Incremental Segment-Based Localization in 3D Point Clouds

Renaud Dubé*, Mattia G. Gollub*, Hannes Sommer, Igor Gilitschenski,
Roland Siegwart, Cesar Cadena, and Juan Nieto[1]

***Abstract*— Localization in 3D point clouds is a highly challenging task due to the complexity associated with extracting information from 3D data. This paper proposes an incremental approach addressing this problem efficiently. The presented method first accumulates the measurements in a dynamic voxel grid and selectively updates the point normals affected by the insertion. An incremental segmentation algorithm, based on region growing, tracks the evolution of single segments which enables an efficient recognition strategy using partitioning and caching of geometric consistencies. We show that the incremental method can perform global localization at 10Hz in a urban driving environment, a speedup of x7.1 over the compared batch solution. The efficiency of the method makes it suitable for applications where real–time localization is required and enables its usage on cheaper, low–energy systems. Our implementation will be available open source after publication along with instructions for running the system.**

I. INTRODUCTION

Perception capabilities are a key requirement for robots to perform high–level tasks like navigation and interaction. For this reason, mobile robots are often equipped with 3D time-of-flight sensors which can produce precise reconstructions of the environment. Processing these detailed data can however result in high computational costs. Amongst important capabilities which can strongly benefit from efficient solutions, we focus on the challenging task of localization in 3D point clouds. Making global associations in 3D data permits us to construct an unified representation without making an assumption of low drift, or known relative starting position, in case of multi-robot applications.

This work presents an incremental solution to localization in 3D point clouds based on the principle of segment extraction and matching from accumulated data [1]. As demonstrated in our previous work, accumulating data can help recognize places which are observed from different viewpoints [2]. However, fully processing such representation at each time-step leads to expensive redundant computations. Therefore, we propose to reuse the processed information by retaining data structures that are likely to save computations in later localization attempts.

The first module of the presented solution accumulates and filters a continuous stream of 3D point cloud data through a Dynamic Voxel Grid (DVG), providing information about newly occupied voxels. This key information is leveraged for estimating normals by re-computing only those affected by newly occupied voxels and by caching information to incrementally compute covariance matrices. Incremental region growing segmentation is then performed by using only the newly occupied voxels as seeds and by merging with previously clustered points. As illustrated in Fig. 1, this strategy enables to robustly track segments between successive observations which offers multiple benefits to the overall framework. Amongst others, it enables the final stage of our solution: an efficient recognition strategy based on partitioning and caching of geometric consistencies.

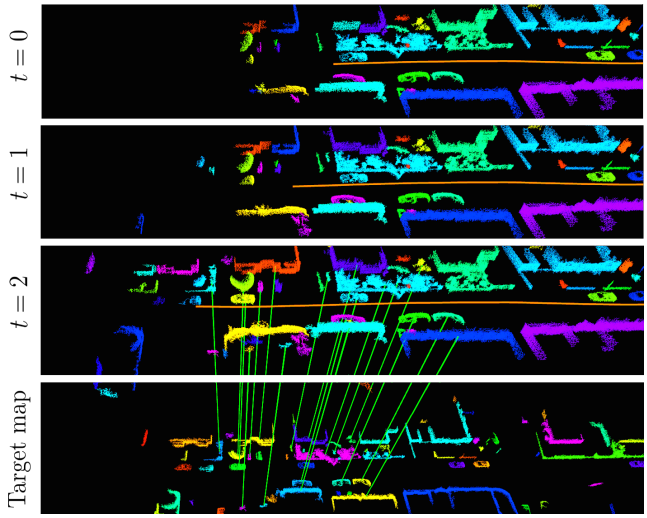

Fig. 1: Example of localization made by our incremental approach in a urban driving scenario. The robot is driving from right to left and its trajectory is shown in orange. Segments are extracted, tracked and merged over successive observations. At $t = 2$ a localization is made against the target map with segment correspondences denoted by green lines.

To the best of our knowledge, this is the first work to propose combining incremental solutions to normal estimation, segmentation, and recognition for finding global associations in 3D point clouds. The full solution is evaluated in real-world experiments demonstrating localization rates of up to 10Hz. We also show that the efficiency of the presented method makes it suitable for real–time applications and enables its usage on cheaper, low–energy systems.

To summarize, this paper presents the following contributions:

- An incremental method for localization in 3D point clouds based on segment matching.
- A set of incremental algorithms for the normal estimation, segmentation, and recognition steps.

*The authors contributed equally to this work. [1]Authors are with the Autonomous Systems Lab, ETH, Zurich. Corresponding authors: renaudube@gmail.com and gollubm@ethz.ch.

This work was supported by the European Union's Seventh Framework Programme for research, technological development and demonstration under the TRADR project No. FP7-ICT-609763.

- An exhaustive comparison of the incremental approach with a batch solution through disaster response and urban driving experiments.

The remainder of the paper is structured as follows: Section II provides an overview of the related work, and Section III describes the proposed incremental approach. The incremental approach is compared to a batch solution in Section IV, and Section V finally concludes the work.

