POLYTECHNIC OF TURIN

Master degree course in Computer Engineering
Data Analytics

Master Degree Thesis

NN-based approach for Object Detection and 6DoF Pose Estimation with ToF Cameras in Space

Supervisors
Prof. Elena Maria BARALIS
Ing. Andrea MERLO

Candidate
Alkis KOUDOUNAS
ID Number: s278266

Thesis Tutor
Thales Alenia Space - Italy
Ing. Alessandro SCARCIGLIA

DECEMBER 2021

Summary

Recently introduced 3D Time-of-Flight (ToF) cameras have shown a huge potential for mobile robotic applications, proposing a smart and fast technology that outputs 3D point clouds, lacking however in measurement precision and robustness. One advantage of their usage is the complete removal of the typical stereo vision pipeline, but they are subject to noise depending on the density and reflectivity of the materials hit by their illuminators. With the development of this low-cost sensing hardware, 3D perception gathers more and more importance in robotics as well as in many other fields, and object registration arouses everyday more attention. Registration is a transformation estimation problem between two input point clouds, seeking the transformation that best aligns the source to the target. This thesis work aims at providing a comprehensive survey on ToF cameras' calibration and denoising techniques, mostly based on deep learning, and on point cloud registration approaches. After having studied and compared the state-of-the-art frameworks according to several important metrics, the goal is to design a NN-based solution able to robustly detect known objects observed by a ToF (PMD Camboard PicoFlexx) camera within a short range, estimating their 6 DoF position. This is focused on demonstrating the capability to detect a part of a satellite (i.e., a gripping interface) to support in-orbit servicing missions. Experiments reveal that deep learning techniques can obtain higher accuracy and robustness than classical methods, handling significant amount of noise while still keeping real-time performance and low complexity of the models themselves. The developed AI-based approach offers an interesting new range of possibilities, that mated with the increasing precision and output richness of ToF cameras could enable efficient and light-embedded solutions for robot sensing.

Acknowledgements

First and foremost I would like to thank Thales Alenia Space for having given me this amazing opportunity to work with this topic, allowing me to jointly deepen my two passions, AI and Space, and to conclude my studies in the best possible way. I'd love to sincerely thank Andrea for having made it all possible, Alessandro for having followed me from the very beginning of this work.

I would also like to personally thank all those people that helped me in understanding several gloomy aspects throughout the whole work. First of all, Gianluca Agresti for having enlightened me with his brilliant thoughts about his previous works, Fabio Manganaro and Stefano Pini for having provided me some useful insights about the PMD PicoFlexx camera behaviour. Last but not least also Fernando Santana Falcón and Eduard Mann for having offered me important explanations regarding some characteristics of the ToF sensors, and of the PicoFlexx in particular.

I finally want to thank professor Baralis for having accepted and supported this project, being always available when needed the most.

Contents

List of Tables

List of Figures

Part I

State-of-the-Art Methods Analysis

Chapter 1

Introduction

This thesis work represents the last chapter of my school career, and it couldn't have done anything else but connecting my two passions, AI and Space, in a deeply tight way. It is indeed focused on the development of a NN-based solution to robustly detect known objects observed through a Time-of-Flight (ToF) camera within a short range, estimating their 6 *DoF* position. However, having been developed in Thales Alenia Space - Italy, everything in this research is done thinking of a real application in a space environment, thus every choice is made always having in mind the complexity of an embedded domain such as the avionics. The aim of the designed approach is in fact to demonstrate the capability to detect a part of a satellite (such as a gripping interface) to support in-orbit servicing missions.

In the domain of mobile robotics, 3D ToF cameras have recently represented new fundamental and vast prospects [65]. They are clever and swift devices, lacking however in accuracy of measurements and mostly in robustness [64]. One advantage of their usage is the complete removal of the typical stereo vision pipeline since they output 3D point clouds. Nonetheless, they are subject to noise depending on the density and reflectivity of the materials hit by their illuminators. With the development of this low-cost sensing hardware, 3D perception gathers more and more importance in robotics as well as in many other fields, and object registration arouses everyday more attention [80]. Registration is a transformation estimation problem between two input point clouds, seeking the transformation that best aligns the source to the target [48].
This thesis can thus be split in two parts. Firstly, the ToF cameras' working principles have been studied and analyzed, along with the state-of-the-art methods for what concerns Multi-Path Interference (MPI) and shot denoising of ToF raw data (Chapter 2) and, lastly, with the state-of-the-art methods regarding the point cloud registration task (Chapter 3). Secondly, the best approaches are selected on the basis of six metrics among the over twenty different studied frameworks and implemented adapting them to the input acquired by a PMD Camboard PicoFlexx sensor (Chapters 4, 5 and 6). The goal of this second part, as anticipated, is the design of an end-to-end framework capable of robustly detecting known objects and estimating their 6 *DoF* pose in real-time.

ToF cameras are VCSEL-based active sensors that output 3D point clouds. Based on the photon mixer device (PMD) technology [90], these sensors use the principle of modulation interferometry: an illumination module attached to the camera emits a near-infrared

(NIR) light which illuminates the focused 3D scene. The diffusely remitted light transports the distance information in terms of phase delay with respect to the emitted signal. The high data rate of these sensors, their low weight, and the very small size make them very suitable for autonomous robotics tasks [65]. Moreover, their usage could completely remove the typical stereo vision pipeline. However, these cameras are extremely vulnerable to changing lighting conditions: this inevitably results in inaccurate data and mistaken raw measurements. These errors are influenced by both the physical properties of the sensor and the environmental conditions. The latter is of a paramount importance in the robotics field since the real world is unpredictable, therefore it is obvious that without online environment adaptation algorithms and data pre-processing this kind of sensor is not appropriate for autonomous mobile robotic tasks. In fact, depending on external interfering factors, such as sunlight, and scene configurations (i.e., distance, orientation, density, and reflectivity of the materials), the depth measurements of the same scene can be captured in very different ways according to different perspectives [64]. The performance of distance measurements with ToF cameras is thus limited by several errors: (1) systematic errors, that are predictable and addressable by calibration due to their systematic occurrence and (2) non-systematic errors, which are much more difficult to remove and need more sophisticated techniques.

This research wants to study the state-of-the-art methods both for the systematic errors' correction via lateral and depth calibration and for the non-systematic errors' removal via deep learning techniques. For what concerns the first part, the Pinhole method [89] is thoroughly studied for the lateral calibration, while for the depth one several approaches based on global adjustment and per-pixel distance calibration are introduced [33, 58, 90, 103]. Regarding the denoising of the depth map and the non-systematic errors' correction, many deep learning frameworks are presented in Chapter 2, starting from the very first and going through the literature reaching the last-introduced and nowadays-used methods. After having studied their characteristics and architectures, all the proposed methods are compared according to six different metrics [86, 46], namely accuracy, size of model, robustness, time cost, latency, and range of application, and two of them are finally selected to be reproduced (Chapter 5), *SHARP-Net* [27] and Coarse-Fine CNN [3, 2]. The latter turned out to be impracticable to work in real-time, which is a major constraint for the (avionics) robotic applications, thus the former is finally used for the denoising phase in this research. *SHARP-Net* [27] ("Spatial Hierarchy Aware Residual Pyramid Network") is able to refine the depth measurement by removing *MPI* (Multi-Path Interference) noise, a phenomenon for which -due to inter-reflections in the scene- the remitted near-infrared (NIR) signal is a superposition of NIR light that has travelled different distance, having the side effect of sanding off hollows and corners [64].

The main goal of the thesis is that of developing a framework capable of recognizing a known object and estimating its pose with respect to a ground truth value, according to its rotation and its translation within 6 *DoF*. As already anticipated, this problem is referred to as Point Cloud Registration, and indeed the output of the ToF camera is a (noisy) point cloud that, after being denoised with the previously described methods, must be aligned with a target one.
Thus, several state-of-the-art architectures are studied (Chapter 3) and compared (Chapter 5) according to the same six indicators already used for the denoising task. After having given a historical perspective of the registration problem [74], the solutions are cast and

differentiated, based on a few elements, into three frameworks [48]: (1) optimization-based models are the ones that iteratively compute the correspondences and the transformation between the input point sets, (2) feature-learning techniques estimate the features (either with a deep neural network or with another approach) and only in a second moment iteratively estimate the correspondences and the transformation, while finally (3) end-to-end learning-based methods employ an end-to-end deep learning architecture to calculate the final transformation that best aligns the point clouds. Out of all the proposed approaches, one algorithm per each type of registration framework is chosen, that is, Fast Global Registration [106] for the optimization-based models, FPFH-RANSAC [79, 31] for the feature-learning methods and Feature-Metric Registration [47] for the end-to-end learning-based ones.

This study aims at demonstrating that learning-based approaches can behave better with respect to classical methods, even with serious constraints for what concerns the complexity of the models and the real-time performance. After having understood how the PMD Camboard PicoFlexx sensor [73] (the ToF camera kindly offered by the company) works, its properties and its intrinsic and extrinsic parameters [71] are thoroughly analyzed by way of several experiments (Chapter 4). Then, a solution is finally designed (Chapter 5) to take the depth map acquired by the camera at a certain frame rate, along with the amplitude image, stack them together to feed the *SHARP-Net* that produces the denoised point cloud, which is further taken by one of the chosen registration methods to be in real-time aligned with a ground truth reference.

The last gift the company provides me with, besides the ToF PicoFlexx camera and a Coral USB accelerator [35], which is an Edge TPU co-processor that could be connected to the local system in order to achieve high-speed interference for deep neural networks, is a 3D printing of an MTG-I satellite model [29]. After having sanded it and painted it white to make it as less reflective as possible (so to not degrade the performance of the ToF sensor), it is placed in the middle of a semi-empty room, suspended, to be as far as feasible from any existing background. Extensive experiments demonstrate not only that, while being a cheap, light and very small sensor, PicoFlexx's resolution is still good enough to provide interesting point clouds to be studied. Denoising methods (*SHARP-Net* [27] and the proposed variants) are able to effectively remove *MPI* noise and improve the depth map estimation, as demonstrated by several metrics. Lastly, and most importantly, FMR [47] learning-based registration model has proven itself trustworthy, due to its flawless accuracy in the alignment, regardless of the initial pose, and the extremely fast performance. The designed framework thus offers an interesting new range of possibilities, that mated with the increasing precision and output richness of ToF cameras could indeed enable efficient and light embedded solutions for robotic sensing.
These outcomes make this work the first ever proposing a such detailed survey about state-of-the-art methods both for *MPI* and shot denoising of ToF cameras and for point cloud registration, and the first ever applying such approaches using a PicoFlexx camera in avionics applications, surely opening the road for further research on this topic.

Chapter 2

ToF Cameras

In the domain of mobile robotic applications, 3D Time-of-Flight (ToF) cameras have recently represented a new fundamental, enormous potential. In order to address the problem of sensing the spatial properties of the environment in the autonomous robotics field, according to [65] three main different techniques can be taken into account, each of which has its pros and cons: CCD- or CMOS-camera based, laser scanner or lately 3D ToF camera based.

CCD or CMOS Camera. For what concerns the first group, the mainly used approaches for 3D robot vision are based on stereo cameras. They both can't provide reliable navigation or mapping information in real-time, while also being difficult to control in real world environments with changing lighting conditions, as all the passive visual sensors do.

Laser Scanner. 3D laser scanner are another class of techniques quite employed today. They present several advantages, like the high range, high accuracy and high reliability, but they find difficulties in detecting environment dynamics in three dimensions. Indeed, besides the high costs, the huge dimensions and the tremendous power consumption, they also face the problem of a low performing sampling rate.

ToF Camera. Lastly, a very new and promising operating procedure is the one using the already introduced 3D cameras based on the Photonic Mixer Device (PMD) technology. Nowadays, 3D cameras with a resolution of about 20000 pixels and a frame rate over 40 frames per second (*fps*) are at everyone's disposal. The high data rate and the small dimensions, together with a very low weight, make them suitable sensors for mobile robotics applications. Similarly to passive visual sensors though, these cameras are heavily affected by changing lighting conditions: this leads to imprecise data and completely inaccurate measurements. These errors are caused by the physical properties of the sensor, but also -and mostly- by environmental conditions. The latter are of paramount importance in the robotics field since the real world is unpredictable, therefore it is obvious that without real-time environment adaptation algorithms and data pre-processing schemes this kind of sensor is not appropriate for the robotic tasks taken into account. In fact, as shown in [64], external interfering factors (such as sunlight) or scene configurations (i.e., distance, orientation, density and reflectivity of the materials) could greatly change the way in which the same scene is perceived from the camera, thus implying large variations in the measurements from different point of views.

2.1 Taxonomy of ToF Cameras Errors

As authors of [64] illustrate, the potential of ToF cameras is limited by several errors:

- **Systematic Errors:** these errors are predictable and manageable by calibration since they systematically occur. They can be divided into three groups:

  1. *Distance-related errors:* the measurement principle relies on the idea that the emitted light is sinusoidal, but this is only an approximation of the real world.
  2. *Amplitude-related errors:* they are caused by the pixel's electronic components' non-linearities, and they give rise to a not negligible problem, which is the fact that the measured distance varies according to the object reflectivity.
  3. *Fixed pattern phase noise:* due to the pixels' connections on the sensor chip, which happen in series, each of these pixels is triggered depending on its position in the chip. This means that the measurement offset will be higher for those pixels that are situated far enough from the signal generator.

- **Non-Systematic Errors:** these errors are inherently dependent on the measurement principle, thus they are much more difficult to correct. There are three significant non-systematic errors:

  1. *Bad signal-to-noise ratio:* this error causes a distortion in the measurement, and it is highly arduous to stamp out. One way could be that of thoroughly enlarging the exposure time in order to amplify the illumination, but also the usage of an intelligent amplitude filtering could be a solution.
  2. *MPI (Multiple Path Interference):* due to inter-reflections in the observed scene, the remitted near-infrared (NIR) signal may be a superposition of another one that has moved along different distances. This interference has the side effect of making edges and corners smoother, and easily obstructing shapes.
  3. *Light scattering:* this third and last non-systematic error takes place in the ToF camera's lenses and it leads to the fact that near bright objects may superpose the surrounding ones, which for this reason seem to be nearer. It is worth noting that the latter two effects are unforeseeable since the scene's configuration is unspecified *a priori*.

2.2 Systematic Errors Correction

2.2.1 Photonic Mixer Device (PMD)

As particularly shown in [33], ToF cameras use the principle of modulation interferometry, which is a measurement technique making use of wave's interference phenomena. Specifically, the authors of the paper explain that an illumination module connected to the camera produces and discharges a near-infrared (NIR) light $g(t)$ (indicated in red in Fig. 2.1), that is a sinusoidal signal modulated with a frequency ω which floodlights the observed 3D scene:

$$g(t) = cos(\omega t) \tag{2.1}$$

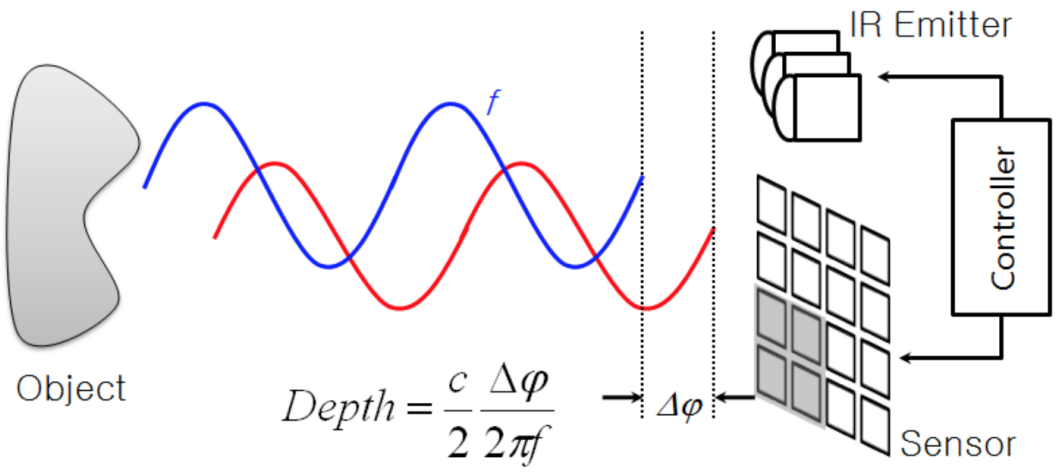


Figure 2.1: **"The principle of ToF depth camera"** (taken from [43]). The phase delay $\Delta\varphi$ between the emitted and the reflected near-infrared (NIR) signals is computed in order to measure the distance from each sensor pixel to the observed objects in the scene.

The diffusely remitted light $s(t)$ (shown in blue in Fig. 2.1) carries the distance information in terms of phase delay $\Delta\varphi$ with respect to the emitted signal $g(t)$.
Moreover, this remitted signal is also identified with the amplitude of remission a and with an unmodulated constant component k coming from the back-light:

$$s(t) = k + a\cos(\omega t + \Delta\varphi) \tag{2.2}$$

The phase delay $\Delta\varphi$ between the signals $g(t)$ and $s(t)$ can be estimated using the so-called 4-phase-algorithm, accurately described in [43]. In practice, the phase difference is calculated starting from the relationship between four different electric charge values, as depicted in Fig. 2.2. The four phase control signals have phase delays $\Delta\varphi = \frac{\pi}{2}$ from each other, and they determine the collection of electrons from the incoming signal. According to [33], this means that the correlation $C(\tau_p)$ between the emitted and the remitted signals is computed for the four internal phase delays $\tau_p = p\frac{\pi}{2}$, where $p = 1,2,3,4$:

$$C(\tau) = (s * g)(\tau) = h + \frac{a}{2}\cos(\omega\tau + \Delta\varphi) \tag{2.3}$$

The four resulting electric charge correlation values $Q_p = C(\tau_p)$ are thus brought into being in order to compute the phase delay $\Delta\varphi$ as:

$$\Delta\varphi = \arctan\left(\frac{Q_3 - Q_4}{Q_1 - Q_2}\right) \tag{2.4}$$

where Q_1, Q_2, Q_3, Q_4 represent the volume of electric charge for the control signals C_1, C_2, C_3, C_4 respectively, as shown in Fig. 2.2.

Let $c \simeq 3 \times 10^8 m/s$ be the speed of light and f the frequency of modulation. At this point, the corresponding distance d for a single pixel can thus be calculated as:

$$d = \frac{c}{2f}\frac{\Delta\varphi}{2\pi} \tag{2.5}$$

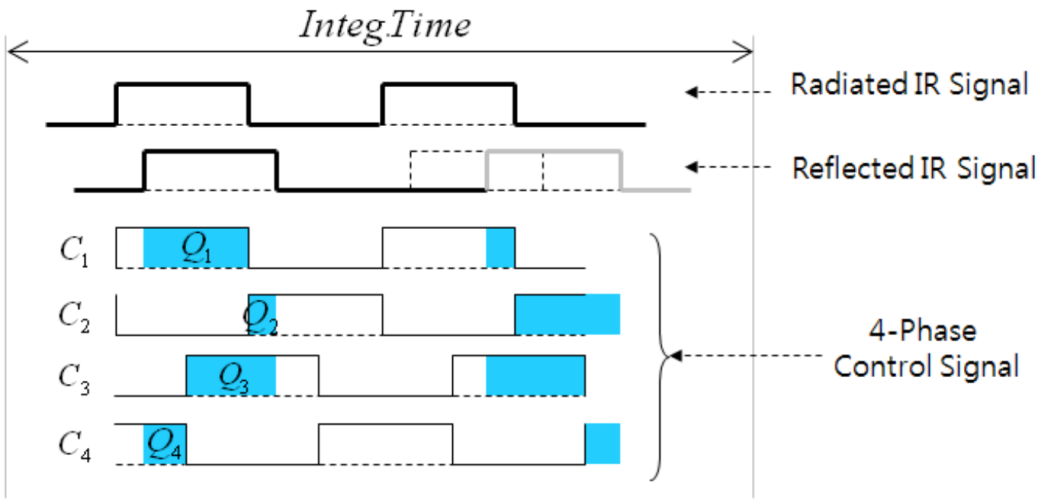


Figure 2.2: **"Phase Delay"** (taken from [43]). The depth measurement is computed by calculating the phase delay $\Delta\varphi$ between the emitted and the reflected NIR signals. $Q1$ to $Q4$ indicate the volume of electric charge for control signals $C1$ to $C4$, respectively.

as well as the amplitude a:

$$a = \frac{\sqrt{(Q_3 - Q_4)^2 - (Q_1 - Q_2)^2}}{2} \tag{2.6}$$

and, as claimed by the authors of [104], also the intensity of the incident infrared light I:

$$I = \frac{Q_1 + Q_2 + Q_3 + Q_4}{4} \tag{2.7}$$

After the measurement of the depth value, using the ToF camera's intrinsic parameters it is further possible to obtain the target object's 3D coordinates based on the following equations, described in [104]:

$$\begin{aligned} z &= r \cdot \frac{F}{\sqrt{F^2 + (X_c d_x)^2 + (Y_c d_y)^2}} \\ x &= z \cdot \frac{X_c d_x}{F} \qquad y = z \cdot \frac{Y_c d_y}{F} \end{aligned} \tag{2.8}$$

where F is the focal length of the camera, d_x and d_y represent the actual length of a pixel in the x and y directions, respectively, while X_c and Y_c indicate the normalized coordinates of the pixels with respect to the optical center.

One point that still has to be made is that, as relatively new sensors, ToF cameras are still "raw" devices with many aspects which need to be improved. Among these one can surely find the low resolution of the depth maps and the system errors which, along with other factors such as spatial depth discontinuity, motion blur and so on, lead to a significant inaccuracy in the depth data measurements.
For all these reasons a camera calibration is thus more than needed, and in particular for a PMD camera the full calibration process is fragmented into two separate calibration steps, as stated in [58]:

1. *Lateral Calibration*, coming directly from traditional 2D sensors

2. *Depth Calibration* of the distance measuring procedure

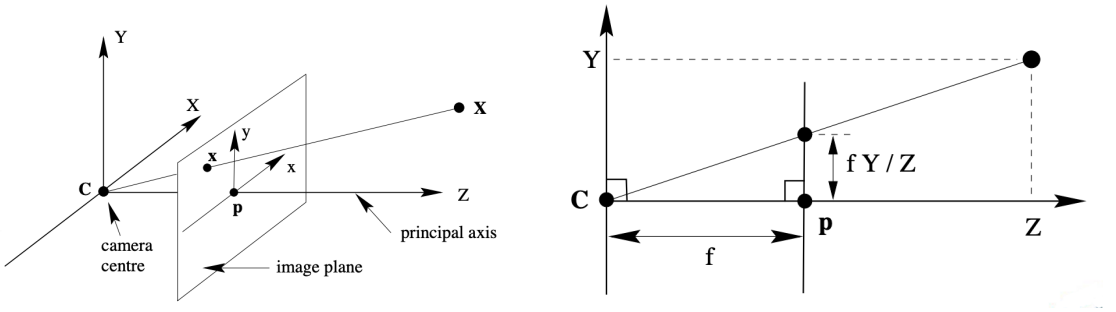
One can think of a lot of reasons behind the distance measurements' errors in the PMD cameras. Perhaps the main cause is a systematic error due to the correlation function's demodulation, but a part from that there are several other agents, such as the IR reflectivity and the orientation of the observed objects. Indeed, the latter introduced aspect may lead to an insufficient incident light to a specific pixel, thus obtaining a completely incorrect distance measurement. Furthermore, the authors of [58] also noticed that the demodulation distance is almost analogous inside the solid angles associated to a PMD pixel, but in the real world one could observe several distances inside a solid angle, thus steering to a superimposed reflected light. This has of course the side effect of reducing the amplitude of the signal, as well as introducing a non negligible phase shift.
The accuracy of the distance measure provided by the PMD camera can only be computed by taking into account the amplitude a of the correlation function, which comprises the saturation, the distance homogeneity and the object reflectivity and orientation (see Eq. 2.6).
For all these reasons, an accurate calibration model is needed.

2.2.2 Lateral Calibration

In order to understand the concepts behind the lateral camera calibration, it is of paramount importance to start from the consideration that what one observes through the camera is a 3D world, that is a collection of objects in space. This 3D world is then mapped into a 2D image, as shown in [89]'s slides.
Depending on the type of camera that one is using and the position of the camera that is observing the 3D world, the way in which the 3D world is mapped into the image can obviously change. So in order to elaborate the information that is acquired, that is in order to get meaningful information about the 3D world that is reproduced in a 2D image, one has first of all to understand how this mapping between 3D and 2D takes place. Therefore it is necessary to be able to build a model of the camera, and in a second moment to derive the parameters that define the behaviour of this model itself. For this purpose, the most basic model that can be considered (and that will be taken into account in this thesis work) is the Pinhole model, which is thoroughly explained in [89].

The starting point of this model is the consideration of the central projection of points in space into a plane, where the centre of projection is the origin of an Euclidean coordinate system and the plane $Z = f$ is called *image plane* or *focal plane*. The *image plane*, as shown in Fig. 2.3a, is a representation of the 2D plane that will be used to obtain a 2D picture of the 3D world that the camera is looking at. The image point X that is under observation in the 3D world will be reproduced by x in the 2D *image plane* that is taken into account. This *image plane* is placed at a certain distance from the camera centre: such distance is the focal length f.

A point in space $X = (X,Y,Z)^T$ is thus mapped to the point $(fX/Z, fY/Z, f)^T$ on the *image plane* where a line joining the point X to the centre of projection meets the *image plane* itself. The mapping from Euclidean 3D space to Euclidean 2D space, by ignoring



(a) *Mapping of a 3D point into a 2D one.*

(b) *Mapping from Euclidean 3-Space to Euclidean 2-Space (ignoring the third image coordinate).*

Figure 2.3: **"Mapping from 3D to 2D space"** (taken from [89]). Mapping of a point from a 3D space into a 2D one, considering three (a) and two (b) axes respectively.

the third image coordinate, is defined as:

$$(X,Y,Z)^T \longmapsto (fX/Z, fY/Z)^T \tag{2.9}$$

As it is possible to notice, due to the projection of the 3D image onto the focal *image plane*, that is at distance f along the z axis, the focal length f can be omitted.

At this point, the mapping previously introduced can also be written using homogeneous coordinates according to:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.10}$$

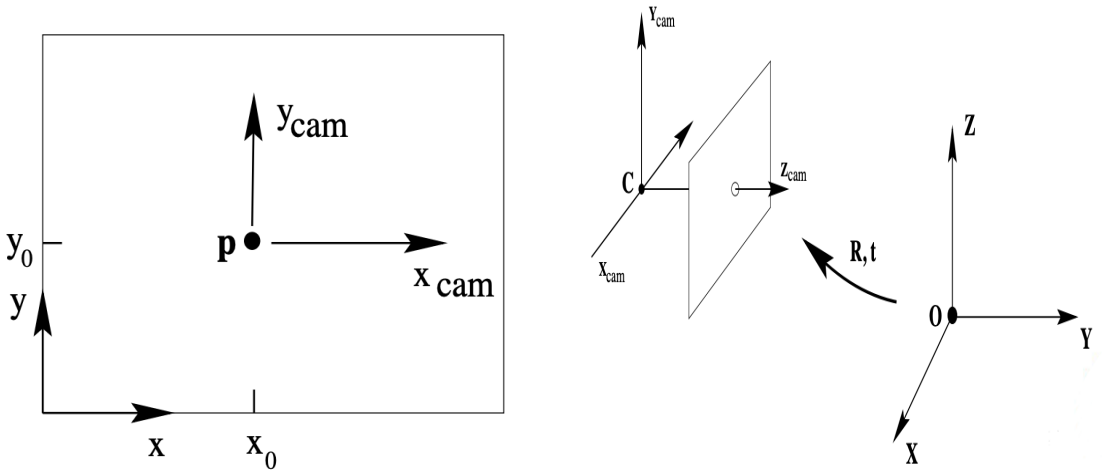
Let now $(p_x, p_y)^T$ be the coordinates of the image center p in the camera.
Thus, the mapping described in Eq. 2.9 can be expressed as follows:

$$(X,Y,Z)^T \longrightarrow (fX/Z + p_x, fY/Z + p_y)^T \tag{2.11}$$

In matrix form, it is finally possible to formulate:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longrightarrow \mathtt{X} = \mathtt{K}[\mathtt{I}|0]\mathtt{X}_{cam} \tag{2.12}$$

where K is the *camera calibration matrix* which carries the position of the camera centre and the focal length of the camera itself. These are information *intrinsically* related to the properties of the camera itself: whichever is the position of the camera, the focal length is not going to change, nor will the position of its centre.
$(X,Y,Z,1)^T$, denoted as $\mathtt{X}_{cam}$ in Eq. 2.12, emphasizes the assumption earlier drew up, which is that of considering the camera located at the origin of an Euclidean coordinate



(a) *Camera Coordinate Frame.* (b) *World to Camera Coordinate Frame.*

Figure 2.4: **"World Coordinate Frame to Camera Coordinate Frame"** (taken from [89]). Transition from *World Coordinate Frame* to *Camera Coordinate Frame* through rotation and translation.

system, with its principal axis pointing straight down the Z-axis. Thus, the $\mathtt{X}_{cam}$ point itself is expressed in this coordinate system, that may be called the *Camera Coordinate Frame* [Fig. 2.4a].

It is therefore possible to map the points in the real world (belonging to *World Coordinate Frame*) to the ones belonging to the *Camera Coordinate Frame* by applying translation and rotation, as it is possible to see by looking at Fig. 2.4b.
Hence, it is possible to state that:

$$\tilde{\mathtt{X}}_{cam} = \begin{bmatrix} \mathtt{R} & -\mathtt{R}\tilde{\mathtt{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} \mathtt{R} & -\mathtt{R}\tilde{\mathtt{C}} \\ 0 & 1 \end{bmatrix} \tilde{\mathtt{X}} \tag{2.13}$$

where:

$$\begin{cases} \tilde{\mathtt{X}} & \text{is a 3D vector reporting the coordinates of a point in } \textit{World Coordinate Frame} \\ \tilde{\mathtt{X}}_{cam} & \text{is the same point in } \textit{Camera Coordinate Frame} \\ \tilde{\mathtt{C}} & \text{represents the coordinates of the camera centre in } \textit{World Coordinate Frame} \\ \mathtt{R} & \text{is a } 3 \times 3 \text{ rotation matrix showing the } \textit{Camera Coordinate Frame}\text{'s orientation} \end{cases}$$

Putting everything together, it is feasible to obtain an equation that allows the mapping into the 2D space of an object that is a point in the 3D space, taking into consideration the rotation and translation of the reference system when passing from the 3D to the 2D world and the camera properties. This equation is described as follows:

$$x = \mathtt{K}\mathtt{R}[\mathtt{I} \,|\, -\tilde{\mathtt{C}}]\,\mathtt{X} \tag{2.14}$$

This is the model of the camera, named *Pinhole Camera Model*:

$$\begin{aligned} \mathtt{P} &= \mathtt{KR}[\mathtt{I} \mid -\tilde{\mathtt{C}}] \\ \mathtt{P} &= \mathtt{K}[\mathtt{R} \mid \mathtt{t}] \qquad \mathtt{t} = -\mathtt{R}\tilde{\mathtt{C}} \end{aligned} \tag{2.15}$$

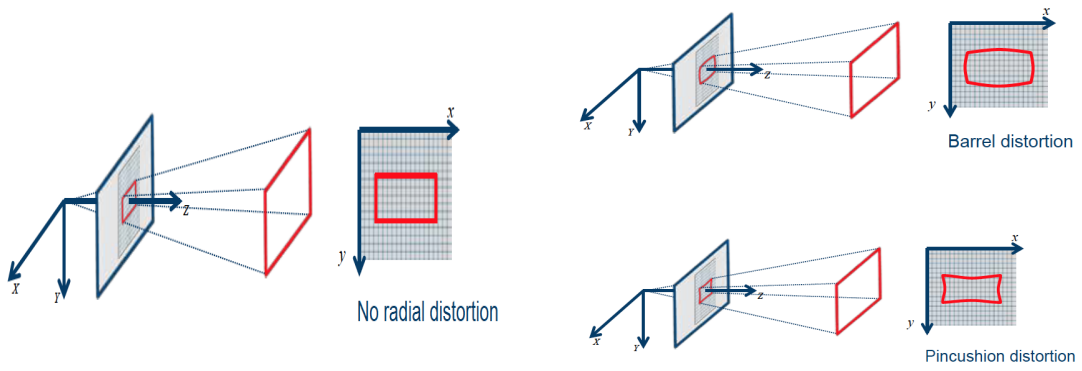
K is referred to as the *intrinsic (internal) camera parameters*: it is a constant matrix that contains information about the focal length of the camera and the centre of the image inside the camera itself. For a CCD (Charged-Coupled Device) it holds that:

$$\mathtt{K} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.16}$$

where $x_0 = m_x p_x$ and $y_0 = m_y p_y$, given m_x and m_y the number of pixels along the x and y axes, respectively. a_x is finally defined as $f m_x$, while a_y is equal to $f m_y$.

R|t is instead called *extrinsic (external) camera parameters* and considers the position and the orientation of the camera: the very same camera in a certain application (that is a certain value of K) will produce different values of R and t depending on where it is placed and how it is oriented.

Unfortunately, the *Pinhole* model is ideal, due to the fact that it doesn't take into account any distortion. If one is dealing with an ideal camera, she may end up with a situation as the one depicted in Fig. 2.5a: the starting point is the observation of a certain object (the red rectangle), whose image is captured from the camera. In the ideal situation, one gets exactly what it is possible to see in the figure: every straight line is precisely reproduced as a straight line, without distortions of any kind.
But in the real case, due to the geometry of the camera lens used to take the image, distortions happen, so there are some alterations to the reproduction of the points in the image that the camera is observing. Fig. 2.5b shows for example, in the upper part, a straight line that is not reproduced as a straight line, but as a curved one. When this kind of defects happen, it is possible to talk about a *barrel distortion*.



(a) *Camera with no Radial Distortion.* (b) *Camera with Barrel and Pincushion Distortion.*

Figure 2.5: **"Camera Distortions"** (taken from [89]). Camera with no Radial Distortion (a), and with Barrel and Pincushion Distortion (b).

The lower part of Fig. 2.5b depicts instead another possible effect, where a straight line is reproduced as a bent one: in cases like this it is possible to refer to this phenomenon as a *pincushion distortion*.

Thus, also these defects have to be taken into account in order to properly elaborate the image. This topic is related to the identification of the so-called *radial distortion*: once the parameters k_1 and k_2 that define the *radial distortion* are known, it is possible to apply a further refinement of the mapping, thanks to the following equations:

$$\begin{aligned} \hat{x} &= x(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \\ \hat{y} &= y(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \end{aligned} \tag{2.17}$$

The lateral camera calibration therefore consists in calculating the internal and the external parameters for the camera taken into account, including the distortion parameters (i.e., focal length f, real image center (c_x, c_y) and radial lens distortion).
While several approaches have been investigated through the years for performing a lateral calibration scheme, like the procedure presented in [58], which sticks to the calibration module contained in Intel's OpenCV library [16], in this thesis work none of them are applied. Indeed, the ToF camera provided for this research is a PMD Camboard PicoFlexx [73] sensor, which is already globally pre-calibrated, as explained more in detail in the dedicated Section in Chapter 4.

2.2.3 Depth Calibration

After the lateral calibration, also the distance information needs to be calibrated. The following subsection describes the depth calibration model thoroughly reported in [58], together with the data analysis which is at the basis of the calibration model itself.

To counterbalance the distance deviation, the depth calibration can be thought as composed by two separated steps, as shown in Fig. 2.6:

1. Global adjustment for the entire image

2. Local per-pixel (pre-)adaption, used to procure slightly more precise results for the previous phase of global adjustment, and to counteract the overall remaining deviation.

Global Adjustment. The idea behind the global adjustment procedure is that of using a high complexity function in order to actually obtain a simpler adjustment for the per-pixel calibration step in a second moment. The latter uses indeed linear adjustment, being therefore much more efficient in terms of the storage of the overall number of calibration parameters. The authors suggest to use uniform, cubic B-splines functions for the correction, due to the fact that they manifest a good local control. A basis spline (B-spline), as reported in [94], is a spline function that has the least possible support as regards predefined degree and smoothness. Thus, given a certain degree, any spline function can be written as a linear combination of B-splines of exactly that degree. Furthermore, the evaluation of these functions always need a constant number of operations, which is of paramount importance for real-time calibration tasks.

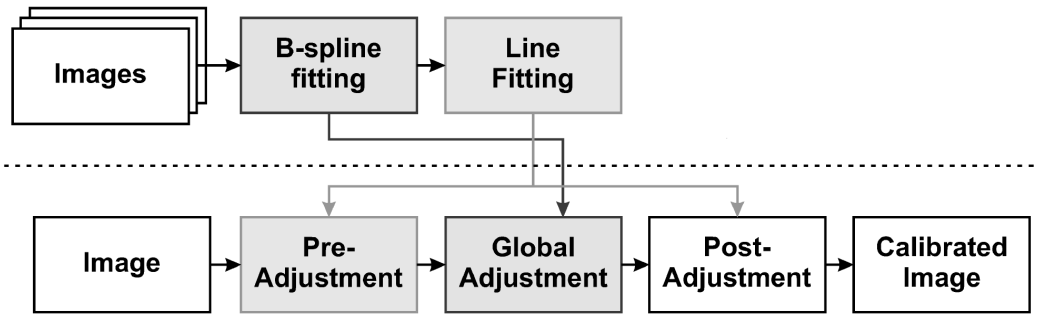


Figure 2.6: **"Depth Calibration Process"** (taken from [58]). The depth calibration process, including the global adjustment and the local per-pixel per-adaption.

