Efficient Large-Scale Point Cloud Registration Using Loop Closures

Takaaki Shiratori[1] Jérôme Berclaz[2] Michael Harville[2] Chintan Shah[2]
Taoyu Li[1] Yasuyuki Matsushita[1] Stephen Shiller[2]

[1]Microsoft Research [2]Microsoft, Bing Maps

Abstract

Alignment of many 3D point clouds, possibly captured by multiple devices at different times, is a critical step for increasingly popular applications such as 3D model construction and augmented reality. For very large data sets, traditional methods such as ICP can become computationally intractable, or produce poor results. We present an efficient method for accurately aligning very large numbers of dense 3D point clouds, and apply it to a city-scale data set. The method relies on the novel combination of 1) partitioning the point clouds based on loop structures detected across a combined network of all device capture paths, and 2) making use of the loop closure property to accurately align point clouds within each sub-problem. Final global alignment of the loop-based results is formulated as a least squares optimization with closed form solution. Experimental results are shown for aligning 3D points across the entire city of San Francisco with centimeter-scale accuracy, via an efficient parallelized architecture.

1. Introduction

With recent advances in depth sensing devices and methods, large 3D point clouds have become an increasingly common source of data for computer vision tasks such as 3D model reconstruction, pose estimation, and object recognition. In most such applications, the point cloud data is not obtained instantaneously, but rather requires sensor motion over time, and perhaps use of multiple sensors or multiple sweeps of a single sensor. Spatially registering these point clouds captured at different times and/or with multiple devices becomes a key problem that must be solved prior to further data analysis.

While many 3D modeling techniques show good results for objects and environments of a few meters in size [1, 11, 9], modeling at the larger scales of indoor environments and entire cities remains technically challenging. In these cases, many point cloud "frames" captured along one or more complex sensor paths must all be placed in a consistent 3D coordinate system, and straightforward application of popular approaches such as Iterative Closest Point (ICP) algorithm [5] and its variants leads to many small frame-to-frame alignment errors that often accumulate to produce gross distortions in the final result. At the same time, computation and memory requirements can easily become infeasible, particularly when methods jointly optimize alignment of many point clouds rather than operating on them pair-wise.

In this paper, we present a method for aligning very large sets of 3D point clouds, potentially obtained by many capture devices and along multiple capture paths, in a manner that is both accurate and highly parallelizable for efficient computation. From an initial estimate of the sensor paths, we construct a 3D graph of their intersection and connectivity, and decompose the overall alignment problem into smaller ones based on the *loop closures* that exist in this graph. Each loop may be composed of segments of different device acquisition paths. We pair this decomposition with a novel, local alignment technique called Simultaneous GICP (S-GICP), based on Generalized-ICP [21], that exploits the loop closure property to produce highly accurate *intra-loop* registration results. The individual loops are then combined into a single, consistent point cloud via an *inter-loop* alignment step that reconnects the graph of loops with minimal distortion, according to a least squares optimization with closed form solution. We also show how to constrain this last step with high-confidence locations within the initial device capture path estimates, thereby producing a final result that is better anchored, for example, to some external reference coordinate system.

There are several important benefits to our approach. First, it is well known that simultaneous registration with loop closure significantly improves point cloud quality, and our method attempts to ensure that all point clouds benefit from this by incorporating them into separate loop sub-problems. Second, the S-GICP method effectively re-estimates the capture path segments within the loop, allowing them to non-rigidly deform in order to jointly optimize alignment of all the points. Finally, intra-loop registration can be applied to each of the loops in parallel, thereby en-

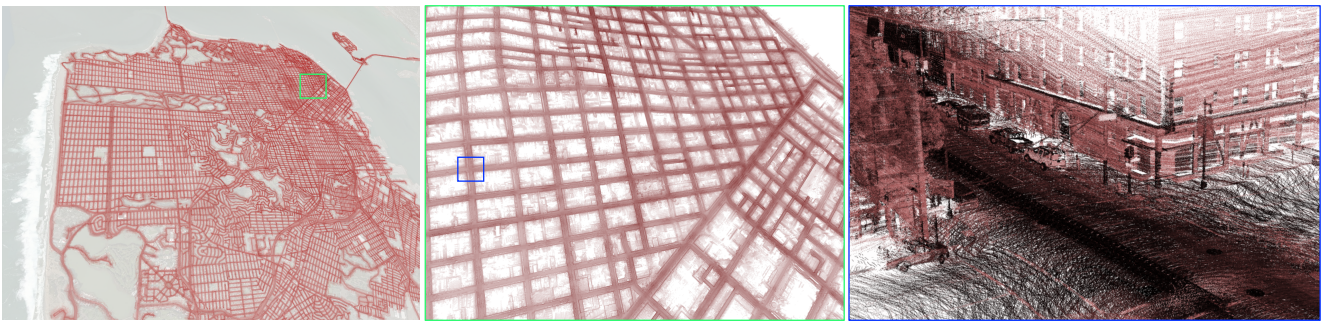


Figure 1. Our method can register very large city-scale point clouds, such as the San Francisco dataset shown in these pictures.

abling significant reduction of computation time and complexity compared with conventional simultaneous registration algorithms.

