POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

AI-Based Point Cloud Registration

Supervisors

Prof. Raffaello CAMORIANO

Dr. Enrico CIVITELLI

M.Sc. Paolo RABINO

Prof. Larissa DRIEMEIER

M.Sc. Luca DI RUSCIO

Candidate

Gabriel CORTELETTI

March 2026

Abstract

With the growing adoption of automation in industrial settings, computer vision has become increasingly important for enabling robots and systems to perceive their surroundings. A core 3D vision task is point cloud registration: estimating the rigid motion (rotation and translation) that aligns two 3D scans of the same subject captured from different viewpoints, often with partial overlap. Reliable registration supports applications such as 3D reconstruction, robot localization and mapping, and object pose estimation. Since pipelines often succeed or fail depending on the quality of the initial correspondences, we isolate how the choice of local feature descriptor affects alignment quality through a controlled comparison between classic, handcrafted descriptors and modern deep-learning based alternatives within otherwise identical registration pipelines.

We study this problem using a deliberately simple, controlled pipeline to isolate the effect of feature descriptors. A fixed global–to-local stack, with feature matching, Random Sample Consensus (RANSAC) for initial alignment, and Iterative Closest Point (ICP) for refinement, is implemented in Open3D twice: once with the handcrafted Fast Point Feature Histograms (FPFH) descriptor and once with the learned Fully Convolutional Geometric Features (FCGF). To ensure a fair comparison, both variants share the same hyperparameters and a conservative cap on ICP iterations, and the system is instrumented with iteration counters and per-stage timing. We evaluate across three regimes: (i) the 3DMatch test split; (ii) a noise-augmented 3DMatch benchmark (Gaussian noise, spikes, pepper); and (iii) a cross-domain CAD to RGB-D setting adapted from SuctionNet, where a lightweight centering step preconditions point clouds. Metrics follow established practice and the official 3DMatch protocol, reporting fitness, inlier RMSE, and precision/recall via rotation/translation error checks (RRE/RTE).

Empirically, stronger correspondences upstream (FCGF) resulted in faster and steadier consensus and lighter refinement downstream: fewer RANSAC steps, better ICP start basin, improved robustness under injected noise while maintaining quality; under cross-domain, generalization remains competitive with minimal preconditioning. These results provide a controlled, reproducible baseline connecting descriptor quality to end-to-end efficiency and accuracy in registration pipelines.

Acknowledgements

I am deeply grateful to my supervisors, Raffaello Camoriano (Politecnico di Torino), Larissa Driemeier (Escola Politécnica da Universidade de São Paulo), and Enrico Civitelli (Comau). This work is the result of a collaboration between universities from different countries and industry, and I thank you all for your continued guidance, support, and encouragement throughout the work. You were always available, offering close supervision, often weekly, and helping me make the best decisions with your expertise.

My deepest gratitude goes to my family and friends. To my parents, Antônio Tápias and Rosangela Corteletti, and to my sister, Gabriela Corteletti, for their unconditional support throughout my life and academic journey. You have been the pillars of my achievements. Also to my friends from Brazil, Italy, and many other nationalities whom I was fortunate to meet, your friendship made the journey lighter and happier.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

BNB
Branch-and-bound

CAD
Computer-Aided Design

CNN
Convolutional Neural Networks

CPD
Coherent Point Drift

DoF
Degrees of Freedom

FCGF
Fully Convolutional Geometric Features

FCN
Fully Convolutional Networks

FPFH
Fast Point Feature Histograms

GNN
Graph Neural Networks

GNC
: Graduated Non-Convexity

ICP
: Iterative Closest Point

LS
: Least-Squares

NDT
: Normal Distributions Transform

NN
: Nearest Neighbors

PCA
: Principal Component Analysis

PCR
: Point Cloud Registration

PFH
: Point Feature Histograms

RANSAC
: Random Sample Consensus

RMSE
: Root Mean Square Error

RRE
: Relative Rotation Error

RTE
: Relative Translation Error

SD
: Standard Deviation

TDF
: Truncated Distance Function

Chapter 1

Introduction

Reliable 3D point cloud registration is the base of a long line of perception tasks in robotics, mapping, augmented reality, and industrial inspection. In its most common form, the problem is posed as pairwise alignment: given a source and a target cloud captured from different viewpoints, estimate the rigid transform that best aligns them. Classical pipelines approach this with a global stage to break symmetries and reject outliers, such as Random Sample Consensus (RANSAC) [1] with a closed-form 6-DoF solver, followed by a local refinement stage to polish the pose, such as point-to-plane Iterative Closest Point (ICP) [2, 3]. Whether these systems succeed or fail depends critically on the quality of correspondences produced by the feature extractor upstream.

Handcrafted descriptors, such as Fast Point Feature Histograms (FPFH) [4], summarize local surface geometry through histograms and have long been the default in real systems due to their simplicity and interpretability. More recently, learned descriptors trained directly on point clouds, such as Fully Convolutional Geometric Features (FCGF) [5], have shown strong repeatability on real scans and better tolerance to clutter and self-similar structures. A practical question that emerges is how much we can gain by replacing the descriptor while keeping the rest of the pipeline (i.e., the downstream registration stages) unchanged. This is the central question of this thesis.

We implement and compare two pipelines built on Open3D[1]: a geometric-only baseline using FPFH, and a deep-learning variant that swaps FPFH for FCGF while keeping the same global (feature-based RANSAC with Kabsch-Umeyama [6]) and local (point-to-plane ICP) stages. We instrument both with iteration counters

[1]Open3D is an open-source Python library for 3D data processing. It provides tools for loading, processing, visualizing, and analyzing point clouds, meshes, and RGB-D (Red, Green, Blue and Depth) images, and includes well-established classical registration algorithms widely used in computer vision and robotics. Available at `https://www.open3d.org/`.

and stage-level timers and evaluate them in three complementary settings: (i) the standard 3DMatch benchmark (Section 4.5), (ii) a noise-augmented 3DMatch variant with controlled corruptions (Section 5.2), and (iii) a cross-domain object benchmark adapted from SuctionNet (Section 5.3) where sources are Computer-Aided Design (CAD) models and targets are RGB-D reconstructions.

Concretely, we ask:

1. Does swapping in learned descriptors reduce the effort of global and local stages (iterations)?

2. Does it improve alignment quality (fitness and inlier Root Mean Square Error, RMSE) and end-to-end runtime?

3. How do both pipelines behave under synthetic degradations (Gaussian noise, sparse spikes, salt-and-pepper dropout)?

4. Do the gains carry to a cross-domain alignment where the appearance gap is large?

Section 4 formalizes the pipeline and implementation choices. Preprocessing includes voxel downsampling and normal estimation. Feature extraction is either FPFH or FCGF. Tentative correspondences are built by 1-Nearest-Neighbor (1-NN) in feature space with a mutual filter. RANSAC estimates a rigid transform from minimal 4-point subsets and uses distance and edge-length checkers before scoring. ICP refines with a Gauss–Newton point-to-plane update.

To analyze effort, we capture RANSAC and ICP iteration counts by parsing Open3D's debug logs. To analyze time, we wrap stages with a high-resolution timer. For accuracy we report fitness and inlier RMSE at both stages, and (on 3DMatch) recall and precision using the official toolbox (Relative Rotation Error, RRRE $\leq 15^\circ$, and Relative Translation Error, RTE $\leq 0.3\,\mathrm{m}$). For robustness we introduce an augmentation module: variable-variance Gaussian noise; sparse, positively skewed spikes; and salt-and-pepper dropout. For the cross-domain study, we adapt SuctionNet scenes by reconstructing per-object point clouds from RGB-D and add a lightweight centering step that de-means both clouds at the origin prior to feature computation, leveraging cheap numerical operations that improved stability under large offsets.

Across standard 3DMatch, replacing FPFH with FCGF reduces RANSAC effort by roughly one third and ICP by about one fifth, while raising fitness at both stages and lowering inlier RMSE (Section 5). End-to-end time shrinks not only because there are fewer iterations to pay for, but also because feature extraction itself is faster with FCGF, matching prior reports on 3DMatch where FCGF exhibits lower per-feature latency than classical descriptors. Under injected corruptions, the deep-learning pipeline remains more resilient: it converges in fewer RANSAC

validations and attains higher fitness and lower RMSE, even when ICP hits a fixed iteration cap in the heaviest-noise settings. In the cross-domain benchmark, FCGF again helps on the global stage (iterations, fitness, RMSE), with ICP performing similarly on average between pipelines. Estimated recall improves modestly overall, with object-to-object variability expected from the domain gap.

Along the way we document practical behaviors that matter in the field: how conservative feature matching under noise can clean correspondences and raise the inlier fraction, shortening RANSAC according to its confidence bound; how a small, fixed ICP budget behaves under noise; and how a simple de-meaning step improves numerical conditioning when clouds start far from the origin.

We summarize the main technical contributions and empirical results below.

- An FPFH–RANSAC–ICP baseline and an FCGF swap-in variant, implemented on Open3D [7], sharing the same global/local solvers and hyperparameterization for a fair comparison. Public code release for end-to-end experiments and figures ([8]).

- A lightweight module that captures RANSAC and ICP iteration counts from Open3D logs and stage-level timers, enabling device-agnostic effort analysis and transparent runtime accounting.

- A controllable noise augmentation module with variable-variance Gaussian noise, skewed spike outliers, and pepper dropout, plus full ablations (none, Gaussian, Gaussian+spikes, Gaussian+spikes+pepper) used to stress-test both pipelines.

- A novel cross-domain benchmark assessing a CAD to RGB-D setup derived from SuctionNet [9] with per-object reconstructions, a simple centering preconditioner, and a protocol for approximate recall without ground truth (fitness-based success proxy).

- With learned features: (i) RANSAC iterations drop by $\sim$30% and ICP by $\sim$20%; (ii) fitness rises at both stages (deep-learning RANSAC alone already surpasses geometric-only ICP); (iii) inlier RMSE decreases at both stages (millimeter-level gains); (iv) end-to-end runtime is markedly lower, consistent with FCGF's faster feature extraction on 3DMatch [5].

- The learned pipeline maintains higher fitness and lower RMSE across all noise settings; RANSAC variability widens as noise grows (explained by the inlier fraction in the confidence bound), while ICP often meets the iteration cap in the heaviest-noise cases yet still finishes with lower error due to better initialization.

- Learned features reduce global effort and error on CAD to RGB-D pairs; ICP averages are similar across pipelines; estimated recall improves modestly overall with notable object-level variation, suggesting that light domain adaptation or coarse-to-fine schedules could close the remaining gap.

Overall, this thesis makes a practical point: modern learned descriptors can be dropped into a familiar RANSAC and ICP stack to yield faster, more accurate registration with minimal changes elsewhere. The rest of the thesis details how we built, measured, and stress-tested that claim, and where the margins still are—and should be targeted next. The full code developed for this thesis and its experiments is provided at [8].

Chapter 2

Related Works

Point cloud registration (PCR) estimates the rigid transform that aligns a source cloud to a target in the presence of noise, partial overlap, and viewpoint change. In practical settings (robotics, mapping, inspection), scans are incomplete, sampling densities differ, and outliers are common. These conditions impose three recurring requirements on a deployable pipeline: (i) robustness to mismatches and missing geometry, (ii) low latency and modest compute, and (iii) stability across scene scales, sensors, and operating regimes. Modern systems address these demands with a coarse-to-fine structure: first obtain reliable correspondences and a robust coarse pose, and then refine locally at geometric resolution.

Most recent progress consolidates around correspondence-based strategies. The central object is the correspondence set, which includes pairs of points hypothesized to represent the same surface location across different clouds. Pipelines differ primarily in how this set is constructed and how it is used to estimate pose. On construction, we distinguish (i) feature-based methods that compute descriptors and match them in feature space, (ii) geometry-based methods that rely directly on spatial proximity (typically assuming a good initialization), and (iii) hybrids that couple a descriptor-driven coarse alignment with a geometric fine stage. Within feature-based approaches, we further separate keypoint-based versus keypoint-free extraction, and hand-crafted versus learned descriptors. Keypoint-based pipelines trade density for repeatability and speed, while dense keypoint-free schemes offer higher recall but require downstream filtering (often via attention mechanisms or graph reasoning) to suppress ambiguous matches from low-texture regions.

Given a tentative correspondence set, global registration estimates an initial SE(3) transform while tolerating high outlier rates. Representative families include hypothesize-and-test consensus (e.g., RANSAC [1]), robust optimization with graduated or truncated penalties, convex/certifiable relaxations, and graph formulations that enforce pairwise consistency among matches. The trade-off is clear: stronger robustness and guarantees typically increase compute, while lighter solvers reduce

latency but rely on well-formed correspondences and effective sanity checks.

With a reasonable initialization in hand, local registration refines the pose by minimizing geometric residuals. Classical ICP [2] and its variants (point-to-point, point-to-plane, generalized) remain widely used due to simplicity, speed, and reliable convergence in the small-residual regime. Alternatives such as Normal Distributions Transform (NDT) [10] and Coherent Point Drift (CPD) [11] adjust the residual model for smoother objectives or probabilistic robustness. Practical implementations rely on acceleration structures (e.g., KD-trees), multi-scale strategies, and robust kernels or trimming to resist the residual outliers that survive the global stage.

Across these threads, a few design principles emerge for reliable PCR: (i) adopt a coarse-to-fine structure to decouple outlier handling from precision fitting, (ii) prefer descriptors and normalizations stable under sampling density and moderate rotations, (iii) attach confidence to matches and enforce geometric consistency before pose estimation, (iv) use estimators with explicit outlier models at the global stage and well-conditioned residuals at the local stage, (v) control runtime via compact keypoint sets or sparse convolution backbones, efficient neighbor search, and early rejection tests, and (vi) evaluate with metrics that reflect deployment goals, such as pose error, recall, inlier ratios, and time.

This chapter is organized accordingly. Section 2.1 reviews feature extraction strategies, contrasting keypoint-based and keypoint-free pipelines and discussing descriptor design and correspondence refinement. Section 2.2 surveys global registration methods that convert tentative matches into a robust initial transform, before deep diving in RANSAC. Section 2.3 covers local refinement algorithms and the practical choices that make them fast and stable, before focusing on ICP and its alternatives. Together, these sections motivate the pipeline choices and evaluation criteria adopted in the remainder of this work.

2.1 Feature Extractors

In point cloud registration (PCR) tasks, the end goal is to align two point clouds (source and target). However, since different scans usually capture distinct views of an object or scene, it is common for them to contain points representing parts that are absent in the other cloud. Thus, not all points should be aligned. So, the first step is to identify which points are present in both clouds, representing the same thing and, therefore, should be aligned. These point-to-point matches are called correspondences pairs, and the collection of all these pairs form the correspondence set.

Methods that rely on this correspondence set to estimate the rigid transformation are commonly referred to as correspondence-based methods, and they dominate

recent advances in 3D registration. In order to build the correspondence set which will later be used to generate the alignment, there are typically three approaches: (i) feature-based, (ii) geometry-based, and (iii) hybrid.

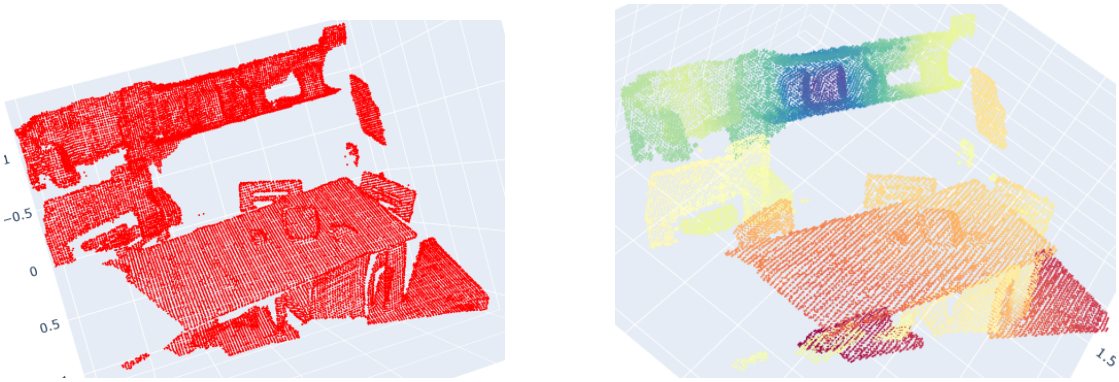
First, the feature-based approach establishes correspondences between points based on the geometric features and aspects of the cloud, such as shape and curvature. These models encode local geometry information for each point (or patch) in each cloud, before matching them across clouds. In this way, the comparison and matching accounts for how similar are the overall structures each point or set of points form (e.g., edges and corners). This branch of models includes classical, hand-crafted ones such as FPFH for fast local normal histograms [4] and Signature of Histograms of Orientations (SHOT) for orientation histograms in a repeatable local reference frame [12], and modern learned ones like FCGF for dense learned features [5], D3Feat for joint keypoint detection and description [13], GeoTransformer for superpoint matching with geometric attention [14], and Deep Global Registration (DGR) [15] for deep correspondence weighting and global registration. These methods have driven much of the recent progress due to improved robustness under partial overlap, sensor noise, and large initial misalignment [4, 5, 14].

However, it is not strictly necessary to rely upon these features, and, instead, the geometry-based approach simply relies on the raw coordinates (and sometimes normals) directly. In these cases, the correspondences are set by straightforward distance metrics (i.e., geometric proximity). This means that the basic assumption here is that points that are close to each other are similar and, therefore, must be aligned. Because of this, although these methods have the advantage of skipping the feature extraction stage, the downside is that they are heavily sensitive to the initial alignment (i.e., they consistently present high performance if, and only if, the clouds are already fairly aligned and close to each other). If the initialization is not precise enough, with a pose far from the optimum, they may fall into local minima or fail altogether, becoming much less reliable [16]. Hence, this class of methods is typically adopted as a fine-alignment stage to refine an obtained coarse initial transformation estimate [17]. It comprises mainly classical and geometry-driven algorithms, most notably the Iterative Closest Point (ICP) [2] and its variants (like point-to-plane ICP [3] and Generalized ICP (G-ICP) [18]), as well as related matching formulations such as the NDT [10] and CPD [11].

Then, the so-called hybrid approach, which has become prevalent in many modern PCR pipelines, as the name suggests, works by combining the previous two approaches: a feature-based model is used to reliably produce an initial rigid transformation that aligns the clouds well enough to serve as a robust initialization to a following local refinement stage, which increases the fit and precision of the previous transformation [4, 19]. This leverages the strengths of both branches while mitigating their weaknesses, resulting in a robust approach.

A feature in this context is a compact, discriminative representation of local

geometry around a point (or patch). It goes beyond coordinates by encoding structural context such as surface variation, angles, or neighborhood configuration, thereby supplying richer information to the pose estimator [4, 5, 14, 20]. Figure 2.1 shows a RedKitchen fragment and the same cloud colorized by per-point feature descriptors. Points that are similar in feature space (i.e., locally similar geometry) tend to share matching colors, so similar objects appear in related hues (e.g., the chairs).



(a) Raw cloud **(b)** Feature-colored cloud

Figure 2.1: Feature visualization on a 3DMatch RedKitchen fragment. (a) Raw point cloud. (b) The same cloud, with each point colored by its descriptor (projected to RGB). Similar colors indicate similar local geometry. Note that absolute colors have no semantic meaning, only relative similarity matters.

The feature extraction strategy adopted represents a major design choice, and PCR pipelines can be typically split into two categories: (i) keypoint-based and (ii) keypoint-free methods.

Key-point based methods first detect salient and distinct, or repetitive, points (called keypoints), which can be understood as a location of interest that stands out from its surrounding (e.g., corners and edges) and can therefore be leveraged as a reference to align two clouds. Several detectors have been proposed, and they can be hand-crafted, such as Intrinsic Shape Signatures (ISS) [21] and 3D Scale-Invariant Feature Transform (SIFT) [22], or learned, like Unsupervised Stable Interest Point (USIP) [23]. Once the keypoints have been detected, their descriptors are computed. A descriptor works similar to an embedding of the keypoints, it encodes the local geometry information into a vector space, i.e. the feature space. This process of summarizing and converting geometric information into a specific space domain, allows for easier and faster matching and operations in the following steps [17]. Descriptors too can be obtained through classical approaches, like FPFH [4] and SpinImages [24], or on deep-learning, like FCGF [5] and D3Feat [13].

Then it is finally possible to match the descriptors between source and target points by finding those that are closer in the feature space. Therefore, the distance measured here is not geometric, but a representation of similarity [17]. With the correspondence set built, the rigid transformation can be estimate through methods such as RANSAC [1].

Although selecting a set of few but well-descriptive keypoints makes the previous approach efficient, it also results in sparse correspondences, meaning less information to be leveraged when estimating the transformation. Furthermore, in cases where the cloud presents low-texture or noisy regions it might be hard to detect reliable keypoints. The alternatives arises as the keypoint-free methods, which avoid the explicit keypoint detection step by, instead, processing the whole cloud (or a dense subset). Then, after obtaining the descriptors for all (or at least most) points, as the previous approach, the points are matched based on pairwise feature similarity scores (such as distance in feature space).

Since features were obtained for a large number of points, this step yields a dense correspondence set [5, 14], which would be computationally demanding to be considered whole. Additionally, by neglecting the detection of locally salient points that presented high geometric distinctiveness, keypoint-free methods are sensitive to ambiguous points, without any geometric peculiarity. Therefore, many points from flat, featureless and repetitive regions of the cloud will be analyzed. Due to their uniform and bland geometry, these points might appear similar locally, producing closely resembling descriptors even if they are not true matches globally (e.g., points from two separate flat areas of the cloud). For these two reasons, most models that follows this methodology introduce a matching-refinement stage that uses contextual reasoning to filter the initial dense correspondence set. For this end, several implementations have been proposed, with common ones relying on the use of attention mechanisms or Graph Neural Networks (GNNs) [5, 14, 25].

For attention-based refinements, the descriptor of a source point is updated by aggregating information from other points identified as relevant via learned attention weights. With cross-attention, each point in one cloud attends to points in the other cloud. This means that, for each source point, the model compares it to all (or top-k) target points, computes attention weights based on feature similarity, and performs a weighted aggregation of target descriptors to update the source point descriptor or to produce match confidences for filtering false positive [14, 26]. Because target points that appear more compatible with the current source point receive higher weights, they contribute more to the update, making the source descriptor increasingly similar to theirs. This allows the source descriptor to become aware of the target points or regions it aligns with, thereby facilitating correspondence matching and reducing outlier correspondences [14]. Besides that, a simpler approach is to adopt the attention weights directly as matching-confidence scores, since they quantify how strongly each source point attends to each target

point [14, 26].

Alternatively, with self-attention, instead of comparing both clouds, each source point attends to its neighboring points within the same cloud. This injects local context (i.e., information about the geometry immediately around the points), and long-range context (i.e., how this region compares to farther structures in the same cloud). This results in more discriminative descriptors where, for instance, flat regions that looked locally identical become distinguishable once the surrounding geometry (curvatures, edges, corners, etc.) are encoded. Thus, by reducing ambiguous features, the subsequent cross-matching yields fewer outliers [13, 14].

On the other hand, GNN-based refinements usually represent tentative correspondences as nodes which are connected by edges that encode geometric proximity or consistency. For instance, two correspondences (nodes) are linked (by edges) if they are mutually consistent under a rigid motion, meaning that given two correspondences, the distance, normal, and angle between both source points are similar to those between the two target points. Then, the descriptors of the points are updated by the neighboring (i.e. connected) nodes, similar to the attention based approach. A GNN then passes over this correspondence graph, updating each node's embedding based on its compatible neighbors (linked nodes) and producing a confidence (inlier) score per correspondence. These scores are used to filter the tentative matches [25, 27].

In summary, while keypoint-free pipelines have impressive robustness, they are often heavier and more complex, which can be incompatible with industrial settings that operate under tight latency and low-compute constraints [5, 14, 26]. For this reason, after this overall review of feature extractors, we now concentrate on keypoint-based registration. As noted earlier, we can further split this class based on the strategy they rely on to obtain the descriptors: (i) classical geometric algorithms, grounded in analytic models and heuristic design (i.e., manually engineered descriptors), or (ii) AI-based models, that leverage learned detectors and descriptors. In what follows, we give a quick overview of notable advances in each category and deep-dive into those that will be selected to build the pipelines to be analyzed in this study.

The initial work on hand-crafted 3D local descriptor extractors set the foundation for correspondence-based registration [28]. SpinImages projects a point's neighborhood to a 2D accumulator anchored on the surface [24]. Unique Shape Context (USC) anchors a shape-context histogram with a unique local reference frame to improve distinctiveness and memory use [29]. SHOT aggregates histograms of normals within a robust and repeatable local frame [12]. Point Feature Histograms (PFH) and FPFH summarize pairwise angular relations of normals and relative geometry as compact orientation-aware histograms [4, 30].

Among these, we focus on FPFH because it preserves much of PFH's geometric discriminative power, offering approximately pose-invariant local features, while

dramatically reducing the computational complexity [4, 17, 28]. Hence, to understand FPFH, one must first comprehend PFH. Given a set of 3D points and their estimated normals, often obtained by Principal Component Analysis (PCA) in a local patch [31], PFH defines, for each point p, a sphere of radius r around it, delimiting its neighborhood. Then, all points within this sphere are selected (k-neighbors of p). For each pair of different points p_i and p_j in this neighborhood and their estimated normals n_i and n_j, where p_i is defined as the point of the pair that has the smaller angle between its normal and the line connecting the pair. A Darboux frame is defined using n_i through:

$$\begin{aligned} u &= n_i \\ v &= (p_j - p_i) \times u \\ w &= u \times v \end{aligned} \tag{2.1}$$

Then, using $(u, v, w))$ and n_j, PFH computes three angular variations:

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= \frac{u \cdot (p_j - p_i)}{\|p_j - p_i\|} \\ \theta &= \arctan\left(w \cdot n_j, u \cdot n_j\right) \end{aligned} \tag{2.2}$$

PFH repeats this for all point-pairs in the k-neighborhood accumulating the occurrences of α, ϕ and θ into a multi-dimensional histogram (i.e., it saves each feature into discrete fixed bins of uniform ranges). Thus, this histogram, often normalized, represents the consolidation of the local geometry around point p, i.e. its descriptor (or feature) [4, 30].

While PFH produces highly descriptive features, its computational complexity of $\mathcal{O}(n \cdot k^2)$ where n is the number of points and k is the select neighborhood size, grows too quickly, making this approach unfit for real-time, or even near real-time, industrial applications. To solve this issue, FPFH was proposed, reducing the complexity to $\mathcal{O}(n \cdot k)$ while maintaining roughly the same feature performance of PFH [4].

FPFH begins similar to PFH, where for each point p a sphere defines its k-neighborhood. Then, it too computes the angular variations using (2.2). However, instead of computing these relations for all pairs between neighbors themselves, FPFH does it only for pairs between p and a neighbor. The resulting histograms are defined as the Simplified Point Feature Histogram (SPFH). Then, after SPFH has been obtained for all keypoints, a refinement step takes place. In it, for each point and its $k - neighbors$, FPFH tunes its SPFH descriptor by aggregating the SPFH descriptors of its neighbors through distance-based weights ω_k (i.e., neighbors that are closer to p will have more influence on the refinement of its descriptor):

$$FPFH(p) = SPF(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}SPF(p_k) \tag{2.3}$$

Finally, this is followed by the usual histogram normalization [4]. In practice, the results are compatible with the results of PFH, with the major gain of higher scalability [4, 17, 28], which is the motivation of our choice for this extractor later on this study.

Now, we switch context to learned feature extractors. As mentioned, these represent the most recent advances in 3D descriptors. By adopting deep-learning architectures (whether supervised or self-supervised), they are able to learn more constant and wider geometric context directly from data.

Early neural approaches segmented local neighborhoods and learned volumetric Truncated Distance Function (TDF) descriptors, as in the 3DMatch descriptor [32]. Other methods align local patches to a consistent local frame, convert the points into a smooth 3D grid (a blurred occupancy map), and read features at multiple neighborhood sizes to capture both fine details and overall shape, for example 3DSmoothNet [33] and GeDi [20]. Fully Convolutional Networks (FCN) backbones, such as FCGF [5] produce dense per-point descriptors efficiently. Joint detect-and-describe networks improve repeatability, as shown by D3Feat [13]. Rotation-aware kernels strengthen robustness to orientation changes, as in the Convolutional Neural Network (CNN) based SpinNet [34]. Additionally, approaches focused on transformers have recently gained attention, such as GeoTransformer [14], where the point matcher incorporates long-range geometric priors to improve the correspondence stage.

Among these, we focus on FCGF due to its strong balance of accuracy and efficiency. It yields dense, high-quality descriptors in a single forward pass using sparse 3D convolutions, scales well on large point clouds, and should integrate smoothly with robust transformation estimators under tight latency constraints [5].

As the name suggests, FCGF is based on a FCN architecture. An FCN is a CNN that replaces the usual fully connected layers at the end with only convolutional (and the regular required normalization and activation) blocks. This design enables the network to accept inputs with variable sizes (while CNN demands fixed sizes) and produce dense, spatial outputs with the same spatial structure of the input (since the fully connected layers that would typically flatten the output were removed). In 2D, for example, a standard classification CNN predicts one label for the whole image, estimating what is being shown, whereas an FCN can produce a label at each pixel, estimating what is being show in each part of the image, producing a segmentation of the picture [35]. In 3D, this means we can voxelize a point cloud and obtain a descriptor at every occupied voxel, rather than a single global embedding (which would be of little use for PCR). FCNs are therefore a natural fit for local feature extraction, since they share computation across the

entire scene in one pass and return a dense descriptor field that can be mapped back to the original points.

However, a traditional FCN with dense 3D convolutions must operate over all voxels, and each 3D convolution has a cubic cost with the grid resolution. Doing this for every voxel quickly becomes infeasible. FCGF keeps the fully convolutional design but implements it with sparse 3D CNNs, using Submanifold Sparse Convolutions and Minkowski Convolutions, that compute only on occupied voxels and their immediate neighborhoods. This retains the benefits of FCNs while dramatically reducing compute and memory [36, 37]. FCGF then learns descriptors with a contrastive objective, so that true correspondences are left close together in the feature space and non-matches are far apart, enabling robust matching downstream[5].

Given an input point cloud and a chosen voxel size (set according to sensor noise and scene scale), the first step in FCGF is voxelization. To do this, the 3D space is partitioned into a regular grid with that constant resolution (set by the voxel size), and each point is quantized to its voxel index (i.e., mapped to the nearest voxel center). Points landing in the same voxel are aggregated into a single representation of them, or a per-voxel feature, which encodes the information of all those points into a single entry in the grid. These per-voxel features can be kept minimal, often just a constant or a simple statistic, because in sparse CNNs, the geometry is encoded by the voxel coordinates themselves (the set of active voxel indices and their adjacency), so even very simple features suffice [36, 37]. Empty voxels are then discarded, yielding a sparse 3D grid represented as coordinate-feature tuples, which serves as the network's input tensor [5].

Next, FCGF feeds the sparse grid into an encoder-decoder, UNet structure, backbone, as shown in Figure 2.2. A receptive field is the specific area of an input, that influences a particular feature. It is known that its size determines the complexity level of what a feature can represent, small fields capture local basic shapes (low-level features, such as edges, curvature and corners), while larger fields capture broader and more meaningful shapes (high-level features). Because of this, FCGF adopts this UNet structure, where the encoder progressively downsamples the input through strided sparse convolutions, reducing the resolution, expanding the receptive field and, consequently, gathering wider context, while the decoder uses transposed sparse convolutions to upsamples it back, restoring the original resolution, allowing the obtainment of descriptors for every occupied voxel. Besides that, the skip connections fuse fine detail (low-level features) from early layers with coarse semantics (high-level features) from deeper ones [5, 37, 38]. To keep this efficient in 3D, layers use submanifold sparse convolutions, which compute outputs only at already active voxel coordinates, preventing the grid from densifying after each convolution [36].

