Registration with the Point Cloud Library PCL

A Modular Framework for Aligning 3D Point Clouds

Dirk Holz, *Member, IEEE*, Alexandru E. Ichim, Federico Tombari, *Member, IEEE*
Radu B. Rusu, *Member, IEEE*, and Sven Behnke, *Member, IEEE*

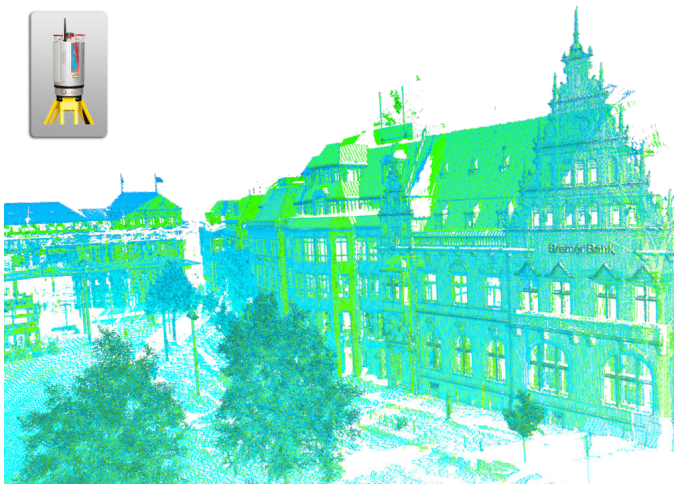
***Abstract*—Registration is an important step when processing 3D point clouds. Applications for registration range from object modeling and tracking to simultaneous localization and mapping. This article presents the open-source Point Cloud Library (PCL) and the tools therein available for the task of point cloud registration. PCL incorporates methods for the initial alignment of point clouds using a variety of local shape feature descriptors as well as for refining initial alignments using different variants of the well-known Iterative Closest Point (ICP) algorithm. The article provides an overview on registration algorithms, usage examples of their PCL implementations, and tips for their application. Since the choice and parameterization of the right algorithm for a particular type of data is one of the biggest problems in 3D point cloud registration, we present three complete examples of data (and applications) and the respective registration pipeline in PCL. These examples include dense RGB-D point clouds acquired by consumer color and depth cameras, high-resolution laser scans from commercial 3D scanners, and low-resolution sparse point clouds captured by a custom lightweight 3D scanner on a micro aerial vehicle.**

***Index Terms*—PCL, 3D Registration, Point Clouds, ICP**

I. Introduction

3D *Registration* is the problem of consistently aligning two or more point clouds, i.e., sets of three-dimensional points. Often the point clouds are acquired by 3D sensors from different viewpoints . The registration finds the relative pose (position and orientation) between views in a global coordinate frame, such that the overlapping areas between the point clouds match as well as possible. See Figure 1 for two examples of registration. The overall objective of registration is to align individual point clouds and fuse them to a single point cloud, so that subsequent processing steps like segmentation and object reconstruction can be applied.

Point cloud registration is a recurring task for a number of applications related to computer vision, computer graphics, robotic perception, photogrammetry, cultural heritage modeling, digital archaeology, and architecture. This wide applicability motivated intense research in the past years, aimed on the one hand at obtaining accurate alignments for challenging 3D data such as isotropic or non-distinctive objects, and on the other hand at making the registration process as automatic as possible.



(a) Registered pair of high-resolution 3D laser scans (green and blue) in a city[1]

(b) Registered RGB-D image sequence of toy cars in a table setting

Fig. 1. Examples of 3D registration in different scales and applications.

Publicly available software libraries and data sets are a way to promote such research and to make results comparable. This article presents the open-source Point Cloud Library (PCL) and the tools therein available for the task of point cloud registration. PCL is a stand-alone, large scale, open project for 3D point cloud processing. It is released under the BSD license and contains numerous state-of-the-art algorithms for various applications including filtering, feature estimation, surface reconstruction, model fitting, segmentation, and visualization, as well as higher-level tools and applications for performing mapping and object recognition.

D. Holz and S. Behnke are with the University of Bonn, Bonn, Germany.
A. E. Ichim is with École Polytechnique Fédérale de Lausanne, Switzerland.
F. Tombari is with University of Bologna, Bologna, Italy.
R. B. Rusu is with Open Perception, Inc., CA, USA.
Contact: dirk.holz@ieee.org

This paper is accompanied by a webpage containing the complete datasets and working applications for all the examples provided in the paper, as well as other samples: http://pointclouds.org/media/registration-paper.html.

[1]Data recorded by Dorit Borrmann and Jan Elseberg, Robotic 3D Scan Repository: http://kos.informatik.uni-osnabrueck.de/3Dscans.

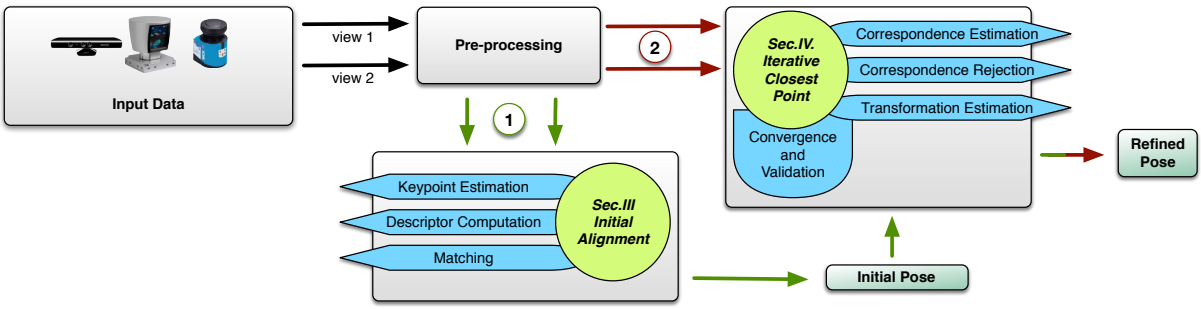


Fig. 2. Overview of the process of registering a pair of point clouds. After the two views are in memory, a pre-processing sequence is applied in order to improve the signal to noise ratio or to recover information about the surface (normals) that is lost because of the sampling involved in the scanning procedure. There are two available paths: the first one consists of computing keypoints and their descriptors and then estimating a relative transformation between the two clouds. This can be used as a good initial alignment for the dense registration via Iterative Closest Point, represented by the second path.

In this article, we focus on the pairwise registration of 3D point clouds, i.e., aligning two point clouds by estimating the transformation between the two view poses under which the point clouds have been acquired. Techniques and tools for tackling sequences of point clouds in a globally coherent fashion (i. e., the so called *multi-view registration*) will be covered in future publications.

Pairwise registration is usually carried out by means of one of the several variants of the Iterative Closest Point (ICP) algorithm. Due to the non-convexity of the optimization, ICP-based approaches require initialization with a rough initial transformation in order to increase the chance of ending up with a successful alignment. Good initialization also speeds up their convergence. This paper will cover all stages required for point cloud alignment—from coarse alignment based on descriptor matching to the ICP-based refinement—illustrating the tools currently available in PCL to carry out each task. Moreover, we present three pipelines implemented in PCL together with typical results for different types of input data, ranging from small-scale object modeling to outdoor 3D mapping.

The article is structured as follows: In Section II, we first introduce typical registration pipelines and a common scheme for registration approaches. Section III covers ICP-based registration for input data that is already roughly aligned and Section IV discusses feature-based registration in case no such initial alignment is available. Finally, we present three complete examples of input data and the respective registration pipelines implemented using PCL in Section V.

II. General/Basic Registration Pipelines

A. Problem Formulation

A point cloud is a data structure P used to represent a collection of multi-dimensional points $\boldsymbol{p} \in \mathbb{R}^n$. In a 3D point cloud, the elements usually represent the X, Y, and Z geometric coordinates of an underlying sampled surface. When more information is available, e.g., color information or information about local surface normal $\boldsymbol{n}$ or curvature κ, the points $\boldsymbol{p} \in P$ are represented by a longer vector.

Given a *source* point cloud P with points $\boldsymbol{p} \in P$, and a *target* point cloud Q with points $\boldsymbol{q} \in Q$, the problem of registration relies on finding *correspondences* between P and Q, and estimating a transformation T that, when applied to P, aligns all pairs of corresponding points $(\boldsymbol{p}_i \in P, \boldsymbol{q}_j \in Q)$. One fundamental problem of registration is that these correspondences are usually not known and need to be determined by the registration algorithm. Given correct correspondences, there are different ways of computing the optimal transformation w.r.t. the used error metric, as detailed in the following.