We illustrate the proposed method in the context of city-scale 3D modeling, from point clouds acquired from multiple vehicles moving throughout the city streets (Fig. 1). In this scenario, point clouds are obtained from LiDAR sensors mounted on the vehicles, and initial estimates of the capture paths are derived from onboard GPS and IMU sensors. The graph of capture paths parallels the connectivity of the city streets, and the "loops" found are often city blocks containing several buildings. We show results based on about half a trillion points captured over the entire city of San Francisco, with accuracy on the scale of centimeters, processed on a computer cluster of several thousand nodes in about 4 hours.

2. Prior Work

Given a pair of point clouds without point correspondences, Besl and McKay are the first to introduce the ICP algorithm [5], which iterates search of point correspondences and optimization of transformation parameters (translation and rotation) by minimizing point-to-point distances between the correspondent points until convergence. Many improvements have been made for point correspondence search such as considering normals of each point [7] and sensor-to-point rays [3] and for optimization such as point-to-plane distance [7], approximate plane-to-plane distance [21] and L_p norm ($p < 1$) [6]. Papers by Rusinkiewicz *et al*. [20] and Tam *et al*. [24] evaluated variants of the pair-wise ICP algorithms in the literature.

For the case where multiple point clouds are provided, Chen *et al*. sequentially applied pair-wise ICP to multiple point clouds [7]. Masuda [13] extended Chen *et al*.'s method, and registered a single point cloud incrementally by minimizing distances between points and meshes created from registered point clouds. These sequential approaches are quite inaccurate because of accumulation of pair-wise registration errors. One approach is to extend the ICP algorithm to optimize transformation parameters of all point clouds simultaneously, referred to as *direct approaches*. Bergevin *et al*. [4] and Neugebauer [14] first applied pair-wise ICP to point cloud pairs, and then refined all point clouds simultaneously by minimizing point-to-plane distances. Nishino *et al*. [16] used point-to-point distances with M-estimator to improve robustness. Pfaff *et al*. [17] used a Kalman filter-based sequential ICP for large-scale mapping, and performed simultaneous registration when loops were detected.

More recent work focuses on extracting *features* from a point cloud and minimizing feature distances for multi-view registration, referred to as *feature-based approaches*. Because features enable to establish absolute correspondences between point cloud pairs, the point correspondence update in the ICP framework is not necessary. Gelfand *et al*. [8] extracted volumetric features from surfaces computed from input point clouds. Multi-view registration based on these features provide sufficiently good initial guess for the multi-view ICP algorithms. Zou *et al*. [25] considered Gaussian curvature for surface geometry for feature extraction. Bariya *et al*. [2] extracted scale-dependent/invariant features from 2D normal maps for alignment and recognition.

The other category is to distribute registration errors to point clouds by minimizing coordinate frame errors from pair-wise registration, referred to as *motion averaging approaches*. Pulli [18] introduced the *virtual mate* approach that used transformations estimated with pair-wise registration as constraints, and repeatedly registered point clouds. Sharp *et al*. [22] introduced an analytic solution in this category and minimize coordinate frame errors of rotation and translation separately. Shih *et al*. [23] formulated coordinate frame error minimization as a quadratic programming problem for Lie algebra parameters. Govindu *et al*. [12] introduced a Lie-algebraic averaging methods to distribute coordinate frame errors.

These three approaches have different characteristics in terms of accuracy and computational cost. The latest feature-based approaches are the most efficient and qualitatively accurate, but require input point clouds to be dense and uniform for computing underlying geometry such as

surface and volume. However, such dense and uniform point clouds are not necessarily available, particularly for outdoor scenes, because of the inside-out manner of outdoor depth sensors. Between the other two approaches, the direct approaches consider geometric distances (*e.g.*, point-to-point, point-to-plane), and often achieve better accuracy than the motion averaging approaches, while the direct approaches are computationally more expensive [23]. However, none of the above methods are scalable for very large point clouds: The more point clouds, the higher computational cost. While we take the direct approach for intra-loop registration due to its accuracy, the parallelization capability of our method achieves significant efficiency improvements.

In terms of efficiency, the method of Pylvänäinen *et al*. [19] shares a few similarities to ours, in particular the decomposition of the large registration problem according to city block loops. However, within each loop sub-problem, they use sequential ICP for translational registration that fails to take advantage of the loop closure property, and thus suffers from error accumulation, similarly to Chen *et al*.'s method [7]. Our approach minimizes the approximate plane-to-plane distance with respect to translation and rotation for simultaneous registration. We relax the non-linear properties of this optimization with an alternating optimization, and achieve accuracy with reasonable efficiency.

3. Point Cloud Registration with Loops

Our point cloud data is collected by one or more vehicles outfitted with LiDAR sensors, traveling along multiple overlapping paths through a city. Data along each capture path is divided into local point cloud "frames" , each of which is captured within a small spatio-temporal window. The estimated vehicle location and orientation, derived from on-board GPS/IMU sensors, is also associated with each point cloud frame, and allows them to be approximately aligned in a global coordinate system. Due to GPS signal loss and other factors, alignment errors of up to several meters in location and a few degrees in orientation are often observable where there is spatial overlap between point cloud frames captured by different vehicle drives.