The final layer outputs a low-dimensional descriptor at each active voxel, and

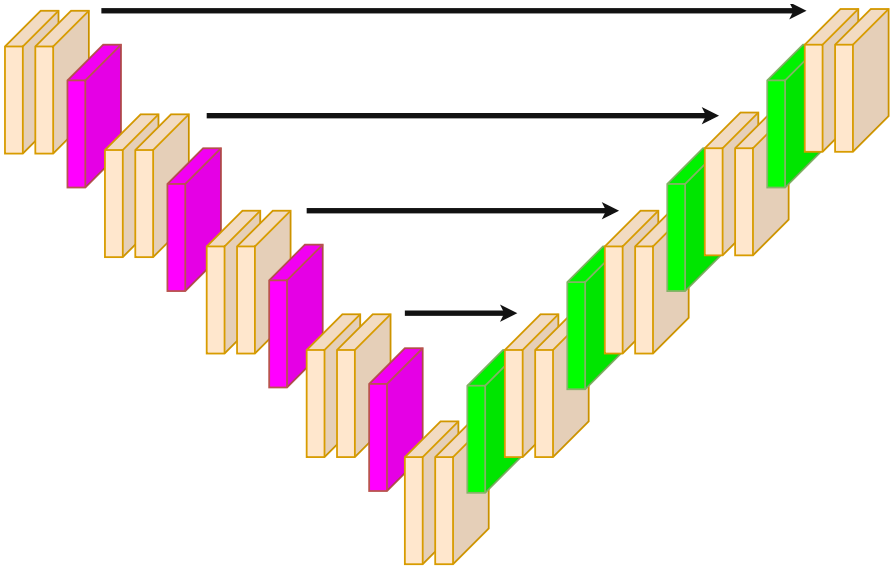


Figure 2.2: Typical UNet architecture illustrating the encoder–decoder structure with skip connections. The orange blocks represent convolutional layers followed by ReLU activations, the pink blocks denote max-pooling layers, and the green blocks correspond to transposed convolutions used for upsampling. The black arrows indicate skip connections linking encoder and decoder feature maps.

each vector is normalized on L_2 so that cosine similarity and Euclidean distance are consistent. Then, using the same quantization map from the voxelization process (initial step), every input point inherits the descriptor of its voxel (or an average if multiple points share the voxel), producing a dense per-point descriptors set ready for the following matching stages [5].

Regarding how FCGF's network is trained to extract feature descriptors from point clouds, a metric learning strategy is adopted, where a contrastive loss is used during training. In metric learning, the goal is learn how an embedding (i.e., a representation system) according to specific task metric, which in this case is a contrastive loss. This encourages FCGF to learn an embedding where true matching points (positives) lie close together in the feature space, while non-matching, dissimilar, points (negatives) are far a part from each other (beyond a certain margin). To enable this, labeled training data were constructed from fragment pairs with known relative poses (e.g., from the 3DMatch benchmark [32]). Then, using the ground truth transformation to align the point cloud pair, voxels from both clouds were compared: those whose centers present an Euclidean distance between them that falls within a small radius were labeled as positives, while all other pairs were set as negatives [5, 32].

Additionally to the metric learning, hard-negative mining is also employed,

which means that among available negatives in a mini-batch, the most confusable (i.e., closest) ones are emphasized so the network learns to push them apart. By focusing on the most difficult cases, it improves descriptor discriminativeness and training stability [5, 39, 40]. In practice, mini-batches include multiple fragment pairs from diverse scenes, and standard 3D augmentations (e.g., random rotations, translations, and noise/jitter) promote robustness. This supervision setup encourages invariance to sampling density, moderate viewpoint changes, and sensor noise, producing descriptors that are reliable for nearest-neighbor matching in downstream registration steps [5, 32].

2.2 Global Registration

In the last section, we focused on feature extractors, which, as mentioned, are vital for the downstream registration stage. Additionally, as already hinted, most advanced pipelines usually adopt a two-stage mechanism: obtaining an initial alignment followed by a refinement. In this section, we focus on the former, and in the next section we will discuss the latter.

Global registration estimates the alignment between two point clouds without any good initialization, aiming to handle large initial misalignment, partial overlaps, and high outlier rates. It serves as a coarse alignment stage: given a correspondence set (often from matching the descriptors obtained by the feature extractors reviewed earlier), a global method returns a robust initial pose that is later refined by a local solver [2, 3, 17, 18].

The obtained registration is given by a rigid transformation matrix, as in (2.4), although typically the scaling factor is neglected, and only rotations and translations are considered.

$$T_a^b = \begin{bmatrix} sR_a^b & t_a^b \\ 0^T & 1 \end{bmatrix} \tag{2.4}$$

Historically, the work on this topic progressed along a few recurring ideas. First, the were methods based on hypotheses generation followed by validations routines (called hypothesize-and-test methods), which try to maximize the consensus of correspondences over a certain hypothesis (candidate transformation). The canonical example is RANSAC, and its specific variants towards PCR, which remain a strong baseline for transformation estimation robust to outliers [1].

Along similar lines, geometric congruent set methods generate more informed hypotheses by leveraging approximately congruent set of correspondences. Here, a hypothesis is derived from a small set whose geometric configuration is almost congruent across clouds (e.g., 4 coplanar points). By avoiding exhaustive matching, it provides speed but can be sensitive to noise in geometric invariants and to scenes

lacking distinctive structure. Notable examples include 4-Points Congruent Sets (4PCS) [41] and Super4PCS [42].

Another branch of methods formulates the registration task as a global optimization problem over the full 6 Degrees of Freedom (DoF) transformation space (including rotations and translations), often solved with Branch-and-Bound (BNB), such as Go-ICP [16]. By defining this 6-DoF space that contains all possible combinations of rotations and translations, and defining an objective metric, such as point-to-point Euclidean distance between the transformed source and the target, the goal of optimizing this metric represents obtaining a transformation that better aligns the cloud pair. Additionally, the nested BNB allows it to efficiently explore the SE(3) search space, maintaining tight upper and lower bounds over regions to prune regions that cannot contain better solutions (suboptimal regions) [16]. Another common example is Globally-Optimal Gaussian Mixture Alignment (GOGMA) [43], which instead of considering the individual points themselves, it aligns Gaussian Mixture Models (GMMs) of the clouds (i.e., a probabilistic model that approximates how the points are distributed within the cloud), leading to smooth, differentiable objective while retaining global optimality guaranteed via BNB [43]. These methods can provide deterministic optimality but are computationally heavier. Regarding these methods, their the major advantage is the deterministic guarantee of global optimality, when possible. However, they are much computationally heavier, due to the expensive 6-DoF optimization.

Viewing registration as robust optimization over correspondences leads to methods such as Fast Global Registration (FGR) [19], which formulates PCR as a robust least-squares (LS) over correspondences with a nonconvex penalty function (Geman-McClure penalty) that down-weights outliers, and solves it via Graduated Non-Convexity (GNC) for speed and robustness [19].

To facilitate the optimization approach, TEASER [44] uses convex relaxations and decoupled subproblems to achieve guaranteed robustness (certifiable global solution) without exhaustive search. It first decouples scale, rotation and translations into individual subproblems. Then, it leverages Truncated Least Squares (TLS), where residuals beyond a certain threshold stop contributing. Finally, it solves each subproblem individually through Semi-Definite Programming (SDP). It is slower than approximate methods, and requires good noise models for tight relaxation, but provides guaranteed solutions even under extreme outlier rates [44].

More recently, approaches based on graph theory gained attention, especially Maximal Cliques (MAC) [25]. It models the correspondences as nodes in a graph and connect those nodes that represent mutually consistent correspondences under certain geometric constraints (e.g. if the relative distance or angle between source points matches that of the target points, it makes these correspondences compatible under a rigid transform). Then, it searches for the maximal cliques (i.e., a consistent subset of connected nodes that cannot be increased by adding any other node from

the graph) and, for each one, it generates a transformation hypothesis through Singular Value Decomposition (SVD), before evaluating to select the best-scoring one [25, 6].

Among these approaches, we now take a closer look at RANSAC, which is the method we will later adopt for our global registration stage. We first recap the core idea of consensus maximization from minimal samples as introduced by [1], then describe the adaptations that make RANSAC effective for PCR.

RANSAC is a general iterative algorithm which can be applied to several different contexts, always with the goal of estimating the parameters of a mathematical model from data, being especially efficient when outliers are present. The overall intuition is simple and can be split into few steps. First, randomly select the minimum amount of data points needed to fit the model (e.g., two points for a line or three points for a circle). Notice however, that it can take more than the exact minimum, but it always selects a tiny subset of all available data. Then, fit a candidate model to this random minimal subset and evaluate how many of the remaining points that were not sampled can also fit this model within a certain threshold (e.g. how many extra points are close enough to the estimated line or circle). These points are called inliers, and if their number is not high enough, RANSAC keeps repeating this process. Otherwise, if the number of inliers suffices, we select all the inliers and re-estimate a final model taking into account all of these points. The iteration continues until we reach the convergence criteria which are either: a good enough model is found (i.e., some candidate obtained the minimum number of inliers needed), or it reaches a maximum iteration limit [1].

Now, focusing on our task (PCR), RANSAC can be used for global registration. In this scenario, our "data points" become the point pairs (correspondences) between source and target. One could try to estimate what points are a match simply by their geometric distance (i.e. which point in the target is closest to each point in the source). However, this is neither reliable nor efficient, given how dense the clouds are. Thus, a clever and more robust approach is the feature-based RANSAC, where instead of looking simply at the raw spatial coordinates of the points, it also leverages the information about their geometric features [5, 14, 17].

Given the feature sets for the source P and target Q clouds (previously obtained by the feature extractor) we begin the algorithm by using them to build the tentative correspondence set $\mathcal{C}$ (2.5), where p_i is a source point and q_i is its match in the target.

$$\mathcal{C} = \{(p_1, q_1), (p_2, q_2), \dots, (p_n, q_n)\} \tag{2.5}$$

Hence, in this feature-based approach, we build this set by, for each source point, pairing it to the target point which is closest in the feature space (usually through nearest-neighbor search) to obtain a tentative correspondence (it is not possible to affirm whether they are a true match or not). Since we are dealing with the feature

space, and not the Euclidean space, the match will be formed by the similarity of the features of each point, not by their proximity. With the correspondence set ready, we sample n random pairs from it. In 3D, a rigid transformation has 6-DoF (3 for rotations and 3 for translations), so we need at minimum 3 pairs (3 points in the source and 3 in the target) to compute the transformation, if they are not collinear. Then, from these pairs, a candidate rigid transformation is computed. There are multiple closed-form solvers able to do this, and a very common one is the Kabsch-Umeyama method [6, 45]. First, compute the centroids of each cloud and subtract their points by their centroids, effectively centering the cloud around the origin.

$$\begin{aligned} \bar{p} &= \tfrac{1}{n}\sum_{i=1}^{n} p_i \qquad & \bar{q} &= \tfrac{1}{n}\sum_{i=1}^{n} q_i \\ \tilde{p_i} &= p_i - \bar{p} \qquad & \tilde{q_i} &= q_i - \bar{q}. \end{aligned} \tag{2.6}$$

Form the covariance matrix between both centered clouds.

$$H = \sum_{i=1}^{n} \tilde{p_i}\, \tilde{q_i}^{\top} \tag{2.7}$$

Compute its SVD.

$$H = U\,\Sigma\,V^{\top} \tag{2.8}$$

Use this to compute the optimal Rotation Matrix R^* and check if it is a proper rotation (no reflection).

$$R^* = V \begin{bmatrix} I_2 & 0 \\ 0 & \operatorname{sign}\Big(\det(VU^{\top})\Big) \end{bmatrix} U^{\top} \tag{2.9}$$

Apply the obtained rotation in the raw source and compute the distance between the centroids of the raw target and the rotated source to obtain the translation.

$$t^* = \bar{q} - R^*\bar{p} \tag{2.10}$$

Finally, assemble the obtained rotation and translation into the rigid transformation matrix, as in (2.4).

According to the general RANSAC loop, the next step is to evaluate the hypothesis against all points. Doing so requires applying the candidate transform to the entire source cloud and counting inliers, which can be expensive when many hypotheses are tested. To reduce overhead, practical pipelines insert an intermediate step: apply the transform only to the sampled source points used to compute it, then run a set of cheap checkers (filters) to decide whether the hypothesis is worth scoring on the full set. If any checker fails, the hypothesis is discarded.

Common checks include (i) an edge length (pairwise distance) consistency test, which verifies that two correspondence pairs preserve inter-point distances within a tolerance (ratio near 1), and (ii) a distance test, which verifies that, after applying the hypothesis to the sampled source points, enough mapped points fall within a positional threshold of their targets [5, 7]. This process repeats until a satisfactory inlier count is achieved or a maximum iteration budget (derived from a desired confidence) is reached, at which point the best-scoring transform is returned [1].

2.3 Local Registration

After a coarse global alignment, local registration refines the pose by minimizing a geometric objective under the assumption of a good initialization and sufficient overlap. In modern pipelines, this stage tightens the alignment from the robust but coarse estimate delivered by global methods, typically converging via Gauss-Newton iterations on small residuals [17].

The canonical example for this class is ICP, where starting from an initial pose, it alternates between correspondence update (e.g., nearest neighbors) and pose update to reduce a chosen error metric. The original formulation minimized point-to-point distances [2], and the widely used point-to-plane variant improved convergence speed and stability by projecting errors onto target normals [3]. Systematic analyses and speed-ups (sampling, correspondence search, rejection, and linearization choices) were surveyed and benchmarked by [46], establishing many practical variants that remain standard today (e.g., normal-space sampling, and multi-resolution pyramids). Beyond pure geometry, Colored ICP augments the objective with a photometric term to better constrain tangential drift on low-texture surfaces, improving convergence on RGB-D data [47].

Alternative local matchers adopt different residual models. The NDT replaces NN search with registration against a grid of Gaussian cells, yielding smooth objectives and robust behavior for LiDAR scan matching [10]. Subsequent 3D variants quantify distances directly between NDT models to further stabilize and accelerate alignment [48]. CPD casts alignment as probabilistic density fitting with a GMM and Expectation-Maximization (EM) optimization (rigid and non-rigid), offering effective handling of noise, outliers, and missing data [11]. Generalized ICP (GICP) unifies point-to-point and plane-to-plane by modeling local surface covariances, producing a single least-squares that adapts across scene structure [18]. On RGB-D streams, projective ICP and multiscale strategies enabled real-time dense tracking and mapping (e.g., KinectFusion) by aligning each frame to a predicted surface model [49]. Robust variants such as Trimmed ICP mitigate outliers and partial overlap by optimizing over a subset of correspondences with smallest residuals, improving convergence in difficult scenes [50].

In what follows, we deep-dive into the classical ICP and some of its most notable variations. Although ICP is sensitive to local minima in cases of poor initializations and may require multiple iterations, it remains an established and intuitive method with strong practical performance, capable of tightening coarse global alignments. Additionally, numerous variants have been proposed to mitigate its limitations [2, 3, 46].

Although ICP was originally introduced for point-set (shape) registration (i.e., PCR), its iterative behavior of alternating between recomputing correspondences and then solving a least-squares pose, has inspired related geometric optimization schemes in other domains. The standard ICP framework, however, remains specific to registration [17, 2, 3].

Once again, let us denote p_i as points in the source P, q_i as points in the target Q, and $T_0 = (R_0, t_0)$ as the initial transformation (obtained by global registration) that roughly aligns P to Q such that $Q \approx \{R_0 p_i + t_0 \mid p_i \in P\}$. ICP starts its first iteration ($k = 1$) by applying this initial transformation to the original source:

$$P_0 = \{R_0 p_i + t_0 \mid p_i \in P\} \tag{2.11}$$

Equivalently, with $\tilde{P}$ representing P in homogeneous coordinates:

$$\tilde{P}_0 = T_0 \tilde{P}, \qquad \tilde{P} = \begin{bmatrix} P \\ 1^\top \end{bmatrix} \tag{2.12}$$

The next step is establishing correspondences between the current source P_0 and the target Q. The standard ICP, also called point-to-point ICP, simply evaluates the Euclidean distance between pairs, matching, for each source point, the nearest target point. There are many alternatives proposals to this matching strategy, which we will discuss in the sequence. However, the main downside of the classical approach is its computational cost, since computing all the pairwise distances gets expensive very quickly when dealing with dense clouds. For instance, considering that both source and target have roughly the same size, we can say N represents the number of points in them, so the matching complexity would be squared proportionally to the clouds size: $\mathcal{O}(N^2)$. Additionally, this would be repeated for each iteration k, producing a final complexity of $\mathcal{O}(kN^2)$. But, in practice we have that $N \gg k$, so we can reduce the cost back to $\mathcal{O}(N^2)$ [3, 46].

To overcome this overhead, the most common approach is to adopt a KD-tree (Figure 2.3), enabling efficient nearest-neighbor search by reducing the search space [51].

This works similarly to a regular binary tree, but instead of a singular value per node, we have K coordinates, and the tree has depth of D. The KD-tree is built on the target cloud Q, which remains stationary through all ICP iterations, meaning that the tree can be build just once and reused for all following iterations. In 3D,

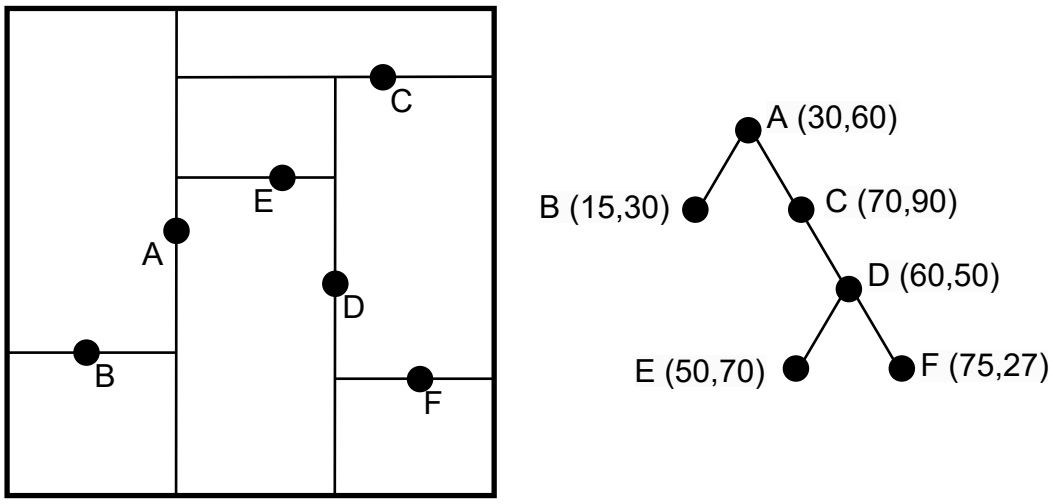


Figure 2.3: Typical KD-tree structure and its space division procedure visualized on 2D. Notice that the space on the left is divided by descending through the tree and, at each level, alternating the axis in which the split plane is drawn.

we have $K = 3$ for the three axes. At each node, we must choose the splitting axis (i.e., the direction of the splitting plane) and its order. This can be done by either cycling through the axes ($e.g., x \to y \to z$) or, more elegantly, selecting at each node the axis with the largest variance in that node's subset (directions along which the cloud is more spread come first, while more concentrated directions come later). We then split along that axis at the median point (i.e., split the space at the point that keeps its roughly balanced in terms of number of points on each side). When splitting the space, we store the median coordinate (split value) and the chosen axis at the node (each node thus stores the median point's coordinate on the split axis and the axis index used to divide the space). Each subset is split into left and right parts, and we keep recursively partitioning until a stopping criterion is met: either we reach the desired subset size (i.e., the number of points at a leaf node is below a certain threshold, usually between 8 and 32 points) or the maximum depth. This building process has complexity $\mathcal{O}(N \log N)$ and is performed only once [46, 51].

Then, with the KD-tree ready, we query it for each p_i to find its closest q_i. Starting at the root, we compare p_i coordinates to the node's split value along that node's split axis (as defined earlier). If p_i is smaller, we descend to the left child, otherwise, to the right. At each level, the split axis follows the same order/selection rule used during construction. Upon reaching a leaf node, we compute the exact distances between p_i and the few points in the remaining subset, keeping the current best one. As with other tree searches, we can also backtrack by ascending one level

to the immediately previous node and exploring the sibling branch we skipped. We do this only if the distance from p_i to the parent node's splitting plane is smaller than the best distance so far (i.e., if the search sphere intersects the other side). Considering a balanced tree this search process has expected $\mathcal{O}(\log N)$ time in low dimensions like 3D, with worst case of $\mathcal{O}(N)$. This search is performed N times (once for each p_i in P) for each iteration k: $\mathcal{O}(kN \log N)$. As mentioned before, in practice we usually have $N \gg k$, so the overall search cost across all iterations can be approximated as $\mathcal{O}(N \log N)$ [46, 51]. Thus, the final complexity of the KD-tree approach can be estimate as:

$$\begin{aligned} \mathcal{O}_{\text{total}} &= \mathcal{O}_{\text{build}} + \mathcal{O}_{\text{search}} \\ &= \mathcal{O}(N \log N) + \mathcal{O}(N \log N) \\ &= \mathcal{O}(N \log N) \end{aligned} \tag{2.13}$$

Therefore, when comparing the final time of the KD-tree approach $\mathcal{O}(NlogN)$ with that of exhaustive search $\mathcal{O}(N^2)$, the benefit of adopting this structure in terms of reduced cost becomes evident[46, 51].

Returning to ICP, once the NN search is complete and the correspondence set for iteration k has been obtained, we estimate the updated rotation R_k through SVD of the covariance matrix of the demeaned clouds, as in (2.9), and the translation t_k by, as in (2.9), computing the distances between $\bar{p}_{k-1}$ (i.e., the centroid of the rotate current source $R_k P_{k-1}$) and $\bar{q}$ (centroid of Q) [6, 45]. Assembling these, yield the transformation T_k, which ICP uses to proceed to the next iteration:

$$P_{k+1} = \{R_k p_i + t_k \mid p_i \in P_k\} \tag{2.14}$$

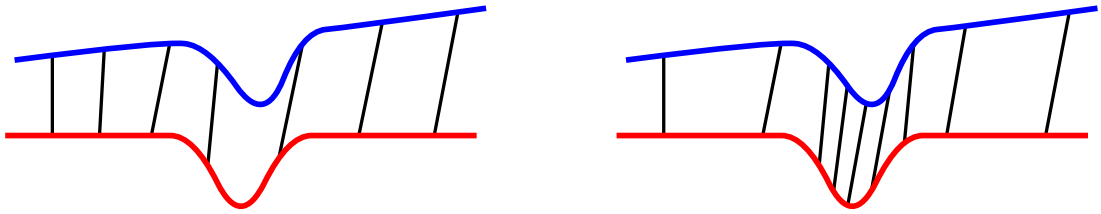
Or, once again in homogeneous coordinates:

$$\tilde{P}_{k+1} = T_k \tilde{P}_k, \qquad \tilde{P}_k = \begin{bmatrix} P_k \\ 1^\top \end{bmatrix} \tag{2.15}$$

There are several reasonable convergence criteria, with common choices being to continue iterating while the error (e.g., the Root Mean Square Error, RMSE, over correspondences) decreases significantly and remains above a threshold, or until the pose update is sufficiently small, or a maximum iteration limit is reached [2, 46].

As hinted earlier, many alternatives have been proposed to the standard ICP, and they typically target a specific stage of the loop. A common focus is the correspondence sampling strategy at each iteration. Even with the KD-tree acceleration we previously presented, attempting to match every point can be expensive, so practical variants adopt sub-sampling strategies, such as uniform sub-sampling, random sub-sampling, or feature-based sub-sampling that favors points in distinctive regions (corners, edges, et.c). Another interesting strategy that is also widely used is normal-space sampling, which selects points so that their surface normals

are well distributed on the unit sphere (i.e., avoiding an over-representation of large planar areas). This yields a set of samples that are more concentrate around regions where normals are changing considerably, and sparser in flat regions, which make it fit for scenes with dominant featureless surfaces (Figure 2.4) [46].



(a) Uniform Sampling **(b)** Normal-Space Sampling

Figure 2.4: Visualization of different sampling strategies used during ICP iterations (black lines). (a) Uniform sub-sampling selects points evenly in space. (b) Normal-space sampling selects points to balance surface-normal directions, reducing bias from large planar regions.

Another branch of alternatives changes the correspondence rule itself. While standard ICP uses nearest neighbors in Euclidean space, many implementations add compatibility filters before accepting a match. For examples, checking on normal angles, distances, curvatures, or even color and feature similarities to reject closest points that are geometrically or photometrically inconsistent [46, 47]. A different policy is normal shooting: from a source point, cast a ray along its surface normal and choose the first target intersection (or the nearest point to that ray). This can improve convergence on smooth surfaces where the closest point often lies along the normal direction, though it may behave worse on complex, feature-rich geometry [46].

Moreover, we have the renowned point-to-plane ICP, a notable variation that changes the error metric rather than the data-association logic (i.e., the correspondence matching strategy). In this version, correspondences are still usually formed by NN search, but the residual to be minimized, becomes the distance from the given source point to the tangent plane at its matched target point, as shown in Figure 2.5. By projecting the error along the target normal direction, it ignores the tangential component of the distance. Here the tangent plane acts as a first-order approximation of the local surface: around the target point, the (generally curved) surface is linearized by its tangent plane (whose normal is the surface normal), capturing the surface's behavior to first-order and ignoring higher-order curvature. By discarding tangential components and optimizing this linearized residual, point-to-plane often converges faster and more stably than point-to-point ICP on smooth scenes, especially under small pose errors [3, 46]. Additionally,

instead of the straightforward SVD approach presented for the standard ICP, the point-to-plane alternative often adopts a least squares method to minimize this new residual solved through iterative Gauss-Newton minimization. When normals are noisy or at sharp edges, robust kernels or hybrid formulations such as GICP can be used to further improve stability [18].

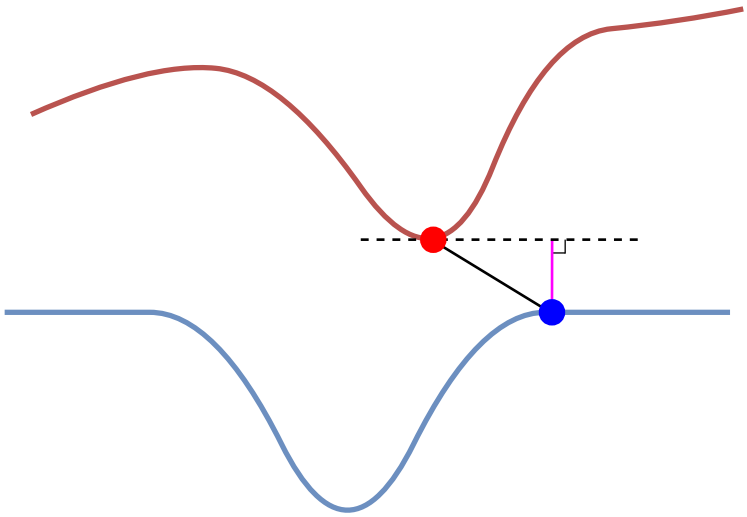


Figure 2.5: Illustration of the error metric. The blue dot is a source point and the red one its matched target. The black segment depicts the point-to-point residual, while the pink segment shows the point-to-plane residual, aligned with the target normal (perpendicular to the target's tangent plane).

Finally, another important line of ICP variants focuses on rejecting, or down-weighting, outliers before solving the pose. Common filters impose hard thresholds on correspondence distance or normal-angle compatibility, while others enforce local consistency, e.g., neighboring source points should map to neighboring target points, or pairwise distances should be preserved. In all these cases, the inconsistent pairs that fail these verifications/filters are pruned [46]. A well-known method is Trimmed ICP (TrICP), which sorts correspondences by residual (e.g., Euclidean error for point-to-point or the normal-projected error for point-to-plane) and minimizes a trimmed objective over only the best $\tau\%$ of pairs. The trimming fraction τ reflects expected overlap and can be fixed or selected by line search, yielding robustness to gross outliers and large unmatched regions [50].

2.4 Registration Benchmarks and Evaluation

A plethora of benchmarks exist for evaluating PCR. Generally, they can be grouped by scale: either object-scale, which centers on individual objects, or scene-scale, which covers larger environments. Additionally, scene-scale datasets are commonly further divided by context into indoor (e.g., bedrooms, kitchens, offices) and outdoor (e.g., buildings, streets, urban blocks) settings, reflecting where the scans were acquired and the sensing conditions they entail.

Regarding object-scale benchmarks, widely used options include the Stanford 3D Scanning Repository with laser-scanned meshes of canonical objects such as the Bunny, Dragon, and Armadillo, commonly used to stress-test geometric algorithms under clean mesh conditions [52]; ModelNet, an immense CAD collection that enables controlled perturbations for learning and evaluation [53]; and ShapeNet, an information-rich collection of CAD models with broad category coverage and annotations organized by a taxonomy (e.g., labels follow a hierarchy, such as furniture $\rightarrow$ chair $\rightarrow$ office chair, rather than a single flat label), again synthetic and thus free of real sensor artifacts [54].

Then, at scene-scale, common outdoor benchmarks include KITTI Odometry, which provides synchronized LiDAR (Light Detection And Ranging) and stereo sequences, captured by a vehicle-mounted setup, with Global Positioning System (GPS) and IMU (Inertial Measurement Unit) ground-truth trajectories, supporting scan-to-scan and scan-to-map (i.e. SLAM, Simultaneous Localization and Mapping) registration at driving scenarios [55]. For ICP-focused comparisons under real sensing conditions, the ETH laser-scanner datasets used by [56] remain a classical reference, covering diverse indoor, urban, and natural environments and enabling systematic evaluation of ICP variants. On the indoor side, RGB-D datasets such as TUM RGB-D [57], ICL-NUIM [58], and Redwood [59] are frequently used for SLAM and pairwise alignment in household scenes, with TUM offering motion-capture ground truth and challenging handheld trajectories, ICL-NUIM providing photorealistic synthetic sequences with a calibrated noise model, and Redwood supplying reconstruction sequences and a dedicated pairwise registration benchmark. Finally, the most notorious one is 3DMatch which remains the go-to benchmark for correspondence learning and pairwise registration.

The 3DMatch Geometric Registration Benchmark (Figure 2.6) aggregates indoor RGB-D reconstructions (e.g., bedrooms, offices, kitchens) into surface fragments and provides ground-truth relative poses between fragment pairs. The standard protocol selects pairs with at least 30% geometric overlap. Because the data are captured with commodity RGB-D sensors, the benchmark stresses robustness to depth noise, low-texture surfaces, occlusions, and partial views, while remaining broadly reproducible via public fragments, ground truth, and evaluation scripts [32]. Recent works also adopt community-curated splits and preprocessed fragments

(e.g., PREDATOR's release) to ensure comparability under identical voxelization and overlap mining [14, 26].

