IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 13, 2021, accepted May 11, 2021, date of publication May 21, 2021, date of current version June 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3082848

Template Matching for 3D Objects in Large Point Clouds Using DBMS

DÁNIEL VARGA[1], JÁNOS MÁRK SZALAI-GINDL[1], BENCE FORMANEK[2], PÉTER VADERNA[2], LÁSZLÓ DOBOS[3], AND SÁNDOR LAKI[1], (Member, IEEE)

[1]Department of Information Systems, Eötvös Loránd University (ELTE), 1117 Budapest, Hungary
[2]Ericsson Research, 1117 Budapest, Hungary
[3]Department of Physics of Complex Systems, Eötvös Loránd University (ELTE), 1117 Budapest, Hungary

Corresponding author: Sándor Laki (lakis@elte.hu)

Dániel Varga and János Márk Szalai-Gindl contributed equally to this work.

The work of Dániel Varga was supported by the European Union, co-financed by the European Social Fund, through the Project "Integrated Program for Training New Generation of Researchers in the Disciplinary Fields of Computer Science," under Grant EFOP-3.6.3-VEKOP-16-2017-00002. The work of János Márk Szalai-Gindl and Sándor Laki was supported by the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme (National Challenges Subprogramme) Funding Scheme, through the Project Application Domain Specific Highly Reliable IT Solutions, under Grant TKP2020-NKA-06.

ABSTRACT LIDAR and depth cameras have gone through a profound technological evolution, making large-scale recording of 3D point cloud data possible which raises new challenges for data processing. Most of the existing 3D point cloud processing methods were developed to work properly when the entire data set fits into the memory of a single server. When point clouds are significantly larger than the main memory and data are only available on slow storage, new approaches are necessary. In this paper, we propose a DBMS-based point cloud processing pipeline that solves the template matching problem, i.e., finding the – potentially multiple – occurrences of a small query point cloud in an extensive scene data set that is preprocessed and stored in a database. The storage layer uses a compact and novel data representation to exploit the benefits of efficient indexing structures whereas the query algorithm consists of a novel combination of existing point cloud processing and matching methods. To the best of our knowledge, this is the first template matching proposal in the literature that exploits the benefits of databases.

INDEX TERMS 3D point cloud, template matching, database, registration, PCA.

I. INTRODUCTION

In the past years, LIDAR sensors and 3D sensors built into mobile devices, such as RGBD cameras, have gone through a significant evolutionary step, enabling the easy collection of massive 3D point cloud data with high resolution. These data can be exploited by novel applications, including robot navigation, localization, object search, augmented (AR) and virtual reality (VR) [1]–[3]. The increased volume and accumulation of such data sets cannot be handled by in-memory methods and poses new challenges against efficient data handling and processing, requiring more robust and scalable solutions.

The literature distinguishes 2.5D and real 3D point clouds with very different characteristics. The former category includes, for example, airborne laser scanning data – basically height maps – where the Z coordinates change very little with respect to X and Y and covering of background objects is unlikely or unimportant. Whereas the real 3D category includes, for example, indoor LIDAR scans or scans of objects from all directions, where the point cloud varies along all coordinates similarly. In addition to the 3D positions, points often have additional attributes such as color that can also be used by the different search and matching methods. Note that most GIS use cases, where databases are extensively applied, work with 2.5D data. There is only a limited number of related work on the DBMS context of real 3D point cloud processing, c.f. Sec. II.

Template matching aims to solve the problem of finding the occurrences of a small point cloud, the *query* (e.g., a scan of an interesting object, a point cloud captured at a specific location in a building, etc.) in a significantly large point cloud we will refer to as the *scene*. All existing template matching algorithms [3] in the literature assume a scene point cloud that can entirely be loaded into the memory, thus the query matching can be executed quickly. Nowadays, there are large-scale point clouds where in-memory solutions are not feasible, and the application of DBMSs seems a natural choice for

The associate editor coordinating the review of this manuscript and approving it for publication was Qichun Zhang.

accelerating various preprocessing and query tasks that are necessary to solve problems such as template matching.

In this paper, we propose a DBMS-based template matching pipeline that uses a compact and novel data representation and exploits the benefits of efficient indexing structures. The pipeline can be split into offline and online phases. The offline phase includes the preparation of scene data (e.g., noise filtering, sub-sampling, keypoint selection, feature vector computation, dimension reduction), loading into the database, and generating indexes. Note that this phase is time-consuming because of the vast amount of 3D points. In the online phase, a small query point cloud first goes through the preprocessing steps of the offline phase to obtain a uniform data representation needed for the selection of candidates and the application of the matching algorithm to get the query results. Note that this phase is accelerated by the various mechanisms of DBMSs to get reasonable query execution time. The evaluation has been carried out with our prototype implementation relying on PostgreSQL database. To the best of our knowledge, this is the first study that investigates how DBMSs can accelerate the resolution of template matching queries.

In summary, our contributions are:

1) We define a special case of template matching problem where the 3D scene data is available in advance (e.g., in a database) and thus can be pre-processed offline, while the template (or query) point cloud to be found in the scene arrives online.
2) We propose a novel feature-based registration pipeline to solve this template matching problem.
3) A prototype of the proposed pipeline has been implemented using PostgreSQL.
4) Our evaluation shows that the proposed DBMS-based method has several practical benefits: trivially scalable, able to work with datasets which do not fit in memory, using index structures for fast searches, etc.

The paper is organized as follows: In Section II, we briefly overview the related literature. In Section III, we define the problem and talk about how a general solution can be divided into offline and online phases. Section IV introduces the main elements of the propose template matching pipeline. The evaluation of the method and the discussion of the results are presented in Section V. Finally, Section VI summarizes the main results and concludes the paper.

II. RELATED WORK

Though most of the recent 3D point cloud processing methods including template matching assume point cloud data that can entirely be loaded into the memory, few papers propose the use of database management systems (DBMS) to store, prefilter or process such data sets.

Oosterom *et al.* [4] propose the application of DBMS as convenient and flexible alternatives for storing point clouds. Their paper has evaluated different database systems such as PostgreSQL, MonetDB, Oracle and LAStools, using an airborne data set captured in the Netherlands. It compares different techniques for storing point cloud data. The authors propose the flat table model where each point is stored in a simple row of the database. The data can easily be organized, filtered or ordered for various use cases. E.g., for ordering the Hilbert- or Morton-code of the points can be used. On top of the ordered rows blocks can easily be defined, having many benefits: 1) spatial dependence, 2) simple compression, 3) reduced overhead and 4) data caching. This paper also proposes a query algorithm relying on Morton-codes and their relationship to quad-trees.

Cura *et al.* in [5] work with point blocks instead of individual points and focus on how compression techniques can be used to reduce the size of these massive data sets without sacrificing the query execution times. Accordingly, grouping of 3D points has to be compatible with relevant queries of the given use case. Block sizes are not necessarily uniform, can be determined adaptively reflecting the local properties of the point set. E.g. grouping based on point density or prior classification are both possible. In their paper, they only consider point clouds with homogeneous density (e.g. air-bone LIDAR data) which – as they also mention – cannot generally be hold for indoor point cloud recordings. To reduce the query execution time they apply B- and R-trees, and function-based indexing.

In [6], Meyer and Brunn shows how 3D point clouds can be integrated into PostgreSQL/PostGIS with the use of the Pointcloud extension and the functions of Point Data Abstraction Library (PDAL). They form blocks from the closest points in the 3D space, and store each block as a table row in their database. During importing the point cloud into the database, they use PDAL to group points into blocks. PDAL offers two methods for grouping: a regular and an irregular one. However, both have been designed for 2D point sets, applying 2D tessellation on 3D point clouds by simply omitting the Z-coordinates. The authors combine these 2D approaches by introducing limits on the Z-coordinates in the groups, resulting in a real 3D tessellation method. They consider queries which return data points inside a given polyhedron. However, the Pointcloud extension and PostGIS do not implement algorithms to answer such queries, so they apply a workaround and show the proposed 3D tessellation is definitely worthwhile, but they do not clearly conclude whether regular or irregular tiling is better.

Based on a thorough review of related work, we can conclude that though there are approaches and ideas relevant to our work, we have not found any articles in the context of DBMSs that address our specific task of template matching in a scene point cloud. Note that our study relies on PostgreSQL DBMS, and since the benefit of organizing points into blocks is not conclusive, a flat table model is used without the Pointcloud extension (see more details in Section IV-D).