Our method for correcting this error begins by constructing a graph representation of the multiple overlapping vehicle paths, and assigning point cloud frames to edges of the graph (Fig. 2(a)). The graph is segmented into a set of adjoining loops, each of which may be composed of frames from different vehicle capture paths. Next, S-GICP (see Section 3.2) is used to jointly optimize alignment of all frames within each loop (Fig. 2(b)). This *intra-loop registration* step is applied to each loop independently, and makes use of loop closure to produce self-consistent results. Finally, the loop point clouds are aligned via a closed-form, least squares *inter-loop registration* step that also integrates high-confidence GPS/IMU data, to produce a globally consistent and accurate city-scale point cloud (Fig. 2(c)).

3.1. Point Cloud Partitioning

Each node in our graph is a location at which a vehicle path crosses either itself or another path. Edges are created between node pairs that are directly connected (*i.e.*, no intervening nodes) along at least one vehicle's path. The geometric shape of the vehicle path between two directly connected nodes is retained, and these paths are frequently not straight lines between the nodes' respective geographic locations. If multiple drives occurred between two nodes, these path segments are clustered according to their shape, and each cluster becomes a separate edge between the nodes.

The graph can be formed directly from the paths estimated from GPS/IMU data, by first creating nodes where paths converge within a threshold distance from sufficiently different directions, or where a path begins traversal through a location previously visited by itself or another path. Our experiments showed good results with this approach, which is also applicable to other 3D capture scenarios such as indoor environment modeling with an RGBD sensor [15], with initial paths estimated from inertial sensors or techniques such as SLAM. However, to better relate our work to a variety of geospatial information sources, we find it useful to first associate point cloud frames with locations on a known street map of the city, and then form a graph based on the street connectivity.

HMM-based association of sensor paths to a map The shapes of the streets are provided with the map, and are resampled at two-meter spacing to produce candidate point cloud assignment locations. With these candidate locations as the hidden states, an HMM framework performs this assignment independently for each vehicle drive, using observation probabilities based on the distance from the GPS/IMU-based point cloud location estimate and coherence between the local direction of the street and the estimated vehicle path. State transition probabilities are determined by the length of the street route between a pair of locations, thereby encouraging continuity of assignment of a vehicle path along a connected sequence of road links.

Loop enumeration from sensor paths Once the graph of capture paths is constructed, we wish to divide it into a set of loops that, ideally, cannot be further subdivided, do not overlap, and provide complete coverage of the graph. This is related to the problem of finding a minimal cycle basis of an undirected graph, for which algorithms have been proposed in the fields of graph theory and computational geometry [10]. For graphs that can be embedded in a plane, such that every node has a 2D coordinate and no edges cross except at nodes, polynomial time algorithms have been pro-

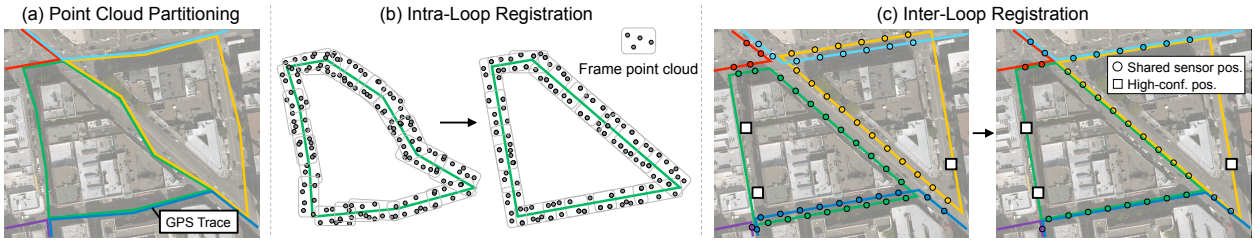


Figure 2. Overview of our approach. (a) We first partition the entire point cloud into loop-based small point cloud sets. (b) Then intra-loop registration refines point clouds for all loops in parallel. (c) Inter-loop registration reconnects the graph of loops based on the refined sensor locations and produces a geospatially registered point cloud.

Algorithm 1 Algorithm for enumerating graph loops

procedure FINDALLLOOPS(G) ▷ G: the graph
  $S \leftarrow$ all edges in G ▷ S: Edges at which to start
  $L \leftarrow \varnothing$ ▷ L: Set of all loops found
  while $S \neq \varnothing$ **do**
    $e \leftarrow dequeue(S)$ ▷ Get next start edge
    $l \leftarrow \varnothing$ ▷ New loop edge set, initially empty
    for each end node n of e **do**
      if FOLLOWNEXTEDGE(G,e,n,l) **then**
        $L \leftarrow L \cup$ TRIMLOOP(l) ▷ Found a loop
        if $e \notin l$ **then** ▷ Start edge not part of it
          $enqueue(S, e)$ ▷ Try it again later
  return L

procedure FOLLOWNEXTEDGE(G,e,n,l)
  if $size(l) > MaxLoopSize$ **then return** $false$
  $l \leftarrow l \cup e$ ▷ Add edge e to loop being built
  LEFTSIDEUSED(e) $\leftarrow true$ ▷ Mark as used
  if CLOSED(l) $\wedge \neg$INVERTED(l) **then**
    return $true$ ▷ Found loop
  for each edge $ee \in$ CLOCKWISEORDER(n, e) **do**
    if $(e \neq ee) \wedge \neg$LEFTSIDEUSED(e) **then**
      $nn \leftarrow$ end node of e such that $n \neq nn$
      if FOLLOWNEXTEDGE(G,ee,nn,l) **then**
        return $true$ ▷ Found loop recursively
  $l \leftarrow l \setminus e$ ▷ No loop found; remove edge from set
  LEFTSIDEUSED(e) $\leftarrow false$ ▷ Free edge for re-use
  return $false$

posed to find a "two-basis" for the graph, where each edge participates in at most two loops. Our problem is not quite this simple, in that the road network is actually 3D, due to overpasses, stacking of highways above surface streets, and other road designs. "Dead-end" streets (only one connected edge), as well as relatively long sections of sparsely connected highways, further complicate our problem.