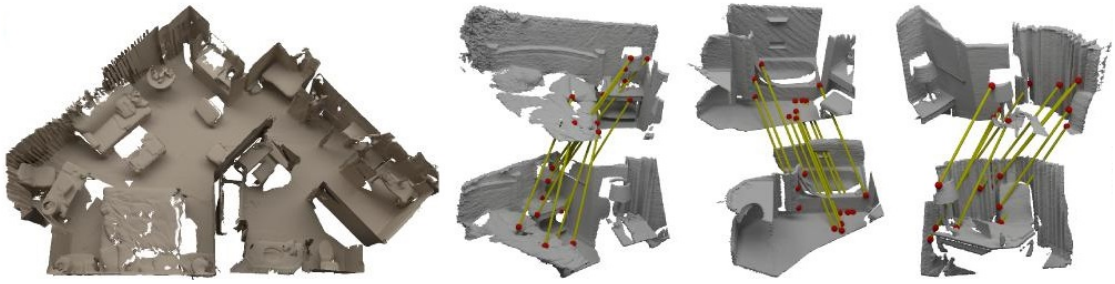


Figure 2.6: Visualization of the 3DMatch geometric registration benchmark. On the left, the whole reconstructed indoor scene, and on the right, examples of fragment pairs obtained from it. Red dots and yellow segments mark matched keypoints. Source: adapted from [32].

Beyond canonical PCR benchmarks, object grasping and manipulation datasets also provide synchronized RGB-D streams and object CAD models, enabling cross-domain registration (i.e., aligning ideal synthetic meshes to real sensor acquired scans). This is useful to explore domain gaps (mesh vs. depth scans), heavy occlusion, and cluttered tabletops. A representative choice is SuctionNet-1Billion, which builds on GraspNet-1Billion by reusing its multi-view RGB-D captures from commodity sensors (Intel RealSense D435 and Azure Kinect) and adding suction-specific supervision [9, 60]. The dataset covers 190 cluttered tabletop scenes populated by 88 everyday objects, each with an associated CAD model [60]. For every scene, the two RGB-D cameras traverse a fixed trajectory around the table, in a synchronized way, yielding 256 viewpoints per scene, providing substantial viewing angles diversity. Typically, the same set of objects is rearranged into several distinct scene configurations (about four scenes per set of objects), varying poses and placements while preserving object identities. In addition to RGB and depth images, label images are also provided (Figure 2.7), where each of its pixels stores the integer value related to the ID of the visible object (or 0 for background), which can be used to identify where each object is included in the scene, behaving similar to an image segmentation view. This also allows to isolate a single-object out of the whole cluttered tabletop view. This combination of CAD meshes, synchronized multi-view RGB-D, and dense instance labels makes SuctionNet an effective source for constructing object-scale, cross-domain PCR benchmarks under realistic clutter, occlusion, and sensor noise [9, 60].

Accompanying benchmark choice, evaluation metrics and protocols determine whether results are comparable and meaningful across papers. In PCR, three types of metrics are most common: correspondence-level scores, registration success

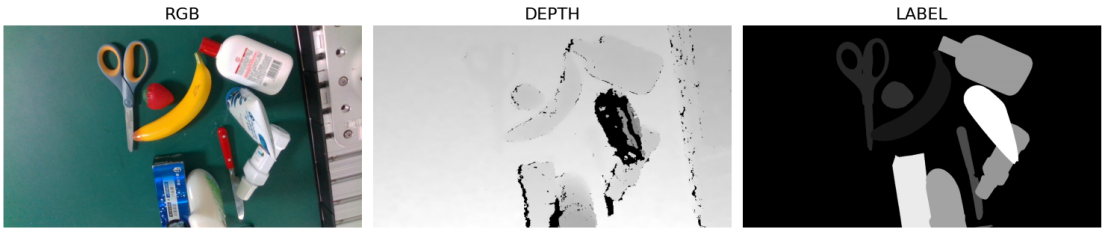


Figure 2.7: Example of a sample data triplet from SuctionNet. Left: standard RGB colored image. Center: depth image, where each pixel's value indicates its distance from the camera (for visualization, the image was plot in grayscale, where the brighter a pixel is, the higher is its distance; notice that the upright shampoo tube is darker because it is the closest object to the camera). Right: instance label image where each pixel store the value of an object ID or 0, if it is the background (once again, in grayscale, so brighter pixels indicate higher the IDs).

criteria, and efficiency measures. First, for the correspondence-level metrics, given the tentative correspondence set $\mathcal{C}$ of the source P and the target Q cloud, an estimated transformation T, and an inlier distance threshold τ, the inlier set is defined as follows:

$$\mathcal{I}(T, P, Q, \tau) = \{\, i \;\mid\; \|Rp_i + t - q_i\|_2 \leq \tau,\ (p_i, q_i) \in \mathcal{C} \,\} \tag{2.16}$$

Then, two residual metrics are widely reported: fitness and inlier RMSE. The fitness measures the proportion of correspondences that lie within a maximum distance after applying the estimated transformation to the source:

$$\text{Fitness} = \frac{|\mathcal{I}|}{|\mathcal{C}|} \tag{2.17}$$

And the inlier RMSE measures geometric tightness conditioned on inliers [7]:

$$\text{Inlier RMSE} = \sqrt{\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\|Rp_i + t - q_i\|_2^2} \tag{2.18}$$

Learned matchers additionally report Inlier Ratio (IR) per pair and Feature Matching Recall (FMR), the fraction of pairs whose IR exceeds a fixed threshold, to summarize how often a method produces a workable correspondence set for downstream pose estimation [14, 26].

Following this, the pose-level metrics and success protocols are applicable when ground-truth relative poses (R_{gt}, t_{gt}) are available (e.g., 3DMatch). In these cases, the canonical errors are RRE and RTE:

$$\begin{aligned} \mathrm{RRE}(R, R_{gt}) &= \arccos\left(\frac{\mathrm{Tr}(R^\top R_{gt} - 1)}{2}\right) \\ \mathrm{RTE}(t, t_{gt}) &= \|t - t_{gt}\|_2 \end{aligned} \tag{2.19}$$

And a registration is deemed successful if RRE $\leq \theta$ and RTE $\leq \epsilon$, where popular choices, such as on 3DMatch, are $\theta = 15°$ and $\epsilon = 0.3$ m. This enables the computation of Registration Recall (RR) as the fraction of successful pairs, and related success rates [14, 26, 32]. Some works also report accuracy curves (Area Under the Curve, AUC) by sweeping θ and ϵ. For sequence benchmarks (SLAM/odometry), frame-to-frame registration quality is commonly summarized by Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) over camera tracks [55, 57], but these target multi-frame pipelines rather than isolated pairwise PCR.

Finally, for the last family of metrics, for assessing efficiency and robustness, results typically report runtime and sometimes, when applicable, iteration counts to give quantify computational effort [7]. Protocol details (such as overlap filtering of $\geq 30\%$, voxel size, correspondence radius, and preprocessed fragments and splits) are typically fixed to community settings to ensure reproducibility and fair comparison [26, 32].

Chapter 3

Problem Formulation

Given two points clouds, a source P and a target Q (kept stationary), where p_i and q_i denote points in P and Q, respectively, point cloud registration seeks to obtain a rigid transformation $T = (R, t) \in \mathrm{SE}(3)$, as in (2.4), that better equalizes the relative pose between P and Q, aligning them such that

$$Q \approx \{Rp_i + t \mid p_i \in P\} \tag{3.1}$$

Or, in homogeneous form:

$$\tilde{Q} = T\tilde{P}, \qquad \tilde{P} = \begin{bmatrix} P \\ 1^\top \end{bmatrix}, \quad \tilde{Q} = \begin{bmatrix} Q \\ 1^\top \end{bmatrix} \tag{3.2}$$

Moreover, in this study, we adopt only scale-invariant benchmarks (introduced later in this section), focusing on purely rigid registration. Hence, the scaling component can be ignored, reducing (2.4) to:

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \qquad R \in \mathrm{SO}(3), t \in \mathbb{R}^3, T \in \mathrm{SE}(3) \tag{3.3}$$

In short, at a high level, the purpose of the registration stage in PCR pipelines is to estimate a transformation that aligns a pair of clouds. With this conceptual idea laid out, we can translate it to a more strict format by casting it as the goal of minimizing an adequate alignment loss (i.e., a metric that measures the discrepancy between the transformed source and the target). Let $\mathcal{L}(T, P, Q)$ denote such a loss as a function of a transformation and the cloud pair. Then, we can formulate the registration problem as

$$T^* = \arg\min_{T \in \mathrm{SE}(3)} \mathcal{L}(T, P, Q) \tag{3.4}$$

Notice that here we keep an abstract loss function $\mathcal{L}$, because each stage (global or local) has a particular loss function depending on its working principle. The specific choices used in our pipelines are detailed next.

Therefore, within the general PCR objective, this study evaluates a deep-learning–based pipeline against a classical geometric-only pipeline under consistent datasets, metrics, and runtime measurements, comparing alignment quality, latency, and robustness. Guided by prior work, as covered in the comprehensive overview in Section 2, we adopt a coarse-to-fine correspondence-based design: a feature extractor backbone to form tentative matches, a robust global estimator to obtain an initial pose, and a local optimizer to refine it. The decision to adopt a coarse-to-fine structure was motivated because it separates outlier handling from precision fitting, improving robustness under large misalignment and partial overlap while ensuring stable convergence in the small residual regime [17, 46]. Figure 3.1 showcases the concept behind the coarse-to-fine approach that both pipelines will follow.

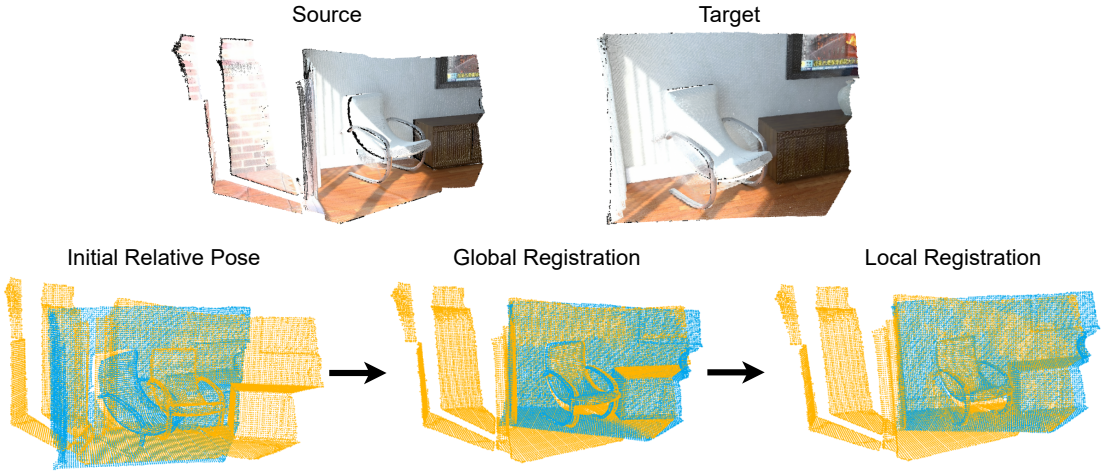


Figure 3.1: Conceptual overview of the coarse-to-fine approach adopted in our pipelines. The top row displays the source and target RGB-D scans of the same scene captured from different viewpoints, illustrating appearance and occlusion differences. The bottom row presents the stages of a coarse-to-fine registration. Initial relative pose: the point clouds are shown as acquired (target is blue and yellow is source). Global registration: a robust alignment brings the clouds into coarse agreement, where large structures align (e.g., the chairs), but noticeable gaps and residual errors remain. Local registration: an optimization stage refines the previous alignment producing the final registration, where both clouds are tightly matched.

We further opt for correspondence-based pipelines for their modularity and interpretability (explicit correspondence sets enable geometric sanity checks). Finally,

we choose keypoint-based extraction to meet latency and compute constraints: selecting salient, repeatable points reduces search and scoring cost without sacrificing reliability in structured scenes, while dense keypoint-free schemes typically demand heavier downstream filtering and higher runtime [17, 28].

Notably, both pipelines share the same registration stage, with feature-based RANSAC for global alignment and point-to-plane ICP for local refinement. They differ only in the feature extractor: a learned sparse-CNN descriptor (FCGF) for the deep pipeline versus a hand-crafted histogram descriptor (FPFH) for the geometric pipeline. We fix the global and local stages to isolate the effect of the feature extractor. RANSAC is selected for its strong tolerance to high outlier rates, and lightweight geometric checkers/filters that, as previously mentioned, are used to prune weak hypotheses before full scoring, reducing runtime while preserving robustness [5, 7]. For refinement, point-to-plane ICP typically converges faster and more stably than point-to-point in the small residual regime, which is exactly the setting produced by a good global initialization [3, 46]. The deep pipeline adopts FCGF for its dense, high-quality descriptors computed efficiently in a single pass by sparse 3D convolutions and contrastive training, yielding a strong trade-off between accuracy and latency [5]. Meanwhile, The geometric pipeline uses FPFH for its rotation-aware, analytic histograms that require no learning, scale well, and are widely adopted as a classical baseline [4, 17]. In summary:

- Deep-learning pipeline: FCGF [5], feature-based RANSAC [1] and point-to-plane ICP [3].
- Geometric-only pipeline: FPFH [4], feature-based RANSAC [1] and point-to-plane ICP [3].

Now, having introduced our overall pipeline architectures, we return to the abstract loss in (3.4) and specify it for each stage. First, for the global stage, RANSAC does not exactly minimize a smooth loss, but instead, it maximizes the size of the consensus set (i.e., the inlier set as defined in 2.16) given the inlier threshold τ, which is the maximum distance between a transformed source point and its tentative target correspondence for the pair to be counted as an inlier to the transformation. So, the canonical RANSAC objective is defined below [1].

$$T^* = \arg\max_{T \in \mathrm{SE}(3)} \mathcal{I}(T, P, Q, \tau) \tag{3.5}$$

Then, for the local stage, point-to-plane ICP follows a classical least squares formulation, where given the correspondence set $\mathcal{C}_k$ at iteration k and target normals n_i at q_i, it minimizes the sum of squared normal-projected residuals [3, 46]. Hence:

$$\mathcal{L}_{\mathrm{ICP}}(T, P, Q) = \sum_{(p_i, q_i) \in \mathcal{C}_k} \left(n_i^\top \cdot (R p_i + t - q_i) \right)^2 \tag{3.6}$$

For testing, we use two complementary benchmarks to assess robustness across scales. For scene-scale, we adopt 3DMatch, which offers indoor RGB-D fragments with ground-truth relative poses and a standardized protocol based on pairs with at least 30% overlap. This dataset captures real sensor noise, partial overlap, and viewpoint change and is widely used in registration studies [32]. To match community practice, we rely on the publicly released data preprocessed by PREDATOR for reproducibility [26]. As an object-scale complement, we repurpose SuctionNet-1Billion to form CAD-to-scan registration pairs under realistic clutter, occlusion, and noise. Although originally designed for grasp planning. its synchronized depth streams and available object meshes make it well suited for cross-domain registration (i.e., synthetic sources to real acquired targets) evaluation, under heavy occlusion, self-similar shapes, and realistic sensor noise. We use the Intel RealSense data in our experiments to mirror common industrial setups [9, 60].

For evaluation (see Section 2.4), we report fitness (2.17) and inlier RMSE (2.18), where fitness reflects the fraction of correspondences within a distance tolerance and inlier RMSE measures geometric tightness over those inliers [7]. Additionally, on 3DMatch, where ground-truth relative poses are available, we compute Registration Recall (RR) and Accuracy using the standard success thresholds [14, 26, 32]. On SuctionNet, where our cross-domain setup does not provide pairwise ground-truth transforms, we approximate recall by declaring a successful alignment when fitness ≥ 0.3, echoing the 30% overlap criterion from 3DMatch [32]. Finally, we measure runtime per stage (feature extraction, global registration, and local refinement) on a fixed hardware and also log iteration counts (i.e., RANSAC and ICP iterations to convergence). Iteration counts are useful due to their hardware-agnostic nature, which supports fair comparison even when absolute runtime may differ across machines.

Chapter 4

Methodology

This chapter details the design and implementation of the two point cloud registration pipelines presented in Section 3: a classical, geometry-only pipeline built around FPFH descriptors, and a deep-learning pipeline that employs FCGF as the feature backbone. Figure 4.1 summarizes the end-to-end flow shared by both pipelines: preprocessing (voxel downsampling and normal estimation), feature extraction (FPFH or FCGF), global alignment via feature-based RANSAC (with Kabsch-Umeyama SVD and lightweight hypothesis checkers), and local refinement with point-to-plane ICP. All modules were implemented on top of Open3D 0.19, with small compatibility updates to the public FCGF codebase. The complete source code developed in this project is available at [8].

Here, we focus on the implementation details and design choices of both the pipelines themselves and the experiments setups we used. We proceed as follows. Section 4.1 specifies the classical feature extractor and the preprocessing choices. Section 4.2 documents the FCGF integration and the minimal adaptations required for our benchmarks. Then, Section 4.3 describes the registration stage (RANSAC and ICP), including convergence criteria and scoring metrics. Section 4.4 introduces our instrumentation for iteration counts and per-stage timing. We then present the 3DMatch evaluation protocol and hyperparameters (Section 4.5), extend robustness testing with a controlled noise generator (Gaussian, spike, and salt-and-pepper) for an augmented 3DMatch benchmark, and finally construct a cross-domain object-scale benchmark from SuctionNet (Section 4.7), including an origin-centering step to improve numerical conditioning.

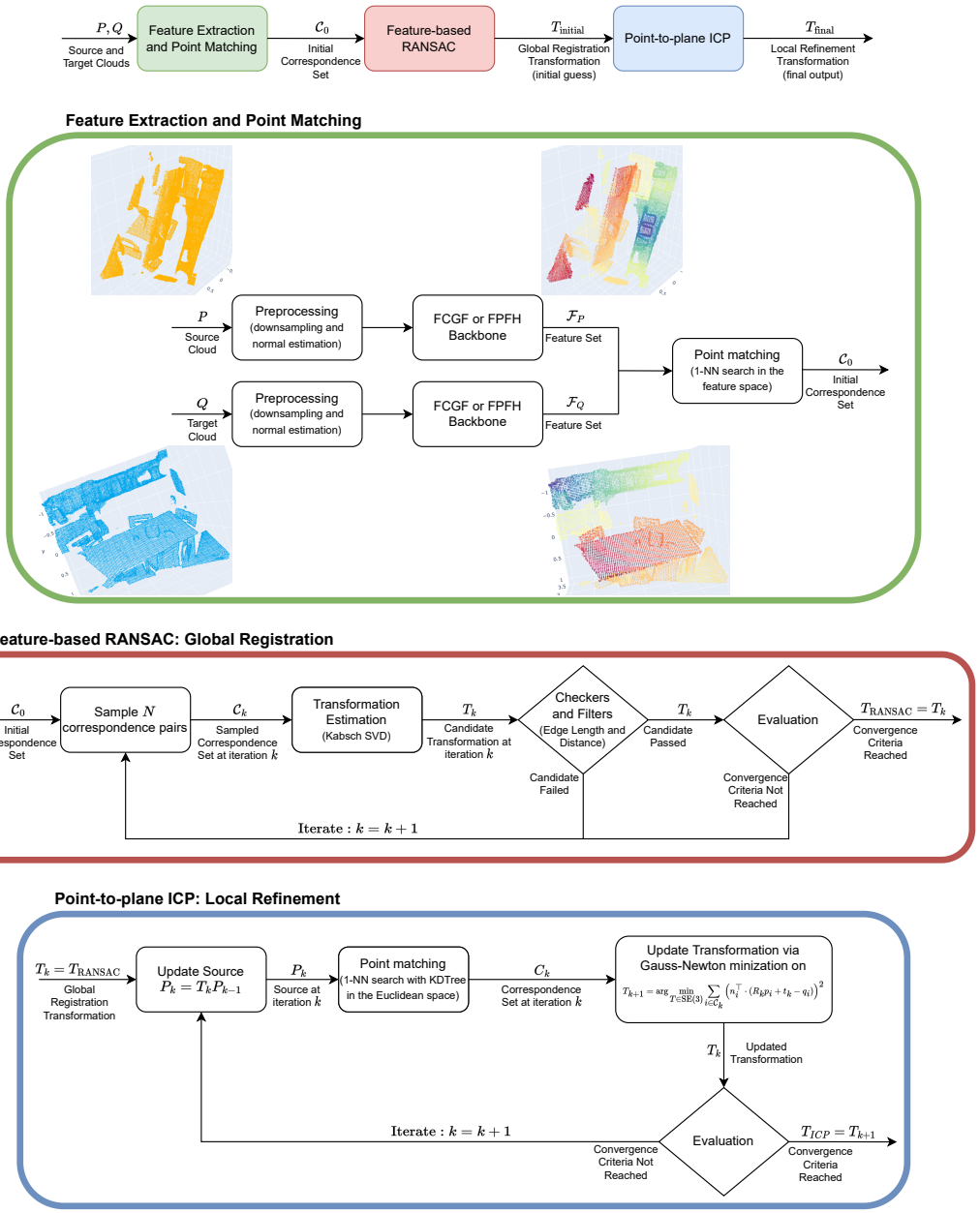


Figure 4.1: Overview of our registration pipelines. Clouds are preprocessed, then keypoint features (shown in color-coded clouds) are computed with FPFH (for geometric-only) or FCGF (for deep-learning). Tentative correspondences come from 1-NN search in feature space. Global registration uses feature-based RANSAC with Kabsch SVD and checkers. Its output initializes point-to-plane ICP, which alternates 1-NN matches and LS updates to refine the final transformation.

4.1 Classical Feature Extractor Stage

We begin with the geometric-only pipeline, whose first step is the feature extractor. As noted in Section 3, we adopt FPFH for the handcrafted pipeline. However, before extracting features, we must first define a short preprocessing chain, as illustrated in Figure 4.1, with two operations: downsampling and normal estimation.

Downsampling is useful because we are dealing with dense clouds, so reducing the amount of points accelerate subsequent computations such as feature extraction with FPFH, and it results redundancy caused by the abundance of points. This process is implemented through a voxelization approach parameterized by a given voxel size (resembling the first step of FCGF's review detailed in Section 2.1). Following the standard approach, the 3D space is partitioned into cubes of edge length equal to the voxel size. All points that fall in the same voxel (cube) are replaced by a single representative located at the centroid of those points [7], behaving similar to an encoding of all points. This yields a reduced sparser cloud which still preserves the geometry characteristics of the raw dense version. Figure 4.2 illustrates this idea in 2D for easier visualization. Note that only occupied voxels produce a representative point, while empty voxels are simply ignored.

This approach is preferred over random sampling (i.e., selecting n random points with n smaller than the cloud size) because it provides geometry-aware, uniform, and reproducible results, avoiding non-deterministic stochastic behavior. The voxel grid enforces a minimum spacing (approximately the voxel size) between retained points, preventing clusters in dense areas and holes in sparse ones. Random sampling preserves density bias and can miss thin structures. In practice, this yields more robust downstream performance [46, 56].

Then, as mentioned in Section 2.1, FPFH requires the surface normal at each keypoint in order to provide the feature descriptors. Hence, we estimate said normals on the downsampled clouds using a KD-tree with hybrid k-NN search [7]. The KD-tree allows rapid queries, as detailed in Section 2.3, and the hybrid search combines the efficiency of both radius and k-NN search. Thus, given a search radius r_n and a neighbor limit k_n, for each point x we gather neighbors within r_n, sorted by distance, and keep up to k_n of them (if available). Let the retrieved set have size $m \leq k_n$ with neighbors $x_{i\,i=1}^{m}$. Then, we compute the neighborhood centroid as:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{4.1}$$

With this, we can form the covariance matrix, which measures how the neighbors are spread in each direction around the centroid:

$$H = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu) (x_i - \mu)^{\top} \tag{4.2}$$

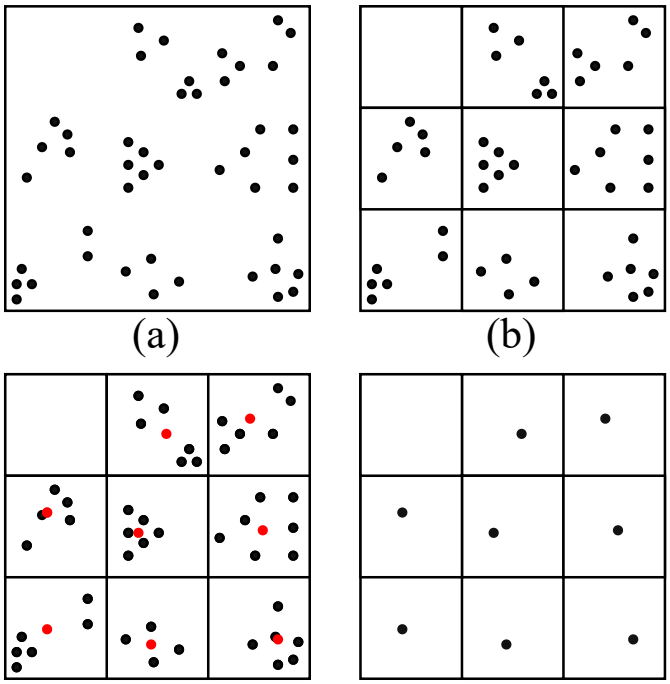


Figure 4.2: Visualization of the voxelization process in a 2D space. (a) Raw points distributed across the space. (b) Space partitioned into voxels segmenting the points into discrete locations. (c) The centroids (red) of each occupied voxel are computed. (d) Only the centroids are kept as voxel representatives.

And perform PCA by computing the eigenvalues and eigenvector of H:

$$Hv_j = \lambda_j v_j, \quad \lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0 \tag{4.3}$$

Where the eigenvector v_3 (associated with the smallest eigenvalue λ_3) aligns with the direction of minimal variance in the local neighborhood. Hence, v_3 is taken as the surface normal, because minimal variance along this axis implies the points are most planar (i.e., flat) in the plane orthogonal to v_3, which is consistent with the expected behavior of a normal vector [31].

Note that this PCA approach provides directions but not sign, as both positive and negative candidates might be true without knowing the global geometry (the normal orientation problem) [31, 7]. In our pipeline we do not explicitly orient normals, which is acceptable because the downstream stages are largely sign-agnostic: we do not apply a normal-angle checker in RANSAC, and point-to-plane ICP minimizes a squared normal projection that is invariant to sign.

Next, once the clouds have been properly preprocessed, we extract FPFH descriptors on the downsampled point set. The core FPFH logic was presented in

great details in Section 2.1, so here we present a high-level overview focusing on the particularities and choices regarding implementing it using Open3D. As mentioned, FPFH routine begins by, for each kept point, building its neighborhood set, which here is also done through a KD-tree hybrid k-NN approach: all neighbors within a user-specified feature radius r_f, capped by a maximum number of neighbors k_f. In the feature extraction context, this hybrid approach offers two practical benefits: (i) a radius anchors the descriptor to a consistent physical scale, and (ii) a cap on k_f prevents runtime blow-ups in dense regions [7]. Given those neighbors and their estimated normals, the algorithm computes the SPFH (Simplified PFH) at the center point. This means computing a histogram over the three Darboux-frame angular features (α, ϕ, θ) from 2.2, but considering only pairs between the current center point and each neighbor (neglecting pairs formed between only neighbors themselves). After iterating over all kep points and obtaining their SPFH, it iterates again and, for each point, it aggregates the SPFHs of its neighbors with distance-based weights to form the final FPFH at the center point [4].

Open3D returns a 33-dimensional (i.e., 11 bins per angular component), L_1-normalized histogram per kept point, aligned one-to-one with the downsampled cloud. These descriptors are translation-invariant by construction and approximately rotation-stable due to the use of local frames [4, 7]. In our pipeline, the feature radius r_f is set larger than the normal radius r_n to provide a wider geometric context for description. Thus, in summary, this stage produces a compact per-point feature descriptor matrix on the downsampled cloud, ready to feed the feature-space correspondence search in the global registration stage.

Given the different scales of our benchmarks (scene-scale vs. object-scale), we present concrete hyperparameters (e.g., voxel size, normal and feature radii, and neighbor caps) when discussing benchmark setup later in this chapter. The exact values and any dataset-specific adjustments are reported in Section 4.5 for 3DMatch, and Section 4.7 for SuctionNet.

4.2 Deep-learning Feature Extractor Stage

For the deep-learning pipeline, we implement FCGF as the feature-extraction backbone, as noted in section 3. As in the geometric-only case, we first implement the same preprocessing chain (Figure 4.1) detailed previously in Section 4.1, which includes: voxel downsampling and normal estimation on the downsampled cloud. While FCGF does not require normals, since its descriptors are learned directly from the voxelized coordinates, and adding normals did not improve results in the original work [5] we still estimate them here to support the downstream point-to-plane ICP stage (Section 2.3), which requires them for the normal-projected residuals. Regarding the downsampling process, it serves the same purpose as

before: reduce redundancy in dense scans and set a consistent spatial scale for both runtime and descriptor stability.

Then, following this preprocessing chain, we instantiate the FCGF feature extractor. We start from the authors' reference implementation [5, 61], but it no longer builds cleanly with current toolchains and dependencies used in our project. To address this, we forked the official repository [61] to our maintained copy [62], applying minimal changes needed for compatibility with our pipeline and benchmarks. For reproducibility and modularity, the fork is included in the main project as a Git submodule [8], allowing the registration pipeline and the descriptor backbone to be updated or pinned independently at specific commits.

The complete list of fixes can be inspected in the commit history of our fork [62]. Below are the highlights:

- Updated syntax calls for Open3D and scikit-learn to reflect recent releases (e.g., parameter renames and signature changes), ensuring compatibility with Open3D 0.19 and scikit-learn 1.7.

- Consolidated console output into structured logging to improve runtime progress monitoring.

- Removed `future fstrings` headers, which has become obsolete since Python 3.6.

- After voxelization, the coordinates are discretized to integer voxel indices. Previously, they were re-wrapped into a new integer tensor, which needlessly allocates fresh memory, hurting performance. This was adapted to, instead, convert the data type while keeping the data on the existing device (i.e., hardware location), avoiding extra copies. Overall, this reduces memory cost, explicitly indicates voxel coordinates as non-differentiable indexing data, and improves performance and stability with the sparse tensor backend (e.g., MinkowskiEngine) [37].

- Aligned feature output saving to our pipeline's folder layout (directory structure and filenames), so downstream stages can consume descriptors without additional adapters.

- Fixed the 3DMatch benchmark script to consider only non-consecutive pairs, following the standard evaluation protocol. Previously, consecutive pairs would be considered.

- Added a subset-evaluation flag to accelerate debugging: given a desired number of fragments/clouds N_{sub}, we randomly select N_{sub} items and form admissible

pairs following the 3DMatch protocol (non-consecutive and, when ground truth is available, meeting the overlap criterion), [7]. This yields up to

$$\binom{N_{\text{sub}}}{2} = \frac{N_{\text{sub}}!}{2!(N_{\text{sub}}-2)!} = \frac{N_{\text{sub}}(N_{\text{sub}}-1)}{2}$$

pairs samples to be tested. After that, we produce a text file listing the selected pairs and save the ground-truth transformations of said pairs in a log file. This allows both pipelines to run the exact subset test with the same samples, ensuring reproducibility and fair comparison, and the reduced ground-truth log simplifies the evaluation procedure.

Then, with the source code updated, we initialize the FCGF backbone model and load the authors' released pretrained weights. Specifically, we use the ResUNetBN2C configuration with base feature dimension 16 and three residual blocks per stage ("ResUNetBN2C-16feat-3conv"), which offers a good trade-off between speed and accuracy for large fragments. The model is moved to the selected device and set to evaluation mode, since we are not re-training it. Voxel quantization and sparse-tensor construction follow the reference implementation's conventions [5, 37, 61].