II. RELATED WORK

An overview of the related work in the field of localization in 3D point clouds was presented in our previous work [1]. In this section we review previously proposed methods related to the two core modules of our approach: efficient point cloud segmentation, and geometric verification.

a) Incremental point cloud segmentation: Closely related to our work, Whelan et al. [3] proposes an incremental region growing method for segmenting dense point cloud maps. Segmentation is done only once for each input cloud with a merging step afterwards. Only planar segments were considered whereas our generic region growing algorithm allows for different tuples of growing policies. Tateno et al. [4] merge RGB-D data into a *global segmentation map* that is maintained by matching and propagating segments extracted from the current depth map. Similarly, Finman et al. [5] propose to segment the depth maps using an incremental variation of the graph-based Felzenszwalb algorithm. A voting algorithm is proposed for recomputing parts of the segmented map given new data. None of the above works proposed a solution for retrieving models based on the generated segments. Contrastingly, we show through multiple experiments that our incremental region growing algorithm can effectively be leveraged for localization.

b) Efficient geometric verification: Strategies for reducing the number of correspondence pairs have been proposed for stereo images. Ayache and Faverjon [6] describe a partitioning scheme for efficiently finding neighbor segments in stereo images, while [7] performs RANSAC only on *spatially consistent* correspondences, i.e. correspondences that have a minimum fraction of matching neighbor features in both images. Both methods rely on assumptions about the disparity between images, thus their accuracy is influenced by the presence of high disparity and strong variation in viewing angles. In this work we present a method for performing localization efficiently through partitioning and caching, reducing the asymptotic complexity of the geometric consistency grouping method [8].

III. METHOD

This section introduces our incremental solution to localization in 3D point clouds and the contributing modules. An overview of the approach is depicted in Fig. 2.

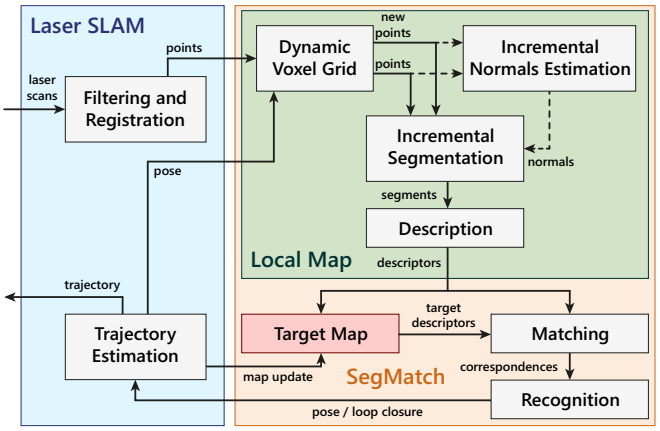

Fig. 2: Architecture of the incremental segment matching pipeline and its interface with a laser SLAM framework[1].

A. Dynamic Voxel Grid

The continuous input stream of 3D points is filtered and accumulated in a local cloud using a voxel grid approach. Instead of performing batch voxel filtering of the entire local cloud with each new measurement, we add and update only voxels that are affected by the new points. We implemented a DVG, an efficient data structure supporting dynamic insertion and removal of points. Occupied voxels are stored in a vector in increasing voxel index order. Each voxel maintain its index, centroid, and the number of points it contains. In order to reduce noise, a voxel is considered *active* if it contains at least a desired amount of points. For the successive stages of the pipeline, only active voxels are used.

1) Voxel Indexing: Virtually, the voxel grid is a regular grid of size $l \times w \times h$ voxels, where each voxel has a unique index in the interval $[0, l \cdot w \cdot h - 1]$. The grid has a fixed resolution r and a rigid transformation from the grid frame to the map frame T_{mg} which is initialized such that the robot starts at the center of the grid. Indexes are stored in b–bits unsigned integers, for computational efficiency we require the sizes of the grid to be powers of 2: $l = 2^{l_{bits}}$, $w = 2^{w_{bits}}$ and $h = 2^{h_{bits}}$, where $l_{bits} + w_{bits} + h_{bits} \leq b$. The voxel index $I(q)$ of a point q is computed as:

$$I(q) = \lfloor t.x \rfloor + \lfloor t.y \rfloor \ll l_{bits} + \lfloor t.z \rfloor \ll (l_{bits} + w_{bits}) \quad (1)$$

where $\ll$ is the bitwise left shift operator and t corresponds to the grid coordinates of the point q according to: $t = T_{mg}^{-1} \cdot q \cdot r^{-1}$.

2) Insertion and Removal: When new points are inserted, the DVG computes their voxel indices and sorts them in increasing voxel index order. Considering that sorting has an asymptotic complexity $O(n \log(n))$, this is an important optimization over sorting all the points as performed in batch voxelization. Once new points are sorted, they are added to

[1]In this work we use an incremental pose-graph SLAM system which performs registration between successive LiDAR scans using Iterative Closest Point (ICP) [2]. The implementation is available at `github.com/ethz-asl/laser_slam`.

the existing voxels in linear time with a merge operation: When m points q_i are inserted in a voxel with centroid p downsampled from n points, its properties are updated as:

$$p \leftarrow \left(n \cdot p + \sum_{i=1}^{m} q_i \right) \cdot \frac{1}{n+m}, \qquad n \leftarrow n+m \tag{2}$$

In addition, the indices of the voxels that turned active after the insertion are collected in a set U, so that the normal estimator and the segmenter can operate on the new voxels only. When the robot moves, the DVG is updated by removing voxels that fall outside the radius of the local map.

3) Rigid Transformation: When a loop closure is detected, the SLAM application re–evaluates the trajectory of the robot and provides a new estimation of its pose. Consequently, the local cloud must be updated with a rigid transform T which is applied to the centroid of each voxel. Moreover, in order for new points to be assigned to the correct voxel, the transformation of the DVG is updated as $T_{mg} \leftarrow T T_{mg}$.