An iterative least-square fitting for B-spline curves is thus performed to single out the optimal number of control points. This procedure is described according to:

$$b_{glob}(d) = \sum_{l=0}^{m} c_l \cdot B_l^3(d) \tag{2.18}$$

The control points m are progressively escalated, until the approximation error lies within a predefined threshold or m itself goes beyond a maximum value.
The global adjustment of the distance $d(x,y,k)$, achievable through to the fitted B-spline curve b_{glob}, determined according to the previous Eq. 2.18, is simply defined as:

$$d_{glob}(x,y,k) = d(x,y,k) - b_{glob}(d(x,y,k)) \tag{2.19}$$

Per-Pixel Distance Calibration. An additional improvement of the results obtained with the global adjustment just described could be accomplished by further considering individual pixel inaccuracies. The calibration process for polar $[d^p(x,y,k)]$ and cartesian $[d^c(x,y,k)]$ coordinates differ for what concerns the coordinate transformation and the distance adjustment. Specifically, in the latter case the process of B-spline fitting could be made more comprehensible by employing average values with respect to acknowledged plane distances. Thus, the average mean $d_{avg}(k)$ of all per-pixel distances $d(x,y,k)$ could be applied for B-spline fitting to lower the sample points' number. Since the B-spline fitting is associated to an average deviation computed across the actual $d(x,y,k)$ and the expected $d_{ref}(x,y,k)$ distances, assessing the B-spline at a per-pixel distance may conduct to errors in the global adjustment:

$$b(d(x,y,k)) \approx d_{avg}(k) - d_{ref}(x,y,k) \tag{2.20}$$

With the aim of reducing this "artificially-introduced" error, a per-pixel pre-adjustment is then employed: the goal is that of obtaining a pixel's distance that better harmonizes with the average distance d_{avg} considering the totality of distance images k. This objective can be achieved by fitting a line $l_{x,y}$ for each (x,y) pixel, through the following minimization:

$$\sum_k \| d(x,y,k) - d_{avg}(d(x,y,k)) \|^2 \tag{2.21}$$

Considering cartesian coordinates, the average deviation d^c_{avg} is directly specified according to:

$$d^c_{avg} = \frac{1}{n} \sum_{(x,y)} d^c(x,y,k) \tag{2.22}$$

where n is the number of pixels (x,y) that are taken into account.
In the case of polar coordinates instead, such an image averaging is not applicable. Thus, the authors of [58] introduce the following dependence between $d^p_{avg}(k)$ and the B-spline $b_{glob}(d)$:

$$d^p_{avg}(k) \approx b_{glob}(d(x,y,k)) + d^p_{ref}(x,y,k) \tag{2.23}$$

where $d(x,y,k)$ is the actual distance and $d^p_{ref}(x,y,k)$ the expected one.
Therefore, the overall distance calibration is given by:

$$d_c(x,y,k) = b_{glob}(d(x,y,k)) - l_{x,y}(d(x,y,k)) \tag{2.24}$$

If some distance deviations endure, a second line fitting, equivalent to the one performed for the pre-adjustment, can be further implemented. In this case though, as the authors Lindner and Kolb suggest, the differences between the global corrected distances and the reference plane must be taken into account.

2.2.4 An improved calibration approach

The traditional calibration methods are everything but straightforward and efficient, since there exist multiple error sources. This is the reason why [90] proposes a newer and advanced calibration method based on the electrical analog delay.
Fig. 2.7 depicts the comprehensive calibration model, in which grayscale and depth images are calibrated in conjunction.
In the following some details are provided for each of the different phases of the process.

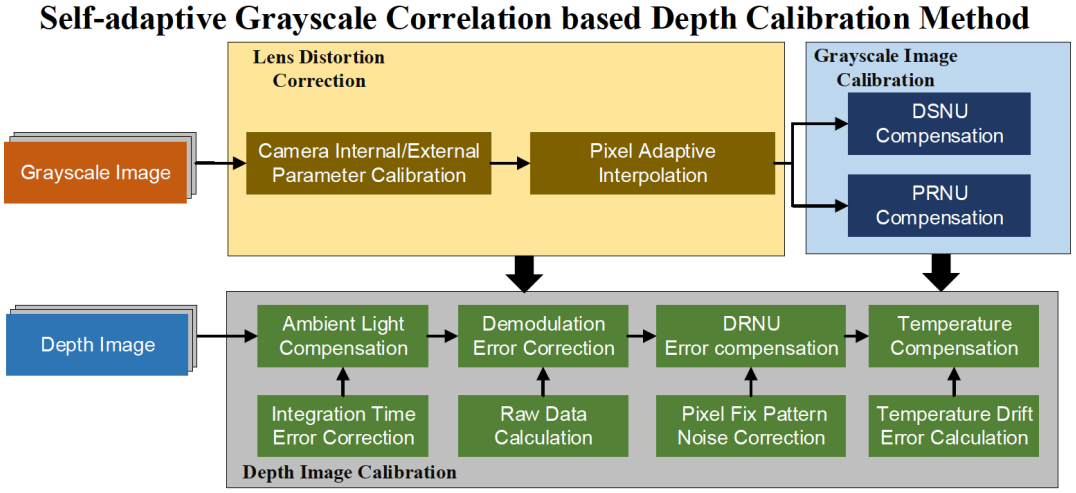


Figure 2.7: **"Self-adaptive Grayscale Correlation-based Depth Calibration Method"** (taken from [90]). The comprehensive improved calibration model process, in which grayscale and depth images are jointly calibrated.

Lens Distortion Correction. The internal and the external parameters of the camera are attained following Zhang's calibration procedure described in [105].

Grayscale Image Calibration. Generally speaking, the grayscale image is susceptible to *DSNU* (Dark Signal Non-Uniformity) and *PRNU* (Photo Response Non-Uniformity). The former indicates the dissimilarities of gray values between pixels took over dark conditions while the latter stands for the divergences of gray values between pixels acquired at ordinary condition.
The grayscale image calibration approach is therefore described as follows:

1. Put the integration time to 0 μs in order to replicate a dark condition environment. Accumulate $N = 100$ frames of grayscale images and compute the mean value.
2. Change the integration time to imitate different ambient light conditions. Gather $N = 100$ frames of grayscale images with amplitudes of $10\%, 30\%, 50\%$ and 80%, and finally calculate the average values.
3. Under different ambient circumstances compute the spatial variance, which is a measure of the spatial non-uniformity useful to estimate *DSNU* and *PRNU*.
4. Compute the rectifying values of *DSNU* and *PRNU*.
5. Finally compute grayscale compensation of pixel (x, y).

Depth Image Calibration. The PMD sensor is able to jointly acquire grayscale and depth images due to its particular configuration. Thus, Wang et al. propose in their paper a self-adaptive grayscale correlation-based depth calibration method (*SA-GCDCM*). In the newly introduced procedure, the parameters of the depth calibration are estimated employing the grayscale image. Specifically, three main aspects have to be taken into account when performing depth calibration:

1. *Ambient Light Compensation:* the key point is that of attenuating the ambient light's influence in the sampling stage through the introduction of an ambient light correction factor K_{AL}, that can be precisely managed by operating on the integration time.
2. *Demodulation Error Correction:* the modulated (emitted) and the demodulated (received) continuous wave signals are generally considered as sinusoidal waves. Nonetheless, the actual received signal is closer to a rectangular wave due to the constraints of the generator bandwidth. This behaviour inevitably leads to an extra error, which is corrected following the approach described in [103].
3. *DRNU (Distance Response Non-Uniformity) Error and Temperature Compensation:* starting from the camera specifics, linear interpolation and ad-hoc parameters computation are performed to nullify the *DRNU* error and the temperature, respectively.

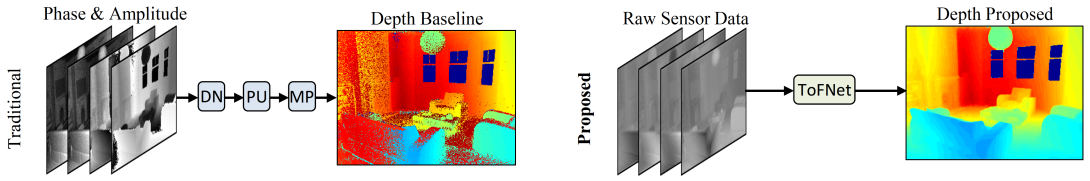
With the aim of improving the range accuracy of the PMD cameras and obtaining uniform and steady grayscale image, thus getting rid of the impact of *DSNU* and *PRNU*, the cited paper [90] proposes to integrate *SA-GCDCM* with demodulation error correction. In this way, it is possible to compensate also the *DRNU* and the temperature errors, achieving an overall multi-scene adaptive calibration model which works based on multiple factors. A sequence of several experiments performed by the authors verify the effectiveness, the accuracy and the flexibility of the method.

2.3 Non-Systematic Errors Correction

As already mentioned, ToF cameras are also subjected to a number of non-systematic errors that cannot be corrected through normal (lateral and depth) calibration techniques. Specifically, as accurately described in [85], ToF cameras capture depth measurements by illuminating a scene with a periodic amplitude-modulated signal, which is reflected back to the camera following direct and indirect light paths. The phase shift between incident and illumination signals is then estimated. To calculate depth from these raw phase estimations though, several challenging reconstruction problems must be addressed first. When a single reflector is present in the observed scene, the phase estimates are able to uniquely code depth up till an integer phase wrapping. This is tackled by phase unwrapping methods, as shown in [43]. If instead the scene is globally illuminated, multiple light paths stand in the way of direct and indirect paths, providing a route to a phenomenon called Multiple Path Interference (*MPI*), that provokes the depth maps' distortion. Lastly, raw ToF measurements are heavily influenced by the noise caused by the low absorption depth of the IR modulation, along with yet undeveloped sensor technology ([53]) if compared to RGB CMOS image sensors.

Historically, the three reconstruction problems just presented, namely phase unwrapping, *MPI* reduction and denoising, are tackled in a pipeline architecture where each step disentangles an individual sub-problem alone. This design surely makes *divide-and-conquer* algorithms smoother, but at the same time it disregards the coupling between individual sub-modules. In this way it infuses cumulative error and information loss in the reconstruction pipeline. By way of illustration, [85]'s authors highlight that traditional multi-frequency unwrapping methods are extremely inaccurate when *MPI* or noise are present, presenting notable unwrapping errors and consequently inaccurate shape recovery.

This is the reason why, recently, instead of building a reconstruction pipeline or depending on auxiliary hardware, new data-driven approaches were introduced. These models, using the unrivaled strength of neural networks, are able to generate a depth map directly from the raw modulated exposures of the ToF camera (see Fig. 2.8). From now on, a list of the state-of-the-art methods is presented, describing their architectures along with their pros and cons. From all these approaches, at the end two of them, e.g. the ones that provide a good compromise in terms of complexity, real-time performance and accuracy (along with several other metrics) are selected in order to be reproduced and analyzed in detail.



(a) *Traditional pipeline approach.* (b) *NN (Neural Network)-based approach.*

Figure 2.8: **"Comparison between traditional and NN-based approaches for non-systematic errors correction"** (taken from [85]). Traditional pipelines apply a sequence of techniques for depth map generation, i.e., denoising (DN), phase unwrapping (PU) and multipath correction (MP). Deep CNN approaches predict instead the scene depth directly from ToF camera's raw measurements.

2.3.1 Deep End-to-End Time-of-Flight Imaging

As far as I know, [85] is one of the first in introducing a deep learning attempt to correct ToF cameras' non-systematic errors. Diverging from the conventional pipeline model, the authors propose a deep neural network which redeems scene depth from raw dual-frequency ToF measurements. They design *ToFNet*, an encoder-decoder network architecture with skip connections [78] and ResNet [45] bottleneck layers, as shown in Fig. 2.9. In particular, the network they present takes as input two modulated exposures $[B_{\omega_i,\psi_j}]$, with $i,j = [1,2]$, in order to output a phase-unwrapped, *MPI*-compensated depth image, which is then further converted to a depth map using calibrated camera intrinsic parameters.

The encoder (F1_1 to D2 in Fig. 2.9 below) of the generator G squeezes the input up to $1/4$ of its original resolution, and it creates feature maps with an increasing receptive field. The ResNet blocks at the tailback, while maintaining the same number of features, makes the network able to reestablish a finer depth after upsampling.
As it is possible to notice by looking again at the same figure, the authors of [85] propose symmetrical skip connections between F1_2-U2 and F2-U1 by element-wise addition.
The PatchGAN discriminator network D [50] is instead composed of three down convolutional layers with Leaky ReLU as activation function. Its goal is that of classifying whether each patch in a G's prediction is real or not. This discriminator is convolutionally run across the image itself, averaging all responses to provide the ultimate output of D.

Moreover, Shuochen et al. suggest to apply pixel-wise normalization to the depth inputs, along with their corresponding amplitudes, instead of taking for granted the network capability to learn amplitude-invariant features. By doing this, they manage to enhance the model's robustness to illumination power and scene albedo, that is, its brightness, while reducing at the same time the training time.
One downside of the proposed normalization procedure is the fact that the input may incorporate outstandingly amplified noise in regions where reflectivity is low or distances are too large. This is mainly the reason why the authors present also an edge-aware smoothness term to control the unused amplitude information, by giving the amplitude maps as input to the total variation (*TV*) regularization layer.

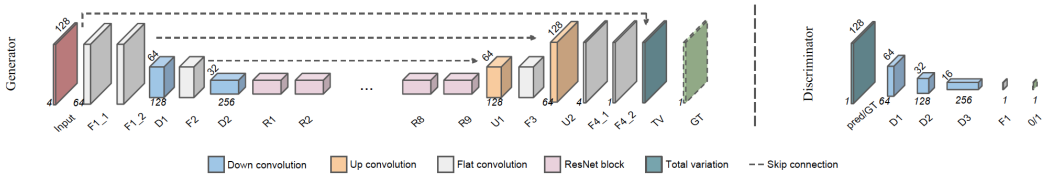


Figure 2.9: **"The ToFNet Architecture"** (taken from [85]). It consists of a symmetrically skip-connected encoder-decoder generator network G and a patchGAN discriminator network D.

Due to the huge difference in image statistics between RGB and depth image data, conventional ℓ_1/ℓ_2-norm pixel losses that behave well in RGB generation tasks do not perform equally well in depth reconstruction. Consequently, the authors of [85] propose a

domain-specific loss function tailored to distance image statistics, that is described as:

$$\mathscr{L}_{total} = \mathscr{L}_{L_1} + \lambda_s \mathscr{L}_{smooth} + \lambda_a \mathscr{L}_{adv} \tag{2.25}$$

where:

- L_1 **Loss** is the mean absolute error (*MAE*) between the generator's output d and the ground truth distance $\tilde{d}$.
- $\mathscr{L}_{smooth}$ **Depth Gradient Loss** is a total variation loss weighted by image gradients in an edge-aware fashion.
- $\mathscr{L}_{adv}$ **Adversarial Loss** is a patch-level conditional adversarial loss [108] minimizing the gap between output and target depths.

During the training phase, G and D are optimized alternatively, so that G progressively refines the generated depth trying to persuade D to take the result as correct (label 1), while on the other side D constantly improves at discerning correct and incorrect depth maps. More details in training and implementation can be found in the paper.
Due to the fact that large raw ToF datasets with ground truth depth are not available (at least, when the paper was published), the authors simulate synthetic measurements with known ground truth. In this way they were able to train the model, by synthesizing a sequence of transient images and then correlating the pixels with the frequency-dependent correlation matrix.

To validate the results, real data from a number of scenes are collected under both controlled and *in-the-wild* conditions. Experiments' outcomes show an improvement with respect to the previous conventional pipeline in terms of accuracy, even if the model is totally faulty when the measurement includes saturation, imperfect modeled materials, low reflectivity or finer geometric structures.
Despite the fact that the network is able to reach better and more stable performance than tradition techniques, it still lacks in robustness and it is not completely capable of working in real-time applications.

2.3.2 DeepToF: Off-the-Shelf Real-Time Correction of Multipath Interference in Time-of-Flight Imaging

The authors of [63] have a novel idea: since Multi-Path Inference (*MPI*) can be modelled as a spatially-varying convolution, it is thus possible to express the *MPI* compensation as a sequence of convolutions and deconvolutions in depth space, that is, *MPI* errors could be straightened out through the design of an accurate convolutional neural network (CNN).
Concretely, since correct and incorrect depth maps are slightly different one another but yet structurally similar, a convolutional autoencoder (CAE) could be used to tackle this problem, because of its capability to use the same input and output to learn hidden representations of lower-dimensional feature vectors through unsupervised learning. These lower-dimensional vectors allow to retain the input's relevant structural information and to remove errors, successfully returning the reestablished depth image.

Nonetheless, due to the fact that a labeled dataset of the correct size is not available for the training step, a straightforward CAE cannot be employed for this purpose. Thus, even

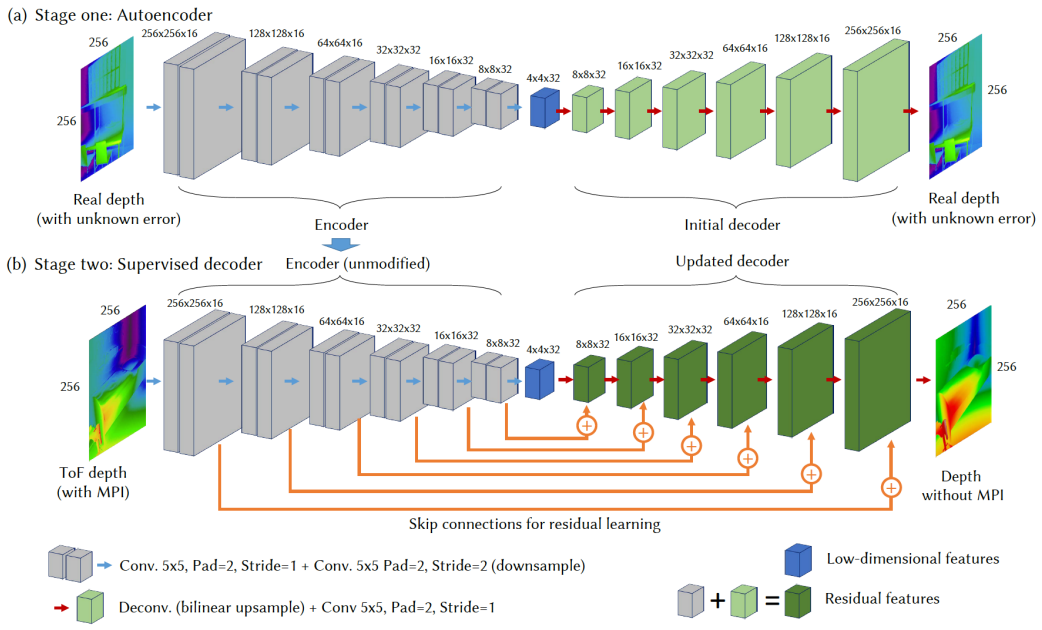


Figure 2.10: **"DeepToF Architecture in two-stage training process"** (taken from [63]). In the first, unsupervised training stage a convolutional autoencoder learns lower-dimensional depth representations, while in the second, supervised training stage both depths with and without *MPI* are taken as input to update the weights of a reconstruction decoder. The final DeepToF model is made by both the original encoder and the updated decoder, working as a regression network.

if real world ToF depth images are extensively accessible and ready to be used, measuring their ground truth value is definitely a non-trivial problem that has to be solved. Conversely, rendering images from synthetic scenes is tremendously time-consuming, without considering the fact that the obtained results would only cover a small-scale chunk of real world scene alterations.

This is the reason why the authors of [63] propose *DeepToF*, a two-step training scheme to infer the CNN from both unlabeled real depth images and labeled synthetic depth ones (both with and without *MPI*). Fig. 2.10 depicts an overview of the neural network approach they introduce, while the next paragraphs explain more in detail the novelties of their approach.

First Stage. As it is possible to notice by looking at the figure above, Marco et al. firstly train a convolutional autoencoder in an unsupervised fashion using a real dataset, which contains real depth images with unknown errors. A synthetic (smaller) dataset with *MPI* is instead used as validation set.
With this first stage the network is initialized and the encoder is made capable of producing lower-dimensional feature vectors of correct and incorrect depth maps.

Second Stage. Once the training of the encoder's parameters is terminated, its convolutional layers are kept frozen and the decoder is trained through supervised learning using the synthetic dataset. The incorrect depth maps with *MPI* are used as input, whereas the ground truth reference depths without *MPI* are introduced as output. Since the encoder

performs down-sampling operations to perceive features at multiple scale levels, symmetric skip connections are added so that detailed features of the encoding convolutions are blended together.
Lastly, being the difference between the input depth with *MPI* and reference depth output without *MPI* of about 12% on average, *MPI* is handled as a residue by implementing element-wise summations between the up-sampled features and the skipped ones.

More details on the implementation of the network and the training procedure can be retrieved in the cited paper [63].
One aspect that has to be highlighted is that the newly introduced model only takes depth maps as input, without the need for any other ToF data, i.e., phase images or pixel amplitudes information. This brings to a high robustness of the model itself, since the latter properties are exceedingly influenced by the peculiar ToF characteristics, thus they are highly volatile.

Results of the conducted experiments show that this network perform well in real-time, reaching visible improvements with respect to previous traditional approaches for *MPI* correction. Nonetheless, wiggling error caused by approximated sinusoidal waves is not examined and studied in the training dataset, and this leads to poor performance in several scenarios. Moreover, the authors take diffuse (or nearly diffuse) reflectance for granted. Thus, while behaving fine in some real world scenarios with more general reflectances, *DeepToF* presents some problems if very glossy materials are present and it is most likely to fail, as well as in the presence of objects that are very close to the camera (again due to the lack of relevant depth information for close distances in the training dataset). A better design of the datasets could therefore improve, in principle, the architecture's capabilities.

2.3.3 Tackling 3D ToF Artifacts Through Learning and the FLAT Dataset

With the aim of jointly addressing all the problems characterising the ToF camera sensing, i.e. dynamic scenes, *MPI* and shot noise, the authors of [39] propose a novel learning-based approach developed on a two-stage architecture. The model they introduce straightforwardly works with raw ToF measurements and outputs improved depth maps estimates. The first stage is composed by an encoder-decoder architecture (inspired by [30]) that weakens motion artifacts. Its goal is that of enlarging the amount of pixels whose depth can be accurately estimated, for the most part in the proximity of moving objects' boundaries. The second stage is instead based on a kernel predicting network [67] that enervates *MPI* and shot noise.

Specifically, the authors introduce a two-modules Deep Neural Network (DNN), whose architecture is shown in Fig. 2.11. The first module is called *MOM* (MOtion Module) whereas the second one is denoted as *MRM* (Multi-Reflection Module). The goal of the *MOM*, that is an encoder-decoder network highly influenced by FlowNet [30], is that of producing as output a map that aligns the raw channels measured by the ToF camera. This module is capable of taking in input nine misaligned channels and giving as output eight optical flows simultaneously. This is the biggest difference with respect the FlowNet model just cited.
MRM, the second module, is instead composed by a kernel-predicting network (KPN)

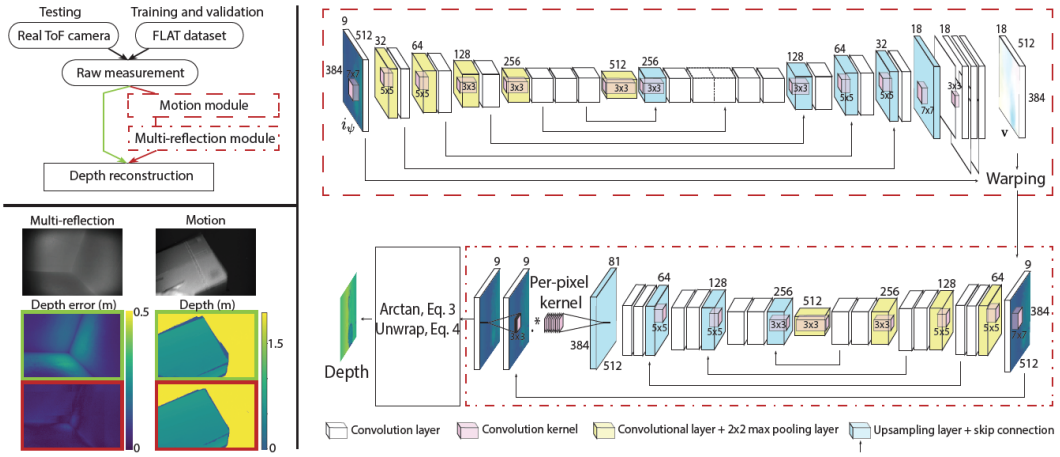


Figure 2.11: **"MOM + MRM Deep Neural Network Architecture"** (taken from [39]). The traditional ToF processing pipeline (green) and the framework proposed in this paper[39] (red). The lower left side of the image depicts artifacts generated by *MPI* and motion (green), which are highly decreased by the newly introduced architecture (red). On the right panel of the figure one can instead have a look at the proposed framework, with the motion (top) and the multi-reflection (bottom) modules.

that works on the compensation of shot noise, as greatly explained in the previous work of [67]. This module brings as output nine spatially varying kernels for each pixel. Every one of these kernels is locally convolved with the input raw measurements in order to finally output a raw estimate without shot and *MPI* noise on every channel.

Experiments illustrated by the authors have indeed shown that *MOM* helps in slightly increasing the depth accuracy, but its biggest advantage is the reduction of the undependable pixels. The introduction of this module can lessen the holes' existence in the reconstructed scene, to a large extent in the object neighborhoods. The presence of the *MRM* module additionally enlarges the density, while also lowering the bias in the depth error due to *MPI*.

Due to the fact that a complete large dataset for training and validating ToF denoising frameworks did not exist, and at the same ground truth depth maps for real world scenes are difficult to retrieve, Guo et al. also introduce in this paper a synthetic dataset named *FLAT* (Flexible, Large, Augmentable, ToF) for training and evaluation. *FLAT* allows to precisely reproduce different ToF cameras' raw estimates in the presence of shot noise and *MPI* artifacts, and to expose at the same time raw correlation measurements. This dataset also provides the chance to perform data augmentation over these estimations by introducing shot noise, as well as vignetting and texture.

However, *FLAT* dataset only contains scenes reporting object with diffuse material: the *MRM* module cannot thus perform well in the presence of highly specular reflections. Another limitation of the proposed model is the fact that the receptive field of *MRM* itself is too small to perceive global information from the scene, and therefore to rectify large-range *MPI*. Moreover, the loss function itself is built to highlight short-range *MPI* correction too, since the signal is significantly more powerful for shorter distances.

Furthermore, the *MOM* module is only partially capable of removing the motion artifacts for large scene. Finally, their proposal takes constant ambient light for granted, i.e., typical indoor conditions, without considering blur within a single raw measurement.

Thus, while being able to effectively remove motion, *MPI* and shot noise in some conditions, all these drawbacks limit the performance of this model in several real world scenarios, damping down its robustness in accurately reconstructing the depth of ToF images.

2.3.4 Denoising 3D Time-Of-Flight Data

[40] proposes a novel method for removing noise caused by *MPI* in ToF camera sensing. They suggest the usage of a two-parts Convolutional Neural Network (CNN), trained on the *FLAT* synthetic dataset described in the previous Subsection and introduced in [39]. The first CNN is used to learn the fundamental scene properties, e.g. the specularity of the existent objects, the ambient light density and the camera position. The second CNN takes instead the trained parameters coming from the first network, together with a noisy depth map, and it outputs the true depth map.
The proposed method is based on the idea that *MPI* noise is expressly influenced by the essential properties of the scene, therefore learning these features could provide a huge assistance in reconstructing a highly-accurate depth map. This is a key point that marks a fundamental contribution in the ToF *MPI* denoising literature: this approach of considering a double CNN model will in fact be often re-used in several situations.

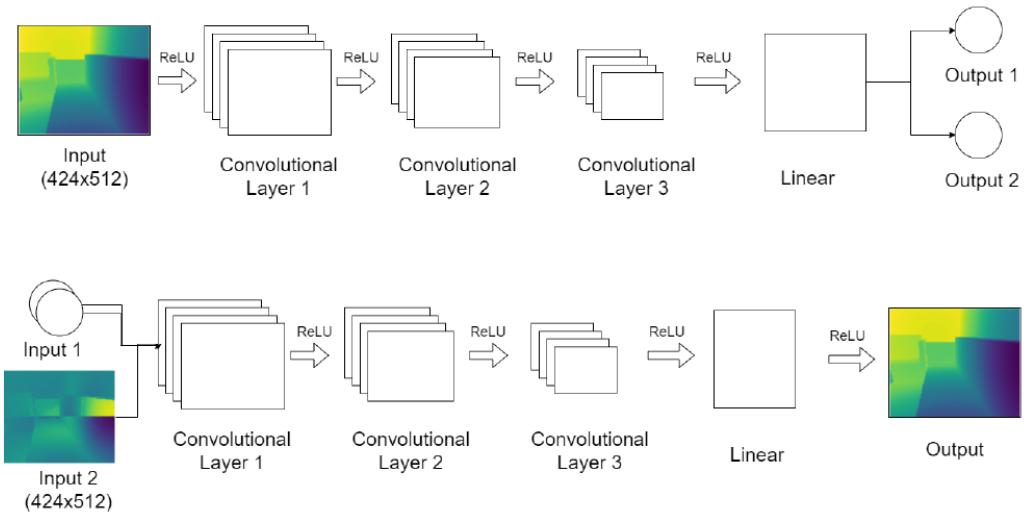


Figure 2.12: **"2-parts CNN Architecture"** (taken from [40]). The first part takes the inaccurate depth map as input and provides as output the position of the camera and the specular value of the objects present in the scene. The second part received instead as input both the inaccurate depth map and the trained hyper-parameters of the first model, and supplies the accurate (correct) depth map as final output.

More in detail, as shown in Fig. 2.12, Gupta et al. present a two-parts CNN model to

attenuate *MPI* noise from images obtained through the usage of a ToF camera, demonstrating its robust performance on *FLAT* synthetic data. The aim of the first CNN is that of trying to predict correct parameters from a noisy depth map given as input, whereas the ground truth parameters are available in the synthetic dataset used for training. These correct parameters are taken as input, together again with the noisy depth map, from the second CNN, consisting of just three convolutional layers followed by a linear one. This final linear layer provides the vector outputs, that is, the undistorted depth map.
The layers of this second model have same hyper-parameters as the first CNN. Using a two-part model like the one presented allows to achieve a better guidance over training, thus supplying the network with essential information to reconstruct accurate and concrete depth maps.

Experiments performed by the authors of the paper show that the novel approach behaves better with respect to *DeepToF* [63] and static *MRM* [39]. Besides its simplicity and real-time performance, the model presents some cons too. It has been trained on a synthetic dataset, therefore the performance on real dataset is not as satisfactory as proven in testing, because of the clear difference existing in noise statistics between real and simulated data.
Moreover, the framework is built upon data containing noise coming from a Kinect camera, thus using another ToF sensor leads to inaccurate undistorted depth maps.

Lastly, the authors of [40] state that they haven't comprised enough scenes with great specularity in the dataset they used for training, hence the proposed network architecture is not capable of working well with all those objects present in a scene with high reflectance, i.e. mirrors and metal surfaces.
Therefore, besides some brilliant intuitions, there is a lot of room for improvement.

2.3.5 Deep Learning for Multi-Path Error Removal in ToF Sensors

In the wake of previous works that apply Convolutional Neural Networks (CNNs) to data acquired with ToF cameras for denoising and *MPI* removal, the paper [3] described in this Subsection surely leaves a distinguishable mark. By exploiting the information acquired through Multi-Frequency ToF (*MF-ToF*) sensors, the proposed deep neural network architecture is able to reconstruct a depth map by precisely removing the *MPI* corruption. The authors design an ad-hoc CNN for this task, consisting of two parts, similarly to [40]. The first part is a coarse CNN capable of acknowledging the global structure of the observed scene and simultaneously globally locating *MPI*, whereas the second one is a fine CNN which takes as input the output of the previous coarse network and aims at completely removing the *MPI* while still conserving the finer details.

Furthermore, the authors propose an innovative pre-processing step which integrates the information about the depth and amplitude acquired at different frequencies. In this way they are able to build a representation that let the network understand key information to finally remove the *MPI* error. The crucial training phase has been addressed with the usage of a synthetic dataset that has been built by the authors themselves and that is publicly available. A ToF simulator has been used to generate the multi-frequency ToF data.
Fig. 2.13 shows the principle at the basis of the newly introduced approach. The workflow is very simple: after a data pre-processing phase that will briefly explained in the following, the just obtained input feeds the coarse and the fine CNNs. The output of the latter

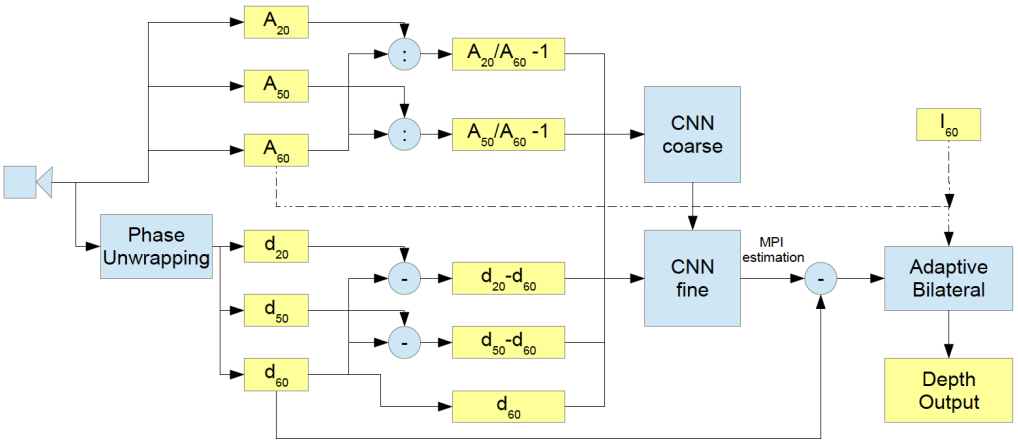


Figure 2.13: **"Coarse-Fine CNN Approach"** (taken from [3]). The CNN architecture is made of two main blocks, a coarse network that estimates the *MPI* at low resolution and a fine network that estimates the *MPI* interference at full resolution. The estimated error is then directly subtracted from the ToF depth map (at $60\,MHz$), obtaining a final undistorted depth map.

is the *MPI* interference estimation and it is subtracted to the input depth map (acquired at 60 *MHz*) in order to remove the *MPI* corruption. Lastly, an adaptive bilateral filter is used to remove also the zero-mean error.

ToF Data Representation. As anticipated, one of the key and novel aspects of this paper is the determination of those candidates among the input data which are most informative about the presence of *MPI* error. The authors settle to build a stack of five elements and to feed it to the double CNNs. In particular:

- The first input $C_1 = d_{60}$ represents the ToF depth map taken at 60 *MHz*. Since the higher is the modulation frequency the more precise is the depth estimation, this frequency has been chosen as it represents the best geometry that can be estimated before the *MPI* deletion step.
- The differences between the depth maps acquired at different modulation frequencies ($C_2 = d_{20} - d_{60}$ between 20 *MHz* and 60 *MHz*, and $C_3 = d_{50} - d_{60}$ between 50 *MHz* and 60 *MHz*) are used as second and third input, respectively, since the presence of *MPI* error becomes higher as the modulation frequency gets smaller.
- Finally, since it also holds that the higher is the modulation frequency, the lower is the deriving amplitude, juxtaposing the amplitudes acquired at different frequencies allows to clearly understand how vigorous is the presence of *MPI* in the observed scene. Thus, the ratio of the amplitudes images taken at different modulation frequencies ($C_4 = A_{20}/A_{60}$ between 20 *MHz* and 60 *MHz*, and $C_5 = A_{50}/A_{60}$ between 50 *MHz* and 60 *MHz*) are used as fourth and fifth input, respectively. Note that the "-1" term has been added to center the data around 0 if *MPI* noise is absent.

Coarse CNN. With the aim of understanding the geometrical scene structure, the coarse sub-network applies downsampling through the usage pooling layers, as can be seen by

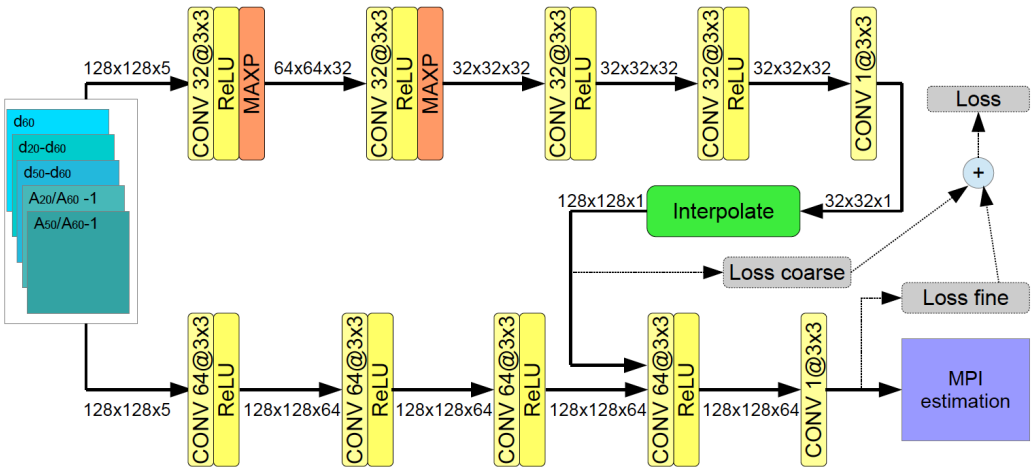


Figure 2.14: **"Coarse-Fine CNN Architecture"** (taken from [3]). Architecture of the Coarse-Fine CNN proposed in this paper [3], used for *MPI* and shot noise correction in ToF raw measurements.

looking at the upper half of Fig. 2.14, which exhaustively depicts the architecture of the model. The downsampling performed by the layers' sequence leads to an enlargement of the receptive field as a clear outcome. The coarse CNN takes as input the stack of five data channels just described (including the depth maps and the amplitude images acquired at different modulation frequencies) and is composed of five sequential convolutional layers, each followed by a ReLU activation function, except for the last one. The network permits to get a final trustworthy estimate of the regions where the *MPI* presence is stronger, even if the interference localization is not as accurate as it should be. Thus, through the straightforwardly subtraction of its output to the input data one would obtain serious artifacts, mostly in the edges' boundaries. This is the reason why another CNN, a finer one, is further introduced by the authors.

Fine CNN. This second sub-network operates at full resolution to finally get a more accurate error localization. The bottom half of Fig. 2.14 clearly shows how also this CNN presents five convolutional layers with 3×3 convolutions and ReLU activation functions, with the only exception of the last one as in the previous case. The input of the first layer is again the stack of the five data channels (described in the previous paragraph) which was also the input to the previous coarse CNN. Nonetheless, the fourth layer takes as input the concatenation between the output of the third layer and the up-sampled output of the coarse sub-network. This is of a great importance since it permits to simultaneously get the low resolution estimate with a wide receptive field through the usage of the first coarse sub-network and a more detailed yet local measurement via the second fine sub-network. In this way the authors are able to come into possession of an *MPI* error estimation that apprehends both the global and the finer details of the observed scene.

After having performed data augmentation and *K-fold* cross validation on the training dataset to avoid overfitting and at the same time to choose the best hyper-parameters to obtain the minimum Mean Absolute Error (*MAE*), Agresti et al. perform in this paper several experiments to evaluate the performance of their approach. Results indicate that,

while being extremely fast (it takes only 9.5 *ms* to evaluate a single frame), the model achieves good performance with both synthetic and real world datasets. Regarding this last case, though, there are still many limitations due to the differences between the simulated training data and real world acquisitions. Furthermore, the model is incapable of working in real-time, due to the fact that it constantly needs an input made by a stack of five data channels, as thoroughly described in this Subsection, even in inference mode.

2.3.6 Learning to Remove Multipath Distortions in Time-of-Flight Range Images for a Robotic Arm Setup

The authors of [84] propose a novel learning-based approach for removing multipath distortions in ToF raw data, consisting again of a double convolutional neural network model. They manage to use a robotic arm to mechanically gather a huge amount of ToF images affected by different *MPI* errors. These can be categorized in two classes: (1) the range *over-shooting* distortions caused by the superposition of the reflected signals coming from neighbouring structures, and (2) the range *over-smoothing* distortions arise from the superposition of the reflected signals coming from foreground and background objects.