As with the geometric pipeline, all concrete hyperparameters are reported later alongside the benchmark setups in Sections 4.5 and 4.7. Because our FCGF codebase is an adaptation of the authors' reference implementation, this section records only our compatibility changes. For the network architecture, training procedure, and working principle explanation, see Section 2.1, where we presented a detailed literature review of FCGF [5]. And, for the technical implementation choices, refer to the original source code [61].

4.3 Registration Stage

The registration stage is the same for both pipelines and it comprises two sub-stages: a global alignment via feature-based RANSAC followed by a local refinement with point-to-plane ICP (Figure 4.1). Both theses algorithms have been explained during our literature review (Sections 2.2 and 2.3). Here we detail the implementation choices and their motivations.

For feature-based RANSAC, the inputs are the downsampled point sets and their descriptors: $\mathcal{F}_P$ for the source and $\mathcal{F}_Q$ for the target (FPFH or FCGF, depending on the pipeline). Before starting the RANSAC loop, we estimate the correspondence set $\mathcal{C}$ 2.5 by, for each kept source point, executing a 1-NN search to obtain the closest target point in the feature space, forming matches by their descriptor similarity and not simply by Euclidean distance. Notice that this correspondence

set is an estimate, as we still cannot be sure if these points are actually matches. To further improve our confidence is this correspondence set, we employ a mutual filter strategy, where a matching pair is only valid if the correspondence of a source point's target match is the source point itself (i.e., a pair is accepted only if it is each other's nearest neighbor). This reciprocal filter, prunes many weak correspondences before any of the other computations, improving runtime, and, by providing better correspondence sets, it improves convergence [7].

With $\mathcal{C}$ ready, RANSAC can begin its loop. First, it randomly samples pairs from $\mathcal{C}$ which will be used to estimate the transformation candidate. Thus, we must estimate a 6 DoF rigid transformation (3 rotation angles and 3 translation directions), and each pair provides 3 equations (one for each coordinate). So 2 pairs would already provide 6 equations, matching the 6 unknowns we must estimate. However, this does not uniquely constrain rotations in 3D, because it is still possible to rotate the pair around the axis connecting it without changing their position relative to each other. Hence, a minimum of 3 non-collinear pairs are required to uniquely determine a 3D rigid transformation. However, in our case, we opted to use 4 pairs to add extra stability by making the system overdetermined [6].

With this random sampled set, we compute a candidate transformation through the Kabsch-Umeyama closed-form solver detailed in Section 2.2 from Equations 2.6 to 2.10.

After obtaining the candidate transformation (the hypothesis), we implement an intermediate step before proceeding with evaluation. Here, we apply the hypothesis to only the 4 pairs we sampled from $\mathcal{C}$, and them invoke two cheap checkers to quickly discard weak hypothesis before scoring it against all correspondences (avoiding the cost of full scoring not promising candidates). The first one, a distance-based checker, evaluates if the pairs were aligned well enough by the hypothesis simply by measuring if the Euclidean distance between the transformed source points and matching target points are under the inlier threshold. The second one, an edge-length-based checker, takes pairs two by two and verifies if the edge (i.e. the distance) between the two source points e_{src} is similar to that of the two target point e_{tgt} [7]. This checker is controlled by a similarity threshold $s_{\text{th}} \in (0,1]$:

$$s_{\text{th}} \leq \frac{\|e_{\text{src}}\|}{\|e_{\text{tgt}}\|} \leq \frac{1}{s_{\text{th}}} \tag{4.4}$$

We ignored normal-based checkers to avoid the normal-orientation ambiguity problem discussed in Section 4.1.

If, and only if, a hypothesis pass both checkers, we fully evaluate it by applying it to all source points in the tentative correspondence set and scoring it with the inlier count (i.e., number of matching pairs that were aligned under an inlier threshold by the hypothesis). Once we have the inliers set (also called the consensus set), we also compute fitness and inlier RMSE as described in 3. RANSAC iterates this

loop until it reaches the convergence criteria.

We adopt two convergence criteria, where at least one of them must be met to terminate execution. The first one is a hard cap on iterations, useful to prevent RANSAC from executing indeterminately when no decent transformation can be obtained.

For the second criterion, recall that RANSAC repeatedly samples a small minimal set of correspondences of size m (here $m = 4$), estimates a candidate transformation from that set, and then counts how many of the remaining correspondences are consistent with it (inliers under a distance threshold). Let w denote the inlier fraction (or the fitness, as we called it earlier) in the current correspondence pool with respect to the best-so-far hypothesis (e.g., if 60 of 100 pairs are inliers for the current best transform, then $w = 0.6$). Given w, the number of iterations sufficient to achieve a target success probability c (i.e., the confidence) is

$$N_{\text{RANSAC}} = \frac{\log(1-c)}{\log(1-w^m)} \tag{4.5}$$

Here, c is the probability that at least one of the first N_{RANSAC} iterations samples an all-inlier minimal set (i.e., every element in that minimal set, sampled from the correspondence set, is consistent with the unknown ground-truth transformation under the inlier threshold). During the run, RANSAC maintains the best-so-far hypothesis and its inlier fraction w. Initially, while no decent hypothesis has been found yet (small w), the bound N_{RANSAC} remains large, allowing the algorithm to continue iterating (i.e., exploring) rather than stopping prematurely. Whenever a better hypothesis is found (larger w), the bound in (4.5) is recomputed, and the increase in w results in a decrease in N_{RANSAC}, enabling early termination once the current iteration count reaches this updated bound.

Regarding the choice of c, (4.5) shows that higher confidence requires more iterations (keeping all other parameters constant), while higher inlier fraction w requires fewer. Thus, c reflects how sure we want to be that at least one all-inlier sample has been drawn from the correspondence set. Setting a larger c trades additional computation for a higher probability of success (i.e., greater confidence that RANSAC's random sampling has produced a good estimate), while setting a smaller c does the opposite.

Combining both criteria, RANSAC terminates when the iteration count reaches the minimum between the maximum iteration threshold (first criterion), and the confidence-based iteration bound (second criterion), at which point, the algorithm returns the best hypothesis (with the highest inlier ratio w) found so far.

Once RANSAC terminates, we initialize point-to-plane ICP with its best hypothesis T_0. At each ICP iteration k, the current transform $T_k = (R_k, t_k)$ is applied to the source, and new correspondences are then formed by a 1-NN search in Euclidean space on a KD-tree built on the target, as detailed in Section 2.3. These

correspondences are also pruned by ICP's inlier threshold so that only pairs within this distance bound are retained.

For each retained pair (p_i, q_i) with target normal n_i, the point-to-plane normal-projected residual is

$$r_i(T_k) = n_i^\top \left(R_k p_i + t_k - q_i\right) \tag{4.6}$$

which can be stacked into one global residual vector:

$$r(T_k) = \begin{bmatrix} r_1 & \dots & r_M \end{bmatrix}^\top \tag{4.7}$$

where M is the number of current correspondences kept. Hence, as previously mentioned, this ICP alternative defines a least-squares objective as

$$\mathcal{L}(T_k) = \|r(T_k)\|_2^2 \tag{4.8}$$

which is equivalent to (3.6). However, since rotations are always nonlinear, (4.8) also becomes nonlinear in T_k. To handle this, we linearize the residuals around T_k using a Gauss-Newton step [3, 63, 64, 65]. With $\Delta\xi = \begin{bmatrix} \delta w & \delta t \end{bmatrix}^\top \in \mathbb{R}^6$ as a small 6-DoF pose increment (3 for rotation, 3 for translation), the first-order Taylor expansion of the residuals around T_k is:

$$r_i(T_k \oplus \Delta\xi) \approx r_i(T_k) + \left.\frac{\partial r_i}{\partial \Delta\xi}\right|_{T_k} \Delta\xi \tag{4.9}$$

We can expand the perturbed transformation as:

$$T_k \oplus \Delta\xi = (\exp([\delta w]_\times) R_k, \;\; t_k + \delta t) \tag{4.10}$$

where $[\delta w]_\times$ is the skew-symmetric matrix such that $[\delta w]_\times v = \delta w \times v$. We can combine (4.10) with (4.6) to obtain

$$r_i\left(T_k \oplus \Delta\xi\right) = n_i^\top \left(\left(\exp([\delta w]_\times) R_k\right) p_i + t_k + \delta t - q_i\right) \tag{4.11}$$

Here, $\exp([\delta w]_\times)$ is the exponential map that converts the small rotation vector into a rotation matrix. For small angles ($|\delta w| \ll 1$), we can approximate this exponential map as:

$$\exp([\delta w]_\times) \approx I + [\delta w]_\times \tag{4.12}$$

So the perturbed rotation of point p_i becomes:

$$\left(\exp([\delta w]_\times) R_k\right) p_i \approx \left(I + [\delta w]_\times\right) R_k p_i = R_k p_i + [\delta w]_\times R_k p_i \tag{4.13}$$

Using the cross-product identity $[a]_\times b = a \times b$ in (4.13), we obtain:

$$(\exp([\delta w]_\times)R_k)\, p_i \approx R_k p_i + [\delta w]_\times R_k p_i = R_k p_i + \delta w \times (R_k p_i) \tag{4.14}$$

So substituting (4.14) in (4.11):

$$\begin{aligned} r_i\left(T_k \oplus \Delta\xi\right) &= n_i^\top \left(R_k p_i + \delta w \times (R_k p_i) + t_k + \delta t - q_i\right) \\ &= n_i^\top \left(R_k p_i + t_k - q_i\right) + n_i^\top \left(\delta w \times (R_k p_i)\right) + n_i^\top \left(\delta t\right) \\ &= r_i(T_k) + n_i^\top \left(\delta w \times (R_k p_i)\right) + n_i^\top \left(\delta t\right) \end{aligned} \tag{4.15}$$

We can derive it with respect to translation δt. Since the first two terms are constant, we get:

$$\frac{\partial r_i}{\partial \delta t} = n_i^\top \tag{4.16}$$

And with respect to rotation δw, where the first and last terms are constant, we can use the cyclic permutation of the scalar triple product $a^\top(b \times c) = (c \times a)^\top b$ to obtain:

$$\frac{\partial r_i}{\partial \delta w} = \frac{\partial\ n_i^\top \left(\delta w \times (R_k p_i)\right)}{\partial \delta w} = \frac{\partial\ \left((R_k p_i) \times n_i^\top\right) \delta w}{\partial \delta w} = (R_k p_i) \times n_i^\top \tag{4.17}$$

We can assemble (4.16) and (4.17) into the per-pair Jacobian matrix as

$$J_i = \begin{bmatrix} \frac{\partial r_i}{\partial \delta w} & \frac{\partial r_i}{\partial \delta t} \end{bmatrix} = \begin{bmatrix} (R_k p_i \times n_i)^\top & n_i^\top \end{bmatrix} \tag{4.18}$$

By stacking these into the complete Jacobian matrix $J = \begin{bmatrix} J_1 & \dots & J_M \end{bmatrix}^\top \in \mathbb{R}^{M \times 6}$, we can also stack all residuals in the linearized form (4.9), yielding

$$r \approx r_0 + J\Delta\xi \tag{4.19}$$

The Gauss-Newton normal equations are

$$\left(J^\top J\right) \Delta\xi = -J^\top r_0 \tag{4.20}$$

Solving for $\Delta\xi$ gives the small pose increment, which is composed with T_k (through the exponential map) to obtain the new updated transformation T_{k+1}. Then, this loop repeats until convergence: recompute the new correspondences under T_{k+1}, linearize the residuals, solve the least-squares system, and then update the transformation.

We adopt three convergence criteria: a hard cap on the maximum number of iterations, and relative change thresholds on both fitness and inlier RMSE between

successive iterations (i.e., a threshold that verifies if the drop in fitness and inlier RMSE is still significant). If any criterion is met, ICP stops and returns the refined transformation [7, 46].

As already mentioned, all parameters adopted for RANSAC and ICP are reported in the methodology section corresponding to each dataset (Sections 4.5 and 4.7).

4.4 Iteration Counters and Timers

Besides the metrics directly related to the alignment quality produced by the obtained transformation (i.e., fitness and inlier RMSE), we also want to compare the results in terms of runtime. A good approach is to measure the number of iterations of the registration stages (RANSAC and ICP), which allows for a device-agnostic view, since the number of iterations is independent of the underlying hardware, whereas time itself can vary significantly. Since Open3D does not provide a built-in way to retrieve iteration counts, we must implement our own method using a workaround. Our approach is to first set the Open3D verbosity level to debug. This setting causes the console to print detailed information during execution, including the iteration counts we are interested in.

However, to be able to access this log information, we must capture the output by redirecting it from the console to a variable. The output redirection mechanism expects a file-like object with a write method. A regular Python variable (like a string) is not a file-like object, so it does not provide the necessary interface to receive this log data. Instead, we use the `StringIO` class from the `io` module, which acts as an in-memory stream (a file-like object) that can be written to and then queried to retrieve its contents as a string. The problem with this approach is that it fully redirects the output to this buffer, which prevents us from visualizing anything in the console during runtime, making it impossible to monitor the execution. To avoid this, we define a custom `Tee` class, which is a subclass of `io.StringIO`. A "tee"-like class writes to both the console and an in-memory buffer. Then, we can redirect `stdout` to this object. The name Tee comes from the Unix `tee` command, which reads from standard input and splits or duplicates that stream to both standard output and one or more files. The name alludes to the shape of a T, where a single "flow" is split into two paths. In the same way, our Python `Tee` class duplicates the stream of text so that it goes both to the console and into an internal buffer.

Once we have access to the captured output, we can parse it. First, we observe the console output to identify Open3D's debugging logs and locate the relevant lines. For RANSAC, the number of validations is reported after it completes, producing lines such as

```
[Open3D DEBUG] RANSAC exits after 198 validations. Best inlier ratio
    4.652035e-01, RMSE 3.927418e-02.
```

Meanwhile, for ICP, instead of one single log summarizing the results of all iterations, we have individual logs for each iteration, as shown below:

```
[Open3D DEBUG] ICP Iteration #0: Fitness 0.5680, RMSE 0.0133
[Open3D DEBUG] Residual : 1.03e-04 (# of elements : 256714)
[Open3D DEBUG] ICP Iteration #1: Fitness 0.5680, RMSE 0.0133
[Open3D DEBUG] Residual : 1.03e-04 (# of elements : 256704)
...
```

We can then simply find the last iteration of each alignment to obtain the total number of ICP iterations. Note that in these logs Open3D considers the ICP iterations to be zero-indexed (i.e., it begins with "ICP Iteration #0" instead of "ICP Iteration #1"), so we need to add one to get the proper total.

For the deep-learning pipeline, we cannot use the `Tee` object during the RANSAC stage, because it executes in a separate Python script launched as a subprocess (the same script provided by the authors that contains the feature extraction stage). The `Tee` object works by overriding Python's standard output in the current process, but a subprocess does not share the same `stdout` environment as the parent process. Thus, it would have no effect on the subprocess's output stream. Instead, we manually pipe the subprocess's `stdout` back into the parent's environment, read it line by line, and forward each line both to the actual console and to a buffer for later use. More details on this implementation can be seen in Notebook 5 of the codebase [8].

We also propose implementing a timer for a refined analysis focusing on a time-based comparison of the pipelines rather than just iteration counts. Although the iteration counts are very useful to assess how well each method performs in a device-agnostic way, we must evaluate whether the feature-extraction time is comparable between both pipelines, since it is not captured by the iteration counters (i.e., the feature extraction stage does not have an iterative behavior that we can measure as we did for RANSAC and ICP). Since the deep-learning pipeline has a much more complex structure and working principle, one would expect it to take longer than the geometric-only one, so we must confirm whether the time saved in the RANSAC and ICP stages is enough to compensate for the extra time required by the feature extractor.

To implement this, we define a timer decorator that can be added to any function (without requiring any modification to its code) to return the time taken by that function. To measure the time, the decorator uses `perf_counter` from the

`time` module, which has a higher resolution and therefore provides more accurate measurements. In practice, we implemented it with a decorator factory pattern, which allows us to pass input values when invoking it (in this case, the names of the stages), facilitating its application across different parts of the code.

We split the pipelines into three stages: Preprocessing (which involves feature extraction, downsampling, and normal computation, but the time here is predominantly due to feature extraction), RANSAC, and ICP. Notice that all experiments were run on the same hardware, which is crucial to guarantee a fair comparison in terms of runtime.

4.5 3DMatch Benchmark

With our pipelines implemented, we first evaluate them on the 3DMatch benchmark [32]. Following common practice, we use the publicly released 3DMatch data preprocessed by PREDATOR [26]. The first step is therefore to adapt both pipelines to iterate over the entire dataset. According to the 3DMatch guidelines, for each scene we must evaluate our pipelines by registering all non-consecutive fragment pairs, i.e., pairs (P_i, Q_j) such that $j > i + 1$. Additionally, the official 3DMatch evaluation protocol also states to consider only fragment pairs whose overlap ratio is greater than 30%.

Each pipeline loops over all scenes in 3DMatch and attempts to align all such non-consecutive pairs. There is, however, a small structural difference between the two implementations. The geometric-only pipeline was fully developed by us, and is organized so that, for each pair, it executes the complete sequence of stages (feature extraction, RANSAC alignment, ICP refinement) before moving on to the next pair. In contrast, the deep-learning pipeline is based on the authors' original FCGF code, which is organized in a batched fashion: it first extracts features for all pairs, then runs RANSAC for all pairs, and finally runs ICP for all pairs. Thus, instead of running the full pipeline pair by pair, it processes the dataset stage by stage.

As discussed in Section 4.2, we also had to modify the original 3DMatch evaluation setting in the FCGF code, since it was considering all fragment pairs, including consecutive ones, rather than restricting to non-consecutive pairs as recommended in the 3DMatch protocol.

For both pipelines, we provide an optional setting to run only a subset of scenes or pairs instead of the full benchmark, which is useful for debugging and sanity checks. Nevertheless, for all tests presented in this text, we considered the whole set.

Once both pipelines have finished running on the whole dataset, we parse the captured Open3D console logs to extract the iteration counts, as described in

Section 4.4. For this benchmark, we record the number of RANSAC and ICP iterations, as well as the fitness (2.17) and inlier RMSE (2.18) of the transformation obtained after each registration stage (RANSAC and ICP). These metrics are described in detail in Section 2.4.

We then aggregate the results as follows. For each scene, we compute the mean and standard deviation (SD) of the alignment metrics over all evaluated pairs. From these per-scene statistics, we then compute the global average and SD over the whole benchmark, as well as the inter-scene SD, which measures the variability of performance across different scenes. In addition, using the timer described earlier, we estimate the runtime of each feature-extraction stage (FPFH or FCGF) and the average time per iteration of RANSAC and ICP.

Table 4.1 summarizes the parameters used in the 3DMatch benchmark. Notice that the preprocessing and registration parameters are shared by both pipelines.

4.6 Noise Generator and Augmented 3DMatch Benchmark

Although the standard 3DMatch benchmark is already based on real data and therefore includes real sensor noise, we further assess the robustness of each pipeline by evaluating them under increased noise conditions (simulating degraded environments or lower-quality sensors). To this end, we design a data-augmentation stage responsible for generating additional noise and injecting it into the 3DMatch point clouds. It can produce three different types of noise, described in this section.

We begin with Gaussian noise. Gaussian noise models the small, random measurement errors commonly observed in real cameras and LiDARs. It follows the normal distribution, defined by a mean μ and variance σ^2, whose probability density function is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{4.21}$$

Figure 4.3 illustrates how different choices of mean and standard deviation affect the shape of the distribution.

We implement two variants of Gaussian noise. The first applies a single normal distribution to all points, meaning that every point is perturbed by noise drawn from the same distribution. The second variant introduces spatially varying noise levels, so that different points are affected by different variances, making the perturbation less deterministic and more representative of the heterogeneity typically observed in real-world data.

In the fixed-variance case, we sample noise from a normal distribution $\mathcal{N}(0, \sigma^2)$

Stage	Parameter	Value	Note
Downsampling (Voxelization)	Voxel size VS	0.05	5 cm
Normal estimation	Search radius r_n	0.10	10 cm ($2VS$)
	Max neighbors k_n	30	
FPFH	Search radius r_f	0.25	25 cm ($5VS$)
	Max neighbors k_f	100	
FCGF	Network configuration	ResUNetBN2C -16feat-3conv	Pretrained weights [5, 61]
RANSAC	Inlier threshold	0.05	5 cm ($= VS$)
	Sample size m	4	Minimal set
	Mutual correspondence	enabled	Reciprocal NN in feature space
	Edge-length similarity s_{th}	0.9	Checker Threshold (4.4)
	Max iterations	10000	Hard cap
	Confidence c	0.999	As in (4.5)
ICP	Inlier threshold	0.05	5 cm ($= VS$)
	Max iterations	30	Hard cap
	Fitness drop threshold	10^{-6}	Successive criterion
	Inlier RMSE drop threshold	10^{-6}	Successive criterion
Benchmark Protocol	Minimum overlap	30%	3DMatch protocol [32]
	Pair selection	(P_i, Q_j)	$j > i + 1$

Table 4.1: Hyperparameters and evaluation protocol used in the 3DMatch benchmark. Preprocessing and registration parameters are shared by both pipelines.

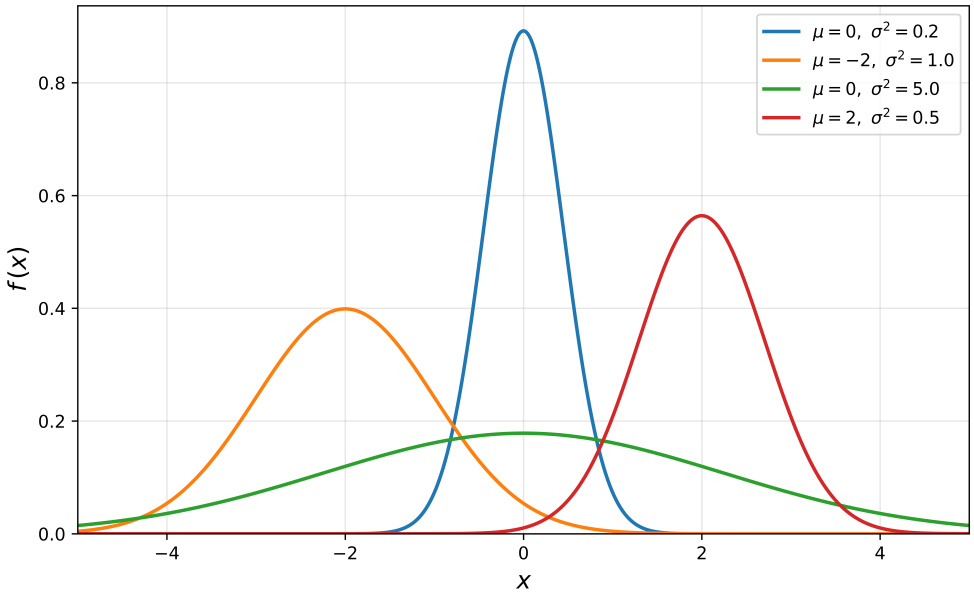


Figure 4.3: Examples of normal distributions for different combinations of mean μ and standard deviation σ. Increasing σ broadens the distribution, while changing μ shifts it along the horizontal axis.

with a user-defined standard deviation σ, and add it to each point. In the variable-variance case, we first construct an array of standard deviations $\{\sigma_i\}$, one per point in the cloud. To do so, each σ_i is sampled from a uniform distribution $\sigma_i \sim \mathcal{U}(\sigma, \sigma_{\max})$, where σ and $\sigma_{\max}$ are user-specified minimum and maximum noise levels. Then, for each point, we generate noise from its own distribution $\mathcal{N}(0, \sigma_i^2)$, using the corresponding entry σ_i (i.e., the i-th point from the cloud uses the i-th standard deviation from the array). Hence, each point is perturbed by a different normal distribution, which breaks the deterministic behavior of the fixed-variance case and yields a more diverse and realistic noise pattern.

To further simulate real cameras, we also introduce spike readings for a small fraction of points. These points represent gross outliers acquired by the sensor with large depth errors. The spike magnitudes are allowed to vary over a range much larger than the Gaussian noise described previously. Conceptually, these spikes should occur in an asymmetric manner, with many small outliers and only a few huge ones. To model this behavior, instead of a standard normal distribution we consider a positively skewed distribution (Figure 4.4), so that smaller spikes are sampled more often than the largest ones.

Rather than sampling directly from a skew-normal distribution, we implement a simple and efficient sampling scheme that induces a similar positive skew over the

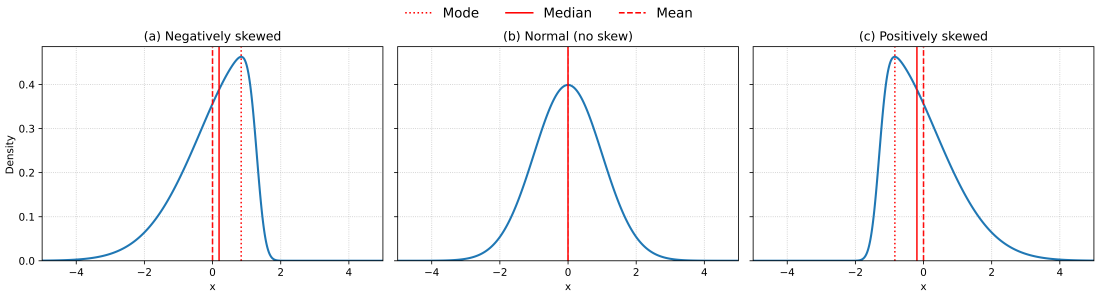


Figure 4.4: Examples of distributions with different skew settings. (a) Negatively skewed: the mass is concentrated toward larger values. (b) Symmetric (no skew): standard symmetric normal distribution. (c) Positively skewed: density on small values and a long tail toward larger values.

spike magnitudes. We first create an auxiliary array u by drawing n_{spikes} samples from a uniform distribution:

$$u_i \sim \mathcal{U}(0,1), \quad i = 1, \ldots, n_{\text{spikes}} \tag{4.22}$$

If we were to map these directly to magnitudes via

$$m_i = spike_{\text{min}} + (spike_{\text{max}} - spike_{\text{min}})\, u_i \tag{4.23}$$

we would obtain uniformly distributed spike magnitudes in $[\text{spike}_{\text{min}}, spike_{\text{max}}]$. To bias the magnitudes toward smaller values, we instead apply a power transform to the uniform samples:

$$m_i = spike_{\text{min}} + (spike_{\text{max}} - spike_{\text{min}})\, u_i^{\gamma} \tag{4.24}$$

where $\gamma > 1$ is a user-specified spike-skew parameter. Hence, if u_i is close to 0, then u_i^{γ} is even closer to 0, so $m_i \approx spike_{\text{min}}$. Otherwise, if u_i is near 1, then u_i^{γ} remains close to 1, so $m_i \approx spike_{\text{max}}$. This shows that, because of the exponent, most of the transformed values u_i^{γ} cluster near 0, producing many small spikes and only a few large ones, which is exactly the desired behavior.

The discussion above concerns the spike magnitudes. For the directions, real depth sensors tend to produce outlier spikes approximately along the camera's viewing ray, i.e., the local optical axis (often the camera's Z-axis) [66]. However, the 3DMatch fragments are pre-fused from many RGB-D frames into a single global point cloud, so we no longer know which original camera ray produced each point, nor which points were visible in which views. Figure 4.5 illustrates this situation: the same surface point may be observed from different camera positions, resulting in different viewing directions.

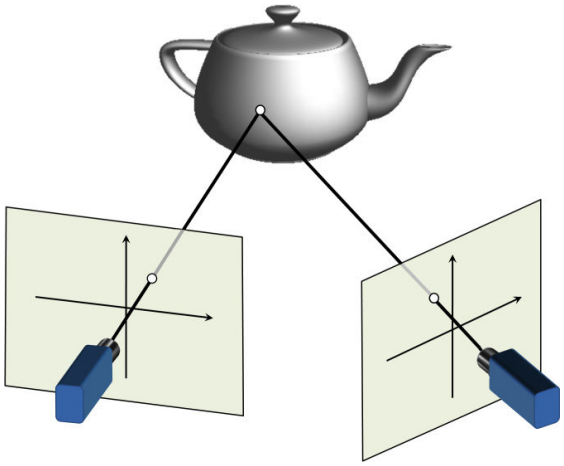


Figure 4.5: Illustration of a point on an object being observed from two different camera positions, yielding different viewing rays and image-plane projections. In a pre-fused point cloud, the original per-frame viewing directions are no longer explicitly available, which complicates injecting spike noise along the true camera rays. Source: [67].

Recovering accurate per-point camera-ray directions would require injecting spikes before fusion: for each raw depth frame, we would back-project pixels into 3D using the camera intrinsics, perturb selected points along their individual rays, and then re-fuse all frames into a single point cloud [66, 67]. This would significantly increase the complexity of the preprocessing pipeline. For our purposes (a controlled robustness test) we instead adopt a simpler approximation and sample spike directions as random 3D unit vectors. This trades strict physical realism for ease of implementation, while still allowing us to probe the sensitivity of the registration pipelines to sparse, large magnitude outliers.

For the third and final noise, we use salt-and-pepper noise, which refers to sparse corruptions in which a small fraction of samples are either set to extreme values ("salt") or removed entirely ("pepper") [68], as illustrated in Figure 4.6.

In point clouds, we focus on the pepper case by randomly deleting a percentage of points to mimic missing or occluded measurements. In practice, the user specifies a percentage $p_{\text{pepper}} \in [0,1]$. We compute the number of points this percentage represents and uniformly sample this amount of point indices (without replacement), and build a boolean mask that is `false` at the sampled indices and `true` elsewhere. Applying this mask filters out the selected points from the coordinates (and any associated color/normal arrays). We then rebuild the point cloud from the remaining points. This lightweight scheme is intuitive and provides a simple, single-parameter control over dropout sparsity.

Our deep-learning pipeline uses MinkowskiEngine for sparse tensors. When

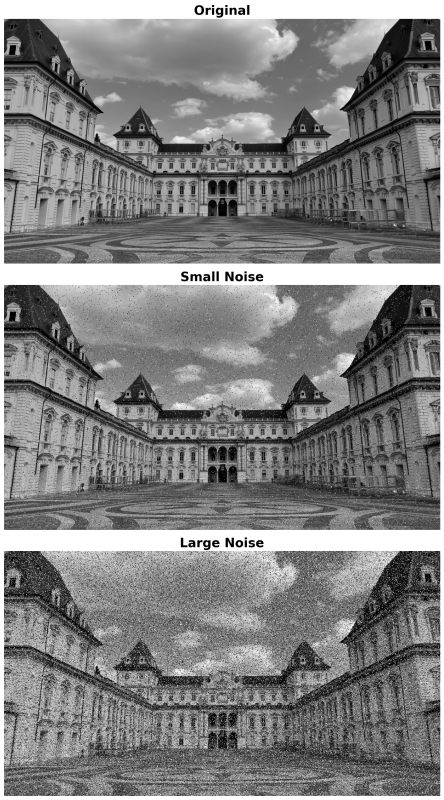


Figure 4.6: Illustration of increasing salt-and-pepper corruption levels. Top: clean data. Middle: moderate perturbation. Bottom: heavy perturbation.

combining our augmentation with sparse quantization, we observed occasional issues under multi-threaded loading. For the augmented benchmarks we therefore enforced single-worker execution, which eliminated nondeterministic failures while leaving results unaffected. Details, diagnostics, and impact analysis are reported in Appendix A.

