Alma Mater Studiorum - Università Di Bologna

Scuola di Ingegneria e Architettura

Dipartimento di Informatica - Scienza e Ingegneria (DISI)

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea
in
Computer Vision and Image Processing

Learning a Local Reference Frame for Point Clouds using Spherical CNNs

Candidato:
Federico Stella

Relatore:
Prof. Luigi Di Stefano

Correlatori:
Prof. Samuele Salti
Dott. Riccardo Spezialetti
Dott. Marlon Marcon

Anno Accademico 2018/2019

Abstract

One of the most important tasks in the field of 3D Computer Vision is surface matching, which means finding correspondences between 3D shapes. The most successful approach to handle this problem involves computing compact local features, called descriptors, that have to be matched across different poses of the same shape and thus need to be invariant with respect to shape orientation. The most popular way to achieve this property is to use Local Reference Frames (LRFs): local coordinate systems that provide a canonical orientation to the local 3D patches used to compute the descriptors. In the literature, there exist plenty of methods to compute LRFs, but they are all based on hand-crafted algorithms. There is also one recent proposal deploying neural networks. Yet, it feeds them with manually engineered features, and therefore it does not fully benefit from the modern end-to-end learning strategies.

The aim of this work is to employ a data-driven approach to fully learn a Local Reference Frame from raw point clouds, thus realizing the first example of end-to-end learning applied to LRFs. To do so, we take advantage of a recent innovation called Spherical Convolutional Neural Networks, that process signals living in SO(3) and are therefore naturally suited to represent and estimate rotations and LRFs. We test our performances against existing methods on standard benchmarks, achieving promising results.

Abstract

Uno dei problemi più importanti della 3D Computer Vision è il cosiddetto surface matching, che consiste nel trovare corrispondenze tra oggetti tridimensionali. Attualmente il problema viene affrontato calcolando delle feature locali e compatte, chiamate descrittori, che devono essere riconosciute e messe in corrispondenza al mutare della posa dell'oggetto nello spazio, e devono quindi essere invarianti rispetto all'orientazione. Il metodo più usato per ottenere questa proprietà consiste nell'utilizzare dei Local Reference Frame (LRF): sistemi di coordinate locali che forniscono un'orientazione canonica alle porzioni di oggetti 3D che vengono usate per calcolare i descrittori. In letteratura esistono diversi modi per calcolare gli LRF, ma fanno tutti uso di algoritmi progettati manualmente. Vi è anche una recente proposta che utilizza reti neurali, tuttavia queste vengono addestrate mediante feature specificamente progettate per lo scopo, il che non permette di sfruttare pienamente i benefici delle moderne strategie di end-to-end learning.

Lo scopo di questo lavoro è utilizzare un approccio data-driven per far imparare a una rete neurale il calcolo di un Local Reference Frame a partire da point cloud grezze, producendo quindi il primo esempio di end-to-end learning applicato alla stima di LRF. Per farlo, sfruttiamo una recente innovazione chiamata Spherical Convolutional Neural Networks, le quali generano e processano segnali nello spazio SO(3) e sono quindi naturalmente adatte a rappresentare e stimare orientazioni e LRF. Confrontiamo le prestazioni ottenute con quelle di metodi esistenti su benchmark standard, ottenendo risultati promettenti.

Prefazione

A differenza del resto della tesi, questa piccola prefazione sarà in italiano, perché italiani sono coloro a cui è rivolta. E non sarà proprio una prefazione, quanto più un flusso di pensiero con qualche ringraziamento qua e là, sparso a zuccherare un discorso che è perlopiù rivolto a me stesso.

Ho pensato a lungo, negli ultimi (tanti) mesi, a cosa farò in futuro, nella vita e per lavoro, e non sono riuscito a darmi pace. Con la Laurea Magistrale, più che un traguardo, dovrei aver raggiunto un punto di partenza, un punto in cui posso finalmente impegnarmi a fondo in qualcosa per dimostrare quello che ho imparato negli anni e quello che posso scoprire d'ora in poi. Il problema, in questa bella visione, è che non mi sento né particolarmente preparato, né particolarmente appassionato per poter andare avanti con successo in una qualunque attività, men che meno se di ricerca. Inoltre, mi sento completamente spaesato dalla mancanza di un percorso chiaramente definito davanti a me.
Ho riletto la prefazione che ho scritto per la tesi triennale, qualche giorno fa, e questo mi ha fatto riflettere su cosa sia cambiato da allora e cosa no. Ho scritto che era solo una piccola tappa, che avrei continuato a studiare ancora a lungo, e che non sentivo di essere arrivato ad alcun traguardo: sono parole che, per fortuna, condivido ancora in buona misura, ma mi ha fatto bene rileggerle.
Al momento mi sento tornato a come stavo negli ultimi anni di Liceo, quando ottenevo buoni risultati ma lo facevo soltanto perché lo sentivo come un obbligo nei confronti di me stesso, oppresso. La necessità di libertà che provavo allora è stata tale da avermi fatto scegliere un percorso totalmente dettato dal mio interesse, ovvero l'informatica, materia di cui non sapevo nulla ma che mi aveva sempre incuriosito, e di dedicarmi parallelamente a tante altre attività di svago personale. Dopo un bellissimo percorso triennale, durante il quale mi sono divertito, non posso negarlo, mi sono trovato col rimorso per non aver scelto un percorso differente, che più mi rispecchiasse, e sentivo persa la mia identità. Questi sentimenti sono continuati per tutto il percorso magistrale, che ho scelto per portare a termine uno studio che altrimenti sentivo incompleto e sprecato, e sono ora mutati in una nuova sensazione di oppressione, con conseguente nuova necessità di libertà. Mi sono quindi servite le mie parole della prefazione triennale, perché ricordo la voglia

di fare con cui le ho scritte, ma, pur condividendole ancora, le devo conciliare con la necessità di libertà che è tornata a farsi vedere. Al momento della stesura di questa prefazione non ho ancora una scelta definitiva su quello che farò nei prossimi mesi, ma conto di averla presto. Ringrazio tanto, anche per il supporto nella scelta, il Prof. Di Stefano.

Ringrazio poi anche il Prof. Salti, Riccardo e Marlon, correlatori, che mi hanno sempre prontamente aiutato, anche a orari improbabili, e spesso real-time. D'ora in poi smetto di fare nomi perché, così come sono timido nel parlato, lo sono anche nello scritto quando c'è da ringraziare.
Saluto e ringrazio i miei amici e compagni di corso, con cui ho condiviso tante esperienze, e ai quali non mando un messaggio di possibile addio a causa del termine degli studi comuni, anzi, mando un messaggio di speranza di poter saldare ulteriormente i rapporti in una convergenza di percorsi che possa giovare a tutti.
Parlando di termine degli studi, mi tornano in mente i pensieri riguardanti la scorsa Laurea, ovvero l'impossibilità fisica di invitare conoscenti alla mia discussione, la delusione di non poter mostrare a chi tiene a me il punto a cui ero giunto, e la speranza di poterlo fare finalmente al termine della Magistrale. Ho assistito a tante Lauree, è una cosa bellissima poter discutere del proprio lavoro davanti ad altre persone, per poi uscire a festeggiare per il proprio successo. Peccato che non sia stato così, peccato che non potrà esserlo neanche questa volta, e peccato avere quindi una seconda delusione riguardante il termine di un percorso di studi.

Essendo un flusso di pensieri, la parola termine mi riporta all'altro percorso che si conclude insieme a quello magistrale, ovvero il Collegio Superiore. Un luogo che mi ha dato tanto, a cui spero di aver dato tanto, e che non vorrei mai lasciare. Un'esperienza veramente unica, sia per le persone che ci vivono, sia per il senso di comunità, e soprattutto per la condivisione della propria vita con tanti altri ragazzi brillanti provenienti da percorsi estremamente diversi tra loro e dal mio. Li saluto tutti, li ringrazio, e spero di poter continuare a fare qualcosa per il Collegio anche una volta uscito.

Infine, ringrazio la mia famiglia, i miei genitori, mia sorella e i miei nonni, che sono sempre stati pronti a darmi ogni tipo di aiuto e non mi hanno mai fatto mancare nulla: è raro trovare famiglie del genere. E ovviamente ringrazio la mia ragazza, che non so come possa aver sopportato prima la mia distanza, poi il mio morale basso, e poi anche il mio poco tempo libero. Vedrai che andrà meglio :)

E ora, per chi vuole, buona lettura.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer Vision is a highly interdisciplinary scientific field that aims to provide computers with human-like or superhuman abilities in terms of vision-related tasks, such as object recognition, pose estimation, video tracking, 3D scene reconstruction, augmented reality and many more. It involves mathematics, computer science and artificial intelligence, allowing computers to carry out countless industrial activities (e.g. quality control, defect detection), surveillance duties and consumer-related tasks (e.g. face recognition and visual search), eventually enabling futuristic applications such as augmented reality and autonomous vehicles. Computer Vision processes images and videos, both 2D and 3D, provided by cameras, scanners and depth sensors.

In the context of this work, we will be dealing with 3D data, namely with point clouds: sets of points in the 3D space, generated by scanners and depth cameras. One of the most crucial and researched on problems in 3D Computer Vision is *surface matching*, i.e. the process of finding similarities between surfaces [29] [15], in order to tell if a surface matches another and, if so, how to align them. Surface matching has applications in a wide variety of fields, such as robotics, automation, biometric systems, 3D scene reconstruction, search in 3D object databases [29], augmented reality, obstacle avoidance and path planning for autonomous vehicles, and has become increasingly relevant due to the availability of off-the-shelf depth sensors such as Microsoft Kinect and Intel RealSense [18]. It also represents the foundation of two very important tasks [22]: surface registration, which is the process of aligning different 3D fragments into a single common coordinate system [27], and 3D object recognition.
In the past, surface matching was handled with global approaches, that suffered from clutter and occlusion problems. Following the trend of 2D Computer Vision, in the last two decades the paradigm has undergone a shift towards local methods, that are able to more effectively withstand such nuisances [22] [29]. A local approach means finding interesting points (called keypoints) on the 3D surface, and

considering only a limited neighborhood (hereinafter *support*) of such points for the computation of compact representations (named *descriptors*) of such neighborhoods. These compact descriptors should be at the same time highly distinctive and robust, in order for corresponding points of two views of the same surface to be successfully matched together. If enough descriptors are matched between a model and a target point cloud, the model object could be present in the target cloud (object recognition); moreover, if enough descriptors are matched, they allow for the estimation of rigid body transformations, that are the key to surface registration [22].
In order to have such results, the descriptors should be informative enough to avoid false matches as much as possible, while still being robust enough to be matched across point clouds with very different orientations, and withstanding nuisances such as noise, point density variations, clutter and occlusion. The property of being invariant to 3D rigid body transformations is called *rotation invariance*, meaning that a descriptor should encode information about the shape and not about the pose, and it is a key property for descriptors to carry out their work. In order to achieve it, some authors sacrifice part of the informativeness, some others try to develop a process that is inherently rotation invariant, while a third group relies on the definition of local systems of Cartesian coordinates, called *Local Reference Frames* (LRFs), which provide a canonical orientation for every local support. The last approach is highly researched on, and aims to create an invariant descriptor by coupling an informative description of the support with a way to orientate the points under consideration.

1.1 Objectives

In the past, a lot of effort has been put into defining high-performing handcrafted Local Reference Frames and descriptors. Recently, following their success in advancing the state of the art for other Computer Vision tasks, there is an increasing interest into employing Artificial Neural Networks to achieve better performances also in surface matching. While interesting results with learned descriptors have already been published, there is still quite little going on about learned LRFs: there exist plenty of methods to compute them but they are all based on handcrafted algorithms, with the only exception of a recent proposal that deploys neural networks. Yet, it feeds them with manually engineered features, and therefore it does not fully benefit from the modern end-to-end learning strategies. Learned descriptors have allowed for advancements in matching precision and recall, and a successful end-to-end learned solution has been proposed for the first time last year by the team of CV Lab (University of Bologna) [26], showing better results than methods that feed manually engineered features to a network. Thus, the objective of this thesis is to develop an end-to-end learned LRF for point clouds, leveraging the architecture described in [26] and [35], and assess its performances compared to the current state of the art. The final objective is to do so with *unsupervised* learning methods, because they do not require ground-truth data, which is cumbersome to obtain for surface matching, but partial results deriving from supervised methods will be presented as well.

1.2 Thesis outline

The first part of this thesis (Chapter 2) will briefly review the most important existing works on LRFs for point clouds, showing their working princples and the absence of fully learned LRFs. Hints on local descriptors will be given as well.
The second part (Chapter 3) will present a mathematical explanation of the concepts that underlie the proposed method, followed, in Chapter 4, by a description of such method, of the employed neural network and of the training and benchmarking processes.
The last part (Chapter 5) will describe the datasets used for the experimental validation, and show the results.

Chapter 2

Related works

A wide variety of different LRFs have been proposed in the literature. This chapter will list the most influential and performant ones, describing their working principles. The last section will also mention some local descriptors and the use they make of LRFs, showing the lack of a fully learned LRF.

2.1 Local Reference Frames

As stated in [18] and [33], hand-crafted LRF estimation methods can be divided as based upon *Covariance Analysis* (CA) or *Geometric Attributes* (GA). The former group includes the LRF used by *Mian et al.* [19], SHOT [29] [24], EM [20], RoPS [14], along with many others, while the latter group includes PS [3], Board [22], FLARE [21], TOLDI [34] and GFrames [18]. To the best of our knowledge, the only example of a data-driven approach for LRFs is represented by the work of *Zhu et al.*, posted as pre-print on ArXiv in January 2020 [37], that will also be briefly presented in this chapter.

Definition 2.1. (LRF). [18] Given a point cloud $\mathcal{P}$, the LRF of point $p \in \mathcal{P}$ is defined as

$$\mathcal{L}(p) = \{\hat{\mathbf{x}}(p), \hat{\mathbf{y}}(p), \hat{\mathbf{z}}(p)\} \tag{2.1}$$

where $\hat{\mathbf{x}}(p), \hat{\mathbf{y}}(p), \hat{\mathbf{z}}(p)$ are the orthogonal axes of the coordinate system, satisfying the right-hand rule $\hat{\mathbf{y}} = \hat{\mathbf{z}} \wedge \hat{\mathbf{x}}$.[1]

Thus, defining an LRF means defining a way to compute each one of its axes. Moreover, for the following sections:

- let $p \in \mathcal{P}$ be the point for which we want to compute the LRF,

[1]Bold means vector notation (column matrices) and the circumflex means unit vector. Brackets with the indication of the point can be omitted if the point is not ambiguous.

- let $\mathcal{N}_R(p)$ be the support of radius R around p,
- let $\mathbf{p}_i \in \mathcal{N}_R(p)$ for $i = 0, \dots, k$,
- let $\hat{\mathbf{n}}(p)$ be the normal to the surface at point p.

2.1.1 CA-based methods

All the methods belonging to this category share the need of a covariance matrix of the points in the local support, and they differ from each other in the way they use this information to compute the axes.

Mian [19]. It has been proposed in 2010, and basically employs Principal Component Analysis in order to determine 3 orthogonal unit vectors. In fact, given a point p and its local support, it computes the normalized eigenvectors of the following covariance matrix:

$$\boldsymbol{\Sigma}_{\boldsymbol{b}_p} = \frac{1}{k} \sum_{i=0}^{k} (\boldsymbol{p}_i - \boldsymbol{b}_p)(\boldsymbol{p}_i - \boldsymbol{b}_p)^T \tag{2.2}$$

with $\boldsymbol{b}_p$ being the the barycenter of the points in $\mathcal{N}_R(p)$:

$$\boldsymbol{b}_p = \frac{1}{k} \sum_{i=0}^{k} \boldsymbol{p}_i \tag{2.3}$$

The problem with this method is that eigenvectors define the direction only, whereas their sign is ambiguous. Disambiguation happens for the z axis only, that is chosen so as to have a positive product with the normal $\hat{\boldsymbol{n}}(p)$, but ambiguity remains for the other axes, implying that this method potentially defines multiple LRFs for a single point, a fact that greatly increases the time requirements to match points in different point clouds [21] [33].

SHOT [29] [24] It has also been proposed in 2010. It is the first CA method that removes the ambiguities in the axes [29] [33], and it has become one of the most popular LRFs in the literature. It computes a slightly different, weighted covariance matrix, that proved to be more robust against clutter:

$$\boldsymbol{\Sigma}_{\boldsymbol{pw}} = \frac{1}{\sum_{i=0}^{k}(R - d_i)} \sum_{i=0}^{k} (R - d_i)(\boldsymbol{p}_i - \boldsymbol{p})(\boldsymbol{p}_i - \boldsymbol{p})^T \tag{2.4}$$

with the L2 distance $d_i = \|\boldsymbol{p}_i - \boldsymbol{p}\|_2 < R$. Notice that the barycenter has been substituted by the the point p, thus also reducing computation times.

The eigenvectors resulting from the covariance matrix are assigned in decreasing eigenvalue order to $\mathbf{x}^+, \mathbf{y}^+, \mathbf{z}^+$, while their opposites are referred to as $\mathbf{x}^-, \mathbf{y}^-, \mathbf{z}^-$ Disambiguation is carried out on the $\mathbf{x}$ and $\mathbf{z}$ axes as follows:

$$\begin{aligned} S_x^+ &:= \{i : d_i < R \wedge (\boldsymbol{p}_i - \boldsymbol{p}) \cdot \mathbf{x}^+ \geq 0\} \\ S_x^- &:= \{i : d_i < R \wedge (\boldsymbol{p}_i - \boldsymbol{p}) \cdot \mathbf{x}^- > 0\} \\ \hat{\mathbf{x}} &= \begin{cases} \mathbf{x}^+ & \text{if } |S_x^+| \geq |S_x^-| \\ \mathbf{x}^- & \text{otherwise} \end{cases} \end{aligned} \tag{2.5}$$

and analogously for $\mathbf{z}$, meaning that the sign of an axis has to be coherent with the majority of the points in the support. For the remaining axis, $\hat{\mathbf{y}} = \hat{\mathbf{z}} \wedge \hat{\mathbf{x}}$.
This method provides a good balance in terms of efficiency, repeatability and robustness to clutter and occlusion. However, its feature matching performance is not always paralleled by its repeatability [33].

2.1.2 GA-based methods

These methods typically compute their axes successively, possibly in different ways, leveraging geometric attributes such as normals and signed distances [33].

Board [22]. Proposed in 2011, it is the result of an extensive analysis of existing proposals. It is based on the definition of two different supports: $\mathcal{N}_{R_z}(p)$ and $\mathcal{N}_{R_x}(p)$, typically set to have R_z small and $R_x > R_z$.
Firstly, it fits a plane on the points of $\mathcal{N}_{R_z}(p)$ and takes its normal ($\mathbf{z}^+/\mathbf{z}^-$). It then computes the average normal $\tilde{\mathbf{n}}$ across support points, and chooses between $\mathbf{z}^+$ and $\mathbf{z}^-$ so to have a positive inner product with $\tilde{\mathbf{n}}$.
Secondly, for each point $\mathbf{p}_i \in \mathcal{N}_{R_x}(p)$ it computes the cosine of the angle between $\hat{\mathbf{z}}$ and the average normal at that point, named $\tilde{\mathbf{n}}(\mathbf{p}_i)$. The point $\mathbf{p}_{min}$ with the lowest cosine (highest angle) is selected, and the vector from $\mathbf{p}$ to $\mathbf{p}_{min}$ is projected onto the tangent plane on $\mathbf{p}$ and normalized, to become the $\mathbf{x}$ axis. Since $\mathbf{p}_{min}$ almost always lies in the outer regions of the support, the computation is sped up by considering only the support points with distance from $\mathbf{p}$ greater than a defined threshold, usually $0.85 \cdot R_x$.
As usual, the $\mathbf{y}$ axis is computed by cross-product.
Board also takes into account the possibility to have missing regions, defining a heuristic procedure to estimate if $\mathbf{p}_{min}$ lies in the missing region or not.

FLARE [21]. Proposed in 2012, it is a modified version of Board, by the same authors. It computes the $\mathbf{z}$ axis as in Board, but for the $\mathbf{x}$ axis it uses the signed

distance instead of the cosine with the normal.
The signed distance of a point $\mathbf{p}_i$ is defined as:

$$d_{SD}(\mathbf{p}_i) = \mathbf{pp}_i \cdot \hat{\mathbf{z}} \tag{2.6}$$

with $\mathbf{pp}_i$ being the vector going from $\mathbf{p}$ to $\mathbf{p}_i$. The algorithm selects the point with the largest signed distance, which is the point whose component parallel to the $\mathbf{z}$ axis is the largest, meaning the most distant point from the tangent plane at point $\mathbf{p}$. As before, the projection of this point to the tangent plane is normalized and taken as the $\mathbf{x}$ axis.
FLARE has proven to be more repeatable than Board, but they both have similar problems with noise and outliers [33].

TOLDI [34]. Proposed in 2016, it computes the $\mathbf{z}$ axis in a very similar fashion to CA methods, as it computes the covariance matrix (using the barycenter) and considers as $\mathbf{z}^+$ the unit eigenvector with the smallest eigenvalue. The first difference is that this covariance is computed on $\mathcal{N}_{\frac{R}{3}}(p)$ instead of $\mathcal{N}_R(p)$. The other difference is in the disambiguation of the sign, that follows the rule:

$$\hat{\mathbf{z}} = \begin{cases} \mathbf{z}^+ & \text{if } \mathbf{z}^+ \cdot \sum_{\mathbf{p}_i \in \mathcal{N}_R(p)} \mathbf{p}_i\mathbf{p} \geq 0 \\ -\mathbf{z}^+ & \text{otherwise} \end{cases} \tag{2.7}$$

The $\mathbf{x}$ axis will lie in the tangent plane of $\mathbf{p}$ with respect to the normal $\hat{\mathbf{z}}$, but finding an orientation for such plane is more difficult than finding $\hat{\mathbf{z}}$.
The first step is the computation of the projections of the support points on the plane. For each $\mathbf{p}_i \in \mathcal{N}_R(p)$,

$$\mathbf{v}_i = \mathbf{pp}_i - (\mathbf{pp}_i \cdot \hat{\mathbf{z}}(p)) \cdot \hat{\mathbf{z}}(p) \tag{2.8}$$

The $\mathbf{x}$ axis is then computed as a weighted sum:

$$\hat{\mathbf{x}} = \frac{1}{\left\| \sum_{i=0}^{k} w_{i1} w_{i2} \mathbf{v}_i \right\|_2} \sum_{i=0}^{k} w_{i1} w_{i2} \mathbf{v}_i \tag{2.9}$$

In this sum, w_{i1} is related to the distance of the point and is designed to improve robustness to clutter, occlusion and incomplete border regions, while w_{i2} is set to make the points with larger projection distance contribute more to the $\mathbf{x}$ axis, since such distance feature is a distinctive cue and can provide high repeatability on flat regions. They are defined as follows:

$$w_{i1} = (R - \|\mathbf{p} - \mathbf{p}_i\|_2)^2 \tag{2.10}$$

$$w_{i2} = (\mathbf{pp}_i \cdot \hat{\mathbf{z}}(p))^2 \tag{2.11}$$

As usual, the **y** axis is computed by cross product.
TOLDI can produce great performances, but it has been shown to be quite sensitive to keypoint localization error [33].