III. A GENERAL SOLUTION FOR FEATURE-BASED TEMPLATE-MATCHING PROBLEM

A. PROBLEM DEFINITION

Let $\mathcal{S}$ denote the scene representing a large-scale 3D point cloud (e.g., high resolution recordings of one or more rooms,

houses, cities or even larger areas) stored in a database, and $\mathcal{Q}$ the template (or query) point cloud which is generally much smaller and represents an object (e.g., a household object, a car or a snapshot about a local environment) or a smaller area. In this context, the goal of template matching is to find all occurrences of $\mathcal{Q}$ in scene $\mathcal{S}$ and estimate the transformations $T_1, \ldots, T_n$, where $n \geq 0$ is the number of query occurrences in $\mathcal{S}$, and T_i ($i \in [1..n]$) is a transformation fitting query point cloud $\mathcal{Q}$ to its i^{th} occurrences in scene $\mathcal{S}$. A template-matching is a special case of the point set registration problems where additional presumptions hold:

1) $\mathcal{S} \gg \mathcal{Q}$, meaning that the scene point cloud is orders of magnitude larger than the template point cloud (e.g., an object in a house, a car in a district, a building in a city).
2) The number of occurrences is not known in advance.
3) Scene $\mathcal{S}$ is known in advance, enabling preprocessing steps as offline phases, but the template point cloud only becomes known during the online processing phase.

B. GENERAL PIPELINE

In this section, we define the main steps of a general template-matching pipeline and also indicate which steps could benefit from integration with databases. According to our problem definition, the pipeline can be split into offline and online phases.

1) **Offline phase.** Since the scene point cloud is available in advance, preprocessing steps (e.g., transformations, feature computations, indexing) can be performed offline to make the online phase fast. These includes downsampling, detecting and removing outliers, estimating and orienting normal vectors, and any other operations which can speed up the online processing. Feature-based pipelines can also benefit from offline processing: keypoint detection, calculating feature descriptors, dimension reduction of feature vectors and creating point pairs or point triplets. Results of this phase including the point cloud and the calculated metadata with appropriate indexing structures can be loaded into a DBMS.
2) **Online phase.**
   a) **Preprocessing of template cloud.** During online preprocessing we can perform those computations which necessary to find the template occurrences in the scene. We can do the same preprocessing steps as on the scene cloud. In this case, however, we need to pay attention to the computational cost, because it is included in the total time of query evaluation. Note that if a specific use case has a requirement on the query execution time, the preprocessing steps can be accelerated by choosing faster but less accurate methods or changing the parameters of the algorithms. In these algorithms there is a trade-off between run time and reliability.
   b) **Correspondence estimation.** To estimate transformations we need to find all possible matches. One way to do this could be to determine point-to-point correspondence (or correspondences between point pairs or point triplets [7]). Due to the number of possible correspondences, it is advisable to use some kind of heuristics. As we will see later, different methods may use very different approaches. Because of the large size of the scene cloud, this step can be a good place to use a database.
   c) **Transformation estimation and validation.** While some methods perform transformation estimation in one step, feature-based registration pipelines often generate an initial transformation, and then refine in with some kind of iteration algorithm. If there is an initial transformation, the task can be interpreted as a point set registration problem. There are many algorithms which are able to produce an accurate transformation from a not so accurate initial transformation [8]. One of the best-known algorithm is the Iterative Closest Point and its improved versions [9]. After the transformation estimation the next step is the transformation validation in which we need to determine the goodness of these. In general goodness is defined by some kind of score which shows what is the percentage of overlap between the two clouds (usually called alignment or fitness score). The key to successful registration is to adjust well the acceptance threshold of the goodness score. Defining a goodness score and its acceptance threshold are not trivial tasks. There are several characteristics need to be consider: noisiness and density of the input clouds, object occlusions in the scene cloud.

IV. PROPOSED METHOD

As mentioned previously, we assume a scene point cloud that cannot fit into the memory of a single server and thus the application of existing in-memory template matching methods are not feasible. To solve this problem at large scale, we propose a DBMS-based pipeline consisting of the following steps (see Figure 1):

1) Data preprocessing
   a) Outlier detection and normal vector computations (Section IV-A)
   b) Keypoint detection (Section IV-B)
   c) Generating feature descriptors (Section IV-C)
2) Correspondence estimation (Section IV-E)
3) Transformation estimation (Section IV-F)

The proposed pipeline consists of online and offline phases. The offline phase includes the preprocessing of the scene point cloud: the detection of keypoints that capture the most informative points of the cloud that reflect the main

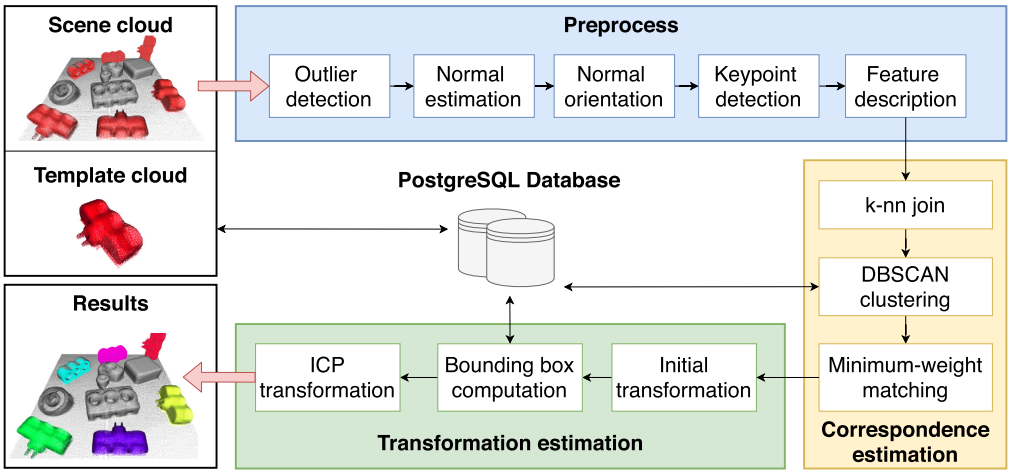


FIGURE 1. An overview of the proposed DBMS-based template matching pipeline.

structure of the scene data properly, and the computation of feature descriptors for keypoints moving the point cloud from 3D to a high dimensional feature space. Note that in addition to the raw 3D points all the results of the offline phase are also stored in the database and used to accelerate the online processing. In the online phase, a query cloud is given as an input. Then all the previous steps are applied to the query point cloud to get a uniform representation in the feature space. Finally, the correspondence and transformation estimations are carried out to answer the query. In this section, we detail the various building blocks of the proposed template matching pipeline and the DBMS-related aspects of its implementation. In other works where the problem definition is similar to ours, the authors divide the offline and online phases differently. Namely, considering the template point cloud processing as a offline step and the scene cloud processing as a online step [3].

A. OUTLIER DETECTION AND NORMAL VECTOR COMPUTATIONS

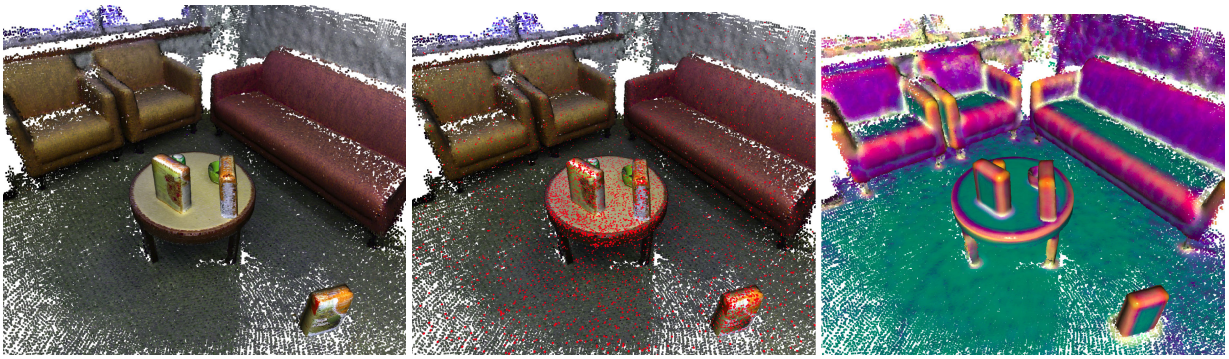
Different point clouds can be captured by devices with various noise properties and thus may have quite different characteristics that should be taken into account during the preprocessing phase. A widely used metric in the literature for characterizing point clouds is the approximated point cloud resolution (*pcr*). It averages the distances between each point and its nearest neighbor in point cloud. Many preprocessing algorithms (e.g., normal estimation, feature description) use *pcr* to determine their internal parameters. Note that *pcr* can be misleading if the typical distance between a point and its nearest neighbor is not uniform. To avoid this, the standard deviation of nearest neighbor distances from *pcr* can be taken into consideration.