We follow the same evaluation procedure described in Section 4.5, now using the noise-augmented clouds. All pipelines' parameters remain as in Table 4.1. The specific augmentation settings used to generate the 3DMatch augmented benchmark are listed in Table 4.2.

4.7 Custom cross-domain dataset creation and testing

To complement the previous scene-scale experiments on 3DMatch, we also evaluate our pipelines at object-scale. For this, we adapt SuctionNet [9] into a cross-domain

Stage	Parameter	Value	Note
Noise Generator	Gaussian Mode	variable	
	$\sigma_{\min}$	0.01	1 cm
	$\sigma_{\max}$	0.05	5 cm
	Spike ratio	0.005	0.5% of points
	$spike_{\min}$	0.1	10 cm
	$spike_{\max}$	0.5	50 cm
	Skew exponent γ	2	Positive skew
	p_{pepper}	0.01	1%

Table 4.2: Noise–generator parameters used in the augmented 3DMatch benchmark. All preprocessing and registration parameters remain as in Table 4.1.

registration benchmark in which the source is a clean model of the object and the target is a real RGB-D capture of the same object. This setting stresses robustness to real sensor noise and the appearance gap between ideal CAD geometry and measured data.

SuctionNet already provides dense point clouds sampled from the CAD meshes. Rather than converting triangle meshes (CAD models) to point clouds ourselves (e.g., via Poisson-disk sampling), we directly adopt the authors' dense point clouds as source data and only convert their format for Open3D compatibility [7]. For our targets, we reconstruct point clouds from the provided real RGB-D images, considering both the whole tabletop scene and the individual objects. For convenience and storage constraints, we use the fourth training split, since it is the smallest subset. This does not affect evaluation fairness because we do not train any network on SuctionNet.

Hence, to build this dataset, our goal is to convert 2D RGB-D images into 3D point clouds. The dataset provides, for each scene and each view, a color image (RGB) and a matched depth map in which each pixel encodes its distance from the camera. By combining these, we recover a 3D point for each pixel, yielding an RGB-D point cloud. This conversion requires the camera's intrinsics and, in general, the extrinsics. Intrinsics describe the camera's internal geometry under the pinhole model [7, 69, 70] and map 3D points in the camera frame to the image plane. The intrinsic matrix K is:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.25}$$

with focal lengths (f_x, f_y), in pixels, and principal point (o_x, o_y). A 3D point $p_c = (x_c, y_c, z_c)^\top$ in the camera frame is projected to image coordinates (u, v) through

$$u = \frac{f_x x_c}{z_c} + o_x, \qquad v = \frac{f_y y_c}{z_c} + o_y \tag{4.26}$$

The extrinsics describe the camera pose in a larger frame (e.g., world or robot) and consist of the rotation R and translation t:

$$p_w = R p_c + t \tag{4.27}$$

Together, intrinsics and extrinsics form the standard camera matrix $P = K\,[R \mid t]$ that maps world points to image points [69, 70]. However, in our particular case, since we are not interested in reconstructing the absolute pose of the 3D points, but rather just the object geometry, we rely solely on the intrinsics.

Given a pixel (u, v) with stored depth $d(u, v)$ and device depth scale s (to convert device units to meters), we can obtain the coordinate along the camera's axis $Z_c = s \cdot d(u, v)$. Thus, we can apply the inverse projection process (back-projection), where we convert pixels (image coordinates) to 3D (world coordinates):

$$\begin{aligned} x_c &= \frac{(u - o_x)}{f_x} Z_c, \\ y_c &= \frac{(v - o_y)}{f_y} Z_c, \\ z_c &= Z_c \end{aligned} \qquad \Longleftrightarrow \qquad p_c = Z_c\, K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.28}$$

Additionally, RGB values at (u, v) can be attached to each 3D point to form a colored point cloud. In practice, the authors provide the intrinsic matrix for each scene in a dedicated file, which we load and wrap in Open3D's `PinholeCameraIntrinsic` object (along with image height and width) [7]. Each scene contains multiple views along a fixed camera trajectory (Figure 2.7 presents an RGB-D sample example).

With intrinsics and images in hand, we convert each RGB-D pair into a point cloud. We first build an `RGBDImage` in Open3D (an object that simultaneously stores the color and depth images) and then invoke Open3D's routine to create the point cloud via the back-projection in (4.28). Figure 4.7 shows the resulting point cloud for a full tabletop scene.

After obtaining the full scene cloud, we extract per-object clouds. Along with RGB and depth, each scene and view also provides a label image in which each

Figure 4.7: Point cloud of an entire SuctionNet tabletop scene obtained from a single RGB-D view using the intrinsics. Note that depth noise is clearly visible.

pixel stores the integer ID of the object at that location (or 0 for background). Figure 4.8 annotates a label image to reveal object IDs per pixel (downsampled for legibility).

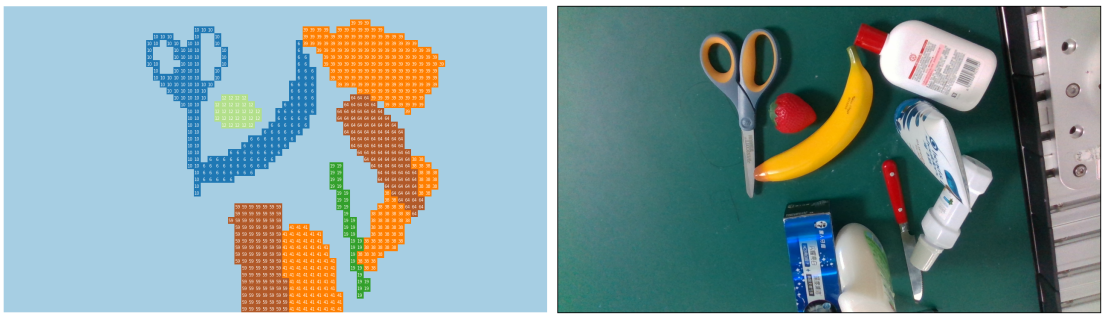


Figure 4.8: Left: label image, where each pixel value is the ID of the visible object (blanks are 0, which indicates background). Right: corresponding RGB image.

Given a target object ID, we form a binary mask by setting pixels equal to that ID to 1 and all others to 0, thereby isolating the object region. We apply this mask to both the RGB and depth images and then repeat the same RGB-D to point-cloud conversion, yielding a cloud that contains only the selected object. Because SuctionNet's object list is 0-indexed while labels use 1 for the first object

(with 0 reserved for background), there is an offset of one between the object index and the label value. Figure 4.9 shows an example mask and its effect on RGB and depth.

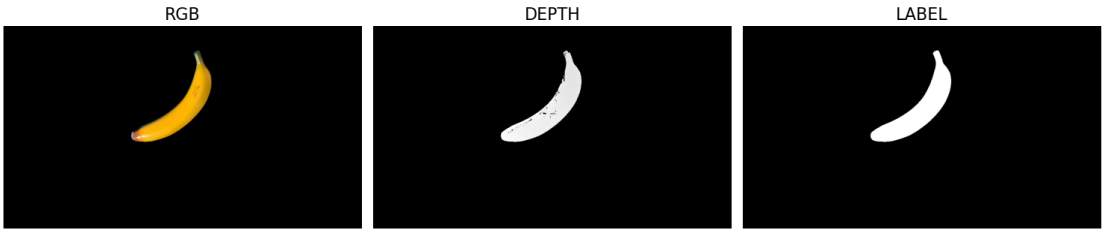


Figure 4.9: Right: binary mask derived from the label image (white = object, black = background). Left and center: RGB and depth after masking, which retain only the selected object's pixels.

Finally, applying the same intrinsics to the masked RGB-D pair yields the reduced, per-object point cloud (Figure 4.10).

With the procedures for extracting both the full-scene and per-object clouds properly implemented, we create a simple pipeline to process all available samples. For each scene, and for each view along the trajectory, we first obtain the full-scene cloud and then, while on the same view, we read the label image to enumerate all the unique object IDs present. Then, for each such object, we apply the masking procedure above and generate its corresponding point cloud. In this way, we obtain a novel cross-domain dataset adapted from SuctionNet [9] which provides both full-scene and per-object point clouds along with the original CAD models. The script that performs this extraction is provided in our repository [8].

Additionally, another test idea was to crop the CAD models in half to better match the visibility of the real acquired clouds, which are typically only partially visible due to the camera viewpoint. To do so, we performed PCA to obtain the three principal axes of variation. The eigenvectors associated with the two largest eigenvalues represent the directions along which the point cloud has the largest spread (typically corresponding to the longitudinal and transversal axes of the object). The third eigenvector typically corresponds to the width direction. After obtaining the eigenvectors, we rotate the cloud to align with these directions and then apply a mask to keep only one half along a chosen axis (either the positive or negative side). Figure 4.11 shows the resulting splits along each PCA axis. As expected, the third axis splits the object along its width direction, which is also the most common type of partial visibility in the cluttered tabletop configurations of SuctionNet. Therefore, for this test case, we use CAD models cropped along the third principal axis as our sources.

Then, with the dataset ready, we can set up the benchmark on it. In this

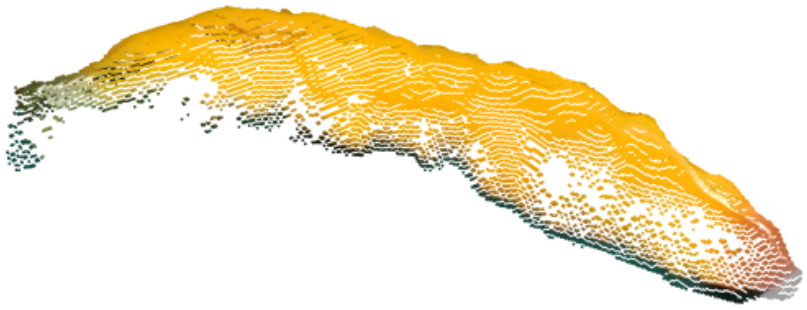


Figure 4.10: Point cloud of a single object obtained from the masked RGB-D image via back-projection with the intrinsics. Notice the large noise and heavy perturbations across the cloud.

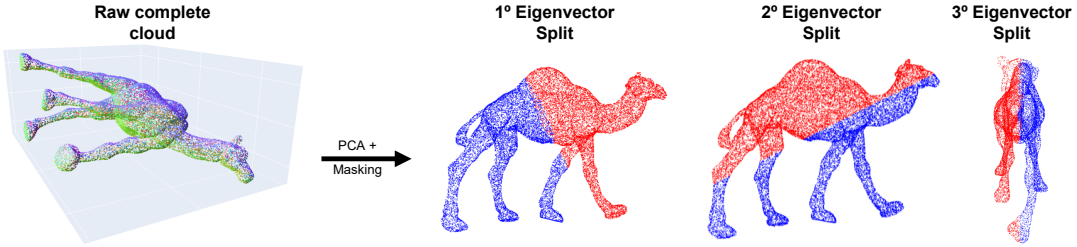


Figure 4.11: Given a complete CAD model, we perform PCA and split the cloud into two halves along each principal axis, keeping the positive (blue) or negative (red) half. The first two axes typically correspond to the longitudinal and transversal directions, while the third axis usually corresponds to the width direction.

cross-domain setting, the source cloud (the CAD model) is always centered near the origin, whereas the target cloud reconstructed from RGB-D lies at a nonzero offset determined by the distance between the object and the camera lens (i.e. the optical center). Consequently, the two clouds may start with a substantial initial offset, as illustrated in Figure 4.12.

Although this might seem harmless, it can complicate the pipelines. Recall that our geometric pipeline uses FPFH and the learned one uses FCGF for feature extraction. Both descriptors aim for local invariance: translation invariance by construction, and approximate rotation stability via local frames for FPFH and

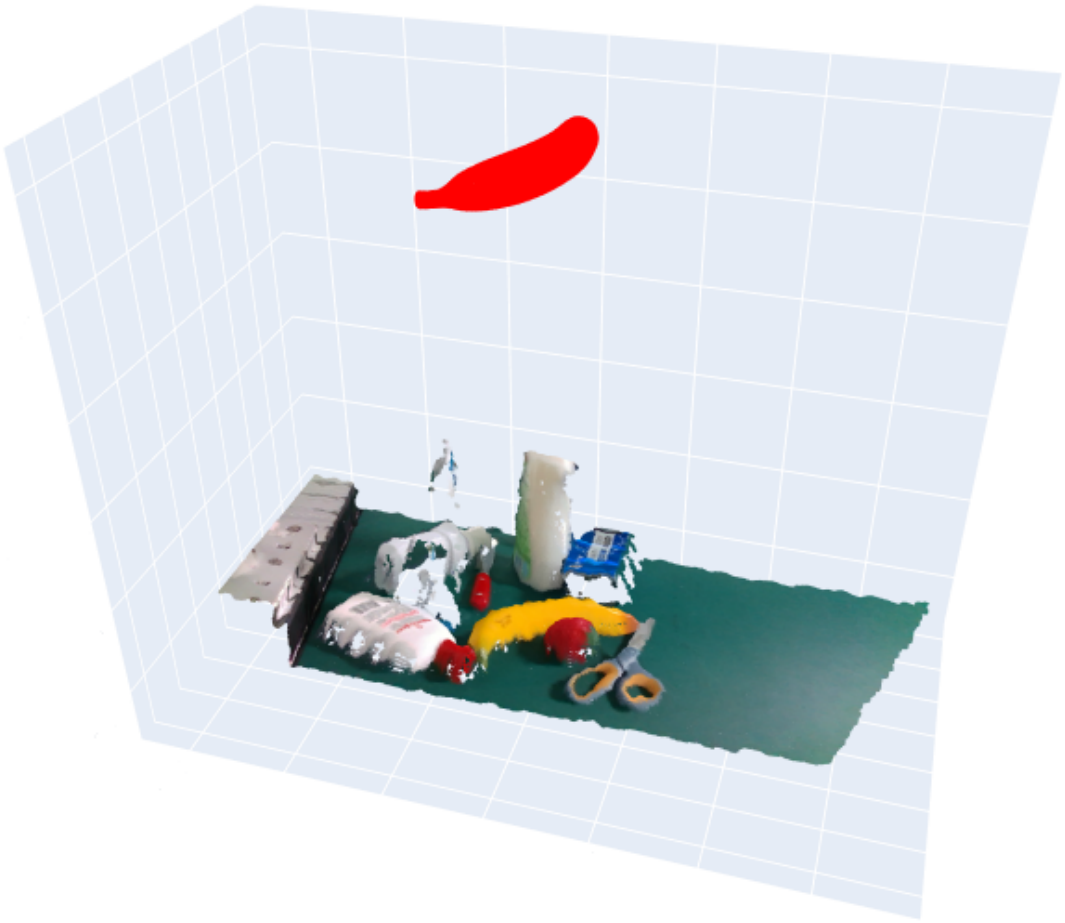


Figure 4.12: Because the source clouds (colored tabletop reconstructions) are centered around the origin while the targets (CAD models, shown in red) are expressed in the camera frame, there is an inherent distance between object and camera. This yields a sizable initial translation offset between source and target before registration.

learned invariances for FCGF [4, 5], but they remain sensitive to such large pose discrepancies.

When the initial misalignment is large (due to significant translation, rotation, or both), corresponding surface patches can exhibit significantly different local contexts (e.g., normal orientations, neighborhood layouts, visibility and occlusion patterns). Because descriptors are computed independently in each cloud, these differences cause descriptor vectors at physically corresponding points to diverge. As a result, nearest-neighbor matching in feature space fails to assemble a reliable

correspondence set, and without meaningful correspondences, RANSAC cannot estimate a valid transformation.

FCGF is generally more tolerant than FPFH in this regime, since it is trained to match similar structures under broader pose and appearance variations [5]. Still, even FCGF may fail when the initial offset is extreme.

Although such failures may not be frequent, we address them to make both pipelines more robust. A tempting fix is a coarse pre-translation that aligns centroids before feature matching. However, if the target lies far from the global origin, simply shifting the source to the target centroid still leaves both clouds with large absolute coordinates. Since downstream computations (KD-tree queries for normal/feature neighborhoods, correspondence search, and point-to-plane residuals in ICP) operate in single precision (float32), subtracting large, nearly equal values can amplify rounding error (loss of significance) and destabilize normals and descriptors, degrading matching quality [71]. In short, large dynamic ranges combined with misalignment can hurt both correspondence building and the subsequent estimation stages.

A more robust solution is to demean (center) both point clouds at the origin before computing features or running the registration stages. Keeping coordinates near the origin reduces dynamic range and improves numerical conditioning for subsequent steps (KD-tree queries, plane fitting, and feature matching), which are typically carried out in single precision [69, 71].

Hence, using the source P and target Q clouds, we first compute the centroid of each point cloud, as in (2.6). Second, we compute the translation vectors from the centroids to the origin:

$$t^o_{\text{source}} = (0,0,0) - \bar{p},\ t^o_{\text{target}} = (0,0,0) - \bar{q}, \tag{4.29}$$

where t^o_{source} and t^o_{target} translate the source and target centroids, respectively, to the origin. We then build the rigid transformations that map coordinates from the original source/target frames to the origin-centered frame (rotation set to identity since we only translate):

$$\begin{aligned} T^o_{\text{source}} &= \begin{bmatrix} I_3 & t^o_{\text{source}} \\ 0^T & 1 \end{bmatrix} \\ T^o_{\text{target}} &= \begin{bmatrix} I_3 & t^o_{\text{target}} \\ 0^T & 1 \end{bmatrix} \end{aligned} \tag{4.30}$$

To move the clouds to the origin (Figure 4.13), we apply these transformations in homogeneous coordinates:

$$\begin{aligned} \tilde{P}^o &= T^o_{\text{source}} \tilde{P} \\ \tilde{Q}^o &= T^o_{\text{target}} \tilde{Q} \end{aligned} \tag{4.31}$$

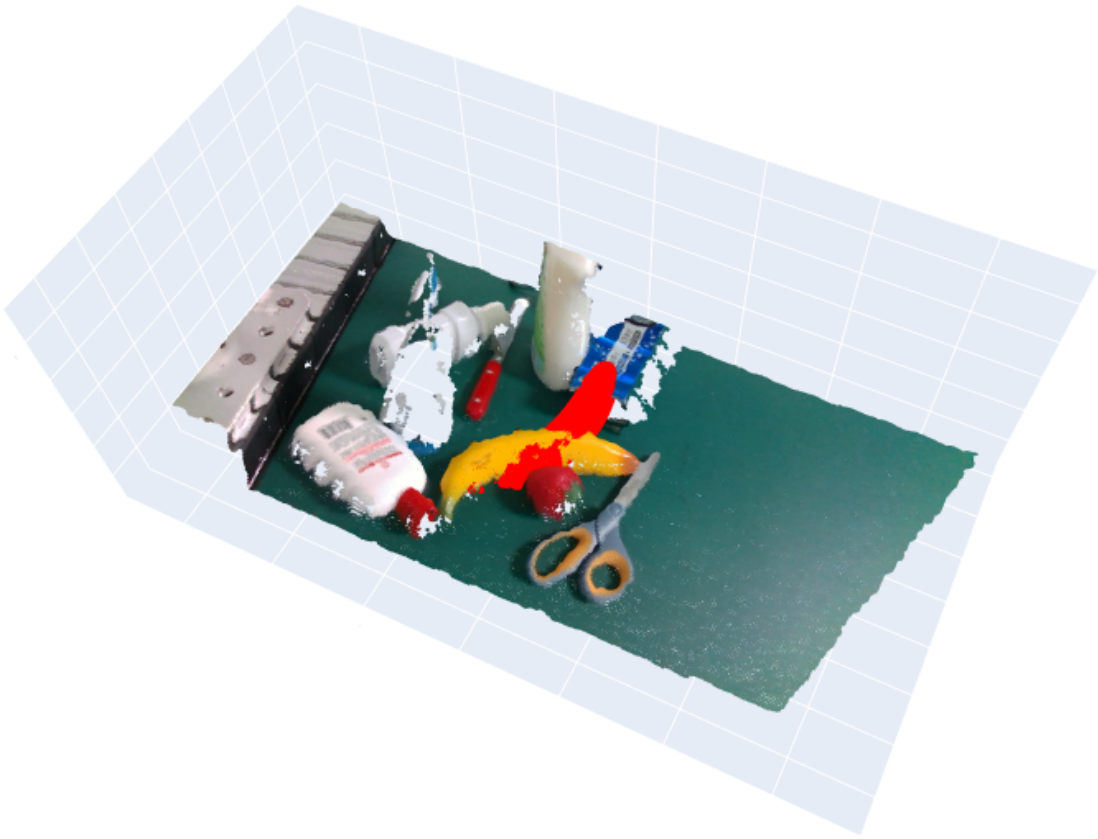


Figure 4.13: Both clouds are translated so that their centroids coincide with the origin. We then feed these origin-centered clouds to the registration stage. This inexpensive preconditioning step improves robustness under large initial misalignment by stabilizing feature matching.

Here, $\tilde{P}^o$ and $\tilde{Q}^o$ denote the source and target clouds represented in the origin-centered frame (homogeneous form). Next, we input $\tilde{P}^o$ and $\tilde{Q}^o$ to the selected registration pipeline to obtain the transformation T_{reg} that, ideally, aligns the (centered) source to the (centered) target:

$$\tilde{Q}^o = T_{\text{reg}} \tilde{P}^o \tag{4.32}$$

Lastly, we compose T_{reg} with the initial translations to recover the overall transformation T_{final} from the original (unmoved) source to the original (unmoved) target. This can be derived using elementary properties of rigid transformations and matrix arithmetic from (4.31) and (4.32), as shown below.

$$\begin{aligned} \tilde{Q}^o &= T_{\text{reg}}\tilde{P}^o \\ T^o_{\text{target}}\tilde{Q} &= T_{\text{reg}}T^o_{\text{source}}\tilde{P} \\ (\text{target}^o)^{-1}\, T^o_{\text{target}}\tilde{Q} &= \left(T^o_{\text{target}}\right)^{-1} T_{\text{reg}}T^o_{\text{source}}\tilde{P} \\ \tilde{Q} &= \left(T^o_{\text{target}}\right)^{-1} T_{\text{reg}}T^o_{\text{source}}\tilde{P} \end{aligned} \tag{4.33}$$

We can rewrite the final equation as:

$$\begin{aligned} T_{\text{final}} &= \left(T^o_{\text{target}}\right)^{-1} T_{\text{reg}}T^o_{\text{source}} \\ \tilde{Q} &= T_{\text{final}}\tilde{P} \end{aligned} \tag{4.34}$$

This procedure is implemented as a lightweight, fully automatic preprocessing step in both pipelines. While computationally inexpensive, it significantly improves robustness by enabling successful feature matching in cases of large initial displacement.

Besides this initial centering step, the pipelines themselves remain unchanged. In the full-scene setting, for each view from each scene, we obtain the list os objects present (using the label image to identify the set of unique object IDs present in the current view) and, for each of those objects, we apply both pipelines with the source as the object's CAD model and the target as the full tabletop cloud. Similarly, for the single-object setting, we loop through all scenes, views and objects within views, aligning the CAD models to the corresponding single-object acquired cloud. In all scenarios, we also apply the same iteration-counter module described in Section 4.4, with minor adjustments to match this dataset folder layout.

To evaluate the consistency of each pipeline, we report an approximate recall: among all evaluated pairs for which the pipeline outputs a transform (attempted registrations), how many are deemed correct (successful) under a fixed criterion. In standard settings such as 3DMatch, correctness relies on ground-truth poses. Specifically, (i) pair selection considers only pairs with overlap greater than 30% under the ground-truth pose, and (ii) success is assessed by comparing the estimated transform against ground-truth (e.g., via inlier ratios or RMSE under the ground-truth alignment) [32].

By contrast, when ground-truth is unavailable, prior work often uses an overlap proxy evaluated under the predicted pose. For example, FCGF computes an overlap ratio after applying the estimated transform and accepts a registration if the overlap exceeds 30% [5]. This mirrors the 3DMatch "30% rule" but uses the predicted instead of the true pose and functions as a post-registration success filter rather than a pair-selection gate.

In our case, SuctionNet does not provide ground-truth transforms, so we adopt Open3D's fitness (defined in (2.17)) as a pragmatic overlap proxy without requiring

ground-truth: a high fitness indicates a large, well-aligned intersecting region [7]. Accordingly, we approximate recall by counting positives as registrations whose fitness exceeds 30%, a threshold consistent with the 3DMatch guideline and FCGF practice [32, 5].

However, using this overlap proxy for evaluation comes with a warning in the full-scene setting. If we directly use the fitness reported by Open3D during registration, it can be overly optimistic because fitness counts inliers from an estimated correspondence set (obtained through NN search). When aligning a specific object's CAD model to a full tabletop point cloud, the target contains many unrelated points (e.g., the table and other objects). The NN search can therefore match CAD points to these "false positives" unrelated points, yielding a high fitness even when the model is aligned to the wrong object. To ensure that fitness reflects alignment to the correct object, we estimate the transformation using the full-scene target but evaluate it against the corresponding single-object cloud (instead of reusing the whole tabletop cloud). This is valid because the full-scene and single-object clouds are expressed in the same coordinate frame (masking removes points but does not change coordinates), and our centering step is internal to the pipeline, with the reported T_{final} mapping the original CAD coordinates to the original target frame, regardless of the setting.

As in Section 4.5, we summarize only successful alignments when computing per-metric aggregates (means and standard deviations). We consider the same metrics: iteration counts, fitness and inlier RMSE. We report results both per object and per scene to expose variability across object identities and across scenes (which differ in object placement and degree of occlusion).

Finally, for this benchmark we set the voxel size and the inlier distance threshold to 0.005 (0.5 cm), while all other parameters remain exactly as in Table 4.1.

Chapter 5

Results

5.1 3DMatch Benchmark

This section reports the results obtained for the experiments done on the 3DMatch benchmark. For implementation details, experimental setup, and hyperparameters, see Section 4.5 and Table 4.1. We evaluate the entire 3DMatch test split. For a scene with N_{clouds} fragments, the number of admissible non-consecutive pairs is

$$N_{\text{pairs}} = \binom{N_{\text{clouds}}}{2} - (N_{\text{clouds}} - 1) = \frac{(N_{\text{clouds}} - 1)(N_{\text{clouds}} - 2)}{2}. \tag{5.1}$$

Table 5.1 summarizes the per-scene counts. Each scene contributes between 630 and 2,080 admissible non-consecutive pairs, totaling 11,480 pairs across the eight scenes. However, these counts are an upper bound because, following the 3DMatch protocol, we compute metrics only on the subset whose ground-truth overlap exceeds 30%, so the effective number of evaluated pairs is smaller. In any case, even after this filtering, the sample size remains large, supporting the robustness of the statistics reported below.

We begin with iteration counts, presented in Table 5.2. Using learned features (FCGF) reduces effort in both stages while introducing slightly higher variability. For RANSAC, the overall mean drops from 75.02 to 52.24 iterations ($\approx 30.4\%$ reduction). For ICP, the mean falls from 27.08 to 21.57 ($\approx 20.4\%$ reduction). In exchange, dispersion grows: the aggregate standard deviation (SD) rises from 44.78 to 49.04 for RANSAC and from 6.01 to 8.14 for ICP. The inter-scene SD row (computed from the per-scene means) also slightly increases (RANSAC: 10.72 to 11.14; ICP: 2.01 to 3.59), indicating a modest increase in scene-to-scene fluctuation under FCGF.

At the scene level, FCGF consistently lowers RANSAC means, often substantially (e.g., Hotel 2: 87.94 to 45.70; Hotel 3: 83.77 to 29.12). Two scenes (Study Room and

Scene	Number of Fragments	Possible non-consecutive pair combinations
Kitchen	60	1711
Home 1	60	1711
Home 2	60	1711
Hotel 1	55	1431
Hotel 2	57	1540
Hotel 3	37	630
Study Room	66	2080
Laboratory	38	666
TOTAL	**433**	**11480**

Table 5.1: Per-scene fragment counts and the resulting number of admissible non-consecutive pairs in the full 3DMatch test split.

Laboratory) exhibit RANSAC means close to the geometric baseline (Study Room: 59.52 to 55.47; Laboratory: 60.44 to 63.65), but here the learned pipeline shows much broader spread (SDs 67.71 and 71.43 versus 22.77 and 25.50). This pattern is consistent with RANSAC's confidence bound (4.5): when learned descriptors yield high inlier fractions w, some pairs terminate very early, while harder pairs (low overlap, clutter, or self-similar geometry) still demand many trials, widening the distribution.

For ICP, the reduction is uniform across all scenes (e.g., Kitchen: 24.00 to 17.37; Hotel 1: 24.11 to 17.48; Laboratory: 29.14 to 25.86). The increased ICP SD under FCGF (e.g., Home 1: 6.44 to 8.85; Hotel 2: 5.22 to 8.21) is expected: better global initializations produce fewer iterations on average, but the remaining spread reflects pair difficulty after the global step (surface coverage, normal quality, and local minima in point-to-plane refinement).

In short, learned features reduce iteration budgets by $\approx$30% (RANSAC) and $\approx$20% (ICP) without sacrificing reliability, while concentrating "wins" on easier pairs and leaving a heavier tail of difficult cases. Practically, this means faster convergence most of the time, interspersed with occasional outliers that still require many RANSAC validations or a few extra ICP steps.

Fitness results (Table 5.3) mirror the iteration count speed-ups: the deep-learning pipeline delivers consistently higher alignment quality at both stages. For RANSAC, mean fitness improves from 0.3823 to 0.5468, a gain of +0.1645 (i.e., +16.45 percentage points, or +43.0% relative to the geometric-only baseline). For ICP, mean fitness rises from 0.4419 to 0.5583, a gain of +0.1164 (+11.64 percentage points, or +26.3% relative). Improvements are uniform across all scenes and, for

Scene	RANSAC Iterations				ICP Iterations			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	83.90	35.62	55.54	34.64	24.00	7.52	17.37	6.74
Home 1	70.71	39.16	60.01	59.58	27.04	6.44	18.89	8.85
Home 2	78.21	72.01	48.26	46.54	27.46	5.88	21.18	8.66
Hotel 1	75.66	40.66	60.19	39.49	24.11	7.59	17.48	6.98
Hotel 2	87.94	45.04	45.70	40.63	27.91	5.22	21.51	8.21
Hotel 3	83.77	39.02	29.12	20.88	27.79	5.33	23.53	7.95
Study Room	59.52	22.77	55.47	67.71	29.20	3.17	26.71	5.69
Laboratory	60.44	25.50	63.65	71.43	29.14	3.22	25.86	6.91
TOTAL	**75.02**	**44.78**	**52.24**	**49.04**	**27.08**	**6.01**	**21.57**	**8.14**
Inter-scene SD	–	**10.72**	–	**11.14**	–	**2.01**	–	**3.59**

Table 5.2: Per-scene mean and standard deviation (SD) of iteration counts for global RANSAC and local ICP on the 3DMatch benchmark. Left: geometric-only pipeline (FPFH-based). Right: deep-learning pipeline (FCGF-based). TOTAL: reports dataset-wide means and SDs over all evaluated pairs. Inter-scene SD: the standard deviation of the per-scene means (variability across scenes). Learned features reduce average iterations (RANSAC ≈30%, ICP ≈20%) at the cost of slightly higher dispersion. Settings and protocol as in Section 4.5 and Table 4.1.

example, in Hotel 2 the RANSAC mean jumps from 0.3946 to 0.5816 (+18.70% absolute), and in Home 1 from 0.3280 to 0.5232 (+19.52 absolute).

