior choice for creating accurate local point cloud maps.

5.4 | Limitations

In the years since the publication of the original work on GICP several other studies have iterated further on the original algorithm to improve its accuracy and efficiency. Some have already been mentioned in Sections 1–2.3. Other studies were published after the work conducted for this study, or too late to be incorporated into it. One of particular note is the study by Vlaminck et al. (2018). This method adds a surface reconstruction step, and then projects the cloud onto

TABLE 2 Computation times for Experiment 1: GICP versus MGICP

	Total time (min:sec)			Mean per-cloud time (sec)		
Data set	Garage	Carpark	Forest	Garage	Carpark	Forest
Mesh-GICP	10:29	06:42	11:07	0.52	0.34	0.61
GICP	124:05	69:37	126:01	6.20	3.48	6.87

Note: Note that "total time" includes the repetition of 50 registration operations for each cloud.

TABLE 3 Median trajectory length at which the position and rotation thresholds are exceeded for Experiment 3

	Position (m)			Rotation (°)		
Data set	Garage	Carpark	Forest	Garage	Carpark	Forest
Keyscan	16.75	8.33	7.96	19.18	8.33	11.50
Pairwise	12.78	7.03	5.39	13.70	7.03	5.39
Metascan	1.12	1.19	1.28	1.12	1.19	1.28

Note: Position error threshold: 0.25 m. Rotation error threshold: 1.5°.

the resulting surface. The weighted distributon of these projected points is then included in the calculation of covariance matrices $\mathbf{C}_i^A$ and $\mathbf{C}_i^B$. It also uses Voxel Grid Dilation to fill holes in the point cloud.

Another paper of note is Koide et al. (2020), which introduces voxelized generalized ICP (VGICP). This algorithm is presented as a hybrid of the GICP and NDT. The paper reports that VGICP has an accuracy similar to that of GICP, but computation speeds of up to 120 Hz when optimized and run on a GPU, making it even faster than MGICP. The work of this study is limited in that it only compares the base GICP method with one variant. Future work could compare both the GICP and MGICP methods with some of these more recent variations like VGICP.

Another limitation is in the ability to compare algorithmic results. This study, because of the dependency of MGICP on organized clouds, uses three bespoke data sets. As previously discussed in Section 3.1, there are many online public data sets which provide unorganized 3D point clouds. These online data sets are considerably larger, more varied, and are widely used in academia. Some like the KITTI data set already have a substantial body of research associated with them.

For these reasons the ability to compare the results in this paper to prior work is limited. And due to their size, the attractiveness of the data sets used here is limited compared to these established online data sets.

However, any 3D point cloud could be theoretically organized by converting Cartesian coordinates to polar, and organizing the points by azimuth and altitude, with identification of missing points. Future work could develop a suitable algorithm for this purpose, and allow existing online data sets to be more easily used with registration methods that rely on organized clouds.

6 | CONCLUSION

This study expands the work of Holz and Behnke (2014, 2016) and Razlaw et al. (2015) by comparing ordinary versus mesh-based GICP with a range of data sets and target/source cloud distances. The results clearly show that MGICP is significantly more accurate, more precise *and* significantly faster at registering sparse point clouds, at a variety of distances between the target and source cloud. MGICP takes approximately 9% of the computation time that GICP takes, and results in a registered cloud that is consistently more accurate in both position and orientation. It is also more accurate at registering clouds that are far apart, remaining as or more accurate than GNSS positioning with up to 15–20 m between registering clouds. Meanwhile GICP can cease to be this accurate with only a few meters difference.

This study also compares different point cloud aggregation methods (pairwise, metascan, and keyscan) with MGICP to determine which is the most accurate at positioning clouds via registration. This comparison is done with two different approaches to initial cloud placement: a VO-type approach where initial placement depends on the previous registration operation, and a mapping-type approach where initial placement is an error-added deviation from the ground truth. The results of these tests show that accumulated registration error in the SLAM scenario exceeds any differences in accuracy between the aggregation methods. While in the mapping scenario, keyscan aggregation outperforms the other methods in both accuracy and precision up to a given distance between source and target clouds.

ACKNOWLEDGEMENTS

Thank you to Jakub Horejsi for his tutelage and help in acquiring the SX10 clouds. Callaghan Innovation and Trimble Inc., Contract Number: TNNX1601.

REFERENCES

Besl, P. J., & McKay, N. D. (1992). Method for registration of 3-d shapes. In *Sensor fusion IV: Control paradigms and data structures* (Vol. 1611, pp. 586–606). International Society for Optics and Photonics.

Borrmann, D., Elseberg, J., Lingemann, K., Nüchter, A., & Hertzberg, J. (2008). Globally consistent 3d mapping with scan matching. *Robotics and Autonomous Systems*, *56*(2), 130–142.

Bouaziz, S., Tagliasacchi, A., & Pauly, M. (2013). Sparse iterative closest point. In *Computer graphics forum* (Vol. 32, pp. 113–123). Wiley Online Library.

Chen, Y., & Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, *10*(3), 145–155.

Donoso, F., Austin, K. J., & McAree, P. R. (2017a). How do icp variants perform when used for scan matching terrain point clouds? *Robotics and Autonomous Systems*, *87*, 147–161.

Donoso, F., Austin, K. J., & McAree, P. R. (2017b). Three new iterative closest point variant-methods that improve scan matching for surface mining terrain. *Robotics and Autonomous Systems*, *95*, 117–128.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. In' *International Journal of Robotics Research (IJRR)*.