We therefore propose Algorithm 1, to efficiently partition a graph into a maximum number of loops with minimal overlap. It relies on projecting the 3D graph onto a planar coordinate system, so that an ordering of edges exiting a node, relative to a given incoming edge, can be defined. We use the 2D geospatial latitude and longitude coordinate system, and order edges in a clockwise manner. **FindAllLoops** initiates two depth-first searches (implemented via **FollowNextEdge**) at each graph edge, in the directions of each end node of the start edge. The depth-first search explores subsequent edges according to **ClockwiseOrder**, which results in a preference for taking the left-most available turn at each node. As traversal progresses, **LeftSideUsed** updates a "winged-edge" data structure to indicate that the "left" side of the edge (defined relative to the direction of traversal) is part of a new loop under construction. Edges are bypassed in the exploration if they have previously been incorporated into a loop on their left side. The **Closed** predicate is true when traversal returns to a node that has already been visited in exploration from the current start edge, and **TrimLoop** removes any initial edge sequence prior to the first loop node. It can occur that many left-most available turns during an exploration were actually rightward turns, such that all edges in the final loop have their left side on the exterior of the loop, rather than the interior as expected. We found that exclusion of such loops (accomplished via **Inverted**) greatly improved both the algorithm speed and simplicity. We initially impose a maximum loop length and a constraint that no loop can be self-crossing (i.e. edges crossing over others in the same loop) in **FollowNextEdge** to find all the smallest, simplest loops first, and then slowly raise the maximum and remove the constraint after no more such loops can be found. Our method finds loops such that a near-maximal number of graph edges participate in exactly two loops.

3.2. Intra-Loop Registration

Once loops are determined, our method performs the point cloud registration for each loop. A loop contains a set of point cloud *frames*, and the goal of this step is to align the frames in a loop, and to refine sensor positions accordingly. Our method is built upon the Generalized ICP algorithm [21], which we combine with a simultaneous optimization using loop closure. We call the developed method *Simultaneous Generalized ICP* (S-GICP). Similar to conventional ICP methods, our S-GICP iterates (1) point correspondence search and (2) optimization of transformation

parameters of every frame, until convergence.

For point correspondence search, we use KD-tree-based nearest neighbor search, followed by thresholding for correspondent point distances. The thresholding is an important step to remove unreliable correspondences with large distances, which are likely to be outliers. The threshold parameter is set to 3 meters in all experiments. To reduce the computational cost of point correspondence search, we first perform nearest neighbor search for each frame based on the mean point position, and pair the frames if the distance between frames is less than 14 meters. Then, point correspondence search is performed for the detected frame pairs. We empirically found that the frame pairing step is sufficient to ensure loop closure, given positioning sensor data.

For the optimization, we use an approximate plane-to-plane distance derived from maximum likelihood estimation [21]. We use a rigid transformation model, *i.e.*, rotation and translation, for each frame to be aligned. Given a set of point correspondences $\mathcal{S}$ found in pairs of frames, the objective function E to be minimized over translation $\mathbf{t}$ and rotation R is defined as

$$E = \sum_{(\mathbf{p}_i^m, \mathbf{p}_j^n) \in \mathcal{S}} \mathbf{d}\left(\mathbf{p}_i^m, \mathbf{p}_j^n\right)^{\mathsf{T}} \Sigma_{mn}^{-1} \mathbf{d}\left(\mathbf{p}_i^m, \mathbf{p}_j^n\right), \quad (1)$$

where $\mathbf{p}_i^m$ is the position of i-th point in m-th frame. The distance function $\mathbf{d}$ and weighting factor Σ^{-1} are defined as

$$\mathbf{d}\left(\mathbf{p}_i^m, \mathbf{p}_j^n\right) = \left(R_m \mathbf{p}_i^m + \mathbf{t}_m\right) - \left(R_n \mathbf{p}_j^n + \mathbf{t}_n\right), \quad (2)$$

$$\Sigma_{mn} = R_m \tilde{C}_{m,i} R_m^{\mathsf{T}} + R_n \tilde{C}_{n,j} R_n^{\mathsf{T}} \quad , \quad (3)$$

$$\tilde{C}_{m,i} = U_{m,i} \ \text{diag}\ (1,\ 1,\ \epsilon)\ \ U_{m,i}^{\mathsf{T}} \quad , \quad (4)$$

where $U_{m,i}$ contains eigenvectors of the covariance matrix of $\mathbf{p}_i^m$, and ϵ is a small constant representing variance along the normal direction, set to 10^{-3}.