Listing 1 Example Code Listing and Terminology

```
// specifying types of input and output data
Algorithm<PointT, NormalT, FeatureT> alg;

// providing input data
alg.setInputCloud (input_cloud);
alg.setInputNormals (input_normals);

// computing output
PointCloud<FeatureT> output_features;
alg.compute(&output_features);
```

To support processing various types of point clouds, in PCL, all components are templated. The remainder of the article will use code snippets in the style of Listing 1.

B. Typical Registration Pipelines

Referring to Figure 2, and following the modularization of Rusinkiewicz and Levoy [1], the registration of two point clouds can be split into the following steps:

1) *Selection:* The sampling of the input point clouds.
2) *Matching:* Estimating the correspondences between the points in the subsampled point clouds.
3) *Rejection:* Filtering the correspondences to reduce the number of outliers.
4) *Alignment:* Assigning an error metric, and minimizing it to find the optimal transformation.

Each of these four stages will be now detailed, with specific reference to the tools available in PCL.

C. Matching Closest Points vs. Feature-based Registration

Two major classes of registration algorithms can be distinguished (see the two paths in Figure 2):

1) *Feature-based registration algorithms* (path 1) for computing initial alignments, and
2) *Iterative registration algorithms* (path 2) following the principle of the ICP algorithm to iteratively register point clouds (that are already roughly aligned).

For feature-based registration, geometric feature descriptors are computed and matched in some high-dimensional space. The more descriptive, unique, and persistent these descriptors are, the higher is the chance that all found matches are pairs of points which truly correspond to one another.

In the ICP algorithm by Besl and McKay [2] no feature descriptors are computed, but instead closest points in Cartesian space are considered to correspond to one another. A transformation is estimated that minimizes the Euclidean distances between found pairs of closest points in the least squares sense. The process of determining corresponding points in the two data sets and computing the transformation that aligns them is iteratively repeated. The source point set is expected to converge towards the target set as the correspondences increasingly become better and better. Simultaneously, Chen and Medioni [3] formulated a similar algorithm, but instead of minimizing the squared Euclidean distances between corresponding points, applied a point-to-plane error metric. In the following two sections, we discuss both approaches.

III. Iterative Registration of Closest Points

In contrast to feature-based registration, iterative registration algorithms do not match salient feature descriptors in order to find correspondences between source and target point clouds, but instead 1) search for closest points (*matching step*) and 2) align the found point pairs (*alignment step*). These two steps are repeated until convergence or until reaching another termination criterion, thereby iteratively refining the alignment of the source to the target point cloud. Under various assumptions such as perfect overlap, the alignment converges to the global minimum (w.r.t. the error metric used) for optimal alignment. The main drawback of iterative registration is that the algorithms may get caught in local minima if the assumptions do not hold, e.g., if the clouds only partially overlap and/or if the initial alignment is off. In this case, false correspondences can negatively affect the registration result. However, several methods exist to sort out false correspondences (*rejection step*) and improve convergence. Furthermore, if the clouds are already roughly aligned, iterative registration provides efficient and robust means to refine that initial guess and optimally align the point clouds. In order to speed up registration, another common extension to the original ICP algorithm [2] is to register only subsets of the input point clouds sampled in an initial *selection step*.

A. Selection — Sampling Representative Subsets

Depending on the application and sensor, point clouds can become quite large. Consequently, registering large point clouds is considerably more computationally expensive than registering clouds of smaller cardinality. However, data is often redundant or unnecessarily detailed for the task of registration. Hence, registering only subsets of the original point clouds can yield sufficient results while saving computation time. In principle, two methods of data reduction can be distinguished: automatically extracting a small set of unique and repeatable keypoints (see Section IV-A) and sampling of the original data with respect to a desired target distribution. Whereas the former is intended for feature-based initial alignment (Section IV), the latter can be used to efficiently reduce the amount of data for iterative registration algorithms. PCL implements several such sampling methods, most notably, sampling in index space (simply taking every n-th point), uniform subsampling in the input 3D space (to better capture the sensed environmental structures), and sampling in the space of local surface normals (in order to sample points over all surface orientations). An example of using uniform sampling in PCL is given in Listing 2.

Listing 2 Point Cloud Sampling (Uniform)

```
PointCloud<int> indices;
UniformSampling<PointT> uniform_sampling;
uniform_sampling.setInputCloud (cloud);
uniform_sampling.setRadiusSearch (0.05f);
uniform_sampling.compute (indices);
```

Listing 3 Correspondence Estimation

```
#include <pcl/registration/correspondence_estimation.h>
(...)
CorrespondencesPtr corresps(new Correspondences);
CorrespondenceEstimation<PointT, PointT> est;
est.setInputSource (source_cloud);
est.setInputTarget (target_cloud);
est.determineCorrespondences (*corresps, max_dist);
```

B. Matching — Closest Points Correspondence Estimation

Correspondence estimation is the process of pairing points $\boldsymbol{p}_i$ from the source point cloud P to their closest neighbors $\boldsymbol{q}_j$ in the target cloud Q. This is a greedy approximation of finding the ideal correspondences $(\boldsymbol{p}_i, \boldsymbol{q}_j)$ between the two clouds. A naïve way of searching for the nearest neighbor is to perform an exhaustive search through all the target points for the nearest neighbor of each source point. As this is prohibitively expensive for applications using millions of points, various data structures for rapid searches have been proposed, such as octrees and kd-trees. These data structures have logarithmic search times (as opposed to the linear naïve variant), but take longer to initialize, e.g., $O(N \log N)$ for kd-trees. For this purpose, PCL depends on FLANN [4], an open-source library for fast (approximate) nearest neighbor searches. It has been empirically proven to be one of the best performing solutions [5].

In order to compute a set of correspondences in our framework, we can use the code in Listing 3. Given `source_cloud` and `target_cloud` as input, `determineCorrespondences` returns the set of found correspondence pairs. Each pair consists of the index of the query point in the source cloud and the index of the found match in the target point cloud. The optional second argument `max_dist` can be used to specify a maximum distance between corresponding points such that pairs with a larger point-to-point distance are filtered out.

In the case of input data coming from sensors that comply with the pinhole camera model, the correspondence estimation

Listing 4 Correspondence Estimation using Projections

```
#include <pcl/registration/
correspondence_estimation_organized_projection.h>
(...)
CorrespondenceEstimationOrganizedProjection
  <PointXYZ, PointXYZ> est;
est.setInputSource (cloud_source);
est.setInputTarget (cloud_target);
est.setFocalLengths (f_x, f_y);
est.setDepthThreshold (epsilon);
CorrespondencesPtr corresps (new Correspondences ());
est.determineCorrespondences (*corresps);
```

procedure can be significantly sped up at the loss of some precision. Such sensors include popular RGB-D cameras like the Microsoft Kinect, and collect information from the environment in the form of depth and color images. In PCL, we refer to point clouds with an image-like structure as being *organized*. Instead of using complex data structures such as kd-trees for nearest neighbor searches in 3D space, we can use the projective nature of the depth images to obtain a reasonable approximation. Each point in the cloud corresponds to a pixel in the depth image, allowing projections from source points in world coordinates to the camera plane of the target frame by using the intrinsic and extrinsic camera parameters:

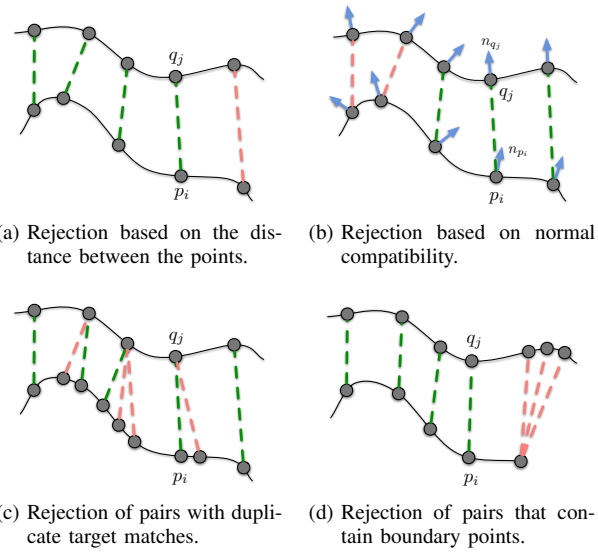
$$\begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R|t \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} d \cdot u \\ d \cdot v \\ d \end{bmatrix}, \tag{1}$$

where the first matrix is the intrinsic calibration matrix of the source camera with (f_x, f_y) being the focal length (divided by image width and height respectively) and (c_x, c_y) the camera projection center, the matrix $[R|t]$ is the transformation of the source camera, d is the depth of the projected point $\boldsymbol{p} = (p_x \ p_y \ p_z)^T$ in the target camera, and u, v are the coordinates of $\boldsymbol{p}$ in the target camera plane. After projection, the query point is assigned to its closest point in image space (if u and v lie within image boundaries) or considered part of the non-overlapping volume between source and target cloud.