B. Incremental normal and curvature estimation

The normal of a point p_i in a 3D point cloud is commonly estimated from the covariance matrix M of a neighborhood $\mathcal{N}(p_i)$ [9]. After finding the neighborhood by fixed–radius Nearest Neighbors (NN) search and arranging the neighbor points in a n_i-tuple $\{\nu_j\}_{j=1}^{n_i} := \mathcal{N}(p_i)$, M_i is computed as a sample covariance, using $\bar{\cdot}$ to denote the average operation, i.e. $\overline{\nu_j} := \frac{1}{|\nu|} \sum_{j=1}^{|\nu|} \nu_j$, omitting the index if unnecessary:

$$M_i := \overline{(\nu_j - \overline{\nu})(\nu_j - \overline{\nu})^\intercal}, \tag{3}$$

The estimate of the normal is equal to the normalized eigenvector of M_i corresponding to the smallest eigenvalue, while the curvature is computed as $\sigma = \lambda_0 (\lambda_0 + \lambda_1 + \lambda_2)^{-1}$ where $\lambda_0 < \lambda_1 < \lambda_2$ are the eigenvalues of M_i.

In this work, we apply two major optimizations to make the process incremental: The covariance matrix M_i is computed incrementally and only normals affected by new scanned points are updated. By expanding the factors in eq. (3), we obtain an incremental formulation:

$$M_i = \overline{\nu_j \nu_j^\intercal} - \overline{\nu}\,\overline{\nu^\intercal} = \frac{1}{n_i} \cdot A_i - \frac{1}{n_i^2} \cdot b_i b_i^\intercal \tag{4}$$

where A_i and b_i are respectively the accumulators for $\overline{\nu_j \nu_j^\intercal}$ and $\overline{\nu}\,\overline{\nu^\intercal}$. The advantage of this formulation over eq. (3) is that it can be computed incrementally, without the need to keep track of the neighborhood $\mathcal{N}(p_i)$ of each point. For reference, a similar formulation is introduced by Poppinga et al. [10] for performing efficient batch plane detection in 3D point cloud data.

1) Incremental Updates: The accumulators of each point are computed incrementally following a *contributions scattering and gathering* procedure. For each new point index $i \in U$ accumulators A_i, b_i and n_i are initialized with 0 and for each $p_j \in \mathcal{N}(p_i)$, including already the new points, contributions are scattered as:

$$A_j \leftarrow A_j + p_i p_i^\intercal, \quad b_j \leftarrow b_j + p_i, \quad n_j \leftarrow n_j + 1 \tag{5}$$

Similarly, contributions are gathered from old points only, i.e. if $j \notin U$:

$$A_i \leftarrow A_i + p_j p_j^\intercal, \quad b_i \leftarrow b_i + p_j, \quad n_i \leftarrow n_i + 1 \tag{6}$$

Finally, covariance matrices, normals, and curvatures are recomputed for points whose accumulators have been updated.

2) Rigid Transformation: In the event of loop closures, the trajectory of the robot is re–estimated and a rigid transformation is applied to the filtered point cloud C. When this happens, we transform the accumulators in order to avoid inconsistencies caused by the accumulation of points belonging to different reference frames.

The transformation, expressed as translation t after rotation R, is applied to all points p_i belonging to the local cloud as $p_i \leftarrow R p_i + t$. Let $\tilde{\nu} := (R\nu_j + t)_{j=1}^{n_i}$ be the transformed neighborhood tuple of the point p_i. Then the updated sample covariance (4) for $\tilde{\nu}$ can be computed using:

$$\overline{\tilde{\nu}_j \tilde{\nu}_j^\intercal} = R\,\overline{\nu_j \nu_j^\intercal} R^\intercal + R\,\overline{\nu} t^\intercal + t\,\overline{\nu}^\intercal R^\intercal + t t^\intercal \tag{7}$$

$$\overline{\tilde{\nu}} = \overline{R\nu_j + t} = R\,\overline{\nu} + t \tag{8}$$

Thus, the accumulators are updated as:

$$A_i \leftarrow R A_i R^\intercal + R b_i t^\intercal + t b_i^\intercal R^\intercal + n_i t t^\intercal, b_i \leftarrow R b_i + n_i t \tag{9}$$

The normals are updated as: $N_i \leftarrow R N_i$, while the point curvatures are not affected by the transformation.

C. Incremental region growing segmentation

This section presents our generic algorithm for incremental segmentation of 3D point clouds based on *region growing policies*. Given the low fraction of new points added with each measurement, we achieve an efficient segmentation by using only new points as seeds for growing regions.

Algorithm 1 shows the pseudocode for growing a region in a 3D point cloud starting from a seed point with index s. Growing is performed using the seeds contained in the ordered seeds list provided by the PREPARESEEDS policy. The growing strategy is controlled by the CANGROWTO and CANBESEED policies, which respectively determine if growing from a seed to a neighbor is allowed and if a point can be used as seed. The result is a *cluster* Γ with a unique *cluster ID* γ. New unclustered points initially have no cluster ID assigned. The NN function on line 5 finds the neighbors of a point by fixed–radius search. In order to reduce the number of k–d tree constructions, the same tree is shared with the normal estimator. Future work could gain efficiency over this approach by exploiting the underlying structure of the DVG through organized region growing segmentation.