Since different capturing techniques and devices add some level of noise to the point cloud, our pipeline starts with applying a simple noise filter first. In our case, it iterates over all of the points in the data set, and for each point locally fits a plane on its neighborhood which can be defined with a radius or the number of nearest neighbors. The algorithm removes outlier points that are farther from the fitted plane than a predefined adjustable threshold. This simple mechanism can easily remove isolated points, but the threshold needs to be set carefully to avoid filtering out important parts such as corners in the data set. This step can be omitted if the input point cloud is not affected by significant noise.

As a second step, the normal vectors, or simply normals, for all points in the cloud needs to be calculated. Similarly to the previous step, the neighborhood of each point in the data set is fitted by a plane, and the perpendicular vector is considered as the normal vector at that point. The radius parameter, which determines the magnitude of the neighborhood, is crucial and has a significant impact on the final result.

Finding the right radius parameters is difficult. We need a radius for the normal estimation and for the feature description too. The values of these parameters strongly depends on the characteristics of the input point cloud, such as density, size, resolution of the capturing device, etc. One solution could be performing grid search on the parameter space [10]. Another solution is to make these parameters depend on the *pcr* or the diameter of the point cloud, and to choose a multiple of these [3]. Despite the problems with *pcr* explained above we decided to choose the latter solution.

After the normal vector was calculated, its orientation should be determined. The normal orientation is a well-known and difficult problem, and many solutions have been proposed in the past few years [11]. The problem is that even two adjacent points in a point cloud may have opposite normal directions. Since feature descriptors [12] generally used for template matching are very sensitive to normal orientations, global consistency needs to be ensured. For this reason, the proposed pipeline applies the well-proven algorithm of Hoppe *et al.* [13] that uses minimum spanning trees to reach globally consistent orientation.



a) b) c)

FIGURE 2. Scene cloud: (a) The indoor dataset (b) Random keypoints (red) in the indoor cloud (c) Color space based on the first 3 dimension (according to the eigenvectors belonging to the three largest eigenvalues).

In the following sections, we assume that a good quality point cloud is loaded into the database. A point cloud has good quality if it is not affected by noise and the typical distance between a point and its nearest neighbor is uniform (fixed resolution). If it is not the case, as with one of our data set used for evaluation, the noise filtering method mentioned previously can be combined with voxel grid filtering to solve the density problem. The voxel grid filter creates a grid in the euclidean space. After applying the filter, every non-empty grid element called voxel is replaced by a single point (centroid or medoid) representing all the points situated in the voxel.

B. KEYPOINT DETECTION

Since we assume a huge scene point cloud, sub-sampling is essential to reduce the computational complexity of the pipeline. Most of the existing approaches for point cloud matching or registration apply techniques for selecting interesting points [14] that are often referred to as keypoints. Numerous keypoint detection algorithms have been published in recent years, some of them are hand-crafted [15], [16], but the most recent ones apply the concept of deep learning [17], [18]. The goal of keypoint detection is to select unique and important points that capture the main characteristics of the entire point cloud, and thus matching algorithms can work with the reduced number of keypoints instead of the whole point cloud.

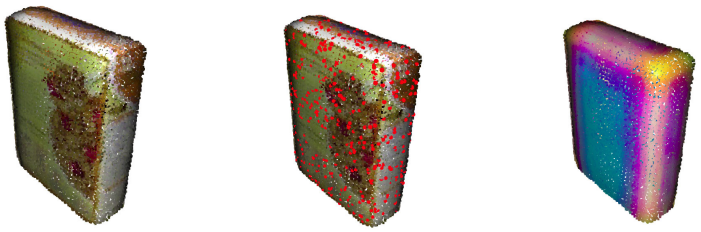
For this purpose, we carried out experiments with the state-of-the-art keypoint detection methods, but none of them worked reliably enough to be used as the basis of indexing nor produced consistent results with from the data sets used in this paper. The main challenge posed by our pipeline is that it includes a clustering step that does not tolerate sparse keypoints these methods generally result in. However, the random keypoint selection proved to be suitable for our template matching pipeline. Note that the selected points are not real keypoints since they are not particularly interesting points, but this technique is good for sub-sampling. Depending on the input point cloud, and based on our experiences selecting 2-5% of all the points as keypoints is enough for successful matching. Since keypoint selection should not only be applied on the scene data set but the query data set as well, random selection has the advantage of fast execution. Figure 2 depicts one of the data sets we use for testing. Figure 2/(a) shows all the points of the point cloud while Figure 2/(b) illustrates the keypoints provided by random sampling from the same data set. One can observe that the keypoints marked by red points capture the key character of the original data set.

In contrast, Vock *et al.* [3] use a different approach. To reduce the potential sample candidates, they detect edge points, and consider only these to make transformation hypothesis. In our experiments, using edge detection instead of random keypoints resulted is slightly better matching probabilities, Unfortunately, edge detection has too high computational cost to keep the online processing times low enough.

C. FEATURE DESCRIPTION

Keypoints result in a compact representation of the original point cloud. To compensate the information loss caused by the sub-sampling, for each keypoint we calculate a feature vector describing its local context within a predefined neighborhood. To determine the neighborhood size, a practical rule of thumb is to use the multiple of *pcr* to set the radius, but the scale of the point cloud also needs to be taken into account to get meaningful feature vectors. Feature vectors can then be used for measuring the similarity between two keypoints (e.g., if we have a corner point in the query point cloud, we can find similar corner points in the scene point cloud). There are many existing approaches for calculating feature descriptors in 3D point clouds. According to recent survey papers [19], [20] we have chosen the Fast Point Feature Histogram (FPFH) method [12] that results in descriptive feature vectors of only 33 dimensions and is computationally efficient. According to the comparative evaluations it provides a good balance between matching accuracy and computational efficiency.

However, 33 dimensions for each keypoint is too high to get efficient query speed in the database. According to Prakhya *et al.*, Principal Component Analysis reduce the dimensionality of the 3D descriptor and retains



a) b) c)

FIGURE 3. Query cloud: (a) The box object which is on the table in the indoor dataset (b) Random keypoints in the box cloud (c) Color space based on the first 3 dimension.

its descriptiveness [21]. Therefore the proposed pipeline applies Principal Component Analysis (PCA) for dimension reduction of the feature space. After PCA we selected the eigenvectors for the 5 largest eigenvalues, reducing the 33 dimensions of the feature space to 5. Figure 2/(c) illustrates the three largest principal components in the color-space. Here, the color-space means that the red, green and blue color ingredients correspond to the values of the first, second and third principal components normalized into interval [0, 1], respectively. Note that in this figure we use the entire point cloud for better illustration, but in the pipeline the features are only stored for the keypoints. One can observe that the three largest principal components can solely distinguish edges, corners and planes. Figure 3 depicts the same for a simple box object: (a) the raw 3D point cloud, (b) selected keypoints in the cloud, and (c) the points colored with the three largest principal components.

For a similar purpose, Vock *et al.* [3] used a 4-dimensional point-pair feature vector, instead of a feature descriptor for every point. The use of PCA is not required for a 4-dimensional feature vector, but generating all point pairs for the template cloud would have been more expensive.

D. DATABASE PREPARATION

The Generalized Search Tree (GiST) [22], as its name suggests, is a template index structure that is an abstraction of search trees such as the well-known B-tree or R-tree. The GiST framework is implemented in PostgreSQL DBMS. It allows to build an index over a column with user defined types for which the so-called support functions required for the index structure and the related algorithms are also implemented. Using these functions, the GiST framework provides other important methods for the tree, such as search, insert, etc. By creating a so-called operator class[1] within PostgreSQL, it can be indicated that a GiST framework is used and it can be specified which support functions are associated with the custom data type. In addition, the operators can be listed here which are indexable, that is, if these are included in certain clauses (usually WHERE clauses) of an SQL query, the GiST can help in speeding up the evaluation.