This approach is fast, but imprecise for point clouds with large depth discontinuities or for frames that are far away from each other. That is why this method is recommended to be used only after the two point clouds have been brought close together, making it good for aligning consecutive point clouds in a stream recorded at high frame rate. A code snippet for projection-based correspondence estimation is presented in Listing 4.

C. Rejecting and Filtering Correspondences

Since invalid correspondences can negatively affect the registration results, most registration pipelines feature a rejection step. It consists of filtering the point pairs matched in the previous stage in order to facilitate the transformation estimation algorithm towards convergence to the global minimum. This step can take advantage of auxiliary information available from the input point clouds, such as local surface normals or statistics about the correspondences. Referring to Figure 3, the following correspondence rejection methods are commonly deployed and available in PCL:



(a) Rejection based on the distance between the points.

(b) Rejection based on normal compatibility.

(c) Rejection of pairs with duplicate target matches.

(d) Rejection of pairs that contain boundary points.

Fig. 3. Correspondence rejection. Good correspondence pairs (green) are kept while outliers (red) are sorted out to improve convergence.

1) Correspondence rejection based on distance: This method filters out point pairs with a distance larger than a given threshold (see Figure 3a). It is used in [1] and was already mentioned in the original formulation of the ICP algorithm [2] (→ `CorrespondenceRejectorDistance`).

2) Rejection based on median distance: Unlike the previous rejector, this one does not use a fixed threshold, but computes it as the median of all point-to-point distances in the input set of correspondences. Hence, it considers the distribution of the distances between the points and adapts to it, becoming smaller as the two point clouds get closer during the ICP iterations. Compared to an adaptive threshold based on the mean value, the median is often more effective in reducing the influence of outliers. (→ `CorrespondenceRejectorMedianDistance`).

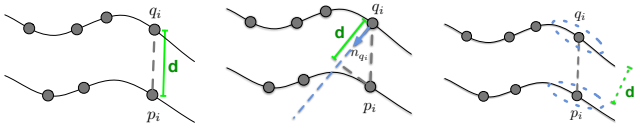
3) Rejecting pairs with duplicate target matches: Usually, each sampled point in the source cloud is assigned to a correspondence in the target cloud. Hence, it might happen that a point in the target cloud is assigned multiple corresponding source points (see Figure 3c). This rejector only keeps a single such pair $(\boldsymbol{p}_{i_{min}}, \boldsymbol{q}_j)$, the one with the minimum distance out of all the pairs $\{(\boldsymbol{p}_i, \boldsymbol{q}_j)\}$ (→ `CorrespondenceRejectorOneToOne`).

4) RANSAC-based rejection: This method applies *Random Sample Consensus (RANSAC)* [6] to estimate a transformation for subsets of the given set of correspondences and eliminates the outlier correspondences based on the Euclidean distance between the points after the computed transformation is applied to the source point cloud. It is very effective in keeping the ICP algorithm from converging into local minima, as it always produces slightly different correspondences and is good at filtering outliers. In addition, it provides good initial parameters for the transformation estimation with all inlier correspondences that follows (→ `CorrespondenceRejectorSampleConsensus`).

For the registration of point clouds coming from projective sensors, there exist correspondence rejectors that exploit the

Listing 5 Correspondence Rejection (based on distance)

```
#include <pcl/registration/
correspondence_rejection_distance.h>
(...)
CorrespondenceRejectorDistance rejector;
rejector.setInputSource<PointT> (cloud_src);
rejector.setInputTarget<PointT> (cloud_tgt);
rejector.setInputCorrespondences (corresps_in);
rejector.setMaximumDistance (max_dist);
rejector.getCorrespondences (corresps_filtered);
```



(a) Point to point error (b) Point to plane error (c) Generalized-ICP

Fig. 4. Error metrics and transformation estimators.

image-like structure of the input data. Most notably, a special 2D variant (→ `CorrespondenceRejectorSampleConsensus2D`) of the RANSAC-based correspondence rejector discards pairs based on their pixel distance in the image plane after the source point is transformed and projected into the target camera plane.

5) Rejection based on normal compatibility: This filter uses the normal information about the points, and rejects those pairs that have inconsistent normals, i.e., the angle between their normals is larger than a given threshold. It can reject erroneous pairs that seem correct when judged only by the distance between the points, such as the case depicted in Figure 3b. (→ `CorrespondenceRejectorSurfaceNormal`).

6) Rejection based on surface boundaries: Furthermore, when two point clouds captured by a projective sensor represent surfaces that have partial overlap, accepting correspondences containing surface boundary points can introduce errors (Figure 3d). In order to detect such points, we can exploit the organized nature of the depth maps and eliminate the correspondences that contain points on depth discontinuities by moving a window across the depth image and checking if there are enough points in the window that are within a threshold depth from the center point (→ `CorrespondenceRejectorBoundaryPoints`).

7) Rejector pipelines: Most often, not only a single rejector is applied, but instead several correspondence rejectors are queued in order to implement a filtering pipeline. For example, Diebel et al. [7] apply a combination of different solutions for rejecting point pairs in their active stereo point clouds, emphasizing the importance of rejecting points on mesh boundaries (see the RGB-D example in Section V-C). In addition, they reject pairs based on their normal compatibility and based on their point-to-point distance using a distance threshold based on the median distance. In PCL, multiple rejectors can be easily concatenated by using the output correspondences of the previous rejector as input to the next one.

We provide a usage example for PCL correspondence rejection in Listing 5. Given the input point clouds and a set of estimated correspondence pairs, `getCorrespondences` runs the rejection and returns the vector of filtered correspondence pairs. Moreover, all correspondence rejection methods in PCL feature `getRejectedQueryIndices` to retrieve the indices of query points of rejected correspondence pairs. This information is useful, for example, to determine non-overlapping parts between the source and target point cloud.

D. Alignment — Error Metrics and Transformation Estimation

Over the years, there have been numerous mathematical approaches for solving for the rigid transformation $\boldsymbol{T}$ that minimizes the error of the point pairs. $\boldsymbol{T}$ is composed of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$. Note that, in the following when referring to a transform $\boldsymbol{T}$ and a point $\boldsymbol{p}$, homogeneous coordinates will be used.

There are two main error metrics to be minimized that have been considered in literature: *point-to-point* (Eq. 2) and *point-to-plane* (Eq. 3), where $(\boldsymbol{p}_k, \boldsymbol{q}_k)$ is the k-th of the N pair correspondences from the source cloud to the target cloud.

$$E_{\text{point-to-point}}(\boldsymbol{T}) = \sum_{k=1}^{N} w_k ||\boldsymbol{T}\ \boldsymbol{p}_k - \boldsymbol{q}_k||^2, \text{ and} \quad (2)$$

$$E_{\text{point-to-plane}}(\boldsymbol{T}) = \sum_{k=1}^{N} w_k \left((\boldsymbol{T}\ \boldsymbol{p}_k - \boldsymbol{q}_k) \cdot \boldsymbol{n}_{\boldsymbol{q}_k}\right)^2. \quad (3)$$

The optional w_k can be used for weighting the pairs in order to give them more or less importance in the least squares formulation ($w_k = 1$ if no weighting is applied):

1) Standard point-to-point error metric: The standard error metric used in the ICP algorithm is the point-to-point error metric (Eq. 2). It was first mentioned by Arun [8]; researchers proposed various ways of minimizing it, followed by the introduction of the ICP Algorithm [2]. Eggert et al. [9] evaluated each of these methods in terms of numerical stability and accuracy reaching the conclusion that they are close performers.

PCL provides an implementation using a closed-form using singular value decomposition (SVD) that was first proposed by Horn [10] (→ `TransformationEstimationSVD`).

2) Point-to-plane error metric: Chen and Medioni [3] introduced the point-to-plane metric (Eq. 3) and proved it to be more stable and converge faster than the previous approaches. It uses the distance between the source point $\boldsymbol{p}_k$ and the plane described by the target point $\boldsymbol{q}_k$ and its local surface normal $\boldsymbol{n}_{\boldsymbol{q}_k}$. Unlike the point-to-point metric, it does not have a closed-form solution, so the minimization is done with non-linear solvers (such as Levenberg-Marquadt, as proposed by [11]), or by linearizing it [12] (under the assumption of small rotation angles, i. e., $\sin\theta \sim \theta$ and $\cos\theta \sim 1$).

Depending on the underlying surface and the distribution of points, using the point-to-plane error metric can be considerably more robust. A standard procedure for minimizing it relies on the Levenberg Marquardt non-linear optimizer [11] (→ `TransformationEstimationPointToPlane`).