GFrames [18]. It has been proposed in 2019 by a team of researchers from different institutions, including the University of Bologna.
It assumes that the point cloud (or the mesh) is a sampling of a 2D Riemannian manifold embedded in the $\mathbb{R}^3$ space. Since the method needs the areas of the mesh triangles, it works straight away with meshes, while for point clouds it uses a procedure that locally computes estimates of triangles.
The **z** axis is defined as the normal on the point, $\hat{\mathbf{n}}(p)$, while for the **x** axis it computes the following formula:

$$\mathbf{x}(p) := \frac{1}{\sum_{t_j \in \mathcal{N}_R(p)} A(t_j)} \sum_{t_j \in \mathcal{N}_R(p)} A(t_j) \nabla f(t_j) \tag{2.12}$$

with t_j a mesh triangle, $\mathcal{N}_R(p)$ the set of triangles within distance R from p, $A(t_j)$ the area of the t_j triangle and f a user-defined scalar function.
The actual $\hat{\mathbf{x}}(p)$ is computed by projecting the result of equation 2.12 on the plane defined by $\hat{\mathbf{n}}(p)$ and normalizing the projection to have a unit vector. The remaining axis is the usual cross product.
In [18], different f functions are tested, such as the mean curvature, the Gaussian curvature, the sum of total Euclidean distances, and the function of FLARE itself (average of the signed distances from tangent plane), showing great robustness and repeatability. GFrames is also particularly suited for non-rigid transformations, thanks to its flexibility.

2.1.3 Data-driven methods

As previously stated, to the best of our knowledge there exist a single public data-driven method applied to learning a Local Reference Frame for point clouds, named LRF-Net. Nevertheless, the aforementioned approach does not rely on feeding the network with the raw information from the point cloud: it feeds hand-crafted features computed on the point cloud, instead. Thus, there are currently no public proposals of end-to-end learning applied to Local Reference Frames.

LRF-Net [37]. Proposed at the beginning of 2020, it is the first data-driven approach for LRFs.

In spite of being data-driven, it still relies on a traditional approach to compute the $\mathbf{z}$ axis, that is in fact set to be equal to the normal $\hat{\mathbf{n}}(p)$ of the keypoint p.
The $\mathbf{x}$ axis, instead, is computed by the following formula:

$$\hat{\mathbf{x}}(p) = \frac{1}{\left\| \sum_{i=0}^{k} w_i \mathbf{v}_i \right\|_2} \sum_{i=0}^{k} w_i \mathbf{v}_i \tag{2.13}$$

with $\mathbf{v}_i$ defined as in equation 2.8 to be the projection of $\mathbf{pp}_i$ on the tangent plane.
The weights w_i are computed by an Artificial Neural Network, consisting of 8 fully connected layers with batch normalization and ReLUs after each layer.
The network has a 2D input layer and a 1D output layer, with the inputs being a^i_{dist} and a^i_{angle} for point $\mathbf{p}_i \in \mathcal{N}_R(p)$, defined as follows:

$$\begin{cases} a^i_{dist} = \|\mathbf{pp}_i\|_2 / R \\ a^i_{angle} = \cos(\hat{\mathbf{z}}(p), \mathbf{pp}_i) \end{cases} \tag{2.14}$$

Basically, the first one is the fraction of the distance of the i-th point with respect to the maximum distance of the support, and the second one is the cosine between the normal on the keypoint and the vector from the keypoint to the i-th point.
The network computes each weight separately.
As stated in [37], these features are complementary to each other, providing information about distance and angle for each point in the support, and are also invariant, provided that $\mathbf{z}$ is repeatable. This is crucial for the network to produce an invariant output, which is used to compute a linear combination of the $\mathbf{v}_i$s, equivariant by definition, thus providing an $\mathbf{x}$ axis that rotates along with the point cloud as long as the $\mathbf{z}$ axis is repeatable.
The network is trained in a weakly supervised, Siamese, way (more on this in Chapter 4), loading corresponding pairs of keypoints from point clouds recorded from different viewpoints, rotating each keypoint with its support according to the LRF computed as defined above, and minimizing the Chamfer Distance [1] between the two rotated local supports.

2.2 Local descriptors

Since local descriptors, which are compact representations of the neighborhood of a point, are out of the scope of this work, this section will not go into detail about how they are computed.

It's worth noting that most of the Local Reference Frames deliver their best performances under very specific circumstances and/or particular applications [33],

or when coupled with their respective descriptor: in fact, part of the papers that define a Local Reference Frame do so in order to support the definition of a custom local descriptor. Mian, SHOT and TOLDI, for example, define both a LRF and a descriptor, whereas Board, FLARE and GFrames do not.
Other notable descriptors, defined separately from LRFs, are USC (Unique Shape Context, 2010) [28], CGF (Compact Geometric Features, 2017) [16] and 3DSmoothNet (2018) [12]. Both the first two use SHOT, and are hand-crafted, while the last one uses a slightly modified version of TOLDI and is a learned descriptor. Other learned descriptors, that do not rely on LRFs, are PFFNet [11], PPF-FoldNet [10] and PRIN [35], all from 2018.

An interesting example of a non-invariant learned descriptor that does not rely on hand-crafted features and that also defines its own LRF, is described in [26] by *Spezialetti et al.*[2]. By taking advantage of spherical convolutions, their network processes a spherical signal (directly computed from the point cloud), and computes a 512-dimensional codeword that is used as an equivariant descriptor. Being equivariant, it is theoretically possible to compute the rotation between two descriptors (or intermediate feature maps) by computing the $\arg\max$ of the correlation between the two. However, this is practically impossible to do when matching descriptors, since doing that for every possible pair of descriptors requires a tremendous amount of computing time. Because of this reason, they decided to choose a heuristic on the descriptor, for example the highest value. If the descriptor is perfectly equivariant, this maximum will be present in different bins in the descriptors of corresponding keypoints in two rotated point clouds. Thanks to the structure of the network and to the mathematical properties of spherical convolutions, it is possible to analyze the position of these maxima and infer the rotation operation that aligns them. As a consequence, the position of the maximum can be used to define a Local Reference Frame in such a way that the descriptor, orientated according to the LRF, becomes invariant to input rotations. While theoretically interesting, the highest value alone is not robust enough to reach good results in practice, and a different heuristic has been used, involving the computation of the density of the top k values.
Such learned descriptor, coupled with the proposed LRF, achieves great performances, but they can be further improved by using FLARE as the LRF, as shown in [26]. Due to the promising mathematical properties of the network, though, it seems to be possible to use the proposed framework to specifically learn a Local Reference Frame, and this is where this thesis lays its foundations.

To sum up, here is a table (figure 2.1) that categorizes all the cited methods. Hand-crafted LRFs are divided as in [33]; each descriptor is followed by the LRF it makes use of, except for PPF-based descriptors and PRIN; lastly, there is an

[2]CVLab, University of Bologna

evident gap in learned Local Reference Frames, as the present method doesn't use end-to-end learning and is employed for just one out the three axes of the coordinate system.

As of now, FLARE, TOLDI and GFrames are considered state-of-the-art methods, with FLARE being selected for direct comparisons with the method developed in the next chapters.

	Descriptors (LRF used, if any)	Local Reference Frames	
		CA:	GA:
Hand-crafted	SHOT (SHOT) USC (SHOT) CGF (SHOT) TOLDI (TOLDI)	Mian SHOT EM RoPS	Board FLARE TOLDI GFrames
Learned	3DSmoothNet (TOLDI) *Spezialetti et al.* (own) PPFNet PPF-FoldNet PRIN	LRF-Net (partially)	

Figure 2.1: Categorization of Local Reference Frames and Descriptors.

Chapter 3

Theoretical background

The aim of this chapter is to provide all the required concepts to understand Spherical Convolutional Neural Networks and the architecture proposed in Chapter 4.

Most of this chapter is based on the renowned book "Deep Learning", by Goodfellow, Bengio and Courville [13] and on the work of Cohen, Geiger, Koehler and Welling about group equivariant neural networks and Spherical CNNs [4] [5] [6], although with changes in notation. For this reason, citations to the aforementioned works will be avoided, whereas different sources will be cited when used.

The chapter will first present a brief recap on standard convolution, correlation and CNNs: their short description is mainly provided to set the notation for the equations, and to offer a parallel presentation of them and the core theoretical part, Spherical CNNs.
The chapter will then describe the possible ways to represent rotations in 3D space, followed by the definition of a quality measure for Local Reference Frames, that will be used to assess the performance of the proposed method.

3.1 Standard CNNs

Definition 3.1. (1D Convolution). For continuous real-valued scalar signals $f, g : \mathbb{R} \to \mathbb{R}$, convolution is defined as:

$$h(t) = [f * g](t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \tag{3.1}$$

while for discrete signals it is sufficient to substitute the integral with a sum.

The result of convolution is not a number, it is a function $h : \mathbb{R} \to \mathbb{R}$ whose domain is the same of f and g. The output of h for a certain input t is the integral

(or sum) of the product between f and g, with the latter being flipped around the origin and translated by t. As a consequence, it is possible to visualize the h function as a 1D signal, with each value computed as stated above.
Convolution is a commutative operation.

Definition 3.2. (1D Correlation). Similarly, correlation is defined as:

$$i(t) = [f \star g](t) = \int_{-\infty}^{+\infty} f(\tau)g(\tau + t)d\tau \tag{3.2}$$

with $i : \mathbb{R} \to \mathbb{R}$. Again with a sum in case of discrete signals.

As a difference, correlation does not flip the g function, and is not commutative.

These two operations are strongly connected to each other, and are often used interchangeably if the g function is symmetric. In fact, it is possible to compute convolution using correlation by simply flipping the second function and inverting the order (so that the first function gets translated instead of the second), and vice-versa, with flipping being unnecessary if g is symmetric.

All of this can be straightforwardly generalized to signals on the plane, and thus it can be applied to images, that are in fact 2D discrete signals. All the following formulas can be expressed with summations instead of integrals to cope with discrete signals.

Definition 3.3. (2D Convolution). Let $f, g : \mathbb{R}^2 \to \mathbb{R}$, the 2D Convolution is defined as:

$$h(x, y) = [f * g](x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\alpha, \beta)g(x - \alpha, y - \beta)d\alpha d\beta \tag{3.3}$$

with $x, y \in \mathbb{R}$ and $h : \mathbb{R}^2 \to \mathbb{R}$.

Definition 3.4. (2D Correlation). Similarly, the 2D Correlation is defined as:

$$i(x, y) = [f \star g](x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\alpha, \beta)g(\alpha + x, \beta + y)d\alpha d\beta \tag{3.4}$$

with $i : \mathbb{R}^2 \to \mathbb{R}$.

As for the 1D case, computing a convolution between two signals means overlapping (by multiplication) the input functions and computing their sum or integral, with the second function being flipped around the origin and translated by a quantity specified by the input of the convolution function.
If we say that f is an image and g is a kernel (or filter), i.e. two real-valued discrete functions on the plane, the output of a correlation/convolution can be

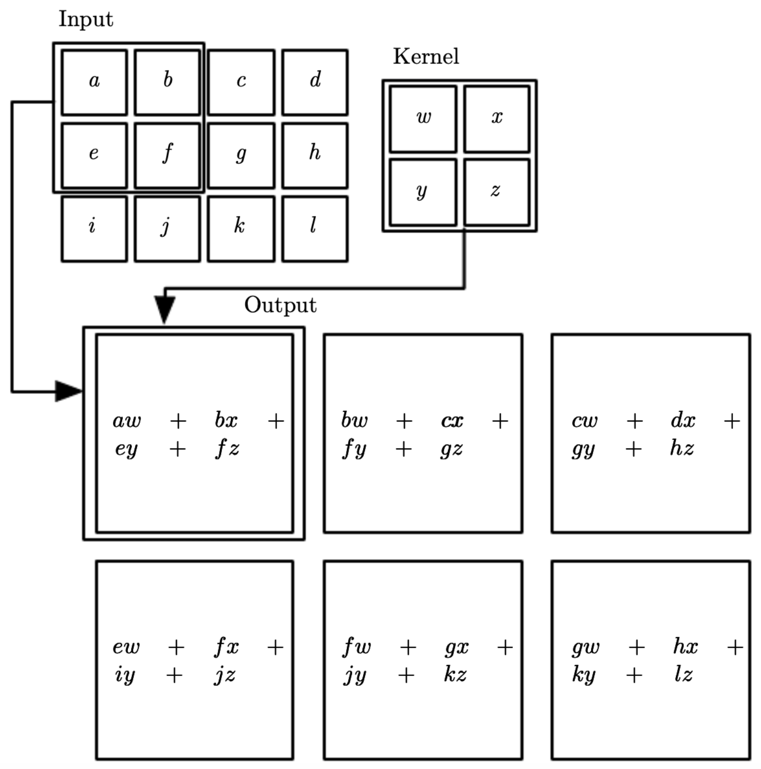


Figure 3.1: Visualization of 2D discrete correlation, image from [13]. Notice that in case of a finite domain the output dimensions are smaller than the input, because placing the kernel out of the image is a (usually) not allowed operation.

easily visualized as in figure 3.1. It is an image itself, whose pixels are computed by multiplying the original image and an accordingly shifted kernel. The actual operation in this image is a correlation between the kernel and the image (not commutative), with convolution being analogous (with a flipped kernel).

Convolutional Neural Networks (CNNs) take their origin from the aforementioned operations of convolution and correlation and, in spite of the name, they usually employ correlation as the layer operation, while convolution arises during backpropagation. In fact, the key difference between a MultiLayer Perceptron (MLP) and a CNN is that the former computes a matrix multiplication between inputs and weights as the layer operation, while the latter uses correlation between the input to the layer and a set of *filters*, that are simply small kernels like in figure 3.1.
In the previously defined correlation, an input of (x, y) means translating the filter by $(-x, -y)$, thus practically flipping the output map with respect to the origin. In order to have a direct correspondence between a point of the output and a point

of the input, as in figure 3.1, the usual operation that happens in CNNs is a variant of correlation that corresponds to a correlation with commuted input functions.

Definition 3.5. (CNN Layer Operation). Consider a CNN with n layers. The input of each layer l is a feature map $f : \mathbb{Z}^2 \to \mathbb{R}^{K^l}$, the number of output channels for layer l is K^{l+1}, and the K^{l+1} filters for layer l are $\psi^i : \mathbb{Z}^2 \to \mathbb{R}^{K^l}$ for $i = 1, \dots, K^{l+1}$. The output of layer l is defined as:

$$[f \star \psi^i](x, y) = \sum_{\alpha=-\infty}^{+\infty} \sum_{\beta=-\infty}^{+\infty} \sum_{k=1}^{K^l} f_k(\alpha, \beta) \psi_k^i(\alpha - x, \beta - y) \tag{3.5}$$

with $i = 1, \dots, K^{l+1}$. Feature maps and filters are considered to be defined everywhere in $\mathbb{Z}^2$ and 0-valued in out-of-map domain points.
Interestingly, each filter has as many channels as the input, and each correlation produces a real-valued output. Thus, each correlation is computed on all the input channels, which are eventually summed together to produce the output.

The parameters learned by a CNN (i.e. the weights) are the values of the filters. This approach has 3 main advantages:

- It greatly reduces the number of parameters, because filters are chosen to be much smaller than the input images;
- It allows for weight sharing, because each pixel of the output is computed with the same weights but with a different input (portion of the input image);
- It is **equivariant** to translations, meaning that if a certain input produces a certain output when correlated with a filter, then the same input in a different portion of the image will produce the same output but in a correspondingly different portion of the output feature map.

The last property is the most interesting for the purposes of this work, so here is a more precise definition of it.

Definition 3.6. (Equivariance). [31] [2] Let G be a set of transformations. Consider a function $\Phi : \mathcal{X} \to \mathcal{Y}$ that maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. A transformation $g \in G$ can be applied to any $x \in \mathcal{X}$ by means of the operator $\mathcal{T}_g^{\mathcal{X}} : \mathcal{X} \to \mathcal{X}$, such that x is transformed to $\mathcal{T}_g^{\mathcal{X}}[x]$, and analogously for $y \in \mathcal{Y}$ transformed to $\mathcal{T}_g^{\mathcal{Y}}[y]$. Φ is said to be equivariant to G if:

$$\Phi(\mathcal{T}_g^{\mathcal{X}}[x]) = \mathcal{T}_g^{\mathcal{Y}}[\Phi(x)] \tag{3.6}$$

The operators $\mathcal{T}_g^{\mathcal{X}}[x]$ and $\mathcal{T}_g^{\mathcal{Y}}[y]$ are basically the application of the same transformation g to elements from different domains, and are related by the equation above. **Invariance**, instead, is a particular case of equivariance in which the operator $\mathcal{T}_g^{\mathcal{Y}}[y]$ is the identity transformation, meaning that the result of Φ doesn't change with a transformation of the input.

It is possible to apply this definition of equivariance to the operation defined in equation 3.5, and thus to Convolutional Neural Networks. In fact, if we consider Φ as the layer operation, $x \in \mathcal{X}$ as a point of the input image, $y \in \mathcal{Y}$ as a point of the output, and G as the set of 2D translations (with $\mathcal{T}_g^{\mathcal{X}}[x] = \mathcal{T}_g^{\mathcal{Y}}[y]$ as the application of such translations), it is trivial to see that CNNs are translation-equivariant. By defining the operator $L_{t,s}$ to take a 2D function and translate it by (t, s) as

$$[L_{t,s}f](x, y) = f(x - t, y - s) \tag{3.7}$$

translation-equivariance can be proved as follows, by substituting α with $\alpha + t$ and β with $\beta + s$ (channels are dropped to slim down notation):

$$\begin{aligned}
[L_{t,s}f \star \psi](x, y) &= \sum_{\alpha=-\infty}^{+\infty} \sum_{\beta=-\infty}^{+\infty} f(\alpha - t, \beta - s)\psi(\alpha - x, \beta - y) \\
&= \sum_{\alpha=-\infty}^{+\infty} \sum_{\beta=-\infty}^{+\infty} f(\alpha, \beta)\psi(\alpha + t - x, \beta + s - y) \\
&= \sum_{\alpha=-\infty}^{+\infty} \sum_{\beta=-\infty}^{+\infty} f(\alpha, \beta)\psi(\alpha - (x - t), \beta - (y - s)) \\
&= [f \star \psi](x - t, y - s) \\
&= [L_{t,s}[f \star \psi]](x, y)
\end{aligned} \tag{3.8}$$

It is also worth noting that in a convolution or correlation, let's say for continuous 2D signals, the domain of the input functions f and g is the plane $\mathbb{R}^2$, whereas the domain of the output function (so the domain of the convolution or correlation itself) is the subgroup of 2D translations (because computing the operation for (x, y) means translating the g function by (x, y)), which is isomorphic to the plane $\mathbb{R}^2$. This is a subtle difference, because such subgroup acts on itself, but it will be useful for the next section as this is not always the case.

Going back to the aim of this thesis, which is finding a Local Reference Frame, it is important to notice what follows: given a keypoint with its local support, a LRF will set 3 axes, defining an orientation for such keypoint[1]. If the same keypoint is rotated and then a LRF is computed, we expect the LRF to produce

[1]By orientating (or rotating) a keypoint, we intend with its own local support.

the same axes as before with respect to the shape of the support, meaning that the two keypoints can be perfectly aligned. In other words, this means that the LRF should rotate in the 3D space along with the input, and thus that the LRF should possess the **3D rotation-equivariance** property. How to achieve it?

3.2 Spherical CNNs

Cohen et al. [5] [6] have shown that it is possible to achieve equivariance to rotations in space by using spherical signals and spherical convolutions. Their theory has been developed for spherical images, i.e. images that are defined on a sphere instead of a plane, because the usual way to handle these images was to use standard CNNs with multiple planar projections, a sub-optimal approach due to planar projections producing distortion. By using their method, it is possible to define a natural convolution on the sphere itself.
The input format for this work, though, is not a spherical signal, it is a point cloud. However, there are multiple ways to transform point clouds into spherical signals: ray casting [6] and spherical voxelization [35] are two examples, with the latter being deployed in this context and described in the next chapter. Thus, we can consider to be able to use spherical signals and continue with the dissertation.

Definition 3.7. (S^2). The unit sphere S^2 is defined as the set of points $x \in \mathbb{R}^3$ with norm 1. It's a 2-dimensional manifold that can be parametrized with spherical coordinates $\alpha \in [0, 2\pi]$ and $\beta \in [0, \pi]$.

Definition 3.8. (Spherical Signal). A spherical signal is a vector-valued function $f : S^2 \to \mathbb{R}^K$, with K as the number of channels.

Definition 3.9. (SO(3)). 3D linear transformations can be represented by 3x3 matrices, and the set of 3x3 matrices with orthonormal columns, along with matrix multiplication, forms the Orthogonal group O(3). This group is made of two connected components, one with determinant 1 and the other one with determinant -1. The first component also contains the identity matrix, and thus is a subgroup containing all the rotations in 3D space. This subgroup is called SO(3), the Special Orthogonal group for 3 dimensions, also known as *rotation group*.

SO(3) is a 3-dimensional manifold, and can be parametrized in many different ways. The following formulas will consider a parametrization wih ZYZ-Euler angles $\alpha \in [0, 2\pi]$, $\beta \in [0, \pi]$ and $\gamma \in [0, 2\pi]$. Moreover, in order to apply rotations to spherical signals by matrix multiplication, points on the sphere S^2 can be considered as 3D unit vectors. Rotations and their representations are further described in section 3.3.

Definition 3.10. (Rotation of Spherical Signals). Given a spherical signal $f : S^2 \to \mathbb{R}^K$ and a rotation $R \in SO(3)$, the rotation operator L_R rotates the signal as:

$$[L_R f](x) = f(R^{-1}x) \tag{3.9}$$

with $x \in S^2$. By rotating the input by the inverse of R, it effectively rotates the signal by R. Notice that $L_{RR'} = L_R L_{R'}$.

Definition 3.11. (Inner Product). Spherical signals constitute a vector space. Thus, given two K-valued spherical signals $f, \psi : S^2 \to \mathbb{R}^K$, their inner product can be defined as:

$$\langle \psi, f \rangle = \int_{S^2} \sum_{k=1}^{K} \psi_k(x) f_k(x) dx \tag{3.10}$$

with dx as the standard rotation invariant integration measure on the sphere, $dx = d\alpha \sin(\beta) d\beta / 4\pi$ in spherical coordinates α and β.

An invariant measure means that the integral doesn't change with rotations, and thus that the inner product between two functions doesn't change if they are rotated by the same rotation R:

$$\begin{aligned} \langle L_R \psi, f \rangle &= \int_{S^2} \sum_{k=1}^{K} \psi_k(R^{-1}x) f_k(x) dx \\ &= \int_{S^2} \sum_{k=1}^{K} \psi_k(x) f_k(Rx) dx \\ &= \langle \psi, L_{R^{-1}} f \rangle \end{aligned} \tag{3.11}$$

With the concepts defined above, it is possible to proceed to define a correlation for spherical signals, that will constitute the fundamental block of Spherical CNNs.

Definition 3.12. (Spherical Correlation). Let $f, \psi : S^2 \to \mathbb{R}^K$, their spherical correlation is:

$$[\psi \star f](R) = \langle L_R \psi, f \rangle = \int_{S^2} \sum_{k=1}^{K} \psi_k(R^{-1}x) f_k(x) dx \tag{3.12}$$

Let's now compare this function to equation 3.5, that is the standard CNN correlation. They look very similar to each other, with the spherical correlation that looks to be its direct generalization to spherical signals. But here the subtle difference about the domain of the resulting function arises. In fact, while standard correlation is defined on the subgroup of 2D rotations, that is isomorphic to $\mathbb{R}^2$ and can act on itself, there is no way to define a group on the sphere S^2, so the domain of a spherical correlation is the set of 3D rotations: SO(3). It is possible to visualize this by thinking of two spheres, that can be multiplied to each other after being rotated in all 3 dimensions. In the 2D domain it would be equivalent to being able not only to translate the maps, but also to rotate them around one of their points.