Both 3D points and feature vectors are stored in the form of `cube` data type[2] in PostgreSQL DBMS. This data type with several useful operators represents multidimensional points or cubes. For our purpose, the most important operators are operator `@>` checking if a cube contains another cube, and operator `<->` that determines Euclidean distance between two cubes. A Generalized Search Tree (GiST) index operator class is also implemented over `cube` values which can be applied to the former operator in `WHERE` clauses, and to the latter operator in `ORDER BY` clauses, and thus it can speed up the search for nearest neighbors. The GiST implementation of the cube extension can be considered as the regular R-tree for multidimensional cubes [23]. It is interesting to note that Pointcloud extension of PostgreSQL DBMS[3] can also be used for storing point cloud data. In contrast to the flat table model we use in our pipeline, points are assigned to blocks that can then be used to create an index on spatial data. However, it is shown in [24] that block-based approaches have high computational overhead (especially constructing and decoding blocks) in most of the use cases. Furthermore, the Pointcloud extension with PostGIS integration also supports the regular R-tree indexing.[4] From our perspective, there is no essential difference between the cube extension and the Pointcloud extension. In the proposed pipeline the DBMS follows the flat table model.

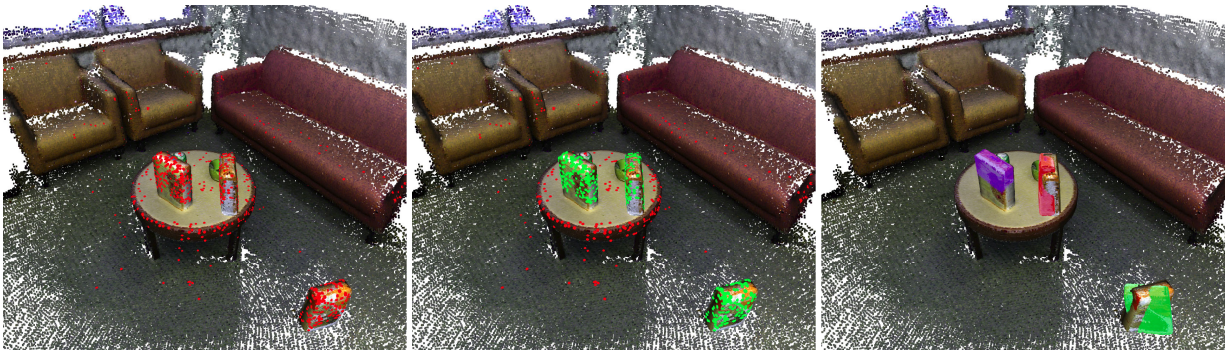
Accordingly, all the 3D points of scene point cloud are stored in table `scene_points_table (id number, coords cube)` while the feature vectors, for those of them that are keypoints, in table `scene_keypoints_table (p_id number, feats cube)` where column `p_id` is a foreign key to `id` of table `scene_points_table`. To accelerate the k Nearest Neighbor (k-NN) join operation, GiST index structures are built on top of `scene_points_table` on column `coords` and of `scene_keypoints_table` on column `feats`, respectively. For query point cloud, we also create two tables, table `query_points_table (id number,`

[1] See https://www.postgresql.org/docs/current/sql-createopclass.html and https://www.postgresql.org/docs/current/xindex.html

[2] https://www.postgresql.org/docs/current/cube.html

[3] https://github.com/pgpointcloud/pointcloud

[4] https://postgis.net/workshops/postgis-intro/indexing.html



a) b) c)

FIGURE 4. (a) Keypoints which remained after the knn-join (b) Keypoints after the DBSCAN clustering (green: good cluster points, red: bad cluster points) (c) Transformed clouds based on transformations created by minimum weight matching.

`coords cube)` and table `query_keypoints_table (p_id number, feats cube)` with GiST index structures for similar purposes. (Note that in table `query_points_table`, the transformed query point cloud is stored before the final refinement step using the ICP method, see below.)

E. CORRESPONDENCE ESTIMATION

A stored function `get_feat_neighbors` has been created to execute k-NN join operation between the scene point cloud (`scene_keypoints_table`) and the query point set (`query_keypoints_table`). It returns a 'virtual' table in which each tuple is in the form of `(qf_id int, sf_id int, nn_idx int)` where

- `qf_id` identifies a query point,
- `sf_id` identifies a scene point,
- `nn_idx` represents the rank of the scene point among the nearest neighbors of the query point in the feature space.

Note that the function `get_feat_neighbors` iterates over all the records of `query_keypoints_table` in a loop and finds k nearest neighbors of `feats` of a given record, using the metric operator `<->` in `ORDER BY` clause and `LIMIT` k clause to take benefit of the GiST. Otherwise if the above-mentioned loop is omitted and 'real' join is applied, the query optimizer of PostgreSQL may use sequential scan.

After the k-NN join we perform a DBSCAN clustering [25] on all scene keypoints from the previously returned virtual table. DBSCAN is a density-based spatial clustering, and we use it with Euclidean norm. One of the most important parameter of the algorithm is the ϵ (*epsilon*). It defines a distance which is used to get the points' neighborhood. Basically, this parameter decides which points belong to the same cluster. Since we prefer one cluster per object, keypoints cannot be farther from each other than ϵ. This is the reason why we cannot use the well-known keypoint detectors that provide spare keypoints that in many cases are farther than the ϵ threshold. The DBSCAN step computes a set of clusters, whose keypoints are very similar to the keypoints in the query point cloud.

After the k-NN join operation and DBSCAN, to discover the best correspondence set between keypoints of the query point cloud and a cluster, we construct a bipartite graph (V_s, V_q, E) over the keypoints such that each node of vertex set V_s (V_q) represents a scene keypoint of the cluster (query keypoint). There is an edge e with weight w_e between a node sn of V_s and a node qn of V_q if and only if the feature vector of the scene point represented by sn is the w_e^{th} nearest neighbor of the feature vector of the query point represented by qn, as illustrated in Figure 5. Note that the value of w_e can be between 1 and k. To find a minimum-weight matching, we use the Hungarian algorithm [26]. (Finding the best feature matching based on minimum-weight matching of bipartite graph is not a new idea, a similar approach can be seen in [27].)

F. TRANSFORMATION ESTIMATION

Now, we have a correspondence set between every cluster and the query point cloud. To obtain the transformations, the [28] method is used, calculating a transformation matrix matching the two point clouds. This matrix is then used to create a transformed query point cloud that is stored in table `query_points_table`. Figure 4 illustrates the result of these transformations. As you can see, the query point cloud does not fit perfectly into the occurrences of the object in the scene cloud. Usually, the initial transformation which based on the correspondence set doesn't achieve perfect fit. Therefore, we have to refine the transformation using an accurate point set registration method [9].

For refining the transformation, our pipeline applies a well-known iterative method, called Iterative Closest Point (ICP) [29]. The goal of ICP is to minimize the distance between two point clouds. The algorithm works with two point clouds as input, and computes a transformation that matches the two cloud. The algorithm consists of 3 steps: (1) for each point from one point cloud it searches for its nearest neighbor from the other point cloud. (2) the algorithm estimates a transformation which minimize a metric error, usually the sum of squared distances between point pairs from the previous step. (3) it applies the transformation to the

TABLE 1. Datasets used in the evaluation. The indoor dataset contains many outlier points, thus in the preprocessing step many points was removed. The last column shows how many times an object occurs in the corresponding scene cloud.

Cloud name	Number of points	Number of points after pre-processing	Number of keypoints	Object occurrences in the scene cloud
Indoor scene	1 108 688	537 428	10 811	3
Indoor box	75 911	27 358	538	-
TLESS scene	771 781	771 781	38 904	6
TLESS object	57 565	57 565	2885	-

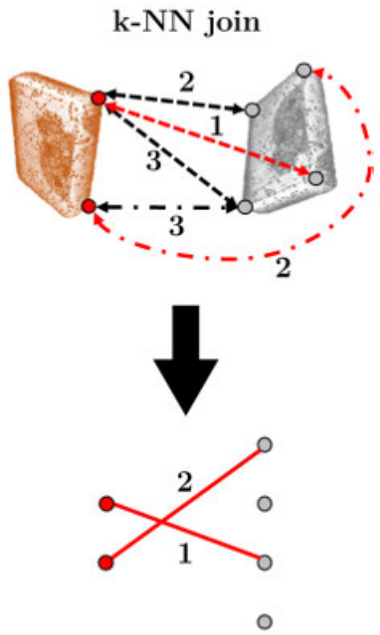


FIGURE 5. Minimum weight matching of bipartite graph. Red nodes: keypoints in the query cloud. Grey nodes: keypoints in a scene cloud cluster. There is an edge between a red and a gray node if the feature vector of the gray point is in the *k* neighborhood of the red point's feature vector. The weight of the edge is *1* if the gray node the first nearest neighbor, the weight is *k* if the gray node the *k*th nearest neighbor. In the figure, the two red lines will be selected as correspondences.

point cloud, then iterates. We can set the maximum number of iteration, or a threshold for the error, which tells the algorithm when to stop.