Once all points have been clustered, clusters that reached a minimum number of points are promoted to *segments*

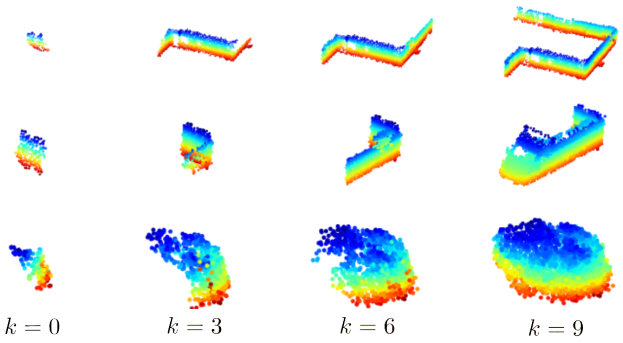

Fig. 3: An illustration of three segments being incrementally grown over successive observations.

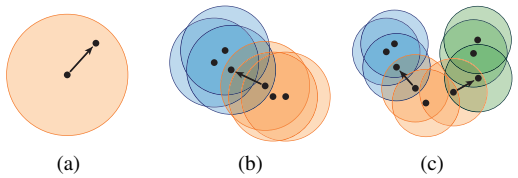

Fig. 4: Region growing and merging examples using Euclidean distance policies in 2D. (a) An unclustered point is reached and added to the current cluster. (b) An existing cluster (blue) is reached and linked to the growing region (orange). (c) A current cluster (orange) reaches and is linked with two other clusters (blue and green).

and obtain a unique *segment ID*. A mapping from cluster to segment IDs is maintained in order to enable *segment tracking* (section III-C.3). An example of segments being grown over multiple observations is depicted in Fig. 3.

Algorithm 1 Incremental region growing given a starting seed s, a point cloud P and a new cluster ID γ.

```
1:  function GrowFromSeed(s, P, γ)
2:      Γ ← {s} , S ← {s} // Initialize cluster and set of seeds.
3:      while Seeds ≠ ∅ do
4:          s ← PopFront(S)
5:          for each n : NN(s, P) do
6:              if CanGrowTo(s, n) then
7:                  if HasClusterId(n) then
8:                      LinkClusters(s, n, γ)
9:                  else
10:                     SetClusterId(n, γ)
11:                     Γ ← Γ ∪ {n}
12:                     if CanBeSeed(n) then
13:                         S ← S ∪ {n}
14:     return Γ
15: end function
```

1) Clusters merging: While batch algorithms only need to cover the cluster growing case, the incremental version must also handle the cluster merging case (see Fig. 4). This is illustrated on line 8 where a point that has been previously assigned to another cluster is reached. In this case, the two clusters are merged and obtain the same cluster ID. If both clusters already have a valid segment ID (Fig. 4c), the minimum (i.e. the oldest) segment ID is used for the resulting set. In SLAM applications, this causes the segments to be merged in the target map as well.

2) Growing policies: In this work, we present two triples of policies for our incremental region growing algorithm.

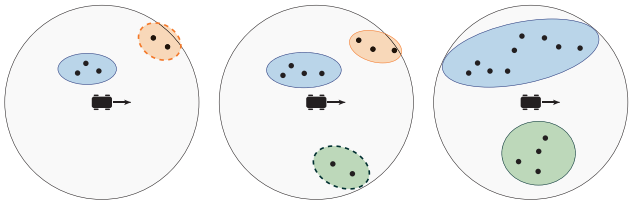

Fig. 5: Evolution and tracking of segments as the robot moves. The black circle represents the boundary of the local map. In this example, the minimum size a cluster must have in order to be considered a segment is 3. (a) The robot observes a segment (blue) and a cluster (orange). (b) As the robot moves, more points are inserted in the local map. The orange cluster turns into a segment and another cluster appears. The blue segment grows but maintains the same ID. (c) From a different perspective, more points are observed, triggering the merge of the blue and orange segments.

The *smoothness constraint policies* are derived from the work by Rabbani et al. [11]. During the preparation phase, PrepareSeeds collects the indices of the points that pass the CanBeSeed test and sorts them in increasing curvature order. This guarantees that regions are grown starting from the flattest points, reducing the number of segments created. CanGrowTo returns true if the normals of the seed and neighbor points are close to parallel. Since the orientation of the normals is unknown, this is approximated by checking that the magnitude of the dot product between the two normals falls below a given threshold. Another maximum threshold is applied on the point curvature in order for the CanBeSeed test to pass.

The *Euclidean distance policies* are straightforward as the incremental region growing algorithm already finds candidate neighbors based on their euclidean distance. Therefore, both CanGrowTo and CanBeSeed always return true and PrepareSeeds simply collects the indices of points that are not yet assigned to a cluster.

3) Segment Tracking: While cluster IDs are only temporary values used for identifying points belonging to the same clusters, segment IDs are lifetime–long identifiers of segments. The segmentation procedure presented in this section allows us to robustly track segments and their successive *views* in the local map which offers multiple benefits.

In the previous work [1], multiple views could not be associated to the same segment and would obtain different IDs, causing the insertion of *segment duplicates* in the target map. Although heuristically identifiable by small distances between segment centroids, precise detection of such segments was not possible. Contrastingly, our method can robustly track segments and update them in the target map. Segment tracking also enables correspondences caching, which is needed by our incremental recognition approach (Section III-D). Moreover, having access to the complete history of the observations of each segment enabled the development of a method for learning segment descriptors that are more robust to changes in point of view [12]. Future work will explore different methods of leveraging the segment view history for improving segment matching performance.