The presented method is composed of two neural networks, referred to as F (Range-Recovery NN) and G (Boundary-Detection NN), both shown in Fig. 2.15.
The first neural network gains an understanding of the mapping from the ToF raw estimates to the undistorted depth values, whereas the second sub-network becomes proficient in detecting object boundaries in order to conduct the geometry content propagation over a framework based on geodesic filtering, as thoroughly explained in [59].
Ideally speaking, F should be able to reconstruct alone the undistorted depth estimates, but the authors state that, while it works well in the smooth regions, it finds some troubles in the object boundary ones. Thus, G is employed, together with a geodesic filtering algorithm, to spread the reestablished range estimations from the smooth to the boundary regions. In this way, the range measurement accuracy in the latter fields gets remarkably better.
In the following, each paragraph explains more in detail the relevant blocks of the approach introduced in this paper.

Calibration. The calibration method delineates a linear model for each pixel position to couple the observed measurements with the correct ones, using the traditional checkerboard pattern (with 770 separate poses). Through the employment of this scheme it is possible to remove the intrinsic systematic errors before applying the proposed algorithm, described in the next paragraphs.

Range-Recovery NN. The Range-Recovery Neural Network is a feed-forward architecture composed by three layers, in which the first contains 40 fully-connected (FC) neurons, while the second and the third layer contain 10 FC units each. Each layer uses ReLU as activation function, but the particular choice of Son et al. is that of avoiding the usage of the pooling layers (differently from almost any other existing approach), since they lead to a high reduction in the output resolution. The goal of the training is that of minimizing the Euclidean loss: the learned F is used to reconstruct the true depth measurement. More details on this and the other sub-network, explained in the following paragraph, can be found in the paper.

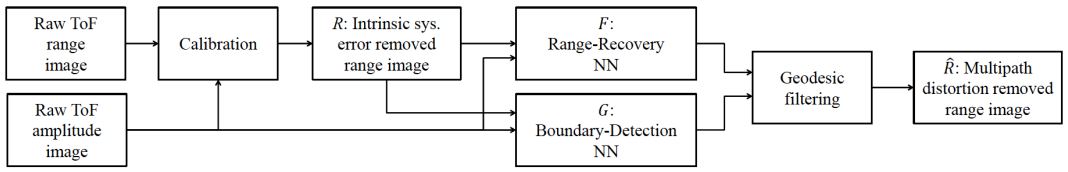


Figure 2.15: **"Range-Recovery + Boundary-Detection NNs Architecture"** (taken from [84]). After a calibration model that removes the systematic errors outputting R, the Range-Recovery NN F is used to estimate the true depth values, while the Boundary-Detection NN G is employed to detect the object boundaries. Lastly, a geodesic filtering algorithm is utilized to compute the final depth estimate $\hat{R}$.

Boundary-Detection NN. The authors of [84] decide to use four separate feed-forward NNs, one for each edge detectors' groups. The ground truth boundaries and their directions are calculated through the usage in the target depth maps of the Canny edge detector described in [19], that is an algorithm capable of identifying a wide range of images' edges using multiple stages. The detected edges are thus split into four groups based on a uniform division of the edge orientation.
Each of the four networks consists of two FC layers with ReLU as activation function: the first has 40 neurons while the second counts 20 units. The output layers have two neurons, that stand for the edge and non-edge likelihood scores, respectively, after the employment of a softmax operation. These four networks are trained through the minimization if the cross entropy loss.

Geodesic Filtering. For each pixel of the image, the maximum comeback is taken from the four neural networks for the boundary likelihood score, following the approach explained in [6]. Finally, non-maximum suppression and hysteresis thresholding are carried out to calculate the binary edge map for geodesic filtering.

Several experiments show that the proposed architecture achieves good performance for what concerns the removal of *MPI* noise, while not being able to work in all real world scenarios. Indeed, it isn't capable of addressing the dependence on material properties, which is known to affect the ToF raw measurements. Moreover, its overall computation time is relatively high, due to the presence of the geodesic filtering phase that is quite slow. Thus, for applications working in real-time, as the ones in the embedding and especially robotic fields, this approach cannot be applied.

2.3.7 Very Power Efficient Neural Time-of-Flight

Conventional methods for *MPI* and shot noise removal of ToF raw data are usually built upon groundless hypotheses and conjectures, but these often turn into being unacceptable when different scenes and received signals' intensities -in particular for weak input signals- are taken into account. It is thus highly strenuous to choose the best parameters that allow to obtain good results for all real world scenarios. As opposed to this traditional trend, the authors of [21] propose an end-to-end learning-based approach to correct the distorted and noisy ToF raw measurements and produce as output high quality depth maps. Doing so, they are able to abstain from the extremely compound parameters tuning step, especially that of particularly weak signals.

Network Architecture												
Name	D1	D2	D3	D4	U1	Conv1	U2	Conv2	U3	Conv3	U4	Conv4
Layer	conv+ Relu	conv+ Relu+ conv+ Relu	conv+ Relu+ conv+ Relu	conv+ Relu	conv+ Relu+ upsample	conv+ Relu+ conv+ Relu	conv+ Relu+ upsample	conv+ Relu+ conv+ Relu	conv+ Relu+ upsample	conv+ Relu+ conv+ Relu	conv+ Relu+ upsample	conv
Kernel	5×5	3×3	3×3	3×3	3×3	1×1/ 3×3	3×3	1×1/ 3×3	3×3	1×1/ 3×3	3×3	3×3
Stride	2	2/1	2/1	2	1	1	1	1	1	1	1	1
I/O	4/16	16/32	32/64	64/128	128/128	192/128	128/96	128/64	64/32	48/16	16/8	8/4
Input	ToF raw	D1	D2	D3	D4	D3+U1	Conv1	D2+U2	Conv2	D1+U3	Conv3	U4

Figure 2.16: **Power Efficient NN Architecture** (taken from [21]). In the shown framework, based on an autoencoder architecture, *conv* stands for a convolutional layer, while *upsample* represents an up-sampling layer.

The solution introduced in this paper consists of an encoder-decoder architecture with the usage of skip connections to enhance the accuracy of the results. Fig. 2.16 depicts the described framework. In particular, the decoder is composed by a sequence of four down-sampling layers (indicated with $D1$ to $D4$ in the figure above), whereas four up-sampling layers (denoted as $U1$ to $U4$ in the same image) add up to form the encoder. These layers combine the strided convolution layers with the up-sample ones.

Differently from conventional RGB camera tomography, in the case of ToF cameras the depth maps and the raw estimates should be compatible with the geometry and the underlying scene structure. Moreover, the latter can be affected by spatial structures, modulation frequency, material properties of the observed objects and a number of other factors. This is the reason why Chen et al. suggest to use a new loss for the training procedure, named *ToF Loss* $\mathscr{L}_{ToF}$, that is the sum of two terms, denoted raw loss and depth loss.

Raw Loss. The first is the *Raw Loss* $\mathscr{L}_{Raw}$, which is defined in this way:

$$\mathscr{L}_{Raw} = \frac{1}{N}\sum_{i,j}^{N}[|r_{i,j}^{gt_0} - r_{i,j}^{pre_0}| + |r_{i,j}^{gt_1} - r_{i,j}^{pre_1}| + |r_{i,j}^{gt_2} - r_{i,j}^{pre_2}| + |r_{i,j}^{gt_3} - r_{i,j}^{pre_3}|] \tag{2.26}$$

where $r_{i,j}^{pre_n}$ indicates the ToF raw estimates produced by the network, while $r_{i,j}^{gt_n}$ stands for the ToF raw measurements captured under long exposure settings as ground truth. In both cases, n is thought to be in the range $[0,1,2,3]$. The *Raw Loss* $\mathscr{L}_{Raw}$ is used to minimize the Mean Absolute Error (*MAE*) between the raw estimates and the corresponding ground truth frames.

Depth Loss. With the aim of guaranteeing high quality undistorted depth maps, via the minimization of the Mean Absolute Error between the depth value calculated from ToF raw measurements and the ground truth one, the *Depth Loss* $\mathscr{L}_{Depth}$ is defined as follows:

$$\mathscr{L}_{Depth} = \frac{1}{N}\sum_{i,j}^{N}|d_{i,j}^{gt} - d_{i,j}^{pre}| \tag{2.27}$$

where $d_{i,j}^{pre}$ represents the depth value computed starting from ToF raw estimates generated by the network according to:

$$d_{i,j}^{pre} = \frac{c}{2}\cdot\frac{1}{2\pi f_{LED}}\cdot\left[\pi + atan2\left(\frac{r_{i,j}^{pre_3} - r_{i,j}^{pre_1}}{r_{i,j}^{pre_2} - r_{i,j}^{pre_0}}\right)\right] \tag{2.28}$$

where $f_{LED} = 6\ MHz$ and $c = 299792458\ m/s$.

ToF Loss. The introduced ToF Loss is defined as the weighted sum of the two previously described losses:

$$\mathscr{L}_{ToF} = \alpha \mathscr{L}_{Depth} + \beta \mathscr{L}_{Raw} \tag{2.29}$$

The main novelty introduced by this paper, though, is the introduction of a complete ToF dataset, containing scenes captured under a number of different conditions. The exhaustive experiments performed by the authors demonstrate that the proposed method is able to actively correct raw ToF noisy data. It is also capable of working at a higher depth frame rate, due to the fact that the exposure time could be outstandingly brought down. Moreover, the presented framework reveals a better performance in the depth estimation of low reflectivity objects with respect to previous works.
Despite its robustness and its power efficiency, the main limitations of the suggested model are the complexity of the model itself and the low accuracy of the results, if compared with other state-of-the-art architectures.

2.3.8 Deep Learning for Transient Image Reconstruction from ToF Data

The authors of [18] introduce a deep learning unconventional approach for correcting *MPI* noise in ToF raw data by estimating the direct components of the incoming signal. The strength of this new architecture lies in the capability of the network to roughly calculate the scene impulse response and to reconstruct an undistorted version of the input depth map by reducing the amount of *MPI* error present.

Since deep neural networks can present several troubles when facing high dimensional data, which is exactly what backscattering vectors are, the framework presented in this paper is composed by two key blocks, as depicted in Fig. 2.17. The first out of the two is a predictive model that tries to understand the relationship existing between the errors-prone ToF measurements and the encoded rendering of the transient data, whereas the second is a fixed backscattering model which instead moves the encoded version into a higher dimensional light response, that is, into the corresponding transient vector.

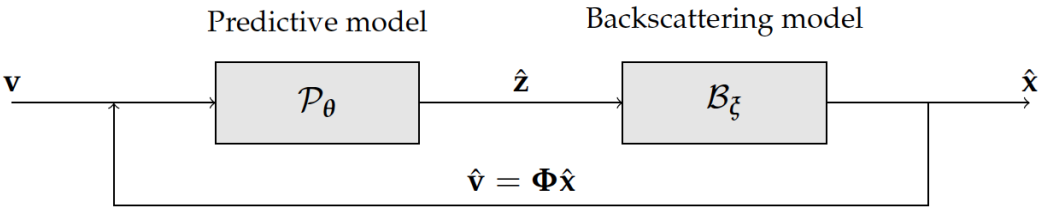


Figure 2.17: **"Predictive + Backscattering Model Architecture"** (taken from [18]). The figure depicts the structure of the proposed model. As stated in the paper, the predictive model represents the deep learning network of the entire pipeline, whereas the principal task of the backscattering part is that of compressing the high dimensional transient information into a more straightforward representation.

Backscattering Model. The proposed approach marks the beginning of a completely new

type of data-driven models aiming at correcting noisy ToF raw data with the reconstruction of transient images. In fact, the key idea at the basis of the backscattering model is that of transforming the high dimensional transient data into a more simple version which can be manageable more effortlessly. Thus, the goal of this module, denoted as B_ζ in Fig. 2.17, is that of mapping a latent variable z into the respective backscattering vector x, as described in the following equations:

$$B_\zeta = \mathbb{R}^L \mapsto D_x \subseteq \mathbb{R}^N, \qquad z \mapsto x = B_\zeta(z) \tag{2.30}$$

where $L \ll N$, ζ are some trainable parameters and D_x is the domain of all possible backscattering vectors. B_ζ could be any generative model, e.g. a generative adversarial network (GAN) or a Variational Autoencoder (VAE), which is capable to accurately map low-dimensional data into high-dimensional transient ones.
Nonetheless, in real world scenarios the first and second order reflections contain most of the backscattering vector energy, as thoroughly reported in [28]. For this reason, the authors of this paper, [18], propose to use as backscattering model a simple stochastic mapping from a 4-dimensional z vector ($z \in \mathbb{R}^4$, where the four values can be thought at as the amplitudes and the path lengths of the first two interfering rays) to an approximation of the backscattering data.
Thus, the backscattering model must transform these four values to the approximated backscattering vector. Due to what has been previously said regarding the first and second order reflections, the newly obtained backscattering vector presents all zeros in every entry, except for two peaks in correspondence of the first two interfering rays.

Predictive Model. This module represents the deep learning part in the proposed architecture, which takes as input a matrix containing raw ToF estimations acquired at several different modulation frequencies. The aim of this network is that of providing for each measurement the corresponding values in the latent space Z. More in detail, the predictive model is based on a non-linear function $P_\theta(\cdot)$, with parameters θ, which receives as input the vector v and generates as output the corresponding vector estimation z, referred to as $\hat{z}$ in the following.
Having in mind the goal of making the most out of the spatial information for every pixel prediction, the authors propose the usage of a local neighbourhood around the pixel itself of size $(2P+1) \times (2P+1)$, with $P = 1$, thus forming a window of 3×3.
Thus, the predictive model is composed by a CNN, whose first layer merges a weight kernel and another small $(2P+1) \times (2P+1)$ kernel centered around the pixel itself. While the former has the objective of recovering the relevant information from every considered pixel, the latter is instead focused on the acknowledgment of the local information.
Fig. 2.18 shows the structure of this module and the mapping from v to $\hat{z}$.

A huge advantage of the proposed approach is that the global information is present in transient data form and hitherto carries the knowledge concerning the scene structure. The backscattering vector, simply composed by all zeros a part from two peaks in the presence of the first and the second interfering rays, might seem a little "wild" and yet it demonstrates to be very effective for precise depth estimation. Thus, the model results very straightforward if compared to many other approaches, such as [63], [40] or even [3]. Despite its simplicity, the model is yet very successful in reconstructing the noisy depth maps, since the neural network of the predictive model just has to estimate two peaks

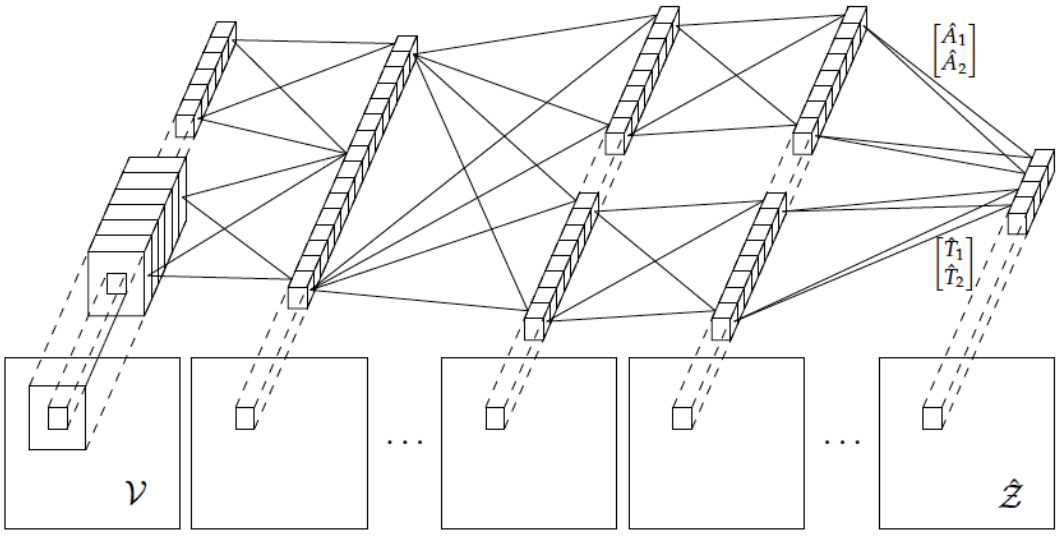


Figure 2.18: **"Predictive Model Structure"** (taken from [18]). This figure depicts the structure of the proposed predictive model. The first layer of the CNN combines a weight kernel and a small 3×3 one centered around each pixel: the former retrieves the relevant information from each pixel while the latter tries to acknowledge local information.

from the input measurements. Extensive experiments prove good performance even in real world data, with results close to state-of-the-art frameworks, even without complex architectures or huge amount of training data (the model is supposed to contain only a few thousand parameters). Nonetheless, more advanced structures have to be studied to further draw on the spatial context in the backscattering measurement and to reconstruct transient data, but this is doubtless a huge step towards less computational-demanding neural network architectures applied for the denoising ToF raw measurements.

2.3.9 Spatial Hierarchy Aware Residual Pyramid Network for Time-of-Flight Depth Denoising

The Multi-Path Interference (*MPI*) critically corrupts the depth image acquired by ToF cameras. [68] and [20] have recently gone in the direction of using the hierarchical representations of the scene to depth and 3D shape estimation. Thus, in [27] Dong et al. decide to enhance the *MPI* denoising of ToF raw data through the usage of the spatial hierarchical information. They propose *SHARP-Net*, a "Spatial Hierarchy Aware Residual Pyramid Network" for ToF raw measurements denoising. The goal is that of fully making use of scene structures at different levels: in fact, they state that a multiple-scale feature pyramid based on the spatial hierarchical structure of the scene can supply a suitable receptive field, eventually leading to an improved *MPI* and shot noise removal.

Specifically, *SHARP-Net* is composed of three different modules, namely Residual Regression Module, Residual Fusion Module and Depth Refinement Module. Through the combined usage of all these blocks the proposed framework is capable of precisely removing noise for ToF depth images.
Fig. 2.19 displays the architecture of the network, depicting in detail each of these modules that are briefly described in the next paragraphs.

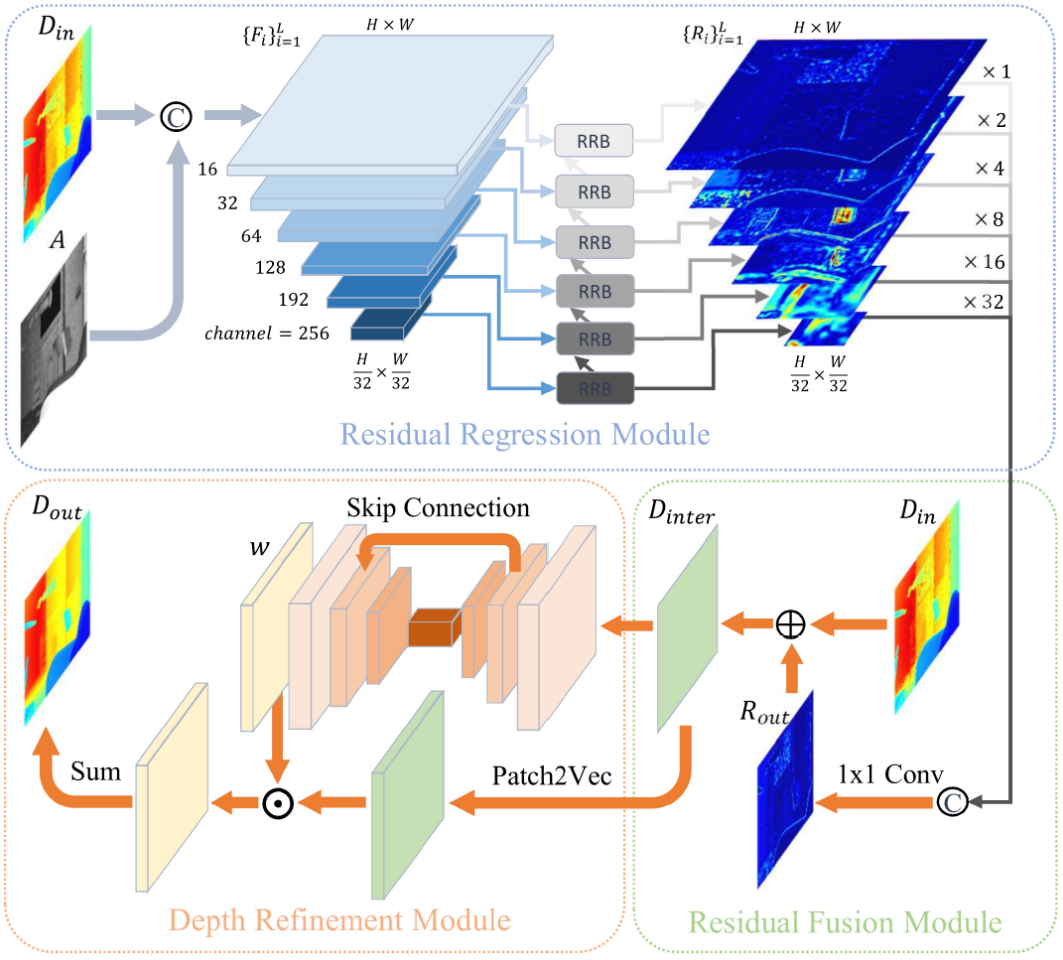


Figure 2.19: **"SHARP-Net Architecture"** (taken from [27]). This framework is able to exhaustively exploit scene structures at different levels through three different modules, denoted as Residual Regression, Residual Fusion and Depth Refinement Modules. In this figure, $\odot$ represents the dot product operation, © denotes the concatenate operation while $\oplus$ introduces the addition operation. "Patch2Vec" stands instead for the reshaping operation of the pixels' neighbourhoods to a vector.

Residual Regression Module. Being the backbone of *SHARP-Net* for multi-scale feature extraction, the Residual Regression Module is like an encoder that squeezes out a feature pyramid $\{F_i\}_{i=1}^{L}$ on different levels, starting from an input composed by a stack of the depth map D_{in} and the amplitude image A. In the definition of the feature pyramid, F_i designates the feature map extracted at the i^{th} level whereas L is the totality of the pyramid layers. Going from the bottom to the top, the feature pyramid gradually extract the more detailed geometric structure data. At each level, the depth residual map coming from the lower level, after being up-sampled by a factor of 2 via bicubic interpolation and coupled with the feature map of the layer at hand, is given as input to five sequential convolutional layers, whose output is the residual map R_i for the current layer. By doing that, the residual maps at lower resolutions are capable of encoding the depth noise present in larger shapes whereas the ones at higher resolutions give more attention on the noise existing in smaller

and more detailed scene structures. At then end the Residual Regression Module outputs a residual pyramid $\{R_i\}_{i=1}^{L}$ composed by the depth residual maps extracted at each level.

Residual Fusion Module. Even if the highest level of the just described pyramid includes the information from all the levels below and can thus be interpreted as a depth error estimation, after the convolutional operation the details about lower resolution levels may go away and depart. This is the reason why the Residual Fusion Module is further introduced in this paper work [27] to unambiguously integrate and homogenize the depth residual maps acquired at all scales. At each level the depth residual map is up-sampled to its original resolution, again via bicubic interpolation, and all the up-sampled depth residual maps are then concatenated together. The newly obtained residual volume is lastly given as input to a 1×1 convolutional layer, whose output is the depth residual map R_{out} that is added to the original input depth image in order to reconstruct the final depth image D_{inter}.

Depth Refinement Module. The two introduced modules are helpful in removing the *MPI* noise, but they do not successfully work with the shot denoising as well. To tackle this problem, a third block is lastly introduced by the authors, that is the Depth Refinement Module, based upon a Kernel Prediction Network [11]. This module takes the output D_{inter} of the previous Residual Fusion Module as input, and it produces through a U-Net architecture with skip connection a weight matrix, that is nothing more than a vectorized filter kernel for each of the depth image pixels. After the "Patch2Vec" operation, that consists in making a patch matrix by vectoring a neighbourhood for every depth image pixel, the two obtained matrices are element-wisely multiplied, originating a 3D volume. By summing over it, the refined depth image D_{out} is lastly derived.

Extensive experiments conducted by the authors of the paper demonstrate that *SHARP-Net* highly outclasses state-of-the-art methods, for what concerns both quantitative and qualitative metrics on synthetic and realistic datasets, such as ToF-FlyingThings3D (TFT3D) [77], FLAT [39] and True Box [2]. Moreover, the ablation studies managed by Dong et al. reveal that, even removing one of the modules or decreasing the number of the backbone's levels, the obtained results are satisfying. Thus, despite its complexity, the model can be adapted to the needs of a real-time demanding application by partially removing a module or limiting the number of layers, and this represents a great advantage that can be taken into account when dealing with the embedded world.

2.3.10 Unsupervised Domain Adaptation for ToF Data Denoising with Adversarial Learning

Inspired by the previous work of [3], Agresti et al. in [2] propose a revolutionary transfer learning architecture able to successfully remove the noisy errors in real world data by putting to use unlabeled real world ToF measurements to adapt the training performed on synthetic to the real world data. Specifically, the Coarse-Fine CNN drawn on the one introduced in [3] is jointly trained in a supervised fashion on a synthetic dataset, for which the ground truth depth is known, and at the same time in an unsupervised way on a real dataset, for which the depth ground truth is unknown.
For the training on the latter data, the authors manage to use an adversarial loss and a learning framework based on the Generative Adversarial Network (GAN) model. In

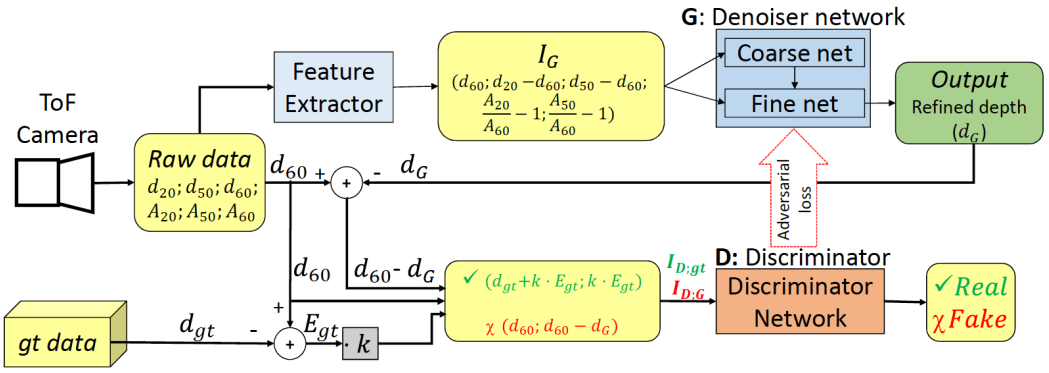


Figure 2.20: **"DA-F Architecture"** (taken from [2]). The generator is a coarse-fine CNN inspired by the previous network architecture described in [3], while the discriminator is a straightforward CNN with four convolutional layers.

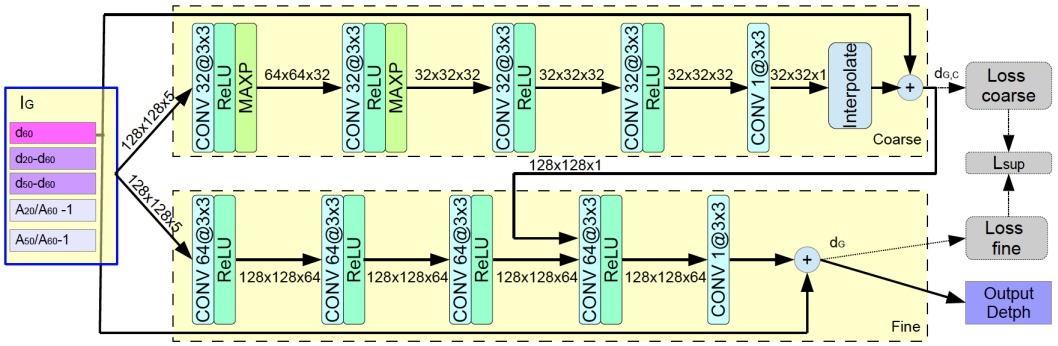
doing so, this approach paves the way for research field concerning the application of the domain adaption to the denoising of ToF raw depth data.

Fig. 2.20 reveals the architecture of the proposed approach. As it is possible to notice by looking at the figure above, the generator is based on a Coarse-Fine CNN that is fed with several features learned from the ToF raw measurements and outputs the error-free depth map estimation. The discriminator network is instead employed for the adversarial learning and domain adaptation task.
Following the approach described in [3] and already explained in detail in Subsection 2.3.5, the input data is pre-processed so that it may be feasible to draw out a representation I_G which includes suitable insights about the *MPI* appearance and robustness. In particular, five different feature channels are pulled out from the ToF estimates and stacked together according to:

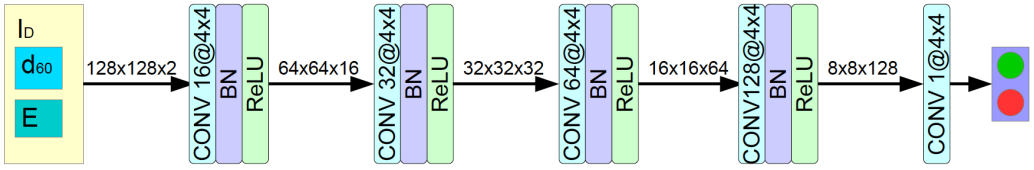
$$I_G = \left(d_{60}; d_{20} - d_{60}; d_{50} - d_{60}; \frac{A_{20}}{A_{60}} - 1; \frac{A_{50}}{A_{60}} - 1 \right) \tag{2.31}$$

where d_x and A_x represent the depth maps and the amplitude images, respectively, acquired at x *MHz*. Differently from [3], without any further pre-processing of the input data, I_G is given as input to the generator Coarse-Fine convolutional neural network, that outputs the undistorted depth map.
As previously anticipated, the discriminator CNN block is instead employed to implement an unsupervised domain adaptation from synthetic to real data, through the usage of an adversarial learning strategy. This idea represents a key point of clear advancement with respect to previous works on this topic.
The following paragraphs explain more in detail the structure of the proposed architecture, whose specifics can be thoroughly read out in the already cited paper.

Generator Network Architecture. As shown in Fig. 2.21a, the proposed generator network G is composed by a Coarse-Fine CNN, as already apprehended. The coarse network, made of a stack of five convolutional layers with 3×3 kernels with ReLU as activation function (except for the last one), is able to acknowledge and perceive at the end a broad



(a) *Architecture of the generator network G.*



(b) *Architecture of the discriminator network D.*

Figure 2.21: **"Generator and Discriminator of DA-F"** (taken from [2]). Architecture of the generator network G (a), and of the discriminator network D (b) of the proposed framework.

receptive field. Its output, denoted as $d_{G,C} = G_C(I_G)$, is directly a scene geometry estimation at low resolution (and not the *MPI* alteration, differently from [3]). The fine network consists again of five convolutional layers with 3×3 kernels with ReLU as non-linear activation function (except for the last one), but without max pooling layers in the first two blocks. The aim of this second sub-network is that of labouring at full resolution in order to finally get precise insights regarding the observed scene details. The output of the fine CNN is denoted as $d_G = G(I_G)$. Worthy of notice is that, similarly to the approach described in [3], the output of the first coarse sub-network $(d_{G,C})$ is first up-sampled using a bilinear interpolation and then fed to the 4^{th} layer of the fine network. This is done so to make fully use of the global scene information.

Discriminator Network Architecture. Having in mind the idea of carrying out the unsupervised domain adaptation task, the discriminator convolutional neural network D is introduced. Its goal is that of apprehending the association existing between the noisy depth data and the related noise image, so to recognize how to distinguish the ground truth depth maps from the undistorted ones produced by G . Thus, the discriminator is fed with the inaccurate depth map d_n and the error map E (that could be the difference either between the noisy and the ground truth depth maps $E_{gt} = d_n - d_{gt}$ or between the noisy and the generator output depth maps $E_G = d_n - d_G$). The output of D should be 0 if the input is $I_{D;G} = (d_n; d_n - d_G) = (d_n, E_G)$ or 1 if the input is $I_{D;gt} = (d_n; d_n - d_{gt}) = (d_n, E_{gt})$. This means that the discriminator, by focusing on the raw ToF measurements and their relationship with the estimated error, gets rid of all the data generated by G that are not

coherent with the ground truth information, thus preventing the output of G to diverge from its input.
The architecture of the proposed discriminator network is shown in Fig. 2.21b: it is made of five sequential convolutional layers with 4×4 kernels, each followed by a batch normalization layer and a ReLU activation function, again except for the last one. The network is trained by minimizing the loss function described as follows:

$$\mathscr{L}_D = -E\left(log(D(I_{D;gt})) + log(1 - D(I_{D;G}))\right) \tag{2.32}$$

The generator G is trained both on a synthetic dataset in a supervised fashion and on a real dataset in an unsupervised way, through the minimization of a loss function constituted by two parts:

$$\mathscr{L}_G = \mathscr{L}_{sup} + w \cdot \mathscr{L}_{adv} \tag{2.33}$$

where:

$$\begin{cases} \mathscr{L}_{sup} = E[|d^s_G - d^s_{gt}|] + E[|d^s_{G,C} - d^s_{gt}|] & \text{in which } s \text{ is the synthetic dataset.} \\ \mathscr{L}_{adv} = E[-log(D(I^r_{D;G}))|] & \text{in which } r \text{ is the real dataset.} \end{cases} . \tag{2.34}$$

More details on the training procedure are found in sections 6 and 7 of the paper [2] that is explained in this part.
The effectiveness of the proposed approach is demonstrated by the evaluation on two different real world datasets, proving that the framework outperforms all other state-of-the-arts methods, without adding too much complexity to the overall structure. Moreover, the introduced approach, firstly thought for ToF denoising, could be theoretically applied to all data denosing tasks where the learning phase done over a supervised (synthetic) dataset should be adapted to (completely) different domains. However, the model incurs on the same limitation that affects the previous work on which this is based on, that is the impossibility to work in real-time due to the need of an input acquired at different modulation frequencies.

Chapter 3

Point Cloud Registration

Point clouds have recently become more and more employed as a way of representing the 3D world, as thoroughly explained in [48]. This is particularly true if one considers the accelerated growth of high precision sensors, i.e., LiDAR, Kinect and ToF cameras. The main limitation of these devices is the narrow sight range they could observe the scene at, and this is the reason why registration schemes come of paramount importance to build and extract wider 3D real world sections. When talking about point cloud registration the problem which has to be tackled is that of estimating the transformation matrix between two point clouds. Employing this transformation matrix, it is feasible to obtain a full 3D point cloud by incorporating several incomplete scans of the same scene. The importance of point cloud registration can be mostly appreciated by looking at its precious contribution in several computer vision tasks, briefly described in the following.

3D Reconstruction. Generating a complete 3D scene is a basic and significant technique for various computer vision tasks, including high-precision 3D map reconstruction in autonomous driving, 3D environment reconstruction in robotics and so on so forth.

3D Localization. Discovering an agent location in a 3D scene is of extremely interest for robotic fields. Point cloud registration algorithms may be able to take a real-time 3D scene insight and precisely locate it onto the corresponding 3D environment.

Pose Estimation. Given a point cloud A that represents 3D real-time scene and a second point cloud B that instead indicates the 3D domain in which the first belongs to, retrieving the right translation and rotation parameters could result in finding the right pose information about point cloud A with respect to B. This may be eventually employed for decision-making in robotic tasks. A possible usage could be that of getting the robotics arm's pose information to decide where to move to grab an object accurately. This last application is particularly important for this thesis work, that has the objective of building a NN-based approach for computing the 6 DoF pose estimation of objects detected with ToF sensors.

An example of point cloud registration is illustrated in Fig. 3.1, taken from [74]. By looking at the image, it is possible to notice a tree, a lamppost and a bench observed through a laser from two different poses. The key element is that the points are imperceptible, thus only the information regarding their position in the scene can be used to align the two point clouds. Fig. 3.1-*Left* shows the starting position of the two point clouds, while

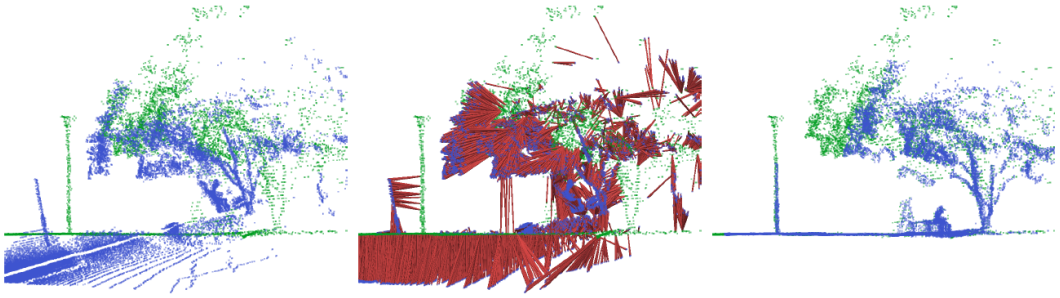


Figure 3.1: **"Example of Point Cloud Registration"** (taken from [74]). On the *left* the initial position of the two point clouds is depicted, on the *middle* is shown the alignment error with dark red lines, whereas on the *right* the final alignment of the two point clouds is represented.

Fig. 3.1-*Middle* represents with dark red lines the initial error alignment existing between the two input scans. The individual points are all alike and indistinguishable one another. Nonetheless, the information coming from the closeness to other points provides everything that needs to automatically align the two point clouds, as it is possible to see in Fig. 3.1-*Right* where the final result is depicted.

As shown in this basic example, the alignment step is a non-trivial task, and it usually needs enough overlap between the two point clouds so that one could use the overlapping parts of these views to find the best alignment. In the history and literature of point cloud registration methods, this alignment procedure is most of the times thought as an optimization problem relying on the point clouds geometry, as exhaustively illustrated in [44]. However, many other methods leveraging on features extraction, correspondences and deep learning have been recently proposed and used to tackle this intricate task.

In the following Sections, the state-of-the-art frameworks for point cloud registration are described in detail. In particular, according to the comprehensive survey exemplified in [48], these methods can be divided into three main categories, named **optimization-based** methods, **feature-learning** methods and **end-to-end learning-based** methods.
Fig. 3.2 briefly shows the main differences of these methods. As it is possible to notice, given two input point clouds, optimization-based methods iteratively compute the correspondences between the source and the target point clouds, finally creating as output the transformation T that best aligns them. Feature-learning methods estimate the features using a deep neural network (or any other different methods), and only in a second moment they iteratively estimate the correspondences based on these features extracted, until they reach the final transformation T. Finally, end-to-end learning-based methods use an end-to-end framework to estimate the optimal transformation T without performing any iterative approach nor relying on extracted features.