To avoid excessive rotation and resulting erroneous point correspondences over iterations, we take a two-stage optimization strategy. Specifically, we first restrict the transformation to translation only, and once it has converged, we then relax the transformation to be rotation and translation. These steps are as follows.

Estimation of translation $\mathbf{t}$ In the first stage with translation only, the rotation parameter in Eqs. (2) and (3) are set to identity ($R = I$). This case makes the objective function E quadratic with respect to the translation $\mathbf{t}$, and the optimal solution can be efficiently obtained via the normal equation derived from $\partial E / \partial \mathbf{t}_m = \mathbf{0}$.

Estimation of translation $\mathbf{t}$ and rotation R The second stage of translation $\mathbf{t}$ and rotation R estimation assumes small rotation θ_z only around the vertical z axis (yaw). In our experimental setup, rotations about other axes are typically negligible, due to the way IMU sensors measure rotation. By assuming a small rotation, the rotation matrix can be approximated to a linear form as

$$R = \begin{pmatrix} 1 & \theta_z & 0 \\ -\theta_z & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (5)$$

Note that the orthogonality of the rotation matrix is enforced using SVD after the linear estimation. Due to the non-linearity of the objective function E, we take an alternating optimization approach by treating Σ as an auxiliary variable. Namely, $\{\mathbf{t}, R\}$ and Σ are updated one after another, by first solving Eq. (1) using the previous estimates of Σ, then updating Σ by

$$\Sigma_{mn} \leftarrow R_m \tilde{C}_{m,i} R_m^{\mathsf{T}} + R_n \tilde{C}_{n,j} R_n^{\mathsf{T}} \quad (6)$$

using the previous estimates of R. The alternating optimization is repeated until convergence.

In the above optimization stages, the convergence criterion is defined using the norm of the parameter variations; when it becomes less than 10^{-8}, the iteration is terminated.

3.3. Inter-Loop Registration

We rely on the dual inclusion property of edges of a capture path graph to serve as a basis for inter-loop registration. Specifically, given a rigid transformation consisting of rotation matrix A and translation $\mathbf{b}$ for each loop, sensor positions $\mathbf{s}$ shared by i-th and j-th loops satisfy

$$A_i \mathbf{s}_i + \mathbf{b}_i = A_j \mathbf{s}_j + \mathbf{b}_j, \quad (7)$$

where A is defined in the same manner as Eq. (5), and $\mathbf{s}_i$ represents a device position in i-th loop after intra-loop registration. Our GPS sensor produces a confidence value for each pose measurement $c = [1; 6]$, where 1 is the most confident. In normal operation, about 8% of the poses have the highest confidence, which results in centimeter accuracy. We make use of the high confidence poses to anchor the transformations by defining an additional constraint

$$A_i \mathbf{s}_i^H + \mathbf{b}_i = \hat{\mathbf{s}}^H, \quad (8)$$

which ensures that high confidence poses $\mathbf{s}^H$ stay in their original position $\hat{\mathbf{s}}^H$.

Bringing together all loops with Eqs. (7) and (8), we formulate a sparse linear system of equations with respect to A and $\mathbf{b}$. The solution is efficiently obtained by solving the system in a least-squares sense. Once A and $\mathbf{b}$ are estimated for all loops, we apply these rigid transformations for all points in each loop to produce a single, final point cloud.

3.4. Computational Cost

For the loop detection part, our algorithm can achieve a time complexity of $O(|V|+|E|)$, where $|V|$ is the number of

nodes and $|E|$ the number of edges in the graph. In the city map settings, both of them can be considered proportional to the number of sensor positions in the capture paths.

The computational cost of intra-loop registration mainly consists of four parts. Given m frames, each of which containing n points, the KD-tree construction for nearest neighbor search takes $O(n \log^2 n)$ for each frame, resulting in $O(mn \log^2 n)$ in total. The covariance matrix calculation for each point costs $O(mn)$. The time to construct the linear system will be proportional to the number of point correspondences, which is approximately $O(mn)$ if the frame distance threshold is fixed. The constructed linear system is solved in $3m \times 3m$ for the translation-only case, and $4m \times 4m$ for translation and z-rotation, so the time complexity is $O(m^3)$. The total time complexity for intra-loop registration is $O(mn \log^2 n + m^3)$.

As inter-loop registration algorithm solves a sparse linear system, the complexity is $O(k^2)$, where k is the number of loops. This complexity bound depends on the sparsity assumption that the amount of adjacent loops for any single loop is bounded.

The time complexity for the entire registration process is $O(k^2 + km^3 + kmn \log^2 n)$. If a typical direct registration method such as simultaneous ICP is performed, the time complexity is $O(k^3 m^3 + kmn \log^2 n)$. We significantly reduce the computation cost by applying the three-step approach described above, especially on a city-scale registration problem with large k.

Figure 3. Result of loop detection. Top: all the capture paths, middle: detected loops from our point clouds, and bottom: close views. Detected loops are colorized based on loop IDs.

4. Experiments

Our dataset was acquired by vehicles equipped with LiDAR and GPS/IMU sensors over the city of San Francisco. It represents an area of 125 km^2 and 1,900 km of roads, with a total of about 0.4 trillion points. We consider points in a 360° sweep as a single point cloud frame, and collect every four-meter frames for each detected loop.