By using this spherical correlation as the layer operation for Spherical CNNs, we immediately face a change in the domain of the signals. Indeed, despite the fact that the input to the first layer is an S^2 signal, and so are its filters, the output of such layer will be a signal defined on SO(3), with as many channels as the number of filters. We should then repeat this process for SO(3) signals, as follows.

Definition 3.13. (Rotation of SO(3) Signals). Given an SO(3) signal $f : SO(3) \to \mathbb{R}^K$ and a rotation $R \in SO(3)$, the rotation operator L_R can be straightforwardly generalized to rotate the SO(3) signal as:

$$[L_R f](Q) = f(R^{-1}Q) \tag{3.13}$$

with $Q \in SO(3)$. Notice that, in this case, both R and Q belong to SO(3).

Definition 3.14. (SO(3) Correlation). Let $f, \psi : SO(3) \to \mathbb{R}^K$, their spherical correlation is:

$$[\psi \star f](R) = \langle L_R \psi, f \rangle = \int_{SO(3)} \sum_{k=1}^{K} \psi_k(R^{-1}Q) f_k(Q) dQ \tag{3.14}$$

with dQ as the invariant measure on SO(3). If we parametrize it with the aforementioned ZYZ-Euler angles, it is $dQ = d\alpha \sin(\beta)\beta d\gamma/(8\pi^2)$.

This time, the domain of SO(3) correlation is SO(3) itself, which is a group. At this point, we have both layer operations required to build a Spherical CNN, whose peculiar characteristic is the **equivariance to rotations**, and is thus suitable to our needs. By recalling the definition of equivariance (definition 3.6), and considering Φ as the correlation, x as point on the sphere or on SO(3), y as a point on SO(3), G as SO(3) itself, and $\mathcal{T}_g$ as L_Q, the proof proceeds as follows.

$$[\psi \star [L_Q f]](R) = \langle L_R \psi, L_Q f \rangle \tag{3.15}$$

Using the invariance property of the inner product (equation 3.11) and the property at the end of definition 3.12, we have:

$$\langle L_R \psi, L_Q f \rangle = \langle L_{Q^{-1}R} \psi, f \rangle \tag{3.16}$$

By using, respectively, the definition of correlation (3.12 and 3.14) and the definition of rotation (3.10 and 3.13), we can end the proof:

$$\langle L_{Q^{-1}R} \psi, f \rangle = [\psi \star f](Q^{-1}R) = [L_Q[\psi \star f]](R) \tag{3.17}$$

This proof applies to both spherical and SO(3) correlation.

We have now a complete definition of rotation-equivariant Spherical CNNs, whose first layer computes the spherical correlation and the following layers compute the SO(3) correlation. In these peculiar CNNs, the first layer has spherical signals as filters, whereas the other layers have signals on SO(3).

In practice, though, to speed up the computation, layers employing spherical or SO(3) correlation do not directly compute those transformations. Instead, they compute their results using a Generalized Fast Fourier Transform, but this is out of the scope of this work. More on this can be found in [6].

3.3 Representations of rotations

There are many different ways to represent rotations in the 3D space. This section will briefly show the ones used for this work, namely: Euler angles, 3x3 matrices, axis-angle and Rodrigues' vectors. Quaternions are another very interesting representation, especially for machine learning purposes, but they will not be covered since they haven't been used.

Before starting, it's worth noting that a Local Reference Frame can be identified exactly as the rotation that transforms the original coordinate system of the point cloud into the new coordinate system, or as the inverse of such rotation. So, the following representations also apply to LRFs.

3.3.1 Euler angles

Let us have a coordinate system (or Local Reference Frame, they are synonyms in this context) to be rotated. It is possible to visualize its rotation by duplicating the coordinate system, fixing one copy, and rotating the other. Since rotations in 3D space have 3 degrees of freedom, only 3 parameters are necessary to characterize a rotation. If we imagine to rotate the moving copy around the Z axis first, then around the Y axis, and finally around the X axis, the 2D rotation angles around such axes are known as the 3 Euler angles α, β and γ. Consequently, a full rotation will be the composition of such 3 transformations (XYZ).
As previously stated, in this context we will consider the ZYZ-Euler angles, meaning that both the first and the last angles will refer to rotations around the Z axis. This does not lower the degrees of freedom as long as the rotation around Y is not null, as will be clearer when dealing with matrices.

3.3.2 3x3 matrices

2x2 rotation matrices can be straightforwardly generalized to 3D rotations by combining 2D rotations around different axes. If we take the previously defined XYZ-Euler angles, and by defining $\mathbf{R}_x(\alpha)$ as the rotation matrix of α around the X axis, and analogously for the other axes, a full 3D rotation matrix is:

$$\mathbf{R} = \mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma) \tag{3.18}$$

where:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{3.19}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{3.20}$$

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

In case of a ZYZ representation it is easy to see that, as long as the Y rotation is not null (and thus the $\mathbf{R}_y$ matrix is not an identity matrix), all the rows and columns of the resulting rotation $\mathbf{R}$ have values that, overall, depend on 3 different parameters, effectively producing a new coordinate system with 3 degrees of freedom.

The rotation matrix is the most used representation to identify Local Reference Frames throughout this work: if we use the rotation matrix that transforms the original axes into the LRF, then the axes of the LRF are the columns of such matrix; whereas if we use the transposed[2], the axes are the rows, and such matrix can be directly applied to points to change their coordinate system into the new one. Definition 2.1 considers the LRF as the former case, thus with axes on columns, whereas the code used for learning and benchmarking will mostly use the latter for practical reasons involving arrays.

3.3.3 Axis-angle representation

Another very intuitive way to refer to a 3D rotation is the axis-angle representation. In fact, defining an axis (a line through the origin) and a rotation angle along that axis is sufficient to completely characterize a rotation in $\mathbb{R}^3$. Having a

[2]Rotation matrices are orthogonal and, as such, their inverse equals their transpose: $\mathbf{R}^{-1} = \mathbf{R}^T$.

rotation matrix $\mathbf{R}$, the points of the rotation axis stay the same before and after the rotation, and are thus the invariants of the rotation. As a consequence, for such points $\mathbf{p}$ we have:

$$\mathbf{Rp} = \mathbf{Ip} \tag{3.22}$$

where $\mathbf{I}$ is the 3x3 identity matrix.
By solving the equation above we can find the eigenvectors corresponding to the unitary eigenvalue, and use one of them to refer to the rotation axis.

To find the rotation angle θ, it is possible to use the following formula:

$$\mathrm{Tr}(\mathbf{R}) = 1 + 2\cos\theta \tag{3.23}$$

where Tr is the trace operation.

3.3.4 Rodrigues' vector

This representation will be very useful for visualization purposes, because it allows to represent a rotation as a single 3x1 vector, whose elements are called *Euler-Rodrigues parameters.*

Definition 3.15. (Rodrigues' vector.) [9] Let $\hat{\mathbf{u}} = [u_x, u_y, u_z]^T$ be the 3D unit vector that represents the rotation axis in the axis-angle representation of rotations, and let θ be the angle. The Rodrigues' vector $\mathbf{b}$ is defined as:

$$\mathbf{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} u_x \tan\frac{\theta}{2} \\ u_y \tan\frac{\theta}{2} \\ u_z \tan\frac{\theta}{2} \end{bmatrix} = \tan\frac{\theta}{2} \cdot \hat{\mathbf{u}} \tag{3.24}$$

The vector $\mathbf{b}$ thus represents the rotation axis, with norm equal to the tangent of half of the rotation angle.

In Chapter 5, this representation will be used to show the output feature maps, but the tangent will be dropped in order to have a better scale for visualization.

3.4 LRF quality measure: repeatability

The last thing we need is a measure of how good a LRF is performing.

In the literature, plenty of different measures have been proposed, mostly involving the computation of the distance between the computed LRFs on a keypoint from two different (i.e. rotated) viewpoints. But, as noticed in [22] and [21], the problem with this approach is that it doesn't take into account the fact that LRFs work only if they are almost perfectly equivariant to rotations of the keypoint, and are able to align different views of the same keypoint. If a pair of rotated keypoints produces LRFs that do not match, it doesn't matter if it is by 30 degrees or by 180 degrees (the maximum possible): they are simply not repeatable, and thus they would not work if used for pose estimation or keypoint matching. For this reason, *Petrelli and Di Stefano* proposed in [21] a new quality measure, called *repeatability.* In this work, their definition has been extended to include 2 axes of the LRF instead of a single one, because here the normal axis is not computed in a standard way as in methods described in Chapter 2. Computing it also for the third axis is not needed, since it is uniquely identified from the other two.

Definition 3.16. (LRF Repeatability.) Let D be a dataset with M different views $\{V_1, V_2, \dots, V_M\}$, and let $p_h \in V_h$ and $p_k \in V_k$ (with $h, k = 1, \dots, M$ and $h \neq k$) be the same point p in two different views. Suppose to have an algorithm to determine a LRF and let $\mathcal{L}(p_h), \mathcal{L}(p_k)$ be the computed LRFs for such point[3]. We say that the LRFs for corresponding points p_h and p_k are repeatable if:

$$\begin{gathered}\hat{\mathbf{x}}(p_h) \cdot \hat{\mathbf{x}}(p_k) \geq Th \\ and \\ \hat{\mathbf{x}}(p_h) \cdot \hat{\mathbf{x}}(p_k) \geq Th\end{gathered} \tag{3.25}$$

with Th as a user defined threshold, usually set to 0.97 in this context. The dot means scalar product, and thus produces the cosine between the axes, since their norm is unitary.

Intuitively, this means that a LRF is repeatable if, when computed on corresponding points, it is able to align their coordinate systems with a minimum cosine of Th for each axis.
The repeatability on a whole scene or dataset is then computed as the percentage of repeatable keypoints over the total number of identified keypoints.

[3]See definition 2.1.

Chapter 4

Architecture and methodologies

This chapter will show how we have employed the previously described rotation-equivariant operations to estimate LRFs. It will illustrate the architecture of the neural network, the process that transforms input data (point clouds) to output data (LRFs), and the methodologies that have been used to train and test the network. In the course of this chapter there will be some numeric values: they refer to figure 4.1 and represent the best hyperparameters we have found for the network. For the purpose of this chapter they are to be taken as examples.

As for project technicalities, the programming language we used is Python (v3.6.8), with PyTorch (v1.0.0) as the framework for neural networks; Spherical CNNs are built using the original implementation of [6], that can be found on GitHub[1]. Training and testing Spherical CNNs is a very heavy and time-consuming task, so it has been partially sped up by employing GPUs and CUDA. All the trainings and tests have been executed on an Intel i7-4790K with a Gigabyte GeForce GTX 970 G1, or on an Intel i7-8700 with an NVIDIA Titan V (Volta) and an NVIDIA GeForce RTX 2080 Ti.

4.1 Network architecture

The neural network we used is almost entirely made of convolutional layers, in order to achieve equivariance to rotations. In figure 4.1 it's possible to see a graphical scheme of the network: the input is a spherical signal, the first layer employs the spherical correlation (definition 3.12) and all the subsequent layers employ the SO(3) correlation (definition 3.14). While not present in the figure for motivations related to clearance, each convolutional layer, except from the last one, is followed by a Batch Normalization layer and by a ReLU that acts as the non-linearity. The last SO(3) layer is followed by batch normalization only, and

[1]https://github.com/jonas-koehler/s2cnn.

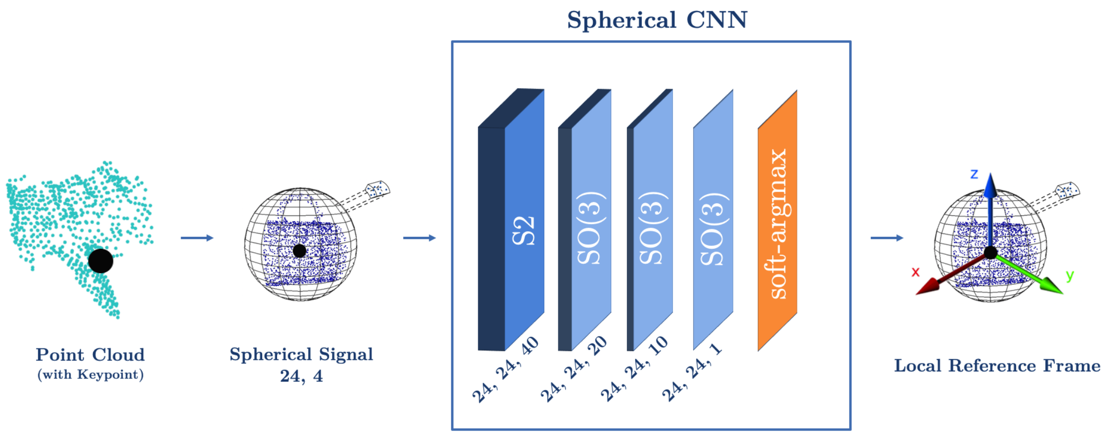


Figure 4.1: Graphical representation of the network. The numbers under the spherical signal represent the bandwidth and the number of channels. The numbers under the correlation layers represent the input bandwidth, the output bandwidth and the output channels. The last layer is a custom version of the soft-argmax function: notice that the output of the last SO(3) layer is a 3D feature map. The spherical signal representation is from [35].

by a soft-argmax layer that selects the output value. Notice that, since our main goal is to achieve equivariance, there are no pooling layers of any kind. The only small hindrance to the perfect mathematical equivariance, besides quantization, is represented by the ReLUs.

As one may notice, the input of the network is a spherical signal, and not a point cloud, because Spherical CNNs work on spherical signals. The generation of a spherical signal starting from a point cloud will be explained in the next section, so here we will simply address the network itself.
The triplets under each convolutional layer are, respectively, the *bandwidth* of the input to the layer, the bandwidth of its output, and the *number of channels* of the layer.
The number of channels has a direct correspondance to standard CNNs, because it represents the number of filters for that layer. For example, the output of the first layer is an SO(3) signal with 40 channels, where 40 corresponds to the K variable in definition 3.14. This means that each filter of the second layer is an SO(3) signal with 40 channels (as many as its input), and there are 20 of them, and so on for the other layers.

The bandwidth, instead, is a less trivial concept because it is involved in the Generalized Fast Fourier Transform that happens in the layers. Practically speaking, it translates to the resolution of the signal: an output bandwidth of 24 means that the output of each layer is an SO(3) signal with a 48x48x48 discrete domain[2]. The exact same concepts apply to the spherical signal as well: the first number, 24, is a bandwidth, and it means that the signal is defined on a 48x48 domain[3]; the second number means that it has 4 channels.

The soft-argmax layer does not implement the standard soft-argmax function: it uses a custom operation and it will be explained in the next section. Its output is a point location in the 48x48x48 domain of the output of the last SO(3) layer: the location of such point is identified by 3 SO(3) coordinates that correspond to ZYZ-Euler angles. Such angles unambiguously determine a rotation, and thus a Local Reference Frame.

Note. Number of layers, bandwidths and channels are hyperparameters of the network and have been tested in different combinations. The values indicated here represent the best-performing network, but the next chapter will contain results from networks with different settings as well.

4.2 Training data flow (forward)

This section illustrates the process that transforms a point cloud into a list of keypoints with their support, and each of them into a Local Reference Frame, thus showing the entire forward step of the pipeline. The operations from "cloud subsampling" to "spherical voxelization" are carried out by a dataloader, and are used for the training process. The benchmark follows a very similar procedure, but without data augmentation, as will be explained in section 4.4.

Cloud subsampling. A point cloud may contain a huge number of points in the 3D space. In order to speed up the computation, a subsampling may be performed by using the `voxel_down_sample()` function of the Open3D library, with the `leaf_subsampling` parameter provided by the user (the higher the leaf the lower the number of extracted points). Figures 4.2 and 4.3 show how subsampling modifies an example input cloud.

[2]The SO(3) subgroup is 3-dimensional.

[3]The domain is S^2, a 2-dimensional manifold in $\mathbb{R}^3$.

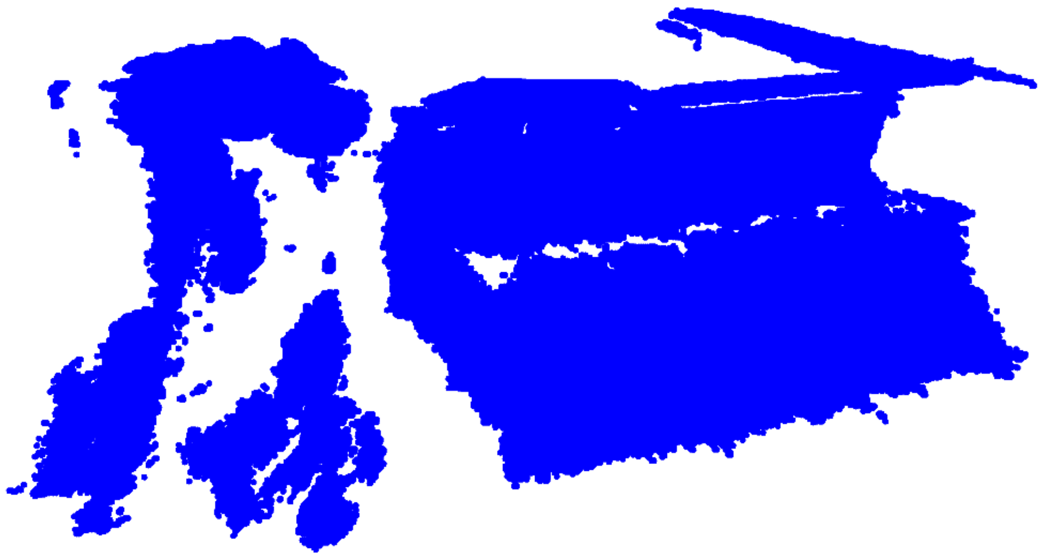


Figure 4.2: Example point cloud right after loading. (Cloud 0, rgbd-scenes-v2-scene_02 [17], 3DMatch dataset [36])

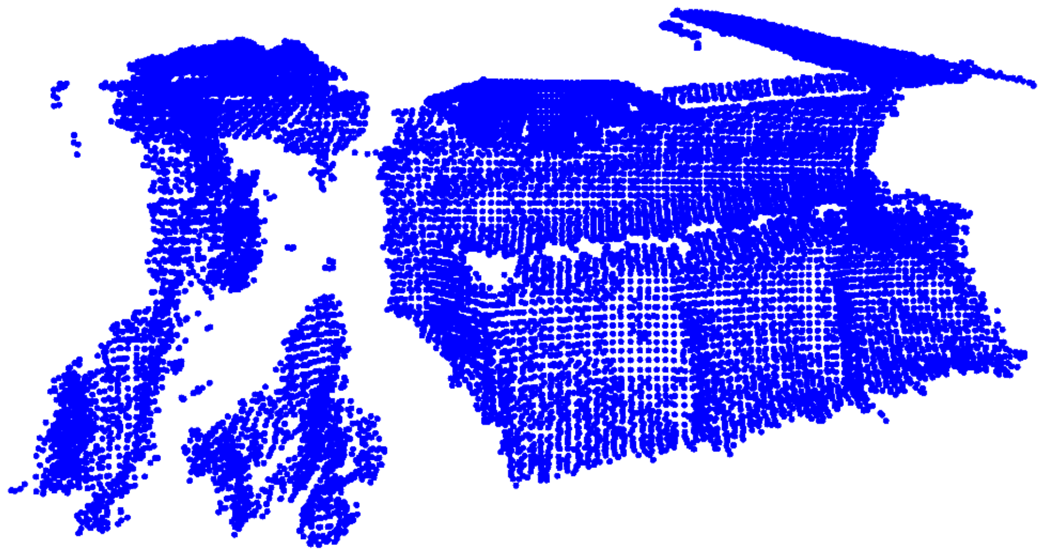


Figure 4.3: Same cloud as figure 4.2, after subsampling. Subsampling has been exaggerated for illustration purposes.

Keypoints selection. To select the keypoints, we uniformly sample the point cloud. This procedure ignores the underlying structure of the point cloud, and is employed because selecting *interesting* points requires an effective and general purpose 3D keypoint detector, which has not emerged yet. Indeed, this is the standard way to define keypoints when computing local descriptors. The procedure is implemented by using the same `voxel_down_sample()` function as above, with the user-specified `leaf_keypoints` parameter: the lower the parameter and the higher the number of extracted keypoints. Figure 4.4 shows keypoints in a point cloud, sampled with a high leaf value.

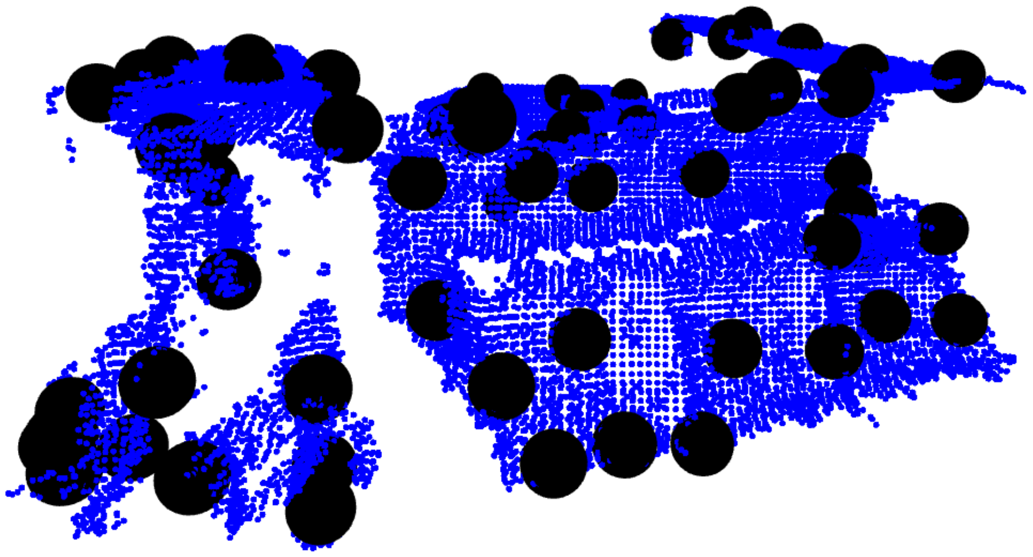


Figure 4.4: Same point cloud. Keypoints highlighted with black spheres enclosing their support. During actual execution the number of keypoints is much higher and their support can be larger.

Local support extraction. For each keypoint, it is necessary to create a local point cloud that comprehends the points in the neighborhood of the keypoint itself. A `radius` is chosen by the user and a `KDTree` (from Open3D) is used to select the points within the radius. All such points are shifted in order to refer their coordinates to the position of the keypoint and achieve translation invariance. As an example, figure 4.5 shows the support of a keypoint.