Nowadays, there are many improved ICP variants and other registration methods exists [30]. Some works have shown that it is difficult to study the convergence properties of ICP variants, because their cost function changes from iteration to iteration, while in case of other methods the convergence to a fixed point is guaranteed [31]. Because ICP has remained a widely used method and is easy to implement, we decided to use it in our work, and we have chosen the point-to-plane ICP variant [32].

Because of the in-memory algorithm of ICP, the use of the whole scene point cloud would not be efficient. Therefore, we cut the environment of the transformed query point cloud from the scene point cloud, using database tables (`scene_points_table` and `query_points_table`), GiST on column `coords` and operator `@>`. To define the cut, we create a bounding box around the transformed query point cloud, and then extend it. To do this, we multiply the diameter of the bounding box with a parameter, called *diametercoefficient*. The higher the *diametercoefficient*, the more time the ICP needs. But if the *diametercoefficient* is too low, the ICP cannot give a good transformation. After we cut the environment from the scene point cloud, based on this extended bounding box, ICP is executed on the cut environment and the transformed query point cloud. Finally, ICP refines transformations for every environment-query pair consisting of a limited number of points.

V. RESULTS

Experiments are performed on an instance of PostgreSQL 10.12 DBMS, running on a single virtual machine (Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz (4 Cores) with 11GB RAM) with Ubuntu 18.04.4 LTS OS.

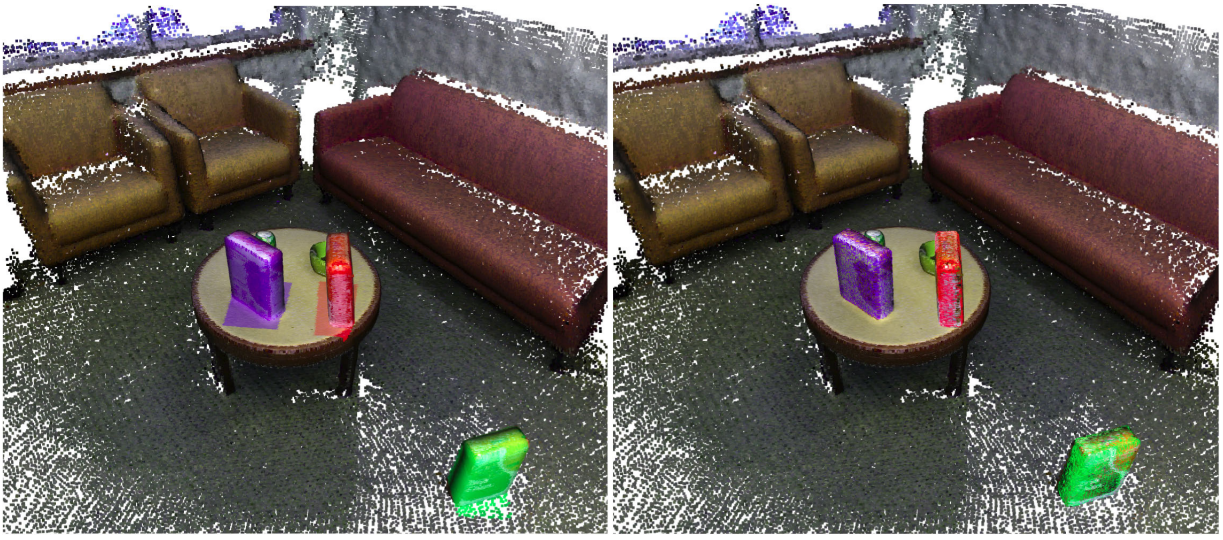
A. EVALUATION

For evaluation we use two data sets described in Table 1. The first is the "RGB-D Scenes Dataset v2" [33] which consists many scenes containing furniture (chair, sofa, table) and objects (cup, box). This data set is good for testing, because it has many outlier points and the density of the point clouds also varies. Thus, it can test the robustness of the proposed template matching pipeline. The indoor scene point cloud contains the query object three times.

The other is the TLESS data set [34]. It consists of 20 object meshes. We selected 6 objects from it, and put it on a plane using the MeshLab software. We put one of the objects on the table 6 times with varying pose (Figure 7/a). After that we used a tool from the Point Cloud Library [35] called "mesh2pcd" which can convert a CAD model to a point cloud using ray tracing operations. This gave us a point cloud which also contains noise. The TLESS scene point cloud contains the query object six times. Figure 7/b shows us the result of template matching with using the TLESS data set.

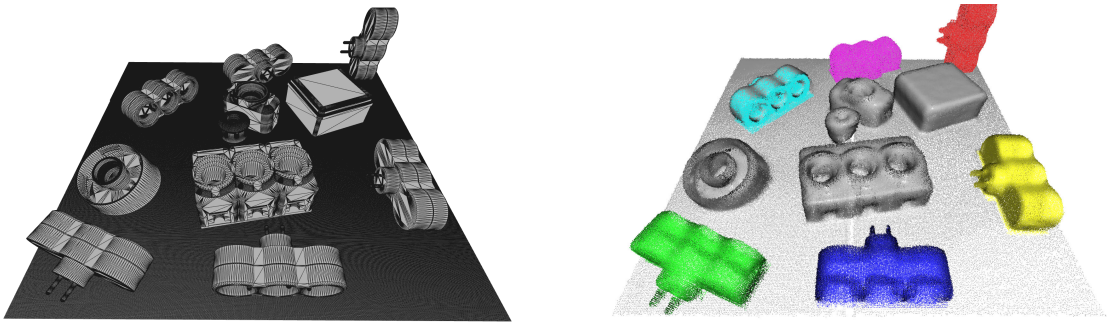
Note that we have not run our method on datasets which do not fit in the memory of our test bed. To do this we would need massive parallelization of feature descriptor computation on GPU because the high cost of it. In this work we did not focus on that.

When the whole process is over we have the final transformations, but not all clusters may have found the correct object. At this point, we can separate the clusters into two classes: true positives and false positives. We call a cluster



a) b)

FIGURE 6. (a) The environment of the query cloud in the scene cloud (b) Final registration result (after ICP).



a) b)

FIGURE 7. (a) The CAD model of the simulated dataset with 6 object occurences (b) Found object occurences in simulated point cloud.

true positive, if its keypoints are on a real object occurrence. Otherwise, the cluster is a false positive. To do this separation, we look at the overlap ratios in Table 2, which shows the percentage of overlap between the query clouds and the scene cloud.

We say that a point cloud fully overlaps with another point cloud, if every point from the first cloud have a neighbor from the other point cloud within a given distance threshold. This distance threshold is also the multiple of *pcr*, like many other parameters. In our experiments we used the double of *pcr*. Other works use similar approaches. The method of Vock *et al.* generate many transformation hypotheses and fast hypotheses validation is essential for them. Therefore, they use a fast voxel-based approach to estimate a goodness score.

Based on our experiments, if a cluster is true positive, then its overlap ratio will be higher than 85%, but in the most cases 90% is also gives good results (e.g. for the indoor data set we got higher overlap ratio than 90% in all true positive cases, but for the simulated data set, we got 89,5% in 1 out of 6 true positive clusters). After we separated the clusters, we need to remove the false positive clusters from the result.

TABLE 2. Overlap ratios for clusters before and after transformation refinement using ICP. The false positive clusters are aggregated while the true positive clusters (real object occurrences) have a number in their names. The data shows that before ICP refinement true positive clusters already have a higher overlap ratio than false positive clusters. This difference increased further after ICP refinement.

Dataset	Cluster name	Overlap (before ICP)	Overlap (after ICP)
Indoor	Cluster #1	84%	100%
	Cluster #2	35%	100%
	Cluster #3	40%	99%
	False positive clusters	8% - 25%	0%-50%
TLESS	Cluster #1	70%	100%
	Cluster #2	83%	100%
	Cluster #3	70%	100%
	Cluster #4	85%	100%
	Cluster #5	53%	90%
	Cluster #6	81%	100%
	False positive clusters	18% - 41%	32% - 50%

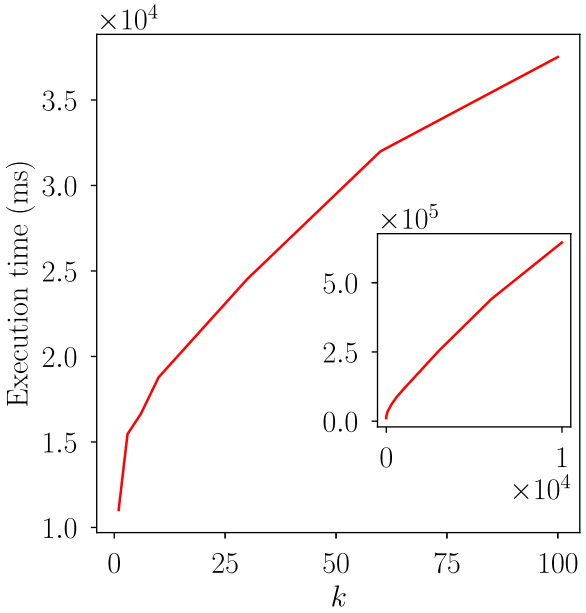


FIGURE 8. The performance of *k*-NN join operation.