Georgiev, K., Creed, R. T., & Lakaemper, R. (2011). Fast plane extraction in 3d range data based on line segments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3808–3815). IEEE.

Grant, W. S., Voorhies, R. C., & Itti, L. (2013). Finding planes in lidar point clouds for real-time registration. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4347–4354). IEEE.

Holz, D., & Behnke, S. (2014). Registration of non-uniform density 3d point clouds using approximate surface reconstruction. In *ISR/Robotik 2014; 41st International Symposium on Robotics* (pp. 1–7). VDE.

Holz, D., & Behnke, S. (2016). Mapping with micro aerial vehicles by registration of sparse 3d laser scans. In *Intelligent Autonomous Systems* (Vol. 13, 1583–1599). Springer.

Koide, K., Yokozuka, M., Oishi, S., & Banno, A. (2020). Voxelized gicp for fast and accurate 3d point cloud registration. *EasyChair Preprint* (Vol. 2703).

Korn, M., Holzkothen, M., & Pauli, J. (2014). Color supported generalized-icp. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)* (Vol. 3, pp. 592–599). IEEE.

Marton, Z. C., Rusu, R. B., & Beetz, M. (2009). On fast surface reconstruction methods for large and noisy point clouds. In *2009 IEEE international conference on robotics and automation* (pp. 3218–3223). IEEE.

Mavridis, P., Andreadis, A., & Papaioannou, G. (2015). Efficient sparse icp. *Computer Aided Geometric Design*, *35*, 16–26.

O'Quin, J., Khandelwal, P., Vera, J., & Pütz, S. (2019). velodyne_pointcloud (Version 1.5.2). http://wiki.ros.org/velodyne_pointcloud

Pathak, K., Birk, A., Vaskevicius, N., Pfingsthorn, M., Schwertfeger, S., & Poppinga, J. (2010). Online three-dimensional slam by registration of large planar surface segments and closed-form pose-graph relaxation. *Journal of Field Robotics*, *27*(1), 52–84.

Pomerleau, F., Colas, F., Siegwart, R., & Magnenat, S. (2013). Comparing ICP variants on real-world data sets. *Autonomous Robots*, *34*(3), 133–148.

Pomerleau, F., Liu, M., Colas, F., & Siegwart, R. (2012). Challenging data sets for point cloud registration algorithms. *The International Journal of Robotics Research*, *31*(14), 1705–1711.

Razlaw, J., Droeschel, D., Holz, D., & Behnke, S. (2015). Evaluation of registration methods for sparse 3d laser scans. In *2015 European Conference on Mobile Robots (ECMR)* (pp. 1–7). IEEE.

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling* (pp. 145–152). IEEE.

Rusu, R. B., & Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation* (pp. 1–4). IEEE.

Segal, A., Haehnel, D., & Thrun, S. (2009). Generalized-icp. In *Robotics: Science and systems*, Seattle, WA (Vol. 2, p. 435).

Servos, J., & Waslander, S. L. (2014). Multi channel generalized-ICP. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3644–3649). IEEE.

Sprickerhof, J., Nüchter, A., Lingemann, K., & Hertzberg, J. (2009). An explicit loop closing technique for 6d slam. In *ECMR*. Mlini/Dubrovnik, Croatia (pp. 229–234).

Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*.

Trimble Inc, T. (n.d.). Trimble sx10 scanning total station. Technical Report. https://geospatial.trimble.com/sites/geospatial.trimble.com/files/2019-03/Datasheet%5C%20-%5C%20SX10%5C%20Scanning%5C%20Total%5C%20Station%5C%20-%5C%20English%5C%20A4%5C%20-%5C%20Screen.pdf

Viejo, D., & Cazorla, M. (2007). 3d plane-based egomotion for slam on semi-structured environment. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2761–2766). IEEE.

Vlaminck, M., Luong, H., & Philips, W. (2018). Surface-based GICP. In *2018 15th Conference on Computer and Robot Vision (CRV)* (pp. 262–268). IEEE.

Wulf, O., Nüchter, A., Hertzberg, J., & Wagner, B. (2008). Benchmarking urban six-degree-of-freedom simultaneous localization and mapping. *Journal of Field Robotics*, *25*(3), 148–163.

Xiao, J., Adler, B., & Zhang, H. (2012). 3d point cloud registration based on planar surfaces. In *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)* (pp. 40–45). IEEE.

Young, M., Pretty, C., Agostinho, S., Green, R., & Chen, X. (2019). Loss of significance and its effect on point normal orientation and cloud registration. *Remote Sensing*, *11*(11), 1329.

Zhang, J., & Singh, S. (2017). Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, *41*(2), 401–416.

How to cite this article: Young, M., Pretty, C., McCulloch, J., & Green, R. (2021). Sparse point cloud registration and aggregation with mesh-based generalized iterative closest point. *J Field Robotics*, 38, 1078–1091. https://doi.org/10.1002/rob.22032

APPENDIX

TABLE A1 Parameters for mesh creation with PCL's Greedy Projection Triangulation algorithm

Parameter	Value	Unit
Search radius	2.0	meter
μ	2.5	–
Maximum nearest neighbours	1000.0	–
Maximum surface angle	45.0	degrees
Minimum angle	10.0	degrees
Maximum angle	120.0	degrees
Normal consistency	False	–

TABLE A2 Parameters for GICP and MGICP

Parameter	Value	Unit
Max iterations	200	–
Fitness epsilon	-Inf	–
Max correspondence distance	1.0	meter
Transform epsilon	0.0005	–
Optimizer iterations	20	–
Correspondence randomness	5	–
Rotation epsilon	0.002	–