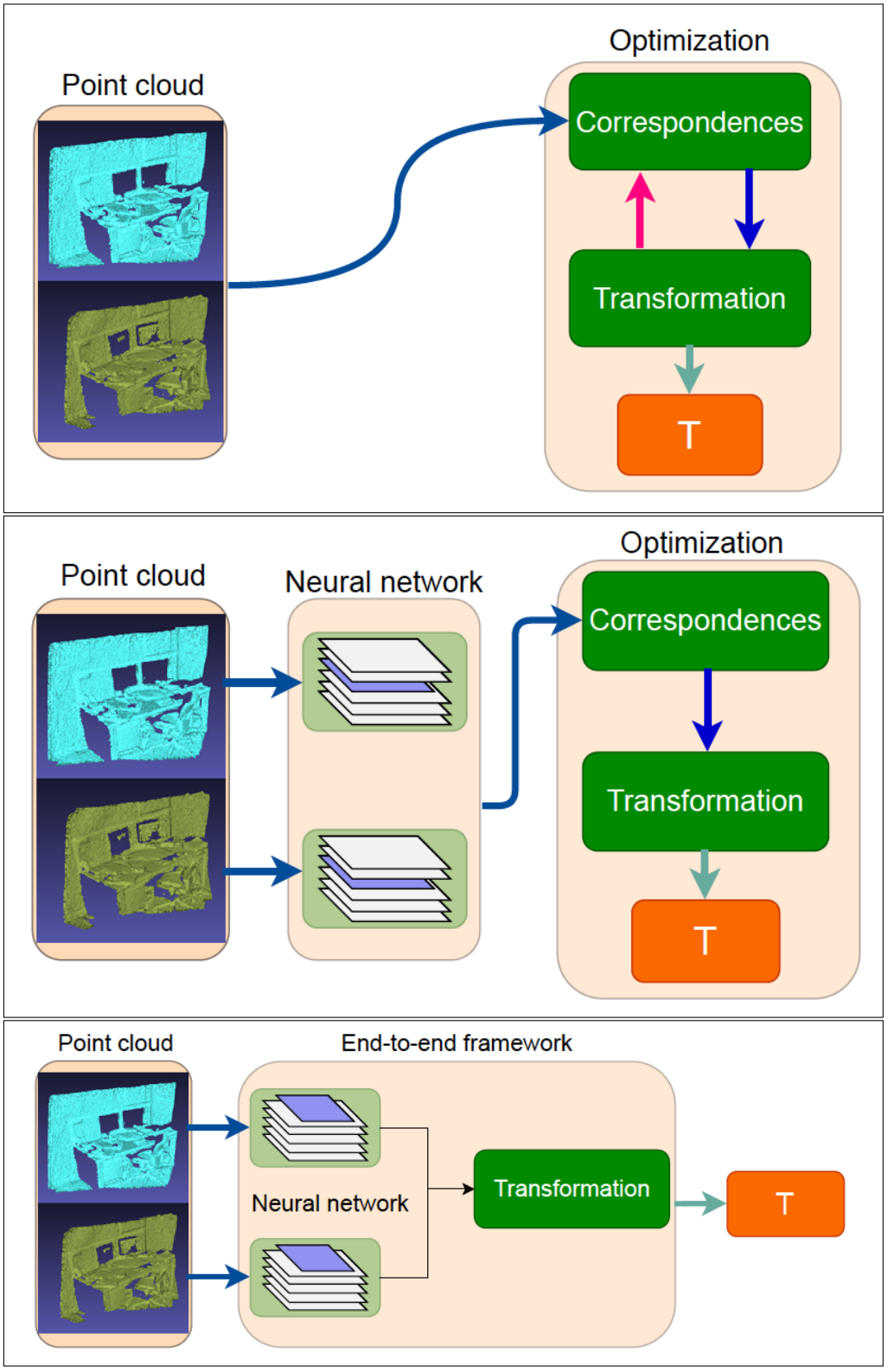


Figure 3.2: **"Point Cloud Registration Methods"** (taken from [48]). (a) depicts an Optimization-Based Framework for Point Cloud Registration, (b) shows a Feature Learning-Based Framework while (c) represents an End-to-End Learning-Based Framework.

3.1 Optimization-Based Methods

Optimization-based registration consists of using optimization policies to accurately estimate the transformation matrix. It is usually possible to distinguish two different steps: searching for the correspondences and estimating the transformation.
Fig. 3.2(a) summarizes the main stages at the very basis of these methods: correspondence searching refers to the finding of the matched point for every point in another point cloud, while transformation estimation makes reference of the estimation of the transformation matrix by using the computed correspondences. These two steps are iteratively conducted in order to find at the end the optimal transformation.

The main advantage of these methods relies on the fact that they do converge without the needing of any training data. Moreover, they generalize well to unknown scenes.
Unfortunately though, many sophisticated strategies are required to overcome the variations of density and noise in the input data, the outliers and the partial overlap, and this inevitably increases the computational cost of these algorithms.

In the following subsections, the state-of-the-art optimization-based methods are presented: in particular, subsection 3.1.1 concentrates on the ICP [12] algorithm, subsections 3.1.3 and 3.1.2 focuses on its variants, namely Go-ICP [99] and LM-ICP [32], while subsection 3.1.4 finally strengthens the *FGR* [106] approach, that is the one chosen to be reproduced in this thesis work, as will be explained more in detail later on.

3.1.1 ICP

In the early 1990s [12] introduces the ICP (Iterative Closest Point) algorithm, that is one of the most used in the Point Cloud Registration history (alongside its many variants that have been developed during the years). As stated by the authors, ICP algorithm only requires a procedure to find on a geometric system the closest point for each given one. Thus, it is formulated in such a way that it invariably monotonically converges to the closest local minimum. Experiments and experience show that the rate of convergence is fast especially during the first few iterations.

After having identified a suitable starting set of transformations (that is, rotation and translation) for a particular class of objects, it is feasible, through the evaluation of every initial registration, to minimize in a global manner a metric that takes into account the mean squared distance over the 6 *DoF*. Given a theoretical form X, ICP algorithm could be expressed according to this geometric entity, provided that its inner model and characteristics are known in advance.
Thus, this algorithm can be equally applied to sets of points, line segments, parametric and implicit curves, triangles, parametric and implicit surfaces.

In the method's illustration, a "data" shape P is registered, that means positioned, to be oriented in the best possible way with respect to a "model" shape X. The "data" shape must therefore be broken up into a point set, in the case that it is not yet in that form.
The distance metric d between an individual data point $\vec{p}$ and a "model" shape X is defined by the authors as:

$$d(\vec{p},X) = \min_{\vec{x}\in X} \|\vec{x} - \vec{p}\| \tag{3.1}$$

The closest point in X that gives the minimum distance is denoted as $\vec{y}$ and it is defined so that:

$$d(\vec{p},\vec{y}) = d(\vec{p},X), \quad \text{where } \vec{y} \in X. \tag{3.2}$$

Indicating with Y the resulting set of the closest points and with C the closest point operator, it is possible to write:

$$Y = C(P,X) \tag{3.3}$$

Given the resultant corresponding set Y, the least squares registration is calculated as:

$$(\vec{q},d) = Q(P,Y) \tag{3.4}$$

and the positions of the "data" shape point set are finally corrected as $P = \vec{q}(P)$.
Overall, ICP algorithm can be reported following Alg. 1 as stated by the authors of [12].

Despite the fact that ICP is supposed to monotonically converge to a local minimum from any given transformation of the data point set, it could happen that it doesn't converge on the desired global minimum. Therefore, the only way to be sure to reach the global minimum is that of finding the minimum of all the local minima.
Unfortunately though, it is difficult to precisely identify, in general, the partitioning of the registration state space into local minima regions of attraction, named *wells*. This is caused by the fact that the partitioning could be very different for every possible shape encountered in the point dataset.

ICP algorithm has always represented a benchmark method for all the point cloud registration approaches presented in the following years, and this is because of its capability of working in a 6 DoF space without pre-processing the 3D point data, its independence of shape representation and its robustness to normally-distributed vector noise.
Besides its great advantages, it also presents some important drawbacks: it is susceptible to evident statistical outliers, the cost of local matching can get quite high for small admissible occlusions percentage and it could get stuck in local minima if it lacks of a proper initialization.

Algorithm 1 Iterative Closest Point (ICP) Algorithm [12]

Require: Point set P with N_p points $\{\vec{p}_i\}$ from the "data" shape and "model" shape X
Initialize $P_0 = P$, $\vec{q}_0 = [1,0,0,0,0,0,0]^t$ and $k = 0$
while convergence **do**

1. Compute the closest points: $Y_k = C(P_k,X)$ ▷ cost:$O(N_pN_x)$
2. Estimate the registration: $(\vec{q}_k,d_k) = Q(P_0,Y_k)$ ▷ cost:$O(N_p)$
3. Apply the registration: $P_{k+1} = \vec{q}_k(P_0)$ ▷ cost:$O(N_p)$
4. Conclude the loop when the mean squared error value drops down below a preset threshold $\tau > 0$ specifying the desired registration precision: $d_k - d_{k+1} < \tau$

3.1.2 LM-ICP

[32]'s main novelty is the rejection of one of the basic principles of ICP [12], that is its closed-form internal iterations, and the usage as a replacement of a conventional iterative non linear optimizer, the Levenberg–Marquardt (LM) algorithm presented in [75]. Following this approach, the proposed method incurs no significant loss of speed, but remarkably increases the robustness of ICP itself.

In its easiest version, the ICP algorithm performs two stages iteratively. Starting from an initial estimate of the registration parameters, a_0, the algorithm forms a sequence of estimates a_k, which progressively reduce the error $E(a)$ defined as:

$$E(a) = \sum_{i=1}^{N_d} w_i \min_j \varepsilon^2(|m_j - T(a;d_i)|) \tag{3.5}$$

where $\{m_j\}_{j=1}^{N_m}$ and $\{d_i\}_{i=1}^{N_d}$ are the "model" and the "data" elements respectively, $\varepsilon^2(|x|)$ is an error function that measures the alignment (typically it holds that $\varepsilon^2(|x|) = \|x\|^2$) whereas w_i are the weights set to 0 for points with no match and to 1 otherwise. T is finally defined as the optimal transformation which best aligns "model" and "data" points while $a = [\theta, t_x, t_y]$ is a p-vector that carries T's parameters.

The proposed approach directly minimizes the model-data fitting error $E(a)$ via non-linear minimization. The LM algorithm is an optimization procedure that is specifically tailored to functions that could be formulated according to a sum of squared residuals, such as the just described error E. Indeed, this error function $E(a)$ can be easily dashed off as the sum of N_d residuals as:

$$E(a) = \sum_{i=1}^{N_d} E_i^2(a) \tag{3.6}$$

where the residual for the i^{th} data point is provided with the following formula:

$$E_i(a) = \sqrt{w_i} \min_j \varepsilon(|m_j - T(a;d_i)|) \tag{3.7}$$

Algorithm 2 LM-ICP Algorithm [32]

1 Function $a =$ **LMICP** $(\{m_j\}_{j=1}^{N_m}, \{d_i\}_{i=1}^{N_d}, a_0)$.
2 Set λ to an initial value.
3 Set $a = a_0$.
4 **Loop**
5 Compute $e_k = e(a)$. ▷ One closest-point computation.
6 Calculate J. ▷ p closest-point computations.
7 Update λ until $a_k = a - (J^T J + \lambda I)^{-1} J^T e_k$ reduces the error $\|e(a_k)\|^2$. ▷ One or more closest-point computations.
8 Set $a = a_k$.
9 **until** λ **is high.** ▷ So only a small gradient descent step lowers the error.

Thus, the error function $E(a)$ can be expressed in terms of residuals vector by writing:

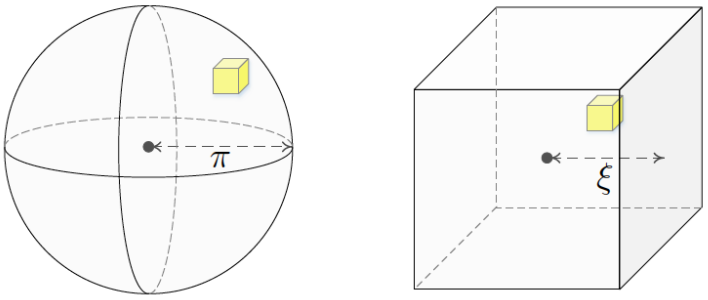
$$e(a) = \{E_i(a)\}_{i=1}^{N_d} \quad \Longrightarrow \quad E(a) = \|e(a)\|^2 \tag{3.8}$$

The LM algorithm integrates the gradient descent technique with the Gauss–Newton approach in order to minimize the objective function. The main objective of each iteration is that of selecting an update to the current estimate a_k -let's call it x-, so that by imposing $a_{k+1} = a_k + x$ the error $E(a)$ could be decreased. Introducing Jacobian matrix J, Gauss-Newton approximations and a bit of complex math it is possible to reach the final algorithm, that is explained in [32] and summarized in Alg. 2.

The derivatives of E are computed via finite differencing, paying an extra cost of p function evaluations for each internal loop. This leads to the assumption that the cost of each iteration raises by a factor of $1 + p$, for the most straightforward LM-ICP version. Nonetheless, LM typically needs fewer iterations to obtain a pre-determined accuracy, hence this threshold is an upper bound. Most of the times thus the lightening in the iterations number will greatly surpass the increase of the cost of every single iteration.
The fundamental contribution of Fitzgibbon is that of having established in this paper a non-linear optimization procedure for point cloud registration tasks, which is not limited however to a single purpose. Without losing too much in speed, the introduced multipurpose framework results in being simpler to program and leads to a more robust estimation, working overall better with respect to the conventional ICP algorithm.

3.1.3 Go-ICP

Among the number of registration methods proposed in literature, ICP [12] is the most famous algorithm for accurately performing an alignment between two 2D or 3D point clouds through a rigid transformation. Given an initial set of rotations and translations, ICP algorithm repeatedly acts two phases in sequence: it first builds correspondences between nearest points under the current transformation, then starting from the extracted correspondences it tries to estimate a new transformation, and this goes on until convergence.



(a) Rotation domain (b) Translation domain

Figure 3.3: **"SE(3) space parameterization for BnB"** (taken from [99]). (a) shows the rotation space $SO(3)$ parameterized in a solid ball with radius π, (b) depicts the translation that is assumed to be within a 3D cube $[-\xi, \xi]^3$. The octree structure is used to divide *(branch)* the domains: the yellow small boxes inside each diagram indicate a sub-cube.

Nonetheless, ICP presents a serious drawback, which is its vulnerability to local minima. Due to its inner iterative nature, it could effortlessly get stuck in a local minimum if an adequate initialization is not provided. If this is the case the estimation may greatly diverge from the optimal one, leading to a completely wrong transformation result. More faultfinding, there is no definitive nor well-grounded way to understand whether it is stuck in a local minimum or not. To deal with this issue, [99] is the first research to ever introduce a solution to the Euclidean registration problem that works at a global scale. The newly presented approach is always capable of formulating the solution that best aligns the two input point clouds, in an exact and globally-optimal fashion, until a predefined precision. This model takes the name of Globally-Optimal ICP (Go-ICP).

The Branch and Bound (BnB) algorithm, which provides a strong global solution illustrated in [56] to optimize NP-hard problems, is used by the Go-ICP algorithm and applied to 3D registration.
The overall structure of the proposed method can be summarized in this way:

> Use BnB to search the space of $SE(3)$
>
> Any time a greater solution is retrieved, call ICP starting from this solution to improve the objective function value. Utilize ICP's outcome as a correct upper threshold to carry on with BnB procedure.
>
> Loop until convergence.

As reported in [99], Go-ICP algorithm can be explained in detail by following Algs. 3 and 4. Briefly explicating the behaviour of this model, Yang et al. use a nested BnB search structure: an outer BnB explores the rotation space of $SO(3)$ and finds a solution for the bounds and the corresponding optimal transformation through the usage of a second inner BnB. Please find more information about BnB algorithm by looking at Fig. 3.3 and reading the relative paper [56].
In the implementation of the inner and the outer BnBs the authors propose to employ a best-first search strategy, in which each of the BnBs conserves a priority queue. When the difference between the finest error E^* achieved at a certain point and the lower limit $\underline{E}$ of the current cube is lower than a predefined threshold ε, the BnB procedure terminates.

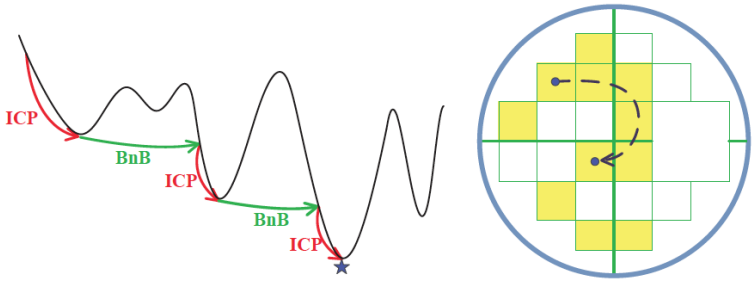


Figure 3.4: **"Relationship between ICP and BnB"** (taken from [99]). BnB and ICP jointly work to update the upper bounds during the search procedure.

Lines 13-14 of Alg. 3 show that every time the outer BnB retrieves a cube C_r with the upper limit lesser than the value of the function which is currently the best one obtained,

it calls the traditional ICP, setting its center of rotation to C_r. Being advised by this global BnB, ICP algorithm converges to local minima. Specifically, each local minimum it finds and surpasses has a lower error with respect to the previous ones, being able to reach in this way the global optimum minimum.

Algorithm 3 Go-ICP Algorithm [99]: BnB search for optimal registration in $SE(3)$

Input: "Data" and "Model" points; threshold ε; initial cubes $\mathscr{C}_r, \mathscr{C}_t$.
Output: Globally minimal error E^* and corresponding r^* and t^*.
Put $\mathscr{C}_r$ into priority queue Q_r.
Set $E^* = +\infty$.
Loop
    Read out a cube with lowest lower-bound $\underline{E}_r$ from Q_r.
    Quit the loop if $E^* - \underline{E}_r < \varepsilon$.
    Divide the cube into 8 sub-cubes.
    For each sub-cube C_r **do**
        Compute $\overline{E}_r$ for C_r and corresponding optimal t by calling Alg. 4 with r_0,
        zero uncertainty radii, and E^*.
        if $\overline{E}_r < E^*$ **then**
            Run ICP with the initialization (r_0, t).
            Update $E^*, r^*,$ and t^* with the results of ICP.
        end if
        Compute $\underline{E}_r$ for C_r by calling Alg. 4 with r_0, γ_r, and E^*.
        if $\underline{E}_r \geq E^*$ **then**
            Discard C_r and continue the loop.
        end if
        Put C_r into Q_r.
    end for
end loop

Fig. 3.4 exemplifies the synergistic relationship associating ICP and BnB. Due to the main property of ICP, that is its monotonically convergence to the current best error E^*, its local search trajectory is confined to encourage sub-cubes with little lower limits. By doing that, the global BnB and local ICP algorithms are combined together in the proposed Go-ICP approach. The global BnB assists the latter in jumping out of local minima by piloting its search of the next minimum to reach, whereas the speeds up the former's convergence by improving and rectifying the superior limit, thus ameliorating the final efficiency.

Go-ICP algorithm presented by Yang et al. provides overall good results for 3D registration tasks and it guarantees global optimality without taking into account the initialization. This method is particularly useful when an exactly optimal solution has to be achieved or a fine initialization cannot get reliably acquired. However, due to the slowness of the convergence, it is not suitable for real-time applications, such as the ones in robotics.

Algorithm 4 BnB search for optimal translation [99]

Input: "Data" and "Model" points; threshold ε; initial cube $\mathscr{C}_t$; rotation r_0; rotation uncertainty radii γ_r, so-far-the-best error E^*.
Output: Minimal error E^* and corresponding t^*.
Put $\mathscr{C}_t$ into priority queue Q_t.
Set $E_t^* = E^*$
Loop
    Read out a cube with lowest lower-bound $\underline{E}_t$ from Q_t.
    Quit the loop if $E_t^* - \underline{E}_t < \varepsilon$.
    Divide the cube into 8 sub-cubes.
    For each sub-cube C_t **do**
        Compute $\overline{E}_t$ for C_t as $\overline{E}_t = \sum_i max(e_i(R_{r_0}, t_0) - \gamma_{r_i}, 0)^2$, with r_0, t_0 and γ_r.
        if $\overline{E}_t < E_t^*$ **then**
            Update $E_t^* = \overline{E}_t, t^* = t_0$.
        end if
        Compute $\underline{E}_t$ for C_t as $\underline{E}_t = \sum_i max(e_i(R_{r_0}, t_0) - (\gamma_{r_i} + \gamma_t), 0)^2$, with r_0, t_0,
        γ_r, and γ_t.
        if $\underline{E}_t \geq E_t^*$ **then**
            Discard C_t and continue the loop.
        end if
        Put C_t into Q_t.
    end for
end loop

3.1.4 Fast Global Registration

Observing two point clouds P and Q, the task of finding a rigid transformation T that best aligns Q to P is carried out by [106] through the optimization of an objective function based on the correspondences between P and Q. A feature matching algorithm, completed before the optimization of the objective function, is used to acknowledge these correspondences. The key idea at the basis of this approach is that these extracted correspondences are not further recalculated during the optimization procedure. Thus, it is of paramount importance that the optimization is able to deal with very noisy correspondence sets.

Let $\kappa = \{(p,q)\}$ be the set of correspondences gathered by pairing up points from P and Q point sets. To provide this initial correspondence set κ, the authors suggest to use the Fast Point Feature Histogram (*FPFH*) feature [79], because it is very fast to be calculated (a fraction of a millisecond) and it also obtains a fine matching accuracy covering a wide amount of datasets. Being κ_I the set that accumulates all these correspondences, Zhou et al. decide then to use two tests to further enhance the amount of inliers in the correspondence set employed by the Fast Global Registration algorithm, namely *reciprocity* and *tuple* tests. Applying these tests, the final set to be considered is the one that contains only the correspondences that are compatible one another, and it is referred to as κ_{III}.

The objective of *FGR* algorithm is the optimization of the transformation T that minimizes the distances between matching points over the two point clouds, while ideally leaving apart fake correspondences from the κ set. In mathematical terms, it is possible to express the objective as follows:

$$E(T) = \sum_{(p,q)\in\kappa} \rho(\|p - Tq\|) \tag{3.9}$$

where $\rho(\cdot)$ denotes a robust penalty.

Algorithm 5 Fast Global Registration Algorithm [106]

Input: A pair of surfaces (P,Q).
Output: Transformation T that aligns Q to P.
Compute normals $\{n_p\}$ and $\{n_q\}$.
Compute FPFH features $F(P)$ and $F(Q)$.
Build κ_I by computing nearest neighbors between $F(P)$ and $F(Q)$.
Apply *reciprocity* test on κ_I to get κ_{II}.
Apply *tuple* test on κ_{II} to get κ_{III}.
$T \leftarrow I$, $\mu \leftarrow \delta^2$.
while not converged or $\mu > \delta^2$ **do**
    $J_r \leftarrow 0$, $r \leftarrow 0$.
    for $(p,q) \in \kappa_{III}$ **do**
        Compute $l_{(p,q)}$.
        Update J_r and r of Objective 3.11.
    Update T
    Every four iterations, $\mu \leftarrow \mu/2$.
Verify whether T aligns Q to P.

In order to obtain inflated computing proficiency, it is mandatory not to sample, corroborate, cut back or recalculate correspondences during the optimization, but a fair estimator ρ should automatically carry out validation and pruning steps without forcing supplementary computational costs. Having this in mind, the authors of [106] use a scaled Geman-McClure estimator, defined as:

$$\rho(x) = \frac{\mu x^2}{\mu + x^2} \tag{3.10}$$

Small residuals are disadvantaged according to a least squared metric, whereas the outliers are counterbalanced through the swift flattening out of the estimator. The domain across which the residuals show a noteworthy consequence on the objective is managed by the parameter μ.

The direct minimization of Objective 3.9 is not straightforward. This is the reason why Zhou et al. decide to making use of the "Black-Rangarajan duality between robust estimation and line processes" [15]. Concretely, being $\mathbb{L} = \{l_{p,q}\}$ a line process over the correspondences, they optimize over T and $\mathbb{L}$ the objective that follows:

$$E(T,\mathbb{L}) = \sum_{(p,q)\in\kappa} l_{p,q}\|p - Tq\|^2 + \sum_{(p,q)\in\kappa} \Psi(l_{p,q}) \tag{3.11}$$

In the equation above, $\Psi(l_{p,q})$ is a prior set to:

$$\Psi(l_{p,q}) = \mu(\sqrt{l_{p,q}} - 1)^2 \tag{3.12}$$

Optimizing Objective 3.11 returns a solution T that is also optimal for the original objective 3.9. The main benefit of the latter objective is that the optimization can be executed in an extremely efficient way by alternating between optimizing T and $\mathbb{L}$.
In this way, the alternating algorithm is guaranteed to converge. Alg. 5 summarizes the fast global registration algorithm described in the cited paper.

Many applications require aligning multiple scans to obtain a single model of a large scene. To solve this multi-way registration problem, existing approaches proceed in two sequential steps: (1) they firstly calculate pairwise alignments between pairs of scans and then (2) they try to synchronize these alignments to get a final global registration. Nonetheless, the pairwise alignment stage is computationally wasteful and could potentially yield a sub-optimal alignment due to local minima. The alternative approach developed in [106] consists of directly aligning all the input scans on the basis of matching point correspondences. Doing so, it is feasible to straight optimize a global registration objective over all surfaces.

The major benefit of *FGR* approach is the fact that it is significantly faster than prior approaches and state-of-the-art variants of ICP. The key reason is that it doesn't need to recompute correspondences: indeed, the majority of the time is spent on computing the *FPFH* features and building the input correspondences, but these operations are performed only once before the optimization, and the correspondences are never updated. By doing that, it basically accomplishes in a single stage what is commonly done in two. Thus, the optimization itself is extremely fast. In addition, it validates the algorithm only once, after the optimization procedure has reached a minimum, without considering the fact that it is much more robust to noise.

3.2 Feature-Learning Methods

Differently from the traditional optimization-based registration methods, feature-learning frameworks exploit deep neural networks (or any other available method) to acknowledge a strong feature correspondence search. The transformation matrix is lastly concluded through a one step estimation (e.g. *RANSAC*) without iteration.
Fig. 3.2(b) summarizes the main steps of these methods.

Their advantages lie in the robustness and the accuracy, achieved by virtue of the usage of deep learning approaches. Nevertheless, they need large training data and the registration performance greatly falls in unknown scenes, in the case they are particularly different from the training data.

In the following subsections, the state-of-the-art feature-learning methods are handed out: specifically, subsection 3.2.1 focuses on the *PPF-FoldNet* [25] model, subsection 3.2.2 concentrates on the *IDAM* [57] approach, subsection 3.2.3 explains the *DCP* [91] architecture while subsection 3.2.4 finally strengthens the *FRR Global Registration* approach, using *FPFH* features [79] followed by the (classic) *RANSAC* [31] method.
This last framework will be reproduced in this thesis work, as explained in Chapter 5.

3.2.1 PPF-FoldNet

Deep learning techniques for the extraction of 3D local features are affected by one or more out of the following problems:

1. Being supervised and needing a huge amount of labels
2. Being sensitive to 6 *DoF* rotations
3. Demanding noteworthy hand-crafted preparation of the input data
4. Lacking of good enough performance

The authors of [25] try to jointly address all these issues by presenting *PPF-FoldNet*, an unsupervised feature-learning network capable of achieving greatly precise performance over the 6 degrees of freedom. *PPF-FoldNet* is straight fed with input point clouds, and it is designed to be capable of considering the point sparsity and working properly with density dissimilarities. This deep neural network is proved to be rotational invariant by virtue of the point pair feature (*PPF*) [13] which embeds local 3D properties into patches. *PPF* lives in a 4D space: hence, it is a not straightforward task to picture it. While simple solutions such as PCA would work, the authors of the presented paper prefer to rely on a more geometrically meaningful solution.
Therefore, as opposed to a previous approach, *PPFNet* (described in [26]), Deng et al. propose an original variant: the collection of the 4D *PPFs* is fed to an end-to-end auto-encoder (AE), trained to autonomously reconstruct the *PPFs*.

The input point cloud is composed by a point set $X = \{x_i \in \mathbb{R}^6\}$, in which every point is enhanced with its normal, i.e., tangent space, $n \in \mathbb{R}^3 : x = \{p, n\} \in \mathbb{R}^6$. From these points the local patches are computed: they can be considered as a subset of the input $\Omega_{x_r} \subset X$ center around a reference point x_r. These patches are finally fed to the neural network as input.

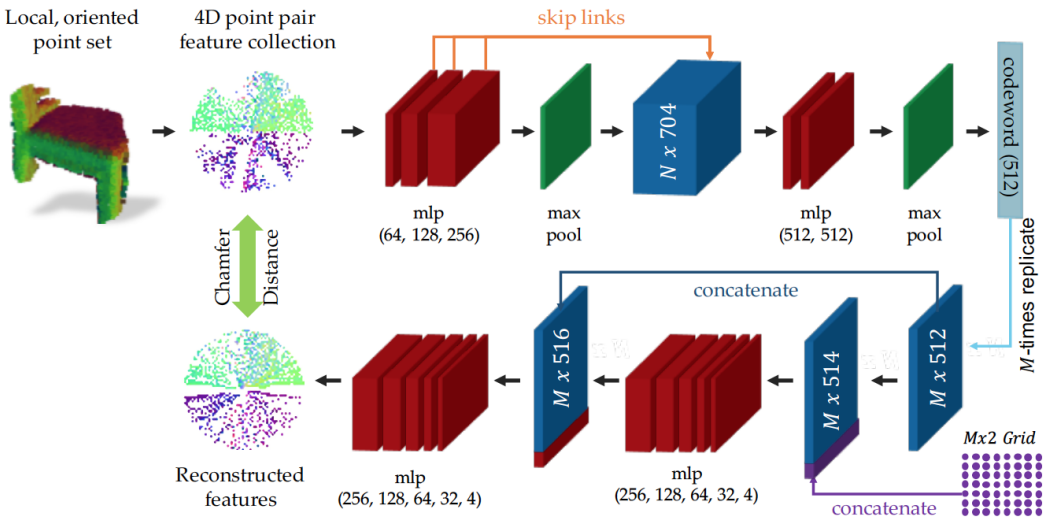


Figure 3.5: **"PPF-FoldNet Architecture"** (taken from [25]). From the input point cloud local patches are first of all extracted, and then converted into *PPF* models. Thus, they are fed to the encoder part of the network, which squeezes them into *codewords*, which contains the most crucial and important information. Lastly, starting from these compressed *codewords* the decoder part of the neural network searches a way to rebuild the initial *PPFs*.

The overall architecture of the network is shown in Fig. 3.5. As mentioned before, *PPF-FoldNet* depends on the idea of learning a rotation-invariant representation of the input point cloud, named *PPFs*, through the usage of autoencoders. Doing that, the whole point is that of getting a low dimensional embedding space which shows to be accurately invariant. Specifically, after the input layer the proposed architecture is composed by a three-layer Multi-Layer Perceptron (MLP) followed by a max pooling layer, whose aim is that of combining the individual features into a more global one. The local features are thus compounded with the just obtained global one through the usage of skip connections, obtaining in this way a stronger representation. Lastly, a second two-layer MLP takes as input this just-formed features concatenation in order to obtain the final *codeword*, which is the encoding employed as the local descriptor assigned to the point at the center of each extracted patch.

At this point the part of the network employed as a decoder works in order to rebuild the initial *PPFs* representation starting from a single *codeword*. This procedure constrains on the other side the *codeword* itself to refine the most characteristic information from the input of the network, which is encoded at a higher dimensional space. The folding performed by the decoder is a without any doubt a non-linear operation. Thus, it is performed by two MLPs one after the other: the first aims at collapsing the *codewords* to obtain a warped grid, which is then attached to the *codewords* themselves, while the second takes the output of the previous MLP and works in order to rebuild the original *PPFs* which served as input to the network. Overall, the folding could be considered as extremely important not only for simplifying the decoding operation, but also and most importantly for letting the network itself being more explicable.

To summarize, *PPF-FoldNet* is built in such a way that it doesn't need supervision to reach the goal it is designed for. Built above its immediate forefathers such as *PointNet*, *FoldingNet* and *PPFNet*, it takes the best from all of them.
A part from presenting a very useful rotation invariance, extensive experiments conducted by the authors with several datasets, such as 7-Scenes [82], SUN3D [98] and many others, prove that this framework outperforms all the state-of-the-art methods, including the ones learned in a traditional supervised fashion. In this sense, [25] offers a new encouraging perspective to one of the major issues in the 3D features extraction phase, that is its unsupervised performing.
The complexity of the model (with three MLPs between the encoder and the decoder working together) and the non real-time performance don't make it a good candidate for robotic applications though.

3.2.2 IDAM

[57] proposes Iterative Distance-Aware Similarity Matrix Convolution Network (*IDAM*), an original learning-based method for extracting features starting from 3D point clouds which may in part overlay.
The idea behind the model is that of building an iterative approach that combines both geometric and distance features extracted from the input point sets, being able in this way to obtain improved results with respect than using only one of them. Furthermore, the authors of [57] introduce a module that is capable of calculating the similarity score on the basis of the totality of the features extracted from the two point clouds. All of this is implemented through an isolated convolutional block based on a similarity matrix which is fed with an input represented in feature and classical geometry space.

Mathematically speaking, the aim of 3D features-learning registration methods is that of finding the ground truth rigid body transformation (R^*, t^*) that (almost) perfectly aligns an input source point cloud S, composed by N_S points, to a target one T, constituted of N_T points.
In order to obtain this result, a matching correspondences' set between the source and target point sets has to be retrieved. In the literature of feature-learning approaches, most of them try to employ the scalar product (or $L2$ distance) of the features extracted from the point clouds as a similarity indicator. In doing so, they tend to straight take the ones with the highest (or lowest for $L2$) value, but this leads to two major drawbacks. Firstly, one point of the source point cloud S could in principle have more than one correspondences in the target point cloud T, thus a single-pass matching process could provoke a mistake, since due to randomness the selected points may not be the right corresponding ones. Secondly, a straight calculation of the similarity between the extracted features cannot properly recognize the similarity between two points since the matching procedure is identical even for (very) distinct point pairs.
[57] therefore suggests to tackle these problems by using a similarity matrix convolution module that is capable of retrieving the correspondences between matching points.
Fig. 3.6 shows the overall architecture of the proposed model.

Denoting with p_i the i^{th} point of the source point cloud and with q_j the j^{th} point of the target point cloud, and assuming to have already extracted the features $u^S(i)$ for $p_i \in S$

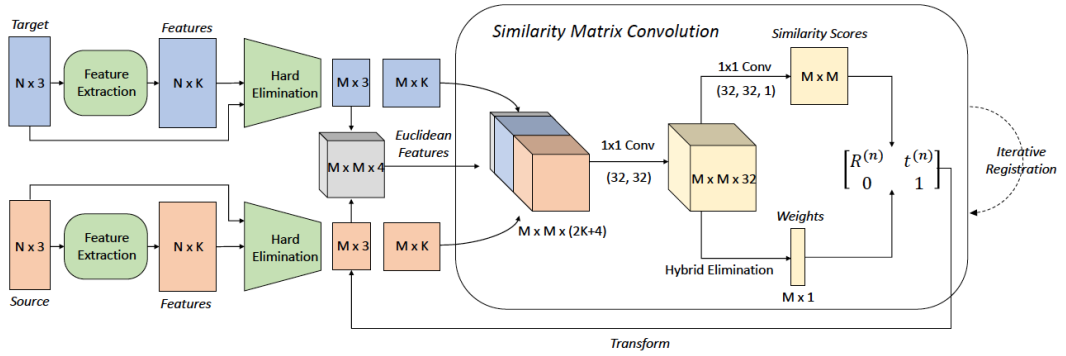


Figure 3.6: **"IDAM Registration Pipeline Architecture"** (taken from [57]). It is possible to notice the presence of the distance-aware similarity matrix convolution, useful for retrieving point correspondences. Moreover, worthy of notice are the *hard point elimination*, which autonomously removes individual points which are not expected to be certainly coupled, and the *hybrid point elimination* which instead utilizes the information coming from both the points of the considered pair to calculate weights that will be then employed in the optimization step.

and $u^T(j)$ for $q_j \in T$, both with dimension K, the authors propose to identify a distance-augmented feature tensor described at iteration n as follows:

$$T^{(n)}(i,j) = \left[u^S(i); u^T(j); \|p_i - q_j\|; \frac{p_i - q_j}{\|p_i - q_j\|} \right] \tag{3.13}$$

where $[\cdot;\cdot]$ indicates the concatenation operation. The $(2K+4)$-dimensional vector at the (i,j) position of the feature vector $T^{(n)}$ is calculated using geometric and Euclidean features for the point pair (p_i, q_j) together. This feature tensor can thus be thought at as a $(2K+4)$-channel 2D image.
With the aim of finding a similarity score for each of the considered point pairs, the authors decide to employ a number of 1×1 2D convolutional layers on $T^{(n)}$, obtaining in this way as output an image of the same spatial size of the last layer but with a single channel. This resembles the application of a Multi-Layer Perceptron (MLP) on each location of feature vector $T^{(n)}$. Lastly, they decide to implement a softmax function on every row of the output image in order to obtain the similarity matrix, named $S^{(n)}$. Thus, $S^{(n)}(i,j)$ stands for the *similarity score* for p_i and q_j (the higher is its value the more similar are the points).
Moreover, since each row of $S^{(n)}$ can be considered as a normalized probability distribution over the whole feature vector T for some $p \in S$, the authors highlight that one could think about $S^{(n)}(i,j)$ as the probability that q_j is p_i's correspondence. Thus, in order to retrieve the correspondence pairs they decide to compute the maximum value of each row of the similarity matrix $S^{(n)}$.
By performing these steps, one could be able to obtain a set of correspondence pairs $\{(p_i, p_i') \,|\, \forall p_i \in S\}$, whose usage may lead to puzzle out the optimization problem described here, in order to learn the rigid transformation $(R^{(n)}, t^{(n)})$ that is the final goal of the 3D registration:

$$R^{(n)}, t^{(n)} = \underset{R,t}{\operatorname{argmin}} \sum_i \|Rp_i + t - p_i'\|^2 \tag{3.14}$$

$R^{(n)}$ and $t^{(n)}$ are finally employed to rotate and translate the source point cloud to a new position, in order to prepare it for the following iteration. The last (R^*, t^*) pair is estimated through the compound of the intermediate $(R^{(n)}, t^{(n)})$ computed throughout all the loops.