It would be great if we could distinguish between true positive and false positive clusters before ICP. Unfortunately, it is not a trivial task. It can easily happen that a false positive cluster has a larger overlap ratio, than a true positive cluster's overlap ratio after the initial transformation. This occurs if the scene cloud contains very similar objects to the query cloud, but these are not real occurrences. A solution could be filtering out the clusters after the initial transformation and before the ICP refinement with a high threshold, but it would exclude many true positive clusters too. For the data sets that we used, we can give a threshold which can separate between true positive and false positive clusters before the ICP. But for a completely new data set it is difficult to predict a definitely good value, even if it exists.

B. THE PERFORMANCE OF *k*-NN JOIN OPERATION

Figure 8 depicts the performance of k-NN join operation. The curve shows logarithmic behavior at low k due to GiST usage and linear behavior at high k because presumably (almost) all index pages are accessed for each query keypoint but index structure is small to fit in main memory so it is cached therefore time cost of the function `get_feat_neighbors` is dominated by the cost of `LIMIT` k clause which is linear on k. The most interesting values of k are the little ones because it is likely that the query object does not occur many times in scene point cloud. For this measurement, we used a random data set which contains 100,000 scene keypoints and 10,000 query keypoints because we do not expect larger data set of a contiguous space. As we can see in Table 1, in fact, it could be smaller. (For the sake of completeness, we also tried this measurement with smaller and larger data sets but these had the same tendencies.) Note that if data set contains, for example, points of a whole building, the search can be parallelized.

C. SCALING SCENE POINT CLOUD SIZE

In this section, we scale the data size of scene point cloud up by a factor of 2 and 4 to examine our method in terms of both elapsed time and storage requirements. The number of occurrences of the template point cloud is always three in the scene point cloud and the number of the potentially overlapping points is essentially the same between template point cloud instances. The increase in the data size applies to the other parts of the scene point cloud. Time-cost measurements can be split into offline and online phases. (The offline phase means that the related steps can be performed with the knowledge of the scene point cloud without the template point cloud. For the online phase, the elapsed times of those steps are calculated which are related to template point cloud.) The elapsed times of the offline steps are the followings:

1) loading the source file of the scene point cloud into memory to calculate keypoints and feature descriptors,
2) determination of the random keypoints,
3) calculating the feature descriptors,
4) storing all the 3D points of scene point cloud in a table and the feature descriptors, for those of them that are keypoints, in another table, furthermore, building index structures of the tables (a B-tree over the primary key and GiST index structures over the 3D points and the feature descriptors).

The online steps include the following time costs:

0) preprocessing steps which are the same preprocessing steps like on the scene point cloud (see the offline steps)
1) performing k-NN join query between the tables of the scene and template feature descriptors, respectively
2) executing DBSCAN clustering on all scene keypoints from the result set of the k-NN join query
3) finding minimum weight matching per cluster based on `nn_idx` (see Sec. IV-E)
4) performing initial transformation based on correspondences which are selected by minimum weight matching
5) determination of the context of the transformed point cloud in the scene point cloud (retrieve scene points from the database which are enclosed by the enlarged bounding box of the transformed point cloud)
6) executing ICP to register the template point cloud into the previously mentioned context.

Figure 9 shows the offline time costs. One can observe that the most time-consuming part is the calculation of feature descriptors. It should also be noted that determining keypoints can generally take much longer, but this is negligible for our method, since random selection of keypoints is the fastest way. Furthermore, it should be noted that we compute the feature descriptors for all points, because we project the space of the original feature descriptors into a lower dimensional space using PCA, in order to achieve a better utilization of storage. Creating tables and indices takes a similar amount of time as the current calculation of the feature descriptors.

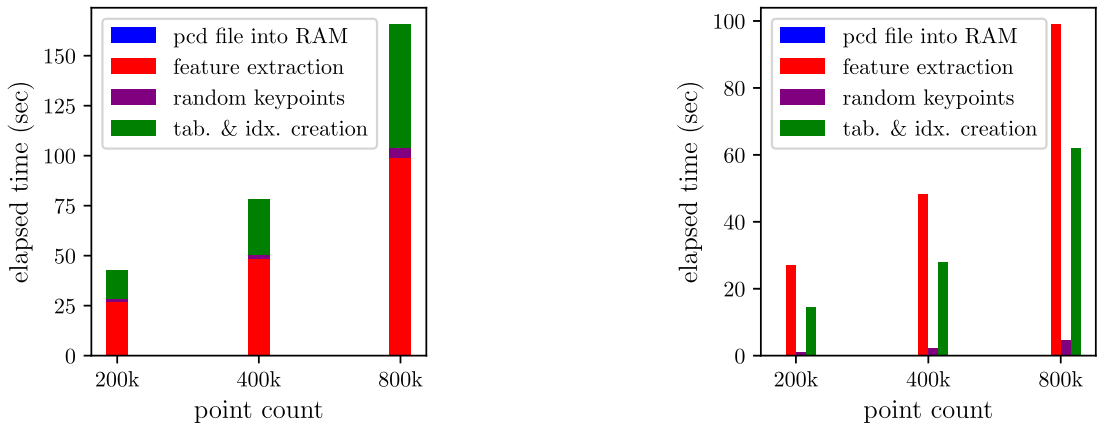


FIGURE 9. Scaling plots of the offline time costs. It shows that the most time-consuming parts are the calculation of feature descriptors and table / index creations.

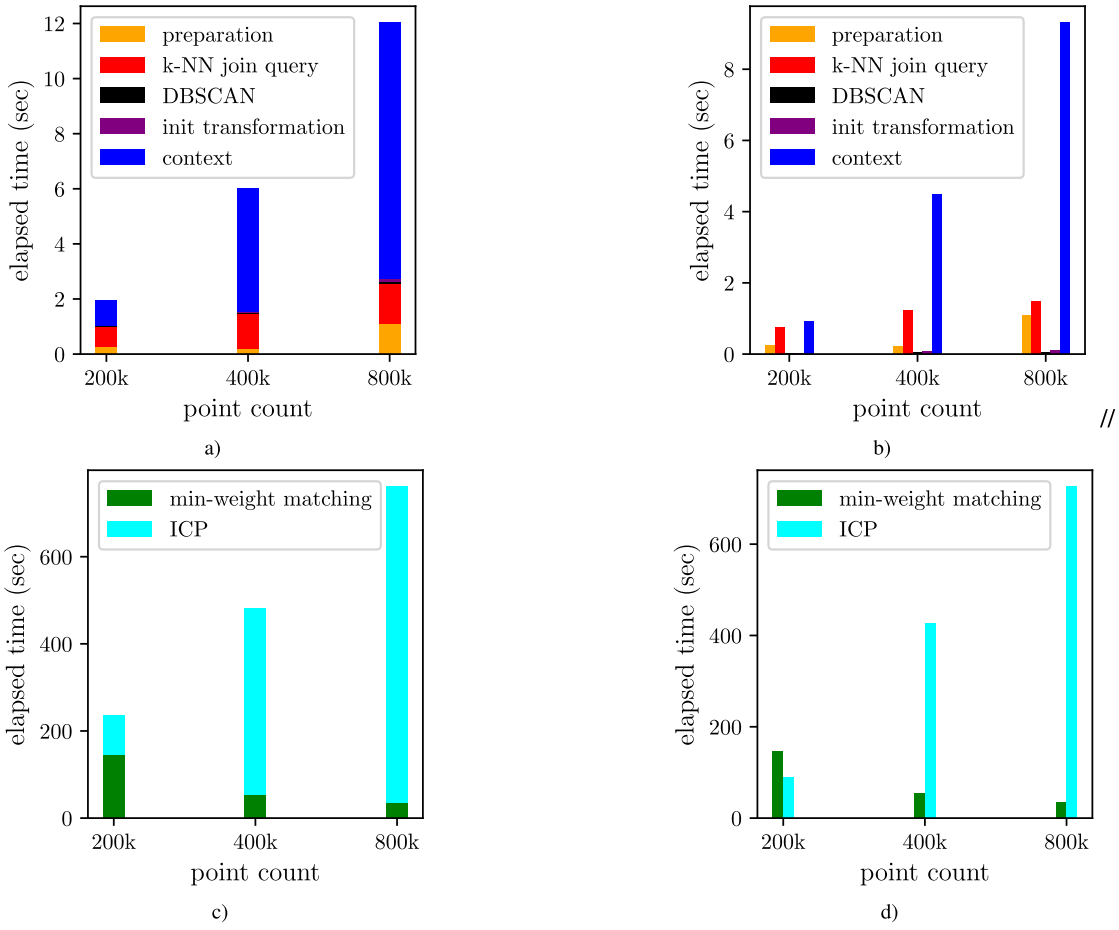


FIGURE 10. Scaling plots of the online time costs. The top panel shows the smaller time-consuming parts. The bottom panel shows that the minimum weight matching and ICP method take more amount of time by two orders of magnitude than the time costs in the top panel.