Data augmentation. The local support undergoes a simple augmentation process, that consists in the application of a random 3D rotation. Since this process

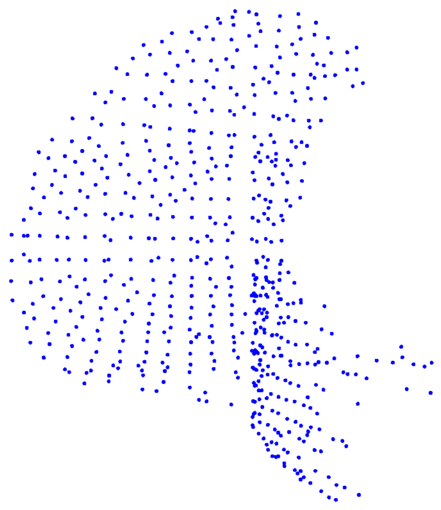


Figure 4.5: Local support of a keypoint from the cloud of previous figures.

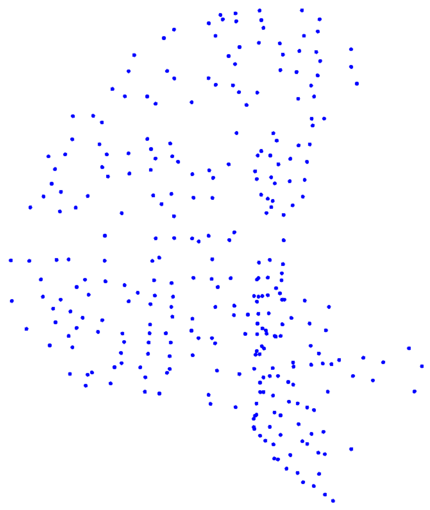


Figure 4.6: Local support of the same keypoint as figure 4.5, after a random sampling. The sampling has been exaggerated for illustration purposes.

is repeated each time a keypoint is extracted, the network potentially never sees the same support twice, because it will always be rotated in a different way. This is useful to increase the diversity of input data and to artificially generate different views of an object, as the real-world datasets often include a small number of different viewpoints.

Random sampling. This operation is optional. It is another form of data augmentation and consists in sampling with replacement a user-specified amount of points from the support, discarding the rest. It can be useful to enhance the robustness of the network to clouds with lower resolutions and to acquisition noise. Figure 4.5 and 4.6 show the support of a keypoint before and after random sampling.

Spherical voxelization [35]. This is the most important operation, since it allows to create a spherical signal from the support points. One of the simplest methods to create a spherical signal is *ray casting* (see [6], figure 5): it consists in surrounding the support with a sphere, and projecting rays towards the enclosed object. As these rays touch the surface of the object, they may be used to extract multiple features such as distance and cosine, and thus define a signal with a spherical domain. This process, though, can only extract information about the surface of the object, and not about the rest of its structure; moreover, vanilla point clouds do not have any information about surfaces, they are simple sets of points, not meshes, and thus ray casting is not suitable for them. Spherical voxelization, instead, can address both problems and has been proposed by *You et al.* in [35]. It can be seen in figure 4.7.

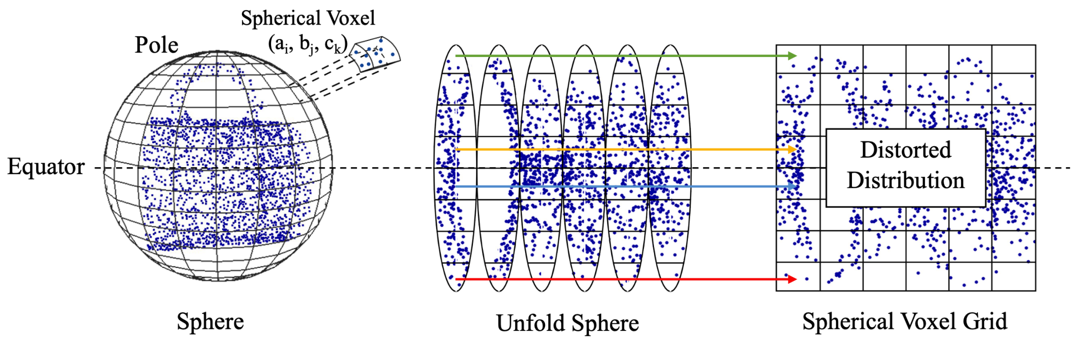


Figure 4.7: Graphical representation of the spherical voxelization operation, figure from [35]. On the left, a keypoint and its support are enclosed in a sphere, which gets discretized and unfolded to a rectangular (or cubic) grid.

Taken a keypoint with its support, suppose to center a sphere on the keypoint, with radius equal to the support radius, and suppose to express the point coordinates in spherical coordinates ($\alpha \in [0, 2\pi]$ azimuth, $\beta \in [0, \pi]$ elevation and d distance from the origin, which is the keypoint). The spherical surface is parametrized with the analogous spherical coordinates a and b. Since we are using computers, everything has to be discrete, so the whole sphere is divided into small curved rectangles depending on the bandwidth. For example, the bandwidth in figure 4.1 is 24 and so there are 48x48 such regions[4]. If we project lines from the keypoint to the boundaries of such regions (up to the support radius), the space itself gets divided into 48x48 3D regions called *spherical voxels*: each point is labeled to be part of the corresponding voxel, and each voxel is identified with the spherical coordinates a, b and c of its *center*.
Now, in each spherical voxel we compute the density of the contained points, where each point is weighted by a function of its *radial* distance from the center of the voxel, $|d - c|$: the radially closer to the center, the higher the contribution. To visualize this, we can imagine to keep the spherical-voxel discretized structure of

[4]Both the domains of a and b are discretized in $2 \cdot 24$ intervals.

the space and to also draw a sphere with half the radius of the support, where all the centers of the voxels lie. Each voxel now corresponds to the space between the keypoint and the half-radius sphere, plus the space between the half-radius sphere and the full-radius sphere. We compute the density by weighting each point in the voxel with a function of its distance from the half-radius sphere. This whole procedure thus produces a real-valued scalar spherical signal, on the half-radius sphere.
Since we want to take into account the internal structure of the cloud as well, this process is generalized to multiple concentric spheres, parametrized in the same way and discretized with the same bandwidth. To produce a scalar signal we have drawn 2 spheres, the outer sphere and the half-radius sphere; thus, to produce two scalar signals we draw 3 spheres: one with 1/3 of the radius, one with 2/3 of the radius, and the outer one which has full radius. This produces 2 concentric and partially overlapped layers of voxels: the first layer is made of the volume between the keypoint and the first sphere, plus the volume between the first and the second sphere; the second layer is between the first and the second sphere, plus between the second and the outer. We proceed to compute the distance-based density on each voxel, producing 2 signals: one for the inner layer, one for the outer. The same process can used with more layers, producing k different spherical signals, or a single k-valued spherical signal, as it will be considered for correlations. In figure 4.1 we have 4 input channels, so we produce 4-valued signals by drawing 5 concentric and equally-spaced spheres.
From a practical point of view, the signal is represented by a 3D PyTorch Tensor, whose dimensions are 48x48x4. In figure 4.7, instead, we have the single-channel case, that is represented by a planar grid.

As represented in the central part of figure 4.7, though, the spherical regions have very different areas depending on their elevation coordinate. In order to produce a more even spacing, a so-called *density aware factor* has been introduced in [35]. It is $\eta = sin(b)$, and its inverse is used in order to produce uniform regions across the sphere: it is 1 at the equator and smaller towards the poles[5]. With this technique, the size of regions closer to the poles increases, but the size of equatorial regions does not decrease, making the total area of the regions bigger than the sphere itself. As a consequence, spherical voxels will have overlapping volumes, sharing part of their points with the voxels of neighboring regions.

Spherical CNN. The whole 4-channel signal is fed to the network, which computes the spherical and SO(3) correlations up to the last layer, as described in the

[5]The original formulas can be found in [35] and will not be reported here, because they unnecessarily weigh the explanation down. The code is also available on GitHub at shorturl.at/gqOST.

previous section. The output of the last layer is a signal on SO(3), which is represented by a cubic feature map. According to figure 4.1, such map is 48x48x48 and has a single channel (i.e. scalar values). Each of the 3 dimensions of this map encodes a Euler angle as its index, making each position of such map a fully identified 3D rotation.

Soft-argmax. A rotation can directly correspond to a Local Reference Frame (as seen in section 3.3), so what we need to do in this layer is extract a single rotation out of this output.
A very simple way to do this is by applying the soft-argmax function, that computes probabilities by taking exponentials and normalizing them to add up to 1. The bin with the highest probability can be chosen to be the Local Reference Frame.
Experimentally, this approach delivered deluding performances. Better results have been obtained by using a Parzen window, as follows[6]. First, we take the argmax of the last layer. Second, we compute the distances between each position and the position of the argmax, producing a 48x48x48 cubic map of distances. After this, we feed such map to the Parzen window, which is a 1D function of the distance: it reaches a maximum in the argmax and decreases monotonically to 0. Figure 4.8 and the following piecewise equation represent such function:

$$f(x) = \begin{cases} 1 - 6x^2(1-x) & \text{if } x \leq \frac{1}{2} \\ 2(1-x)^3 & \text{if } \frac{1}{2} < x \leq 1 \\ 0 & \text{if } x > 1 \end{cases} \tag{4.1}$$

where x is the distance from the argmax divided by half the width of the cube, thus the function has a non-negative domain. This means that points whose distances from the argmax are higher than 24 steps (out of 48) get 0 as a value.
Thus, by feeding the cubic map of distances to this Parzen function, we obtain another cubic map which gets normalized and multiplied element-wise to the output of the last correlation layer, resulting in a signal peaked on the argmax. Each value of this new map will be considered the weight corresponding to the (α, β, γ) coordinates of the position of such value in the map. For this reason, a weighted sum of the coordinates is carried out, and the resulting triple is the output of this whole soft-argmax operation.

LRF extraction. From the last operation we have a triple of SO(3) coordinates corresponding to ZYZ-Euler angles. By applying equations 3.20 and 3.21, we can

[6]From the implementation at https://github.com/MWPainter/cvpr2019

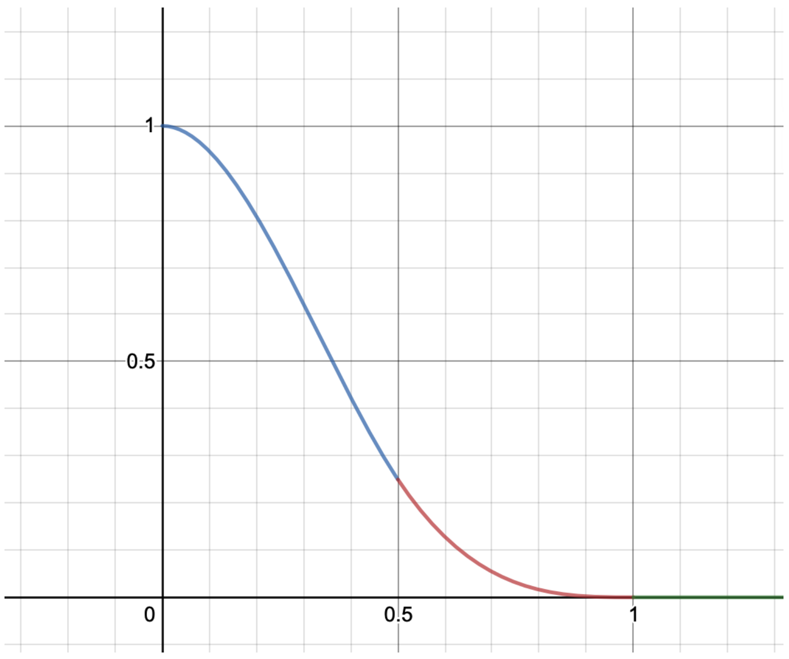


Figure 4.8: 2D graph of the employed piecewise Parzen function. Each interval of equation 4.1 is drawn in a different color. The x-axis represents the ratio between the distance from the argmax and half of the cube width. This means that points farther away than half of such width receive a Parzen value of 0.

compute the rotation matrix that corresponds to the LRF:

$$\mathbf{LRF}(\alpha, \beta, \gamma) = \mathbf{R}_z(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma) \tag{4.2}$$

Please note that, in the actual code, every matrix of the equation above has been transposed (and the multiplication order has been reversed), in order to have the LRF as the rotation matrix that transforms the points into LRF coordinates, as explained in section 3.3.2.

4.3 Training process (backward)

The training process is carried out differently depending on the learning technique. We employed: unsupervised learning, full-fledged adaptation and weakly supervised learning. It's important to notice that there is no way to have labeled data to train our network: having labeled data, in this context, means having a ground truth LRF for each point of the cloud, but there is no absolute truth about LRFs[7]. The only strictly defined property of LRFs is that they have to rotate coherently with the cloud, and thus they only make sense when compared to each other, in a relative way. For this reason, all the following learning techniques will make use of a relative loss between 2 rotated versions of the same keypoint, effectively using a Siamese setting (i.e. with 2 identical networks).

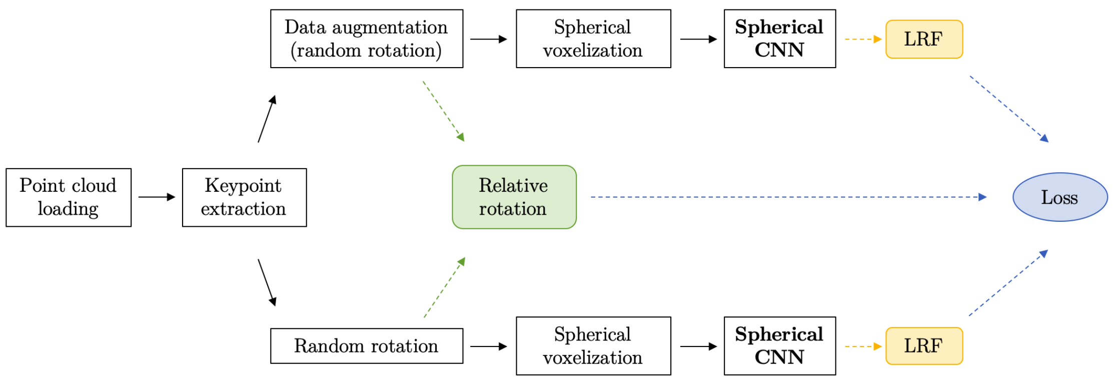


Figure 4.9: Unsupervised learning flow chart. Subsampling and random sampling are not represented to simplify the figure.

Unsupervised learning. Learning without supervision means that we do not use *any* kind of information about data: we only take raw point clouds and compute their LRFs. In order to have a meaningful setting, the unsupervised process proceeds as in figure 4.9: we take a keypoint from a pointcloud, extract its support, and then duplicate it. One copy gets augmented by a random rotation, while the other one undergoes a different random rotation. The resulting rotation matrix that separates such two copies is computed and saved, becoming our synthetic ground truth.
The network is fed both keypoints and it computes their respective LRFs. Then, the LRF of the second copy gets rotated back using the previously saved ground truth matrix (which is the relative rotation between the first and the second copy of the keypoint), and thus the two outputs can be compared with a loss function.

[7]It's possible to have labeled data by computing a hand-crafted LRF and using the network to learn such function, but this has two negative implications: first, it would be constrained to the performances of the hand-crafted LRF; second, it would not learn something new.

Full-fledged adaptation. This technique is unsupervised and proceeds as follows: first, unsupervised training of a network on a certain training set; then, this network is trained with the *test set itself* for a certain number of iterations, in order for the network to adapt to the new data before being tested.
This may seem as a very clear case of data spoofing but, actually, since none of these data has any label or ground truth, there is no spoofing involved. Moreover, it can be legitimized by a simple consideration: is it possible to use this technique when using the network with real data? The answer is yes, because any surface matching or pose estimation program that uses this network can train it first with some of the data loaded by the user, and then ask it to compute the LRFs.
A very extreme version of this technique consists in fully training form scratch a new network with test data, and it has also been tried in this work.

Full-fledged adaptation is expected to achieve slightly better results than the normal unsupervised learning.

Weakly supervised learning. The last learning technique we used is a form of supervised learning. As already said, not having labels means that we cannot perform a normal supervised learning. Nevertheless, we can use the ground truth[8] information contained in datasets to have realistic input data.
The weakly supervised process proceeds as follows (see figure 4.10). We take two views of the same scene, which are contained in the dataset and for which we know the ground truth rotation matrix that transforms one view into the other. We select keypoints in one of them and then we temporarily rotate the cloud of keypoints to check if the same points exist in the other view[9], pairing them, and we discard all the keypoints that do not have a correspondent. Now, for each pair of keypoints, we augment one of the two with an additional random rotation and then we feed them to the network, computing two LRFs. We know the ground truth and the random rotation applied, so we can rotate back the outputs and compute the loss.

By using real-world rotations instead of the synthetic ones, we expect this method to produce better test results. In fact, despite not being able to use the full-fledged adaptation in this setting, the usage of real rotations can make the network more robust to real nuisances such as: sensitivity of the depth camera, occlusion, clutter, noise. In fact, the same object or scene from two distinct viewpoints can appear *very* differently, and this is certainly good for the network to learn. Of

[8]The ground truth usually consists in the rotation matrices that separate the camera orientations that have captured the different point clouds in the dataset. This means that with the ground truth we can align different views of the same object or scene, and use such views to have real rotations instead of the synthetic ones produced in the processes above.

[9]In the actual implementation we check if it exists a point closer than a certain, small, threshold.

course, it's not always possible to use supervised learning, because not all the datasets have enough supervised data to train and test the network.

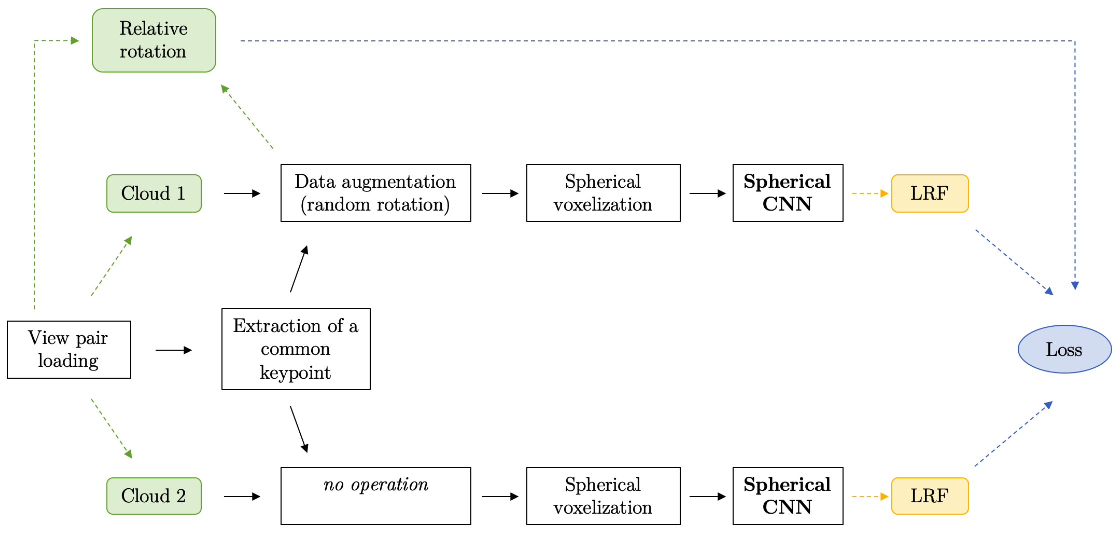


Figure 4.10: Weakly supervised learning flow chart. Subsampling and random sampling are not represented to simplify the figure.

As a closing remark, the actual implementation of all these procedures optimizes execution times and randomizes the input order, so the extraction of all the keypoints from all the clouds/pairs happens all at once at the beginning of the training. When a certain keypoint gets selected for a forward pass, it is loaded with its local support only, without loading the entire point cloud.

4.3.1 Loss function

The last step we are missing for the training process is represented by the loss function. Divergences (such as the Kullback–Leibler or the Jensen–Shannon divergence) or distances (such as the Wasserstein distance from the Optimal Transport problem) between the feature maps of the last layers of corresponding keypoints are very interesting approaches that are yet to be tested. The actual loss used in this network is much simpler, though, and it consists in the 3D angle between two correspondingly computed LRFs. Such LRFs, computed from paired keypoints, are aligned using the ground truth, that can be synthetic (unsupervised) or from real data (weakly supervised). A perfect algorithm would result in a perfectly aligned pair of LRFs, so we compute the angle between them: the lower, the better. Such angle is referred to as *Theta angle*, and is computed by means of the trace of the rotation matrix that separates them, as in section 3.3.3: finding such matrix is easy, since both of the LRFs are actually rotation matrices. This loss function is referred to as *Theta loss*, and minimizing it indirectly causes an improvement in our repeatability measure (section 3.4), because repeatability can be 1 only if the Theta angle is low. The network is mathematically rotation-equivariant but this loss function, along with the soft-argmax operation we have defined, forces the network to learn to robustly produce *equivariant maxima*.

Apart from the loss function itself, computing it with the Parzen window scheme that has been explained before appears to be far from optimal, both in terms of experimental performances and in terms of theoretical significance, even though it is still better than a vanilla soft-argmax operation. From the latter point of view, using this window has a two-fold effect: first, we only let the network accumulate gradients to shift the position of the maximum identified with the window, but we do not encourage the network to produce similar feature maps for corresponding pairs; second, the window itself does not accumulate any gradient, because its values are computed using an argmax, which is a non-differentiable operation. Moreover, another good addition would be to force the output to be clearly peaked in a small area, because the usage of maxima without forcing a peaked output structure produces very non-robust results, as found in [26]: if two or more high values exist, it is likely that a rotation of the input causes them to also shift their values, and thus to change the output of the network. If the output had instead a Gaussian structure, for example, this problem could be solved, as a very close bin would get selected. Using as loss a divergence between corresponding feature maps and/or a divergence with a feature map and a Gaussian could partially address these problems, and they could be worth exploring in the future.

4.4 Benchmarking process

After training a network, it is necessary to assess its performances. Testing the network with synthetically rotated data would not produce a realistic indicator of performance. Thus, independently from the training process, the benchmark follows a standard pattern that is also used for the different methods we tested as baselines (FLARE and SHOT), and that is analogous to the weakly supervised approach to training, except for data augmentation.

Point clouds are loaded in pairs, along with the ground truth rotation that separates them. After a subsampling, the first point cloud is uniformly sampled to select the potential keypoints, according to a user-defined `leaf_keypoints` parameter. The first cloud is then temporarily rotated to match the second one,

using the ground truth, and for each potential keypoint we select the closest point to it in the second cloud. If, for a certain candidate, the closest point is closer than a user-defined `min_distance`, such candidate is rotated back and selected as a keypoint, and its closest point is used as the corresponding keypoint in the second cloud.
All such keypoints undergo a forward pass in the network along with their support, the extracted LRFs are rotated using the ground truth and their alignment is evaluated using the LRF repeatability measure, defined in section 3.4.
This process is repeated for all the view pairs of a certain scene, and a repeatability percentage is produced for every scene. Notice that, for some datasets, information about the percentage of overlapping regions in each pair is also available: for such datasets, only the pairs with an overlap of at least `overlap_threshold` are selected for the benchmark.

Chapter 5

Experimental results

This chapter presents the most interesting results obtained. They will be provided with visual charts and numeric tables, and will be followed by comments and by an analysis of the motivations that have brought to such results. When useful, some examples of clouds or keypoints will be shown, along with a representation in Rodrigues' vectors of the output feature map of the network.