Another way to view this effect is to compare stages: deep-learning RANSAC alone (54.68%) already exceeds the geometric-only ICP result (44.19%). Consequently, adding ICP on top of the deep-learning initialization yields only a modest extra gain (+1.15 absolute, or ≈ +2.1% relative), whereas ICP contributes a bit more visibly to the geometric-only pipeline (+5.96% absolute, or ≈ +15.6% relative). In any case, this indicates that most of the fitness improvement is achieved by the stronger learned descriptors during global registration, with ICP acting primarily as a light refinement.

Regarding variability, the per-pair standard deviations are similar between pipelines. From the geometric-only to the deep-learning pipeline, an increase from 0.1709 to 0.1792 is seem in RANSAC, and from 0.1684 to 0.1793 in ICP, indicating a slightly broader within-scene spread under the deep-learning setup. Across scenes, the inter-scene SD is comparable for RANSAC (0.0424 and 0.0415) and actually lower for ICP with deep learning (0.0500 and 0.0425), suggesting more consistent performance scene-to-scene after refinement.

Scene	RANSAC Fitness				ICP Fitness			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.4482	0.1806	0.5925	0.1590	0.5043	0.1684	0.5957	0.1586
Home 1	0.3280	0.1656	0.5232	0.1847	0.3714	0.1581	0.5272	0.1838
Home 2	0.3341	0.2085	0.5012	0.2496	0.3833	0.2154	0.5129	0.2493
Hotel 1	0.3968	0.1757	0.5565	0.1617	0.4473	0.1583	0.5595	0.1586
Hotel 2	0.3946	0.1395	0.5816	0.1443	0.4722	0.1420	0.6027	0.1474
Hotel 3	0.4218	0.1511	0.5823	0.1587	0.4967	0.1475	0.6016	0.1662
Study Room	0.3504	0.1365	0.4787	0.1579	0.4080	0.1305	0.4937	0.1590
Laboratory	0.3847	0.1507	0.5581	0.1488	0.4523	0.1409	0.5730	0.1468
TOTAL	**0.3823**	**0.1709**	**0.5468**	**0.1792**	**0.4419**	**0.1684**	**0.5583**	**0.1793**
Inter-scene SD	–	**0.0424**	–	**0.0415**	–	**0.0500**	–	**0.0425**

Table 5.3: Alignment fitness on the 3DMatch benchmark. Means and per-pair standard deviations (SD) are reported per scene for RANSAC and ICP, comparing the geometric-only (FPFH-based) and deep-learning (FCGF-based) pipelines. The deep-learning initialization constantly yields higher fitness across scenes and stages. ICP then provides a modest refinement on top of the learned features.

Table 5.4 reports inlier RMSE (lower is better) for both stages. With deep-learning features, RANSAC's mean inlier RMSE drops from 0.0222 to 0.0180 m, a reduction of 0.0042 m (4.2 mm, or 18.9% increase). After refinement, ICP further reduces the error from 0.0177 to 0.0148 m, a reduction of 0.0029 m (2.9 mm, or 16.4% increase). Per-scene reductions are consistent (e.g., Kitchen 0.0223 to 0.0179 m and Hotel 1 0.0211 to 0.0170 m at the RANSAC stage).

Although the absolute differences are on the millimeter scale, they remain meaningful: higher fitness typically brings in harder correspondences that could increase average error, so the fact that RMSE not only does not rise but actually decreases indicates more accurate local alignment. Notably, while ICP adds only a modest fitness gain on top of the deep-learning initialization (as seen in the previous Table 5.3), it yields a more visible RMSE benefit for the same pipeline (0.0180 to 0.0148 m, i.e., 3.2 mm or a 17.8% increase). Variability also tightens: per-pair SD shrinks from 0.0047 to 0.0032 m at RANSAC and from 0.0045 to 0.0031 m at ICP, and the inter-scene SD is lower with deep learning, suggesting more stable performance across scenes.

Additionally, we report runtime broken into three macro stages: preprocessing (including feature extraction), RANSAC, and ICP. The obtained per-iteration time measured in our setup was ≈ 419 ms for RANSAC and ≈ 147 ms for ICP. Combining

Scene	RANSAC Inlier RMSE				ICP Inlier RMSE			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.0223	0.0045	0.0179	0.0025	0.0178	0.0045	0.0149	0.0025
Home 1	0.0222	0.0049	0.0172	0.0032	0.0178	0.0045	0.0143	0.0032
Home 2	0.0248	0.0043	0.0201	0.0039	0.0213	0.0052	0.0169	0.0043
Hotel 1	0.0211	0.0050	0.0170	0.0027	0.0169	0.0046	0.0138	0.0023
Hotel 2	0.0229	0.0044	0.0186	0.0033	0.0181	0.0041	0.0146	0.0031
Hotel 3	0.0218	0.0045	0.0174	0.0032	0.0168	0.0042	0.0143	0.0024
Study Room	0.0208	0.0042	0.0175	0.0030	0.0161	0.0037	0.0147	0.0024
Laboratory	0.0213	0.0045	0.0185	0.0033	0.0170	0.0037	0.0152	0.0027
TOTAL	**0.0222**	**0.0047**	**0.0180**	**0.0032**	**0.0177**	**0.0045**	**0.0148**	**0.0031**
Inter-scene SD	–	**0.0013**	–	**0.0010**	–	**0.0016**	–	**0.0010**

Table 5.4: Inlier RMSE (in meters) on the 3DMatch benchmark. Means and per-pair standard deviations are reported per scene for RANSAC and ICP, comparing the geometric-only (FPFH) and deep-learning (FCGF) pipelines. Deep learning yields uniformly lower errors at both stages. ICP further reduces RMSE on top of the learned initialization. The final row ("Inter-scene SD") summarizes the variability of scene-wise means across scenes.

these per-iteration times with the observed iteration totals (geometric-only: 440,629 RANSAC and 160,227 ICP; deep-learning: 113,253 RANSAC and 45,956 ICP) yields the stage totals summarized in Table 5.5. In addition, the preprocessing stage (which includes feature extraction) was substantially shorter with the deep-learning pipeline (3.69 min vs. 19.12 min).

This outcome is consistent with prior reports for FCGF, [5] shows a per-feature extraction time of 0.009 ms for FCGF versus 0.032 ms for FPFH on 3DMatch, indicating that FCGF is markedly faster than classical handcrafted descriptors. These results support our observation that the deep-learning pipeline reduces end-to-end time primarily by (i) cutting RANSAC and ICP iterations and (ii) using a faster feature extractor.

Finally, we evaluate recall and precision under the official 3DMatch geometric–registration protocol. We use the reference evaluation script from the 3DMatch toolbox [72]. As summarized in Section 2.4, a registration is counted as correct if the rotation and translation errors satisfy RRE $\leq 15^\circ$ and RTE ≤ 0.30 m, in relation to the ground-truth, as shown in (2.19). After labeling each predicted alignment as true/false positive (TP/FP) and each missed alignment as false negative (FN), we compute

Stage	Time (minutes)	
	Geometric-only	**Deep-learning**
Preprocessing	19.12	3.69
RANSAC	3075.29	790.43
ICP	393.50	112.86
TOTAL	**3487.91**	**906.98**

Table 5.5: End-to-end runtime on 3DMatch by stage. Times aggregate all evaluated pairs. RANSAC and ICP totals come from multiplying the measured per-iteration costs (≈ 419 ms and ≈ 147 ms, respectively) by the observed iteration counts of each pipeline. Preprocessing includes feature extraction. Overall, the deep-learning pipeline is markedly faster due to fewer iterations and a faster feature extractor, consistent with FCGF's reported speed advantage, as shown in [5].

$$\text{Recall} = \frac{\sum \text{TP}}{\sum(\text{TP} + \text{FN})}, \qquad \text{Precision} = \frac{\sum \text{TP}}{\sum(\text{TP} + \text{FP})} \tag{5.2}$$

Intuitively, recall measures, of all pairs that are possible to be aligned, how many did our pipelines actually recover. Meanwhile, precision measures, of the alignments we reported, how often were they actually correct (i.e., they are pairs that are indeed feasible to be aligned).

Table 5.6 shows per-scene and overall results. The deep-learning pipeline achieves a marked precision gain (TOTAL: 0.1551 to 0.4207, a +0.2656 absolute gain), while recall also improves modestly (TOTAL: 0.7107 to 0.7430, a +0.0323 absolute gain). Scene-wise, precision increases consistently, and often substantially (e.g., Hotel 1: 0.3390 to 0.7673; Kitchen: 0.3736 to 0.6848), indicating far fewer false positives. Recall improves in five of eight scenes (Kitchen, Home 2, Hotel 2, Study Room, Laboratory), dips slightly in three (Home 1, Hotel 1, Hotel 3), yet ends higher overall. This pattern aligns with earlier metrics: stronger learned descriptors make global alignment more reliable and filter out many spurious matches produced by FPFH, boosting precision without sacrificing recall.

For qualitative reference, Figure 5.1 displays a sample alignment obtained by both pipelines. Notice that, in this sample, DL also obtained higher fitness with lower error. Therefore, this visual analysis illustrates the benefits we observed during the previous results.

Scene	Geometric-only		Deep-learning	
	Recall	**Precision**	**Recall**	**Precision**
Kitchen	0.8129	0.3736	0.8708	0.6848
Home 1	0.8019	0.1400	0.7547	0.5031
Home 2	0.6164	0.1010	0.6478	0.3552
Hotel 1	0.8791	0.3390	0.8516	0.7673
Hotel 2	0.7564	0.0672	0.8077	0.2121
Hotel 3	0.7692	0.0522	0.7308	0.1367
Study Room	0.6496	0.1192	0.7692	0.4390
Laboratory	0.4000	0.0486	0.5111	0.2674
TOTAL	**0.7107**	**0.1551**	**0.7430**	**0.4207**

Table 5.6: Per-scene and overall recall/precision computed with the official 3DMatch toolbox [72], using the protocol thresholds RRE $\leq 15°$ and RTE $\leq 0.30\,\text{m}$. The deep-learning pipeline substantially increases precision across all scenes and yields a modest overall recall gain.

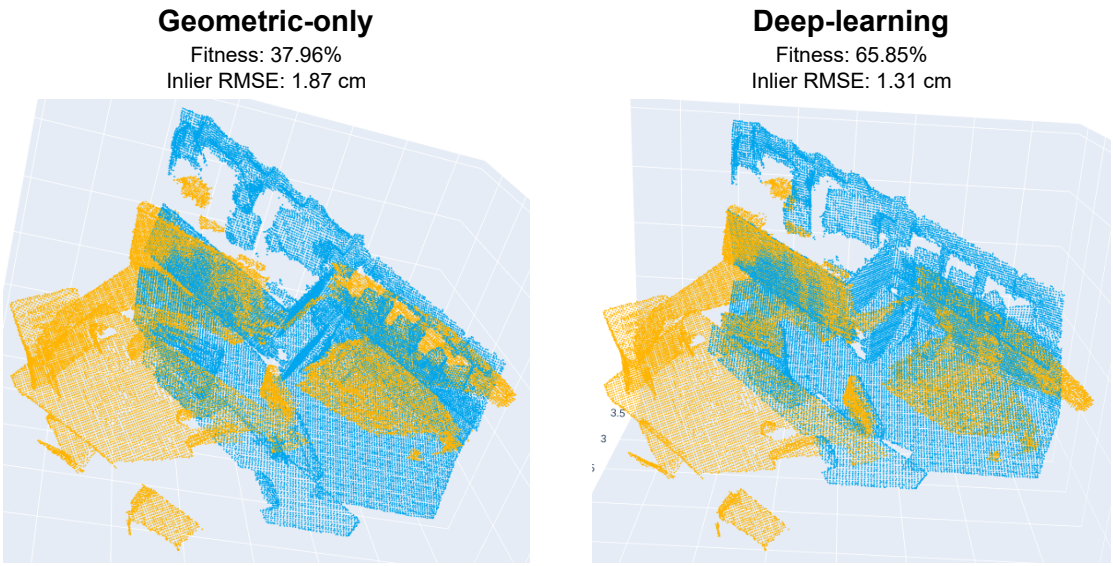


Figure 5.1: Sample alignment obtained for the 3DMatch benchmark. The yellow cloud is the source, and the blue one is the target. Notice that higher fitness and lower inlier RMSE visually translates to tighter alignments, where the overlap of both clouds is more precise and the gaps between them are smaller.

5.2 Augmented 3DMatch Benchmark

We evaluate four scenarios, each adding noise on top of the previous: No Noise (control), Gaussian (with variable variance), Gaussian+Spikes, and Gaussian+Spikes+Pepper. Table 5.7 summarizes both pipelines across all scenarios (means, per-pair SD, and inter-scene SD). Full per-scene tables are in Appendix B.

Metric	Statistic	No Noise		Gaussian		Gaussian +Spikes		Gaussian +Spikes +Pepper	
		GO	DL	GO	DL	GO	DL	GO	DL
RANSAC Iterations	Mean	75.02	52.24	37.63	24.59	37.22	20.13	37.30	5.44
	SD	44.78	49.04	18.50	133.63	18.25	106.65	18.50	30.86
	IS-SD	10.72	11.14	5.09	13.35	5.06	12.78	4.92	2.73
ICP Iterations	Mean	27.08	21.57	30.00	30.00	30.00	30.00	30.00	29.99
	SD	6.01	8.14	0.10	0.17	0.17	0.12	0.11	0.43
	IS-SD	2.01	3.59	0.00	0.01	0.00	0.01	0.00	0.01
RANSAC Fitness	Mean	0.3823	0.5468	0.4279	0.5015	0.4454	0.5103	0.4444	0.4606
	SD	0.1709	0.1792	0.1459	0.1892	0.1445	0.1869	0.1471	0.1775
	IS-SD	0.0424	0.0415	0.0460	0.0555	0.0486	0.0491	0.0477	0.0447
ICP Fitness	Mean	0.4419	0.5583	0.4836	0.5505	0.4968	0.5557	0.4954	0.5181
	SD	0.1684	0.1793	0.1600	0.1966	0.1579	0.1938	0.1588	0.1970
	IS-SD	0.0500	0.0425	0.0514	0.0624	0.0529	0.0559	0.0517	0.0540
RANSAC Inlier RMSE	Mean	0.0222	0.0180	0.0223	0.0212	0.0233	0.0220	0.0233	0.0231
	SD	0.0047	0.0032	0.0019	0.0025	0.0019	0.0027	0.0019	0.0020
	IS-SD	0.0013	0.0010	0.0005	0.0006	0.0004	0.0005	0.0004	0.0004
ICP Inlier RMSE	Mean	0.0177	0.0148	0.0209	0.0195	0.0218	0.0203	0.0218	0.0212
	SD	0.0045	0.0031	0.0024	0.0030	0.0024	0.0031	0.0024	0.0025
	IS-SD	0.0016	0.0010	0.0007	0.0009	0.0006	0.0008	0.0006	0.0010

Table 5.7: Summary statistics (mean, per-pair standard deviation, and inter-scene standard deviation) over all scenes for each metric and noise setting, comparing geometric-only (GO) and deep-learning (DL) pipelines. IS-SD: inter-scene SD (variability of scene-wise means).

Across all noise settings, the deep-learning pipeline needs fewer RANSAC iterations, and the gap widens as noise increases. This can look counterintuitive, but it matches RANSAC's iteration bound (4.5): under noise, the feature matching stage becomes more conservative, pruning weak or ambiguous matches, and yielding a smaller but cleaner correspondence set with a higher inlier fraction w. Therefore,

looking at 4.5, the higher the w, the smaller N_{RANSAC} becomes. And this drop is not linear because of the exponent m (here, 4). So, even modest gains in w translate into large iteration drops. Furthermore, another consequence of RANSAC's runtime being very sensitive to w is that, with more noise, the inlier fraction varies a lot from scene to scene. So, some scenes finish almost immediately while others take longer, which explains the larger standard deviations.

For ICP, most noisy runs hit the configured cap of 30 iterations, hence the saturation in the means. We kept this cap intentionally (to emulate limited compute). Even when iteration counts match, the DL pipeline still ends with better alignments because its global initialization places ICP in a more favorable basin of attraction.

Fitness remains higher with DL in every scenario and at both stages. Notably, DL RANSAC alone already matches or exceeds GO ICP across the board, highlighting that most of the gain comes from better descriptors and global alignments, with ICP acting as a light polish. Inlier RMSE follows the same pattern: DL yields lower errors at both stages for all noise settings. As expected, absolute errors rise when we add spikes and pepper, since we are corrupting and removing points. Learned features mitigate this effect, but cannot eliminate it entirely.

A small observation: the "No Noise" control is not perfectly clean, since 3DMatch fragments already contain real sensor artifacts. So it is plausible (and observed) that some metrics under synthetic Gaussian look close to, or occasionally slightly better than, the control.

We also recomputed recall and precision with the official 3DMatch evaluation code [72] for all noise settings (Table 5.8). The DL pipeline consistently delivers substantially higher precision than GO under added noise (fewer false positives), and it also improves recall for the Gaussian and Gaussian+Spikes cases. Under the heaviest corruption (Gaussian+Spikes+Pepper), recall degrades sharply for both pipelines and the DL advantage in recall disappears, since pepper dropout reduces overlap and removes geometry needed to form stable correspondences. However, the DL pipeline still retains a precision edge, meaning that when it obtains a match, it is more often correct, while GO claims more matches, but they are more frequently incorrect.

5.3 Cross-domain Benchmark

Because this benchmark spans many objects, in this section we report the overall result summaries for both the single-object and full-scene settings. All the complete per-object tables are provided in Appendix C.

Beginning with the single-object setting (Table 5.9) shows that the deep-learning pipeline cuts RANSAC iterations by more than half on average (108.29 to 45.19),

Noise	Pipeline	Metric	Kitchen	Home 1	Home 2	Hotel 1	Hotel 2	Hotel 3	Study Room	Laboratory	TOTAL
No Noise	GO	Recall	0.8129	0.8019	0.6164	0.8791	0.7564	0.7692	0.6496	0.4000	**0.7107**
		Precision	0.3736	0.1400	0.1010	0.3390	0.0672	0.0522	0.1192	0.0486	**0.1551**
	DL	Recall	0.8708	0.7547	0.6478	0.8516	0.8077	0.7308	0.7692	0.5111	**0.7430**
		Precision	0.6848	0.5031	0.3552	0.7673	0.2121	0.1367	0.4390	0.2674	**0.4207**
Gaussian	GO	Recall	0.1537	0.1887	0.1635	0.1813	0.1282	0.1923	0.1068	0.1111	**0.1532**
		Precision	0.0421	0.0122	0.0155	0.0236	0.0066	0.0080	0.0126	0.0079	**0.0160**
	DL	Recall	0.3073	0.3396	0.2893	0.3626	0.2179	0.2308	0.2393	0.2889	**0.2845**
		Precision	0.2222	0.0807	0.0564	0.1828	0.0335	0.0324	0.0858	0.0760	**0.0962**
Gaussian +Spikes	GO	Recall	0.1448	0.1132	0.1572	0.1374	0.1667	0.1154	0.0769	0.0667	**0.1223**
		Precision	0.0387	0.0073	0.0148	0.0176	0.0122	0.0088	0.0088	0.0046	**0.0131**
	DL	Recall	0.3163	0.3208	0.2704	0.3297	0.2179	0.2308	0.2350	0.3111	**0.2790**
		Precision	0.2168	0.0756	0.0521	0.1648	0.0332	0.0314	0.0820	0.0824	**0.0923**
Gaussian +Spikes +Pepper	GO	Recall	0.1336	0.1509	0.1447	0.1538	0.1026	0.1154	0.0427	0.0667	**0.1138**
		Precision	0.0358	0.0096	0.0136	0.0198	0.0052	0.0048	0.0049	0.0046	**0.0123**
	DL	Recall	0.0200	0.0660	0.0377	0.0165	0.0128	0.0385	0.0214	0.0222	**0.0294**
		Precision	0.0455	0.0409	0.0132	0.0234	0.0053	0.0112	0.0185	0.0127	**0.0213**

Table 5.8: Recall and precision computed with the official 3DMatch toolbox [72] (success if RRE $\leq 15^\circ$ and RTE $\leq 0.30\,\mathrm{m}$). The deep-learning pipeline maintains a strong precision advantage under added noise and improves recall for Gaussian and Gaussian+Spikes. Under the heaviest corruption (Gaussian+Spikes+Pepper), recall drops for both pipelines, and DL still keeps higher precision.

with a corresponding drop in SD (102.01 to 47.56). This mirrors what we saw on 3DMatch: stronger correspondences raise the inlier fraction w and shrink RANSAC's expected trials. ICP iterations are similar across pipelines (27.15 vs. 26.32 on average) and sit close to our default cap of 30, indicating that many pairs terminate near the limit in this more challenging, cross-domain setting.

At the global stage, DL improves fitness (0.3666 to 0.4408) and lowers inlier RMSE (3.08 mm to 2.63 mm). After local refinement, the pipelines end up with comparable ICP fitness (0.4949 vs. 0.5074) and essentially identical ICP RMSE (both 2.28 mm). In other words, most of the advantage shows up where it matters most for robust initialization (RANSAC), while ICP then pulls both solutions towards a similar local optimum.

Now, for the full-scene setting (Table 5.10 shows that, once again, the deep-learning obtained lower RANSAC iteration counts on average (33.31 to 7.77), and lower variability (SD 35.57 to 8.99). ICP iteration performance is similar across pipelines, with counts remaining close (27.83 and 28.31), suggesting that many pairs again refine near the iteration cap (defined in the convergence criteria).

In terms of alignment quality, the two pipelines achieve comparable RANSAC fitness (0.3689 vs. 0.3702) and similar RANSAC inlier RMSE (2.73 mm vs. 2.78 mm). After refinement, ICP fitness and RMSE remain close as well (fitness 0.4948 vs. 0.4924; RMSE 2.30,mm vs. 2.33,mm), indicating that the final local solutions are generally similar in this setting.

Therefore, compared to the previous single-object setting results, where the

Metric		Geometric-only	Deep-learning
RANSAC	Mean	108.29	45.19
Iterations	SD	102.01	47.56
ICP	Mean	27.15	26.32
Iterations	SD	5.46	6.20
RANSAC	Mean	0.3666	0.4408
Fitness	SD	0.1214	0.1307
ICP	Mean	0.4949	0.5074
Fitness	SD	0.1168	0.1199
RANSAC	Mean	0.00308	0.00263
Inlier RMSE	SD	0.00030	0.00025
ICP	Mean	0.00228	0.00228
Inlier RMSE	SD	0.00025	0.00024

Table 5.9: Cross-domain benchmark (object-level dataset) under single-object target setting. Means and per-pair standard deviations (SD) over all objects. Learned features (DL, FCGF-based) substantially reduce global RANSAC iterations and improve RANSAC fitness/RMSE. After refinement, both pipelines converge to similar ICP RMSE.

deep-learning pipeline had a more notable improvement in the global-stage, here the main difference is only the drop in RANSAC iterations, while fitness and RMSE are nearly equal across pipelines.

Now that we understood the quality of the successful alignments, we analyze how often they are obtained. Using the procedure described in Section 4.7, Table 5.11 reports the average estimated recall. In the single-object setting, the recalls are 69.28% (geometric-only) and 67.81% (deep-learning), i.e., a small advantage for the classical pipeline. Given the cross-domain shift (CAD to acquired clouds) and the fact that FCGF is not trained on SuctionNet, this suggests that the learned descriptors remain reasonably stable in a object-level benchmark. In the full-scene setting, recall drops substantially for both pipelines (13.36% for geometric-only and 5.65% for deep-learning), reflecting the increased difficulty of registering an object model against a cluttered scene.

Table 5.12 presents the per-object recall. In the single-object setting, DL improves most objects (19 out of 29, 65.5%), with an average gain of 5.5 percentage points among the improved objects. However, we previously observed that the overall average recall of GO was higher. This means that, for objects where DL had higher recall, the margin was typically small, whereas GO, despite achieving higher recall on fewer objects, often did so by a more significant margin. This suggests that DL may be more consistent across objects, while GO performs very well on a

Metric		Geometric-only	Deep-learning
RANSAC	Mean	33.31	7.77
Iterations	SD	35.57	8.99
ICP	Mean	27.83	28.31
Iterations	SD	4.72	4.23
RANSAC	Mean	0.3689	0.3702
Fitness	SD	0.1223	0.1464
ICP	Mean	0.4948	0.4924
Fitness	SD	0.1176	0.1210
RANSAC	Mean	0.00273	0.00278
Inlier RMSE	SD	0.00027	0.00027
ICP	Mean	0.00230	0.00233
Inlier RMSE	SD	0.00029	0.00028

Table 5.10: Cross-domain benchmark (object-level dataset) under full-scene target setting. Means and per-pair standard deviations (SD) over all objects. Learned features (DL, FCGF-based) substantially reduce global RANSAC iterations and improve RANSAC fitness/RMSE. After refinement, both pipelines converge to similar ICP RMSE.

Setting	Geometric-Only	Deep-Learning
Single-object	69.28	67.81
Full-scene	13.36	5.65

Table 5.11: Average overall object recall for geometric-only and deep-learning pipelines across settings.

smaller subset. These cases where GO performs better may correspond to objects with strong symmetries or sparse and ambiguous geometry (e.g., square and flat objects without notable features), where descriptor repeatability and consensus sampling become sensitive under cross-domain conditions. Fine-tuning the deep-learning feature extractor backbone on target-domain data could possibly reduce these gaps. Additionally, once again, the estimated recalls in the full-scene setting are significantly lower for both pipelines across all objects (higher difficulty).

To further analyze these results, we computed the recall margin between the two pipelines for each object, defined as $\Delta R_{\text{obj}} = R_{\text{obj}}^{\text{DL}} - R_{\text{obj}}^{\text{GO}}$. We then ranked objects by ΔR_{obj} and selected three groups: top 3 (largest DL advantage), middle 3 (near-ties), and bottom 3 (largest GO advantage). Table 5.13 reports the summary metrics for these groups in the single-object setting, while Table 5.14 reports the same analysis for the full-scene setting.

Under single-object setting, the top and middle groups exhibit higher fitness

for DL and generally fewer RANSAC iterations, whereas the bottom group shows reduced DL fitness, which is consistent with its lower recall on those objects. Therefore, this extra analysis indicates that, as expected, objects that had lower recall represent more challenging samples, so even when the pipeline is able to register it, the alignment quality is generally low. In the full-scene setting, differences in fitness and inlier RMSE across groups are smaller and less consistent, reflecting the overall increased difficulty of this setting.

Figure 5.2 shows a representative alignment on the SuctionNet cross-domain task. The red cloud is the CAD model (source), overlaid on the target after ICP. In this instance, the deep-learning pipeline produces a visibly tighter fit, with substantially higher overlap and a lower inlier error than the geometric-only baseline.

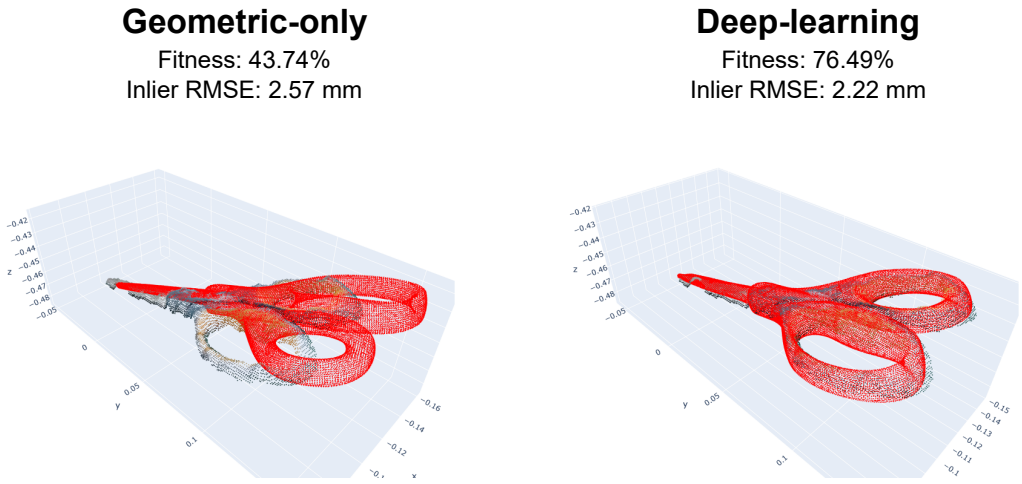


Figure 5.2: The red cloud is the CAD model (source), and poses shown are after ICP refinement. Deep-learning (FCGF) attains higher fitness and lower inlier error than the geometric-only (FPFH) pipeline (DL: 76.49% fitness, 2.22 mm inlier RMSE; GO: 43.74%, 2.57 mm).

Finally, to assess the behavior of both pipelines when using cropped CAD models as the source, we selected the "middle" objects (i.e., those with near-ties in recall between GO and DL) and re-ran the evaluation. The results are summarized in Tables 5.15 and 5.16 for the single-object and full-scene target settings, respectively.

Overall, the deep-learning pipeline remains substantially more efficient at the global stage, requiring far fewer RANSAC iterations in both settings (77.26→26.34 in the single-object case, and 36.95→9.10 in the full-scene case), with a corresponding reduction in variability. However, this improvement in efficiency does not translate into a consistent quality gain for the estimated transformations. In the single-object setting, GO achieves slightly higher RANSAC fitness (0.6148 vs.

0.5950), while both methods converge to nearly identical refinement results (ICP iterations, fitness and inlier RMSE). In the full-scene setting, RANSAC fitness is comparable (0.6053 vs. 0.6107) and RANSAC inlier RMSE is nearly identical, but GO yields better refinement quality, with higher ICP fitness (0.8248 vs. 0.7605) and a slightly lower ICP inlier RMSE (0.00179 vs. 0.00186). These results suggest that, for the selected near-tie objects, learned features mainly improve efficiency (less iterations), while final alignment quality remains similar or can even favor the geometric pipeline after ICP refinement.

For qualitative results, Figures 5.3 and 5.4 display examples of alignments obtained using the cropped CAD as sources under the full-scene and single-object settings, respectively.

Object	Estimated Recall (%)			
	Full-scene		Single-object	
	GO	DL	GO	DL
30	22.17	17.48	98.24	98.83
27	5.57	4.59	90.82	93.85
57	16.99	12.79	89.06	93.75
12	65.23	43.16	85.55	87.30
52	25.78	3.91	90.82	97.27
31	20.72	13.67	80.91	85.90
10	6.64	0.78	97.66	95.51
19	0.00	1.01	81.01	81.41
21	30.80	1.84	84.56	82.49
61	30.27	5.86	83.20	76.95
23	10.74	4.39	87.70	91.60
28	2.28	0.60	62.70	64.14
15	12.65	4.12	85.49	92.08
49	16.02	3.61	63.18	79.79
18	15.02	17.42	62.98	72.74
16	25.39	14.26	70.90	66.89
71	14.06	0.20	76.56	92.48
38	0.98	1.95	41.02	46.09
37	5.67	0.62	45.98	47.32
64	1.17	5.08	60.16	39.84
6	8.40	0.78	48.83	42.19
63	14.65	0.78	86.82	16.21
39	11.72	1.95	42.77	35.55
8	8.17	0.79	80.22	80.51
62	2.34	0.00	30.76	40.82
67	2.93	0.00	42.38	34.08
9	4.79	1.07	41.50	21.19
41	0.39	0.00	19.34	22.27
59	2.15	1.37	25.78	35.16

Table 5.12: Estimated per-object recalls for the cross-domain benchmark. Deep-learning pipeline (DL) improves most objects, but a few remain challenging (e.g., #63, #64), likely due to high symmetry or sparse geometry.