Turning to online time costs, identifying the point cloud context within scene point cloud obtained by the initial transformation based on correspondences between the keypoints is the most time-consuming step in Figure 10 (a) and (b). A box query is executed on the scene point table using the GiST index structure which is built on top of the table on the 3D points to retrieve data of this context. The larger the scene point cloud is, the more time-consuming this query is

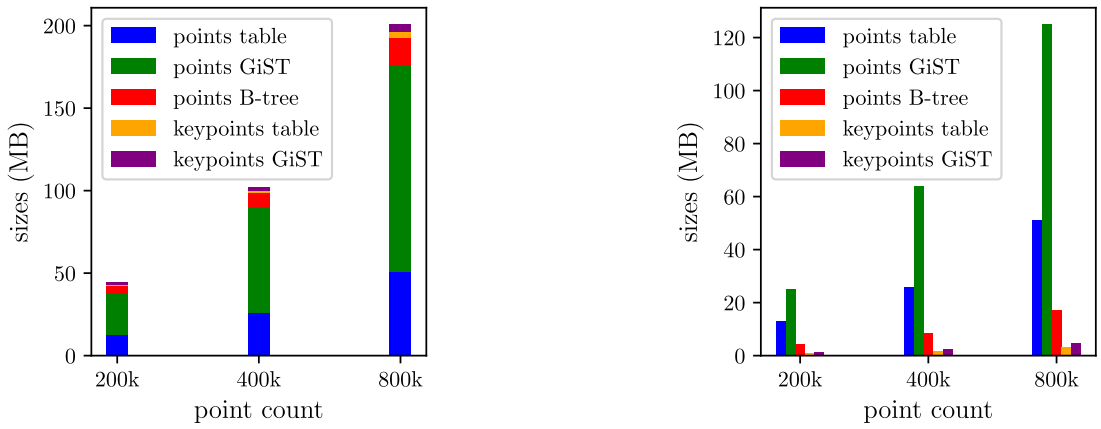


FIGURE 11. Scaling plots of the storage requirements. The GiST index structures of the points require the most space.

(cf. GiST scaling in Figure 11). The time cost of k-NN join is not so pronounced at the given sizes but it also depends on the paging technique used by the DBMS. Further studies on the k-NN join can be found in Section V-B.

Minimum weight matching and ICP method take more amount of time by two orders of magnitude than the time costs mentioned by the previous paragraph, therefore these steps are examined independently of the previous ones. Applying ICP method to each point cloud context within scene point cloud obtained by initial transformations, Figure 10 (c) and (d) show that the larger the scene point cloud is, the longer the total run time of ICP step is. To do this, it should first be stated that the more points the scene point cloud contains, the more clusters DBSCAN clustering is expected to result because the more likely that the feature descriptors of the template keypoints can join such feature descriptors which are not in template point cloud occurrences of the scene point cloud. (Of course, if there were no random keypoints, then the situation would be different, for example, there were no correspondence candidates located on the floor etc.) Our method tries to transform template point cloud for each cluster and ICP commonly takes much longer in the "hopeless" cases because false positive cluster usually contains few keypoints. This explains the relationship between the size of the scene point cloud and the total run time of ICP step. Further investigation showed that, for a given cluster, the Pearson's correlation coefficient of the number of correspondence candidates and the elapsed time of minimum weight matching is 0.96196 (p-value: $8.7636 \cdot 10^{-6}$), more precisely, the matching runs are scaling cubically with the number of candidates in the cluster. Obviously, true positive clusters are expected to have many more matching candidate point pairs, therefore the running time is significantly more for these clusters than for false positive clusters. It is also important, when the size of the scene point cloud increases, then the total time of matching decreases because our experiences selecting 5% of all the points as keypoints but the occurrence number of the template point cloud is fixed, therefore fewer and fewer keypoints fall into true positive clusters. This explains the results.

Figure 11 gives an idea of the scaling of storage requirements. The GiST index structures require the most space, the storage requirement of the scene points is comparable to this, and the other needs are dwarfed.

D. FUTURE WORK

One can see that some parts of the proposed template matching pipeline (e.g., k-NN join) can take advantage of databases. The DBSCAN clustering could also run in a database, but as far as we know, currently, there is no DBSCAN implementation for three dimensional data in PostgreSQL. Our future plan includes its implementation for 3D points. Note that in our prototype pipeline, DBSCAN has been executed outside the database.

According to the evaluation results, our method is robust to noise, but we did not test it with partially overlapping point clouds or with point clouds from different capturing sensors.

One of the biggest challenges is to find the right parameters for the algorithms: radius for normal estimation, radius for feature description, k parameter of k-NN join, ϵ parameter of DBSCAN, *diametercoefficient* for bounding box extension, and threshold to distinguish between true positive and false positive clusters. Almost all of these parameters depends on the characteristic of input point clouds. Many of these parameters can be calculated automatically when we get to know the query cloud, but it is difficult to give good values before knowing the density, scale, and noisiness of it. Important to note that good results are achievable even without the optimal parameters, usually at the cost of online runtime increase. A well-known problem is that if two query objects are too close in the scene point cloud, DBSCAN clustering may recognize it as only one cluster. Thus, the method can only find one object from the two. Further work is needed to solve this problem and automatically determine the correct parameters.

Finally, our most important plan is to test the our method with really large input clouds, and compare it with similar methods [3].

VI. CONCLUSION

In this paper, we introduced a template matching pipeline that exploits the benefits of DBMS, enabling to process large scene point clouds that is not possible with existing in-memory solutions. Several phases of our method can be executed in the database (e.g., k-NN join, DBSCAN). The performance of our PostgreSQL-based prototype implementation has been evaluated with simulated and real-world data sets. To the best of our knowledge, this is the first study that combines the template matching pipeline with databases.

Typical real-time applications expect sub-second or few seconds long response times. One can note that our pipeline does not fulfill this requirement in its current form. We have to note that the main focus of our work is not on real-time query processing but on how large point cloud data sets should be stored and how template matching can be supported by a DBMS. In Section V-C, we have introduced the time costs of the different steps but, on the one hand, this evaluation has been carried out on a normal desktop computer, and thus high performance servers are likely to result in shorter execution times. On the other hand, the components of the pipeline have not necessarily been selected in terms of speed. For example, ICP provides good accuracy but its execution time is too long for real-time applications with the applied settings. If an approximate transformation is enough, then the number of iterations can be maximized in the ICP. Additional acceleration can be introduced by, e.g., using a parallelized version of ICP.

As with many other feature-based registration pipelines, we used an iterative method (Iterative Closest Point - ICP) for refining the initial transformation. Therefore, the accuracy of the resulting transformations depends on ICP and its parameters. For this reason, we think that it is not worth comparing the results from this point of view. Based on our evaluation, we can say that our method can solve the template matching problem, it is robust to noise, and the results look promising to be applicable to large point clouds.

REFERENCES

[1] A. Nüchter and K. Lingemann. (2020). *Robotic 3D Scan Repository*. Universität Osnabrück, Jacobs University Bremen, Julius-Maximilians-Universität Würzburg. Last Accessed: Mar. 03, 2020. [Online]. Available: https://kos.informatik.uni-osnabrueck.de/3Dscans

[2] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1817–1824.