3) Linear least squares point-to-plane: PCL also features an alternative method that accumulates the point-to-plane constraints of all the correspondences in a matrix $\boldsymbol{A}$ and estimates the rotation and translation in the least squares sense by solving a linear system of the form $\boldsymbol{A}^T\ \boldsymbol{A}\ \boldsymbol{v} = \boldsymbol{A}^T \boldsymbol{b}$,

Listing 6 Transformation estimation (weighted point-to-plane)

```
#include <pcl/registration/
transformation_estimation_point_to_plane_weighted.h>
(...)
TransformationEstimationPointToPlaneWeighted
  <PointXYZ, PointXYZ, double> te;
te.setWeights (correspondence_weights);
te.estimateRigidTransformation (*cloud_src, *cloud_tgt,
  *corresps_filtered, transform);
```

with $\boldsymbol{v}$ being a six-dimensional parameterized representation of the minimizing transformation, as suggested by [12] (→ `TransformationEstimationPointToPlaneLLS`).

4) Weighted point-to-plane error metric: Assigning a different weight to each correspondence can improve convergence. The weighting of the point pairs can be seen as a soft correspondence rejection, adjusting the influence of noisy corresponding points in the minimization process. The weighting can be a function of the point-to-point or point-to-plane distance between the points, a function of the angle between the normals corresponding to the points, or a function of the noise model of the sensor that has been used. Numerous publications [1, 7, 13, 14] consider the weighting of the point pairs as a beneficial step for the robustness and convergence rate of the transformation estimation algorithm.

Referring to Figure 4, PCL offers a multitude of techniques for estimating the transformation between two point clouds given a set of correspondences. As an example for transformation estimators, a code snippet showing how to use the weighted point-to-plane error metric (→ `TransformationEstimationPointToPlaneWeighted`) is given in Listing 6.

5) Weighted linear least squares point-to-plane: Similarly to the previous one, the linear least squares point-to-plane variant can be used with non-constant weights (→ `TransformationEstimationPointToPlaneLLSWeighted`).

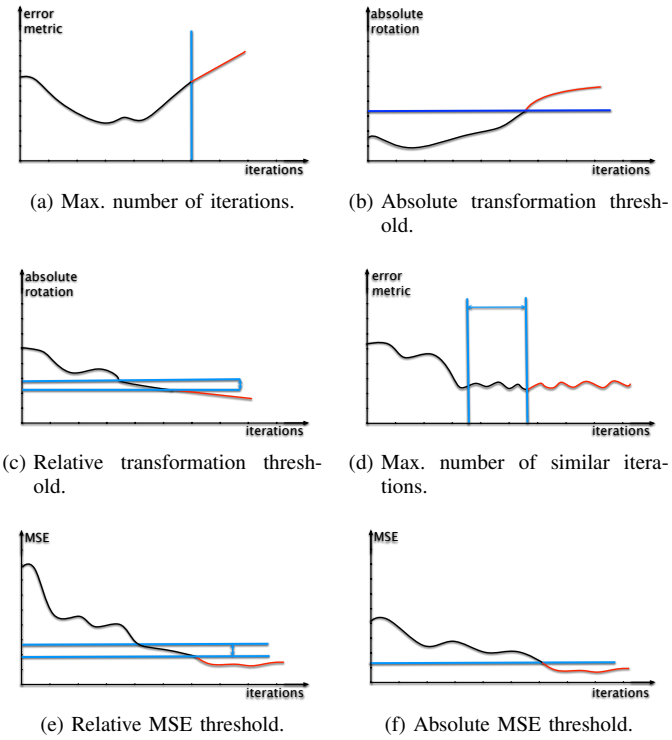
6) Generalized error metric and Generalized-ICP: Segal et al. [15] proposed an error metric that generalizes over point-to-point and point-to-plane. It uses covariances of the local point neighborhoods ($\boldsymbol{\Sigma}_k^P$ and $\boldsymbol{\Sigma}_k^Q$) in order to align the underlying surfaces rather than the points themselves:

$$E_{\text{Generalized-ICP}}\left(\boldsymbol{T}\right) = \sum_{k=1}^{N} \boldsymbol{d}_k^{(\boldsymbol{T})^T} \left(\boldsymbol{\Sigma}_k^Q + \boldsymbol{T}\boldsymbol{\Sigma}_k^P \boldsymbol{T}^T\right)^{-1} \boldsymbol{d}_k^{(\boldsymbol{T})} \tag{4}$$

where $\boldsymbol{d}_k^{(\boldsymbol{T})} = \boldsymbol{q}_k - \boldsymbol{T}\boldsymbol{p}_k$ are the point-to-point distances of the correspondences $(\boldsymbol{p}_k, \boldsymbol{q}_k)$, $\boldsymbol{T}$ is the transformation matrix, and $\boldsymbol{T}^T$ its transpose. In PCL, Generalized-ICP is implemented as a complete registration component that includes computation of the covariances $\boldsymbol{\Sigma}_k^P$ and $\boldsymbol{\Sigma}_k^Q$ for all points in source cloud P and target cloud Q (→ `GeneralizedIterativeClosestPoint`).

E. Termination Criteria

For iterative registration algorithms, termination criteria for the transformation refinement process must be defined. In addition to simply setting a maximum number of iterations, many other criteria can be specified. Among others, the following termination criteria are available in PCL. Using a common interface (→ `DefaultConvergenceCriteria`), they can be efficiently combined and configured.



(a) Max. number of iterations. (b) Absolute transformation threshold. (c) Relative transformation threshold. (d) Max. number of similar iterations. (e) Relative MSE threshold. (f) Absolute MSE threshold.

Fig. 5. ICP termination criteria supported in PCL.

1) Maximum number of iterations: Exceeding the number of iterations means that the optimizer diverged. This threshold has to be tuned depending on the complexity of the registration problem; expect that registering a pair of scans that are far away from each other will require more iterations, than two scans that have a good initial alignment (Figure 5a).

2) Absolute transformation threshold: The iterations are stopped when the currently estimated transformation is far away from the initial transformation. This is an early termination criteria for registration procedures that diverge. The intuition behind it is that the two clouds to be aligned are expected to be within a certain range of distances from each other, and so transformations that are outside that range need to be rejected, e.g., two consecutive handheld-Kinect scans recorded at 30Hz can not be more than 10 cm and 20 degrees apart from each other (Figure 5b).

3) Relative transformation threshold: It specifies the minimum transformation difference from one iteration to the next that is considered small enough for the optimizer to have converged (Figure 5c).

4) Maximum number of similar iterations: The previous stopping criteria has the downside that a minimizer might temporarily seem to have converged, but it is actually oscillating in a local minimum and has a chance of escaping from it and converging into the global minimum or a better local minimum. For this reason, we allow the optimizer to spend a certain number of iterations around a minimum point before considering it converged (Figure 5d).

5) Relative mean square error: This criterion is similar to the relative transformation threshold, using the mean square

Listing 7 Combination of Termination Criteria

```
#include <pcl/registration/default_convergence_criteria.h>
(...)
DefaultConvergenceCriteria<double> conv_crit (iteration,
  transform, *correspondences);
conv_crit.setMaximumIterations (30);
conv_crit.setMaximumIterationsSimilarTransforms (3);
conv_crit.setTranslationThreshold (5e-3);
conv_crit.setRotationThreshold (cos (0.5 * M_PI / 180.0));
conv_crit.setRelativeMSE (0.01);
do
{ (... ICP iterations ...)
} while (!conv_crit.hasConverged ())
ConvergenceState why_conv =
  conv_crit.getConvergenceState ();
```

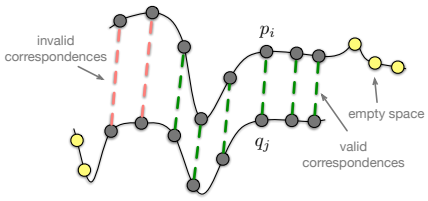


Fig. 6. Transformation validation: if two surfaces match locally, we need to make sure that their empty space also matches and that there is no additional depth information in the overlap region of any of the clouds that does not have correspondences in the other cloud.

error metric instead of the rotation/translation increment (Figure 5e).

6) Absolute mean square error: It stops the iterations when the error between the two aligned clouds is below a certain value (Figure 5f).

All of these techniques represent ways of balancing between the quality and the runtime of the registration procedure. They prove to be essential for tuning algorithms for real-time performance.

F. Transformation Validation (Optional)

Due to the presence of numerous local minima in the ICP error function, a series of checks can be performed at the end of the registration process in order to confirm a successful convergence. We suggest checking for the percentage of the overlapping surface of the two point clouds (→ `TransformationValidationEuclidean`). In the case of sensors that obey the pinhole camera model (RGB-D), this can be done efficiently in image space by rendering one point cloud on top of the other with the computed poses. Similarly, one can check if the overlapping region has enough inliers (see Figure III-F). Another indicator for the success of the registration is the condition number of the overlapping region [16], which tells if the common surface of the point clouds is stable enough so it does not allow slippage.