4.1. Results of San Francisco Dataset

We first show the result of loop detection for the entire San Francisco dataset. Figure 3 illustrates part of all the capture paths (top), and the detected loops (middle and bottom). The process took 5.3 minutes without data loading on our distributed computing system. The entire area was partitioned into around 4,600 loops. Note that our method is straightforwardly generalized to other 3D data capture contexts, provided that the sensor path graph can be projected into a 2D coordinate system in which a meaningful ordering of edges around each node can be defined. Generalization to more arbitrary 3D graphs is a topic of future exploration.

After loops were detected, we ran our registration method on the entire San Francisco dataset. Intra-loop registration was fully parallelized and took about 4 hours on our distributed computing system. Finally, Inter-loop registration took around 24 seconds. Figure 4 shows the resulting point cloud. Large misalignments in the raw point clouds were significantly refined.

4.2. Comparisons on Small Datasets

Based on the loop detection results, we created two small datasets and evaluated our approach for accuracy and computational cost. As a baseline technique, we applied S-GICP to entire point clouds of the datasets without loop detection, which we refer to as *direct simultaneous registration* (DSR). One dataset ("Data 1") consists of four loops, and each loop contains around 260-290 frame point clouds. The other dataset ("Data 2") consists of three loops, and the number of frame point clouds varies from around 100 to 300. Table 1 shows details of these two datasets. The point clouds in "Entire" are used for DSR. For this evaluation, we used a PC with 16-core 2.9 GHz with 256 GB RAM.

Table 2 summarizes the computational timings of each step in our approach and DSR. Because our method can process all the loops in parallel, it took at most 287 minutes for Data 1, while DSR for the entire point clouds took 565 minutes. Similarly, for Data 2, our method took at most 144 minutes, while DSR took 250 minutes. Our approach,

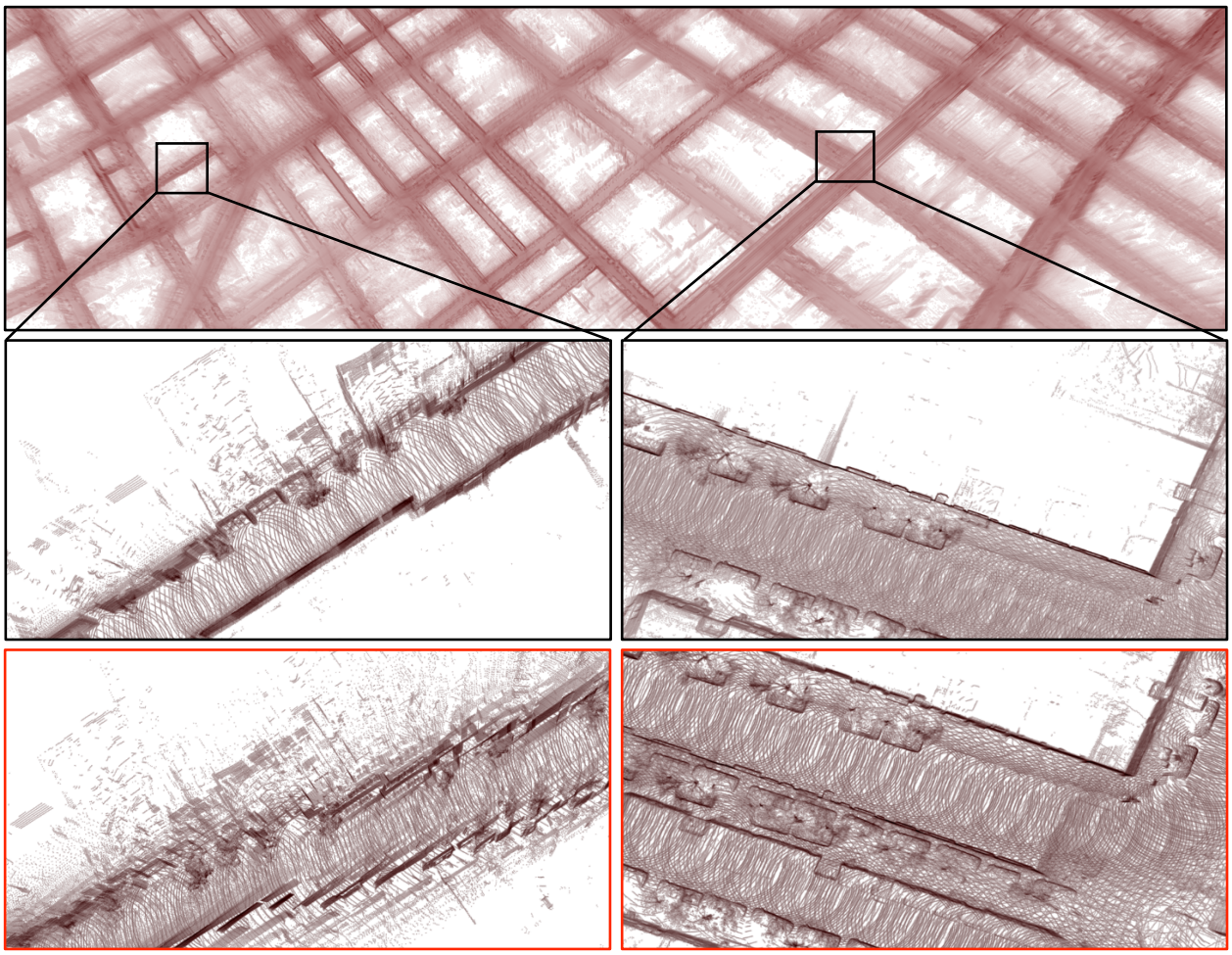


Figure 4. Result of San Francisco point cloud registered by our approach. Top: a far view, middle: closer views, and bottom: corresponding raw point clouds.