[3] R. Vock, A. Dieckmann, S. Ochmann, and R. Klein, "Fast template matching and pose estimation in 3D point clouds," *Comput. Graph.*, vol. 79, pp. 36–45, Apr. 2019.

[4] P. van Oosterom, O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Gonçalves, "Massive point cloud data management: Design, implementation and execution of a point cloud benchmark," *Comput. Graph.*, vol. 49, pp. 92–125, Jun. 2015.

[5] R. Cura, J. Perret, and N. Paparoditis, "A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing," *ISPRS J. Photogramm. Remote Sens.*, vol. 127, pp. 39–56, May 2017.

[6] T. Meyer and A. Brunn, "3D point clouds in PostgreSQL/PostGIS for applications in GIS and geodesy," in *Proc. 5th Int. Conf. Geographical Inf. Syst. Theory, Appl. Manage.*, vol. 1, 2019, pp. 154–163.

[7] S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige, "Going further with point pair features," in *Computer Vision—ECCV* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2016, pp. 834–848.

[8] U. Castellani and A. Bartoli, "3D shape registration," in *3D Imaging, Analysis and Applications*. Cham, Switzerland: Springer, Jan. 2012, pp. 353–411.

[9] B. Bellekens, V. Spruyt, R. Berkvens, R. Penne, and M. Weyn, "A benchmark survey of rigid 3d point cloud registration algorithms," *Int. J. Adv. Intell. Syst.*, vol. 1, pp. 118–127, Jun. 2015.

[10] D. Varga, S. Laki, J. Szalai-Gindl, L. Dobos, P. Vaderna, and B. Formanek, "On fast point cloud matching with key points and parameter tuning," in *Pattern Recognition*, S. Palaiahnakote, G. S. di Baja, L. Wang, and W. Q. Yan, Eds. Cham, Switzerland: Springer, 2020, pp. 498–511.

[11] S. Ochmann and R. Klein, "Automatic normal orientation in point clouds of building interiors," 2019, *arXiv:1901.06487*. [Online]. Available: http://arxiv.org/abs/1901.06487

[12] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 3212–3217.

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *ACM SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 71–78, Jul. 1992.

[14] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke, "Registration with the point cloud library: A modular framework for aligning in 3-D," *IEEE Robot. Autom. Mag.*, vol. 22, no. 4, pp. 110–124, Dec. 2015.

[15] W. Prawira, E. Nasrullah, S. R. Sulistiyanti, and F. X. A. Setyawan, "The detection of 3D object using a method of a Harris corner detector and Lucas–Kanade tracker based on stereo image," in *Proc. Int. Conf. Electr. Eng. Comput. Sci. (ICECOS)*, Aug. 2017, pp. 163–166.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[17] J. Li and G. H. Lee, "USIP: Unsupervised stable interest point detection from 3D point clouds," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 361–370.

[18] Z. J. Yew and G. H. Lee, "3DFeat-Net: Weakly supervised local 3D features for point cloud registration," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 607–623.

[19] X.-F. Han, S.-J. Sun, X.-Y. Song, and G.-Q. Xiao, "3D point cloud descriptors in hand-crafted and deep learning age: State-of-the-art," 2018, *arXiv:1802.02297*. [Online]. Available: http://arxiv.org/abs/1802.02297

[20] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok, "A comprehensive performance evaluation of 3D local feature descriptors," *Int. J. Comput. Vis.*, vol. 116, no. 1, pp. 66–89, Jan. 2016.

[21] S. M. Prakhya, B. Liu, W. Lin, K. Li, and Y. Xiao, "On creating low dimensional 3D feature descriptors with PCA," in *Proc. IEEE Region 10 Conf. (TENCON)*, Nov. 2017, pp. 315–320.

[22] M. J. Hellerstein, F. J. Naughton, and A. Pfeffer, "Generalized search trees for database systems," in *Proc. 21th Int. Conf. Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann, 1995, pp. 562–573.

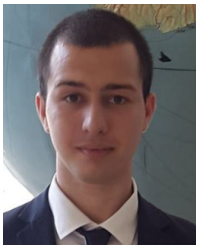
[23] A. Borodin, S. Mirvoda, I. Kulikov, and S. Porshnev, "Optimization of memory operations in generalized search trees of postgresql," in *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek, and D. Kostrzewa, Eds. Cham, Switzerland: Springer, 2017, pp. 224–232.

[24] H. Liu, P. van Oosterom, M. Meijers, and E. Verbree, "Management of large indoor point clouds: An initial exploration," *ISPRS-Int. Arch. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. 4, pp. 365–372, Sep. 2018.

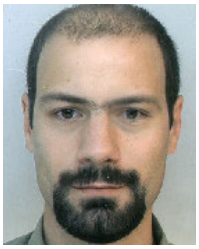
[25] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*. Menlo Park, CA, USA: AAAI Press, 1996, pp. 226–231.

[26] H. W. Kuhn, "The Hungarian method for the assignment problem," *Nav. Res. Logistics Quart.*, vol. 2, nos. 1–2, pp. 83–97, Mar. 1955.

[27] Y. Pan, B. Yang, F. Liang, and Z. Dong, "Iterative global similarity points: A robust coarse-to-fine integration solution for pairwise 3D point cloud registration," in *Proc. Int. Conf. 3D Vis. (3DV)*, Sep. 2018, pp. 180–189.
[28] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U.K.: Cambridge Univ. Press, 2nd ed., 2004.
[29] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992.
[30] F. Wang and Z. Zhao, "A survey of iterative closest point algorithm," in *Proc. Chin. Autom. Congr. (CAC)*, Oct. 2017, pp. 4395–4399.
[31] Y. Tsin and T. Kanade, "A correlation-based approach to robust point set registration," in *Computer Vision—ECCV*, T. Pajdla and J. Matas, Eds. Berlin, Germany: Springer, 2004, pp. 558–569.
[32] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proc. 3rd Int. Conf. 3-D Digit. Imag. Model.*, 2001, pp. 145–152.
[33] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 3050–3057.
[34] T. Hodan, P. Haluza, S. Obdrzalek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2017, pp. 880–888.
[35] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 1–4.



DÁNIEL VARGA received the M.Sc. degree in computer science from Eötvös Loránd University, Budapest, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Information System, Faculty of Informatics. His research interests include 3-D point cloud processing related topics: triangulation, point cloud matching, and pose estimation.

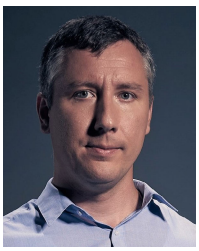


JÁNOS MÁRK SZALAI-GINDL received the M.Sc. degree in mathematics from the Budapest University of Technology and Economics and the Ph.D. degree from Eötvös Loránd University, Budapest, in 2020. He wrote his dissertation on data-intensive methods for managing scientific data. He currently works as an Assistant Professor with the Department of Information System, Faculty of Informatics, Eötvös Loránd University. His main research interests include scientific databases and data science. He has recently participated in multiple research projects in those fields.

BENCE FORMANEK received the degree in electrical engineering from the Budapest University of Technology, in 1998, and the Ph.D. degree in information science and technology in the area of compressed video transcoding algorithms from the University of Miskolc, in 2013. He was a Research Fellow with the Multimedia Group, Matáv PKI, and worked on digital television related projects at CableWorld Ltd. He currently works as a Senior Researcher with Ericsson Research, Budapest. He focuses on the problems of AR related services in cloud environment.

PÉTER VADERNA received the Ph.D. degree in physics from the Physics of Complex Systems Department, Eötvös Loránd University, Budapest, in 2008, in the area of traffic modeling in communication networks. He currently works as a Senior Researcher with Ericsson Research, Budapest. He also focuses on AI in network analytics and also emerging technologies, such as AR/VR that can potentially be involved in various business areas of telecommunication.

LÁSZLÓ DOBOS received the Ph.D. degree in physics from Eötvös Loránd University, Budapest, Hungary, in 2012. He is currently an Assistant Professor with the Department of Physics of Complex Systems, Eötvös Loránd University. His main research interests include scientific databases, point cloud databases, and astrophysics. He participates in multiple research projects in those fields.



SÁNDOR LAKI (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Eötvös Loránd University, in 2007 and 2015, respectively. He is currently an Assistant Professor with the Department of Information Systems, Eötvös Loránd University. He has authored more than 40 peer-reviewed articles and demo articles, including publications at IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *ToN*, INFOCOM, ICC, and SIGCOMM. His research interests include active and passive network measurement, traffic analytics, programmable data planes, and their application for new networking solutions.