IV. Coarse Alignment via Descriptor Matching

As discussed, ICP-based algorithms, due to their greedy nature, require a reliable initial alignment to avoid converging to bad local minima. Typical solutions to this problem are represented by running ICP multiple times, each time with a different, random initialization—prohibitively slow for most applications—or by computing an initial transformation from a hand-chosen set of correspondences inserted by a human user (semi-automatic approach). A further approach which is fast and fully automatic is determining point-to-point correspondences between 3D keypoints extracted from both clouds via matching of the associated keypoint descriptors. This set of correspondences is then successively pruned of mismatches by means of an outlier rejection scheme, so that it can be deployed to obtain the transformation aligning one cloud to the other. The use of a coarse alignment increases the chance of successful alignment, as well as making ICP converge faster. Also, it is a fully automatic approach which does not require any user intervention. PCL includes several state-of-the-art tools for carrying out initial alignment via descriptor matching, which are briefly outlined in the following.

A. Estimating Keypoints

Coarse pairwise alignment can be carried out without an explicit keypoint selection stage, i. e., by random selection of points or by uniform sampling of the cloud (see Section III-A). Nevertheless, several choices are currently available in literature and in PCL that explicitly aim at extracting salient and repeatable keypoints given an input cloud. The advantage of relying on specific 3D detection schemes is that the repeatability and robustness of the computed 3D correspondences with respect to nuisances affecting the data is generally improved, thus enhancing the robustness of 3D registration schemes.

Due to the topic of 3D keypoint detection being fairly new, a common approach consists in porting the key criteria of successful image interest point detectors from the 2D domain (i. e., images) to the 3D domain. This is, e.g., the case of the Harris corner detector [17], a popular approach based on Taylor expansion of the directional intensity variation computed on an image point. In the 3D extension of this criterion (→ `Harris3D`), the directional intensity variation is substituted by computing the angle between the normal of the central point and its neighbors: a point yielding big angle values along several directions will typically denote a corner in 3D. Two variants of such a detector are also available, one detecting corners in the intensity domain (→ `Harris2D`), the other jointly exploiting the intensity as well as the 3D domain (→ `Harris6D`). As for most 3D keypoint detectors [18], a Non-Maxima Suppression (NMS) stage is applied to retrieve only local maxima of the saliency measure evaluated at each point by the detector.

Other 3D keypoint detector approaches were specifically designed for the 3D or 2.5D domain. This is the case of Intrinsic Shape Signature (ISS) [19], which is based on the eigenvalue decomposition (EVD) of the 3×3 scatter matrix computed between each point and all neighbors within a spherical support. Non-distinctive points are rejected by thresholding the ratios of the three eigenvalues. The smallest eigenvalue is used as the keypoint saliency, which is subject to the NMS stage. Thus, keypoints retained by this method exhibit a distinct variance of the point coordinates along one direction with respect to the other two. As an example, Listing 8 shows how to compute keypoints using ISS.

Another recent interest point detector is NARF [20], a method specifically designed to extract salient keypoints on

Listing 8 Keypoint Estimation (Intrinsic Shape Signatures)

```
ISSKeypoint3D<PointT, PointT> iss_detector;
iss_detector.setSalientRadius (salient_radius);
iss_detector.setNonMaxRadius (non_max_radius);
iss_detector.setInputCloud (input_cloud);

PontCloud<PointT>::Ptr keypoints (PontCloud<PointT>());
iss_detector.compute (*keypoints);
```

Listing 9 Feature Estimation (FPFH)

```
FPFHEstimation<PointT, NormalT, FPFHSignature33> fpfh;
fpfh.setSearchSurface (cloud);
fpfh.setInputNormals (normals);
fpfh.setRadiusSearch (radius);
fpfh.setInputCloud (keypoints);

PointCloud<FPFHSignature33> features;
fpfh.compute (&features);
```

range images. It focuses on detecting points along object depth borders. Specifically, a subset of border points is selected such that they exhibit substantially different dominant directions of the surface in the local neighborhood, so that they can be stable and repeatable with respect to viewpoint changes (→ `NarfKeypoint`).

For more details on other relevant proposals in the field, as well as for a recent performance evaluation, we refer the reader to [18].

B. Describing Keypoints — Feature Descriptors

For each detected keypoint a descriptor is computed for being able to determine correspondences between the two point clouds. A *local* descriptor is a compact representation of a point's local neighborhood. In contrast to *global* descriptors describing a complete object or point cloud, local descriptors try to resemble shape and appearance only in a local neighborhood around a point and thus are very suitable for representing it in terms of matching. Among the several approaches currently available in literature, we report here a brief outline of the main solutions available in PCL for the registration of 3D clouds. Apart from the outlined descriptors, PCL includes several other methods: we refer the interested reader to [21], a survey focused on the 3D descriptors available in PCL (including related code snippets), and to [22], focusing on a performance evaluation of PCL 3D descriptors.

The Spin Image descriptor [23] rotates a plane section around the normal of the keypoint. The plane section accumulates over a 2D histogram the number of points of the cloud intersected while doing a complete spin around the normal (see Fig. 7a). One advantage of this approach is that only a repeatable normal - rather than a full Local Reference Frame (LRF) - is required to compute the descriptor.

The FPFH descriptor [24] stores the relative orientation of normals and distances between point pairs falling within the spherical neighborhood of a keypoint. Point pairs are formed by the keypoint and its nearest neighbors, as well as by the nearest neighbors of each keypoints' neighbor. For each pair $(\boldsymbol{p}_i, \boldsymbol{p}_j)$, a repeatable LRF is constructed by means of a Darboux frame coordinate system [24]. Then, three angles α, ϕ and θ are computed out of different comparisons between the unit vectors composing the LRF and the normals of $(\boldsymbol{p}_i, \boldsymbol{p}_j)$, as depicted in Fig. 7b. These three angles are accumulated over three separate histograms for all point pairs, which are in turn juxtaposed so to yield the final descriptor. Listing 9 shows how to compute FPFH feature descriptors for a given set of keypoints.

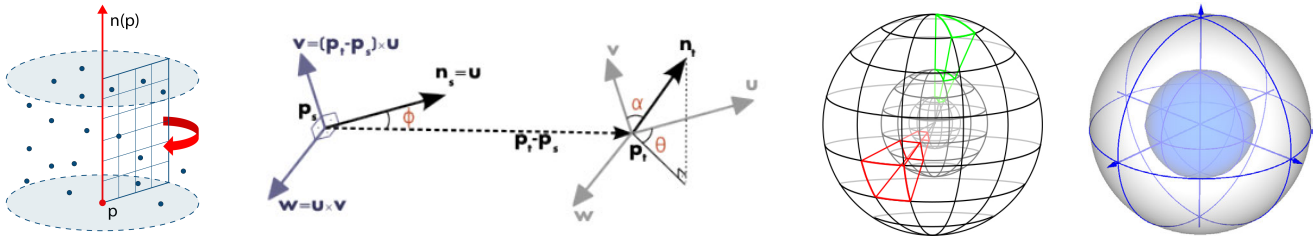
3D Shape Context (3DSC) [25] relies on a spherical grid centered on the keypoint and quantized along its three domains radius, azimuth, and elevation (see Figure 7c for a visual example). This quantization defines a 3D histogram, where each bin is represented by a volume of the grid, and the accumulated value is the number of points falling therein and weighted by the volume size and the local point density. The final descriptor orderly stores all bins of the histogram, by aligning the north pole of the grid with the normal of the central point, and by shifting the grid over the normal plane as many times as the number of azimuth subdivisions. Multiple grid orientations in turn yield multiple descriptions for the same point, so to obtain rotation invariance (→ `ShapeContext3DEstimation`).

The USC [26] descriptor has been proposed as an extension of 3DSC, where an LRF is associated to each point being described. The LRF provides two repeatable principal directions over the normal plane which uniquely orient the 3D grid associated with each descriptor, thus yielding a single descriptor at each keypoint rather than multiple ones. This reduces memory footprint and matching ambiguities during the successive correspondence estimation stage (→ `UniqueShapeContext`).

The SHOT descriptor [27] relies on a 3D grid centered on the keypoint and oriented according to a repeatable LRF (see Fig. 7d for a visual example). Each unit vector making up this LRF coincides with a principal direction obtained via eigenvalue decomposition of the scatter matrix of the points falling within the grid, and its sign is disambiguated according to a specific procedure relying on the relative density of the cloud [27]. For each grid sector, a histogram is computed by accumulating the angles between the normal of the keypoint and the normal of each point falling within that sector. The final descriptor is formed by orderly juxtaposing all computed histograms (→ `SHOTEstimation`).