In the following charts and tables, many different parameters will be changed and tuned to achieve better performances. Unfortunately, despite the use of CUDA-capable GPUs and a constant optimization of the code, trainings and tests require a significant amount of time: a single-epoch training with a bandwidth of 24 can take 10 hours, and a training can need 5-10 epochs to be completed, or more, depending on the number of samples in each epoch[1]; a full benchmark can take from 8 to 20 hours, because it selects more keypoints than the training to have more precise results, and it is bound to using real view pairs, independently from the split we're benchmarking on (train, validation, test). For these reasons, doing a full grid search to look for the best combination of hyperparameters is impossible, at least in the time span of a thesis. In order to converge anyway to a good combination, a substantial amount of trainings and tests have been done heuristically, and obtaining partial results only. This means that these results will not be shown here, but they were useful to select the direction of the search. Moreover, the whole set of produced results is too big to be discussed completely, so only the most interesting and complete results will be shown in the next sections.

Another peculiar fact is that, since benchmarking times are so high, it is im-

[1]The dimension of the dataset defines the number of meaningful keypoints we can extract. For a dataset we were able to extract 25K points or more, for another just a tenth of it: if the first needs 8 epochs to train, the second needs around 80. To have acceptable results, we've found that its the total amount of inputs that are fed to the network that has to stay constant, so the product between keypoints per epoch and the number of epochs. For this reason, independently from the number of keypoints, training times are approximately constant.

possible to meaningfully use real view pairs to select the best training checkpoint with a benchmark on the validation set. Thus, for unsupervised (and full-fledged) trainings, the validation performances are evaluated exactly as during training, so with synthetic rotations, and the checkpoint of the network is selected from this evaluation. The only exception to this impossibility is the supervised training, in which, instead, we employ real rotations to check the validation performance, so we actually benchmark the validation set at each checkpoint. It's worth noticing, though, that it has been possible to apply supervised training only to extremely reduced training and validation sets, due to otherwise unfeasible training and benchmarking times.

In terms of fixed hyperparameters for almost all the shown results, we have: learning rate equal to 0.001; batch size equal to 8 [2]; number of SO(3) layers equal to 3; number of input channels equal to 4; equal bandwidth for all the layers and the input; channels of the layers set as in figure 4.1. If a parameter is set differently, it will be explicitly stated.
For benchmarking, the threshold used in the repeatability measure of section 3.4 is 0.97, meaning that for two LRFs to be repeatable, both their x and z axes should produce a cosine higher than 0.97, and thus that x and z axes can be misaligned each by a maximum of $\arccos(0.97) \approx 14$ deg, or around 0.246 radians.

In order to have a comparison with the state of the art, the results of the network will be compared with the scores of FLARE [21], computed using the exact same benchmark as for the network. A side note about FLARE is that, in order to work, it needs the normal of each point. Thus, it works seamlessly with meshes, but it can also work with point clouds if provided with such normals. Since our network works with point clouds only, the scores of FLARE are shown both at its peak performance (with meshes, when available) and at the same conditions as the network (so with point clouds).

All the following results and charts will be available in this repository, as well as all the partial and less interesting results that are not shown in this chapter:
https://github.com/CVLAB-Unibo/thesis-stella

5.1 3DMatch

The dataset we refer to as "3DMatch" is actually a collection of 5 different real-world datasets, capturing indoor scenes, that have been converted to the same format. This collection has been produced by *Zeng et al.*, from Princeton University, for their work on learning descriptors from 3D data [36]. Their model

[2]Smaller sizes lead to worse results, higher sizes do not improve results significantly. By chance, 8 is usually the maximum parallelism allowed by the VRAM of the used GPUs.

is called 3DMatch, from which the name of the dataset that can be found at http://3dmatch.cs.princeton.edu, that comprises:

- SUN3D [32]: http://sun3d.cs.princeton.edu
- 7-Scenes [25]: https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes
- RGB-D Scenes v2 [17]: http://rgbd-dataset.cs.washington.edu/dataset/rgbd-scenes-v2
- BundleFusion [8]: http://graphics.stanford.edu/projects/bundlefusion
- Analysis by Synthesis [30]: http://graphics.stanford.edu/projects/reloc

The total number of different scenes is 62, and each scene is made of a variable number of views (point clouds), from 6 to almost 300. The test split is the same as in [36] and [26], while the rest has been divided into training and validation. Here are the validation (6 scenes) and test (8 scenes) splits, whereas the training split comprises the remaining 48 scenes. Validation:

- analysis-by-synthesis-apt2-kitchen,
- sun3d-harvard_c11-hv_c11_2,
- rgbd-scenes-v2-scene_10,
- 7-scenes-heads,
- sun3d-brown_bm_4-brown_bm_4,
- bundlefusion-office0.

Test, in brackets the short name they are referred to in this work and in [26]:

- 7-scenes-redkitchen (Kitchen),
- sun3d-home_at-home_at_scan1_2013_jan_1 (Home1),
- sun3d-home_md-home_md_scan9_2012_sep_30 (Home2),
- sun3d-hotel_uc-scan3 (Hotel1),
- sun3d-hotel_umd-maryland_hotel1 (Hotel2),
- sun3d-hotel_umd-maryland_hotel3 (Hotel3),

- sun3d-mit_76_studyroom-76-1studyroom2 (Study),
- sun3d-mit_lab_hj-lab_hj_tea_nov_2_2012_scan1_erika (Lab).

Additionally, we defined two very small training and validation sets, mainly used to get quicker results, to check the generalization of the network, or to train it with the supervised method, that takes much longer and is impossible to run on the full training set. Such sets are referred to as "Train2", containing sun3d-brown_cogsci_1-brown_cogsci_1 and rgbd-scenes-v2-scene_02, and "Validation1", containing only analysis-by-synthesis-apt1-kitchen. The others will be referred to as "TrainFull" and "ValidationFull", or simply by full training set and validation set. Please note that the validation scene sun3d-harvard_c11-hv_c11_2 could not be used for benchmarking because it lacks the ground truth, so it has been used during synthetic validation only.
An example of a point cloud from this dataset is shown in figures 4.2 and 4.3.

For all the following evaluations we kept the support radius constant at 30 cm, a value arisen from short tests and previous works such as [26].

5.1.1 Training vs Validation vs Benchmark

Before proceeding with the chapter, here is a comparison of the performances of the network during a typical unsupervised training and the performances of the same network during the benchmark, which makes use of real rotations. In figure 5.1 and 5.2, we can see that training and validation performances increase very quickly at the beginning, and they continue to increase more slowly in later stages. The network these figures refer to is trained with the full training split for 5 epochs, and uses a bandwidth of 24. The abscissa in the figures indicates the checkpoint number, where each checkpoint corresponds to 1000 forward passes, and each epoch contains around 25K keypoints: at each checkpoint, a full evaluation of the validation set is carried out, whereas the training performances are taken as the epoch-average up to that point.

Another thing we can see in these plots is that the validation performance is *better* than the training performance, at least up to a certain point in which they become approximately identical, whereas in the literature the validation performance is usually lower than the training one. This can be explained by the fact that training performances are epoch-averages, whereas the validation set, being smaller, is fully evaluated at each keypoint. For early checkpoints, in fact, the learning curve is still quite steep and the average of the training performances produces values that are slightly biased towards the worse, underestimating the real performances on the training set. This bias is greatly reduced in later checkpoints, because the learning curve becomes flatter, and in these points the validation per-

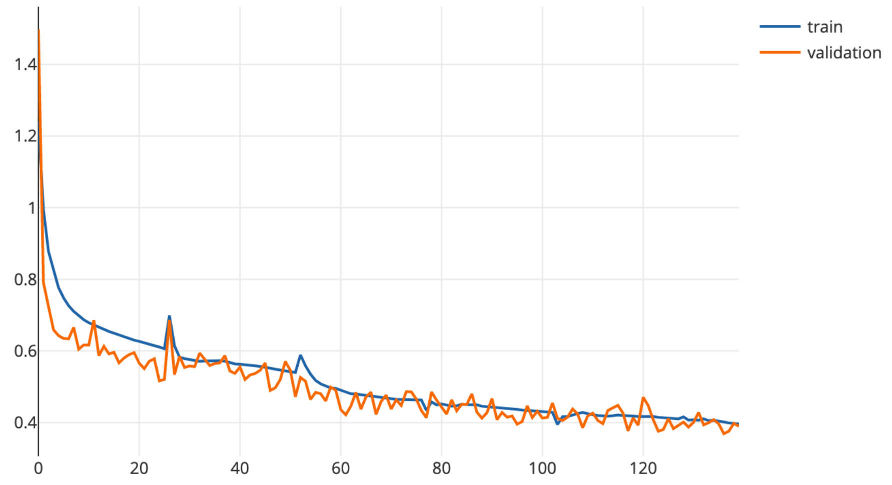


Figure 5.1: Theta loss during a typical training. A unit on the x axis represents 1000 forward-backward passes.

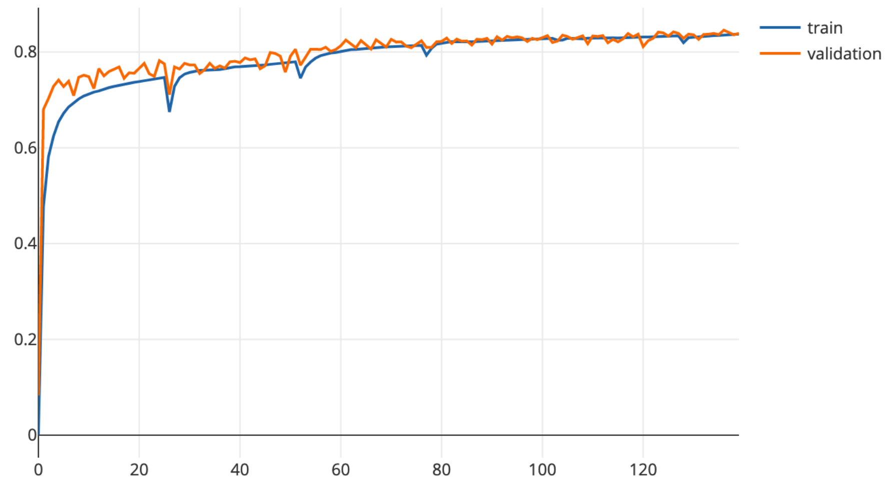


Figure 5.2: Repeatability (%) during a typical training. A unit on the x axis represents 1000 forward-backward passes.

formances are aligned to the training ones. Moreover, there is *no sign of overfitting* involved.

Performances in the benchmark, though, independently from the set we use, are significantly worse: whereas training and validation repeatabilities span from 80% to 90%, the repeatability in the benchmark rarely goes above 40%, even though it can touch 60% on certain scenes. This happens because the benchmark uses real view pairs instead of synthetically rotated keypoints: the latter method produces the same support, but rotated, whereas the former can produce quite different supports because of noise, clutter and occlusions. Changing the viewpoint means that objects can partially occlude other objects in the scene, or that a side of an object that was in front of the camera becomes captured sideways, and thus with a much lower point density. An example of how much clouds can change with a viewpoint variation can be seen in figures 5.3 and 5.4: they show two overlapped views of the same scene in different colors, taken from rgbd-scenes-v2-scene_02 and rotated with the ground truth in order to be aligned. In the scene we can see a sofa in front of a table with some objects, inside a room. Apart from the outer objects that appear in one view and not in the other, the sofa and the table appear quite differently in the two views. Moreover, the objects on the table cast two different "shadows" on it, that can be seen by looking at the missing red and green regions on the table. There are a lot of border keypoints that are correctly found in both clouds but show very different supports: it's mainly because of these differences between viewpoints that benchmark performances are so low when compared to training and validation. In fact, during supervised training, these differences do not arise, and the training and validation performances, while being lower, are good indicators of the benchmark score.

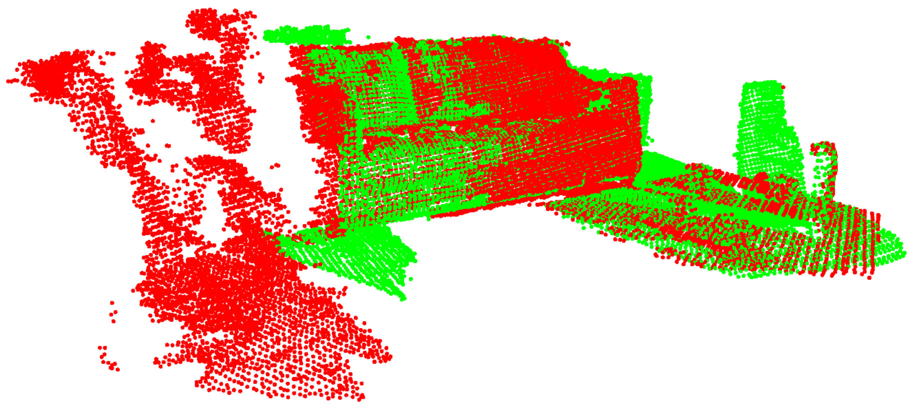


Figure 5.3: Overlap of 2 different views of the same scene, one in green and the other in red.

Moreover, due to the mathematical equivariance of the network, one may expect training performances to almost touch 100%. But, as explained in section 4.1, the presence of non-linearities lowers the equivariance of the network and, in addition, the random sampling makes the pairs of training keypoints non-identical. In figure 5.5 we can see that training without random sampling boosts the equivariance, and the absence of subsampling boosts it even more. However, it is worth highlighting that those operations have been introduced for robustness, and that these performances refer to unsupervised training and validation with synthetic rotations, and do not have any implication on the benchmark performances. The impact of the two sampling operations will be analyzed in the following sections.

All the considerations made in this section apply to the other datasets as well.

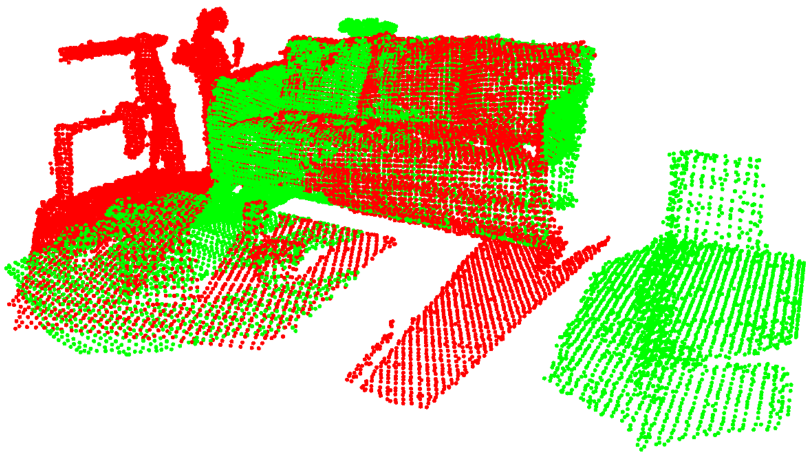


Figure 5.4: Overlap of 2 different views of the same scene, one in green and the other in red. Virtually rotated.

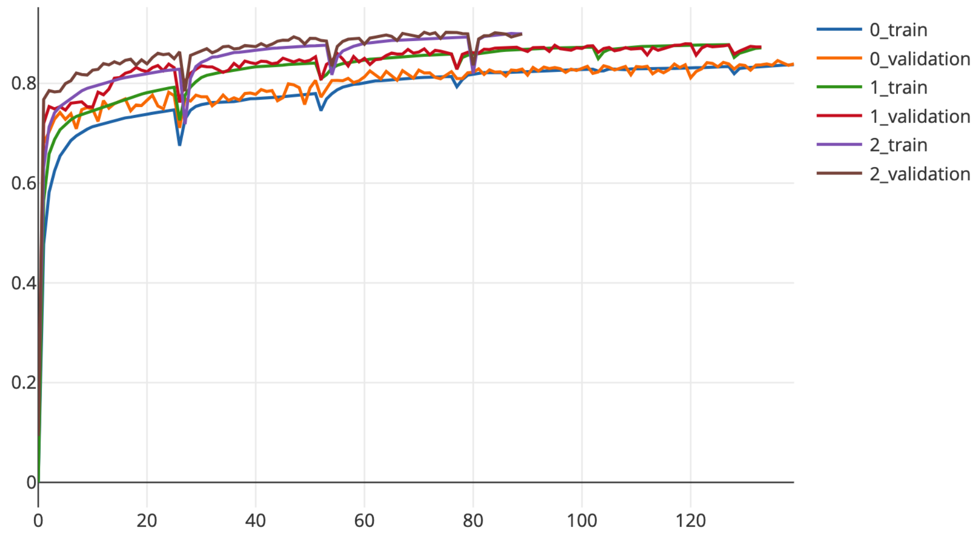


Figure 5.5: Repeatability (%) during typical trainings. A unit on the x axis represents 1000 forward-backward passes. *0* is a normal training, *1* is without random sampling, *2* is without both random sampling and subsampling.

5.1.2 Bandwidth selection

The first thing we tried to optimize is the bandwidth of the network. Assuming that the bandwidths are equal in all the layers, duplicating the bandwidth means duplicating each side of 3-dimensional feature maps, and thus it means multiplying by around $2^3 = 8$ the training time. We fully tested bandwidths up to 24, and only partially tested 32 due to excessive time requirements. Figure 5.6 and table 5.6 show the benchmark repeatabilities on the full validation set. All the networks have been trained with the small Train2 dataset to have quicker results.

Table 5.1: Benchmark repeatability on the validation set for networks with different bandwidths. Best result on each column in bold.

	Apt-Kitchen	Scene10	Heads	Brown_bm4	Office0	Mean
Bandwidth 3	0.05616	0.04186	0.03403	0.1274	0.05089	0.06206
Bandwidth 6	0.2482	0.1818	0.1118	0.3065	0.2150	0.2127
Bandwidth 12	0.4207	0.2948	0.2163	0.3836	0.3271	0.3285
Bandwidth 24	**0.5075**	**0.4108**	**0.3402**	**0.4167**	**0.4234**	**0.4197**

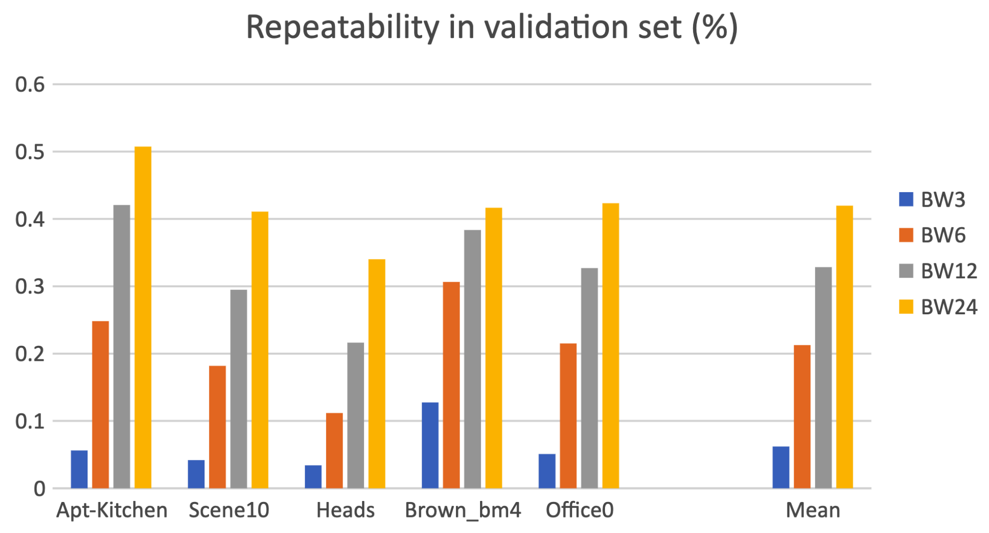


Figure 5.6: Benchmark repeatability chart for networks with different bandwidths (validation set).

As already stated, bandwidth 32 is not included in the chart because it has been trained and tested only up to 2 epochs, producing similar results to bandwidth 24, that is clearly the winner in this comparison. For these reasons, 24 has been selected as the bandwidth for all the subsequent tests, and will be considered a default parameter from now on, event though we suggest further testing with higher bandwidths.

5.1.3 Generalization and robustness

Train2 vs TrainFull. Having selected the bandwidth, the first thing we want to assess is the difference between a network trained on Train2 and a network trained on the full training set. Figure 5.7 and table 5.2 show the benchmark results on the validation set.

Table 5.2: Benchmark repeatability on the validation set for networks trained with different training sets. Best result on each column in bold.

	Apt-Kitchen	Scene10	Heads	Brown_bm4	Office0	Mean
Train2	0.5075	**0.4108**	**0.3402**	0.4167	**0.4234**	0.4197
TrainFull	**0.5358**	0.4049	0.3350	**0.4180**	0.4228	**0.4233**

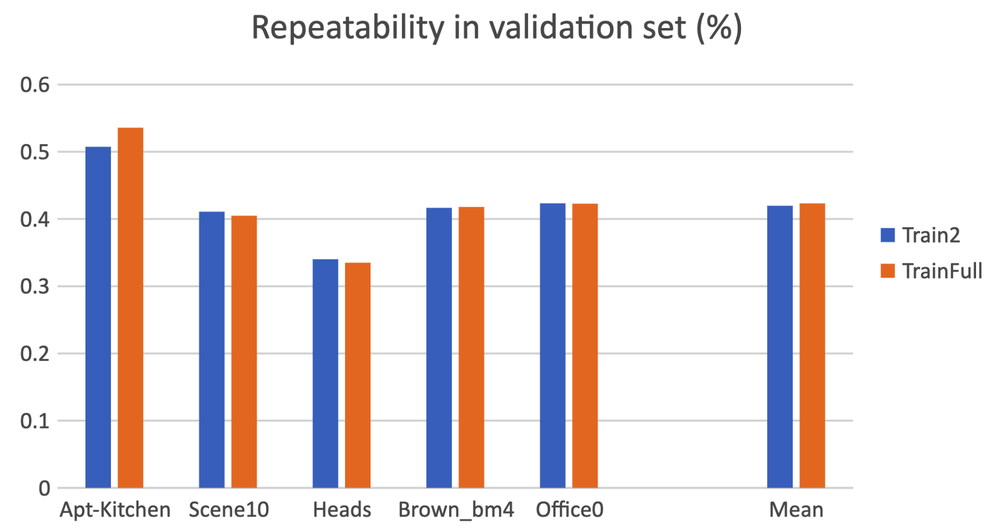


Figure 5.7: Benchmark repeatability chart on the validation set for networks trained with different training sets.

The two networks are extremely similar, with the full training having a very small overall advantage. This means that the networks don't need a lot of data to attain such performances, and a training set that contains two scenes only can be enough to generalize to many unseen scenes. As a side note for this data, though, the network trained on Train2 has been trained for a little longer than the other one, and it samples much more keypoints per point cloud in order to have a similar pool of keypoints to train with. This can also explain why it behaves better on some scenes. Thus, due to the little overall lead of TrainFull despite the training-time advantage for Train2, we decided to use the full training set for the next tests.

Rotating the test set. Speaking of generalization, another important aspect to take into account is how much the network can tolerate rotations, since this is the initial goal of this work. Thus, we benchmarked the TrainFull network of table 5.2 both on the test set and on a randomly rotated version of it, in which every view has been rotated with a different matrix, in order to see the rotation-equivariance of the network.

Table 5.3: Benchmark repeatability on the test set and on a randomly rotated version of it.