D. Graph-based incremental recognition

Segments extracted from the local cloud are described with generic feature vectors (an eigenvalue-based descriptor [13] is used in the experiments of Section IV). Candidate *correspondences* between segments in the local and target maps are then found through NN searches in the feature space. A pair c_i, c_j of correspondences is called *geometrically consistent* if the difference of the Euclidean distance between the segment centroids in the local map and in the target map is less than a threshold ϵ, i.e. if

$$|d_l(c_i, c_j) - d_t(c_i, c_j)| \leq \epsilon, \tag{10}$$

where $d_l(c_i, c_j)$ and $d_t(c_i, c_j)$ are the distances between centroids in the local map and in the target map respectively. In our approach we formulate recognition as a graph problem with the goal of identifying a Maximum Pairwise Consistent Set (MPCS), which is a set of maximum size among all correspondence sets that are pairwise geometrically consistent.

Geometrical consistency relationships are encoded in a *consistency graph* $G = (V, E)$ where $V = \{c_i\}$ is the set of correspondences c_i and $E = \{e_{ij}\}$ is the set of undirected edges e_{ij} connecting all consistent pairs of correspondences (c_i, c_j). Identifying a maximum geometrically consistent set is then equivalent to finding a maximum clique of G.

We take advantage of the segment tracking feature described in section III-C.3 which enables us to track correspondences as well. The number of consistency tests performed is reduced in an incremental fashion by reusing information computed in previous recognition steps. Since the insertion of new points in a segment changes its centroid, caching consistencies directly is not efficient. We rather propose to cache, for each correspondence c_i, a set of correspondences $\mathcal{S}(c_i) \subset V$ that are candidate to be consistent with c_i.

Based on eq. (10) we define the *consistency distance*, a measure for how far two correspondences c_i and c_j are from being consistent:

$$\Delta(c_i, c_j) = |d_l(c_i, c_j) - d_t(c_i, c_j)| \tag{11}$$

For each c_i, its *consistent candidates set* $\mathcal{S}(c_i)$ is then defined as the set of correspondences c_j whose consistency distance to c_i falls below a maximum threshold θ_Δ:

$$\mathcal{S}(c_i) = \{c_j \in V \mid j \leq i \wedge \Delta(c_i, c_j) \leq \theta_\Delta + \epsilon\} \tag{12}$$

where ϵ is the tolerance for consistency and the condition $j \leq i$ prevents duplicate entries of the same pair (caused by the symmetry of the consistency relation) from being stored.

1) Cache Maintenance: When a correspondence c_i is found for the first time, $\mathcal{S}(c_i)$ is computed and stored in the cache, together with the centroids of the local and target map segments. When a correspondence is not observed anymore, all references to it are removed from the cache. Furthermore, the consistent candidates set of a correspondence is invalidated if its two centroids move in total by more than $\frac{1}{2}\theta_\Delta$. The total movement is computed as the sum of the distances of each centroid to its cached position. This ensures that pairs of correspondences initially considered inconsistent are reconsidered if the combined movement of the segment centroids can cause them to be consistent. Correspondences, whose consistent candidate set has been invalidated, are reinserted in the cache as new correspondences. Fig. 6 shows an example of consistency cache maintenance.

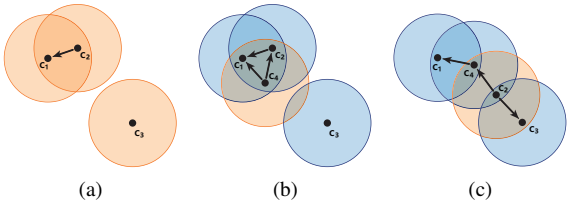

Fig. 6: An example of consistency cache maintenance. Correspondences have arbitrary positions, while the distance between correspondences represents their consistency distance according to eq. (11). Arrows point from correspondences to their consistent candidates and circles represent the threshold for caching θ_Δ. (a) The correspondences c_1, c_2 and c_3 are inserted in the given order, and c_2 is found to be a candidate for consistency with c_1. (b) When c_4 is inserted, caching c_1 and c_2 as candidates for consistency. (c) The centroid of a segment of c_4 changes by a distance smaller than $\frac{1}{2}\theta_\Delta$, thus its cached information is still valid. c_2 changes by a distance greater than $\frac{1}{2}\theta_\Delta$, thus its consistent candidates are recomputed.

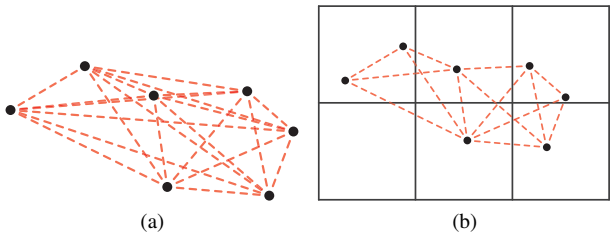

Fig. 7: A consistency graph example where nodes and edges represent correspondences, plotted at their target map centroid's position, and tested pairs respectively. (a) Current approaches need to test all possible correspondence pairs for consistency. (b) Our partitioning approach allows to drastically reduce the number of consistency tests.

2) Consistent candidates set identification: In order to efficiently determine the consistent candidates set of a correspondence, we adopt the following partitioning approach. In order for two correspondences to be consistent, the distance between their target segments must be less or equal to the diameter of the local map. Thus we propose to prefilter candidates using a partition of the correspondences corresponding to the position of their target segment centroids in a regular grid with a cell edge length equal to the diameter of the local map. All the elements of $\mathcal{S}(c_i)$ can then be found among the correspondences whose target map centroid belongs to the same or to a neighbor partition as shown in Fig. 7. This process is described in detail in our workshop report [14].