Metric	Statistic	Bottom 3		Middle 3		Top 3	
		GO	DL	GO	DL	GO	DL
RANSAC Iterations	Mean	90.03	14.08	44.11	29.52	89.09	47.85
	SD	68.97	11.27	53.73	43.43	69.93	39.67
ICP Iterations	Mean	28.12	28.08	28.53	28.26	25.89	24.05
	SD	4.24	4.16	3.84	4.41	6.15	7.13
RANSAC Fitness	Mean	0.3084	0.2436	0.3688	0.4307	0.3307	0.4358
	SD	0.1024	0.0773	0.1167	0.1296	0.1351	0.1386
ICP Fitness	Mean	0.4406	0.3965	0.4836	0.5007	0.5097	0.5086
	SD	0.0819	0.0774	0.1160	0.1297	0.1295	0.1222
RANSAC Inlier RMSE	Mean	0.00304	0.00274	0.00309	0.00260	0.00313	0.00269
	SD	0.00019	0.00021	0.00024	0.00031	0.00041	0.00021
ICP Inlier RMSE	Mean	0.00214	0.00220	0.00233	0.00231	0.00237	0.00235
	SD	0.00019	0.00022	0.00035	0.00033	0.00021	0.00018

Table 5.13: Single-object setting: results over objects ranked by recall margin.

Metric	Statistic	Bottom 3		Middle 3		Top 3	
		GO	DL	GO	DL	GO	DL
RANSAC Iterations	Mean	33.18	13.74	35.19	4.11	7.62	7.33
	SD	43.43	13.29	17.75	4.54	7.64	8.92
ICP Iterations	Mean	27.88	28.94	29.25	29.56	29.58	29.29
	SD	4.67	3.51	2.66	2.28	2.26	3.00
RANSAC Fitness	Mean	0.4115	0.4655	0.3361	0.3170	0.3859	0.3808
	SD	0.1295	0.1516	0.0841	0.1200	0.1260	0.1376
ICP Fitness	Mean	0.5319	0.5451	0.4753	0.4946	0.5132	0.5022
	SD	0.0894	0.0929	0.1287	0.1247	0.0676	0.0715
RANSAC Inlier RMSE	Mean	0.00261	0.00261	0.00279	0.00288	0.00273	0.00273
	SD	0.00032	0.00032	0.00022	0.00017	0.00024	0.00028
ICP Inlier RMSE	Mean	0.00210	0.00211	0.00241	0.00241	0.00225	0.00224
	SD	0.00022	0.00021	0.00024	0.00026	0.00016	0.00020

Table 5.14: Full-scene setting: results over objects ranked by recall margin.

Metric		Geometric-only	Deep-learning
RANSAC	Mean	77.26	26.34
Iterations	SD	79.85	39.51
ICP	Mean	26.78	26.58
Iterations	SD	5.98	6.06
RANSAC	Mean	0.6148	0.5950
Fitness	SD	0.1944	0.2154
ICP	Mean	0.6773	0.6769
Fitness	SD	0.2099	0.2097
RANSAC	Mean	0.00253	0.00248
Inlier RMSE	SD	0.00034	0.00038
ICP	Mean	0.00211	0.00210
Inlier RMSE	SD	0.00042	0.00041

Table 5.15: Cropped CAD test under single-object target setting. Means and per-pair standard deviations (SD) over all objects.

Metric		Geometric-only	Deep-learning
RANSAC	Mean	36.95	9.10
Iterations	SD	23.47	12.37
ICP	Mean	28.25	27.27
Iterations	SD	4.30	5.63
RANSAC	Mean	0.6053	0.6107
Fitness	SD	0.1998	0.2299
ICP	Mean	0.8248	0.7605
Fitness	SD	0.1625	0.2044
RANSAC	Mean	0.00245	0.00242
Inlier RMSE	SD	0.00040	0.00040
ICP	Mean	0.00179	0.00186
Inlier RMSE	SD	0.00033	0.00041

Table 5.16: Cropped CAD test under full-scene target setting. Means and per-pair standard deviations (SD) over all objects.

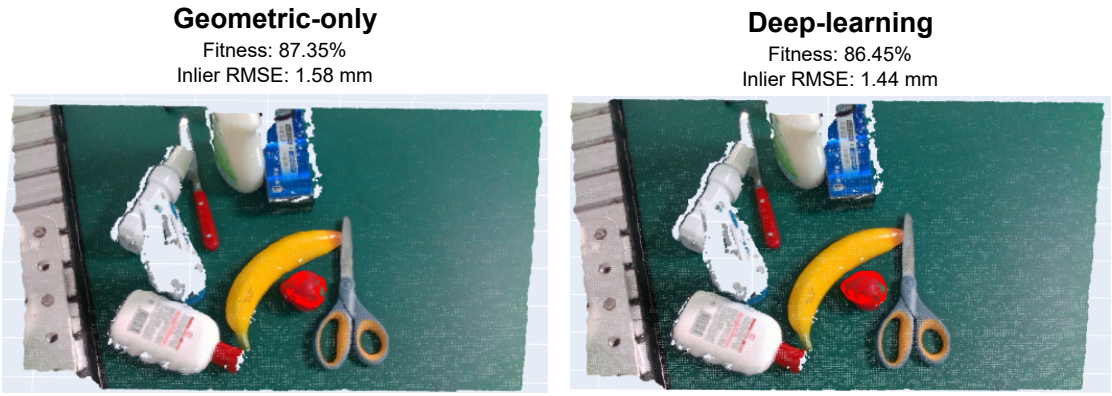


Figure 5.3: Example of an alignment obtained with the cropped CAD model in the full-scene setting. The red cloud is the cropped CAD model (source), and poses shown are after ICP refinement.

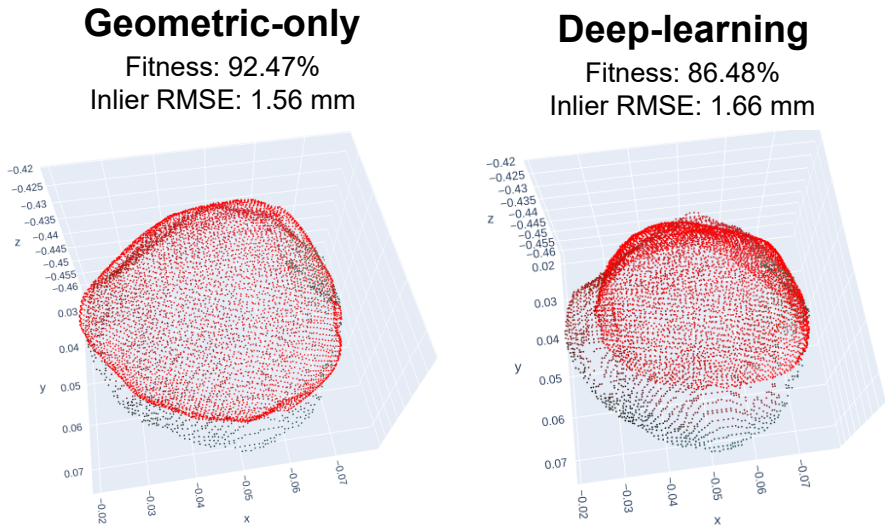


Figure 5.4: Example of an alignment obtained with the cropped CAD model in the single-object setting. The red cloud is the cropped CAD model (source), and poses shown are after ICP refinement.

Chapter 6

Conclusions & Future Work

This thesis compared a geometric-only registration pipeline (FPFH, RANSAC, ICP) to a deep-learning variant that swaps FPFH for FCGF while keeping the same global and local stages. Both were implemented in Open3D with practical instrumentation (iteration counters and timers) and evaluated on three settings: the standard 3DMatch benchmark, a noise-augmented 3DMatch, and a cross-domain object benchmark adapted from SuctionNet.

Across 3DMatch, learned features consistently reduce computational cost and improve quality. In aggregate, RANSAC iterations drop by roughly a third and ICP by about a fifth, while fitness increases at both stages and inlier RMSE decreases (Sections 4.5 and 5). End-to-end time shrinks markedly thanks to fewer RANSAC/ICP iterations and faster feature extraction, consistent with prior reports of FCGF's lower feature-extraction latency on 3DMatch [5]. In short: stronger correspondence sets upstream translate into faster consensus, lighter refinement, and better final alignments.

On the augmented 3DMatch benchmark (Section 5.2), the same trend holds under injected corruption (Gaussian, spikes, pepper). Learned features remain more robust: RANSAC converges in fewer trials and the pipeline reaches higher fitness and lower RMSE at both stages. As noise increases, RANSAC variability widens—traceable to the inlier fraction w in the confidence bound (Eq. 4.5). ICP often hits the configured cap of 30 iterations in the heaviest-noise settings; even then, the deep-learning pipeline typically starts ICP in a better convergence basin, yielding lower final error. Recall and precision (official 3DMatch protocol) degrade for both as noise grows, but the deep variant maintains a clear precision advantage except in the most severe dropout case (Gaussian+Spikes+Pepper), where both struggle.

In the cross-domain benchmark (Section 5.3), with CAD sources and single-view RGB-D targets, we added a lightweight centering step that re-centers both clouds before feature computation and registration to stabilize matching under

large camera–object offsets. With that in place, the learned pipeline again reduced RANSAC iterations and improved RANSAC fitness/RMSE; ICP performance between pipelines was similar on average. Estimated recall improved modestly overall, with object-to-object variability expected in a domain not seen during training. The takeaway: FCGF generalizes reasonably well to CAD→RGB-D alignment when paired with simple preconditioning and a classical RANSAC and ICP stack.

There are a few caveats worth flagging. (i) On SuctionNet we lack ground-truth transforms, so we used Open3D's fitness as a pragmatic overlap proxy to estimate recall, so while consistent and comparable, it is still a proxy. (ii) Our ICP cap (30 iterations) was chosen to emulate time constraints, thus in high-noise runs, many pairs exit early due to the cap rather than full convergence. (iii) Normal orientation was not enforced, which is acceptable for our losses and checkers but may marginally degrade accuracy in some scenes. (iv) Reported runtimes combine measured per-iteration costs with observed iteration counts on a fixed hardware/software stack. Absolute numbers will definitely vary across platforms.

The results suggest a few clear directions:

1. Replace or complement feature-based RANSAC with robust global solvers and outlier-tolerant consensus (e.g., more aggressive correspondence filtering, graph consistency checks, or robust cost variants) to stabilize the heavy-noise tails observed in the augmented benchmark.

2. Swap point-to-plane ICP for more robust refinements (trimmed, GICP, robust losses) and tune the iteration budget adaptively (raise the cap only for hard pairs, using fitness/gradient cues).

3. Fine-tune FCGF on CAD↔RGB-D pairs with synthetic-to-real augmentations (dropout along viewing rays, mixed-scale Gaussian/spikes) to narrow the residual domain gap observed in per-object recall on SuctionNet.

4. Add a coarse-to-fine registration schedule (larger voxels for coarse consensus, then finer voxels for local refinement) to reduce search while improving stability on repetitive geometry.

In sum, replacing handcrafted features with FCGF within a classical RANSAC and ICP stack yields consistent gains in iteration economy, alignment quality, and end-to-end runtime across both standard and stressed settings. With modest upgrades (more robust global estimation, more forgiving local refinement, and light domain adaptation), the same backbone should generalize to broader cross-domain registration problems with minimal engineering overhead.

Appendix A

Augmented 3DMatch Benchmark under MinkowskiEngine Setup

While running the deep-learning pipeline with our noise generator, we observed intermittent segmentation-fault crashes (SIGSEGV) during the RANSAC stage of the downstream subprocess (i.e., the script that executes FCGF's feature extraction and initial global registration). A segmentation fault (SIGSEGV) is a low-level operating-system signal raised when a process attempts to access memory it is not permitted to read or write (e.g., invalid pointer, double free, or heap metadata corruption). The failure was nondeterministic: identical runs (same parameters and seeds) sometimes completed and sometimes crashed at different points.

To localize the issue we: (i) verified argument passing from the main launcher to the subprocess (the `noise_kwargs` dictionary expanded into CLI flags), ruling out malformed inputs; (ii) enabled noise modes incrementally (first zero noise, then Gaussian only, spikes only, and salt-and-pepper), and found that failures were triggered only when pepper (point dropout) was active; (iii) guarded the dropout to prevent empty or near-empty clouds by clamping the effective removal ratio to keep a minimum number of points. Crashes persisted, ruling out out-of-bounds indexing and empty-cloud corner cases.

In our investigation, we found that the intermittent crashes were caused by thread oversubscription triggered by the pepper (dropout) augmentation. Libraries in our stack, MinkowskiEngine and parts of Open3D, use OpenMP (a common parallelization API), which by default spawns one worker per Central Processing Unit (CPU) core unless constrained via the environment variable `OMP_NUM_THREADS`. When pepper is enabled, our code compacts point arrays in place, forcing NumPy to reallocate and copy buffers and, in some cases, calling into multi-threaded

Basic Linear Algebra Subprograms (BLAS) backends. These two behaviors occur together: OpenMP creates many workers while array reallocation/freeing also engages multi-threaded routines, so dozens of threads simultaneously allocate, resize, and free large buffers. Their concurrent calls contend on the same allocator metadata, depending on precise timing. This racing condition, can corrupt the heap and surface as segmentation faults (SIGSEGV) deep in native code. The crashes appear only with pepper and only intermittently because they depend on whether enough threads hit the allocator at the same instant.

We eliminated the failures by enforcing a single-threaded environment for the augmented benchmarks. This must be set before importing heavy libraries (such as PyTorch) or spawning subprocesses so children inherit the environment. Either in the shell:

```
export OMP_NUM_THREADS = 1
```

or, equivalently at the very top of the Python entry point:

```
import os
os.environ["OMP_NUM_THREADS"] = "1"
```

Restricting execution to a single thread removes parallelism and, consequently, increases runtime time. We quantified this overhead by running on the whole benchmark under two settings: default (multi-threaded) vs. single-threaded, both with no noise and with Gaussian plus spikes. As presented in Table A.1, relative to default, total runtime increased by 14.11% with no noise and by 4.25% with Gaussian plus spikes. Pepper was excluded from this comparison because it runs only in the single-thread configuration by design.

Because the aim of the augmented benchmark is robustness (not throughput), we report registration iteration counts (RANSAC and ICP) and estimate noisy-case runtimes using the per-iteration timing from the default configuration multiplied by the measured iteration counts. For experiments that do not involve pepper, we retain the regular multi-threaded setup to accelerate testing. This single-thread constraint is specific to our simulation setup (pepper applied post-fusion within a sparse-tensor pipeline). In practice, real typical deployments without this augmentation pattern remain well-served by multi-threaded execution [37].

Noise Setting	Thread Setting	Stages Runtime (seconds)			
		Preprocessing	RANSAC	ICP	Total
Default (no noise)	Default (Multiple Thread)	223.14	771.07	71.15	1065.36
	Single Thread	221.62	818.55	175.49	1215.66
	Time Increase	**-0.68%**	**6.16%**	**146.64%**	**14.11%**
Gaussian + Spikes	Default (Multiple Thread)	815.21	5208.36	124.79	6148.36
	Single Thread	806.11	5419.30	184.46	6409.87
	Time Increase	**-1.12%**	**4.05%**	**47.82%**	**4.25%**

Table A.1: Comparison of stage runtimes (Preprocessing, RANSAC, ICP, Total) on the augmented benchmark for two noise settings: Default (no noise) and Gaussian + Spikes, both under multi-threaded and single-threaded execution. Times are in seconds, and the Time Increase rows report the percentage change from multi-threaded to single-threaded.

Appendix B

Complete Results for the Augmented 3DMatch Benchmark

This appendix reports the complete per-scene results for the augmented 3DMatch benchmark. For each scene and noise configuration, we list the mean and standard deviation of all metrics considered in Section 5.2 (RANSAC/ICP iterations, fitness, and inlier RMSE) for both the geometric-only and deep-learning pipelines. These tables (B.1, B.2, B.3, B.4, B.5, B.6) provide the raw values underlying the summary statistics reported in the main paper (Table 5.7).

Scene	RANSAC Iterations															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	83.90	35.62	55.54	34.64	37.23	13.52	23.98	84.42	36.89	13.40	19.21	75.91	37.16	13.57	6.23	47.80
Home 1	70.71	39.16	60.01	59.58	37.01	13.16	29.90	187.57	36.34	13.02	22.86	132.18	36.62	13.35	2.04	1.42
Home 2	78.21	72.01	48.26	46.54	43.93	31.34	33.31	126.97	43.45	30.80	29.65	119.75	43.75	31.53	8.10	41.78
Hotel 1	75.66	40.66	60.19	39.49	37.13	15.71	26.39	146.78	36.33	15.45	19.93	89.85	36.34	15.64	3.18	9.09
Hotel 2	87.94	45.04	45.70	40.63	43.52	20.23	11.62	73.66	43.24	20.00	9.13	30.31	42.87	20.23	2.66	2.31
Hotel 3	83.77	39.02	29.12	20.88	40.80	15.70	7.79	23.14	40.95	16.02	7.43	17.21	40.48	15.45	8.34	51.68
Study Room	59.52	22.77	55.47	67.71	29.93	8.15	17.61	101.86	29.88	8.35	12.91	63.23	29.77	8.24	2.05	1.42
Laboratory	60.44	25.50	63.65	71.43	31.59	9.50	49.94	309.26	31.33	9.40	46.90	285.49	31.91	9.27	2.37	3.88
TOTAL	**75.02**	**44.78**	**52.24**	**49.04**	**37.63**	**18.50**	**24.59**	**133.63**	**37.22**	**18.25**	**20.13**	**106.65**	**37.30**	**18.50**	**5.44**	**30.86**
Inter-scene SD	–	**10.72**	–	**11.14**	–	**5.09**	–	**13.35**	–	**5.06**	–	**12.78**	–	**4.92**	–	**2.73**

Table B.1: RANSAC iteration statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Scene	ICP Iterations															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	24.00	7.52	17.37	6.74	30.00	0.00	29.98	0.40	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00
Home 1	27.04	6.44	18.89	8.85	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00
Home 2	27.46	5.88	21.18	8.66	29.99	0.27	30.00	0.07	29.99	0.44	29.99	0.23	29.99	0.28	29.96	0.80
Hotel 1	24.11	7.59	17.48	6.98	30.00	0.00	30.00	0.00	30.00	0.03	30.00	0.00	30.00	0.00	30.00	0.00
Hotel 2	27.91	5.22	21.51	8.21	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00
Hotel 3	27.79	5.33	23.53	7.95	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00
Study Room	29.20	3.17	26.71	5.69	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00	30.00	0.00
Laboratory	29.14	3.22	25.86	6.91	30.00	0.00	30.00	0.00	30.00	0.00	29.98	0.23	30.00	0.00	30.00	0.00
TOTAL	**27.08**	**6.01**	**21.57**	**8.14**	**30.00**	**0.10**	**30.00**	**0.17**	**30.00**	**0.17**	**30.00**	**0.12**	**30.00**	**0.11**	**29.99**	**0.43**
Inter-scene SD	–	**2.01**	–	**3.59**	–	**0.00**	–	**0.01**	–	**0.00**	–	**0.01**	–	**0.00**	–	**0.01**

Table B.2: ICP iteration statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Scene	RANSAC Fitness															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.4482	0.1806	0.5925	0.1590	0.4541	0.1530	0.6088	0.1651	0.4685	0.1546	0.6033	0.1629	0.4662	0.1548	0.5335	0.1438
Home 1	0.3280	0.1656	0.5232	0.1847	0.3789	0.1213	0.4339	0.1721	0.3923	0.1282	0.4530	0.1734	0.3939	0.1219	0.3985	0.1523
Home 2	0.3341	0.2085	0.5012	0.2496	0.4391	0.2040	0.4557	0.2502	0.4540	0.2062	0.4617	0.2515	0.4527	0.2054	0.4248	0.2435
Hotel 1	0.3968	0.1757	0.5655	0.1617	0.3915	0.1150	0.4983	0.1658	0.4108	0.1152	0.5014	0.1680	0.4097	0.1141	0.4517	0.1164
Hotel 2	0.3946	0.1395	0.5816	0.1443	0.4930	0.1312	0.5246	0.1354	0.5148	0.1318	0.5366	0.1349	0.5134	0.1309	0.5009	0.1194
Hotel 3	0.4218	0.1511	0.5823	0.1587	0.4985	0.1290	0.5476	0.1327	0.5240	0.1309	0.5340	0.1339	0.5217	0.1278	0.5059	0.1214
Study Room	0.3504	0.1365	0.4787	0.1579	0.3951	0.1082	0.4733	0.1527	0.4155	0.1124	0.4890	0.1494	0.4135	0.1106	0.4700	0.1378
Laboratory	0.3847	0.1507	0.5581	0.1480	0.4146	0.1081	0.5023	0.1493	0.4290	0.1056	0.5224	0.1472	0.4295	0.1092	0.4535	0.1317
TOTAL	**0.3823**	**0.1709**	**0.5468**	**0.1792**	**0.4279**	**0.1459**	**0.5015**	**0.1892**	**0.4454**	**0.1445**	**0.5103**	**0.1869**	**0.4444**	**0.1471**	**0.4606**	**0.1775**
Inter-scene SD	–	**0.0424**	–	**0.0415**	–	**0.0460**	–	**0.0555**	–	**0.0486**	–	**0.0491**	–	**0.0477**	–	**0.0447**

Table B.3: RANSAC fitness statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Scene	ICP Fitness															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.5043	0.1684	0.5957	0.1586	0.5183	0.1706	0.6670	0.1642	0.5276	0.1685	0.6562	0.1633	0.5235	0.1675	0.6099	0.1596
Home 1	0.3714	0.1581	0.5272	0.1838	0.4274	0.1320	0.4761	0.1774	0.4369	0.1308	0.4905	0.1785	0.4384	0.1312	0.4487	0.1738
Home 2	0.3833	0.2154	0.5129	0.2493	0.4868	0.2170	0.4954	0.2609	0.4996	0.2181	0.4992	0.2620	0.4974	0.2172	0.4741	0.2645
Hotel 1	0.4473	0.1583	0.5595	0.1586	0.4416	0.1288	0.5411	0.1676	0.4570	0.1269	0.5415	0.1729	0.4564	0.1266	0.5006	0.1369
Hotel 2	0.4722	0.1420	0.6027	0.1474	0.5545	0.1345	0.5825	0.1438	0.5745	0.1319	0.5893	0.1428	0.5686	0.1423	0.5570	0.1374
Hotel 3	0.4967	0.1475	0.6016	0.1662	0.5643	0.1407	0.6086	0.1439	0.5817	0.1396	0.6048	0.1268	0.5801	0.1390	0.5813	0.1359
Study Room	0.4080	0.1305	0.4937	0.1590	0.4506	0.1221	0.5215	0.1554	0.4659	0.1218	0.5339	0.1507	0.4643	0.1210	0.5286	0.1476
Laboratory	0.4523	0.1409	0.5730	0.1468	0.4763	0.1227	0.5576	0.1500	0.4886	0.1193	0.5755	0.1445	0.4907	0.1240	0.5194	0.1581
TOTAL	**0.4419**	**0.1684**	**0.5583**	**0.1793**	**0.4836**	**0.1600**	**0.5505**	**0.1966**	**0.4968**	**0.1579**	**0.5557**	**0.1938**	**0.4954**	**0.1588**	**0.5181**	**0.1970**
Inter-scene SD	–	**0.0500**	–	**0.0425**	–	**0.0514**	–	**0.0624**	–	**0.0529**	–	**0.0559**	–	**0.0517**	–	**0.0540**

Table B.4: ICP fitness statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Scene	RANSAC Inlier RMSE															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.0223	0.0045	0.0179	0.0025	0.0226	0.0017	0.0210	0.0028	0.0235	0.0018	0.0218	0.0028	0.0235	0.0018	0.0233	0.0016
Home 1	0.0222	0.0049	0.0172	0.0032	0.0230	0.0016	0.0218	0.0024	0.0235	0.0015	0.0225	0.0025	0.0238	0.0016	0.0236	0.0015
Home 2	0.0248	0.0043	0.0201	0.0039	0.0225	0.0018	0.0219	0.0023	0.0224	0.0020	0.0224	0.0025	0.0231	0.0019	0.0233	0.0018
Hotel 1	0.0211	0.0050	0.0170	0.0027	0.0224	0.0016	0.0210	0.0025	0.0234	0.0016	0.0220	0.0026	0.0235	0.0016	0.0233	0.0015
Hotel 2	0.0229	0.0044	0.0186	0.0033	0.0220	0.0017	0.0217	0.0021	0.0224	0.0017	0.0224	0.0022	0.0230	0.0018	0.0229	0.0016
Hotel 3	0.0218	0.0045	0.0174	0.0032	0.0215	0.0019	0.0214	0.0021	0.0220	0.0019	0.0220	0.0022	0.0225	0.0018	0.0228	0.0018
Study Room	0.0208	0.0042	0.0175	0.0030	0.0220	0.0020	0.0203	0.0027	0.0223	0.0021	0.0213	0.0029	0.0230	0.0020	0.0224	0.0029
Laboratory	0.0213	0.0045	0.0185	0.0030	0.0216	0.0024	0.0205	0.0027	0.0228	0.0021	0.0220	0.0024	0.0228	0.0023	0.0229	0.0026
TOTAL	**0.0222**	**0.0047**	**0.0180**	**0.0032**	**0.0223**	**0.0019**	**0.0212**	**0.0025**	**0.0233**	**0.0019**	**0.0220**	**0.0027**	**0.0233**	**0.0019**	**0.0231**	**0.0020**
Inter-scene SD	–	**0.0013**	–	**0.0010**	–	**0.0005**	–	**0.0006**	–	**0.0004**	–	**0.0005**	–	**0.0004**	–	**0.0004**

Table B.5: RANSAC inlier RMSE statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Scene	ICP Inlier RMSE															
	No Noise				Gaussian				Gaussian + Spikes				Gaussian + Spikes + Pepper			
	Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning		Geometric-only		Deep-learning	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Kitchen	0.0178	0.0045	0.0149	0.0025	0.0213	0.0022	0.0191	0.0031	0.0220	0.0023	0.0199	0.0033	0.0221	0.0022	0.0217	0.0022
Home 1	0.0178	0.0045	0.0143	0.0032	0.0217	0.0020	0.0205	0.0026	0.0226	0.0019	0.0211	0.0028	0.0226	0.0020	0.0222	0.0019
Home 2	0.0213	0.0052	0.0169	0.0043	0.0212	0.0023	0.0205	0.0027	0.0219	0.0024	0.0211	0.0028	0.0219	0.0023	0.0219	0.0021
Hotel 1	0.0169	0.0046	0.0138	0.0023	0.0211	0.0020	0.0194	0.0030	0.0222	0.0020	0.0204	0.0031	0.0222	0.0020	0.0218	0.0018
Hotel 2	0.0181	0.0041	0.0146	0.0031	0.0205	0.0022	0.0200	0.0026	0.0214	0.0023	0.0209	0.0027	0.0214	0.0024	0.0213	0.0024
Hotel 3	0.0168	0.0042	0.0139	0.0030	0.0198	0.0026	0.0192	0.0029	0.0208	0.0027	0.0206	0.0027	0.0208	0.0027	0.0209	0.0025
Study Room	0.0161	0.0028	0.0147	0.0024	0.0205	0.0026	0.0182	0.0029	0.0216	0.0028	0.0190	0.0031	0.0217	0.0026	0.0193	0.0028
Laboratory	0.0170	0.0037	0.0152	0.0027	0.0197	0.0030	0.0184	0.0031	0.0208	0.0032	0.0192	0.0034	0.0208	0.0031	0.0203	0.0033
TOTAL	**0.0177**	**0.0045**	**0.0148**	**0.0031**	**0.0209**	**0.0024**	**0.0195**	**0.0030**	**0.0218**	**0.0024**	**0.0203**	**0.0031**	**0.0218**	**0.0024**	**0.0212**	**0.0025**
Inter-scene SD	–	**0.0016**	–	**0.0010**	–	**0.0007**	–	**0.0009**	–	**0.0006**	–	**0.0008**	–	**0.0006**	–	**0.0010**

Table B.6: ICP inlier RMSE statistics (mean and standard deviation) for geometric-only and deep-learning pipelines under different noise conditions.

Appendix C

Complete Results for the Cross-domain Benchmark

This appendix reports the complete per-object results for the cross-domain benchmark. For each object, we list the mean and standard deviation of all metrics considered in Section 4.7 (RANSAC/ICP iterations, fitness, and inlier RMSE) for both the geometric-only and deep-learning pipelines. We present the results under the single-object target setting in Tables C.1 and C.2, and full-scene in C.3 and C.4. These tables provide the raw values underlying the summary statistics reported in the main paper (Tables 5.9 and 5.10).

Object	Geometric-only results											
	RANSAC Iterations		ICP Iterations		RANSAC Fitness		ICP Fitness		RANSAC Inlier RMSE		ICP Inlier RMSE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
6	61.76	46.61	27.88	4.58	0.3323	0.1015	0.4836	0.0870	0.00312	0.00017	0.00215	0.00024
8	250.34	71.78	29.02	3.31	0.3212	0.0682	0.4488	0.0806	0.00308	0.00015	0.00224	0.00019
9	69.81	35.51	25.49	3.73	0.3208	0.0386	0.4586	0.0862	0.00316	0.00012	0.00232	0.00009
10	98.44	50.29	23.74	3.86	0.3196	0.0755	0.4589	0.0890	0.00315	0.00015	0.00234	0.00024
12	22.52	17.88	29.36	4.09	0.3077	0.1204	0.4387	0.0896	0.00298	0.00024	0.00206	0.00015
15	24.66	30.95	29.65	5.54	0.3360	0.1787	0.4698	0.1200	0.00314	0.00053	0.00238	0.00022
16	59.63	63.14	28.19	3.61	0.3160	0.0646	0.4443	0.0851	0.00300	0.00020	0.00210	0.00018
18	17.12	23.40	29.51	3.27	0.3130	0.1535	0.4455	0.1055	0.00308	0.00079	0.00221	0.00016
19	60.26	37.96	28.28	4.08	0.3378	0.1342	0.4858	0.1219	0.00309	0.00018	0.00234	0.00024
21	169.68	161.89	28.42	3.24	0.3812	0.1412	0.5241	0.1238	0.00302	0.00020	0.00220	0.00014
23	283.06	112.78	28.73	2.92	0.3696	0.1173	0.5080	0.0998	0.00311	0.00014	0.00237	0.00017
27	54.74	40.55	29.17	3.18	0.3180	0.1637	0.4376	0.1142	0.00318	0.00035	0.00245	0.00022
28	11.72	9.00	29.18	3.06	0.3502	0.1129	0.5144	0.1167	0.00325	0.00019	0.00270	0.00023
30	138.98	86.35	28.73	4.14	0.3464	0.1421	0.5060	0.1539	0.00313	0.00016	0.00243	0.00019
31	55.67	34.18	28.78	3.83	0.3374	0.1035	0.4919	0.1129	0.00316	0.00017	0.00242	0.00022
37	103.14	59.34	26.95	4.99	0.3087	0.0870	0.4689	0.0943	0.00302	0.00019	0.00216	0.00019
38	86.57	59.54	26.93	5.46	0.3292	0.1044	0.4697	0.1113	0.00310	0.00019	0.00219	0.00026
39	122.99	97.38	27.22	5.05	0.3644	0.1677	0.5402	0.1515	0.00300	0.00017	0.00212	0.00020
41	57.08	24.29	28.67	3.24	0.3543	0.1107	0.4965	0.1331	0.00308	0.00018	0.00226	0.00012
49	132.77	53.71	21.88	6.27	0.3956	0.1284	0.5551	0.1401	0.00293	0.00019	0.00237	0.00016
52	185.07	95.15	18.98	6.13	0.3424	0.1236	0.5324	0.1441	0.00308	0.00016	0.00220	0.00015
57	141.45	82.41	22.32	6.57	0.3556	0.1515	0.5485	0.1713	0.00299	0.00017	0.00235	0.00021
59	25.39	11.56	27.52	4.53	0.3267	0.0943	0.4836	0.1128	0.00303	0.00017	0.00208	0.00019
61	164.78	104.84	24.90	6.06	0.3773	0.1447	0.5615	0.1442	0.00295	0.00019	0.00195	0.00022
62	32.00	21.58	28.87	3.76	0.3613	0.1462	0.5165	0.1220	0.00304	0.00017	0.00223	0.00018
63	114.52	78.20	28.87	3.46	0.3355	0.1262	0.4960	0.1216	0.00299	0.00017	0.00210	0.00015
64	33.20	22.70	28.72	3.11	0.3556	0.1312	0.5153	0.1465	0.00294	0.00021	0.00205	0.00025
67	58.48	44.53	28.76	3.59	0.3717	0.1252	0.5310	0.1329	0.00294	0.00019	0.00219	0.00017
71	78.41	59.33	28.98	4.49	0.4200	0.1425	0.6037	0.1500	0.00317	0.00030	0.00242	0.00022
TOTAL	**108.29**	**102.01**	**27.15**	**5.46**	**0.3666**	**0.1214**	**0.4949**	**0.1168**	**0.00308**	**0.00030**	**0.00228**	**0.00025**
IO-SD	–	**68.21**	–	**2.65**	–	**0.0763**	–	**0.0771**	–	**0.00008**	–	**0.00016**

Table C.1: RANSAC/ICP iteration, fitness, and inlier RMSE statistics across objects (mean and standard deviation, SD) for the geometric-only pipeline on the cross-domain benchmark under the single-object target setting. IO-SD is the inter-object standard deviation.