Since it is computationally expensive to put in application a convolution over the $N_S \times N_T \times (2K+4)$ tensor, because N_S and N_T counts normally more than a thousand points, the authors introduce a two-stage point elimination technique to achieve a nice trade-off between performance and efficiency. The first stage, named *hard point elimination*, autonomously removes the greater number of individual points which are not expected to be coupled with certainty, while the second, called *hybrid point elimination*, removes correspondence pairs by assigning lower weights to those pairs that are likely to be false positives. With the introduction of the elimination weights w_i through this two-stage point elimination technique, it is possible to obtain $(R^{(n)}, t^{(n)})$ with a marginally dissimilar objective function with respect to Equation 3.14:

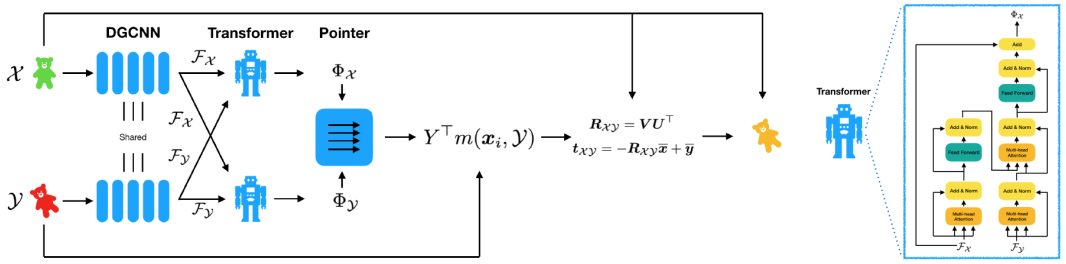
$$R^{(n)}, t^{(n)} = \underset{R,t}{\operatorname{argmin}} \sum_i w_i \|Rp_i + t - p'_i\|^2 \tag{3.15}$$

Theoretically, the second stage, *hybrid point elimination*, should be able to remove the point pairs which are inaccurate due to noise or imperfection, allowing to achieve in this way a better accuracy on the estimation of $R^{(n)}$ and $t^{(n)}$.
The two-stage elimination technique yields a visible speed up in the convergence of the model, which turns out to be remarkably faster than the state-of-the-art learning-based approach, while still achieving more or less the same accuracy and yet being less computationally demanding. Moreover, as shown in the experiments provided by Li et al., the proposed framework is well suited with all the existing feature extraction methods, including the learning-based, i.e., *FPFH* [79] features, and the conventional ones, such as Graph Neural Network (*GNN*) [60, 92, 96] features.

3.2.3 DCP

Having in mind the objective of tackling the major problems of ICP algorithm, predominantly related to the local minima issue, [91]'s authors introduce an original feature-learning approach, named Deep Closest Point (*DCP*), which takes as input two point clouds and tries to estimate the rigid transformation that best aligns them. The proposed model is constituted of three different blocks: (1) a DGCNN [92] is used to recognize correspondences between two input point clouds, that are mapped to a rotation-invariant representation; then, (2) an attention-based module estimates a shadow matching between point pairs extracted from the input point clouds; and lastly, (3) a Singular Value Decomposition (SVD) layer forecasts the final transformation that aligns the source to the target point cloud.
Fig. 3.7 depicts the *DCP* architecture, and the following paragraphs explain in detail each of the related parts.

Initial Features. The goal of the first part of the *DCP* pipeline is that of mapping the input point clouds X and Y to a rotation-invariant representation, in order to retrieve correspondences between the point pairs. Due to the fact that the authors expressly want to transform, through the usage of mapping m, the input into a per-point lower dimensional space to build the final transformation, they look for a feature *per point* in the input point



(a) Network architecture (b) Transformer module

Figure 3.7: **"DCP Registration Pipeline Architecture"** (taken from [91]). This figure shows the *DCP* architecture, remarking all its important parts, namely *DGCNN*, Transformer + Attention Module and SVD layer.

clouds instead of a feature *per cloud*. This is the logic behind the employment, by the authors of [91], of a *DGCNN* [92], which is capable of learning a representation that includes both local geometry properties and global shape information. Specifically, given a set of points X, *DGCNN* builds a k-NN graph G, starting from which it puts in application non linearities in order to derive edge-wise values from boundary points. Lastly, it implements for every layer an edge-wise aggregation, e.g. either max or $\sum$.
In formulas, let x_i^l be the representation in the lower dimensional space of the i^{th} point in the l^{th} layer, and h_θ^l be a non-linear function in the l^{th} layer of a shared Multi-Layer Perceptron (MLP). The forward mechanism of *DGCNN* is thus described as:

$$x_i^l = f(\{h_\theta^l(x_i^{l-1}, x_j^{l-1}) \forall j \in N_i\}) \tag{3.16}$$

where N_i indicates the set of neighbors of the i^{th} edge in graph G. Therefore, the fact that the most interesting features for the final alignment are simultaneously gathered starting from global and local properties learned motivates the usage of *DGCNN* in the proposed pipeline.

Attention and Pointer Generation. Moreover, Wang et al. try to further refine the extracted features by forcing them to be *task-specific*, which means trying to map them to the lower dimensional space on the basis of the joint characteristics of the input point clouds X and Y, instead of trying to transform the point clouds independently. Having this in mind, they propose a module able to apprehend *self-attention* and *conditional-attention* information to retain co-contextual details. This block is designed so to determine a function $\phi : \mathbb{R}^{N\times P} \times \mathbb{R}^{N\times P} \longrightarrow \mathbb{R}^{N\times P}$, where P is the dimension of the embedding space in which the input point clouds are mapped:

$$\begin{aligned} \Phi_X &= F_X + \phi(F_X, F_Y) \\ \Phi_Y &= F_Y + \phi(F_Y, F_X) \end{aligned} \tag{3.17}$$

where F_X and F_Y are the embeddings generated through the usage of the *DGCNN*.
The authors decide ϕ to be an asymmetric function implemented by means of a Transformer [88].
Moreover, a novel idea introduced in this paper to tackle the issue of ICP related to the matching estimate being totally wrong is that of assigning to each $x_i \in X$ a probability

vector over the whole point cloud Y, described as:

$$m(x_i, Y) = softmax(\Phi_Y \Phi_{x_i}^T) \tag{3.18}$$

where $\Phi_Y \in \mathbb{R}^{N \times P}$ is the embedding of the point cloud Y learnt though the attention module, while Φ_{x_i} indicates the i^{th} row of the matrix Φ_X obtained by the latter module itself. It is possible to visualize $m(x_i, Y)$ as a "*soft pointer*" from every x_i to Y's points.

SVD Module. The last block presented in the *DCP* pipeline draws out the rigid transformation from the soft matching thoroughly described in the previous paragraph.
The authors propose the employment of the *soft pointers* to compute an average of the correspondence point in Y for every point in X, described as follows:

$$\hat{y}_i = Y^T m(x_i, Y) \in \mathbb{R}^3 \tag{3.19}$$

where $Y \in \mathbb{R}^{N \times 3}$ is a matrix including the Y's points. Lastly, the rigid transformation, which comprises the rotation R_{XY} and the translation t_{XY}, is computed by using a singular value decomposition (SVD) depending on the mapping $x_i \mapsto \hat{y}_i, \forall i$, expressed as:

$$R_{XY} = VU^T \quad t_{XY} = -R_{XY}\tilde{x} + \tilde{y} \tag{3.20}$$

where $U, V \in SO(3)$, while $\tilde{x}$ and $\tilde{y}$ are the centroids of X and Y defined in this way:

$$\tilde{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{and} \quad \tilde{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{3.21}$$

Loss. Integrating all together the previously described modules, it is possible to obtain a model that maps the input point clouds X and Y to a rigid transformation $[R_{XY}, t_{XY}]$ which aligns them. The first two blocks are based on two neural networks, whose weights should be learned during the training step.
The loss function used is the following:

$$\mathscr{L} = \|R_{XY}^T R_{XY}^g - I\|^2 + \|t_{XY} - t_{XY}^g\|^2 + \lambda \|\theta\|^2 \tag{3.22}$$

where g indicates the ground truth. The first two terms describe a distance on $SE(3)$, while the third one indicates the Tikhonov regularization [95] of the *DCP* parameters θ, which aims at reducing the complexity of the network itself.

Beyond providing a state-of-the-art registration technique, the authors of [91] demonstrate the efficiency of the model and its performance, which outperforms that of ICP and its extensions. By combining the usage of a *DGCNN* and an attention module, *DCP* is capable of accurately learning in a single pass the correspondences from which the best alignment between the source and the target point clouds may be retrieved. This estimated transformation could be further enhanced by iteration or refined via traditional ICP.
Unfortunately, the complexity of this model represents a clear limit, while the performance evaluated by the authors without considering the usage of the attention module, thus with a "lighter" model, lacks in precision and accuracy. Moreover, the computational time for dealing with point clouds with a large number of points grows exponentially, leading to a non-applicability for real-time application.

3.2.4 FPFH-RANSAC Registration

In this Subsection the FPFH-RANSAC Global Registration (*FRR*) approach, as indicated in the procedure available in [107], is going to be explained in detail. It uses FPFH [79] features and RANSAC [31] algorithm for the registration. Besides its low complexity, the method obtained by combining them achieves good results, and this is the reason why it is the algorithm chosen to be implemented in this thesis work, as the representative sample of the feature-learning methods. Before going on with the description of the presented framework, one last note that will be also repeated later is that RANSAC is not guaranteed to converge (unless increasing the number of iterations), thus it may happen that for some point clouds alignments the performance won't be in real-time. But for this thesis work, this approach is taken into account in order to do a fair comparison with the *FGR* optimization-based method described in Subsection 3.1.4 and the *FMR* end-to-end learning-based one that will be eplainec in Subsection 3.3.6.

FPFH

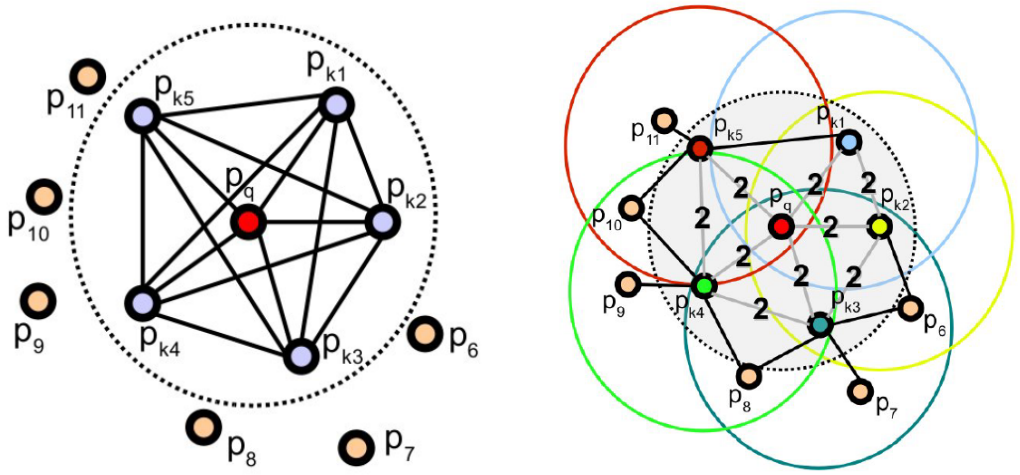
As introduced in [81], Point Feature Histograms (*PFH*) reveal the local features that constitute the essential characteristics of a point p. They are calculated starting from the compound of geometrical associations between all the nearest k neighbors of p. These features include, in their original version, $\langle x, y, z \rangle$ 3D point coordinates and estimated surface normals $\langle n_x, n_y, n_z \rangle$, but this doesn't mean that they couldn't contain also information regarding other properties, e.g. curvature, second order moment invariants, etc.
The calculation of a *PFH* at a point p can be explained as follows (following the details provided in [79]):

1. For each point p, all its neighbors within a sphere with a certain given radius r are chosen (k-neighborhood).

2. For every points pair (p_i, p_j), with $i \neq j$, in p's k-neighborhood and their computed normals (n_i, n_j), a Darboux [9] *uvn* frame $(u = n_i, v = (p_j - p_i) \times u, w = u \times v)$ is defined, and the angular behaviour of n_i and n_j is calculated as follows:

$$\begin{aligned} \alpha &= v \cdot n_j \\ \phi &= \frac{u \cdot (p_j - p_i)}{\|p_j - p_i\|} \\ \theta &= arctan(w \cdot n_j, u \cdot n_j) \end{aligned} \tag{3.23}$$

Fig. 3.8a depicts the diagram of the *PFH*'s region of influence for a query point (p_q). In the figure below, p_q is drawn in red and situated in the centre of a circle (a sphere in 3D) with a given radius r. The totality of its k neighbors, that are the points having a distance from the center lower with respect to the radius itself, are fully-connected, forming a mesh.
Given a point cloud P with n points, the computational complexity for computing the *PFH* can be evaluated as $O(n \cdot k^2)$, where k denotes the number of closest points for every $p \in P$. When the number of points is particularly high, the computation of *PFHs* may become extremely expensive, thus not allowing real-time or near real-time tasks.



(a) *"The influence region diagram for a Point Feature Histogram".*

(b) *"The influence region diagram for a Fast Point Feature Histogram".*

Figure 3.8: **"PFH"**'s (a) and **"FPFH"**'s (b) influence region diagrams, both taken from [79].

Thus, [79] introduces a simpler approach, named Fast Point Feature Histograms (*FPFH*), which forces the complexity of the features computation to be $O(n \cdot k)$. In this way they make its usage possible in real-time applications, without losing most of the *PFH*'s characteristics. In order to disentangle the calculation of the features, the authors decide to perform the following steps:

1. In a first step, denoted as Simplified Point Feature Histogram (*SPFH*), the connections between p's neighbors and the point itself are calculated for every query point p, according to Eq. 3.23;

2. Secondly, the k neighbors are computed for every single point, and the previously calculated *SPFH* features are then employed by the authors to refine the ultimate histogram of p, named *FPFH*, as follows:

$$FPFH(p) = SPFH(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k} \cdot SPFH(p_k) \tag{3.24}$$

   where the weight ω_k denotes the distance between query point p and one of its neighbors p_k in a particular metric space.

Fig. 3.8b shows a diagram of the *FPFH*'s region of influence. As one can see by looking at the figure above, every query point, marked in red, is associated just to its immediate k-neighbors, that lie within the circle colored in gray. Each of these selected points in the gray sphere is further associated to other nearby points, resulting in a weighted histogram that forms the *FPFH*. It is worth noting that some pairs, highlighted with a 2 in the picture above, are included twice in the *FPFH* calculation.

Summarizing, the authors of [79] highlight the differences between *PFH* and *FPFH* according to the following points:

1. In *FPFH* not all p_q's neighbors are connected, therefore the resulting histogram could have insufficient information to learn the whole structure of the point itself.
2. While *PFH* is very accurate in shaping the model on the area surrounding p_q, *FPFH* may contain extra point pairs with a distance from the given point higher than the radius r's length that shouldn't be there.
3. Due to the final weighting strategy the *FPFH* could anyway be capable of learning some point pairs in the neighborhood of the point by merging *SPFH* values.

Besides being extremely fast, the Fast Point Feature Histogram (*FPFH*) features are known for their rotation invariance and their accurate strength, which both make them extremely suitable for the search of correspondences between two input point clouds. Nonetheless, their robustness to very noisy point cloud data coming from predominantly from ToF cameras still has to be completely proved.

RANSAC

[31] introduces a novel algorithm for learning a model's parameters on the basis of experimental data, named Random Sample Consensus (*RANSAC*). The key idea behind this method is clearly contrary with respect to the one at the basis of traditional techniques: instead of being fed with a huge amount of data to achieve a solution which is deprived in a second moment of all the worthless points, *RANSAC* tries to start with a very limited number of data entries in order to expand in a second moment the initial solution whenever it is feasible.

The *RANSAC* algorithm is declared by the authors of [31] in an official manner in accordance to what is reported below:

> "Given a model that requires a minimum number of n data points to instantiate its free parameters and a set of data points P such that the number of points in P is greater than n [$\#(P) \geq n$], randomly select a subset S_1 of n data points from P and instantiate the model".
>
> "Use the instantiated model M_1 to determine the subset S_1^* of points in P that are within some error tolerance of M_1. The set S_1^* is called the *consensus set* of S_1".
>
> "If $\#(S_1^*)$ is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use S_1^* to compute (possibly using least squares) a new model M_1^*".
>
> "If $\#(S_1^*)$ is smaller than t, randomly select a new subset S_2 and repeat the above process. If, after some predetermined number of trials, no *consensus set* with t or more members has been found, either solve the model with the largest *consensus set* found, or terminate in failure".

The algorithm just described presents three unstated parameters, that are going to be explained in detail in the following paragraphs: (1) the *error tolerance* employed to understand if a point is well-matched for a specific model or not, (2) the *number of subsets* and (3) the *threshold* t, that indicates the amount of suitable points above which one could certainly state that the selected model is correct.

Error Tolerance. Given a certain datum, its deviation with respect to a found model can be represented as a function of the errors associated with both the datum and the model itself. Thus, if the found model is a straightforward data points' function, computing proper boundaries for the error tolerance in an analytical way might be feasible. Nonetheless, this manageable procedure is most of the times not feasible: for the cases in which it is not, the authors of the paper suggest to experimentally evaluate the boundaries on the tolerance. Furthermore, being the expected deviation of a datum from a given model a function of the datum itself, the error tolerance should in principle diverge when taking two different datums. Nevertheless, the variation in error tolerances turns out to be significantly lower with respect to the total error's value, therefore considering a unique tolerance threshold for all the available data is frequently adequate.

Maximum number of subsets. In order to conclude the searching of new subsets of P, one could base the decision on the expected number of attempts k necessary to include n good data points in a single subset. If at least one of the random selections has to be a set of n data points without any error, with a certain probability z, one has to presume to perform at least k iteration, with n data points for each iteration. Given the probability w that any chosen data point is under the error tolerance value for a specific model, it is possible to write:

$$k = \frac{log(1-z)}{log(1-b)} \qquad \text{where } b = w^n \tag{3.25}$$

This means that to achieve a 90% guarantee of getting at least one selection completely discharged of errors it must hold that:

$$k = \frac{log(0.1)}{log(15/16)} = 35.7 \tag{3.26}$$

Consensus Set Threshold. The threshold t is employed to understand whether a suitable *consensus set* has been found. Here, "suitable" means that an adequate number of points for a specific n-subset of P has been selected, thus allowing the procedure to stop. Therefore, the criterion at the basis of the choice of t's value should take into account two separate issues: (1) firstly, starting from the input data the correct model has to be retrieved, and (2) secondly an adequate amount of points has to be found in order to allow the model parameters to refine the computed estimations.
Being y the probability that any input data point lies below the error tolerance's value of an incorrect model, to make sure that the ultimate *consensus set* is not compatible with this incorrect model it is desirable to have a very low y^{t-n}'s esteem. There isn't an established way to accurately decide the value of y, but it is rational to take for granted the fact that it has to be lower than w, being w the *a priori* probability that every data point can be found under the error tolerance's threshold of the correct model.

Just summarizing, *RANSAC* is capable of accurately learning a model's parameters even in the case of particularly noisy input data. Unfortunately though, as described in [93], there is no upper bound on the time it takes to compute these parameters. Thus, when the number of iterations is restricted or bounded the solution may be sub-optimal or partially wrong, unable to precisely fit the data. In this sense *RANSAC* supplies a nice trade-off: enlarging the number of iterations performed in the procedure leads to a significant grow also of the probability of finding a proper model.

3.3 End-to-End Learning-Based Methods

End-to-end learning-based methods try to find a solution for the registration problem with an end-to-end approach based on the usage of neural networks. The key point at the very basis of the methods is that of mapping the registration problem into a regression one. The neural network is fed with two input point clouds, and its aim is their alignment through the application of a transformation matrix, that is indeed the output of the model. Therefore, the transformation estimation is actually embedded into the neural network optimization, and this is the main difference with respect to the previously defined feature-learning methods, whose focus is point feature learning instead. Fig. 3.2(c) summarizes the foremost steps of these algorithms.

The main advantages of these frameworks lie in their capability of designing neural network architectures specifically for registration tasks, and most importantly to jointly use the traditional mathematical principles and the strength of relatively new deep neural networks. On the other side, the major drawbacks are that the transformation parameters estimate happens as a black box and that they do not take into account the local structure information.

In the following, the state-of-the-art end-to-end learning-based methods are extended: precisely, Subsections 3.3.1 and 3.3.2 concentrates on the *PointNetLK* [5] model and its variant with *Awe-Net* [51], Subsection 3.3.3 centralizes on the *PointVoteNet* [42] approach, Subsection 3.3.4 nourishes the *DGR* [23] model, Subsection 3.3.5 explains the *3DRegNet* [70] architecture while Subsection 3.3.6 finally strengthens the *FMR* [47] framework, that (together with *3DRegNet*) is the one that will be replicated in this thesis work, as explained in Chapter 5.

3.3.1 PointNetLK

[76] is the first paper to introduce the usage of a DNN fed with point clouds for classification and segmentation. The architecture of the network proposed in this paper, named *PointNet*, is able to obtain good performance, notwithstanding the ease of its structure. In this sense, *PointNet* represents a revolution in the way in which deep learning models approach point clouds, which are intrinsically unstructured.
The novel idea behind [5] is that of extending the study about *PointNet*, trying to employ the alignment scheme initially thought for images also to the domain of point cloud registration tasks. The newly introduced method, called *PointNetLK*, uses the traditional Lukas & Kanade (LK) algorithm [61] to build a recurrent neural network that is merged with the already-existing *PointNet* architecture.

It is given the *PointNet* function $\phi : \mathbb{R}^{3\times N} \mapsto \mathbb{R}^K$, built in such a way that, starting from an input point cloud $P \in \mathbb{R}^{3\times N}$, $\phi(P)$ outputs a feature vector of dimensionality K. This function employs a Multi-Layer Perceptron (MLP) to each point belonging to P, so that the ultimate output dimension of each point is K. Subsequently, a symmetric (maximum or average) pooling layer is applied in order to gather as final output the K-dimensional global feature vector. Fig. 3.9 exhaustively depicts the architecture of the proposed model.

The optimization problem is thus described according to what follows. Given the source P_S and the target P_T point clouds, the objective is to find the rigid transformation $G \in$

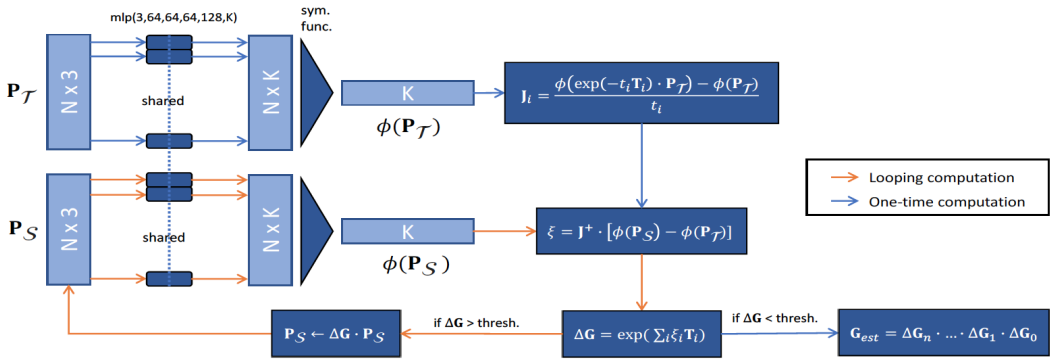


Figure 3.9: **"PointNetLK Architecture"** (taken from [5]). The figure depicts the way in which the source P_S and the target P_T input point sets are fed to a shared MLP, immediately followed by a pooling layer that outputs the feature vectors $\phi(P_S)$ and $\phi(P_T)$ for the source and the template, respectively. The best transformation parameters, learnt through the learning procedure, are employed to gradually refine the pose of P_S. Once these optimal parameters have been found, at the end of the looping computation the global feature vector $\phi(P_S)$ is lastly recomputed.

$SE(3)$ that best aligns P_S to the template P_T point cloud.
The estimated transformation G can be described in terms of an exponential map as:

$$G = exp(\sum_i \xi_i T_i) \qquad \xi = (\xi_1, \xi_2, ..., \xi_6)^T \tag{3.27}$$

where T_i create the exponential map with transformation (rotation and translation) parameters $\xi \in \mathbb{R}^6$. The 3D point cloud registration task can therefore be expressed as the problem of finding G so that it holds:

$$\phi(P_T) = \phi(G \cdot P_S) \tag{3.28}$$

where $(\cdot)$ indicates the transformation of P_S through the application of G. This equation is exactly equivalent to the one being considered in the traditional LK algorithm for 2D images. In that case though, the whole point is that of deforming the source image so that the dissimilarities between the intensity of the distorted source's and target's pixels are reduced as much as possible.

A further important proposal that the authors of [5] take from the LK procedure is the concept of Inverse Compositional (IC) [10], that is of paramount importance in order to reduce the huge computational cost of the LK algorithm itself (that is caused by the fact that at each iteration the image Jacobian should be re-calculated).
The key point of IC is to swap the role of the source and the target point clouds: by doing so, the Jacobian is computed for the target rather than for the source. Thus, this computation is performed just one time before the optimization procedure starts.
Having introduced the IC algorithm, it is now possible to reverse the Objective 3.28 obtaining the following equation:

$$\phi(P_S) = \phi(G^{-1} \cdot P_T) = \phi(P_T) + \frac{\partial}{\partial \xi}[\phi(G^{-1} \cdot P_T)]\xi \tag{3.29}$$

where $G^{-1} = exp(-\sum_i \xi_i T_i)$. At this point it is conceivable, following LK algorithm, to define the Jacobian $J \in \mathbb{R}^{K \times 6}$ as:

$$J = \frac{\partial}{\partial \xi}[\phi(G^{-1} \cdot P_T)] \tag{3.30}$$

Aoki et al. decide to use a slightly different version of the classical LK algorithm, by computing J using a stochastic gradient approach. In particular, every column J_i of the Jacobian matrix can be written according to:

$$J_i = \frac{\phi(exp(-t_i T_i) \cdot P_T) - \phi(P_T)}{t_i} \tag{3.31}$$

where t_i are the infinitesimal perturbations of the transformation parameters ξ.
This original procedure, which forces J to be calculated only once for the template point cloud while warping the source one during the iterative process, allows to combine the usage of the IC-LK algorithm with the *PointNet* features to tackle the point cloud registration task.
It is now immediately feasible solving Objective 3.29 for ξ as:

$$\xi = J^{+}[\phi(P_S) - \phi(P_T)] \tag{3.32}$$

where J^{+} is the Moore-Penrose [49] inverse of J. The optimization algorithm is in practice based on an iterative calculation of the optimal transformation parameters using Eq. 3.32, and results in the update of the source point cloud P_S as follows:

$$P_S \leftarrow \Delta G \cdot P_S \qquad \Delta G = exp\left(\sum_i \xi_i T_i\right) \tag{3.33}$$

The ultimate estimate G_{est} is composed by $\Delta G_i, \forall i$, computed throughout the entirely looping process:

$$G_{est} = \Delta G_n \cdot ... \cdot \Delta G_1 \cdot \Delta G_0 \tag{3.34}$$

The loop stops on the basis of a minimum threshold for ΔG.

Unlike many variants of ICP, the novel proposed approach doesn't require costly computation of point correspondences. Thus, this leads to several advantages in terms of accuracy, robustness to initialization and computational efficiency. Indeed, being n the dimensionality of the point clouds, *PointNetLK* has a complexity $O(n)$, even if the computation could be considerably sped up with a GPU implementation, since the network is greatly parallelizable. Besides being also a very generalizable approach, *PointNetLK* results not to be able to work in real-time. Nonetheless, as the authors themselves suggest, it represents a salient and innovative idea for point cloud registration methods as it provides a constructive technique that is highly differentiable, generalizable and more importantly applicable to other deep learning frameworks.

3.3.2 PointNetLK + Awe-Net

Based on the previous work of [5], [51]'s authors introduce a new framework with the aim of learning an accurate rigid transformation in a faster and more generalized way.

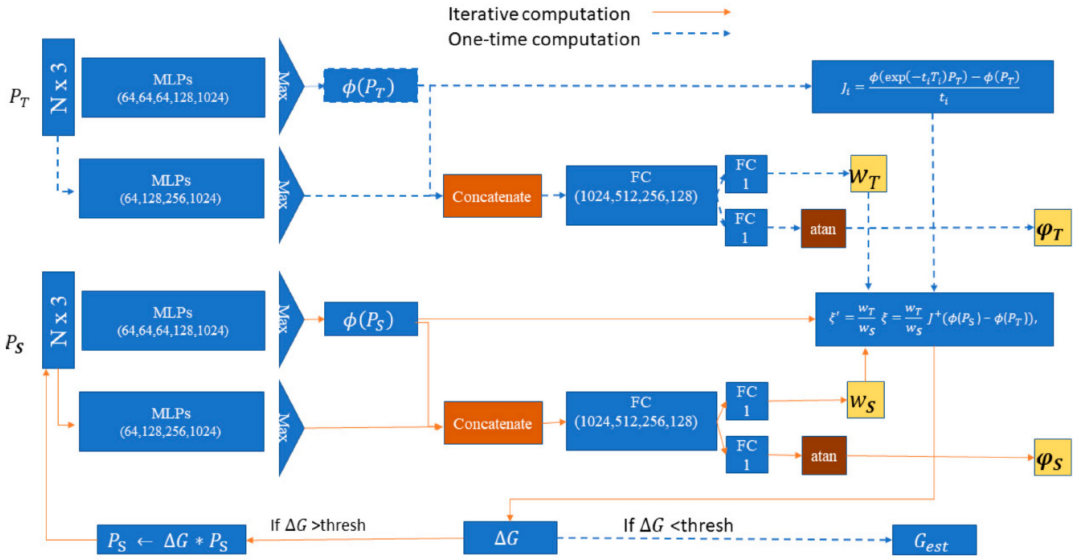


Figure 3.10: **"PointNetLK +Awe-Net Architecture"** (taken from [51]). The figure depicts the workflow of the proposed approach. The source P_S and the target P_T point sets are firstly given as input to a shared MLP. A max pooling layer is then used to compute the feature vectors $\phi(P_S)$ and $\phi(P_T)$, respectively. The *Awe-Net* block is fed with the point clouds to output weight scores $W_S \cdot W_T$ and point cloud orientations, φ_S and φ_T respectively. The best transformation parameters ξ' are finally found and used to incrementally update P_S's pose, and the global feature vector $\phi(P_S)$ is lastly recomputed.

The pioneering Alignment Weight Estimation Network (*Awe-Net*) presented in this paper is a component that points out the role played by every point cloud in estimating the final alignment. The basic idea behind its introduction is that trying to forecast the point clouds' contribution to the final output, through the computation of weight scores, makes the network itself converging in a smaller number of iterations.

The approach proposed by El Khazari et al. is depicted in Fig. 3.10. It basically depends on two modules, the global features extractor and the newly introduced *Awe-Net* module. For the first part they take inspiration from [5] (already described in Subsection 3.3.1), while the architecture of the second original module can be seen in Fig. 3.11.
The architecture of the *PointNetLK+Awe-Net* is based on five Multi-Layer Perceptrons (MLPs) of 64, 64, 64, 128 and 1024 units respectively. The MLPs are fed with the input source P_S and target P_T point clouds in a Siamese architecture. A max pooling layer, coming right after, outputs the global feature vectors $\phi(P_S)$ and $\phi(P_T)$ properly extracted from the source and the target point clouds.
By looking at Fig. 3.10 it is also possible to notice the important role played by the *Awe-Net*, which is aimed at gathering the weight scores w and the point clouds' orientation φ estimates.

The *Awe-Net* block, as shown in Fig. 3.11, is based on a stack of four MLPs with 64, 128, 256 and 1024 neurons respectively, followed by a max pooling operation which collects supplementary features, that pass then through a sequence of four fully-connected (FC) layers. The final FC layer outputs the point cloud orientation φ and the weight score w

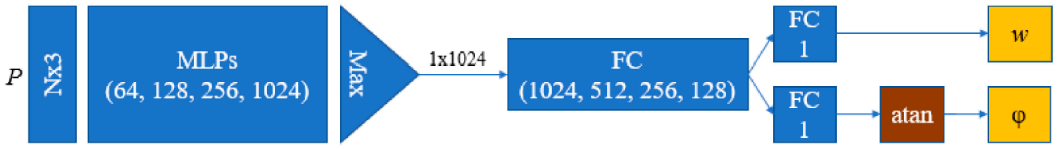


Figure 3.11: **"Awe-Net Architecture"** (taken from [51]). The Awe-Net module is composed by four MLPs of 64, 128, 256 and 1024 neurons respectively, and uses a max pooling layer to collect additional features. These features, passing through four FC layers with (almost) reciprocal sizes of 1024, 512, 256 and 128, permit the orientation φ and weight score w computations.

estimations. The former is a $1D$ rotation angle aiming at keeping a reasonable discerning power, while the latter is a positive number representing the most notable features, and it is learnt in different point cloud's locations due to the iterative learning.

The features extracted through the usage of the *PointNet*-like and the *Awe-Net* modules are concatenated (Fig. 3.10) before serving as input to the FC layers in order to provide the *Awe-Net* with all important features so that the network could converge. The weight scores of the source P_S and the template P_T point clouds must comparably contribute to the transform estimation. Thus, with the presence of *Awe-Net* the Objective. 3.32 introduced in the previous Subsection for the *PointNetLK* approach becomes as follows:

$$\xi' = \frac{w_T}{w_S}\xi = \frac{w_T}{w_S}J^{+}[\phi(P_S) - \phi(P_T)] \tag{3.35}$$

where w_T and w_S are exactly the weight scores of the target and the source point clouds, respectively. On the other hand, the orientations of the source ϕ_S and the target ϕ_T point clouds must be identical: this leads to the fact that at each iteration ϕ_S is inclined to incrementally converge to ϕ_T. Based again on [5], the learning algorithm is based on an iterative computation of the optimized transformation parameters following 3.33, with the newly computed ξ' instead of ξ.

The proposed network, implemented in an iterative fashion, proves to be very robust to noise and initial misalignment due to the presence of the novel *Awe-Net* module. It is capable of achieving precise results, comparable with other state-of-the-art registration methods, and providing a good generalizability to data unseen during the training procedure. Besides these significant advantages, the are still some drawbacks, such as the iterative way in which the network is implemented and above all its high complexity, which makes it not suitable for most of the applications operating in the embedding world.

3.3.3 PointVoteNet

The authors of [42] introduce an end-to-end learning-based approach that takes point clouds as input and estimates their 6 *DoF* pose. The newly proposed network, named *PointVoteNet*, is fed with raw point cloud data in an end-to-end fashion, i.e. it takes point clouds as input not only in the starting detection of candidate sections of the observed scene but also throughout the whole transformation estimation procedure. As for the works described in the previous Subsections, also in this case the inspiration comes

from PointNet [76], that serves as backbone for the network, but the introduced novelties remarkably improve the overall precision achieved by the 6 *DoF* pose estimation scheme.

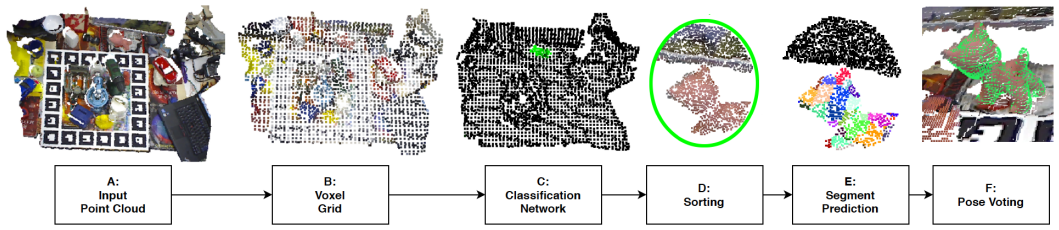


Figure 3.12: **"PointVoteNet Pose Estimation Process"** (taken from [42]). (A) depicts the starting point cloud which is sub-sampled into anchor points, shown in (B), through the application of a voxel grid. A *PointNet* returns a score (C) which represents how likely the object (marked in green in the figure) is present. (D) shows the sorting of the scores, out of which only the highest 16 are selected. A *PointNet* is then employed to generate votes (E) for each point, and these votes are lastly applied to retrieve a 6 *DoF* object's pose estimation, as illustrated in (F).

Fig. 3.12 visualizes the pose estimation process. As already anticipated, the aim of the proposed framework is that of learning the detected object's 6 *DoF* pose. To support the training procedure, a small amount of keypoints, uniformly sampled through the usage of a voxel grid from the object, is chosen to typify the whole model. The following paragraphs briefly describes more in detail each of the steps of the pipeline.

Input pre-processing. During the inference stage the input is a never-seen-before scene, supplied in the form of a raw 3D point cloud comprising *XYZ* and, whenever possible, *RGB* coordinates. The first step of the proposed pipeline is composed by a binary classifier, whose goal is to find candidate points in the observed scene. To greatly reduce the number of data points, the scene is uniformly down-sampled using a voxel grid, as shown in the approach reported in [80].

Classification stage. Each of the uniform anchor points obtained after the down-sampling could in principle be a suitable candidate for being considered as the center of the hunted object. A binary *PointNet* classifier is employed with the aim of retrieving favourable locations. More in detail, the authors decide to sample 2048 (as a nice trade-off between fastness and precision) scene points in the sphere around a previously detected anchor point. Each of these sampled points is then fed to a *PointNet* with a single logistic output unit.

Segmentation stage. After having trimmed the searching space by applying the classifier block, that outputs only the top guesses, the goal of this stage is to link each of the detected 2048 points either to the background or to the corresponding point of the searched object. Considering the object brought down to K essential points, where $K = 16$, it is necessary to use a $(K+1)$-mode segmentation approach, based one more time on a *PointNet*, to tag each point by considering it as background or as one of object's K points.

6 DoF pose estimation. The result of the previous stage is a set of correspondences. Thus, the pose estimation task has been brought down to a traditional correspondence

optimization problem, in which the input is the set of 2048 correspondences for each of the 16 top scoring points detected in previous steps. Hence, the points classified as belonging to the object are passed to the "rotational subgroup voting algorithm" [17], which is an effective method for estimating the relative pose between the two noisy input point clouds. The full 6 *DoF* object's pose estimation can therefore be calculated, and the 16 poses (one per each anchor point) are lastly refined using ICP.

Multi-modal localization loss. A multi-modal loss is finally applied to the 16 estimated poses to establish whether each of these poses is accurately learnt or not. This pose verification process is performed by firstly mapping the object model into the scene through the usage of pose estimations: the authors decide to withdraw all the obstructed points in order to associate the remaining ones with the nearest points in the scene.
A geometric and, whenever feasible, a color losses are thus computed as Root Mean Square (RMS) errors in the Euclidean and the RGB space according to:

$$\begin{aligned} \mathscr{L}_{geometric} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - p_{i,NN})^2} \\ \mathscr{L}_{color} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(c_i - c_{i,NN})^2} \end{aligned} \tag{3.36}$$

where n indicates the number of remaining points after the elimination of obstructing points, NN subscript denotes the closest point in the scene for each of the n object points in p, while c in the second equation indicates the RGB tuple associated to each point in colored point clouds.
The geometric and perceptual losses are lastly integrated with the Kernel Density Estimation (KDE) score of the voting algorithm [17] to obtain the ultimate localization loss, described as follows:

$$\mathscr{L}_{localization} = \frac{\mathscr{L}_{geometric} \cdot \mathscr{L}_{color}}{s_{KDE}} \tag{3.37}$$

This loss is capable of accurately discerning correct from imprecise pose estimates. The final output of the proposed framework is the one that minimizes the localization loss.