Table 1. Datasets for efficiency comparisons. The numbers without and with parentheses represent total point counts and frame point cloud counts, respectively.

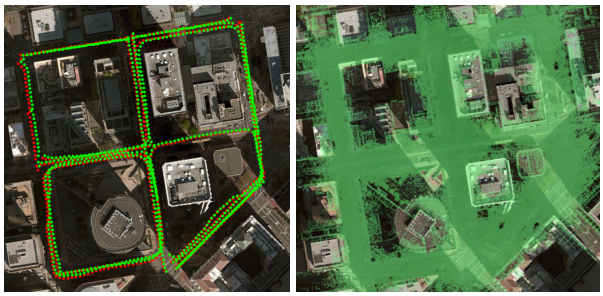
	L1	L2	L3	L4	Entire
Data 1	16M (297)	14M (260)	16M (299)	14M (259)	45M (821)
Data 2	9.7M (170)	5.1M (88)	17M (296)	– –	27M (459)

Table 2. Comparisons of computational timings between our approach and DSR. The numbers in parentheses represent the number of outer loop iterations in DSR, and "m" and "s" represent minute and second, respectively.

	Ours		DSR
	Intra.	Inter.	
Data 1	L1: 105m (5), L2: 163m (6), L3: 287m (17), L4: 185m (8)	0.425s	565m (7)
Data 2	L1: 93m (9), L2: 38m (12), L3: 144m (5)	0.362s	250m (9)

even with the loop detection, was around twice faster than DSR. According to the computational cost analysis, we can expect more efficiency improvements if more loops are considered.

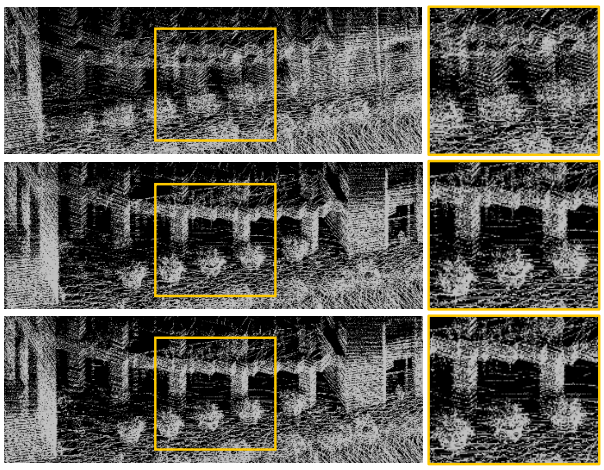
Figure 5 shows raw and refined sensor trajectories and registered point clouds overlaid onto aerial images. In particular, Data 2 contains significant GPS errors highlighted with the orange-colored box. Our approach corrected the errors, and the resulting trajectory and point clouds are reasonably matched to the aerial image. Figure 6(a) shows comparisons of raw point clouds, DSR result, and our result for Data 1, and Figure 6(b) shows similar comparisons for some facade in Data 2. In Data 2, significant GPS errors along the vertical direction were observed as highlighted with red arrows in the raw point clouds. Both DSR and our results were significantly improved from the raw data, and have comparable quality. These comparisons indicate that our approach achieves registration results visually comparable with DSR, while our approach is much more efficient.



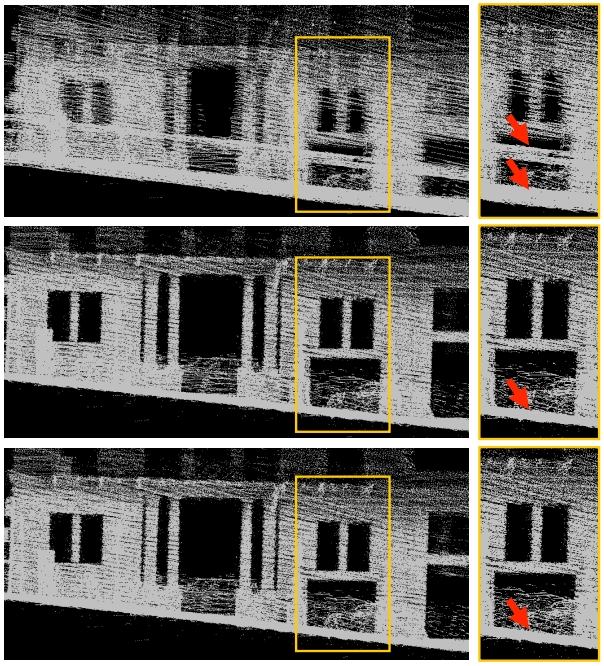
(a) Data 1

(b) Data 2

Figure 5. Results of our approach for the two datasets. The left in (a) and the top in (b) show raw GPS trajectories (red dots) and refined trajectories (green dots). The right in (a) and the bottom in (b) show registered point clouds overlaid onto aerial images.