Object	Deep-learning results											
	RANSAC Iterations		ICP Iterations		RANSAC Fitness		ICP Fitness		RANSAC Inlier RMSE		ICP Inlier RMSE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
6	9.70	7.81	28.21	4.02	0.3065	0.1068	0.4629	0.0962	0.00285	0.00019	0.00226	0.00030
8	78.63	55.76	28.69	3.63	0.3976	0.0869	0.4546	0.0793	0.00258	0.00021	0.00223	0.00019
9	20.39	12.85	25.84	5.35	0.2453	0.0630	0.3654	0.0453	0.00276	0.00016	0.00235	0.00009
10	68.44	55.22	17.17	6.08	0.6297	0.1457	0.6949	0.1274	0.00252	0.00022	0.00218	0.00016
12	12.81	8.79	29.23	3.50	0.5092	0.0844	0.5480	0.0756	0.00234	0.00024	0.00205	0.00013
15	25.52	15.99	29.32	2.97	0.4029	0.0660	0.4257	0.0556	0.00256	0.00022	0.00236	0.00022
16	48.43	25.72	28.72	3.94	0.3725	0.0665	0.4251	0.0557	0.00256	0.00024	0.00211	0.00018
18	26.05	15.03	29.42	2.59	0.4776	0.0791	0.4911	0.0698	0.00245	0.00024	0.00219	0.00016
19	12.93	11.61	27.50	4.72	0.5154	0.1438	0.5820	0.1567	0.00261	0.00025	0.00231	0.00025
21	32.97	24.74	29.09	3.89	0.4802	0.1156	0.5161	0.0968	0.00247	0.00023	0.00218	0.00012
23	122.89	64.86	28.26	4.48	0.4851	0.0960	0.5459	0.0850	0.00265	0.00018	0.00236	0.00015
27	11.52	9.44	28.61	4.25	0.5120	0.1097	0.5771	0.0996	0.00272	0.00023	0.00238	0.00021
28	2.59	1.97	29.02	3.56	0.4451	0.1514	0.5530	0.1527	0.00285	0.00019	0.00265	0.00024
30	25.51	15.84	29.05	3.54	0.5087	0.0898	0.5835	0.0839	0.00272	0.00020	0.00246	0.00021
31	10.89	8.16	28.71	4.22	0.5267	0.1340	0.5969	0.1131	0.00272	0.00023	0.00237	0.00022
37	77.07	49.61	26.41	5.43	0.3376	0.0653	0.3938	0.0574	0.00256	0.00026	0.00217	0.00020
38	89.74	53.74	26.51	5.62	0.3585	0.0756	0.4053	0.0655	0.00252	0.00029	0.00218	0.00025
39	67.34	49.95	27.92	4.24	0.3276	0.0861	0.3921	0.0716	0.00259	0.00025	0.00211	0.00020
41	41.85	23.63	26.41	5.20	0.3090	0.0724	0.3898	0.0579	0.00264	0.00019	0.00218	0.00012
49	68.89	52.95	18.42	5.79	0.5105	0.1596	0.5906	0.1303	0.00265	0.00020	0.00234	0.00012
52	83.32	64.68	16.90	5.32	0.4666	0.1024	0.5293	0.0892	0.00256	0.00022	0.00218	0.00011
57	52.49	28.69	21.62	6.86	0.4282	0.0997	0.5119	0.1123	0.00273	0.00018	0.00235	0.00019
59	28.94	21.91	24.92	6.07	0.3199	0.0659	0.3717	0.0409	0.00250	0.00028	0.00209	0.00017
61	55.78	31.32	25.40	5.93	0.4347	0.0981	0.4958	0.0861	0.00248	0.00026	0.00202	0.00024
62	19.66	12.75	28.36	4.46	0.2870	0.0709	0.3737	0.0391	0.00273	0.00022	0.00227	0.00018
63	6.36	4.54	28.93	3.39	0.2451	0.1044	0.4420	0.0998	0.00276	0.00022	0.00218	0.00020
64	12.97	8.56	29.77	1.19	0.2408	0.0646	0.3925	0.0654	0.00272	0.00026	0.00205	0.00024
67	22.34	24.28	29.52	1.99	0.2830	0.0764	0.3837	0.0551	0.00270	0.00021	0.00223	0.00019
71	42.14	19.28	27.00	5.93	0.4370	0.0740	0.4973	0.0716	0.00271	0.00021	0.00239	0.00020
TOTAL	**45.19**	**47.56**	**26.32**	**6.20**	**0.4408**	**0.1307**	**0.5074**	**0.1199**	**0.00263**	**0.00025**	**0.00228**	**0.00024**
IO-SD	–	**30.57**	–	**3.65**	–	**0.1024**	–	**0.0871**	–	**0.00012**	–	**0.00014**

Table C.2: RANSAC/ICP iteration, fitness, and inlier RMSE statistics across objects (mean and standard deviation, SD) for the deep-learning pipeline on the cross-domain benchmark under the single-object target setting. IO-SD is the inter-object standard deviation.

Object	Deep-learning results											
	RANSAC Iterations		ICP Iterations		RANSAC Fitness		ICP Fitness		RANSAC Inlier RMSE		ICP Inlier RMSE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
6	31.44	28.95	28.47	4.36	0.3194	0.1002	0.5106	0.0762	0.00280	0.00018	0.00213	0.00021
8	11.75	8.45	29.87	0.85	0.2716	0.0577	0.4182	0.0822	0.00287	0.00018	0.00245	0.00025
9	27.35	14.36	28.65	3.50	0.2336	0.0470	0.3842	0.0504	0.00279	0.00014	0.00236	0.00010
10	27.00	24.03	29.76	1.37	0.2643	0.0758	0.4119	0.1503	0.00296	0.00013	0.00270	0.00017
12	16.62	20.83	29.32	2.84	0.4440	0.1407	0.5653	0.0693	0.00265	0.00033	0.00208	0.00014
15	6.36	6.05	29.53	2.45	0.2867	0.1173	0.4343	0.0511	0.00279	0.00030	0.00228	0.00019
16	21.51	19.71	29.08	3.13	0.3111	0.1042	0.4206	0.0676	0.00268	0.00029	0.00213	0.00019
18	6.61	5.01	29.58	2.30	0.3888	0.1269	0.5168	0.0644	0.00274	0.00024	0.00227	0.00014
19	–	–	–	–	–	–	–	–	–	–	–	–
21	65.26	73.21	27.09	5.68	0.4043	0.1095	0.5302	0.0901	0.00265	0.00025	0.00225	0.00015
23	41.08	12.68	29.55	1.99	0.3728	0.0762	0.5419	0.1069	0.00284	0.00016	0.00243	0.00020
27	17.65	15.44	28.51	3.73	0.4386	0.1271	0.5691	0.1324	0.00281	0.00019	0.00251	0.00020
28	13.53	6.44	27.37	4.97	0.3769	0.0913	0.5010	0.1147	0.00294	0.00023	0.00259	0.00015
30	32.23	33.54	27.65	5.26	0.4312	0.1267	0.5317	0.1301	0.00281	0.00019	0.00251	0.00022
31	21.15	13.38	28.23	4.78	0.4148	0.1107	0.5442	0.1513	0.00279	0.00018	0.00252	0.00022
37	28.45	17.83	28.35	3.95	0.3070	0.0618	0.3813	0.0570	0.00257	0.00022	0.00221	0.00015
38	31.60	16.68	23.00	6.40	0.3719	0.0260	0.4157	0.0313	0.00252	0.00012	0.00190	0.00022
39	30.10	23.11	26.28	4.99	0.3345	0.0878	0.4099	0.0575	0.00250	0.00029	0.00207	0.00017
41	7.00	2.83	30.00	0.00	0.2550	0.1268	0.3669	0.0937	0.00285	0.00022	0.00233	0.00001
49	79.49	30.53	26.19	5.62	0.3692	0.1116	0.5092	0.1862	0.00284	0.00015	0.00257	0.00027
52	70.12	39.19	23.16	6.18	0.4127	0.0995	0.5414	0.1113	0.00275	0.00021	0.00231	0.00028
57	36.24	29.19	27.76	4.69	0.3255	0.0833	0.4174	0.1233	0.00290	0.00013	0.00264	0.00025
59	13.45	5.66	28.27	3.10	0.2880	0.0730	0.3874	0.0494	0.00256	0.00032	0.00200	0.00020
61	37.15	34.79	26.68	5.33	0.3795	0.1162	0.4967	0.0947	0.00254	0.00031	0.00204	0.00027
62	27.96	39.87	27.00	6.30	0.2507	0.0679	0.3729	0.0412	0.00266	0.00020	0.00223	0.00021
63	25.51	14.17	28.73	3.50	0.3497	0.0916	0.4954	0.0831	0.00258	0.00023	0.00213	0.00020
64	32.83	16.39	29.67	0.82	0.3136	0.0779	0.4245	0.0901	0.00252	0.00024	0.00188	0.00028
67	23.47	11.80	30.00	0.00	0.2288	0.0618	0.3576	0.0471	0.00269	0.00025	0.00225	0.00021
71	35.35	38.30	27.85	5.06	0.3332	0.1229	0.4972	0.0866	0.00287	0.00018	0.00246	0.00022
TOTAL	**33.31**	**35.57**	**27.83**	**4.72**	**0.3689**	**0.1223**	**0.4948**	**0.1176**	**0.00273**	**0.00027**	**0.00230**	**0.00029**
IO-SD	–	**17.87**	–	**1.80**	–	**0.0639**	–	**0.0680**	–	**0.00014**	–	**0.00022**

Table C.3: RANSAC/ICP iteration, fitness, and inlier RMSE statistics across objects (mean and standard deviation, SD) for the geometric-only pipeline on the cross-domain benchmark under the full-scene target setting. IO-SD is the inter-object standard deviation.

Object	Deep-learning results											
	RANSAC Iterations		ICP Iterations		RANSAC Fitness		ICP Fitness		RANSAC Inlier RMSE		ICP Inlier RMSE	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
6	4.67	3.51	30.00	0.00	0.2223	0.0259	0.3490	0.0156	0.00296	0.00012	0.00256	0.00007
8	3.88	2.90	30.00	0.00	0.2901	0.0931	0.4702	0.0528	0.00283	0.00019	0.00228	0.00015
9	2.36	1.80	28.64	3.11	0.1749	0.0534	0.3696	0.0464	0.00284	0.00020	0.00236	0.00008
10	2.75	2.36	30.00	0.00	0.1158	0.0626	0.3395	0.0608	0.00309	0.00029	0.00258	0.00022
12	16.21	13.86	29.15	3.24	0.5260	0.1036	0.5766	0.0589	0.00255	0.00031	0.00207	0.00013
15	2.93	4.41	29.65	1.92	0.3317	0.1072	0.4420	0.0392	0.00279	0.00026	0.00230	0.00018
16	8.21	7.45	29.03	3.59	0.2556	0.1154	0.3941	0.0556	0.00282	0.00030	0.00216	0.00018
18	7.16	9.04	29.41	2.82	0.4021	0.1348	0.5196	0.0526	0.00273	0.00028	0.00226	0.00018
19	1.00	0.00	30.00	0.00	0.2508	0.0860	0.4495	0.1415	0.00271	0.00012	0.00231	0.00019
21	6.43	6.57	30.00	0.00	0.2355	0.0767	0.4110	0.0700	0.00280	0.00029	0.00240	0.00014
23	4.49	4.90	29.76	1.64	0.3448	0.1086	0.5202	0.1202	0.00286	0.00016	0.00244	0.00024
27	3.60	3.26	29.09	2.94	0.3687	0.1109	0.4839	0.1125	0.00285	0.00018	0.00259	0.00017
28	2.80	1.30	30.00	0.00	0.4439	0.0830	0.4784	0.0344	0.00280	0.00027	0.00263	0.00009
30	5.29	5.12	28.34	4.44	0.3841	0.1410	0.5164	0.1466	0.00288	0.00018	0.00256	0.00024
31	5.71	4.46	27.49	5.16	0.4132	0.1274	0.5505	0.1592	0.00281	0.00020	0.00250	0.00021
37	2.17	0.98	27.83	5.31	0.2420	0.0579	0.4060	0.0633	0.00292	0.00010	0.00208	0.00017
38	5.90	4.33	26.30	6.04	0.2827	0.0539	0.4674	0.0633	0.00276	0.00020	0.00209	0.00028
39	4.00	3.13	26.90	3.87	0.2870	0.1188	0.4490	0.0592	0.00278	0.00025	0.00199	0.00010
41	–	–	–	–	–	–	–	–	–	–	–	–
49	8.97	6.92	24.24	5.22	0.2598	0.0611	0.3988	0.0679	0.00284	0.00020	0.00243	0.00016
52	4.34	3.01	25.08	5.68	0.3306	0.0995	0.5028	0.1094	0.00280	0.00019	0.00234	0.00023
57	6.84	6.27	26.40	5.54	0.3403	0.1151	0.4590	0.1470	0.00290	0.00014	0.00257	0.00026
59	13.29	14.89	23.57	5.09	0.3177	0.0889	0.4025	0.0363	0.00264	0.00033	0.00195	0.00013
61	5.05	5.28	28.05	4.47	0.2730	0.1135	0.4471	0.1127	0.00282	0.00027	0.00225	0.00034
62	–	–	–	–	–	–	–	–	–	–	–	–
63	4.29	2.96	30.00	0.00	0.2340	0.0788	0.3576	0.0464	0.00274	0.00016	0.00219	0.00029
64	9.69	8.30	28.31	4.12	0.2634	0.0816	0.3962	0.0713	0.00280	0.00026	0.00206	0.00022
67	–	–	–	–	–	–	–	–	–	–	–	–
71	–	0.00	30.00	0.00	0.0876	0.0455	0.4020	0.1311	0.00302	0.00018	0.00261	0.00036
TOTAL	**7.77**	**8.99**	**28.31**	**4.23**	**0.3702**	**0.1464**	**0.4924**	**0.1210**	**0.00278**	**0.00027**	**0.00233**	**0.00028**
IO-SD	–	**3.49**	–	**1.90**	–	**0.0966**	–	**0.0630**	–	**0.00011**	–	**0.00021**

Table C.4: RANSAC/ICP iteration, fitness, and inlier RMSE statistics across objects (mean and standard deviation, SD) for the deep-learning pipeline on the cross-domain benchmark under the full-scene target setting. IO-SD is the inter-object standard deviation.

Bibliography

[1] Martin A. Fischler and Robert C. Bolles. «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography». In: *Communications of the ACM* 24.6 (1981), pp. 381–395. DOI: 10.1145/358669.358692 (cit. on pp. 1, 5, 9, 15, 17, 19, 31).

[2] Paul J. Besl and Neil D. McKay. «A Method for Registration of 3-D Shapes». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791 (cit. on pp. 1, 6, 7, 15, 19, 20, 22).

[3] Yang Chen and Gérard Medioni. «Object modeling by registration of multiple range images». In: *Image and Vision Computing* 10.3 (1992), pp. 145–155. DOI: 10.1016/0262-8856(92)90066-C (cit. on pp. 1, 7, 15, 19, 20, 23, 31, 42).

[4] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. «Fast Point Feature Histograms (FPFH) for 3D Registration». In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473 (cit. on pp. 1, 7, 8, 10–12, 31, 37, 58).

[5] Christopher B. Choy, Jaesik Park, and Vladlen Koltun. «Fully Convolutional Geometric Features». In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (cit. on pp. 1, 3, 7–10, 12–15, 17, 19, 31, 37–39, 48, 58, 59, 61, 62, 67, 68, 81).

[6] Shinji Umeyama. «Least-Squares Estimation of Transformation Parameters Between Two Point Patterns». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.4 (1991), pp. 376–380. DOI: 10.1109/34.88573 (cit. on pp. 1, 17, 18, 22, 40).

[7] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. «Open3D: A Modern Library for 3D Data Processing». In: *arXiv preprint arXiv:1801.09847* (2018). URL: https://arxiv.org/abs/1801.09847 (cit. on pp. 3, 19, 27, 28, 31, 32, 35–37, 39, 40, 44, 53, 54, 62).

[8] Gabriel Corteletti. *Project Main Repository: ai-cv-point-cloud-registration.* `https://github.com/gacorteletti/ai-cv-point-cloud-registration`. Accessed: 2025-11-03. 2025 (cit. on pp. 3, 4, 33, 38, 45, 56).

[9] Hanwen Cao, Hao-Shu Fang, Wenhai Liu, and Cewu Lu. «SuctionNet-1Billion: A Large-Scale Benchmark for Suction Grasping». In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 8719–8726. DOI: `10.1109/LRA.2021.3115406`. arXiv: `2103.12311` (cit. on pp. 3, 26, 32, 52, 56).

[10] Peter Biber and Wolfgang Straßer. «The Normal Distributions Transform: A New Approach to Laser Scan Matching». In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2003, pp. 2743–2748. DOI: `10.1109/IROS.2003.1249285` (cit. on pp. 6, 7, 19).

[11] Andriy Myronenko and Xubo Song. «Point Set Registration: Coherent Point Drift». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.12 (2010), pp. 2262–2275. DOI: `10.1109/TPAMI.2010.46` (cit. on pp. 6, 7, 19).

[12] S. Salti, F. Tombari, and L. Di Stefano. «SHOT: Unique Signatures of Histograms for Surface and Texture Description». In: *Computer Vision and Image Understanding (CVIU)* 125 (2014), pp. 251–264. DOI: `10.1016/j.cviu.2014.04.011` (cit. on pp. 7, 10).

[13] X. Bai, Z. Luo, L. Zhou, H. Fu, L. Quan, and C.-L. Tai. «D3Feat: Joint Learning of Dense Detection and Description of 3D Local Features». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020. arXiv: `2003.03164` (cit. on pp. 7, 8, 10, 12).

[14] Zheng Qin, Hao Yu, Changjian Wang, Yulan Guo, Yuxing Peng, and Kai Xu. «Geometric Transformer for Fast and Robust Point Cloud Registration». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 (cit. on pp. 7–10, 12, 17, 26–28, 32).

[15] Christopher Choy, Wei Dong, and Vladlen Koltun. «Deep Global Registration». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 7).

[16] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. «Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (2016), pp. 2241–2254. DOI: `10.1109/TPAMI.2015.2513405` (cit. on pp. 7, 16).

[17] Gary K. L. Tam, Zhi-Quan Cheng, Yu-Kun Lai, Frank C. Langbein, Yonghuai Liu, David Marshall, Ralph R. Martin, Xian-Fang Sun, and Paul L. Rosin. «Registration of 3D Point Clouds and Meshes: A Survey from Rigid to Nonrigid». In: *IEEE Transactions on Visualization and Computer Graphics* 19.7 (2013), pp. 1199–1217. DOI: 10.1109/TVCG.2012.310 (cit. on pp. 7–9, 11, 12, 15, 17, 19, 20, 30, 31).

[18] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. «Generalized-ICP». In: *Robotics: Science and Systems (RSS)*. 2009. DOI: 10.15607/RSS.2009.V.021 (cit. on pp. 7, 15, 19, 24).

[19] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. «Fast Global Registration». In: *European Conference on Computer Vision (ECCV)*. Vol. 9906. LNCS. Springer, 2016, pp. 694–711. DOI: 10.1007/978-3-319-46466-4_42 (cit. on pp. 7, 16).

[20] Fabio Poiesi and Davide Boscaini. «Learning General and Distinctive 3D Local Deep Descriptors for Point Cloud Registration». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022). DOI: 10.1109/TPAMI.2022.3175371 (cit. on pp. 8, 12).

[21] Yan Zhong. «Intrinsic Shape Signatures: A Shape Descriptor for 3D Object Recognition». In: *2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*. 2009, pp. 689–696. DOI: 10.1109/ICCVW.2009.5457637 (cit. on p. 8).

[22] Paul Scovanner, Saad Ali, and Mubarak Shah. «A 3-Dimensional SIFT Descriptor and Its Application to Action Recognition». In: *Proceedings of the 15th ACM International Conference on Multimedia (ACM MM)*. 2007, pp. 357–360. DOI: 10.1145/1291233.1291311 (cit. on p. 8).

[23] Jiaxin Li and Gim Hee Lee. «USIP: Unsupervised Stable Interest Point Detection from 3D Point Clouds». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019 (cit. on p. 8).

[24] Andrew E. Johnson and Martial Hebert. «Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21.5 (1999), pp. 433–449. DOI: 10.1109/34.765655 (cit. on pp. 8, 10).

[25] Xiyu Zhang, Jiaqi Yang, Shikun Zhang, and Yanning Zhang. «3D Registration With Maximal Cliques». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 17745–17754. URL: https://openaccess.thecvf.com/content/CVPR2023/html/Zhang_3D_Registration_With_Maximal_Cliques_CVPR_2023_paper.html (cit. on pp. 9, 10, 16, 17).

[26] Shengyu Huang, Zan Gojcic, Artem Usvyatsov, Andreas Wieser, and Konrad Schindler. «PREDATOR: Registration of 3D Point Clouds with Low Overlap». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 4267–4276. DOI: 10.1109/CVPR46437.2021.00426 (cit. on pp. 9, 10, 26–28, 32, 46).

[27] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. «SuperGlue: Learning Feature Matching with Graph Neural Networks». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4937–4946. DOI: 10.1109/CVPR42600.2020.00499 (cit. on p. 10).

[28] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, Jian Wan, and Nui Ming Kwok. «A Comprehensive Performance Evaluation of 3D Local Feature Descriptors». In: *International Journal of Computer Vision* 116.1 (2016), pp. 66–89. DOI: 10.1007/s11263-015-0824-y (cit. on pp. 10–12, 31).

[29] Federico Tombari, Samuele Salti, and Luigi Di Stefano. «Unique Shape Context for 3D Data Description». In: *Proceedings of the ACM Workshop on 3D Object Retrieval (3DOR)*. 2010, pp. 57–62. DOI: 10.1145/1877808.1877821 (cit. on p. 10).

[30] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. «Aligning Point Cloud Views Using Persistent Feature Histograms». In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2008, pp. 3384–3391. DOI: 10.1109/IROS.2008.4650967 (cit. on pp. 10, 11).

[31] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. «Surface Reconstruction from Unorganized Points». In: *Computer Graphics (SIGGRAPH)* 26.2 (1992), pp. 71–78. DOI: 10.1145/142920.134011 (cit. on pp. 11, 36).

[32] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. «3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on pp. 12, 14, 15, 25, 26, 28, 32, 46, 48, 61, 62).

[33] Zan Gojcic, Caifa Zhou, Jan D. Wegner, and Andreas Wieser. «3DSmoothNet: A Deep Learning-Based 3D Point Cloud Descriptor». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 12).

[34] Shuquan Ao, Qingyong Hu, Bo Yang, Andrew Markham, and Niki Trigoni. «SpinNet: Learning a General Surface Descriptor for 3D Point Cloud Registration». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (cit. on p. 12).

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. «Fully Convolutional Networks for Semantic Segmentation». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965 (cit. on p. 12).

[36] Benjamin Graham and Laurens van der Maaten. «Submanifold Sparse Convolutional Networks». In: *arXiv preprint arXiv:1706.01307*. 2017. URL: https://arxiv.org/abs/1706.01307 (cit. on p. 13).

[37] Christopher Choy, JunYoung Gwak, and Silvio Savarese. «4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3075–3084. DOI: 10.1109/CVPR.2019.00319 (cit. on pp. 13, 38, 39, 84).

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. «U-Net: Convolutional Networks for Biomedical Image Segmentation». In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. Lecture Notes in Computer Science. Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28 (cit. on p. 13).

[39] Raia Hadsell, Sumit Chopra, and Yann LeCun. «Dimensionality Reduction by Learning an Invariant Mapping». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100 (cit. on p. 15).

[40] Florian Schroff, Dmitry Kalenichenko, and James Philbin. «FaceNet: A Unified Embedding for Face Recognition and Clustering». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 815–823. DOI: 10.1109/CVPR.2015.7298682 (cit. on p. 15).

[41] Dror Aiger, Niloy J. Mitra, and Daniel Cohen-Or. «4-Points Congruent Sets for Robust Pairwise Surface Registration». In: *ACM Transactions on Graphics* 27.3 (2008), 85:1–85:10. DOI: 10.1145/1360612.1360690 (cit. on p. 16).

[42] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. «Super 4PCS: Fast Global Pointcloud Registration via Smart Indexing». In: *Computer Graphics Forum* 33.5 (2014), pp. 205–215. DOI: 10.1111/cgf.12446 (cit. on p. 16).

[43] Dylan Campbell and Lars Petersson. «Globally-Optimal Gaussian Mixture Alignment». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2736–2745. DOI: 10.1109/CVPR.2017.293 (cit. on p. 16).

[44] Heng Yang, Jingnan Shi, and Luca Carlone. «TEASER: Fast and Certifiable Point Cloud Registration». In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1087–1094. DOI: 10.1109/LRA.2020.2966415 (cit. on p. 16).

[45] Berthold K. P. Horn. «Closed-form Solution of Absolute Orientation Using Unit Quaternions». In: *Journal of the Optical Society of America A* 4.4 (1987), pp. 629–642. DOI: 10.1364/JOSAA.4.000629 (cit. on pp. 18, 22).

[46] Szymon Rusinkiewicz and Marc Levoy. «Efficient Variants of the ICP Algorithm». In: *Proceedings of the International Conference on 3-D Digital Imaging and Modeling (3DIM)*. 2001, pp. 145–152. DOI: 10.1109/IM.2001.924423 (cit. on pp. 19–24, 30, 31, 35, 44).

[47] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. «Colored Point Cloud Registration Revisited». In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2017 (cit. on pp. 19, 23).

[48] Todor Stoyanov, Martin Magnusson, and Achim J. Lilienthal. «Fast and Accurate Scan Registration through Minimization of the Distance between 3D-NDT Models». In: *The International Journal of Robotics Research* 31.12 (2012), pp. 1377–1393. DOI: 10.1177/0278364912460895 (cit. on p. 19).

[49] Richard A. Newcombe et al. «KinectFusion: Real-Time Dense Surface Mapping and Tracking». In: *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*. 2011 (cit. on p. 19).

[50] Dmitry Chetverikov, Denis Svirko, Dmitri Stepanov, and Pavel Krsek. «The Trimmed Iterative Closest Point Algorithm». In: *International Conference on Pattern Recognition (ICPR)*. 2002 (cit. on pp. 19, 24).

[51] Jon Louis Bentley. «Multidimensional Binary Search Trees Used for Associative Searching». In: *Communications of the ACM* 18.9 (1975), pp. 509–517. DOI: 10.1145/361002.361007 (cit. on pp. 20–22).

[52] Stanford Computer Graphics Laboratory. *The Stanford 3D Scanning Repository*. http://graphics.stanford.edu/data/3Dscanrep/. Accessed: 2025-11-01 (cit. on p. 25).

[53] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. «3D ShapeNets: A Deep Representation for Volumetric Shapes». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920 (cit. on p. 25).

[54] Angel X. Chang et al. «ShapeNet: An Information-Rich 3D Model Repository». In: *arXiv preprint arXiv:1512.03012* (2015) (cit. on p. 25).

[55] Andreas Geiger, Philip Lenz, and Raquel Urtasun. «Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite». In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on pp. 25, 28).

[56] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. «Comparing ICP Variants on Real-World Data Sets». In: *Autonomous Robots* 34.3 (2013), pp. 133–148 (cit. on pp. 25, 35).

[57] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. «A Benchmark for the Evaluation of RGB-D SLAM Systems». In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2012 (cit. on pp. 25, 28).

[58] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. «A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM». In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2014 (cit. on p. 25).

[59] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. «Robust Reconstruction of Indoor Scenes». In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 25).

[60] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. «GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11444–11453 (cit. on pp. 26, 32).

[61] Chris Choy. *FCGF: Official GitHub Repository*. `https://github.com/chrischoy/FCGF`. Accessed: 2025-11-03. 2019 (cit. on pp. 38, 39, 48).

[62] Gabriel Corteletti. *FCGF: Forked Github Repository*. `https://github.com/gacorteletti/FCGF`. Accessed: 2025-11-03. 2025 (cit. on p. 38).

[63] Kok-Lim Low. *Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration*. Tech. rep. TR04-004. Department of Computer Science: University of North Carolina at Chapel Hill, 2004. URL: `https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf` (cit. on p. 42).

[64] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. ISBN: 978-1107159396. DOI: `10.1017/9781316671528` (cit. on p. 42).

[65] Joan Solà. *Lie Theory for State Estimation (Tutorial)*. arXiv preprint arXiv:1812.01537. 2018. URL: `https://arxiv.org/abs/1812.01537` (cit. on p. 42).

[66] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd. Cambridge University Press, 2003. ISBN: 978-0521540513 (cit. on pp. 50, 51).

[67] Steven M. Seitz. *3D Reconstruction from Multiple Images, Part 1: Principles*. Technical report. Available online. 2000. URL: https://www.researchgate.net/publication/220427980_3D_Reconstruction_from_Multiple_Images_Part_1_-_Principles (cit. on p. 51).

[68] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall, 2008. ISBN: 978-0131687288 (cit. on p. 51).

[69] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685 (cit. on pp. 53, 54, 59).

[70] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. DOI: 10.1007/978-1-84882-935-0 (cit. on pp. 53, 54).

[71] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. 2nd ed. Society for Industrial and Applied Mathematics, 2002. DOI: 10.1137/1.9780898718027 (cit. on p. 59).

[72] Andy Zeng. *3DMatch Toolbox: Geometric Registration Benchmark*. https://github.com/andyzeng/3dmatch-toolbox. Accessed: 2025-11-09. 2017 (cit. on pp. 67, 69, 71, 72).