	Kitchen	Home1	Home2	Hotel1	Hotel2	Hotel3	Study	Lab	Mean
Normal	0.2764	0.3575	0.3325	0.3268	0.3365	0.3986	0.2876	0.3430	0.3324
Rotated	0.2718	0.3513	0.3237	0.3179	0.3295	0.3995	0.2771	0.3309	0.3252

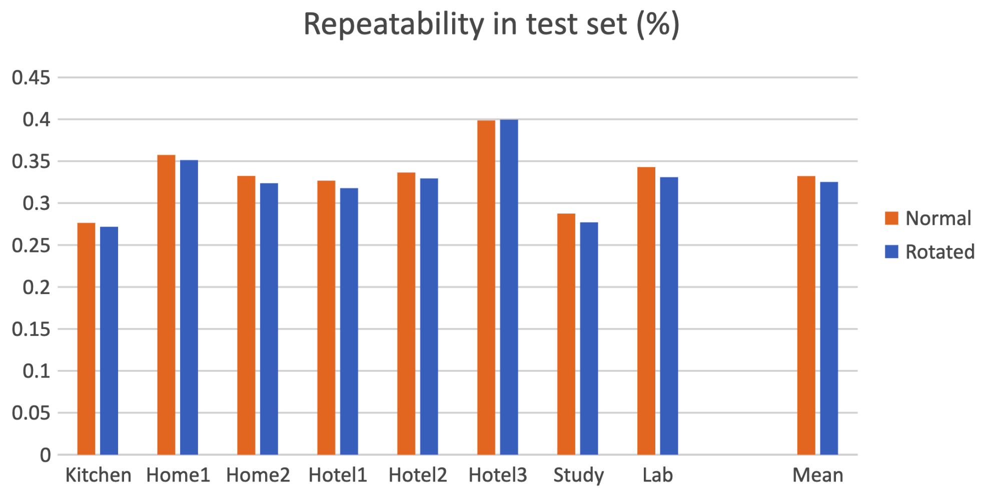


Figure 5.8: Benchmark repeatability chart on the test set and on a randomly rotated version of it.

In figure 5.8 and table 5.3 we can see that the network delivers very similar performances, and that random rotations do not have a meaningful impact: it excellently generalizes to rotated data.

Effect of sampling operations. In the previous chapter, we discussed about the subsampling and random sampling operations, employed to have faster trainings and more robust networks. We also showed in section 5.1.1 that these two operations actually lower training and validation performances, but to really see if they are useful we have to rely to the benchmark. Figure 5.9 and table 5.4 show the performances of a network trained normally (thus with subsampling and random sampling), a network trained without random sampling and a network trained without both random sampling and subsampling, all of them trained without supervision.

Table 5.4: Benchmark repeatability on the test set comparing presence and absence of sampling operations.

	Kitchen	Home1	Home2	Hotel1	Hotel2	Hotel3	Study	Lab	Mean
Normal	0.2764	0.3575	**0.3325**	0.3268	**0.3365**	0.3986	**0.2876**	**0.3430**	0.3324
No-rand	**0.3222**	**0.3578**	0.3307	**0.3743**	0.3353	**0.3993**	0.2770	0.3424	**0.3424**
No-rand&sub	0.2528	0.3451	0.3193	0.3097	0.3206	0.3808	0.2756	0.3314	0.3169

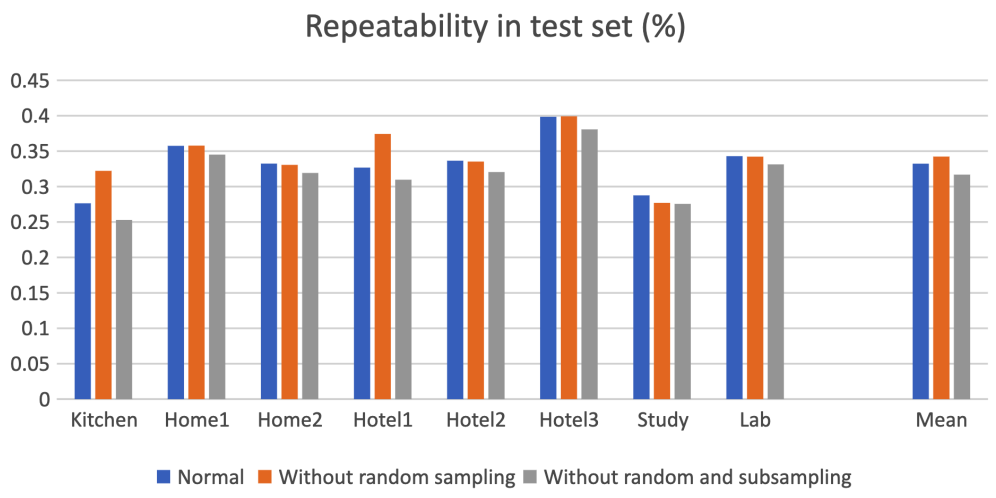


Figure 5.9: Benchmark repeatability chart on the test set comparing presence and absence of sampling operations.

Despite the mean being higher for the network without random sampling, the "normal" network is actually tied with it for the number of scenes in which they achieve the highest scores. Removing subsampling proved to be harmful, instead. Since disabling the sampling operations significantly increases the training time with little to no performance improvement, we will consider the network with bandwidth 24, unsupervised training and both random sampling and subsampling, as the reference network for this dataset and for transfer learning on other datasets as well. Moreover, the absence of sampling operations will not be further tested as it doesn't seem to lead to interesting results.

5.1.4 Overall results

Having set the hyperparameters and checked the robustness of the network, we can present here the best results we have achieved and compare them to some state-of-the-art methods. For these results, we consider 3 networks: the first is the reference one, and followed an unsupervised training on TrainFull, the second one followed a full-fledged adaptation from scratch on the test set, the third one a weakly supervised training on Train2. Their results on the test set are compared to FLARE and SHOT in figure 5.10 and table 5.5. Since 3DMatch doesn't provide meshes, FLARE is executed on the point clouds, with normals computed by means of Open3D.

Table 5.5: Benchmark repeatability on the test set, comparing the network to FLARE and SHOT. Best result on each column in bold.

	Kitchen	Home1	Home2	Hotel1	Hotel2	Hotel3	Study	Lab	Mean
Unsupervised	0.2764	0.3575	0.3325	0.3268	0.3365	0.3986	0.2876	0.3430	0.3324
Full-fledged	0.2841	0.3684	0.3522	0.3364	0.3479	0.3983	0.2988	0.3569	0.3429
Weakly Supervised	0.3152	0.3753	**0.3617**	**0.3800**	**0.3965**	**0.4421**	**0.3429**	**0.3653**	**0.3724**
FLARE	**0.321**	**0.376**	0.325	0.369	0.384	0.387	0.338	0.296	0.3495
SHOT	0.206	0.244	0.227	0.189	0.188	0.228	0.181	0.218	0.2101

As we can see, the full-fledged method does improve over the unsupervised training, but not by a large margin, showing again that the network has great generalization capabilities.
The improvement of the weakly supervised method is not large either, but it is significant and is enough to surpass FLARE. SHOT, instead, is clearly behind all the other methods and will not be included in the next charts.

5.1.5 Result analysis

The numbers clearly show that the performances of the network are very similar to those of FLARE, that represents one of the current state-of-the-art methods, with the supervised network being able to beat it even by using only 2 scenes for

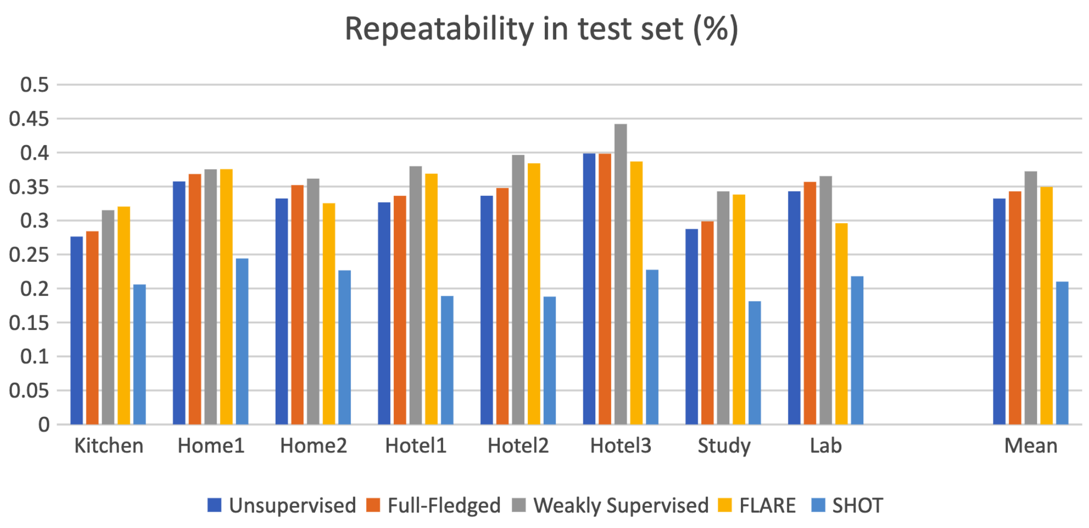


Figure 5.10: Benchmark repeatability chart on the test set comparing the network to FLARE and SHOT.

the training. Despite the results being promising, it is necessary to try to analyze them more in depth, in order to understand the weaknesses and produce better methods for future work.

For all the following figures and considerations, the network used is the reference one, which is the unsupervised row of table 5.5.

Theta histograms. In figure 5.11 we can see a plot of the distribution of the Theta angle in radians between LRFs of *synthetically* rotated pairs of keypoints, for the whole validation set. The great majority of them have angles much lower than 0.5, preluding great validation repeatabilities for synthetically rotated data, as we've already seen. What is interesting about this chart is that the histogram is bimodal, having another peak in the right-end. The whole histogram evolves a lot during training: it starts as an almost flat histogram, then it start to become bimodal, with peaks on the left and on the right, followed by a thick trail. As the training goes on, the peaks become higher and the trail lower, with the left peak eventually overcoming the right peak.
A peak on the right means that there is a certain number of points whose LRFs get misaligned by a full 180 degrees rotation around some axis in the space, which is the maximum possible misalignment. It is still not totally clear why this happens, but particularly symmetrical inputs and imperfect equivariance of the network can be two causes. A further study on this could be needed, for example trying to track down the rotation axis.

The situation gets much worse in the benchmark (figure 5.12a), where the

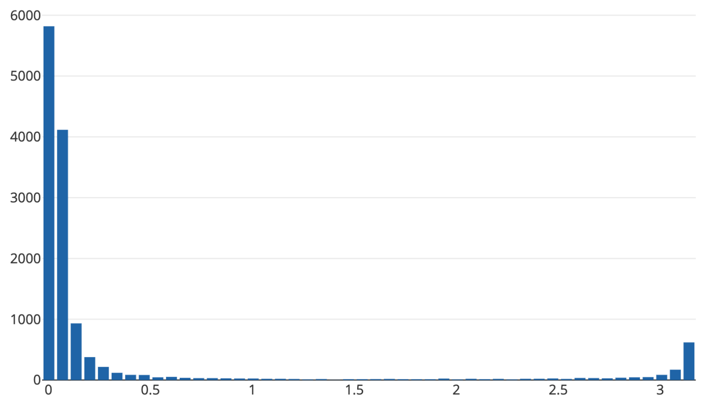
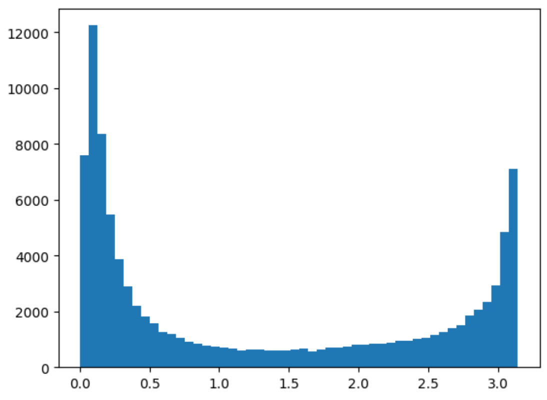
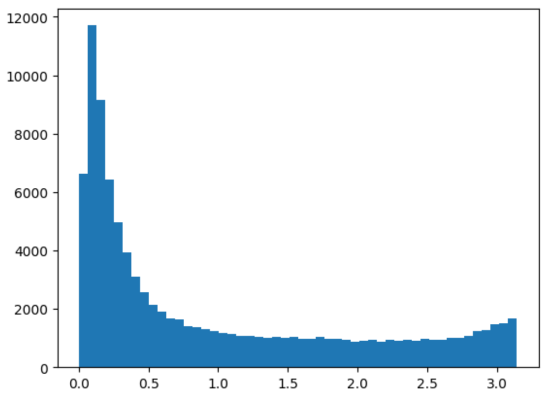


Figure 5.11: Theta histogram of the validation set with synthetic rotations, obtained while training an unsupervised network.



(a) Network histogram.



(b) FLARE histogram.

Figure 5.12: Theta histograms of the test scene Home1 with real-world rotations.

histogram is more clearly bimodal in most of the cases. The figure refers to a test scene, but the situation is totally analogous for training and validation scenes, as long as we use real-world rotations. However, there are particular scenes in which this happens much less (figure 5.13), and they are usually the scenes with the highest repeatability.

An example from the benchmark with FLARE shows that its histogram, while having a similar left peak, shows almost no right peak, at the disadvantage of the trail which is thicker (figure 5.12b).

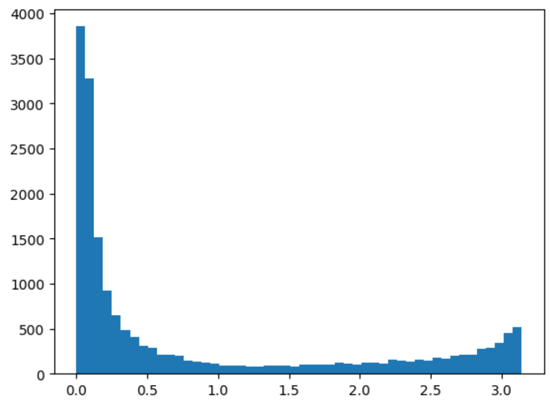


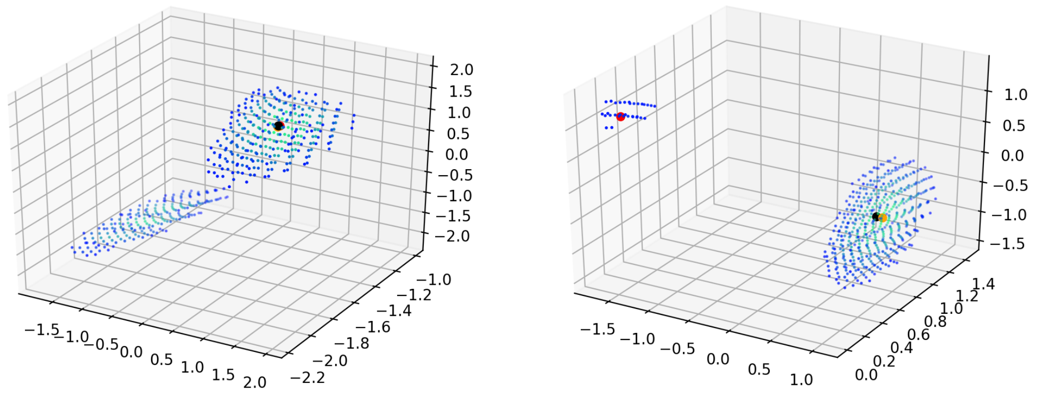
Figure 5.13: Theta histogram of the network on the validation scene Apt-Kitchen with real-world rotations.

Rodrigues' vector visualization. In order to understand what is happening inside the network, it would be great to visualize the weights of the kernels or the feature map of the last layer. In our network, though, these objects are 3-dimensional and are weirdly parametrized with ZYZ-Euler angles. In order to be able to meaningfully visualize them, our approach is to take a pair of corresponding keypoints from the benchmark, feed them to the network, and convert each position of the feature maps of the last SO(3) layer into a Rodrigues' vector (see definition 3.15). At this point, we can visualize the vectors of each keypoint, separately, in a 3D chart in order to see the structure of the output for that keypoint[3]: close rotations will be represented by close Rodrigues' vectors in the space[4]. In order to also understand which are the highest valued vectors (i.e. LRFs), recalling that a high value means high probability in the soft-argmax, we plot them using a heat map color scheme. In figure 5.14 we can see such plot for two different pairs, where we show the top 500 highest values with the following heat map: blue means smaller value, green means higher value. In black, we can see the highest value (argmax), whereas the yellow point is the soft-argmax (the guess of the network): they are usually very close to each other and are of course enclosed in the greenest part of the cluster of points. There is also a red point, which represents the LRF guess for the other keypoint in the pair, aligned to the first one with the ground truth: if the pair is repeatable, the two guesses (yellow and red) should be close to each other. In the figure we can see that there is a clear cluster of points around the yellow point, with probability values decreasing with the distance from

[3]We actually use angles instead of tangents of half-angles in the formula for Rodrigues' vectors, in order to have a better visualization in terms of scale.

[4]The only exception to this is the 180-degree rotation, which can also be represented by a rotation of -180 degrees.

the yellow point itself: this means that the network actually learned to peak the output around its guess for the LRF, even though there are other clusters as well. The distance between the yellow and the red points, instead, is an indicator of the Theta angle: in figure 5.14a they are so close that it's possible to see the black point only, and in fact it is a repeatable pair. Figure 5.14b, instead, shows the output for a non-repeatable pair: the yellow and black points are close, but they are quite far from the red one. However, we can see that there is a little cluster around the red point, meaning that the network was uncertain about its output. This is not always the case, but it indicates that taking the second-best cluster as a second possible guess would slightly improve repeatability at the expense of computation time.



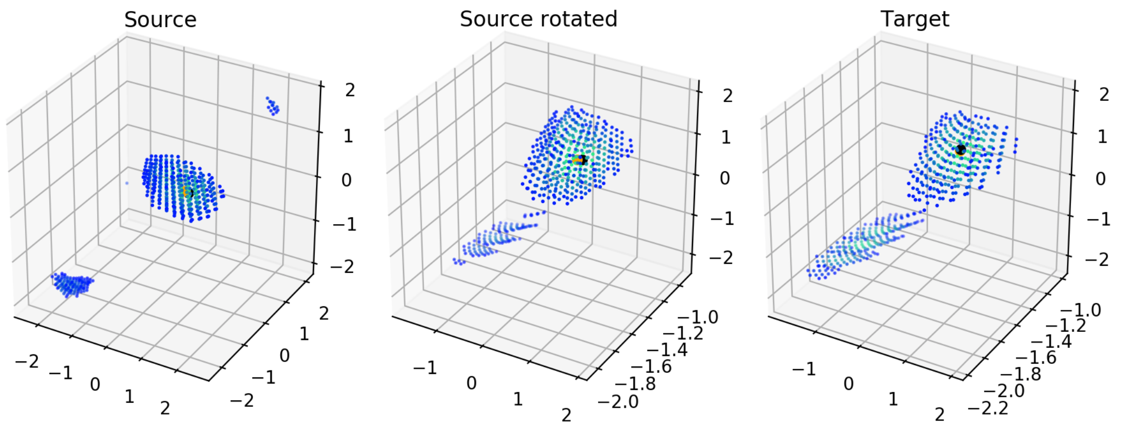
(a) Keypoint from a repeatable pair.   (b) Keypoint from a non-repeatable pair.

Figure 5.14: Visualization with Rodrigues' vectors of the output layer of the network. From blue to green: increasing output value. Black: highest value. Yellow: network guess. Red: guess for the other keypoint of the pair, rotated using the ground truth.

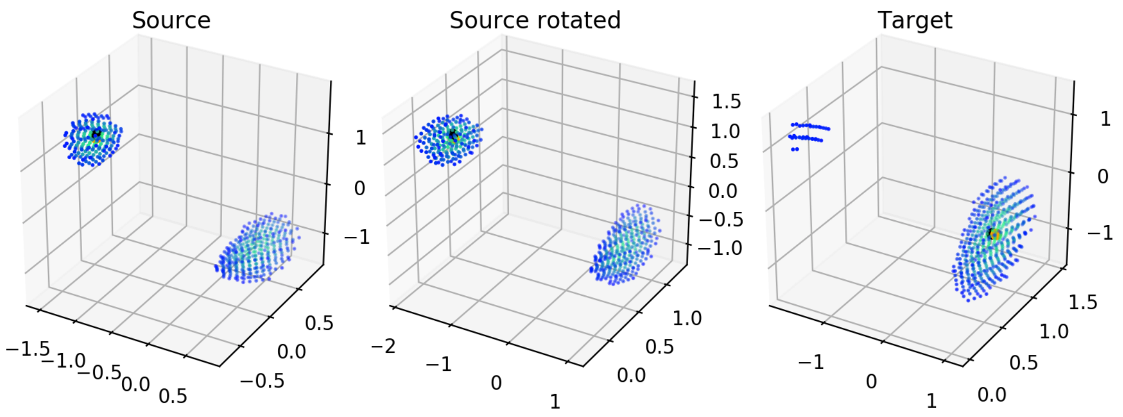
Another interesting plot we can show with Rodrigues' vectors is in figure 5.15, obtained from the same pairs of the previous figures. The two keypoints of a pair are referred to as source and target, and for both of them we show the top 500 values with a heatmap. As before, the black point is the argmax, whereas the yellow point is the guess of the network. The two keypoints belong to differently rotated views, so the central part of the figures shows the rotated version of the source[5], to align it with the target. Figure 5.15a shows a repeatable pair, whereas 5.15b shows a non-repeatable pair: in the former case we can see that the feature maps are much more similar than in the second one, meaning that a loss function that focuses on the similarity of feature maps could be beneficial to this network. In

[5]Rotating Rodrigues' vectors does not correspond to visually rotating the figures, of course.

the latter case, we can see that the network correctly reproduces one of the clusters, but it fails to fully reproduce the one that actually contained the maximum for the source keypoint.



(a) Keypoints from a repeatable pair.



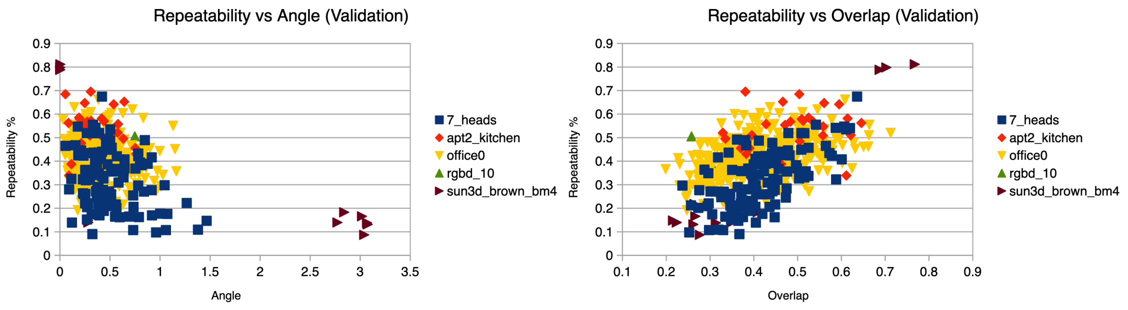
(b) Keypoints from a non-repeatable pair.

Figure 5.15: Visualization with Rodrigues' vectors of the output layer of the network. From blue to green: increasing output value. Black: highest value. Yellow: network guess. On the left, the output for one of the keypoints of the pair, named source; on the right, the output for the other. In the center: the left map rotated with the ground truth of the pair.

Repeatability vs angle and overlap. In order to check why some scenes produced better results than others, here are 2 pairs of plots that show the correlation between the repeatability (y axis) and the overlap percentage or the 3D angle be-

tween each pair of views (x axis)[6]. Figure 5.16 shows the validation set, whereas figure 5.17 shows the test set, all of them computed with the benchmark.
Considering validation data, the correlation is quite clear in both plots, even though with high vertical variability, but the angle seems to carry more information: the Apt-Kitchen scene has the highest performance, and in fact all of its pairs have a relatively small angle separating them. Office0 and Heads, instead, show both very similar overlaps and similar angles: the difference in performance (in favor of the former) is not totally justified by the lower number of high-angle pairs. In fact, Office0 has a higher repeatability even with the same angle and further analysis will be required to understand why.
The same reasoning can be applied to test data as well.