Tests performed by the authors reveal that *PointVoteNet* is an end-to-end framework for 6 *DoF* object pose estimation which achieves good results in point cloud registration. Moreover they claim that, by replacing *PointNet* with other more sophisticated approaches, such as *PointNetLK* [5] and *PointNetLK-AweNet* [51] described in the previous Subsections, the introduced method could potentially improve, at the expense of complexity and real-time performance. Indeed, as all the networks inspired by *PointNet*, also this approach is unfeasible to use for embedding near-real-time applications due to the difficulty of the framework's architecture and its high computational time.

3.3.4 DGR

In [23] Choy et al. propose a novel end-to-end learning-based approach for point clouds registration, named Deep Global Registration (*DGR*). They aim at tackling some of the major disadvantages of end-to-end registration approaches, such as the decreasing of spatial sharpness and precision encountered by *PointNetLK* [5] that uses global features to

encode the whole point cloud geometry, or the speculations on points and correspondences distribution made by *DCP* [91], that may be invalid for overlapping point clouds.
Thus, the proposed architecture is composed by three blocks, namely (1) a convolutional neural network that learns the point correspondences geometry in order to estimate their precision, (2) a Weighted Procrustes technique which outputs a valid outcome for the $SE(3)$ registration in a finite number of iterations, and lastly (3) a $SE(3)$ optimizer for refining the estimated alignment through the usage of gradient descent.
The following paragraphs describe in detail the architecture of the proposed *DGR*.

6-Dimensional CNN. Being $x_i, y_j \in \mathbb{R}^3$, let $X = [x_1, \dots, x_{N_x}] \in \mathbb{R}^{3\times N_x}$ and $Y = [y_1, \dots, y_{N_y}] \in \mathbb{R}^{3\times N_y}$ be two point clouds with N_x and N_y points respectively. The registration pipeline begins by extracting point-wise features, which are vectors capable of encapsulating geometric settings. While being adaptable to several features extractors, the authors decide to integrate the Fully Convolutional Geometric Features (*FCGF*) [24], that present the advantages of being rapid and discriminative. Thus, given the learned features $F_x = \{f_{x_1}, \dots, f_{x_{N_x}}\}$ and $F_y = \{f_{y_1}, \dots, f_{y_{N_y}}\}$ of the two input point clouds, the authors propose to span a set of correspondences (or matches) M using the closest neighbors in this just obtained feature space. The set M is defined as:

$$M = \{i, \underset{j}{\operatorname{argmin}} \|f_{x_i} - f_{y_j}\|) \mid i \in [1, \dots, N_x]\} \tag{3.38}$$

This filtering procedure can be learnt through a convolutional neural network (CNN) with skip connections, whose architecture is shown in Fig. 3.13. The proposed CNN lies in a 6-dimensional space and it is aimed at estimating a likelihood for each correspondence point $[x_i^T, y_j^T]^T$ in 6D space. In practice, the goal of the network is to predict the presence of an inlier, i.e., the likelihood of a certain correspondence must be true for that to be considered as an inlier. Remarkable is the fact that the 6D CNN is always capable of achieving the same outcome, no matter the initial position of the input point clouds.
During the training phase the authors decide to employ a binary cross-entropy loss to optimize the network parameters, expressed as :

$$\mathscr{L}_{bce}(M, T^*) = \frac{1}{|M|}\left(\sum_{(i,j)\in P} log(p_{(i,j)}) + \sum_{(i,j)\in N} log(p^C_{(i,j)})\right) \tag{3.39}$$

where T^* is the ground truth transformation, $|M|$ is the cardinality of the set of supposed correspondences, $N = P^C \cap M$ the set of outliers and $p^C = 1 - p$. As shown by the formula above, the loss is defined taking into account the likelihood prediction that a correspondence (i, j) is an inlier ($p_{(i,j)} \in [0,1]$) and the ground truth correspondences P.

Weighted Procrustes for SE(3). As shown in the previous paragraph, the 6D CNN outputs a weight for each input, that is, each correspondence. The authors propose to optimize, unlike what has been done in the original Procrustes algorithm [37], an objective function based on a weighted MSE, defined as:

$$e^2 = \sum_{(i,j)\in M} w_{(i,j)} \|x_i - y_j\|^2 = \sum_{(i,j)\in M} \tilde{w}_{(i,j)} (y_j - (Rx_i + t))^2 \tag{3.40}$$

where $w = [w_1, \dots, w_{|M|}]$ is the weight vector, while $\tilde{w} = [\tilde{w}_1, \dots, \tilde{w}_{|M|}] := \frac{\varphi(w)}{\|\varphi(w)\|_1}$ indicates the normalized weight vector after the application of the non-linear pre-filtering operation φ.

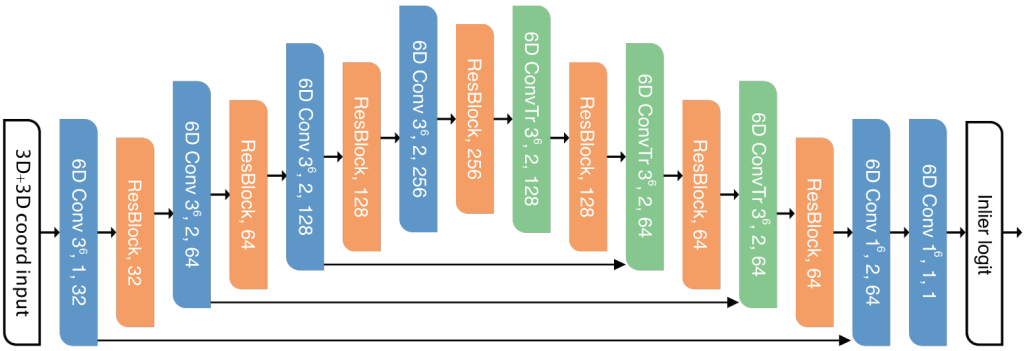


Figure 3.13: **"DGR 6-Dimensional CNN"** (taken from [23]). The convolutional neural network, used for inlier likelihood prediction, relies on a U-net backbone presenting residual blocks in the middle of strided convolutions.

The modifications introduced by the authors of [23] let the gradients move through the weights instead of positions. Thus, the Weighted Procrustes module brings into being the rotation $\hat{R}$ and translation $\hat{t}$ parameters, which are influenced by the weight vector w. These are both straight fed to the robust registration block (described in the following paragraph) as the starting pose.

Robust Registration. The last introduced module aims at minimizing a robust loss function in order to refine the transformation estimate.

Algorithm 6 Deep Global Registration Algorithm [23]

Input: $X \in \mathbb{R}^{n\times 3}, Y \in \mathbb{R}^{m\times 3}$.
Output: $R \in SO(3), t \in \mathbb{R}^{3\times 1}$.
$F_x \leftarrow$ Feature (X).
$F_y \leftarrow$ Feature (Y).
$J_{x\mapsto y} \leftarrow$ Nearest Neighbor (F_x, F_y).
$M \leftarrow \{(i, J_{x\mapsto y,i}) \mid i \in [1,\dots,n]\}$.
$w \leftarrow$ Inlier Probability (M).
if $\mathbb{E}_i \varphi(w_i) < \tau_s$ **then**
    return Safe Guard Registration (X, Y).
else
    $\hat{R}, \hat{t} \leftarrow \underset{R,t}{\operatorname{argmin}}\, e^2(R, t; w, X, Y)$.
    $a \leftarrow f^{-1}(\hat{R}), t \leftarrow \hat{t}$.
    while not converging **do**
        $\ell \leftarrow \sum_{(i,j)\in M} \varphi(w_{(i,j)}) \mathscr{L}(Y_j, f(a)X_i + t)$.
        $a \leftarrow$ Update $(a, \frac{\partial}{\partial a}\ell(a,t))$.
        $t \leftarrow$ Update $(t, \frac{\partial}{\partial t}\ell(a,t))$.
    return $f(a), t$.

The authors choose a gradient-based technique to refine poses. During the iterative procedure, the correspondence likelihoods obtained from the 6D CNN, estimated one single time per initialization, are taken into account.
Moreover, their novel approach is built in such a way that it provides a malfunction detection mechanism. Practically speaking, the framework is able to detect when the system is about to be defective before it actually provides an output. In such situations it could decide to switch to a more precise yet more time-consuming method, such as RANSAC [31] or branch-and-bound [99] techniques.

The authors implement a powerful loss function to refine the final registration, which can be described according to:

$$E(R,t) = \sum_{i=1}^{n} \varphi(w_{(i,J_i)}) \mathscr{L}(y_{J_i}, Rx_i + t) \tag{3.41}$$

where:

$$\begin{cases} \varphi(\cdot) & \text{is a non-linear pre-filtering function.} \\ \tilde{w}_i & \text{denotes the normalized weight after the non-linear transformation } \varphi. \\ J_i & \text{is the correspondence } x_i \Longleftrightarrow y_{J_i} \\ \mathscr{L}(x,y) & \text{is the Huber point-wise loss function between } x \text{ and } y. \end{cases}$$

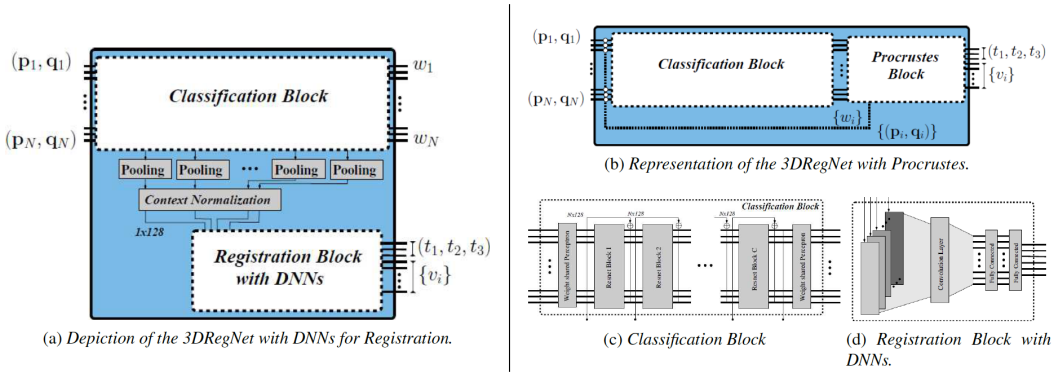
The energy function is parameterized by R and t, which are represented by means of $a_1, a_2 \in \mathbb{R}^3$ and t parameters. To have a glance of the complete *DGR* algorithm, one could look at Alg. 6 which thoroughly describes the presented framework, as reported in [23].

The authors reveal through a large amount of experiments that *DGR* is a generalizable approach, more precise and most importantly faster with respect to both traditional registration algorithms and end-to-end approaches, including among the others *Go-ICP* [99] (described in Subsection 3.1.3), *DCP* [91] (explained in Subsection 3.2.3) and *PointNetLK* [5] (reported in Subsection 3.3.1).
Besides presenting all these salient and high-ranking advantages, the large complexity of its architecture may still remain a serious issue, especially for embedding applications in robotics.

3.3.5 3DRegNet

The authors of [70] introduce an original end-to-end deep learning model for point clouds registration tasks, denoted *3DRegNet*. They aim at tackling two major challenges, namely (1) trying to recognize and label each correspondence as belonging either to inliers or outliers, and (2) finding the transformation parameters that best align the two input point clouds. To solve the second issue the authors propose either to use a registration approach based on a Deep Neural Network (DNN) or to employ the Procrustes algorithm presented in [37] to estimate the rigid motion parameters through the usage of Singular Value Decomposition (SVD).
Lastly, as it will be described later on in this Subsection, they introduce the employment of a tiny *3DRegNet* to refine the obtained outcomes' precision.

Fig. 3.14 shows the proposed architecture, with the two blocks for classification and registration depicted in detail. As it is possible to see by looking at the figure, there are two



(a) *Depiction of the 3DRegNet with DNNs for Registration.*

(b) *Representation of the 3DRegNet with Procrustes.*

(c) *Classification Block*

(d) *Registration Block with DNNs.*

Figure 3.14: **"3DRegNet: the two proposed architectures"** (taken from [70]). (a) depicts the first of the presented approaches, using both the classification and the registration blocks based on DNNs, while (b) illustrates the second pipeline, constituted by the same classification module of the previous one but with another registration block, this time inspired by the differential Procrustes method. (c) and (d) finally display more in detail the classification and registration (by means of a DNN) blocks, respectively.

alternatives for the registration module, one using DNNs and the other employing differentiable Procrustes, but the choice of one over the other does not affect in any way the loss function, that will be presented later on.
The next paragraphs explain more in detail the innovation brought by Dias Pais et al.

Classification. Fig. 3.14(c) clearly shows that the classification module is fed with a set of point correspondences computed allying the two point clouds and represented by $\{(p_i, q_i)\}_{i=1}^{N}$. A fully-connected (FC) layer takes each of the N calculated correspondences in order to provide an output composed by 128 dimensional features per each correspondence. The final output has thus a dimensionality of $N \times 128$, and it is fed to C ResNets [45] architectures (with C controlled by the complexity of the esteemed transformation), implemented with the usage of weight-shared FC layers. A last FC layer (with ReLU and subsequently tanh as activation functions) outputs the final weights $w_i \in [0,1)$.

Registration using DNNs. The features learnt from the correspondences computed according to the classification block cited in the previous paragraph directly serve as input to the registration block leveraging on a deep neural network (DNN), as shown in Fig. 3.14(a) and more in detail in Fig. 3.14(d). First of all, a max pooling layer is used to extract significant 128×1 features from each classification block's layer, and it is immediately followed by a context normalization, implemented to concatenate the $C+1$ feature maps and normalize them. In this way the authors claim that it is possible to extract, independently from the original N correspondences, the suitable number of features to finally derive the rigid body transformation, which is the goal of the registration task. After the context normalization, the features are fed to a last convolutional layer, whose outcome is finally served as input to two FC layers with 256 filters. Being M the rotation parameters and t_i the translation ones, with $i \in [1,3]$, the registration block allows to obtain at the end an output of $M+3$ variables ($v = (v_1, \ldots, v_M)$ and $t = (t_1, t_2, t_3)$), which is exactly the original goal.

Registration using the differentiable Procrustes technique. Differently from the version making use of a DNN, this approach aims at finding the craved transformation without employing any neural network but only relying directly on the point correspondences, as it is possible to see in Fig. 3.14(b). First of all, the points classified as outliers are removed, while the centroid of the ones labelled as inliers is calculated and saved as the origin: these operations can be interpreted as intermediate layers, thus empowering an end-to-end approach leveraging on both classification and pose estimation.
Due to the fact that at this point the calculated centroids are at the origin, the unique operation to perform is the computation of the rotation existing between them. This rotation is determined from the SVD of the matrix $M = U\Sigma V^T$ [7], in which $M \in \mathbb{R}^{3\times3}$ is defined according to:

$$M = \sum_{i\in I} w_i p_i q_i^T \tag{3.42}$$

where I represents the set of points labelled as inliers through the classification module. The rotation is thus derived by:

$$R = U\,\mathrm{diag}(1,1,\det(UV^T))\,V^T \tag{3.43}$$

while the translation parameters are specified by:

$$t = \frac{1}{N_I}\left(\sum_{i\in I} p_i - R\sum_{i\in I} q_i\right) \tag{3.44}$$

where N_I denotes the number of inliers and I the set of inliers themselves.

Loss Function. The overall loss function introduced by the authors of [70] is given by the weighted sum of two separate loss contributions, denoted as classification and registration losses, coming from each of the two modules of the proposed architecture.
The *classification loss* is a cross-entropy loss that penalizes wrong correspondences and it is defined as follows:

$$\mathscr{L}_c^k = \frac{1}{N}\sum_{i=1}^{N} \gamma_i^k H(y_i^k, \sigma(o_i^k)) \tag{3.45}$$

where:

- $\mathscr{L}_c^k$ is the classification loss of a specific pair indexed with k correspondences.
- o_i^k are the outputs of the network before applying ReLU and tanh activation functions in order to compute the weights.
- σ indicates the sigmoid activation function.
- $H(\dots)$ denotes the cross-entropy function.
- y_i^k is the ground truth, telling if the i^{th} point correspondence is an inlier (in this case it is equal to 1) or not (equal to 0).
- γ_i^k stabilizes the loss by the number of samples for each class in the associated point pair k.

The *registration loss* inhibits skewed points in the point cloud employing the distance between the q_i and the transformed (by means of rotation and/or translation) p_i points of

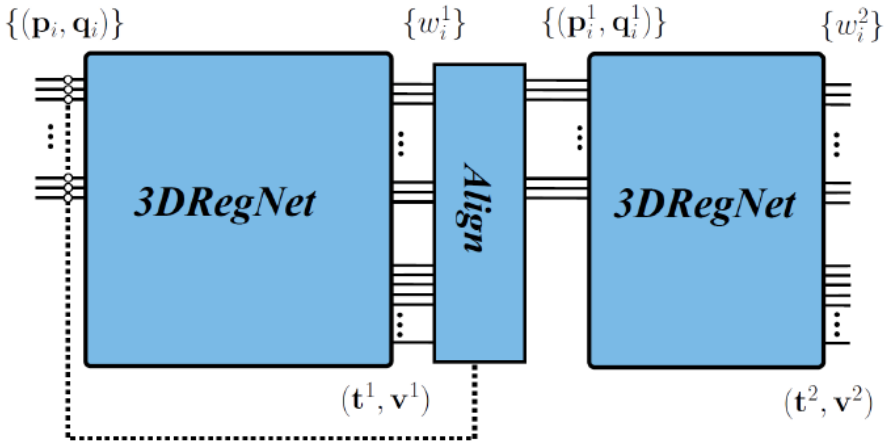


Figure 3.15: **"3DRegNet Refinement Scheme"** (taken from [70]). Two sequential *3DRegNets* (a bigger and a smaller one) can be used together in order to find a rigid body transformation that best aligns two input point clouds (with the first) and to refine the overall alignment (with the second) so to improve the final precision.

the two point clouds, with $i = 1, \ldots, N$. It is thus described as:

$$\mathscr{L}_r^k = \frac{1}{N} \sum_{i=1}^{N} \rho(q_i^k, R^k p_i^k + t^k) \tag{3.46}$$

where:

$$\begin{cases} R^k \text{ and } t^k & \text{are the transformation parameters given by the registration block for a given scan pair with index } k. \\ \rho(\ldots) & \text{denotes the distance metric function.} \end{cases}$$

Given the totality of the scan pairs in the training set K, it is possible to identify the following individual loss functions:

$$\mathscr{L}_c = \frac{1}{K} \sum_{k=1}^{K} \mathscr{L}_c^k \qquad \mathscr{L}_r = \frac{1}{K} \sum_{k=1}^{K} \mathscr{L}_r^k \tag{3.47}$$

The final loss employed for the training procedure is therefore defined as the weighted sum of the classification and the registration loss contributions, and it can be expressed according to:

$$\mathscr{L} = \alpha \mathscr{L}_c + \beta \mathscr{L}_r \tag{3.48}$$

where α and β are hyper-parameters manually set for the classification and registration loss terms.

Refinement 3DRegNet. A technique frequently used for point cloud registration is that of firstly retrieving a rough transformation measurement, and then applying in a second moment a fine-tuning scheme. Thus, the authors argue that it is feasible to evaluate the presence of a supplementary *3DRegNet* in this sense: the first tries to obtain a coarse estimate, while the second tiny network is used for refining the esteemed transformation.

Fig. 3.15 shows the refinement framework, in which the first network is employed to estimate a rough transformation while the second tries to perform the fine-tuning.
Both the networks are parameterized by the regression values $\{(R^r, t^r)\}$ and the classification weights $\{w_i^r\}_{i=1}^N$, with $r = \{1,2\}$. Due to the new scheme, the second network's loss must take into account the increasing regression of the two architectures.

The authors show that their approach is able to tackle the registration problem by jointly estimating the alignment pose of the input point clouds and discarding detected outliers in point correspondences. A substantial amount of experiments proves that the network proposed in [70] is extremely efficient, performing as well as state-of-the-art models while still being significantly faster. For what concerns the usage of a second *3DRegNet* for the fine-tuning procedure, despite a confident gain in the performance, this could reduce too much the real-time performance and could excessively increase the complexity for an embedding (robotic) application.

3.3.6 FMR

The authors of [47] propose an end-to-end Feature-Metric Registration (*FMR*) approach for 3D registration applications. The main proposal behind the novel framework is the optimization of the objective function through the minimization of a feature-metric projection error, which means that no point correspondences are needed. Besides its fastness, the newly introduced method presents several advantages, such as the robustness to noisy input point clouds. The idea at the very basis of this model is that the more the point clouds are aligned one to the other the lower is the feature difference; thus, the model is trained by the authors both in a semi-supervised and in an unsupervised way, leading to results that outperform other state-of-the-art approaches.

The main innovation introduced by Huang et al. in this paper is that of combining the traditional mathematical optimization approaches to the more recent methods influenced by the usage of deep neural networks in order to straight obtain the motion parameters without the need of employing correspondences.
Doing so, they are able to generate an approach that tailors the feature learning for the specific registration task at hand. Thus, it ends up being remarkably different from the traditional registration pipeline that instead tries to find a solution to the correspondences and the motion parameters estimation, and goes on by optimizing the objective function based on the geometric projection error.

The problem formulation and the overview of the proposed framework are described as follows. Given two point clouds $P \in \mathbb{R}^{M\times 3}$ and $Q \in \mathbb{R}^{N\times 3}$, the final objective of the 3D registration is that of retrieving the rigid motion parameters g (composed by the rotation matrix $R \in SO(3)$ and the translation vector $t \in \mathbb{R}^3$) that best aligns the source point cloud Q to the target one P according to:

$$\underset{R\in SO(3), t\in\mathbb{R}^3}{\operatorname{argmin}} \; \|r(F(P), F(RQ+t))\|_2^2 \tag{3.49}$$

where $F(P) \in \mathbb{R}^K$ is the feature extracted from the point cloud P, K is the feature dimension, F is a feature extraction function learnt by the encoder block (that will be briefly explained in a moment) and $r(F(P), F(RQ+t)) = \|(F(P) - F(RQ+t)\|_2$ finally stands for the feature-metric projection error between the target P and the transformed source Q.

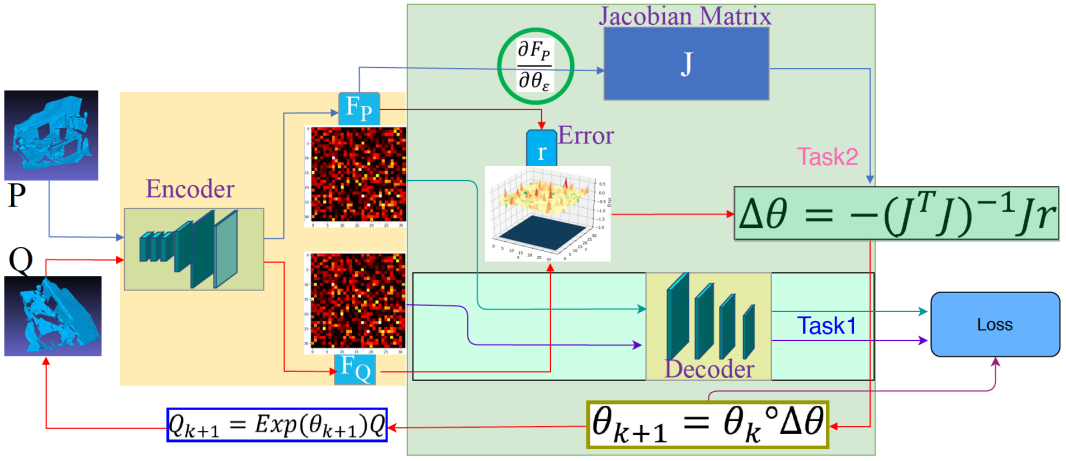


Figure 3.16: **"Feature-Metric Registration Architecture"** (taken from [47]). Starting from the source 3D point cloud Q and the template one P, an encoder unsheathes the corresponding features. A neural network is capable of solving the registration problem at hand without correspondences in a semi-supervised manner. In the first task a decoder apprehends the features and the whole encoder-decoder framework trains the encoder network in a unsupervised fashion. In the second task instead the error is computed directly taking into account the input features, F_Q for the source and F_P for the target. This just obtained error is then fed to a highly non-linear optimization algorithm which calculates the transformation increment ($\Delta\theta$), starting from which the motion parameters are updated in an iterative procedure.

To find a solution the Objective function described above, the authors of [47] propose the *FMR* approach, whose architecture can be seen by looking at Fig. 3.16. As it is shown in this figure, two rotation invariant features are extracted from both the input point clouds and given as input to a module that performs several tasks.
Specifically, in the first of these assignments (namely *Task 1*), a decoder is blocked out in order to train the encoder module in an unsupervised fashion, while the second task (*Task 2*) consists on the computation of the projection error r as the feature difference between the two point clouds. The minimization of the latter lastly leads to the transformation that registers in an optimal way the source to the target point cloud. This motion parameters estimation is an iterative process in which the transformation increment ($\Delta\theta$) of each step is calculated by running the Inverse Compositional (IC) algorithm described in [10] and defined according to:

$$\Delta\theta = (J^TJ)^{-1}(J^Tr) \tag{3.50}$$

where r denotes the feature-metric projection error while $J = \frac{\partial r}{\partial \theta}$ represents the Jacobian matrix of r computed with respect to the motion parameters θ. In order to calculate in an effective and efficient manner the Jacobian matrix, instead of computing it using a stochastic gradient approach the authors follow the technique presented in *PointNetLK* [5]:

$$J_i = \frac{\partial F_P}{\partial \theta_\xi} = \frac{F(R_iP + t_i) - F(P)}{\xi} \tag{3.51}$$

where $\xi = (R_i, t_i)$ represents the infinitesimal perturbations of the motion parameters during the loop. In particular, six transformation parameters are used in this paper, three for the rotation $(v1, v2, v3)$ and three for the translation $(t1, t2, t3)$. After a certain number of iterations (10 is the value set for the experiments that the authors performed), the model finally produces as output a transformation matrix g_{est} (comprising both R and t) and the feature-metric projection error r_{est}.

In the following, the main components of the framework are explained in detail. For an overall visualization, one can follow Fig 3.16.

Encoder. This module, composed by two MLP layers followed by a max pooling one, gains an understanding of a feature extraction function F that aims at extracting a peculiar feature per point cloud. The main characteristic this extracted feature should have is that of being rotation-attentive so that it could mirror and replicate the rotation changes during the motion estimation process.

***Task 1*: Encoder-Decoder Branch.** A decoder block is used to decode the features (previously extracted from the encoder) and map them to the 3D point sets. This framework composed by the sequence of encoder and decoder can be trained in a completely unsupervised manner: in this way, the encoder is able to understand and recognize exclusive rotation-attentive features. The decoder module is based on a stack of four sequential fully-connected (FC) layers with Leaky ReLU as activation function, and it is designed in such a way that its outcome has the same dimension of the input point set.

***Task 2*: Feature-Metric Registration Branch.** As previously anticipated, the authors decide to find a solution to the point cloud registration issue by computing the motion parameters through the employment of the Inverse Compositional (IC) technique. This algorithm is used for the optimization of the objective function describing a feature-metric projection error, that is defined according to:

$$r = \|F(P) - F(g \cdot Q)\|_2^2 \tag{3.52}$$

where $F(\cdot) \in \mathbb{R}^K$ is the distinctive comprehensive feature extracted from either the target point set P or the transformed source one $g \cdot Q$, while g is the transformation matrix (comprising both R and t).
With the goal of better visualizing the role of the feature network learned throughout the whole optimization procedure, one could have a glance at Fig. 3.17. This figure illustrates how the features extracted from the (transformed) source and the target input point sets Q and P change during the iterative process, with the variations specifically displayed after the first, the 5^{th} and the final 10^{th} loops.

With the aim of providing a semi-supervised approach, the authors of [47] introduce two loss functions. To obtain an unsupervised framework, one could simply disregard the supervised geometric loss, that will be described in the following.
Shadowing the approach presented in [38], the Chamfer distance loss, defined as follows, is employed to train the encoder-decoder framework in an unsupervised fashion:

$$\mathscr{L}_{cf} = \sum_{p \in A} \sum_{i=1}^{N} \min_{q \in S^*} \|\phi_{\theta_i}(p;x) - q\|_2^2 + \sum_{q \in S^*} \min_{i \in 1,\ldots,N} \min_{p \in A} \|\phi_{\theta_i}(p;x) - q\|_2^2 \tag{3.53}$$

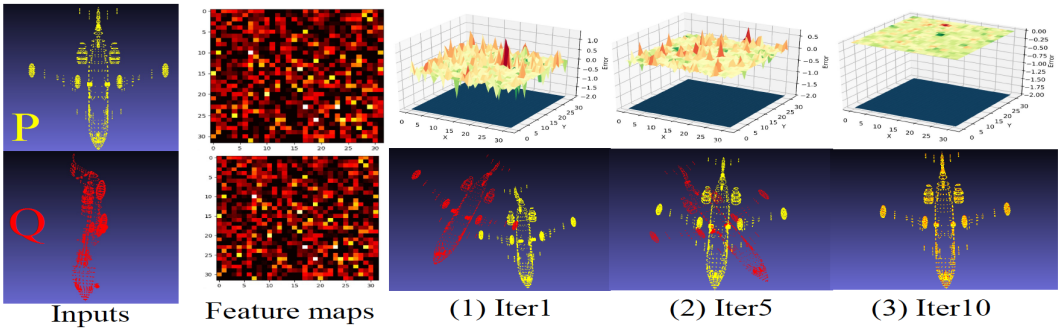


Figure 3.17: **"Feature-Metric Registration Feature Maps"** (taken from [47]). The figure displays the salient features learned by the two-MLPs-based encoder block and the feature maps variations throughout the whole iteration procedure. After the first repetition the difference between the features extracted from P and transformed Q (shown in the top of the sequence) is quite substantial, thus their alignment (depicted in the bottom) cannot be considered precise at all. During the iterative procedure the difference between the two feature maps becomes smaller and smaller, and after the final 10^{th} iteration it is almost negligible, with the alignment between the template and the transformed source point clouds being indeed nearly flawless.

where:

$$\begin{cases} p \in A & \text{denotes a collection of points sampled lies within a unit square } [0,1]^2 \\ x & \text{indicates a feature extracted from a point cloud.} \\ S^* & \text{is the original input point cloud in the 3D space.} \\ \phi_{\theta_i} & \text{represents the } i^{th} \text{ component of the MLP parameters.} \end{cases}.$$

The objective of the geometric loss is instead that of optimizing a function based on the dissimilarity existing between the ground truth (g_{gt}) and the learned transformation matrix (g_{est}), and it is described as follows:

$$\mathscr{L}_{geom} = \frac{1}{M} \sum_{i=1}^{M} \| f(g_{est} \cdot P) - f(g_{gt} \cdot P) \|_2^2 \tag{3.54}$$

where P denotes a point cloud and M indicates the totality of its points.
Thus, the ultimate loss function used for the training in a semi-supervised manner is defined according to:

$$\mathscr{L} = \mathscr{L}_{cf} + \mathscr{L}_{geom} \tag{3.55}$$

while for unsupervised training only the $\mathscr{L}_{cf}$ is employed.

After extensive experiments on traditional 3D registration datasets such as ModelNet40 [97] and 7Scene [82], the authors of [47] declare that their method clearly outperforms other state-of-the-art optimization-based, features-learning and end-to-end learning-based approaches, achieving the overall best performance regardless of the initial point clouds orientation. Moreover, the unsupervised method always achieve better results with respect to the semi-supervised one, in all the experiments they conducted. Thus, the newly introduce approach can optimally work in a broad range of applications, since the network can

be trained without the need of any label.
Lastly, the ablation studies proposed in the paper show that the *FMR* model can handle point clouds with a high degree of noise or even partially overlapped while still obtaining robust, very precise and most importantly incredibly fast outcomes.
For all these reason such approach has been chosen to be implemented and tailored to the needs of this thesis work.

Part II

Implementation of the Solution

Chapter 4

Hardware

4.1 PMD Camboard PicoFlexx

The ToF camera used to conduct all the experiments in this thesis work is the CamBoard PicoFlexx [73], a thin USB device for capturing 3D point clouds with a variety of use cases, working with a VCSEL-based IR illumination. Developed by Infineon in partnership with pmdtechnologies, this sensor is one of the first ToF cameras designed for the end users, concretely conceived for mobile devices applications, as thoroughly explained in [71]. Indeed it doesn't include any other detectors rather than the single depth one, and this leads to a very slim floaty device.
To get a visual idea of the sensor, Fig. 4.1 shows the frontal (a) and lateral (b) view of the ToF camera used in this research.

As synthesized by [62], this camera offers several advantages:

1. **ToF Precision.** The sensor is capable of capturing depth maps at 16 bit with a spatial resolution of 224×171 pixels, with a frame rate that meets (near-)real-time requirements.

2. **Size and Dimensions.** The sensor has limited size ($68\,mm \times 17\,mm \times 7.35\,mm$) and weight ($8\,g$), thus it is suitable for an application in robotic context.

(a) Frontal View (b) Lateral View

Figure 4.1: **"PMD Camboard PicoFlexx"**. Frontal (a) and Lateral (b) views of the PMD Camboard PicoFlexx sensor. The images are acquired using the *.step* file that can be downloaded from the official site [73].

PMD Camboard PicoFlexx Specifications				
Parameters			**Parameters**	
Dimensions [*mm*]	Width Height Depth	68 17 7.35	Operating System	Windows 7/8/10 Linux/ARM, Ubuntu Linux 16.04 MacOS, Android/ARM
Weight [*g*]		9	Depth Resolution [*px*]	224 × 171
Dimensions [*mm*]	H V	62.0 45.0	Acquisition Time [*ms*]	4.8 typ. @ 45 fps 30 typ. @ 5 fps
Range [*mm*]	Min Max	100 4000	Power Consumption [*mW*]	USB2.0 compliant, 300 for IRS chip and illumination
Scan Size [*Points*]		38'304	Illumination [nm]	850, VCSEL, Laser Class 1
HW. Interface	Data Power	USB 2.0/3.0 USB 2.0/3.0	Software	Royale SDK(C/C++ based, supports Matlab, OpenCV, ROS)
Baseline [*mm*]		16	Price[€]	450

Table 4.1: **"PMD Camboard PicoFlexx Specifications".** Sensor parameters according to the vendor, taken from the data-sheet available at the official site [73] integrated with the ones available at [54].

3. **Adaptable Frame Rate.** The sensor has different acquisition modes and it could capture up to 45 images per second (45 *fps*).

4. **Acquisition Range.** The sensor provides two possible depth resolutions, the first spans a range between $0.5\,m$ and $4\,m$, while the second works in a range of $0.1\,m \div 1\,m$.

Table 4.1 summarizes the sensor specifications according to the vendor, by considering the data-sheet available at the official site [73] and the information reported in [54].
As reported by Eduard Mann, Technical Key Account Manager at PMD, after a brief dialog, the PMD PicoFlexx camera has a global calibration which "works fine".
Its parameters concerning the focal length, the image center and the distortion, both radial and tangential, can be read out in Table 4.2.

Thus, instead of proceeding with the calibration of the sensor itself following all the steps described in Section 2.2 for the *systematic errors* correction, regarding both the lateral and the depth calibration, its parameters and performance are analyzed in the next paragraphs, shadowing the approach described in the brilliant work of [71]. This is done in order to better understand the characteristics of the ToF camera at hand, thus learning its behaviour and adjustment in different situations.
Differently, for what concerns the *non-systematic errors* such as *MPI* and shot noise, several approaches among the ones explained in Section 2.3 are considered and two of them are chosen to be replicated and adapted to this thesis' needs according to some particular metrics that will be exhaustively described in the next Chapter.

Regarding this first part related to the analysis of the performance and the intrinsic and extrinsic parameters of the sensor, all tests are performed in a restricted space of $3.5\,m \times 4.5\,m \times 3\,m$. Table 4.3 outlines the final outcomes of the conducted experiments about temperature, amplitude and temporal-related errors, as well as depth distortion, and several characteristics of the PicoFlexx sensor are pointed out, remarking its peculiarities with respect to many other ToF cameras available in the market nowadays. For more details on these tests please read [71], from which they are entirely taken.

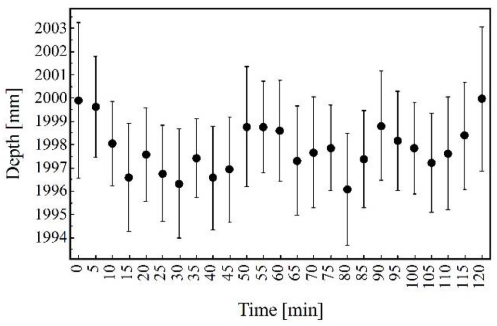
PMD Camboard PicoFlexx Calibration		
		Calibration Parameters
Focal Length	f_x	210.86590576171875
	f_y	210.86590576171875
Image Center	c_x	106.12774658203125
	c_y	87.6448059082031
Radial Distortion	r_1	0.471586138010025
	r_2	-4.898763179779053
	r_3	8.521048545837402
Tangential Distortion	t_1	2.790543903740425e-15
	t_2	9.706180188740181e-16

Table 4.2: **"PMD Camboard PicoFlexx Calibration".** Calibration parameters (provided by the sample code supplied by [73]) of the 224×171 *px* PMD Camboard PicoFlexx camera.

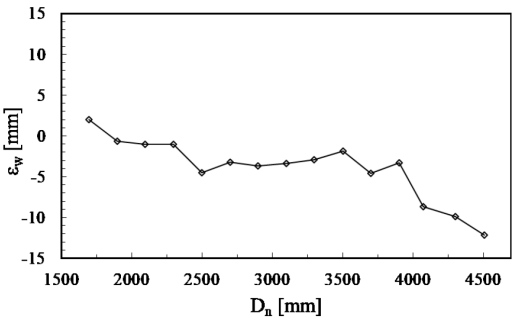
Temperature-Related Errors. To assess the error in the raw ToF measurements that gradually happens during a predefined period of time, the distance of the sensor from the target ($2\,m$) is kept constant. The camera runs at 30 *fps*: using the SDK Royale Viewer provided by [73], a single depth map is registered for a 2-hours period using intervals of 10 seconds each. Then, the frames acquired in each interval of 5 minutes are clustered together. Thus, for each group the mean and the standard deviation of the inner values are calculated.
Fig. 4.2(a) depicts the outcomes as a function of time: as it is possible to notice, the average values are distributed around the nominal value of $2\,m$ during the whole acquisition time, while the standard deviations present a maximum value of $3.37\,mm$. Remarkably, the PicoFlexx camera does not need any warm-up waiting time before being ready for acquiring new data, and this represents a sensational advantage with respect to many other ToF sensors.

Depth Measurements Errors. In this collection of tests, the camera is moved in 15 nominal positions D_n, at each of which 30 frames are acquired. As shown by the authors of [71], the depth values D_m, quantified on a 15×15 pixels square containing all the nearest neighbors around the central pixel of each frame, are normally distributed. This leads to the fundamental consequence that it is possible to evaluate depth measurement errors by only taking into account the average values and the standard deviations of the raw measurements D_m, thus substantially simplifying the computation.
Underneath this observation, in the following three errors are considered, namely the depth distortion, the amplitude-related error and the temporal error.