C. Correspondence Estimation and Filtering

The simplest approach to establish point-to-point correspondences between the keypoints extracted from the two clouds is to associate to each descriptor computed on one cloud its nearest neighbor in the descriptor space among the descriptor set computed on the other cloud. Obviously, several correspondences are going to be erroneous due to the presence of non-overlapping areas between the two clouds, as well as due to spurious matches caused by noise, geometry deformations and non-distinctive descriptors. Correspondence estimation thus needs to match only a subset of keypoints and to reject found correspondences respectively (see Section III-C). One of the most common approaches is to use RANSAC, where the generative model is the 6-Degree-of-Freedom (6DOF) transformation between the two clouds built over three randomly chosen correspondences (Section III-C4).



(a) Spin Images (b) FPFH Descriptor (c) 3DSC Descriptor (d) SHOT Descriptor

Fig. 7. Functional principles of feature descriptors: (a) The Spin Image descriptor spins a plane section around the keypoint normal to compute its 2D histogram. (b) The local reference frame and the three angles α, ϕ and θ computed by FPFH at each point pair. (c) The 3D spherical grid deployed by 3DSC. Two grid sectors, yielding each a bin in the 3D histogram, are depicted respectively in green and red. (d) The 3D grid deployed by SHOT, which is repeatably oriented by the local reference frame denoted by the blue arrows.

D. Transformation Estimation

Once outlier-free correspondences are reliably estimated, an Absolute Orientation algorithm is used to estimate the 6DOF transformation between the two clouds based on all remaining correspondences. Among the several algorithms available in literature, PCL includes an implementation of the method based on the Singular Value Decomposition (SVD) of the matrix representation of the 6DOF transformation proposed in [8, 10] (→ `TransformationEstimationSVD`).

Note that this is the same type of transformation estimation used in iterative registration where correspondences are not known but instead iteratively refined by always using closest points. A code sample showing how to use this transformation estimator has already been presented in Listing 6.

V. Experiments and Examples

A whole range of diverse applications benefits from reliable pairwise alignment between 3D point clouds. These applications differ in terms of: i) characteristics of the acquired data, e.g., high resolution polygonal mesh as opposed to low resolution point clouds; ii) working scale, e.g., large scale as in most urban and photogrammetry scenarios, as opposed to small scale, such as typically required for reverse engineering and object modeling; iii) a-priori knowledge, which depends on whether estimates of the acquisition viewpoints are available (in which case the initial estimate only needs to be *refined*) as opposed to the situation where they are unknown (in which case an *initial alignment* needs to be computed solely using the acquired data).

In addition to these differences which often require adopting ad-hoc solutions, one can also exploit certain characteristics of the data in order to improve speed or robustness. Since a fundamental problem of registration is to find the most suitable algorithm together with an appropriate parameterization, in this section we detail three different examples ranging from dense RGB-D image sequences for object modeling to high-resolution and low-resolution laser range scans for outdoor mapping of buildings and cities.

A. Large-scale 3D Laser Scans

Commercially available 3D laser scanners, as those used by surveying technicians, acquire high-resolution large-scale scans that usually consist of several million points. With a comparably ample maximum measurement range, these sensors can be used to capture large scenes such as buildings and streets. In order to create a complete model (e.g. a whole city, or an urban district), the acquired scans need to be aligned and aggregated. Frequently, surveying technicians use additional means such as special reflectors that assist in obtaining a fine alignment of the data. In our example, we assume that such information is not available and that we want to 1) compute an initial alignment for a pair of large-scale 3D laser scans, and 2) refine the initial estimate to obtain a perfect overlay of the acquired data.

1) Initial Alignment: As already mentioned in Section IV, PCL provides implementations for a variety of 3D descriptors that can be used for initial alignment. Here, the key issue is not only to find an appropriate descriptor for the problem at hand, but also an appropriate scale, i.e., the support size on which to compute each descriptor. Of particular interest in this context is the distinctiveness of each descriptor that influences the number of possible ambiguities in the matching stage as well as the repeatability such that the descriptor computed on the same scene point over different scans looks the same. A particularly robust approach is to compute the same class of descriptor at various scales and to focus the registration on points and associated descriptors that prove to be distinctive at the same scale, as well as to be persistent over multiple scales. PCL provides a joint estimation components that finds such keypoints and descriptors (→ `MultiScaleFeaturePersistence`).

Listing 10 shows an example for computing keypoints for an input point cloud together with the respective persistent feature descriptors. First, the selected descriptor (in our example we use FPFH [24]) is fetched to an instance of the MultiScaleFeaturePersistence class together with a vector of pre-selected scales and a distance metric. The class computes the descriptors with the given estimator for all desired scales and uses the provided distance metric to compute similarity. It returns the indices of all selected interest points that can then be used for an initial alignment.

In order to compute an initial alignment for a pair of target and source point clouds, we first match the estimated descriptors (`CorrespondenceEstimation` in Listing 11). We then reject false correspondences by finding a consistent set (in a RANSAC-like fashion) using `CorrespondenceRejectorSampleConsensus`. It samples from the estimated correspondences, computes a candidate trans-

Listing 10 Multi-scale keypoint and descriptor computation

```
#include <pcl/features/fpfh.h>
#include <pcl/features/multiscale_feature_persistence.h>
(...)
FPFHEstimation<PointXYZ, Normal, FPFHSignature33>::Ptr
  fest (new (...));
fest->setInputCloud (cloud);
fest->setInputNormals (normals);

MultiscaleFeaturePersistence<PointXYZ, FPFHSignature33>
  fper;
std::vector<int> keypoints;
std::vector<float> scale_values = { 0.5f, 1.0f, 1.5f };
fper.setScalesVector (scale_values);
fper.setAlpha (1.3f);
fper.setFeatureEstimator (fpfh_estimation);
fper.setDistanceMetric (pcl::CS);
fper.determinePersistentFeatures (*features, keypoints);
```

Listing 11 Initial alignment using keypoints and descriptors

```
#include <pcl/registration/correspondence_estimation.h>
#include <pcl/registration/
          correspondence_rejection_sample_consensus.h>
#include <pcl/registration/transformation_estimation_svd.h>
(...)
Correspondences::Ptr correspondences (new Correspondences);
CorrespondenceEstimation<FPFHSignature33, FPFHSignature33>
  cest;
cest.setInputSource (source_features);
cest.setInputTarget (target_features);
cest.determineCorrespondences (*correspondences);

Correspondences::Ptr corr_filtered (new Correspondences);
CorrespondenceRejectorSampleConsensus<PointXYZ> rejector;
rejector.setInputSource (source_keypoints);
rejector.setInputTarget (target_keypoints);
rejector.setInlierThreshold (2.5);
rejector.setMaximumIterations (1000000);
rejector.setRefineModel (false);
rejector.setInputCorrespondences (correspondences);;
rejector.getCorrespondences (*corr_filtered);

TransformationEstimationSVD<PointXYZ, PointXYZ> trans_est;
trans_est.estimateRigidTransformation (*source_keypoints,
  *target_keypoints, *corr_filtered, transform);
```

formation and then determines all inliers as correspondences where the respective source and target point are sufficiently close after applying the candidate transformation (parameter set using `setInlierThreshold`). The transformation yielding the highest number of inliers is the most likely initial alignment. All inliers are then used to compute the transformation by means of `TransformationEstimationSVD`.

2) Fine Registration: After the initial alignment, the source and target point clouds are already roughly registered. An iterative algorithm is used to correct the estimated viewpoints and to accurately align the point clouds.

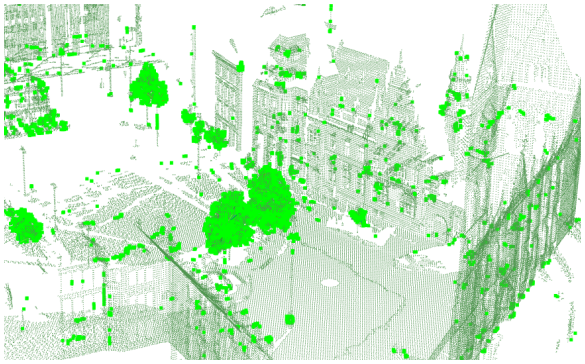
In Figure 8 we show the results of the proposed pipeline on typical large-scale 3D laser scans (from the "Bremen" data set) in the 3D Scan Repository[2].