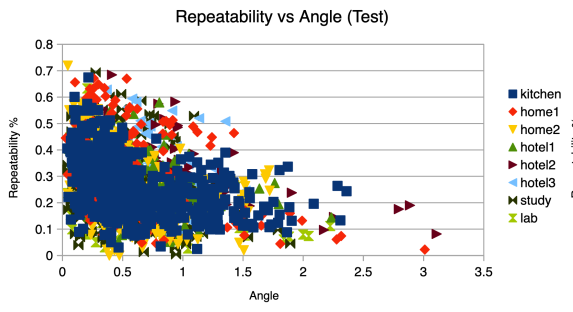
Figures 5.18 and 5.19 show the same plots for FLARE: these results are similar to those of the network, apart from the fact that the repeatability of FLARE does not easily go to 0 even for low-overlap or high-angle pairs, whereas the repeatability of the network does.



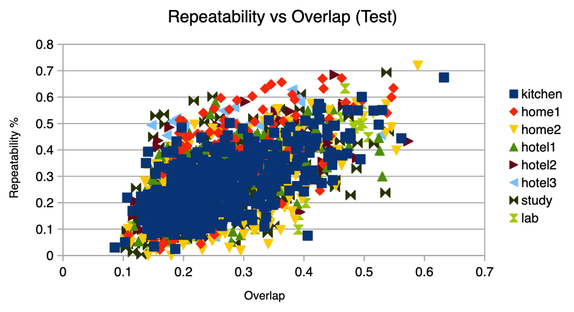
(a) Repeatability vs angle. (b) Repeatability vs overlap percentage.

Figure 5.16: Scatter plots of the benchmark performances of the network on the validation set.

[6]3DMatch doesn't have the overlap information, so it is computed by the benchmark itself.

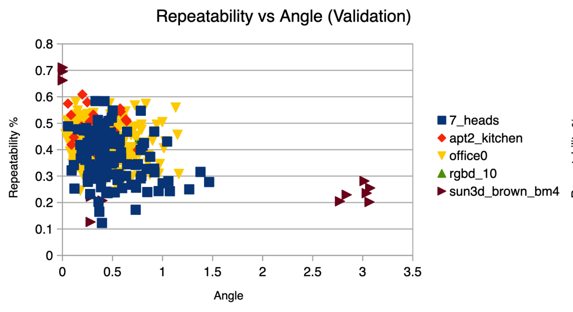


(a) Repeatability vs angle.

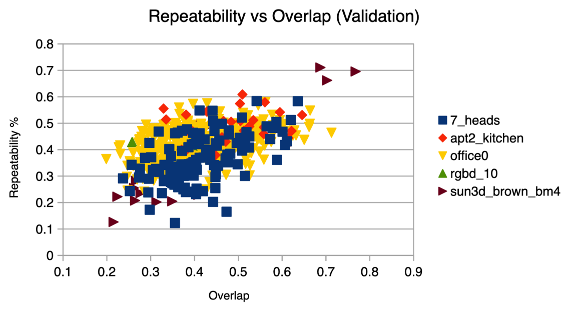


(b) Repeatability vs overlap percentage.

Figure 5.17: Scatter plots of the benchmark performances of the network on the test set.

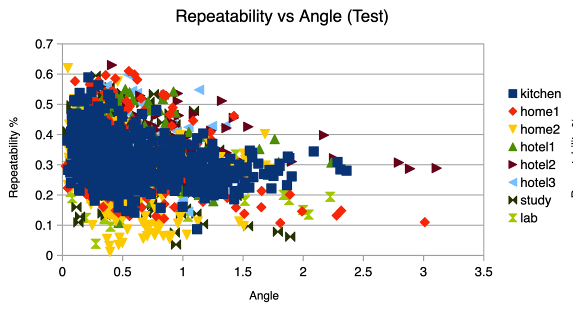


(a) Repeatability vs angle.

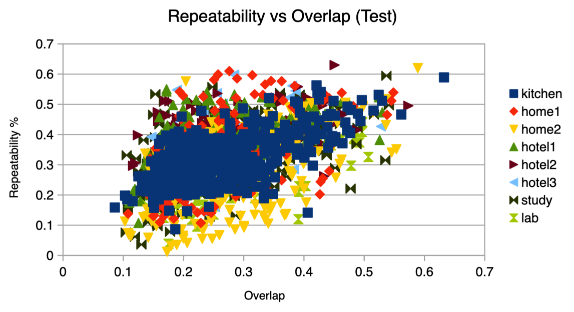


(b) Repeatability vs overlap percentage.

Figure 5.18: Scatter plots of the benchmark performances of FLARE on the validation set.



(a) Repeatability vs angle.



(b) Repeatability vs overlap percentage.

Figure 5.19: Scatter plots of the benchmark performances of FLARE on the test set.

5.2 StanfordViews

This dataset is synthetic and has been created by the CV Lab at the University of Bologna [21]. It consists of random views of models belonging to the Stanford 3D Scanning Repository [7], acquired by using a software simulation of a Kinect device. It's important to notice that this dataset is composed of *meshes*, so FLARE can directly work on it.
It has no training-test splits, and in our work it consists of 3 objects: Bunny, Armadillo and Dragon. They can be seen in figure 5.20. Since there is no training-test split, we could neither proceed to unsupervised learning, nor to weakly supervised learning. Moreover, the amount of data available is considerably lower than with 3DMatch, and in order to have more keypoints we had to rely on lowering the leaf value considerably. For these reasons, the only viable methods are transfer learning from networks trained on 3DMatch and full-fledged adaptation. In addition, we could not use validation performances to select the training checkpoint but, since there were no signs of overfitting on 3DMatch, we simply selected the checkpoint with the best training repeatability.

(a) Bunny. (b) Armadillo. (c) Dragon.

Figure 5.20: Objects of the StanfordViews dataset, figures from the Stanford 3D Scanning Repository [7]

In terms of parameters, the only changes for this dataset are in the learning rate, which is set to 0.0005, and the support radius, whose choice is shown in the next section. The bandwidth has been set to 24, following the considerations made in the previous section. Since this dataset contains overlap information, the benchmark is executed on all pairs with an overlap percentage of at least 10%.

5.2.1 Choice of support radius

Training performances are very similar to those of 3DMatch and thus are not shown here, proving again that synthetic rotations of the same keypoint are easily handled by the network. Instead, here we show the benchmark performances of full-fledged networks, trained from scratch on StanfordViews, comparing support radii (figure 5.21 and table 5.6). The best radius is only 2 cm, very low if compared to the 30 cm used for 3DMatch, but reasonable since the point clouds of StanfordViews represent little objects with higher absolute resolutions. We can also see that increasing the radius above 4 cm improves the repeatability, but quick tests on higher radii showed no advantages in doing so. Thus, a radius of 2 cm will be set for all the following results on this dataset.

Table 5.6: Benchmark repeatability of networks with different support radii. Best result on each column in bold.

	Bunny	Armadillo	Dragon	Mean
1 cm	0.3015	0.2472	0.2807	0.2765
2 cm	**0.3225**	**0.2789**	**0.3013**	**0.3009**
4 cm	0.1622	0.1645	0.1664	0.1644
6 cm	0.1256	0.2009	0.2197	0.1821

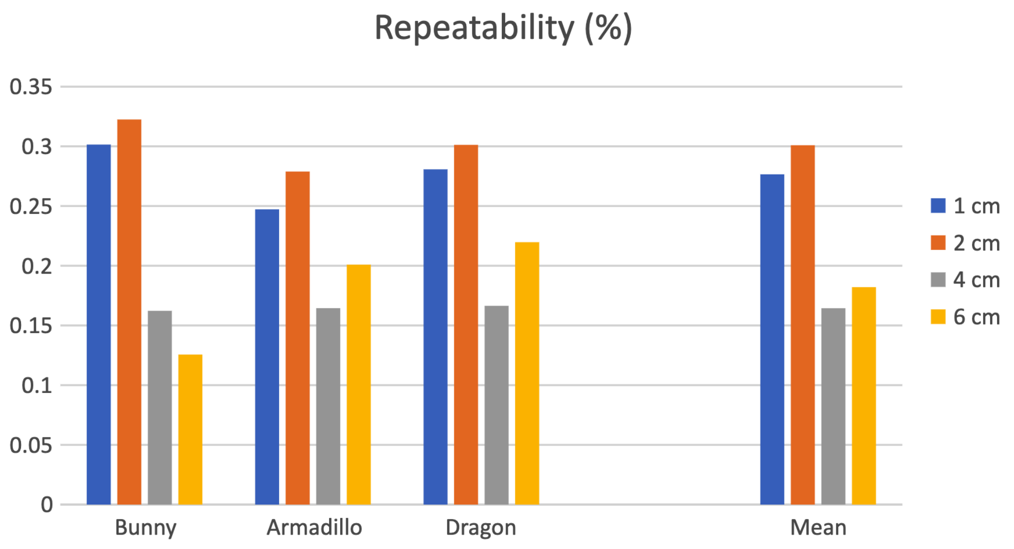


Figure 5.21: Benchmark repeatability chart comparing different support radii.

5.2.2 Full-fledged adaptation.

In this section we show the impact of the full-fledged adaptation on the repeatability scores. We start from the transfer learning of the reference network for 3DMatch, and we gradually adapt it to StanfordViews by means of the full-fledged adaptation. Finally, we show the performances of a network trained from scratch on StanfordViews. Notice that, due to the lower number of keypoints, we have let the network train for much more epochs, in order to maintain approximately constant the total number of keypoint pairs fed to the network. Figure 5.22 and table 5.7 show the comparison.

Table 5.7: Benchmark repeatability of different training methods. Best result on each column in bold.

	Bunny	Armadillo	Dragon	Mean
Transfer Learning	0.05568	0.07506	0.1006	0.07712
FF 2 epochs	0.2495	0.1986	0.2029	0.2170
FF 16 epochs	0.3138	0.2588	0.2778	0.2834
FF 75 epochs	0.2785	0.2656	0.2832	0.2758
FF from scratch	**0.3225**	**0.2789**	**0.3013**	**0.3009**

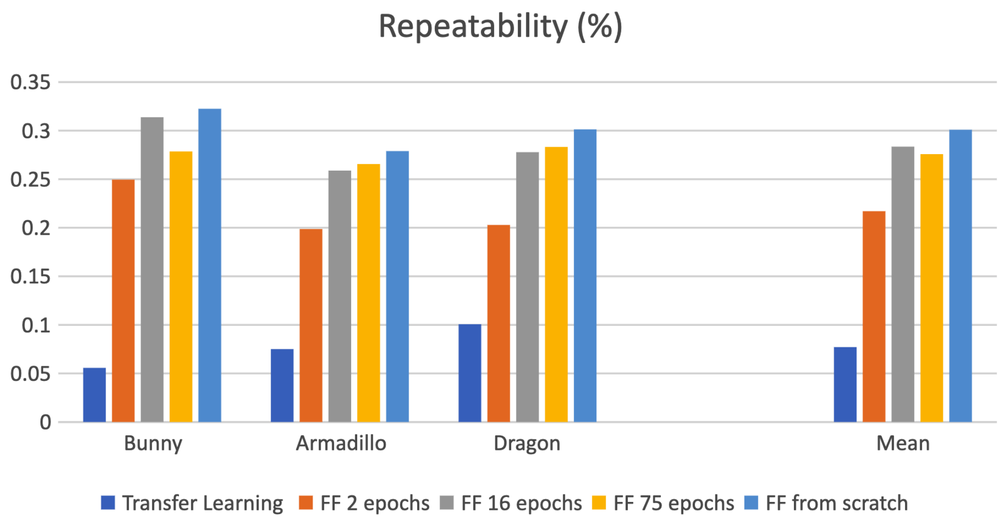


Figure 5.22: Benchmark repeatability chart comparing different training methods.

As we can see, the transfer learning approach doesn't yield good results, but the network quickly adapts to the new dataset, overfitting a little with 75 epochs. The

full-fledged network trained from scratch delivers the best performances anyway, and will be taken as the reference for this dataset.
A very interesting thing we can notice here is that a huge change in the support radius, that has been divided by 15 (from 30 cm to 2 cm), is not as detrimental as one may guess. If we think about it, the network itself has no concept of support radius: it matters only for the spherical voxelization procedure, and it sets the dimension of the visible region to the network. In fact, a small tuning with the full-fledged adaptation is enough to have much better performances. The cause for the lower performances of the transfer learning network is to be correlated with the intrinsic differences between this dataset and 3DMatch, and not with the change in radius.

5.2.3 Overall results

Here we compare our network to FLARE. It is worth reminding that FLARE benefits from meshes, while our network only uses point clouds, and that the performances of FLARE on this dataset have been thoroughly optimized in [21], for each object independently. For this reason, this comparison shows both FLARE on meshes, with parameters from [21], and FLARE on point clouds, with the normal at each point computed using Open3D and a support radius of 2 cm. This latter approach shows a direct comparison between FLARE and our network, since it uses the same support radius as the latter. Both comparisons make sense: the first can compare the network with the best possible results of FLARE; the second, instead, compares the two methods under the same conditions, since meshes are not always available and especially since parameter optimization for each scene/object is not always feasible, for example if no ground truth is available.
Results can be seen in figure 5.23 and table 5.8.

Table 5.8: Benchmark repeatability of the network compared FLARE. Best result on each column in bold.

	Bunny	Armadillo	Dragon	Mean
FF from scratch	0.3225	0.2789	0.3013	0.3009
FLARE on mesh	**0.629**	**0.464**	**0.521**	**0.538**
FLARE on point cloud	0.348	0.170	0.173	0.230

The best performing method is clearly FLARE on meshes, whose parameters are individually optimized for each object. To let the network leverage the mesh information as well, we could create additional input channels when processing meshes. This is left to future work.

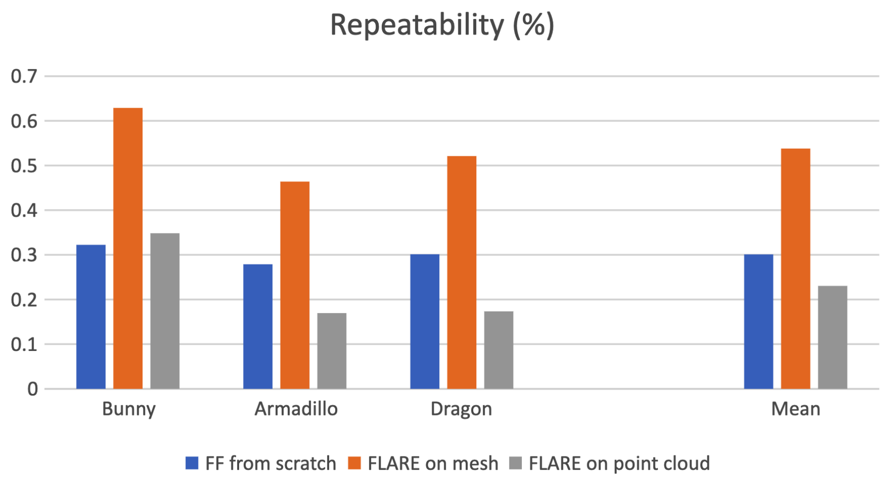


Figure 5.23: Benchmark repeatability chart comparing the network to FLARE.

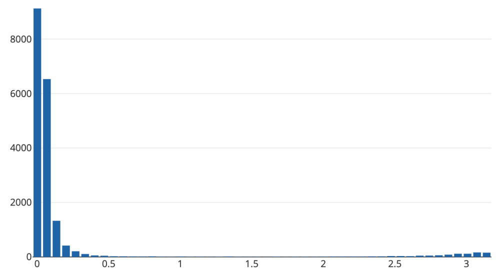
On the contrary, the comparison on point clouds is won by the network, which delivers better performances than FLARE when they both work on the same input representation and with the same parameters on all the objects.

5.2.4 Result analysis

Our results on StanfordViews cannot be considered state of the art, because they are easily surpassed by FLARE, but the fact that we are bound to point clouds only, and that we beat FLARE when it is constrained to work on them, is an indicator that our direction is promising, and that we may be able to improve our performances by endowing the network with the ability to process richer input data.

Theta histograms. As for 3DMatch, the histogram of Theta angles is bimodal. In figure 5.24a we can see it during training, with synthetic rotations: it is apparently almost perfect. However, during the benchmark, it becomes substantially worse, as in figure 5.24b, even though it is still better than the histograms on 3DMatch. Figure 5.25a shows the same histogram for FLARE on meshes, which is clean as usual; instead, figure 5.25b shows that FLARE on point clouds produces a much worse histogram than the one of the network, clearly bimodal and with a right peak even higher than the left peak.

Repeatability vs angle and overlap. Rodrigues' vector do not produce any new information on this dataset, so they are not shown. Instead, in figure 5.26


(a) Training data, on the whole dataset.

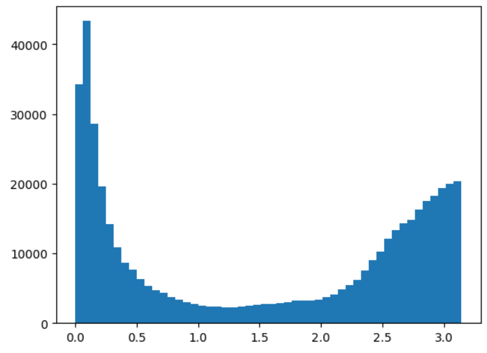
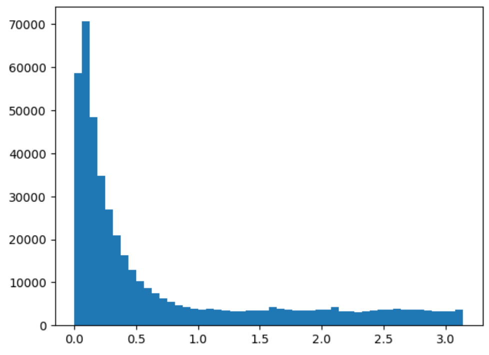

(b) Benchmark data, on Armadillo.

Figure 5.24: Theta histograms of the network.


(a) Using meshes.

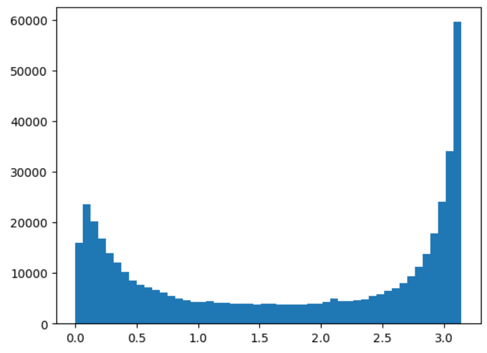
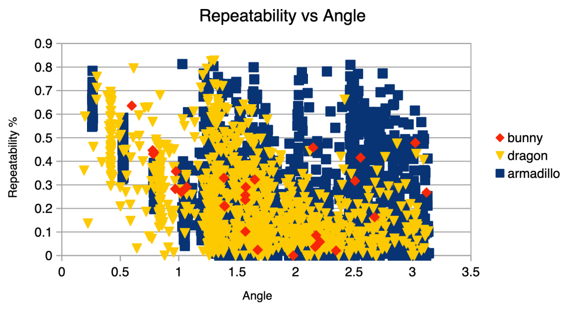

(b) Using point clouds.

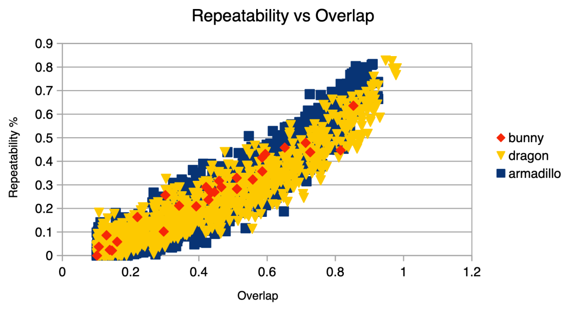
Figure 5.25: Theta histograms of FLARE on Armadillo.

we can see the correlation between repeatability, angle and overlap for the network: this time, the relationship between repeatability and angle seems to be quite unpredictable, but the relationship with the overlap (which is provided with the dataset in this case), is very clear. As a comparison, figure 5.27 shows the same data for FLARE on meshes: the big difference between the two is that the relationship with overlap has a low incline for FLARE, whereas it is much more steep for the network.
Figure 5.28 shows FLARE on point clouds, whose correlation with the overlap has worsened to become almost exponential.

In addition, the angle plots show that FLARE has a lower overall variability, with the network going to 0 much more easily; in the overlap plots its FLARE that has more variability, but its mesh version still doesn't go to 0 even for very low overlaps, indicating that the usage of surface normals is probably beneficial for the estimation of a LRF. This is consistent to what was found in [22].

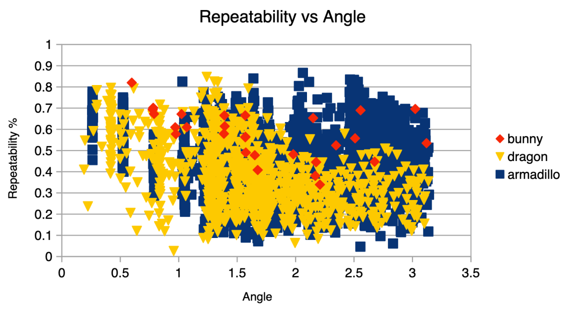


(a) Repeatability vs angle.

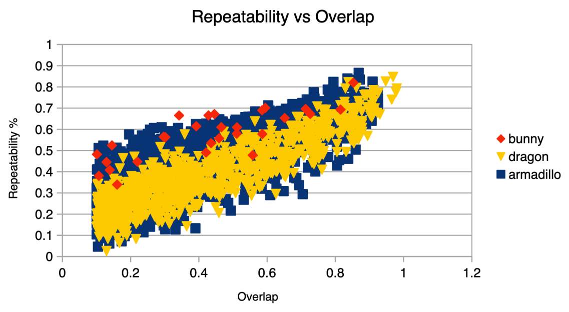


(b) Repeatability vs overlap percentage.

Figure 5.26: Scatter plots of the benchmark performances of the network.

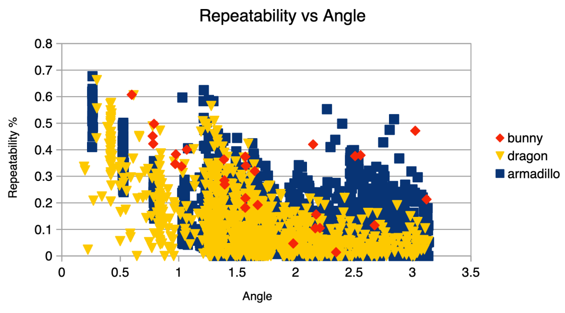


(a) Repeatability vs angle.

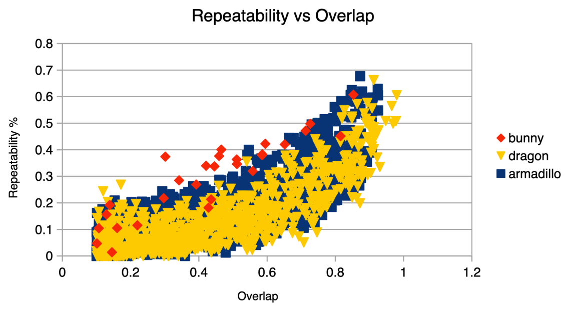


(b) Repeatability vs overlap percentage.