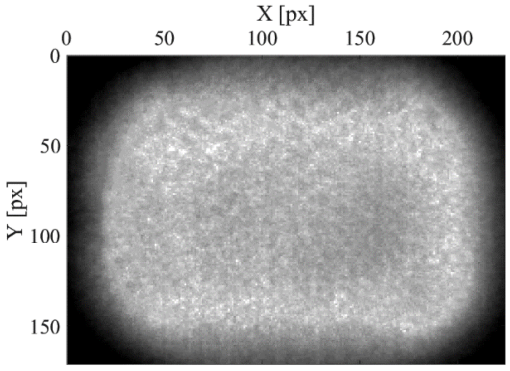
- **Depth Distortion.** Given the mean value μ_m over depths D_m, computed at every frame's pixel in the center, and the relative nominal value D_n, the depth distortion is calculated by taking into account the gap ε_w existing between the two. The outcomes of this measurement are shown in Fig. 4.2(b): the values ε_w are fully extended over a



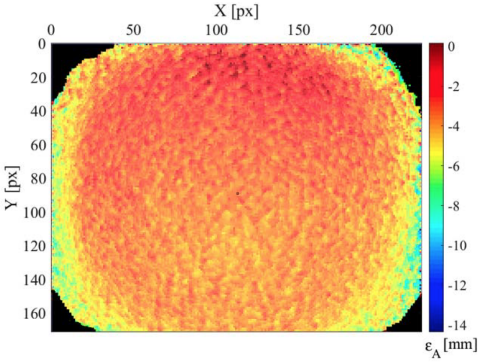
(a) Temperature-Related Error.



(b) Depth Distortion ε_w.



(c) Amplitude Image.



(d) Color Maps of errors ε_A.

Figure 4.2: **"Temperature and Depth Errors, Amplitude and Color Map"** (taken from [71]). Temperature-Related error (a), Depth Measurements error (b), Amplitude Image (c) and Color Maps of errors ε_A (d) for the PMD Camboard PicoFlexx sensor.

range $-4.54\,mm \div 1.98\,mm$ with nominal distances lower than $3.5\,m$, while the minimum value represented by $-12.15\,mm$ is reached at $D_n = 4.5\,m$. As it it possible to notice, the curve doesn't exhibits the typical wiggling behavior with a sinusoidal appearance that can be perceived in several other sensors, and this leads to an important diminution of the error terms with respect to different other cameras.

- **Amplitude-Related Error.** Fig. 4.2(c) shows the amplitude image acquired by the camera at a nominal distance of $D_n = 1.7\,m$. This image documents the intensity of the light mirrored by each pixel's surface in an 8-bit-gray levels range, which decreases as long as the distance from the image center gets larger.
  Thus, the image borders and the corners results to be highly unilluminated, indicating the fact that the signal is weak or completely missing at the corresponding pixels, leading to a depth value which could be either overestimated or completely not calculated. The error is assessed over 30 frames captured at the distance indicated above, that is $D_n = 1.7\ m$. For each pixel the error $\varepsilon_A = \mu_m - D_n$ is quantified, where μ_m represents the mean value of the raw values D_m. Fig. 4.2(d) depicts the derived ε_A values computed for each pixel, using a color map rendering. It can be seen that the amplitude-related error gets larger at the boundaries of the image. The

PMD Camboard PicoFlexx Errors	
Type of Error	**Results of the experiments**
Temperature-Related Errors	No warm-up time
Depth Distortion	No wiggling
Amplitude-Related Errors	$0 \div 5\,mm$ in the central area, up to $10\,mm$ at the borders
Temporal-Related Errors	$2.24\,mm \div 13.1\,mm$

Table 4.3: **"PMD Camboard PicoFlexx Errors"** (extracted from [71]). Details about the ToF sensor errors related to temperature, depth, amplitude and time. Worthy of note is the fact that, differently from many other ToF sensors, PicoFlexx camera does not need any warm-up time and it isn't affected by the wiggling error.

ToF sensor shows an almost uniform distribution of the computed errors over the whole frame, with a huge part of the pixels presenting errors in the range between $-5 \div 0\,mm$, while at the boundaries the fault values decrease up to $-10\,mm$. Lastly, it can be noticed that at the corners no error information is present, remarking the fact that the incident reflected light signal comes with a very shallow intensity, as mentioned while explaining and observing Fig. 4.2(c).

- **Temporal Error.** At nominal distances D_n in a range spanning between $1.7\,m \div 4.5\,m$ the raw amounts D_m are computed for the pixel in the middle of every frame, again following the authors of [71]. The temporal errors are then assessed, as a function of depth, by calculating the standard deviation σ_m over the measured values D_m. These standard deviations enlarge in the range between $2.24\,mm \div 13.1\,mm$ with a rising μ_m as long as also the distance D_n grows towards its maximum.
  Moreover, temporal errors as a function of pixel location over the frame are estimated at each D_n value. As one could expect, the σ_m values gets larger according to the increasing distance from the center of the maps.

The PicoFlexx sensor is thus developed with the aim of acquiring data within the range spanning between $0.1\,m \div 4\,m$ and it is calibrated to operate in an optimal condition in the middle of this interval, as accurately described in [4]. When working close to the minimum value of this scale in a limited area, two problems may happen: the IR emitter is excessively vivid and the effect of the difference between the IR emitter and the imager, which is insignificant at a greater distance, becomes considerable and cannot be ignored anymore. Therefore, in order to reduce and possibly remove *non-systematic errors*, i.e. *MPI* and shot noise above all, several frameworks leveraging on deep neural networks and more or less complex architectures can be used, as thoroughly explained in Section 2.3.

In the following Chapter 5 these methods are compared and a summary table reports the most significant differences on the basis of six important metrics that aim at evaluating their accuracy, their real-time behaviour, their robustness and many other aspects.
In that Chapter, the approaches chosen to be followed and tailored to the requirements of this thesis research are also investigated and illustrated more in details, together with the corresponding implementation, while the obtained results are available in Chapter 6.

4.2 Coral TPU Edge Accelerator

Another important device used for performing the experiments is the Coral USB Accelerator [35], which carries an Edge TPU coprocessor that can be easily attached through a simple USB port to the local computer providing lightning machine learning inferencing on a broad variety of devices. Being available for any local system running an operating system such as Debian Linux, MacOS or Windows 10, this slim tool has the potential to carry out 4 trillion operations (tera-operations) per second (*TOPS*), making use of 0.5 watts for every *TOPS* (2 *TOPS* per watt).
Fig. 4.3 shows the frontal and the lateral views of the device obtained from the *.step* file available at the official site [35]. In the following, more details about the device and the Edge TPUs' architecture are given, according to the specifics provided by [35] and to some reference previous works ([41, 100]).
Table 4.4 summarizes the overall technical aspects of the Coral TPU.

According to the vendor, one Coral Edge TPU accelerator is disposed to efficiently run small and memory-optimized trailblazing approaches, i.e., MobileNet v2, at just about 400 frames per second.
This on-device machine learning coprocessor is designed in such a way to enlarge the data privacy, to work even without the necessity of always being internet connected, and most importantly to greatly decrease the response delay. As stated by Gupta et Akin in their work [41], on-device machine learning hardly tries to achieve good performance in all those systems that present several constraints for what concerns computational and power resources. The recently exponential growth of on-device machine learning for the devices with limited resources has thus encouraged the development of energy-saving neural network architectures, along with tons of dedicated hardware accelerators designed to dexterously execute the most diffuse modules in neural networks.
These hardware accelerators show differences with reference to the implemented models, computational powers and capacities and memory structuring.

(a) Frontal View (b) Lateral View

Figure 4.3: **"Coral Edge TPU Accelerator"**. Frontal (a) and Lateral (b) views of the Coral Edge TPU accelerator. The images are acquired using the *.step* file that can be downloaded from the official site [35].

Coral Edge TPU Specifics	
Specifics	**Google Edge TPU coprocessor**
Dimensions [*mm*]	Width: 30 Height: 65 Depth: 8
Operating System	Linux Debian 10, or a derivative thereof MacOS 10.15 Windows 10
Operations [*s*]	4 trillion per second (*TOPS*)
Consumption [*watt*]	2 *TOPS* per watt
Connections	USB 3.0 (USB 3.1 Gen 1) port Cable (SuperSpeed, 5 *Gbps*)
Ambient Temperature [°]	Reduced Frequency at 35° Maximum Frequency at 25°

Table 4.4: **"Coral Edge TPU Specifics".** Device technical aspects according to the vendor, taken from the data-sheet available at the official site [35].

A very recent and promising research ([100]) agrees on recognizing the greatest strength of Edge TPUs in being able to bring greatly-hastened potential capabilities to devices limited by acute or critical physical and power constraints. This is the main reason behind the fact that since their first appearance Edge TPUs have been employed in several Google devices, such as Coral accelerators themselves [35] but also Pixel phones [36].

In the following, a brief description of the comprehensive Edge TPUs' architecture is provided. Fig. 4.4, taken from [100], visually depicts in details all these concepts.
These devices exploit a representation based on a template that makes fully use of micro-architectural modules which can be greatly described through parameters. This template is characterized by a sequence of processing elements, named *PEs*, that are arranged to form a 2D array. Each *PE* is designed in such a way to carry out a collection of arithmetic calculus in a single-instruction multiple-data (*SIMD*) fashion.
Furthermore, the architecture also includes an on-chip controller, whose aim is that of passing the data from off-chip memory to the *PEs* and understanding and recognizing the low-level instructions (i.e. convolution, pooling, etc.) that will be further performed on the *PEs* themselves.
The principal modules of the *PE* architecture consist of a single or multiple core(s), each of which presents several computation pathways that execute operations in a *SIMD* fashion. In a top-down procedure, each *PE* exhibits a memory block which is shared between all the computational cores, indicated as *PE* memory in Fig. 4.4, and its most important employment is that of gathering and storing model activations and outputs.
The cores enclosed by each of the *PEs* give prominence to a core memory which is mostly employed for storing the parameters of the model. Every one of these cores comprises several computational pathways, each of which presents versatile multiply-accumulate (*MAC*) units.

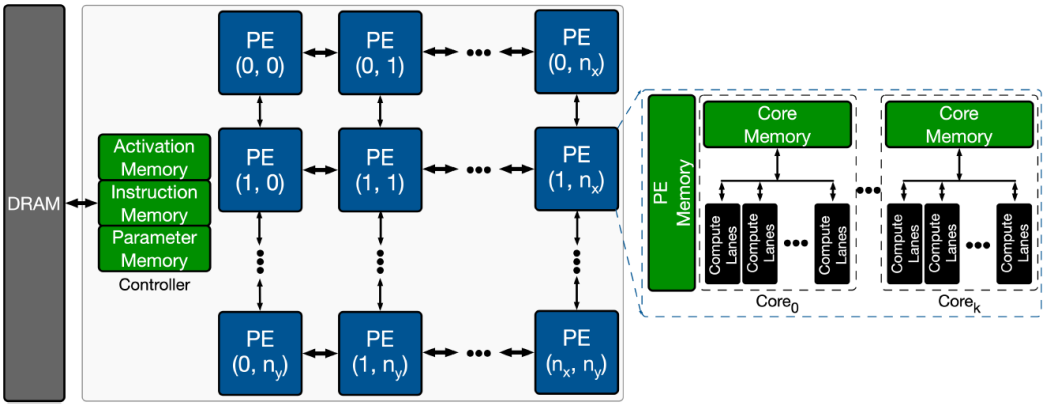


Figure 4.4: **"Edge TPU Architecture"** (taken from [100]). Overall architecture of the on-device machine learning accelerator, based on the following template. The processing elements (*PE*) are lined up to form a 2D array, and each of them embeds one or multiple cores. Every one of these cores presents several computational pathways with multiple *MAC* units working in single-instruction multiple-data (*SIMD*) manner. Each *PE* is constituted by a *PE* memory which is distributed across the several cores, each of which is also provided with an exclusive and customized memory.

The fundamental principle behind their behaviour is as follows. At every round, the computational pathways receive a collection of activations. The calculations joining the activations and the parameters of the model are carried out inside the range of every pathway using the *MAC* units. Once this calculus terminates, the outcomes could either be kept and saved in the *PE* memory for additional algorithmic computations or unloaded back to the *DRAM*.

Thus, summarizing, the Coral USB Accelerator is a tool capable of coming up with an Edge TPU as a coprocessor for the local machine, as officially stated by [35]. With the aim of allowing whistle-stop performance for neural network architectures constrained by a limited resources power, the Edge TPU is designed in such a way to handle a fixed predetermined collection of operations and network models. In particular, the Edge TPU can easily accomplish the execution of convolutional neural networks (CNN) and, generally speaking, of deep architectures.
Rather than creating an entire model and then training it starting from the very beginning, one can think about retraining an already existing model (which of course has to be well-suited with the Edge TPU) employing the so-called technique referred to as *transfer learning*. Indeed, training a deep neural network starting from zero could require a huge amount of computing time and of course of data on which perform the training itself, while by employing *transfer learning* one could simply immediately use an already trained model and conduct an additional training process, perhaps using a smaller training dataset, in order to exactly *transfer* it from the task it was originally thought for to a relatively new one.
One may be able to do this either by retraining the overall network, fine-tuning and calibrating one by one the totality of weights, or by easily freezing all the layers but the last, that is removed and substituted with a new one trained in order to tackle a specific task.

Given an adequate amount of training data and just some hyper-parameters customization and regulation, one could build a very precise model in a single pass by employing this procedure. Supposing to start with a well-suited model, due to the fact that the model architecture is not modified at all during the *transfer learning* process, it will be completely compatible with the Edge TPU.

In this thesis work, whenever possible (which means, whenever starting with a compatible model), the provided Coral Edge TPU accelerator is used to obtain faster results without compromising the power cost.

Chapter 5

Proposed Approach

5.1 Methods Comparison

In this Section, the approaches presented in Chapters 2 and 3, regarding the ToF *MPI* and shot denoising deep learning algorithms and the Point Cloud Registration methods respectively, are compared according to their performance. Specifically, inspired by [46, 86], the metrics taken into account are: (1) accuracy, (2) size of model, (3) robustness, (4) time cost, (5) latency and (6) range of application. According to the information retrieved in the related papers and briefly summarized in the previous Chapters, the mentioned indicators are compared qualitatively in Table 5.1, where *A* represents the best performance and *D* indicates the worst.

While not all being used in this thesis work due to the lack of available data for some models in the presented papers, for sake of knowledge in the following is hurriedly presented *PLASTER*, introduced in [86], a framework for deep learning performance based on the evaluation of seven metrics, namely "**P**rogrammability, **L**atency, **A**ccuracy, **S**ize of model, **T**hroughput, **E**nergy efficiency and **R**ate of learning", as shown in Fig. 5.1.
Some of them are indeed taken into account in this research, as reported in the next Subsections and as it is possible to notice in Table 5.1. In the following paragraphs, each of these metrics is briefly explained according to the official presentation in the previously cited paper.

Programmability. Programmability influences in a serious way the efficiency of the programmer and thus the period needed to release the final product to the marketplace. After a deep neural network is implemented and trained, it is enhanced to run in a particular inference domain. NVIDIA provides two main tools to tackle training and inference major issues, namely CUDA and TensorRT. CUDA is a platform that makes the calculation highly parallelizable and it is used for implementing a wide range of models on GPUs, while TensorRT is a high-level inference accelerator used for the inference stage indeed. The former comes to the aid of making more comprehensible the stages constituting an algorithm implementation, whereas the latter provides a better optimization of an already existing and trained model for the distribution at runtime, assessing several floating point and integer precision measures in order to obtain optimized solution that reaches a nice trade-off between precision and performance. The authors claim that, even if the implementation of deep learning algorithms still requires some good technical skills, these tools

Figure 5.1: **"PLASTER Framework"** (taken from [86], source: NVIDIA). The PLASTER Framework for deep learning performance measurements on the basis of seven indicators, namely "**P**rogrammability, **L**atency, **A**ccuracy, **S**ize of model, **T**hroughput, **E**nergy efficiency and **R**ate of learning".

assists in better exploiting profitable time.

Latency. Outcomes and responses are of paramount importance in order for both humans and machines to be able to remain in control of a certain situation and act in a proper way. Latency is exactly defined as the time passing between the moment in which a certain thing is asked for and the moment in which the related response is given and collected. This time interval is usually quantified in milliseconds for many of the human-facing software frameworks available nowadays. Inference latency has a direct impact on the user experience (UX) and it is instead estimated in seconds or their fractions. Even if there are no precise conventions nor rules for response times, the "$0.1/1\ /10$ second limits" proposed by Jakob Nielsen [69] represents an acceptable heuristic. If the system takes too much time to give an answer, users no longer find the stream of their activities, and this may determine a huge impact not only on their amusement, but also on performance, time and money spent. Image and video management show a clear example of how important is the necessity of having systems performing with a low latency, thus providing (near-) real-time inference-based solutions.

Accuracy. One of the main benefits of deep learning approaches is that, while being coded at low precision, they can be trained at a higher one. The training stage could indeed reach a very high level of algorithmic accuracy, typically Floating-Point (FP) 32, whereas the implementation can normally happen with a much more shallow mathematical accuracy, often FP 16. By doing so, it is feasible to acquire not only an enhanced throughput, but also a better efficiency and possibly to reduce the latency. Always keeping an elevated accuracy is of course of paramount importance for users to have the finest user experiences. Besides everything, there exist a lot of different accuracy measures. According to the *PLASTER* framework, accuracy leverages on the capability of preserving the precision achieved from the model during the training phase also during inference. The main idea behind this behaviour is that of reducing the algebraic precision to let the model be more

Comparison of the models for ToF Denoising and 3D PCD Registration							
	Method	**Accuracy**	**Size of Model**	**Robustness**	**Time Cost**	**Latency**	**Range of Application**
ToF Data Denoising	ToFNet [85]	D	C	C	D	C	C
	MOM+MRN DNN [39]	B	C	B	C	B	D
	DeepToF [63]	C	D	C	B	A	B
	2-parts CNN [40]	C	A	C	A	A	C
	Coarse-Fine CNN [3]	**B**	**A**	**B**	**A**	**A**	**B**
	Range-Recovery + Boundary-Detection NNs [84]	C	A	C	A	A	C
	Power Efficient NN ToF [21]	B	C	B	A	A	C
	Predictive + BackScattering [18]	B	A	B	C	B	B
	SHARP-Net[1] [27]	A	B	B	B	B	A
	DA-F [2]	**A**	**B**	**A**	**B**	**B**	**A**
Point-Cloud Registration	**Optimization-Based Methods**						
	ICP [12]	D	A	C	A	B	C
	Go-ICP [99]	C	B	B	B	D	B
	LM-ICP [32]	C	B	B	A	A	B
	FGR [106]	**B**	**B**	**A**	**A**	**A**	**B**
	Feature-Learning Methods						
	PPF-FoldNet [25]	B	C	A	B	B	A
	IDAM [57]	B	C	B	C	A	B
	DCP [91]	C	D	B	B	A	B
	FRR (FPFH [79] + RANSAC [31])	**B**	**A**	**B**	**B**	**B**	**A**
	End-to-End Learning-Based Methods						
	PointNeLK [5]	D	B	A	C	A	B
	PointNetLK + Awe-Net [51]	C	C	A	B	B	A
	PointVoteNet [42]	B	C	B	C	C	B
	DGR [23]	A	D	B	C	A	B
	3DRegNet [70]	**A**	**B**	**A**	**C**	**A**	**A**
	FMR [47]	**A**	**B**	**A**	**B**	**A**	**A**

Table 5.1: **"Models Comparison".** Comparison of several models both for MPI and Shot Denoising of ToF Depth Maps and for Point Cloud Registration, according to six different indicators, namely Accuracy, Size of Model, Robustness, Time Cost, Latency, Range of Application. The methods marked in **bold** are the ones that present the overall best performance according to the specified criteria, thus the ones that will be reproduced and adapted in this thesis work.

efficient in terms of energy, increasing in this way the overall throughput of the system at hand for the particular task of interest.

Size of Model. The complexity of the neural network heavily affects the performance of the model, especially for what concerns the latency and the throughput when considering the proposed framework with these seven metrics. In a deep neural network the elements responsible of the growing (or shrinking) of the size of the model itself are: (1) the number of layers, (2) the amount of units (neurons) per each layer, (3) the calculation complexity of every layer and finally (4) the quantity of connections existing between a neuron at one layer and the units of its neighbors. It follows that the size of a neural network is proportional to the number of computational (and physical) resources necessary in the inference stage. As an example, as suggested by the authors of [86], one could think about the optimization of a deep neural network with the constraint of remaining within some predefined accuracy and latency limits. In this context, it is obvious that the provided optimization will inevitably decrease the accuracy of the model by making every layer and

[1] While not being the best approach according to the specified criteria, as explained at the end of Subsection 5.1.1 this is the method that is actually used in this thesis work, due to incompatibility of [2] and [3] with the embedding environment.

the connections between one another easier. The improvement that deep learning models have experienced in the last few years in size, complexity and computational needs, together with the rising of low latency requirements for (near-)real-time tasks, underlines how important the metric related to the size of the model is. Thus, an eye must always be kept on the way in which the complexity of the network affects latency and throughput, and customization on hardware and inference precision at runtime may be taken into account in order to tackle the issues discussed so far.

Throughput. This metric outlines how many inferences can be carried out given the size and the complexity of the model at hand. Differently from the latency bound that guarantees a fine user experience, optimizing the throughput within that boundary is of paramount importance for maximizing efficiency and income. During the years there has been a propensity to employ the single throughput as the only performance metric, as generally the higher it is the better is also the performance over other aspects. Nonetheless, if the throughput is not adequately balanced with the latency, the outcome will inevitably be a substandard low-quality user service, missing without any doubt service level agreements (*SLAs*) with the conceivable risk of delivering a wrong system. A common metric for measuring the throughput for deep learning inference is the number of images produced per second for image-based networks, while for speech-based ones the amount of tokens per second is mostly taken into account. As earlier anticipated, the system must be able to obtain the desired throughput without exceeding the predefined latency bound, and this can be managed by scaling the GPUs number. Nevertheless, this scaling can be applied only if the inclusion of further GPUs doesn't make the latency of the first GPU larger.

Energy Efficiency. The performance of a deep learning accelerator is directly proportional to its power consumption: the higher is the former, the more costly is also the latter. Moreover, a growth of the power consumption could easily enlarge the costs necessary for providing a particular service. Thus, the need to focus on energy efficiency not only in systems but also in machines cannot be further ignored. One could think that a possible way to tackle this issue could be that of maximizing energy efficiency for as many inferences as possible within a fixed power threshold. Yet, the solution cannot be simply reduced at recognizing which is the isolated processor that presents the lower power consumption, since energy efficiency is based not only on the absolute power consumption throughout a certain period of time, but also on the throughput during the same interval. Thus, energy efficiency can be considered as another important sample showing how tight *PLASTER*'s metrics are interconnected one another and must be examined together to have a full idea of the inference performance.

Rate of Learning. Not long ago companies have laid the foundations of DevOps, which aims at closely binding development and operations with the support of more powerful techniques and higher-level programming tools. It goes without saying that for particularly complicated deep neural networks it is of paramount importance to keep up with the DevOps trend in order to catch on in nowadays business. As corporations carry on with deep learning frameworks implementation, they are greatly improving their technical skills about finding productive and effective ways to build and train them. One drawback of deep neural networks is that they have to be systematically retrained since new data

are always collected and services become larger and larger, possibly changing their original vision. Thus, in order to keep the pace companies and developers have to necessary increase the rate at which they can retrain models when new data are collected. In this sense, servers composed by multiple GPUs have greatly narrowed training times of neural networks from weeks and days to hours and minutes. Faster training procedures lead to the fact that the models could be more frequently retrained in order to improve precision or keep it high. Programmability also plays an important role in affecting rate of learning. To reduce programmers workflow, Google and NVIDIA have lately made public an integration between TensorRT and TensorFlow, as explained in [55]. The former could be directly called within the latter itself to optimize the architectures to efficiently run on NVIDIA GPUs. The capacity to combine training and inference effortlessly empowers DevOps solutions for deep learning. Thus, summarizing, rate of learning is quantified according to: (1) gain in throughput and model accuracy during the training, (2) gain in throughput, model accuracy and latency for production and (3) gain in programmability, size of model and energy efficiency for both training and production.

After having discussed the *PLASTER* framework and analyzed in detail each of its metrics, in the following are now thoroughly explained the criteria chosen in this thesis work, which as anticipated correspond in part with the ones just presented.
Thus, in Subsections 5.1.1 and 5.1.2 all the approaches studied in the previous Chapters, for *MPI* and shot denoising (Chapter 2) and for point cloud registration (Chapter 3) respectively, are described according to the specified indicators reported in Table 5.1.

5.1.1 ToF Raw Data Denoising

All the approaches reported in Section 2.3 regarding the denoising of ToF raw data are taken into account according to the previously specified criteria. In the following, each of the metrics is discussed more in detail for all the deep learning frameworks, namely accuracy, size of model, robustness, time cost, latency and range of application. Since the objective of this thesis work is that of building a neural network based approach that works in (near-)real-time in an embedding environment, the most interesting criteria are without any doubts the size (and thus the complexity) of the model and its latency, therefore its online performance. Of course, also the accuracy and the robustness of the architectures are looked and analyzed with special attention, along with all the other indicators.

Accuracy. Accuracy is for sure one of the most important metrics, since it regards the level of performance that the model is able to achieve. Obviously one wants the model to be as accurate as possible, meaning that, in this case, capable of removing as much noise as possible. It is feasible to see a vast heterogeneity among all the studied approaches, being some of them very accurate and other much less. What is possible to say is that, throughout the years (the models are mostly ordered in the table by the time of the publication of the relative paper), the accuracy of the networks increased. This could be explained observing that many of these models are based on previous works, thus the authors had the chance to improve what their ancestors were lacking at. Clear example of that is DeepToF [63], which is based on the same idea (about the usage of an autoencoder as deep learning neural network) of the earlier work ToFNet [85], but also DA-F GAN [2] which is an improvement of the Coarse-Fine CNN proposed in [3].
Out of all these cited methods, two are the ones with the highest accuracy (and also the

ones able to achieve the best performance when taking into account other metrics) chosen to be replicated in this thesis, namely Coarse-Fine CNN [3] and its enhanced version [2], as one could see by looking at the **bold** rows of Table 5.1.

Size of Model. Given that the whole point of the research is that of finding a NN-based approach that works fine in an embedding system, the complexity of the model itself (and the real-time performance as already anticipated) plays a crucial role in the final choice. Thus, many algorithms couldn't be selected due to the high complexity and/or the massive delay in the processing of the depth maps. As it happened for the accuracy, also regarding the size of the model it holds that the more recent is the approach the better it is, generally speaking. This is probably due to the fact that new studies were done and new approaches were suggested to exploit all the intrinsic characteristics of the depth images acquired by the ToF cameras. Moreover, worthy of notice is the fact that accuracy doesn't strictly tie with complexity of the model. Indeed, ToFNet [85] is likely the worst approach in terms of accuracy, yet its architecture is definitely not straightforward, and the same goes also for DeepToF [63], just to cite one more.
Between the totality of the proposed frameworks, the ones that present the lowest complexity (coupled with the smallest latency and, possibly, with the highest accuracy) are selected, and these turn out to be the models already cited in the previous paragraph, [3] and [2], suggested by Agresti et al.

Robustness. Being inspired by the definition given by the authors of [46], robustness largely indicates the anti-interference performance of the presented frameworks with respect to noise input. Given the fact that all these approaches are based on a deep neural network, they all take as input a considerable amount of data, thus they suitably examine the information reported in the input scene presenting a generally high level of robustness. However, the specific input data are particularly noisy point clouds and the level of noise in the collected raw data depends on a huge number of factors. It is therefore very difficult for these approaches to generalize and achieve good performance in every situation, even after a long period of training. Sometimes it could happen that they aren't capable of properly discerning relevant features from environmental noise.
According to the papers, some methods are trained in such a way to be extremely robust, and these are the ones most taken into account to perform the final choice on which method has to be selected and reproduced.

Time Cost and Latency. Since the time a method takes to provide a solution dictates its real-time performance, the two metrics are discussed together in this paragraph. All learning-based frameworks time performance can be predominantly split according to two steps: offline training and online measuring. The former is time-consuming because the model has to be trained using a huge amount of input data, and it thus needs several recurrent computations, even though the usage of GPUs could remarkably speed up the overall training procedure. In the latter instead the denoised depth map can be directly estimated by the trained models, thus the latency is theoretically very small. Nonetheless, since what truly matters in the specific situation of this thesis work is the real-time (or near-real-time) performance, not all the presented methods are good enough, since the online step could be considerably faster with respect to the offline one, but still inadequate. Thus, the approaches taken into account in this research are the ones able to provide the result (the depth map with *MPI* and shot noise greatly reduced) within few milliseconds,

and letters shown in Table 5.1 for latency metric are given accordingly to this reasoning.

Range of Application. Due to the theory on which they are based, that is the need of tons of data (and time) to train the network, learning-based approaches are able to handle several different conditions that may happen at test time. However, since there is an enormous amount of agents that can affect the quality of the input data (i.e. the reflectivity of the objects, their density and surface material, the ambient light conditions and so on so forth), not all models behave correctly with all type of inputs. Some of them may present a wider range of application, while some others are more limited in this sense, depending on how they were trained in the first place and how much capable they are of handling the limits within which any alteration takes place. Thus, for this thesis work the methods with the broadest range of application are looked with more consideration in order to use the largest possible input range and adapt the obtained frameworks also to the robotic world, perhaps in a space application with its own kind of needs and data.

Out of the several methods, as it is possible to see by looking at Table 5.1, two are selected in order to be reproduced in this thesis work (marked in **bold**), namely the *Coarse-Fine CNN* network proposed in [3] and its improvement, the *DA-F GAN* defined in [2]. Also *SHARP-Net*, as explained in [27] and in Subsection 2.3, allows to reach very accurate results, but at the beginning of this work the other two approaches were picked in order to do a comparison between the "raw" version of the first *Coarse-Fine CNN* (presented in [3]) and the "final" enhanced version of the network reported in [2], leveraging on GANs. As will be outlined later on in Section 5.2, these networks end up not being able to work in real-time, since even for inference they need the acquisition of depth maps and amplitude images taken at different modulation frequencies. Being the real-time performance a fundamental requirement for this research, another architecture has therefore been selected and implemented, which is exactly that of *SHARP-Net* [27]. More details will be found in Section 5.2, in which also the code of the models is thoroughly explained.

5.1.2 Point Cloud Registration Methods

Similarly to what is done for the ToF *MPI* and shot denoising approaches, also point cloud registration methods described in Chapter 3 are collected and compared, again accordingly to the same six metrics previously introduced. Table 5.1 shows the results of the comparison: for each type of methods, namely optimization-based, feature-learning and end-to-end learning-based, the best one is selected (and marked in **bold**). The reasoning applied before about the main criteria that have to be met for the robotic environment holds also here: size of the model and online performance lead the choice of the finest algorithms which are chosen to be implemented.
In the following, all the cited indicators are described more in detail, specifying the argumentation underneath every single choice performed.

Accuracy. Accuracy is again one of the principal indicators that one can think of when dealing with a predefined task carried out by a specific framework. This is even much more true if taking into account the point cloud registration problem, where the main idea is that of finding the rigid transformation that "accurately" aligns the two input point

clouds. Looking at table 5.1, it is immediately visible that end-to-end learning-based approaches generally allow to get much better results (in terms of accuracy) with respect to feature-learning and, mostly, optimization-based frameworks.
This is because deep learning neural networks take as input a lot of data that grant them the possibility to estimate the rigid transformation, in terms of R and t, using a more comprehensive information. Thus, the results they get are much more precise than the ones obtained by non learning-based approaches. Moreover, not all the methods belonging to this third class of point cloud registration frameworks are equally precise. Indeed, the first (temporally speaking) approaches that were proposed ([5] or even [51]) achieved results that were even less accurate that the ones reached by optimization-based methods. Then, as deep learning techniques evolve and more researches are conducted, all focused on this task, new models are gradually introduced, able to achieve nowadays state-of-the-art outcomes (as *3DRegNet* [70] or *FMR* [47]).

Size of Model. Complexity of the proposed frameworks is, as already anticipated, one of the most critical metrics that has been taken into account in this thesis work, due to the specific space environment it is focused on. Thus, it plays a fundamental role in the final choice of the approaches that will be used. It is obvious that, out of the three classes of models, the methods leveraging on deep neural networks are the ones with the highest complexity. This is due to the intrinsic nature of the models themselves, that are composed by deep architectures indeed, which aim at gathering as much information as possible, exploiting many different ways and possibilities. Since most of the latest feature-learning methods rely on deep neural networks (such as [25, 57, 91]), it follows that optimization-based frameworks are generally the ones with the simplest structure, even if also FPFH-RANSAC Global Registration (*FRR*, [79, 31]), presents a straightforward solution to tackle the registration problem. It has to be lastly said that normally the complexity of the architecture is inversely proportional to the real-time performance of the model itself, as will be discussed in the next paragraphs.

Robustness. Inspired again by the definition in [46], robustness predominantly introduces the concept of the anti-interference performance demonstrated by the considered models with respect to noise and environmental changes. Since end-to-end learning-based approaches use a massive amount of data for the training procedure, their robustness is considerably higher than the one of other methods, in particular the optimization-based ones. In fact, since the models leveraging on deep neural networks adequately view and contemplate the input information of the observed scene, their performance measured under this specific indicator is way better than the other approaches, due to the fact that they are able to correctly distinguish between object features and environmental noise.
Additionally, being dependent on the same idea of using plenty of data to acquire as much information as feasible, also feature-learning models based on deep neural networks are able to achieve an elevated robustness. Nevertheless, even if the optimization-based frameworks are also capable of efficiently discerning in most of the cases prominent features from environment, some wrong and incorrect information may be taken into consideration under some particularly noisy situations, and this may inevitably lead to final mistakes in the esteemed transformation.

Time Cost and Latency. As stated in the previous Section when dealing with *MPI* and shot denoising approaches for ToF raw data, also in this case real-time performance is

a must for models aiming at operating in embedding environments, especially for space applications. Thus, out of all the proposed frameworks, some of them are looked with special attention due to their capability of working with a low latency. In particular, as already anticipated in the paragraph concerning the accuracy metric, it generally holds that the more complex is the model the lower is its latency in real-time point cloud registration. This is true in fact if one takes into account the latency metric column in Table 5.1: one can easily note that the end-to-end learning-based and the feature-learning approaches can operate in (near-)real-time measuring the 6 *DoF* pose directly by the trained model, while optimization-based frameworks present a much higher latency due to the huge amount of computations they have to perform.
For what concerns the offline step, that is the time cost of the algorithm, it must also be said that it is time-consuming for deep neural networks since they need to train a model that is fed with a massive amount of data. This means that recurrent calculations are necessary, even if as explained for ToF models a GPU could obviously speed up the training stage. Thus, regarding the time cost, optimization-based frameworks surely behave better, but vice versa the lack of training data generally compromises their online performance.

Range of Application. Due to their working assumption, optimization-based approaches are not able to handle the problem of 6 *DoF* pose estimation of reflective objects, or many others presenting some peculiar characteristic. Nonetheless, this kind of objects is quite ordinary to be found in industry: one could easily think about, for examples, metal parts, but even walls and ceiling. Learning-based approaches leveraging on deep neural networks are instead capable of addressing this issue, but they need plenty of time to train the model, and this may result in being highly contrasting with real-time performance requirements of the system. Thus, feature-learning and end-to-end learning-based 3D point cloud registration approaches can substantially cover the widest range of application. Out of these several methods, the ones with the lowest complexity and latency (and haply with the highest accuracy and robustness) are the ones chosen to be reproduced and fitted to the purposes of this thesis work.

Specifically, as it is possible to see by looking at the **bold** rows of Table 5.1, the point cloud registration frameworks that have been selected according to the specified indicators are *FGR* [106] for optimization-based methods, *FRR* ([79] + [31]) as a representative of feature-learning methods and finally *3DRegNet* [70] and *FMR* [47] as agents of end-to-end learning-based methods. Two learning-based models have been chosen, instead of one, since they present almost the same behaviour in compliance with the selected criteria. Thus, a better and fair comparison between the two wants to be highlighted, and this is possible due to the fact that they need the same input and give the same output, which is the estimated transformation parameters. Therefore, they could be easily overlapped in the end-to-end framework built for this thesis work, which starts from the ToF data acquisition and ends with the point cloud registration of the detected point set with respect to a provided target one. However, during the implementation of *3DRegNet* [70] model, a problem has been discovered, which was that of the impossibility of the network to work with point clouds of different shapes. More details are given in the following Section, where the implementation of the proposed approaches is reported in detail. Results of the point cloud registration task are available in Chapter 6, where all the implemented models are compared.

5.2 Code Implementation

This Section contains the explanation of the code implemented for this thesis work.
The whole project is based on Python *v*3.8.12 [87], TensorFlow *v*2.6.0 [1] and PyTorch *v*1.9.1 [72], and it is built upon a machine MacOS BigSur *v*11.3.1 with a 2.7 GHz Intel Core *i*5 dual-core processor. Please refer to the provided link[2] for more details about the code itself.

The Section is organized in this way: Subsection 5.2.1 focuses on the data acquisition part using the ToF PMD Camboard PicoFlexx sensor [73], Subsection 5.2.2 strengthens the ToF *MPI* and shot denoising task, emphasizing the two implemented deep learning approaches (namely, *Coarse-Fine CNN* [3], along with its enhancement [2], and *SHARP-Net* [27]), while Subsection 5.2.3 finally defines the point cloud registration problem, carrying out four different methods, one for the optimization-based frameworks (*FGR* [106]), one for the feature-learning models (*FRR* [79, 31]) and lastly two for the end-to-end learning-based ones (*3DRegNet* [70] and *FMR* [47]).

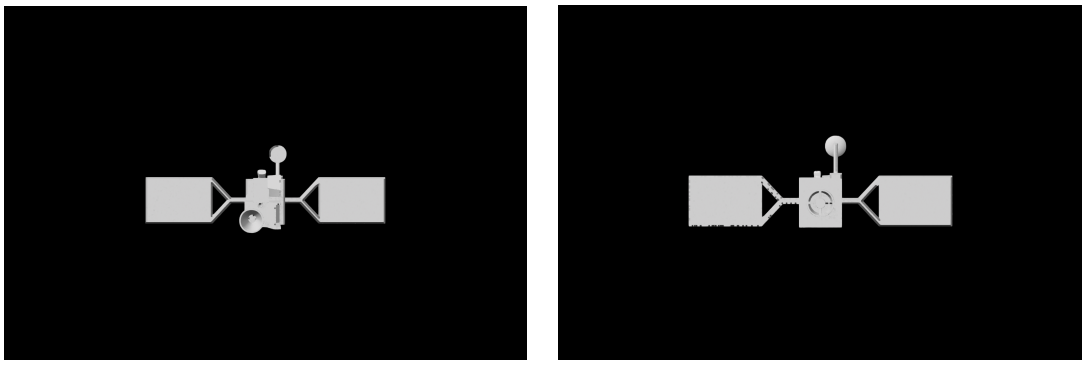
5.2.1 Data Acquisition

The first task of this thesis work is that of acquiring and extracting useful information from the ToF camera provided by the company, a PMD Camboard PicoFlexx. The starting point is represented by the official Royale SDK software directly available at [73], which contains some relevant sample scripts to be used to understand the behaviour of the sensor at hand. After having analyzed and studied them, I tried to understood which were the relevant data among the ones captured from the sensor itself in order to fulfil the task aim of this research. These turned out to be the amplitude image caught by the sensor at a certain modulation frequency, along with the depth map of the same scene, taken at the same frequency, and the point cloud based on 38304 points accordingly generated.