B. Sparse low-resolution 3D Laser Scans

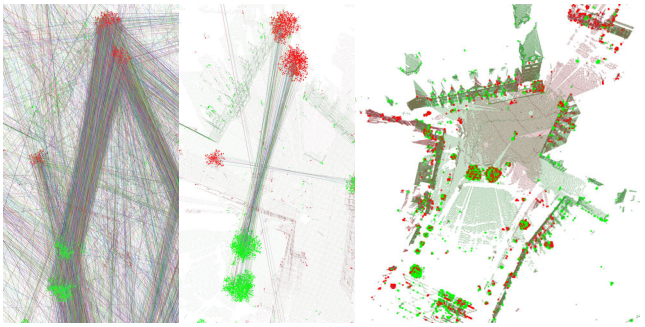
In the previous example, the 3D point clouds were not only very accurate but also very dense. We now focus attention on problems that arise when the data is particularly sparse and how to compensate for the resulting effects.

A particular characteristic of custom built 3D scanners composed of a rotating 2D laser range finder is that the acquired

[2]3D Scan Repository: http://kos.informatik.uni-osnabrueck.de/3Dscans



(a) Target point cloud with estimated keypoints (visualized with larger radii)



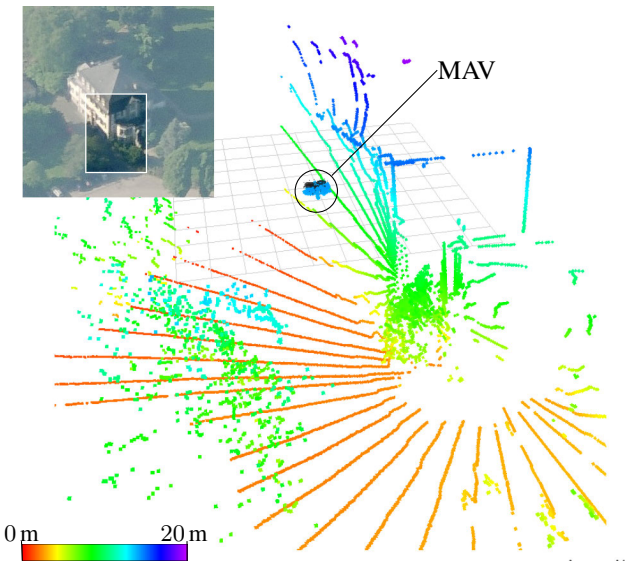
(b) Estimated correspondences (c) Filtered correspondences (d) Aligned point clouds

Fig. 8. Registration result: shown are the target point cloud with persistent keypoints (a), detail views of estimated correspondences (b) and filtered correspondences (c), as well as source and target point clouds after initial and refined alignment (d).

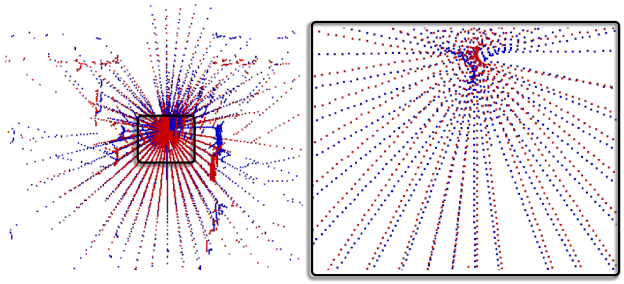
point clouds have non-uniform point densities: usually a high density within each scan line and a certain angle between scan lines which depends on the rotation speed of the scanner. Rotating the scanner fast increases the frequency with which the surroundings can be perceived (important for collision avoidance, e.g., as in our example of an autonomous micro aerial vehicle), but reduces the angular resolution and thus the point density in the acquired point clouds (see Figure 9 for an example). The resulting non-uniform point densities affect classic neighborhood searches in 3D and cause problems in local feature estimation and registration. For example, the *Generalized ICP* algorithm by Segal et al. [15] uses local surface information in the form of covariance matrices to align surfaces rather than individual points. However, as can be seen in Figure 9, inaccurate estimates (as caused by the non-uniform point densities) degrade the performance of the algorithm and negatively affect the registration result.

In order to compensate for these effects, we can approximately reconstruct the underlying surface and then compute the covariances directly on the reconstructed mesh [28]. We thereby exploit the topological information in the organized data and can accurately register such pairs of sparse point clouds. In this example, we show, how to approximate the surface reconstruction and the covariances, and how to use custom covariances with Generalized-ICP.

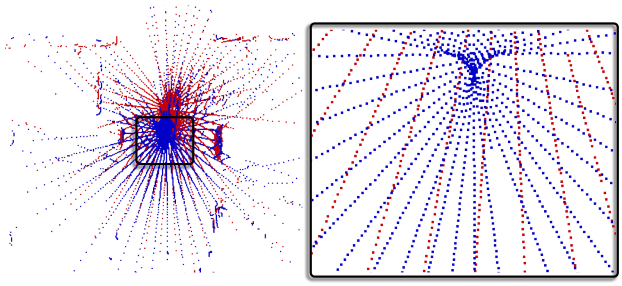
Listing 12 shows the main parts of the implementation using PCL. We first approximate the surface for both source



(a) Perspective view of an example scan with MAV and building. Point colors encode height. A photo highlights the scanned region.



(b) Original Generalized-ICP (top and detail view)



(c) Generalized-ICP w. approximated covariances (top and detail view)

Fig. 9. Registration of sparse laser scans: (a) example of a point cloud acquired while flying close to a building and trees. (b) The standard implementation of Generalized-ICP suffers from inaccurate covariance estimates and incorrectly aligns the two point cloud origins and the individual scan lines. (c) When approximating the underlying surface, computing the covariances on the reconstruction, and using these externally computed covariances for Generalized-ICP, we can accurately align the clouds.

and target point cloud (→ `OrganizedFastMesh`), and then compute estimates of the local surface normals and the covariances directly on the reconstructed meshes. We then use the standard Generalized-ICP algorithm (Section III-D6) with the covariances estimated on the reconstructed surface information (→ `GeneralizedIterativeClosestPoint`).

Listing 12 Generalized-ICP using approximated covariances.

```
#include <pcl/surface/organized_fast_mesh.h>
#include <pcl/registration/gicp.h>
#include <pcl/features/from_meshes.h>
PointCloud<PointT>::Ptr cloud;
PolygonMesh::Ptr mesh;
(...)
// reconstruct meshes for source and target
OrganizedFastMesh<PointT> fast_mesh;
fast_mesh.setInputCloud(cloud);
fast_mesh.reconstruct(*mesh);
(...)
// compute normals and covariances for source and target
PointCloud<Normal>::Ptr normals;
boost::shared_ptr< std::vector<Eigen::Vector3d> > covs;
pcl::features::computeApproximateNormals(
  *cloud, mesh->polygons, *normals);
pcl::features::computeApproximateCovariances(
  *cloud, normals, *covs);
(...)
// setup Generalized-ICP
GeneralizedIterativeClosestPoint<PointT, PointT> gicp;
gicp.setMaxCorrespondenceDistance(1);
gicp.setInputSource(source_cloud);
gicp.setInputTarget(target_cloud);
gicp.setSourceCovariances(source_covariances);
gicp.setTargetCovariances(target_covariances);

// run registration and get transformation
PointCloud<PointT> output;
gicp.align(output);
Eigen::Matrix4f transform = gicp.getFinalTransformation();
```

C. Registering Sequences of RGB-D Images

In the last example, we present a complete pipeline for aligning scans coming from a Microsoft Kinect, which is an example of an RGB-D camera, i.e., a 3D sensor that simultaneously acquires color and depth images at high frame rates (e.g., 30 Hz). These sensors have both data sources synchronized and registered, thus resulting in point clouds where each point has an associated RGB color and lies on a 2D grid structure. As mentioned earlier, in PCL we refer to such image-like 3D representations as *organized* point clouds.

It is assumed that consecutive clouds acquired from the sensor are temporally (and, thus, spatially) close to each other. For this to hold, not only the device must have a sufficiently high frame rate, but also the pipeline must process frames at a comparable speed. This is the working assumption of cloud registration algorithms deployed within SLAM and Kinect Fusion-like frameworks [29]. Under these conditions, the point clouds are close enough to allow ICP to reach a good local minimum in most cases without the need for the initial alignment procedure and with just a few iterations. We will briefly walk through all steps of the pipeline.

1) Preprocessing — Filtering and Fast Feature Estimation: This type of cameras exhibits various forms of noise. Hence, initial filtering becomes necessary when the cameras are used to acquire accurate 3D models. There are multiple ways in which noise can be efficiently eliminated from generic point clouds, but none of them takes into account the difficult quantization effects seen in this new type of data. An edge-preserving filtering method—bilateral filter—has been adapted from the field of 2D image processing (→ `FastBilateralFilter`). It smooths more when neighboring pixels are similar (flat regions), and less when there are jumps, thus not affecting the edges. Listing 13 provides a usage example.