3) Consistency Graph Construction: Consistencies involving new correspondences are identified while searching for their consistent candidates sets. For all other correspondences c_i, consistency tests only need to be performed for all the cached candidates, i.e. for $\{c_i\} \times \mathcal{S}(c_i)$.

4) MPCS Identification: We consider a recognition to be successful in case the size of the detected MPCS set is greater than or equal to a threshold parameter T. Thus we

need to identify a maximum k–clique with $k \geq T$. Since the consistency graph is sparse, we can rely on a particular class of algorithms [15] to find a maximum clique in linear time.

IV. EXPERIMENTS

In this section we evaluate the proposed incremental approach to localization in 3D point clouds. The performance of the system is first evaluated and compared to a baseline solution through three experiments. Finally, we present a challenging, large–scale, multi–robot, SLAM application which is made possible with the proposed approach.

A. Baseline

The baseline used for the comparison is the original *SegMatch* implementation [1] which is composed of standard Point Cloud Library (PCL) components. Specifically, batch voxel filtering is performed by the `pcl::VoxelGrid` while batch normals estimation is achieved with `pcl::NormalEstimation`. Batch segmentation is provided by `pcl::EuclideanClusterExtraction` and `pcl::RegionGrowing`. Recognition is finally performed using `pcl::GeometricConsistencyGrouping`.

B. Performance

The proposed solution and the baseline have been benchmarked in three different conditions:

- **KITTI Localization**: The vehicle drives in a known urban scenario, continuously localizing against a map generated from sequence 00 of the KITTI dataset [16].
- **KITTI SLAM**: The vehicle explores an unknown urban scenario, continuously updating a dynamic target map and trying to detect loop-closures (KITTI sequence 05).
- **Powerplant Localization**: A rescue robot drives in a known indoor scenario continuously localizing. The dataset has been recorded at the *Knepper* powerplant in Dortmund in the context of the TRADR project [2].

For both solutions, segments are extracted using Euclidean distance policies for the KITTI scenarios and with smoothness constraints policies the powerplant scenario. Segments are always described using eigenvalue-based features [13].The other parameters are detailed in Table I and have been found experimentally to yield good performance for both the baseline and proposed solutions. Note that coarser grid resolutions and higher minimum number of points results in fewer active voxels and faster segmentation rates. However, the resulting segments could be less descriptive of the actual scene objects.

1) Hardware: All experiments have been performed on a system equipped with 32GB of RAM and an Intel i7–6700K processor.The segment matching pipelines run in single–threaded mode and, for all experiments presented in this paper, the RAM usage of the whole system, including 3D mapping and trajectory estimation, never surpassed 1.6GB.

Parameter	KITTI	Powerplant
Local map radius (m)	50.0	25.0
DVG resolution r (m)	0.1	0.1
Min number of points for activating voxel	1	2
NN search radius for growing (m)	0.2	0.5
Max angle between normals for growing (degrees)	–	4.0
Max point curvature for using as seed (m^{-1})	–	0.05
Max distance for consistency ϵ (m)	0.4	0.4
Min MPCS set size T	5	6

TABLE I: Parametrization of the segment matching modules.

2) System performance: The timings and speedups resulting from these experiments are presented in Table II. Details are given separately for each module, to which an incremental solution is proposed, and for both the baseline and the proposed approaches. The category *others* includes segment description, matching, and, in the SLAM experiment, target map construction. Note that the batch method additionally requires 10ms for associating segments when updating the target map whereas this is obtained directly in the incremental solution. In event of loop closures, an average of 22.8ms more is required by both methods for updating the target map and the k–d tree used for matching segments.

In all experiments, the proposed localization approach can process the measurements faster than the update rate of the sensor. This is particularly interesting in the KITTI experiments where our approach allows to process in real-time the large data throughput of the Velodyne HDL-64E. The overall speedups achieved by the incremental approach over the batch solution are 8.9x, 7.1x, and 12.4x respectively for each scenario. The processing rates achievable by the incremental pipeline range between 13Hz and 25Hz depending on the experiment. However, in practice, these values are now limited by the sensor frequencies. Further detailed statistics about the experiments are summarized in Table III.

We observe that successful localizations are generally based on a redundant amount of correspondences. Since the recognition step automatically rejects inconsistent candidate correspondences, the algorithm is still able to localize even in moderately dynamic environments. As an example, the localization shown in Fig. 1 is based on 18 candidates. Since we require at least $T = 5$ correspondences, localization would still be successful even if the parked cars would move. Future work could further improve the robustness to dynamic objects by (1) leveraging semantic information that can be extracted from machine learning-based segment descriptors [12] and (2) simultaneously using multiple growing policies in order to generate more candidate segments.

3) Dead reckoning distances: The higher localization rates of the proposed method results in lower dead reckoning distances. This is illustrated in Fig. 8 which shows the probability of traveling a specific distance without successful localization in the map generated from KITTI sequence 00. See [1] for a definition of this metric. With the proposed incremental approach, localization happen within 1.5m more than 90% of the times, while the original approach can

TABLE II: Runtimes of the modules of the batch and incremental approaches (ms).