Figure 5.27: Scatter plots of the benchmark performances of FLARE on meshes.



(a) Repeatability vs angle.



(b) Repeatability vs overlap percentage.

Figure 5.28: Scatter plots of the benchmark performances of FLARE on point clouds.

5.3 ETH

The dataset we refer to as “ETH” has been produced by *Pomerleau et al.* for their work on registration [23]. We used 4 real-world outdoor scenes from their work, recorded with a 3D scanner and saved as point clouds: as a consequence, FLARE will work by computing normals with Open3D, as for 3DMatch, and without optimizing its parameters, just as the network. The scenes we used in this dataset are, namely: Gazebo Summer, Gazebo Winter, Wood Summer and Wood Autumn. Figure 5.29 shows an example from Gazebo Summer.

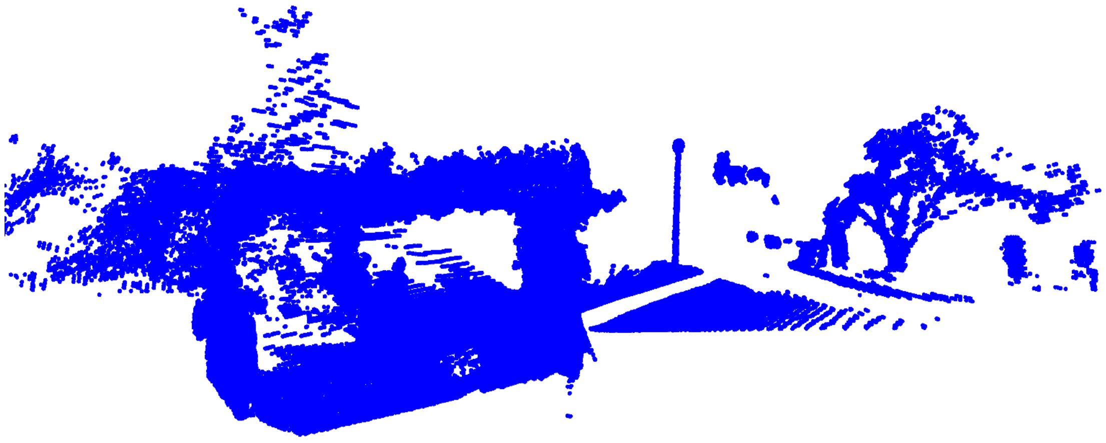


Figure 5.29: A view of Gazebo summer. On the right it is possible to see a tree, in the center a light post, and on the left a gazebo with 3 people sitting on a bench.

This dataset has not been tested extensively: it has been mainly used to assess the generalization capabilities of the network to outdoor scenes. Moreover, since these point clouds are outdoor and involve higher distances than 3DMatch and StanfordViews, the support radius has been set to a higher value: 1 meter.
The rest of the parameters is the same as for 3DMatch. As for StanfordViews, instead, this dataset has no training-test split, and thus is not suitable for unsupervised of weakly supervised learning. Consequently, we are bound to transfer learning and full-fledged adaptation, choosing the checkpoints as in StanfordViews.

5.3.1 Overall results

We tried a transfer learning from 3DMatch[7] by keeping the same radius, 30 cm; a transfer learning with an increased radius, 1 meter; and a full-fledged adaptation

[7]Using the usual reference network: unsupervised learning on the whole training set, random sampling on.

from scratch with a radius of 1 meter as well. These methods are compared to FLARE, as usual, which has been executed with the same radius of 1 meter, in figure 5.30 and table 5.9.

Table 5.9: Benchmark repeatability of the network, with different training methods, compared to FLARE. best result on each column in bold.

	Gazebo Summer	Gazebo Winter	Wood Summer	Wood Autumn	Mean
TL (30 cm)	0.140186	0.111601	0.102966	0.120231	0.118746
TL (1 m)	0.180929	0.156678	0.154393	0.16935	0.1653375
FF (1 m)	0.3142	**0.3236**	**0.3414**	**0.3708**	**0.3375**
FLARE (1 m)	**0.345**	0.268	0.210	0.236	0.265

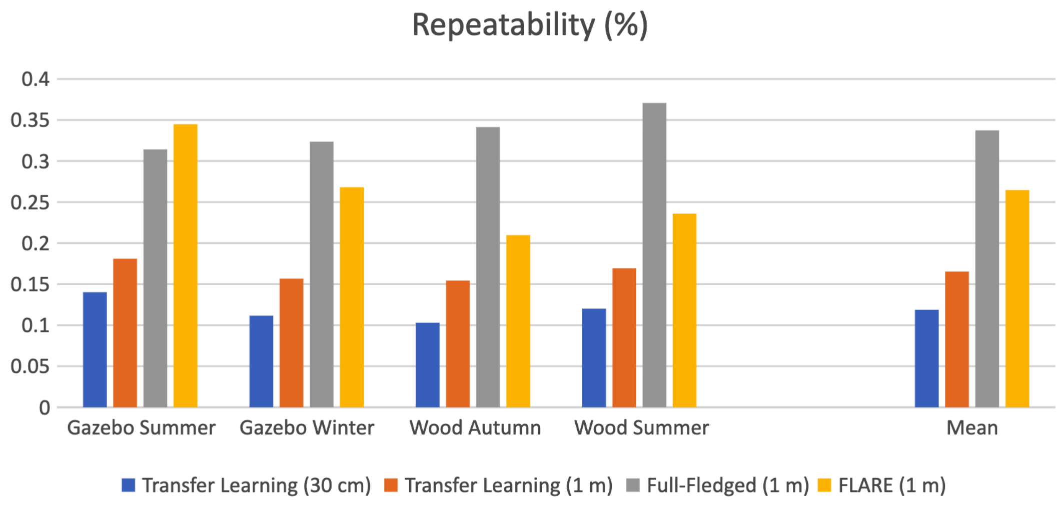


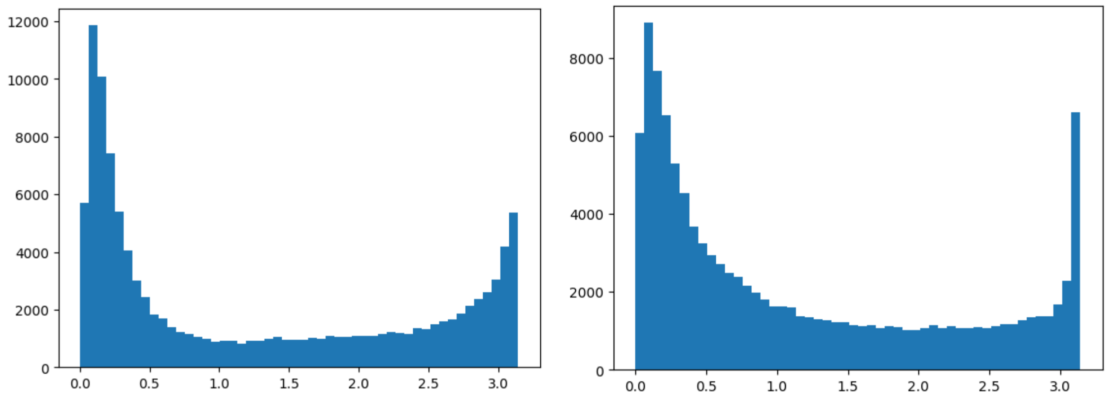
Figure 5.30: Benchmark repeatability chart comparing the network, with different training methods, to FLARE.

As we can see, the best method overall is the full-fledged network, that is able to easily surpass FLARE with the exception of Gazebo Summer; the transfer learning approach appears to perform much worse, both maintaining its original radius and expanding it to 1 meter. As a side note, though, FLARE executed on a 30 cm radius performs slightly worse than the network in transfer learning.
For the transfer learning approaches, we can infer the same considerations as for StanfordViews, and so that a change in the support radius is not detrimental: it is, in fact, beneficial. The lower performances of transfer learning when compared to the full-fledged approach are likely related to intrinsic differences between ETH and 3DMatch, and not to the different support radius.

For the following analysis we consider the full-fledged (1 m) method as the reference network.

5.3.2 Result analysis

Theta histograms. The network still presents a slightly bimodal histogram in this dataset as well, but it is much better than for 3DMatch. Skipping over training histograms, which are analogous to what we have already shown for the previous datasets, in figure 5.31a we can see the benchmark histogram of the network on Gazebo Winter, which is a good indicator of the average performance for both the full-fledged network and FLARE, and we can see that the right peak is smaller than usual and that the trail is lower in height. The histograms for the Wood scenes are even better than this. Figure 5.31b, instead, shows Gazebo Winter for FLARE: it is bimodal as well, the left peak is lower than the one of the network, and it appears to be thicker in width; the right peak, instead, is narrower than the one of the network, but it is slightly taller (mind the scale). The other scenes show a very similar behavior, for both methods.

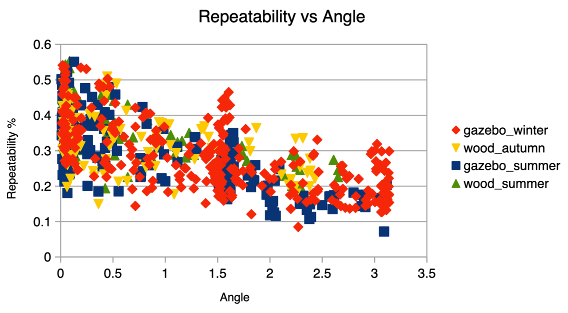


(a) Network. (b) FLARE.

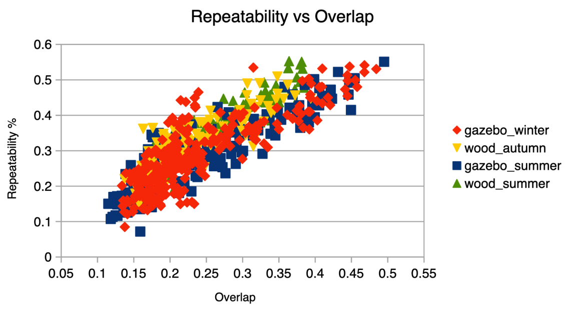
Figure 5.31: Theta histograms on Gazebo Winter.

Repeatability vs angle and overlap. Figure 5.32 shows the repeatability of the network, whereas 5.33 shows the same plots for FLARE. As for 3DMatch, overlap information is computed by the benchmark.
Contrary to the previous datasets, this time the repeatability of the network does not go to 0 even for very low overlaps or very high angles, and this is a good indicator of performances. Moreover, we can see that the correlation between repeatability and overlap has a higher steepness for the network when compared

to FLARE, but they both start from similar values for low overlaps, meaning that the network is able to deliver substantially better performances as soon as the overlap increases.
Analogous considerations can be applied to the correlation between repeatability and angle.

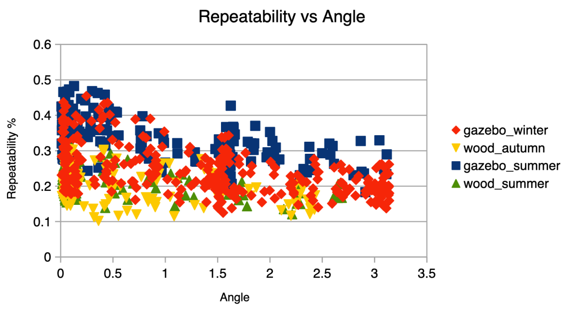


(a) Repeatability vs angle.

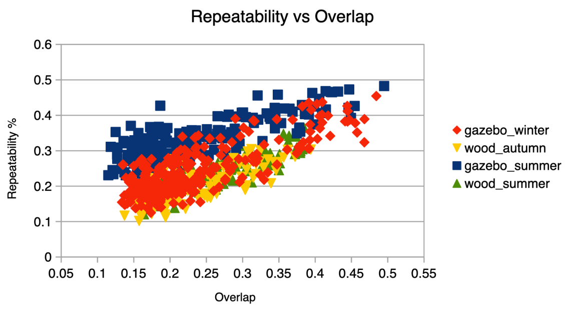


(b) Repeatability vs overlap percentage.

Figure 5.32: Scatter plots of the benchmark performances of the network.



(a) Repeatability vs angle.



(b) Repeatability vs overlap percentage.

Figure 5.33: Scatter plots of the benchmark performances of FLARE.

Chapter 6

Conclusions and future developments

In this thesis, we provided insights into how existing Local Reference Frames work, showing that the vast majority of them are still based on a hand-crafted approach to algorithm design, with the only partial exception of LRF-Net [37], which however processes with a data-driven approach a highly-engineered representation of the input data. After providing the mathematical background to spherical and SO(3) convolutions, we showed how they can be used to shift the CNN paradigm from 2D translation-equivariance to 3D rotation-equivariance, as done in [35] and [26], by using a Spherical Convolutional Neural Network with rotation-equivariance properties, meaning that it can produce rotated outputs in the face of rotated inputs: the key property of Local Reference Frames. Afterwards, we defined three training methodologies (unsupervised, full-fledged adaptation and weakly supervised), as well as a benchmarking framework. We tested the proposed network, trained by means of these methodologies, on three different datasets, namely 3DMatch, StanfordViews and ETH, and we have compared them to the current state of the art, represented by FLARE [21], and to SHOT [29], a very popular LRF. Our results are promising, and show that: we easily beat SHOT; we are on par with FLARE on 3DMatch, having a small lead when we use the weakly supervised training protocol; and we are consistently superior on ETH by using the full-fledged approach, which proved to be very effective for smaller datasets that do not have a training-test split. The situation in StanfordViews is different, because it is made of meshes instead of point clouds. In this dataset, FLARE can use mesh information but our network is constrained to convert it to a point cloud, and FLARE performs better. It is worth noticing that its parameters have been explicitly optimized for StanfordViews. However, when they are both constrained to work on the point cloud version of StanfordViews, the network beats FLARE by a clear margin.

Overall, our method implements end-to-end learning on Local Reference Frames and fills the existing gap on learned LRFs with results that can be compared to, or can improve, the current state of the art, while enjoying the benefits of being an end-to-end learning approach, i.e. no need to be tuned on each specific scene/dataset.

Yet, there is still room for improvements and optimizations of the proposed architecture. The following list contains the most interesting future directions that have arisen during this work.

- Test the performances of the network in other tasks, e.g. descriptor matching, for example by using it as the LRF for descriptors such as *Spezialetti et al.* [26] (whicih leverages Spherical CNNs as well), TOLDI [34], SHOT [29], CGF [16], and others.

- Test higher bandwidths, because our tests indicate that they may be able to provide better results, and test a higher number of layers, using for example a decreasing bandwidth towards the soft-argmax layer.

- Create additional input channels when processing meshes, e.g. by accumulating in the input spherical signal the total mesh area in each bin or the mean orientation of the normals to input triangles, to increase the performance of the network on these input formats.

- Introduce a better data augmentation process that simulates the nuisances of the weakly supervised training, for example by deleting some portions of the support or by adding noise, in order to achieve higher performances with the unsupervised training, since the ground truth necessary for the weakly supervised training is not always available.

- Substitute the Parzen window with a more principled operation and, simultaneously, introduce different loss functions that can help the network to also produce similar feature maps for corresponding keypoints, instead of just concentrating on repeatable maxima. These losses can be, for example, the KL divergence, the JS divergence or the Wasserstein distance between feature maps of corresponding keypoints. It's also possible to apply these losses to a feature map and a Gaussian curve, thus helping the network to uniformly peak its output around the maximum without using the Parzen window.

- Avoid the use of a manually-defined function for the soft-argmax layer, and employ fully connected layers instead. *Spezialetti et al.* [26] have shown that an MLP can produce equivariant outputs in the face of equivariant inputs: their Spherical CNN produces a compact equivariant descriptor, and their

MLP Decoder is able to reconstruct the input from this descriptor. The use of some fully connected layers at the end of our network may let it learn how to convert the SO(3) feature maps into LRFs more effectively.

- Study the Theta histogram in order to understand why it is bimodal, whether the rotation axis is always the same, and what is the common property of the keypoints that end up in its right-most peak.
- Study the view pairs to understand why some of them (even in the same scene) have higher repeatabilities than other pairs with the same overlap or angle.

Bibliography

[1] H. G. Barrow et al. "Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching". In: *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'77. Cambridge, USA: Morgan Kaufmann Publishers Inc., 1977, 659–663.

[2] Gregory S. Chirikjian and Alexander B. Kyatkin. "Engineering Applications of Noncommutative Harmonic Analysis: With Emphasis on Rotation and Motion Groups". In: 2000.

[3] Chin Seng Chua and Ray Jarvis. "Point Signatures: A New Representation for 3D Object Recognition". In: *International Journal of Computer Vision* 25.1 (1997), pp. 63–85. DOI: `10.1023/A:1007981719186`. URL: `https://doi.org/10.1023/A:1007981719186`.

[4] Taco Cohen and Max Welling. "Group Equivariant Convolutional Networks". In: *ICML*. 2016.

[5] Taco Cohen et al. "Convolutional Networks for Spherical Signals". In: Jan. 2017.

[6] Taco Cohen et al. "Spherical CNNs". In: (Jan. 2018).

[7] Brian Curless and Marc Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, 303–312. ISBN: 0897917464. DOI: `10.1145/237170.237269`. URL: `https://doi.org/10.1145/237170.237269`.

[8] Angela Dai et al. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration". In: *ACM Transactions on Graphics 2017 (TOG)* (2017).

[9] Jian S. Dai. “Euler-Rodrigues formula variations, quaternion conjugation and intrinsic connections”. In: *Mechanism and Machine Theory* 92 (2015), pp. 144 –152. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2015.03.004. URL: http://www.sciencedirect.com/science/article/pii/S0094114X15000415.

[10] Haowen Deng, Tolga Birdal, and Slobodan Ilic. “PPF-FoldNet: Unsupervised Learning of Rotation Invariant 3D Local Descriptors”. In: Sept. 2018, pp. 620–638. ISBN: 978-3-030-01227-4. DOI: 10.1007/978-3-030-01228-1_37.

[11] Haowen Deng, Tolga Birdal, and Slobodan Ilic. “PPFNet: Global Context Aware Local Features for Robust 3D Point Matching”. In: (Oct. 2018). DOI: 10.1109/CVPR.2018.00028.

[12] Zan Gojcic et al. “The Perfect Match: 3D Point Cloud Matching with Smoothed Densities”. In: (Nov. 2018).

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[14] Yulan Guo et al. “Rotational Projection Statistics for 3D Local Surface Description and Object Recognition”. In: *International Journal of Computer Vision* 105 (Apr. 2013), pp. 63–86. DOI: 10.1007/s11263-013-0627-y.

[15] Andrew Johnson and Martial Hebert. “Using Spin-Images for Efficient Object Recognition in Cluttered 3-D Scenes”. In: (Aug. 1998).

[16] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. “Learning Compact Geometric Features”. In: (Sept. 2017).

[17] Kevin Lai, Liefeng Bo, and Dieter Fox. “Unsupervised feature learning for 3D scene labeling”. In: May 2014, pp. 3050–3057. DOI: 10.1109/ICRA.2014.6907298.

[18] Simone Melzi et al. “GFrames: Gradient-Based Local Reference Frame for 3D Shape Matching”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[19] Ajmal Mian, Mohammed Bennamoun, and Robyn Owens. “On the Repeatability and Quality of Keypoints for Local Feature-based 3D Object Retrieval from Cluttered Scenes”. English. In: *International Journal of Computer Vision* 89.2-3 (2010), pp. 348–361. ISSN: 0920-5691. DOI: 10.1007/s11263-009-0296-z.

[20] John Novatnack and Ko Nishino. “Scale-Dependent/Invariant Local 3D Shape Descriptors for Fully Automatic Registration of Multiple Sets of Range Images”. In: Jan. 2008, pp. 440–453. DOI: 10.1007/978-3-540-88690-7_33.

[21] A. Petrelli and L. Di Stefano. “A Repeatable and Efficient Canonical Reference for Surface Matching”. In: *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission.* 2012, pp. 403–410. DOI: 10.1109/3DIMPVT.2012.51.

[22] A. Petrelli and L. Di Stefano. “On the repeatability of the local reference frame for partial shape matching”. In: *2011 International Conference on Computer Vision.* 2011, pp. 2244–2251. DOI: 10.1109/ICCV.2011.6126503.

[23] François Pomerleau et al. “Challenging data sets for point cloud registration algorithms”. In: *The International Journal of Robotics Research* 31.14 (2012), pp. 1705–1711. DOI: 10.1177/0278364912458814. eprint: https://doi.org/10.1177/0278364912458814. URL: https://doi.org/10.1177/0278364912458814.

[24] Samuele Salti, Federico Tombari, and Luigi Di Stefano. “SHOT: Unique Signatures of Histograms for Surface and Texture Description”. In: *Computer Vision and Image Understanding* 125 (Aug. 2014). DOI: 10.1016/j.cviu.2014.04.011.

[25] Jamie Shotton et al. “Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images”. In: *Proc. Computer Vision and Pattern Recognition (CVPR).* IEEE, 2013. URL: https://www.microsoft.com/en-us/research/publication/scene-coordinate-regression-forests-for-camera-relocalization-in-rgb-d-images/.

[26] Riccardo Spezialetti, Samuele Salti, and Luigi di Stefano. “Learning an Effective Equivariant 3D Descriptor Without Supervision”. In: *ArXiv* abs/1909.06887 (2019).

[27] G. K. L. Tam et al. “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. ISSN: 2160-9306. DOI: 10.1109/TVCG.2012.310.

[28] Federico Tombari, Samuele Salti, and Luigi Di Stefano. “Unique shape context for 3D data description”. In: (Jan. 2010). DOI: 10.1145/1877808.1877821.

[29] Federico Tombari, Samuele Salti, and Luigi Di Stefano. “Unique Signatures of Histograms for Local Surface Description”. In: vol. 6313. Sept. 2010, pp. 356–369. DOI: 10.1007/978-3-642-15558-1_26.

[30] Julien Valentin et al. “Learning to Navigate the Energy Landscape”. In: *arXiv preprint arXiv:1603.05772* (2016).

[31] Daniel Worrall and Gabriel Brostow. “CubeNet: Equivariance to 3D Rotation and Translation”. In: (Apr. 2018).

[32] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. “SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels”. In: Dec. 2013, pp. 1625–1632. ISBN: 978-1-4799-2840-8. DOI: 10.1109/ICCV.2013.458.

[33] J. Yang, Y. Xiao, and Z. Cao. “Toward the Repeatability and Robustness of the Local Reference Frame for 3D Shape Matching: An Evaluation”. In: *IEEE Transactions on Image Processing* 27.8 (2018), pp. 3766–3781. ISSN: 1941-0042. DOI: 10.1109/TIP.2018.2827330.

[34] Jiaqi Yang et al. “TOLDI: An effective and robust approach for 3D local shape description”. In: *Pattern Recognition* 65 (Nov. 2016). DOI: 10.1016/j.patcog.2016.11.019.

[35] Yang You et al. “PRIN: Pointwise Rotation-Invariant Network”. In: (Nov. 2018).

[36] Andy Zeng et al. “3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions”. In: July 2017, pp. 199–208. DOI: 10.1109/CVPR.2017.29.

[37] Angfan Zhu et al. “LRF-Net: Learning Local Reference Frames for 3D Local Shape Description and Matching”. In: (Jan. 2020).