Listing 13 Filtering (and Upsampling) for RGB-D Data

```
#include <pcl/filters/fast_bilateral.h>
(...)
FastBilateralFilter<PointXYZ> filter;
filter.setInputCloud (cloud_in);
filter.setSigmaS (5);
filter.setSigmaR (0.005f);
bilateral_filter.filter (*cloud_out);
```

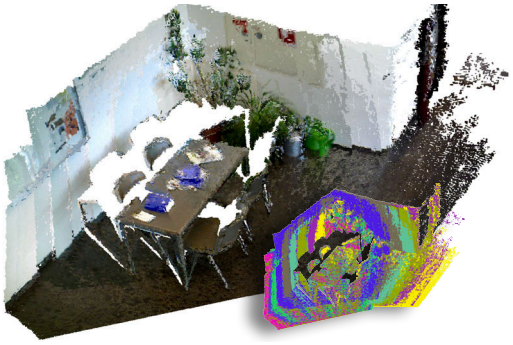


Fig. 10. Typical result of registering a sequence of RGB-D frames.

Accelerating the computation of normals can also be done by using the grid structure of the RGB-D data. The rasterized depth images can be used to speed up the normal estimation process by using integral images [30]. The advantage of this method is that it requires only a linear pre-processing stage while allowing for computing mean vectors and normals within a rectangular area of the image in constant time (→ `IntegralImageNormalEstimation`).

2) Registration Pipeline: For aligning a sequence of RGB-D images, we present a hybrid registration pipeline. Because of the high data rate of these sensors ($640 \times 480 = 307\,200$ points at 30 Hz), the measurements need to be sampled, e.g., such that the registered point clouds contain only 10 % of the original data. As described in Section III-A, there are many approaches available for sampling the point cloud.

a) Normal space subsampling: A particularly robust way to register RGB-D point clouds is to perform normal space sampling (→ `NormalSpaceSampling`). It consists in binning each point into M^3 bins based on the normal angle around each of the x, y, and z-axis, respectively. Next, a fixed number of samples are extracted randomly from each bin. This ensures that the resulting point cloud has samples at locations from all the differently oriented surfaces in the scene, increasing the probability that the registration will converge to the global minimum.

b) Correspondence estimation and filtering: Having only a fraction of the points left, using a complex search data structure is not that prohibitively slow anymore. Consequently, we apply the correspondence estimation based on kd-trees (Section III-B) rather than the projection-based estimation.

In order to reject misleading correspondences, we employ a combination of two correspondence filters (Section III-C): a filter based on normal compatibility (→ `CorrespondenceRejectorSurfaceNormal`) to reject point pairs with mismatched normals, and a distance filter using the median distance between correspondence pairs (→ `CorrespondenceRejectorMedianDistance` to eliminate the pairs with outlying distances.

c) Transformation estimation and weighting: Using the resulting small set of robust correspondences, the transformation between the two point clouds is computed using the weighted point-to-plane distance metric (→ `TransformationEstimationPointToPlaneWeighted`). Because our normals have been computed on the original point clouds, they approximate the underlying surface well. As a result, the point-to-plane error metric is preferred as a better estimation of the distance between the source and target surfaces, as opposed to point-to-point measurements. The weighting of the correspondences can be done by using a sensor noise model. One such example was proposed by [13] and considers the standard deviation of the depth to increase quadratically with the distance of the scanned surfaces from the image plane. As such, the weight for the correspondence between points **p** and **q** is taken as the maximum of the standard deviation of the two points under the noise model:

$$w(\boldsymbol{p}, \boldsymbol{q}) = \max\left(\sigma_z\left(\boldsymbol{p}_z\right), \sigma_z\left(\boldsymbol{q}_z\right)\right), \quad (5)$$

$$\text{with } \sigma_z(z) = 0.0012 + 0.0019 * (z - 0.4)^2. \quad (6)$$

As convergence criteria for the ICP algorithm (Section III-E), a good configuration is to use the number of iterations (chosen based on the time we have allocated for registration) and the length of the incremental translation and rotation from the last iteration (chosen to be lower than the expected camera movement between frames).

d) Typical registration results: For the purpose of the experiment, we recorded sequences of RGB-D frames at 30 Hz. Each frame is registered in a pairwise fashion against the previous one. Results are shown in Figure 10 and Figure 1. Note that if a large number of frames are registered pairwise, individual alignment errors accumulate. There are numerous techniques in the literature to solve this issue, such as loop closing via graph optimization or forms of global registration.

VI. Conclusion

In this article, we have presented the modular registration framework implemented in the open-source Point Cloud Library (PCL). Following the usual scheme of registration algorithms, we explained different modules available in PCL and presented code snippets showing how to use them. Finally, we presented three examples of registration approaches, each tailored to a specific type of input data so as to show how the different components in PCL can be efficiently combined to solve various registration problems.

A sequel to this tutorial focusing on global multi-view registration is planned. The complete example implementations have been made available together with the used datasets at: http://pointclouds.org/media/registration-paper.html.

References

[1] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *International Conference on 3-D Digital Imaging and Modeling (3DIM)*, 2001.

[2] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 14, no. 2, pp. 239–256, 1992.

[3] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, no. 3, pp. 145–155, 1992.

[4] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application (VISAPP)*, 2009, pp. 331–340.

[5] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison on nearest-neigbour-search strategies and implementations for efficient shape registration," *Journal of Software Engineering for Robotics (JOSER)*, vol. 3, no. 1, pp. 2–12, 2012.

[6] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[7] J. Diebel, K. Reutersward, S. Thrun, J. Davis, and R. Gupta, "Simultaneous localization and mapping with active stereo vision," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 3436–3443.

[8] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 9, no. 5, pp. 698–700, 1987.

[9] D. W. Eggert, A. Lorusso, and R. B. Fisher, "Estimating 3-D rigid body transformations: a comparison of four major algorithms," *Mach. Vision Appl.*, vol. 9, no. 5-6, pp. 272–290, 1997.

[10] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629–642, 1987.

[11] A. W. Fitzgibbon, "Robust registration of 2D and 3D point sets," in *British Machine Vision Conference (BMVC)*, 2001, pp. 411–420.

[12] K.-L. Low, "Linear least-squares optimization for point-to-plane ICP surface registration," in *Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill*, 2004.

[13] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling Kinect sensor noise for improved 3D reconstruction and tracking," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, 2012, pp. 524 –530.

[14] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *International Journal of Robotics Research (IJRR)*, vol. 31, no. 5, pp. 647–663, 2012.

[15] A. V. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Robotics: Science and Systems*, 2009.

[16] N. Gelfand and S. Rusinkiewicz, "Geometrically stable sampling for the ICP algorithm," in *International Conference on 3D Digital Imaging and Modeling*, 2003, pp. 260–267.

[17] C. Harris and M. Stephens, "A combined corner and edge detection," in *4th Alvey Vision Conference*, 1988, pp. 147–151.

[18] F. Tombari, S. Salti, and L. D. Stefano, "Performance evaluation of 3D keypoint detectors," *Int. Journal of Computer Vision*, vol. 102, no. 1-3, pp. 198–220, 2013.

[19] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3D object recognition," in *ICCV Workshop on 3D Representation for Recognition (3dRR)*, 2009.

[20] B. Steder, R. Rusu, K. Konolige, and W. Burgard, "NARF: 3D range image features for object recognition," in *IROS Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics*, 2010.

[21] A. Aldoma, Z. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. Rusu, S. Gedikli, and M. Vincze, "Point Cloud Library: Three-dimensional object recognition and 6 DOF pose estimation," *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 80–91, 2012.

[22] L. Alexandre, "3D descriptors for object and category recognition: a comparative evaluation," in *IROS Workshop on Color-Depth Camera Fusion in Robotics*, 2012.

[23] A. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 21, no. 5, pp. 433–449, 1999.

[24] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 1848–1853.

[25] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik, "Recognizing objects in range data using regional point descriptors," in *European Conference on Computer Vision (ECCV)*, 2004, pp. 224–237.

[26] F. Tombari, S. Salti, and L. Di Stefano, "Unique shape context for 3D data description," in *ACM Workshop on 3D Object Retrieval (3DOR)*, 2010, pp. 57–62.

[27] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *European Conference on Computer Vision (ECCV)*, 2010, pp. 356–369.

[28] D. Holz and S. Behnke, "Mapping with micro aerial vehicles by registration of sparse 3D laser scans," in *International Conference on Intelligent Autonomous Systems (IAS)*, 2014.

[29] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: real-time dense surface mapping and tracking," in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011, pp. 127–136.

[30] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 2684–2689.