Module	KITTI Localization			KTTI SLAM			Powerplant Localization		
	Batch	Incremental	Speedup	Batch	Incremental	Speedup	Batch	Incremental	Speedup
Voxel filtering	56.9 ± 19.2	4.2 ± 2.8	x13.5	69.6 ± 35.1	4.11 ± 1.88	x16.9	32.7 ± 18.7	1.7 ± 0.8	x18.8
Normal estimation	-	-	-	-	-	-	166.4 ± 64.9	10.2 ± 1.6	x16.4
Segmentation	389.7 ± 147.8	40.7 ± 13.4	x9.6	395.1 ± 123.0	44.5 ± 15.2	x8.9	275.5 ± 112.5	22.4 ± 8.6	x12.3
Recognition	85.4 ± 39.7	6.0 ± 3.2	x14.2	41.3 ± 54.5	3.5 ± 4.15	x11.8	0.7 ± 0.5	0.1 ± 0.05	x7.4
Others	10.0 ± 3.6	10.2 ± 3.5	-	34.4 ± 19.7	23.9 ± 12.7	-	3.7 ± 1.7	4.4 ± 1.9	-
Total	542.1 ± 210.3	61.2 ± 23.0	x8.9	540.4 ± 232.3	76.0 ± 33.9	x7.1	479.1 ± 198.2	38.7 ± 13.0	x12.4

Quantity (per–step)	KITTI Localization	KITTI SLAM	Powerplant Localization
Sensor rate	10Hz	10Hz	0.33Hz
Created voxels	2.1k±0.6k	2.1k±0.7k	2.2k±0.3k
Local cloud size	156.5k±50k	159.5k±41k	76.5k±34k
Modified normals	-	-	10.6k±3.3k
Local map clusters	9.1k±3.9k	8.6k±2.1k	7.4k±2.4k
Local map segments	52.8 ± 13.7	60.1 ± 13.8	42.7 ± 20.9
Target map segments	1204	847 ± 423	83
Partitions	25	8.2 ± 3.9	1
Correspondences	3.1k±0.9k	1.6k±1.0	103 ± 55
Cached correspondences	2.8k±0.8k	1.5k±0.9k	95 ± 53
Cache invalidations	2.3 ± 8.4	1.8 ± 6.3	0.1 ± 0.3

TABLE III: Statistics (mean and standard deviation) characterizing the different experiments. Values refer to observations made in one localization step. It is interesting to note some strong differences between experiments (e.g. number cache invalidations) caused by the changes in scenario and configuration.

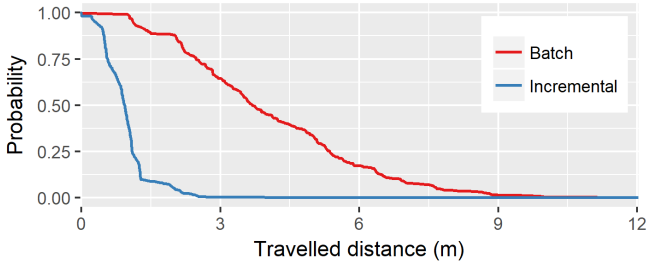

Fig. 8: Probability of travelling a specific distance without a successful localization (data recorded during 5 runs of the KITTI localization example).

localize within the same distance less than 10% of the times, occasionally traveling more than 10m without localization.

4) Dynamic Voxel Grid: As described in Section III-A, voxel filtering requires a sorting step to group points belonging to the same voxel. This $O(n \log n)$ step has a significant impact when the local cloud contains a lot of points. With the incremental approach, this operation is reduced to sorting only the new points and then merging them with the stored sorted points in linear time. As shown in Table III, the new voxels represent only a small fraction of the entire local cloud, justifying the speedups observed in Table II. The timings stated for the voxel filtering include one pose update (removal of voxels outside the radius of the local cloud) and one insertion of the queued scans.

5) Incremental normals estimation: Normal estimation has been evaluated in the powerplant scenario only, as the

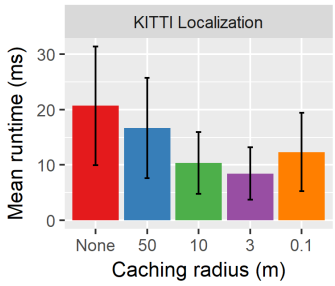

Figure 9: Mean runtime of the recognition stage with different caching radii. *None* indicates the pure partitioned approach without caching. In this dataset the ideal compromise between caching and invalidation is about 3 m.

Euclidean distance policies used in the KITTI examples do not require point normals. In this case, a speedup of 16.4x is observed which is explained by the smaller number of NN searches required and by the caching of the covariance matrices. Furthermore, our implementation allows us to reuse the k–d tree built for segmentation which was not performed in the batch solution. In an equitable comparison where both estimators need to build a k–d tree of the local cloud, the incremental approach is in average 7.1 times faster.

6) Incremental region growing segmentation: Similarly, the most important improvement factor for the segmentation module is the reduction of the number of NN searches performed at every step. This is achieved by reusing stored information about the clusters in the cloud and only starting region growing from new unsegmented points. As stated in Table II, incremental segmentation achieves speedups over the batch methods of 9.6x, 8.9x and 12.3x.

7) Graph–based incremental recognition: Thanks to the partitioning scheme and the incremental caching, our recognition method reached speedups of 14.2x, 11.8x and 7.4x respectively. Moreover, the runtime of our method scales linearly with the number of correspondences, significantly improving over the cubic scaling of the batch algorithm (see Gollub et al. [14] for an asymptotic complexity analysis). Interestingly, cached correspondences represent $> 90\%$ of the total correspondences (Table. III), but are tested for consistency faster than new correspondences. In the KITTI localization experiment, the incremental recognizer performed on average 313k and 30k consistency tests on new and cached correspondences respectively. This is only 7.2% of the ~4.7 million tests performed by the batch solution at each recognition step. In the powerplant experiment, both recognizers can test for consistencies very quickly as the target map contains only a small number of segments.