(a) Data 1



(b) Data 2

Figure 6. Close views of point clouds. Top: raw point clouds, middle: point clouds by DSR, and bottom: point clouds by our approach. Figures on the right show zoomed-in views of points in the orange boxes on the left.

5. Discussion

In this paper, we presented an efficient approach for city-scale point cloud registration. We take advantage of the structure of our dataset to decompose it into several smaller and more tractable subsets, which can be processed independently. Our approach consists of three main steps: We first partition the input point cloud into individual city blocks. We then register each block independently using simultaneous GICP. This step is highly parallelizable, yet it still produces very accurate alignment. Finally, we bring all blocks together in a single closed-form solution, which also takes high confidence GPS measurements into account to globally anchor our final point cloud.

A limitation of our method comes from the assumption of small rotational errors for efficient optimization in both intra- and inter-loop registration. In practice however, we found this assumption to be reasonable: The maximum angle of estimated rotations in the San Francisco dataset was 1.1°. A violation of this assumption, caused for instance by large positioning errors, would decrease registration quality.

An interesting future direction for our work is indoor point cloud registration. Recent developments, such as commodity depth cameras or indoor positioning systems using Wi-Fi or Bluetooth, have made indoor 3D capture possible. Our method would be well suited to register such data and possibly align it with the type of outdoor data we deal with in this paper.

References

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building rome in a day. *Communications of ACM*, 54(10):105–112, 2011. 1

[2] P. Bariya, J. Novatnack, G. Schwartz, and K. Nishino. 3D geometric scale variability in range images: Features and descriptors. *International Journal of Computer Vision*, 99(2):232–255, 2012. 2

[3] R. Benjemaa and F. Schmitt. Fast global registration of 3D sampled surfaces using a multi-z-buffer technique. In *Proc. International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, pages 113–120, 1997. 2

[4] R. Bergevin, M. Soucy, H. Gagnon, and D. Laurendeau. Toward a general multi-view registration technique. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):540–547, 1996. 2

[5] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 1, 2

[6] S. Bouaziz, A. Tagliasacchi, and M. Pauly. Sparse iterative closest point. *Computer Graphics Forum*, 32(5), 2013. 2

[7] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. 2, 3

[8] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In *Proc. Eurographics Symposium on Geometry Processing*, 2005. 2

[9] K. Ikeuchi, T. Oishi, J. Takamatsu, R. Sagawa, A. Nakazawa, R. Kurazume, K. Nishino, M. Kamakura, and Y. Okamoto. The great buddha project: Digitally archiving, restoring, and analyzing cultural heritage objects. *International Journal of Computer Vision*, 75(1), 2007. 1

[10] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig. Cycle bases in graphs: Characterization, algorithms, complexity, applications, 2009. 3

[11] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH*, pages 131–144, 2000. 1

[12] V. Madhav and A. Pooja. On averaging multiview relations for 3D scan registration. *IEEE Transactions on Image Processing*, 23(3):1289–1302, 2014. 2

[13] T. Masuda. Generation of geometric model by registration and integration of multiple range images. In *Proc. International Conference on 3D Imaging and Modeling*, 2001. 2

[14] P. J. Neugebauer. Geometrical cloning of 3D objects via simultaneous registration of multiple range images. In *Proc. Shape Modeling International*, pages 130–139, 1997. 2

[15] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. 3

[16] K. Nishino and K. Ikeuchi. Robust simultaneous registration of multiple range images. In *Proc. Asian Conference on Computer Vision*, 2002. 2

[17] P. Pfaff, R. Triebel, C. S. P. Lamon, W. Burgard, and R. Siegwart. Towards mapping of cities. In *Proc. International Conference on Robotics and Automation*, pages 113–120, 2007. 2

[18] K. Pulli. Multiview registration for large data sets. In *Proc. International Conference on 3D Imaging and Modeling*, 1999. 2

[19] T. Pylvänäinen, J. Berclaz, T. Korah, V. Hedau, M. Aanjaneya, and R. Grzeszczuk. 3D city modeling from street-level data for augmented reality applications. In *Proc. International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 238–245, 2012. 3

[20] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. International Conference on 3D Imaging and Modeling*, 2001. 2

[21] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proc. Robotics: Science and Systems*, 2009. 1, 2, 4, 5

[22] G. C. Sharp, S. W. Lee, and D. K. Wehe. Multiview registration of 3D scenes by minimizing error between coordinate frames. *IEEE Transactions on Image Processing*, 26(8):1037–1050, 2004. 2

[23] S.-W. Shih, Y.-T. Chuang, and T.-Y. Yu. An efficient and accurate method for the relaxation of multiview registration error. *IEEE Transactions on Image Processing*, 17(6):968–981, 2008. 2, 3

[24] G. K. Tam, Z.-Q. Cheng, Y.-K. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X.-F. Sun, and P. L. Rosin. Registration of 3D point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1199–1217, 2013. 2

[25] G. Zou, J. Hua, Z. Lai, X. Gu, and M. Dong. Intrinsic geometric scale space by shape diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 15(6), 2009. 2