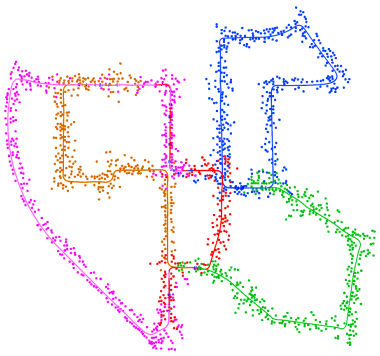

Figure 10: A top-down illustration of a common representation constructed in real-time on a single computer, by simulating five autonomous vehicles equipped with Velodyne HDL-64E sensors. The segments centroids are colored according to their associated vehicle trajectories.

The effect of different thresholds θ_Δ on the consistency distance for caching is compared Fig 9. Whereas high values of θ_Δ result in a lot of cached candidates, requiring a high number of consistencies tests, small values increase the number of invalidated cache entries. Therefore, the best setting for the radius is a trade-off between number of cached candidates and invalidation frequency.

C. Large-scale multi-robot experiment

This final experiment shows that the performance of the presented incremental approach enables us to address challenging SLAM scenarios. In order to simulate a multi-robot scenario, sequence 00 of the KITTI odometry dataset is split into five sequences which are simultaneously played back for a duration of 114 seconds. The data generated by five Velodyne HDL-64E sensors are processed in real-time on a single computer, in order to identify sufficient global associations to link the trajectories.

Although successful results were demonstrated in multi-robot scenarios with the batch approach [2], we found that it did not scale well to this higher number of vehicles. Specifically, its lower processing rate led to the extraction of too few segments, preventing the association of some trajectories. Contrastingly, our incremental approach successfully closed more than 100 loops which enabled to construct, in real-time, the common representation illustrated in Fig. 10. Similarly to the timings presented in Table II, 77.8ms$\pm$38.2ms were on average required to perform a localization step. This shows that the incremental approach effectively managed the higher number of voxels created at each step (5.2k vs 2.1k on average) which is caused by the delay when sequentially processing data from multiple sensors.

V. CONCLUSION

In this work, we presented a novel incremental approach for performing localization in 3D point clouds. We started by identifying the most computationally demanding operations in our previous pipeline. Then, efficient solutions were proposed for the individual sub-problems of the underlying segment extraction and matching technique. Unlike previous works, this approach maintains a segmented local map and performs geometry verification incrementally, reducing the computational burden and then allowing for more frequent localizations. The speed-up achieved allows for localizations at 10Hz, enabling real–time operation of 3D point cloud based SLAM systems. Our results indicate that the proposed approach could allow for seamlessly performing map-tracking, i.e. localization in a known map with a constrained search space based on the current position estimate. In the same direction it is worth to further investigate the application of our incremental recognition scheme to geometric verification for vision-based SLAM. Furthermore, whereas the present work considered eigen-based segment descriptors, it would be interesting to investigate incremental updates of learning-based descriptors that can potentially gain discriminative power and reliability over time.

REFERENCES

[1] R. Dubé, D. Dugas *et al.*, "Segmatch: Segment based place recognition in 3d point clouds," in *IEEE Int. Conf. on Robotics and Automation*, 2017.

[2] R. Dubé, A. Gawel *et al.*, "An online multi-robot slam system for 3d lidars," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2017.

[3] T. Whelan, L. Ma *et al.*, "Incremental and batch planar simplification of dense point cloud maps," *Robotics and Autonomous Systems*, vol. 69, 2015.

[4] K. Tateno, F. Tombari, and N. Navab, "Real-time and scalable incremental segmentation on dense slam," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2015.

[5] R. Finman, T. Whelan *et al.*, "Efficient incremental map segmentation in dense rgb-d maps," in *IEEE Int. Conf. on Robotics and Automation*, 2014.

[6] N. Ayache and B. Faverjon, "Efficient registration of stereo images by matching graph descriptions of edge segments," *International Journal of Computer Vision*, vol. 1, no. 2, 1987.

[7] T. Sattler, B. Leibe, and L. Kobbelt, "Scramsac: Improving ransac's efficiency with a spatial consistency filter," in *Computer vision, 2009 ieee 12th international conference on*, 2009.

[8] H. Chen and B. Bhanu, "3d free-form object recognition in range images using local surface patches," *Pattern Recognition Letters*, vol. 28, no. 10, 2007.

[9] H. Hoppe, T. DeRose *et al.*, *Surface reconstruction from unorganized points*. ACM, 1992, vol. 26, no. 2.

[10] J. Poppinga, N. Vaskevicius *et al.*, "Fast plane detection and polygonalization in noisy 3d range images," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008.

[11] T. Rabbani, F. Van Den Heuvel, and G. Vosselmann, "Segmentation of point clouds using smoothness constraint," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, 2006.

[12] A. Cramariuc, R. Dubé *et al.*, "Learning 3d segment descriptors for place recognition," in *LLM-IROS*, 2017.

[13] M. Weinmann, B. Jutzi, and C. Mallet, "Semantic 3d scene interpretation: a framework combining optimal neighborhood size selection with relevant features," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, 2014.

[14] M. G. Gollub, R. Dubé *et al.*, "A partitioned approach for efficient graph–based place recognition," in *PPNIV-IROS*, 2017.

[15] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," *Algorithms and computation*, 2010.

[16] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2012.