Thus, after having given the user the possibility to select the desired operating mode between the ones provided by the camera (5 *fps*, 10 *fps*, 15 *fps*, 25 *fps*, 35 *fps* and 45 *fps* respectively), the "main" program starts two threads in parallel that overlap. The first out of the two collects the depth maps and the amplitude images at the selected modulation frequency and plots them using OpenCV [16], while the second gathers the point clouds information and draws it (with a bit of transformations to be properly rendered) employing Open3D [107].

For the evaluation of the experiments conducted in this thesis work, it has been considered the usage of a 3D printing of the ESA's Meteosat Third Generation (MTG) [29] satellite model, kindly offered by the company this thesis is done with. In fact, since the whole point of this thesis is that of working in an embedding environment, more specifically in an aerospace robotic embedding environment, all the experiments are managed with the purpose of labouring with (a model of) a satellite. This is done in order to simulate what could happen in a real world condition, with an object resembling as much as possible what a real docking system could face in space, that is, in this case, a satellite indeed.
Fig. 5.2 shows the front and the retro views of the 3D model: after having printed it in a

[2] Link to the GitHub repository



(a) Frontal View (b) Retro View

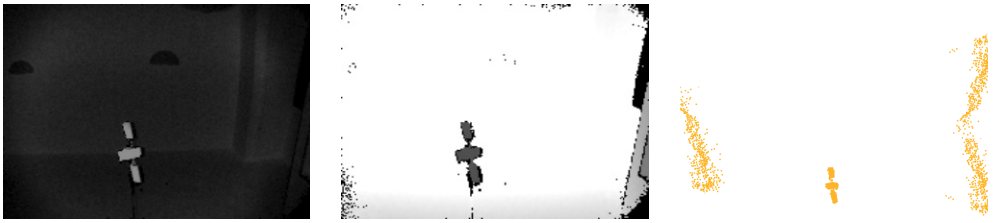
Figure 5.2: **"MTG Satellite 3D Model, in TASI Courtesy".** Frontal (a) and Retro (b) views of the MTG satellite 3D model used in this thesis work.

5 hours process, it has been fully sanded and white re-coloured in order to make it as less reflective as possible, so not to penalize the performance of the PicoFlexx camera, which is particularly sensible to this kind of surfaces as all ToF sensors.

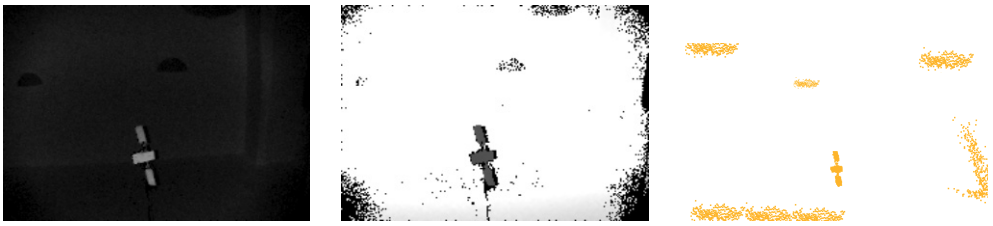
Fig. 5.3 reports the amplitude images and the depth maps (depicted through the usage of OpenCV) and the point clouds (drawn instead with the tools offered by Open3D), acquired observing the satellite model (that was the only object in the room, hanging on from the ceiling) at different operating modes at a nominal distance of $1\,m$. Specifically, the operating modes offered by the camera and available to be selected are briefly explained in the following, according to the information provided by the official guide in [73]:

- **MODE_9_5FPS_2000**: Fig. 5.3(a), (b) and (c) refer to this use case, that focuses on a range between $1\,m$ and $4\,m$, with a frame rate of 5 *fps* and a maximum exposure time of $2000\,\mu s$, and originally thought for indoor room reconstruction.
- **MODE_9_10FPS_1000**: Fig. 5.3(d), (e) and (f) describe this use case, which is set on a $1 \div 4\,m$ range, with a frame rate of 10 *fps* and a maximum exposure time of $1000\,\mu s$, and it is normally employed for room scanning and indoor navigation.
- **MODE_9_15FPS_700**: Fig. 5.3(g), (h) and (i) regard this use case, thought for 3D object reconstruction with a range between $0.5\,m$ and $1.5\,m$, a frame rate of 15 *fps* and a maximum exposure time of $700\,\mu s$.
- **MODE_9_25FPS_450**: Fig. 5.3(j), (k) and (l) cover this use case, exploited for medium size object recognition and face reconstruction within a range of $0.3 \div 2.0\,m$, with a frame rate of 25 *fps* and a maximum exposure time of $450\,\mu s$.
- **MODE_5_35FPS_600**: Fig. 5.3(m), (n) and (o) represent this use case, developed for remote collaboration, step by step instruction and table-top gaming, working in a range between $0.3\,m$ and $2.0\,m$ with a frame rate of 35 *fps* and a maximum exposure time of $600\,\mu s$.
- **MODE_5_45FPS_500**: Fig. 5.3(p), (q) and (r) lastly have to do with this use case, applied within a range of $0.1 \div 1\,m$, with a frame rate of 45 *fps* and a maximum exposure time of $500\,\mu s$, for small object or product recognition and hand tracking.

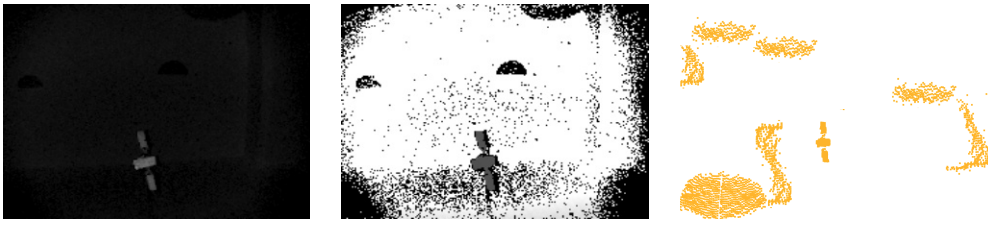
(a) Amplitude 5fps 2000. (b) Depth 5fps 2000. (c) Point Cloud 5fps 2000.



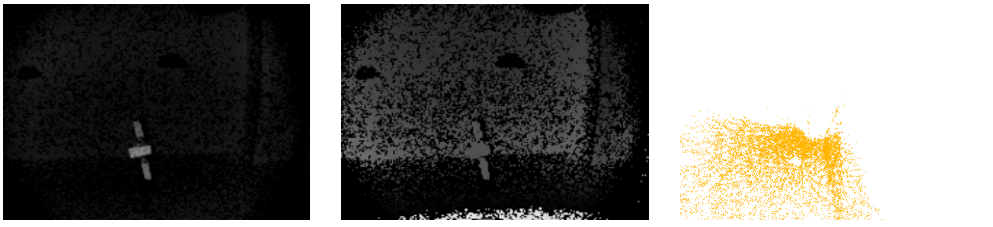
(d) Amplitude 10fps 1000. (e) Depth 10fps 1000. (f) Point Cloud 10fps 1000.



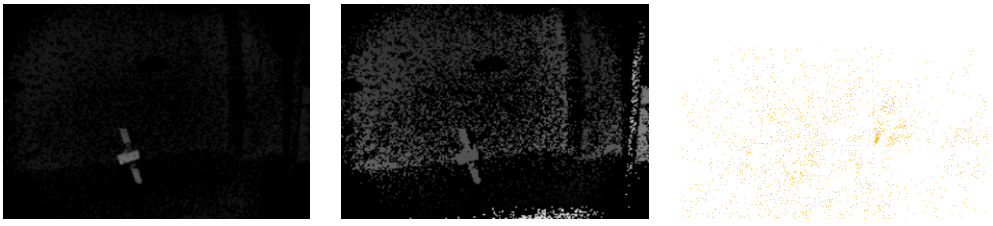
(g) Amplitude 15fps 700. (h) Depth 15fps 700. (i) Point Cloud 15fps 700.



(j) Amplitude 25fps 450. (k) Depth 25fps 450. (l) Point Cloud 25fps 450.



(m) Amplitude 35fps 600. (n) Depth 35fps 600. (o) Point Cloud 35fps 600.



(p) Amplitude 45fps 500. (q) Depth 45fps 500. (r) Point Cloud 45fps 500.

Figure 5.3: **"Raw Data Acquisition: Satellite Model in TASI Courtesy".** Data acquired at different operating modes (5 *fps*, 10 *fps*, 15 *fps*, 25 *fps*, 35 *fps* and 45 *fps*).

In particular, as it is possible to perceive by looking at the figure, higher *fps* correspond to a higher level of noise, but at the same time to a faster data acquisition. Indeed, even if with $fps \geq 25$ one could enjoy a fast acquisition, the results are visibly clattering: sometimes even the only object present in the room (the 3D printing of the satellite model) couldn't be seen due to the high level of noise. Thus, it is feasible to state that the 5 *fps* use case is optimized for long range scanning at a maximum data quality, in order to locate objects or people inside large environments, such as buildings, while on the other side a frame rate of 45 *fps* could lead to a less data quality for precise detection and recognition, but a huge processing speed.
In this thesis work only input data with a low frame rate have been considered, that is up to 25 *fps*, in order to capture quite clean depth maps.

5.2.2 ToF MPI and Shot Denoising

After having compared all the deep learning approaches for ToF raw data denoising according to six different metrics as shown in Table 5.1, two of them have been selected in order to be reproduced and adapted to the needs of this thesis. *Coarse-Fine CNN* proposed in [3] and its enhancement designed in [2] are indeed the two methods that met both the requirements of being straightforward and presenting a low latency, still achieving state-of-the-art performance in terms of accuracy and presenting a good robustness and a wide range of application. The idea was thus to replicate them, adapting them to the PicoFlexx sensor kindly provided by the company this research is done with.

After a brief conversation with Eng. PhD. Gianluca Agresti, the first and main author of the two cited papers, it has been decided to implement just one neural network method, that basically mixes together the two models, introduced in [3] and [2] respectively, taking the structure of the network from one side and the training approach from the other. In this way, indeed, it has been possible to implement a method which does not occupy too much memory neither for the structure itself nor for the training on the augmented dataset. Besides providing a solution for a robotic environment in fact, which has to be as much straightforward as possible *per se*, the whole method had a further constraint, as well as all the other approaches used and designed in this thesis work. It must have been as less complex as feasible since it has been implemented and trained in Google Colaboratory [14], therefore with several limitations for what concerns the RAM and the time available for the training itself.

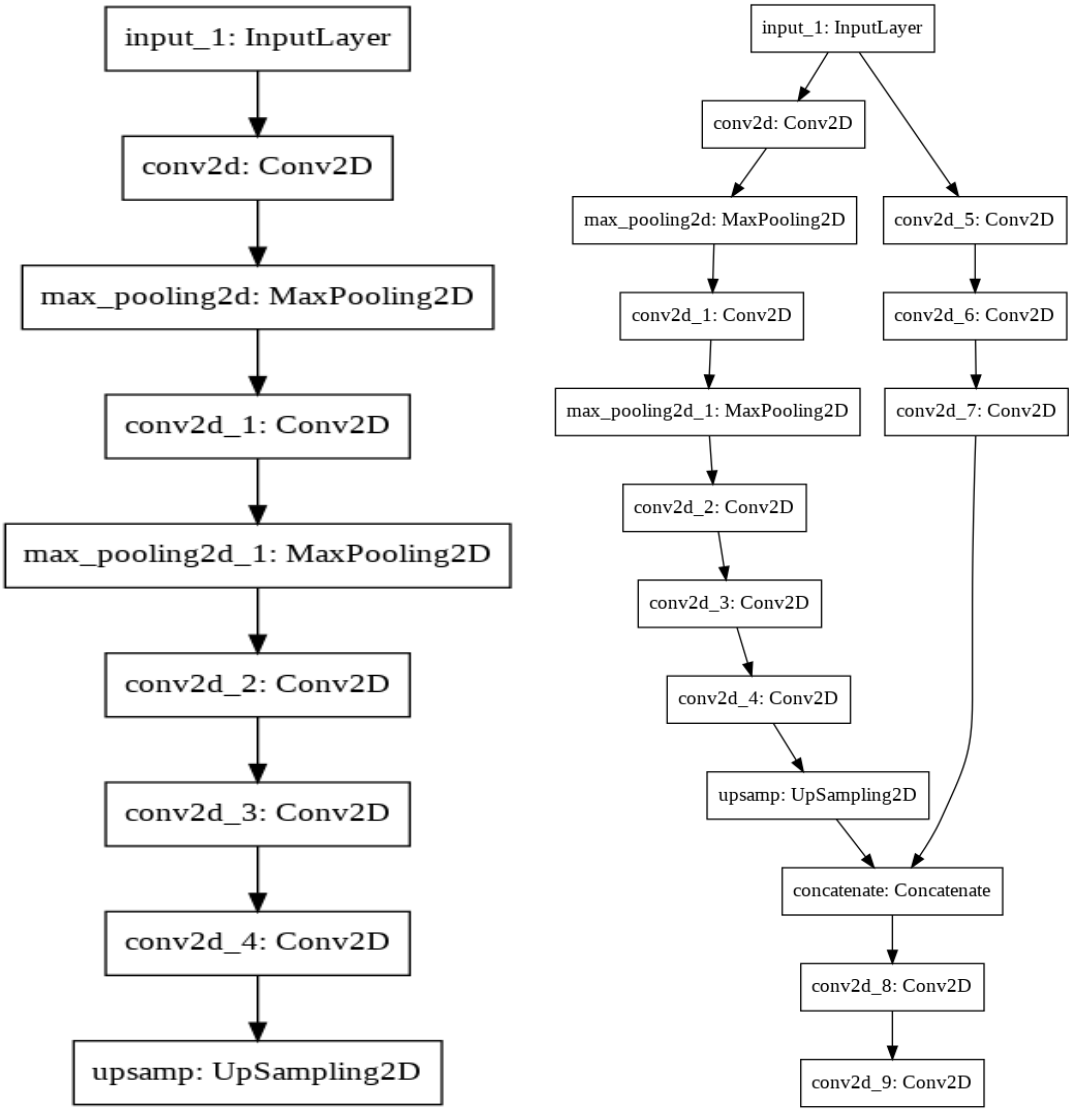
After having implemented this method though, a not negligible problem emerged. The proposed approach, resulting from the integration of the two presented papers, is indeed very fast in getting the results (that is, in denoising the input depth map), but it always needs as input a stack of five elements. This sequence is constituted by the depth map acquired at $60MHz$, the differences between the depth maps captured at $20MHz$ and $60MHz$ and between the ones captured at $50MHz$ and $60MHz$, and finally the ratios between the amplitudes at $20MHz$ and $60MHz$, and between the ones at $50MHz$ and $60MHz$, as thoroughly explained in [3]. Thus, it cannot work in real-time (or near-real-time, for what matter) because it consistently requires to pre-process the input data after having acquired them at three different modulation frequencies, even in inference mode.
Since it is crucial that in the type of embedding applications considered in this thesis

work the implemented framework is capable of running in (near-)real-time (with a hardware powerful enough to let it happen obviously), thus presenting a very low latency, this method cannot no more be considered as a valid solution for the kind of task that has been required.

Therefore, this is the reason why a second approach has been taken into account, that is the one proposed in [27], denoted *SHARP-Net*. While not being the best method according to the selected metrics, as already anticipated and previously summarized in Table 5.1, its low complexity (or better saying, its capability of obtaining fine results even with a simpler architecture with respect to the one originally thought by the authors) and its online performance are still good enough to allow its usage in this research.
Thus, in the following two paragraphs, the main characteristics of the proposed architectures are described in detail, both for what concerns (enhanced) *Coarse-Fine CNN* and *SHARP-Net*. Results of the totality of the conducted experiments are finally available in Chapter 6.

Coarse-Fine CNN. Following the network proposed in [3] and its further improvement in the form of the generator network of the GAN described in [2], a *Coarse-Fine CNN* architecture has been implemented. Specifically, the coarse network, based on a sequence of five convolutional layers, each followed by a ReLU as activation function with the only exception of the last one, is fed with the five data channels previously mentioned. This stack is constituted by the depth map captured at $60MHz$, the differences between the depth maps acquired at $20MHz$ and $60MHz$ and between the ones obtained at $50MHz$ and $60MHz$, and finally the ratios between the amplitudes taken at $20MHz$ and $60MHz$, and between the ones gained at $50MHz$ and $60MHz$. After each of the first two convolutional layers a max pooling layer is present, aiming at applying a down-sampling operation in order to decrease the resolution by a factor of 2. Each kernel of the convolutional layers implements a 3×3 pixels convolution. Every one of these layers present 32 filters, a part from the last one that employs a single 3×3 convolutional filter. Differently from [3], after the final upsampling employing a bilinear interpolation the output of the coarse CNN is not the original esteemed degradation due to the presence of *MPI*. Following instead the approach introduced in [2], this up-sampled output can directly be seen as a low resolution depth map measurement of the observed scene. This is of paramount importance since it enables to avoid the employment of further computationally intensive filtering operations, such as the bilateral filter which is instead used in many frameworks for ToF raw data denosing. Fig. 5.4(a) shows the overall graph of the implemented coarse sub-network.
As suggested by the authors, the decision to employ two different sub-networks is highly dependent on the observation that the reflections causing *MPI* could theoretically take place in many different regions of the observed scene, thus a broad field of view is indispensable. Conversely, convolutions and pooling layers employed by the coarse CNN have the side effect of making edges and small details partially blurred. Therefore, while the coarse network allows to perceive a wide receptive field, the fine network works instead at full resolution and permits to get a final more precise representation of edges and details, thus empowering a better localization of the error.
Fig. 5.4(b) depicts the graph of the whole Coarse-Fine network implemented. It is possible to see that also this second network presents five convolutional layers with 3×3

convolutions and ReLU activation functions, except for the last one, exactly as in the previous network. Nonetheless, instead of 32 filters per layers it works with 64 filters for each layer ans no pooling blocks at all. Moreover, the input of the first layer is again the stack of five data channels as previously described (thus, the same of the Coarse network), but the fourth layer is fed with both the output of the third layer and the up-sampled coarse CNN output. This allows to combine the "global" low resolution esteemed depth map of the previous network with the more detailed but "local" estimation performed by the fine network. In this way, it is feasible to achieve an *MPI*-free depth map measurement which retains both the global scene structure and the finer details.



(a) Graph of Coarse CNN

(b) Graph of Coarse-Fine CNN

Figure 5.4: **"Graphs of the implemented Coarse-Fine CNN"**. Coarse CNN(a) and Coarse-Fine CNN(b) graphs obtained through the "ad hoc" code implementation, after having studied and re-implemented from scratch [3] (with the partial integration provided by [2], on the guide of the enlightening advice offered by Agresti).

Again contrastingly from [3], the input data is fed without any pre-processing for denoising to the *Coarse-Fine CNN* architecture.
The neural network has been trained using the synthetic dataset provided in [3]. Despite the fact that it is one of the most substantial ToF dataset currently available, its size is still relatively small-scale if put together with the datasets commonly employed for deep learning neural networks training. Thus, with the aim of handling this non negligible problem and avoid overfitting during the training procedure, following the suggestions of the authors also in this thesis work data augmentation techniques have been applied on the data. More in detail, 10 128×128 pixels random patches have been extracted from each of the 40 scenes present in the dataset, and each of these patches has been further rotated of ± 5 degrees, as well horizontally and vertically flipped. Lastly, due to the small amount of data, K-fold cross-validation with $K = 5$ has been used on the training dataset in order to validate the network hyper-parameters. Once the hyper-parameters have been properly selected though the cross-validation procedure, the CNN has been trained on the totality of the training data.
For the training itself, following again the approach introduced in [2] instead of the one presented in [3], a combined loss has been minimized. This loss is made by the sum of two terms, one computed on the output of the coarse network (after interpolation) and the other calculated on the outcome of the second fine network. Specifically, it is designed as the sum of the ℓ_1 distances between the outputs of the fine and the coarse networks with respect to the ground truth depth value, according to:

$$\mathscr{L}_{sup} = E[|d_C - d_{gt}|] + E[|d_F - d_{gt}|] \tag{5.1}$$

During the training, ADAM optimizer [52] was used, together with a batch size of 4 and an initial set of weight values calculated by using Xavier's procedure [34].
The learning rate has been set to $5 \cdot 10^{-6}$ and a ℓ_2 regularization with a weighting factor of 10^{-4} has been employed for the norm of the CNN weights.
The overall network has been implemented using TensorFlow [1] and trained using Google Colaboratory [14].
Fig. 5.5, taken from [3], depicts the 14 scenes of the synthetic dataset kept out for the evaluation part and furnishes a glance of the dataset at hand.
However, as has already been said, this network cannot work in real-time even in inference mode, since it always requires to be fed by a sequence of five input data channels acquired at different modulation frequencies. This is of course unfeasible to be done "online", thus another approach has been taken into account, that is *SHARP-Net* [27], described in the next paragraph.

Figure 5.5: **"Synthetic Test Dataset of Coarse-Fine CNN"** (taken from [3]. The 14 scenes (in a colored perspective) of the dataset provided by Agresti et al, used for the evaluation stage of the Coarse-Fine CNN.

SHARP-Net. Starting from [8], the network of the relative paper was implemented. Specifically, three architectures have been built throughout the whole thesis work in order to make several comparisons, taking as major indicators the complexity of the models and their online performance, as well as the visual accuracy of the results.
The three approaches taken into account are *SHARP-Net* in its entirety and its smaller variants, namely *ToF-KPN* and*SHARP-Net* without the Residual Fusion and the Depth Refinement modules.
As illustrated in [27] and replicated here, *SHARP-Net* ("Spatial Hierarchy Aware Residual Pyramid Network") is built upon three blocks, namely the Residual Regression Module, the Residual Fusion Module and the Depth Refinement Module. The implemented Residual Regression Module is an encoder network that takes as input the combination of the depth map and the amplitude image acquired at a certain modulation frequency and returns as output a multi-scale feature pyramid consisting of five layers with 16×16, 32×32, 64×64, 128×128 and 256×256 filters respectively. Given an input image of 224×171 (that in the resolution of the PMD PicoFlexx sensor), the feature maps at each level i present a size specified by the following formula:

$$S_{FM_i} = \frac{224}{2^{i-1}} \times \frac{171}{2^{i-1}} \times C_i \tag{5.2}$$

where C_i is the number of output channels: 16, 32, 64, 128 and 256 respectively.
At each of these five levels, the module predicts a depth residual map, which is up-sampled via bi-cubic interpolation by a factor of 2 before being concatenated with the feature map of the upper level. The output obtained in this way is thus given as input to a sequence of two convolutional layers, each followed by a max pooling one. The final outcome is exactly the residual map for each of the pyramid levels. All these obtained residual maps are first up-sampled again via bi-cubic interpolation by a factor of 2, then concatenated together and given as input to the second module of *SHARP-Net*, that is the Residual Fusion Module. This block is based on a 1×1 convolutional layer, whose output is the final depth residual map. This outcome is then added to the original input depth image in order to recover the depth image.
Lastly, in order to highly attenuate also shot noise, the final Depth Refinement Module has been implemented as suggested by the authors of the original paper cited above. This block is based on a U-Net model with four convolutional layers with 16×16, 32×32, 64×64 and 128×128 filters respectively, each followed by a max pooling layer. Skip connections are also employed to improve the performance and the convergence of the network. Moreover, the weight matrix has been further computed as a vectorized filter kernel, with a 3×3 size, for each pixel in the depth image. Thus, using three sequential convolutional layers, the weight matrix got a size of $224 \times 171 \times 9$. By vectoring a neighbourhood of 3×3 for each pixel in the depth image, another $224 \times 171 \times 9$ matrix, denoted by the authors of [27] as "Patch Matrix", has been calculated. These two obtained matrices are therefore element-wisely multiplied to obtain the final 3D volume. By finally summing over this 3D volume the refined depth image deprived of *MPI* and shot noise is lastly computed.
Being a supervised neural network, the authors of the paper from which this network architecture is inspired propose the usage of three ToF datasets for the training procedure, namely ToF-FlyingThings3D (TFT3D) [77], FLAT [39] and True Box[2, 3]. All these datasets obviously contain ground truth depth values.

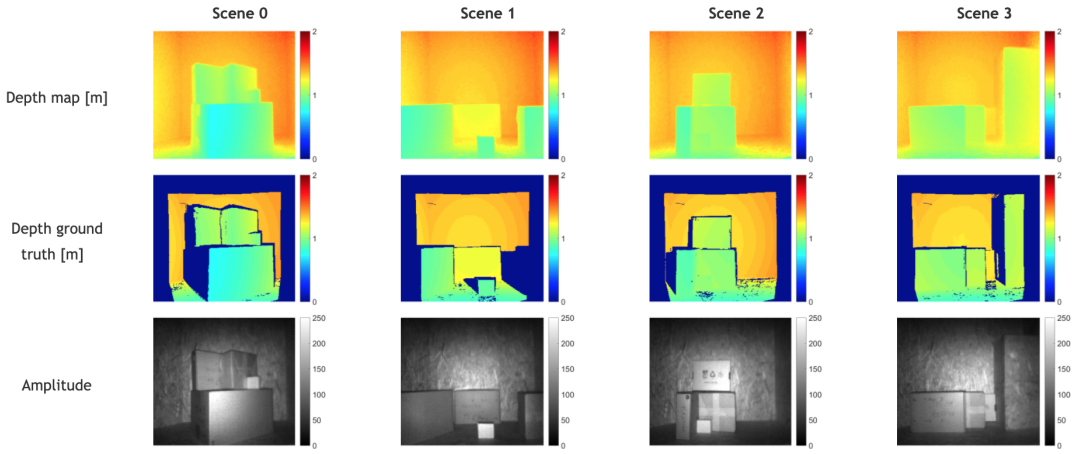


Figure 5.6: **"Examples from the True Box Dataset"** (taken from [2]). Depth maps, depths ground truth and amplitude images for scenes 0, 1, 2 and 3, taken from the "True Box" dataset constructed by Agresti et al. The image is taken from the official cite of the relative paper, available at this link[3].

In this thesis work, only the latter has been used, due to the fact that the resolution of the camera with which the dataset was collected is the closest one to the PicoFlexx sensor. Indeed, True Box [2, 3] contains the depth, amplitude and intensity maps captured with an active stereo and a ToF camera jointly calibrated at 10, 20, 30, 40, 50 and 60 MHz on 8 scenes, thus for a total amount of 48 different scenes, each with a 239×320 resolution. All the data are acquired from the authors in a laboratory without any form of external illumination, and some examples are shown in Fig. 5.6.
The network is implemented using TensorFlow [1] framework with Keras [22] open-source neural-network library, and developed using Google Colaboratory [14].
One of the major drawbacks of this model in its original version is the intricacy of its architecture. Thus, in order to reduce the complexity and make it more feasible to be implemented in an embedding system without sophisticated hardware, two variants of *SHARP-Net* have been considered in this research.
Inspired by the alternative solutions provided by the authors of *SHARP-Net* and following [11], the first of the presented variations is based just on a U-Net structure built upon an autoencoder with four convolutional layers for the encoder part and four transpose convolutional layers for the decoder one. The network, which takes the name of *ToF-KPN*, is fed with an input depth image and extracts the relevant features, used then as input to a stack of three convolutional layers for the weight matrix estimation. After having applied a "Path2Vec" operation to generate a patch matrix by vectoring a neighbourhood for each pixel in the input depth image, the two obtained matrices are element-wisely multiplied, thus generating a 3D output volume. By again summing over it the final error-free depth map is derived.
The second proposed alternative is instead a *SHARP-Net* without the Residual Fusion Module and the Depth Refinement one. Practically speaking, this second surrogate is

[3] Unsupervised Domain Adaptation for ToF Data Denoising with Adversarial Learning

just based on a four-layers encoder that extracts the features from the input depth map and amplitude image concatenated together. After an up-sampling operation via bicubic interpolation, the network returns as output a four-levels features pyramid. Each level of the just obtained output is then fed to a small decoder which estimates the residual depth map, and by considering the totality of all the levels the final denoised depth map is obtained. While simplifying a lot the structure of the model itself, the performed experiments (available in the next Chapter) show that this approach doesn't lose accuracy in the final outcomes, thus providing a valid solution that could be adopted in a space embedding environment.
For what concerns the first alternative instead, the network based only on a U-Net structure and utilizing a second small CNN to combine features and weights, results illustrate that this model is not precise in denoising the input depth map. The conducted tests convey that the *ToF-KPN* network is indeed capable of removing the noise but not in a precise way, sometimes deleting also "wrong" points belonging to the satellite point cloud.
The outcomes of the experiments carried out in this thesis work are shown and explained in detail in Chapter 6, where the three alternatives are qualitatively and quantitatively compared by acquiring input depth images of the satellite model at different frame rates, i.e., 5 *fps*, 10 *fps*, 15 *fps* and 25 *fps*.

5.2.3 Point Cloud Registration

This Subsection focuses instead on the implementation of the code for the point cloud registration task. In particular, out of all the proposed approaches considered in Chapter 3, only few of them have been selected, one per each typology of methods (or two for the last case), according to the already cited six metrics, as reported in Table 5.1. These are the Fast Global Registration [106] (*FGR*) for the optimization-based methods, the FPFH [79] feature extraction + RANSAC [31] Global Registration (*FRR*) for the feature-learning models and finally *3DRegNet* [70] and Feature-Metric Registration [47] (*FMR*) for the end-to-end learning-based frameworks.
Even if this thesis work has the purpose of studying a NN-based approach for tackling the point cloud registration problem, the first two methods were chosen in order to make a fair comparison between "standard" traditional approaches and new ones based on the recent evolving AI, to prove that deep learning techniques could be used even in a surely challenging space environment.

The workflow performed for addressing this task is common to all the selected approaches, and it is briefly explained in the following. First of all, the CAD 3D model of the satellite (again kindly provided by the company) has been uploaded and converted into a point cloud using Open3D library [107]. In order to make several experiments, also taking into account the time spent by each of the proposed algorithms for the final alignment, the target point cloud of the satellite has been created by gradually sampling a different number of points, i.e., 20000, 30000, 50000 and 100000, according to the Yuksel's procedure [101]. This technique basically samples points from a given mesh (that is the converted CAD model) in order to generate a certain point cloud, in which each of the sampled points is round about at the same distance to all the points belonging to its neighborhood. This property is denoted as *blue noise*.

(a) Satellite point cloud with 20000 points sampled from the 3D CAD model.

(b) Satellite point cloud with 30000 points sampled from the 3D CAD model.

(c) Satellite point cloud with 50000 points sampled from the 3D CAD model.

(d) Satellite point cloud with 100000 points sampled from the 3D CAD model.

Figure 5.7: **"Sampling Strategies: Satellite Model in TASI Courtesy".** Several point clouds obtained by the 3D satellite model through the sampling procedure introduced by [101], with 20000(a), 30000(b), 50000(c) and 100000(d) points respectively.

Visual outcomes of this sampling procedure are depicted in Fig. 5.7. In particular, Fig. 5.7 (a) shows the point cloud of the 3D CAD satellite model composed of 20000 sampled points, (b) the one made by 30000 points, (c) the point cloud assembled sampling 50000 points and finally (d) the one formed with 100000 points.
As it is possible to notice by looking at the images, the higher is the number of points the higher the "resolution" of the point clouds, that are better able to fully describe the shape of the satellite itself. On the other side, the sampling process of a point cloud with a higher number of points clearly requires more time with respect to the creation of a smaller point cloud. Quantitatively speaking, the implemented script took 38.365 seconds (as average) to generate the last point cloud with 100000 points, while only 6.122 seconds to create the first point cloud by sampling 20000 points. Table 5.2 reports the time costs required for the sampling procedure of each of the considered point clouds.

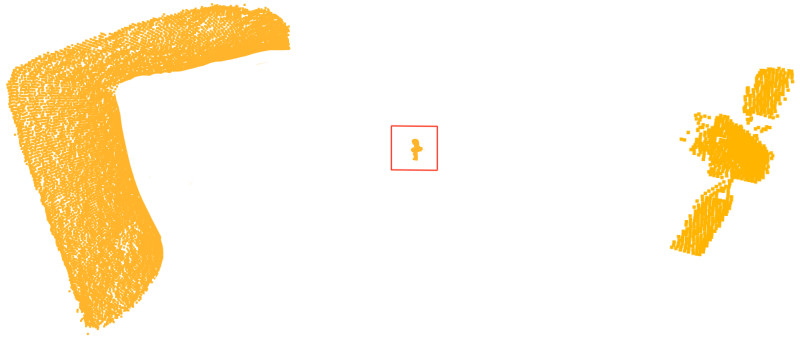
Once the target model is uploaded and sampled, thus "converted" into a point cloud with the selected amount of points, all the 3D registration frameworks considered in this thesis work just need the second (source) point cloud to start performing the alignment between the two. After the denoising phase performed by *SHARP-Net* and its variants, as discussed in the previous Subsection 5.2.2, the obtained error-free point cloud is given as input to the registration algorithms. Before effectively beginning the alignment task, the source point cloud is further cropped so to remove the background, in order to let the registration technique implement its strategy by focusing only on the relevant object, which is the

Execution Time of the Sampling Strategy [101]				
	Target Point Cloud Dimension			
	20000 points	**30000 points**	**50000 points**	**100000 points**
Time Cost	6.122 s	9.557 s	20.320 s	38.365 s

Table 5.2: **"Execution Time of the Sampling Procedure".** Time costs for sampling the 3D CAD satellite model following the technique described in [101], calculated as the average value of 5 different runs. The considered target point clouds have growing dimensions: 20000, 30000, 50000 and 100000 points respectively.

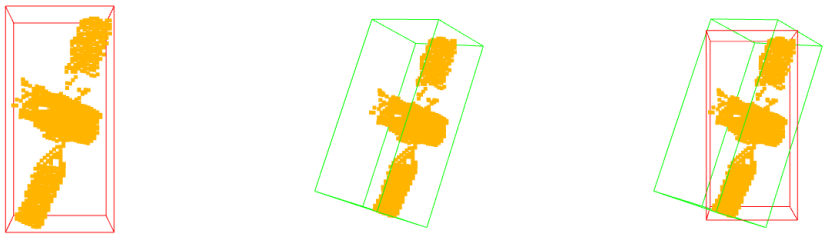
satellite in this case. Indeed, while being the satellite model the only object present in the observed scene, the distance between the target and the neighboring walls lies within the working range of the PicoFlexx camera, therefore they couldn't be removed during the acquisition. Thus, again through a function offered by Open3D library [107], starting from a custom *.json* file containing the polygon volume to be cropped, written "ad hoc" for this situation, the source point cloud is detected and all the background points are cut off accordingly. At this point, the point cloud embracing all the points belonging to the satellite printed model is finally ready to be fed to the registration frameworks. Fig. 5.8 shows the satellite model before (a) and after (b) this cropping procedure: as it can be seen, at the end only the points affiliated to the satellite are retained in the final point set.

Before the final stage, that is the execution of the registration algorithm itself, object detection has been performed on the source point cloud in order to detect the satellite model and to enclose it in bounding boxes. In particular, two different bounding boxes have been created for this object detection task, as can be seen by looking at Fig. 5.9.
First of all, from the coordinate axes a first bounding box that encloses the whole set of points was created (shown in Fig. 5.9(a) coloured in red). Then, starting from the previous one, a second oriented bounding box was generated (depicted in green in Fig. 5.9(b)).



(a) Source point cloud before cropping. (b) Source point cloud after cropping.

Figure 5.8: **"Cropping of the source input point cloud: Satellite Model in TASI Courtesy".** Source point cloud before the cropping phase (a) and immediately after (b). As it is possible to notice, after the cutting phase all the points belonging to the background are removed and only the ones belonging to the satellite printed model are retained.



(a) Axis-Aligned Bounding Box (shown in red).

(b) Oriented Bounding Box (depicted in green).

(c) Axis-Aligned and Oriented Bounding Box.

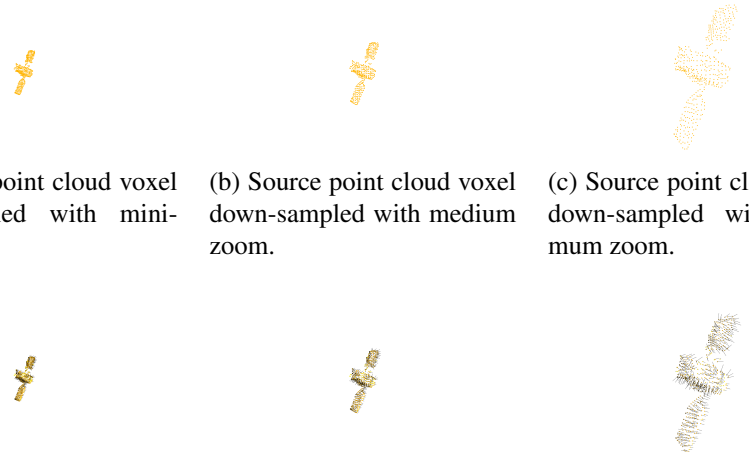
Figure 5.9: **"Object Detection on the Source Point Cloud: Satellite Model in TASI Courtesy".** Axis-Aligned Bounding Box over the source point cloud drawn in red (a), Oriented Bounding Box in green (b) and the two Bounding Boxes together (c). The object detection task has been carried out following [107].

Fig. 5.9(c) illustrates both the axis-aligned and the oriented bounding boxes around the observed point cloud. This object detection task has been implemented and accomplished, shadowing [107], in order to properly narrow the Region of Interest (*ROI*). All of this has been done before passing the source input point cloud to the registration framework to carry out the final goal of this thesis work, which is the estimation of the transformation parameters capable of aligning the input to the target point sets.

At this point, both the source and the target point clouds are finally ready to begin the registration alignment task. One last note that has to be made before going to the explanation of the implemented code is that out of the two chosen and therefore coded end-to-end learning-based procedures, namely *3DRegNet* [70] and *FMR* [47], the latter is the one finally employed for the experiments in this thesis work. This is due to the fact that *3DRegNet* presented a severe drawback which was encountered during the implementation itself, as will be explained in the related paragraph later on. It is indeed designed to align only two input point clouds with exactly the same number of points. Thus, the 3D CAD model of the satellite and the source input point cloud captured from the PicoFlexx sensor should have been equal in terms of size and dimension, and this could have been forced through down-sampling the target model to always be consistent with the source one, regardless of the number of points belonging to the latter. But in doing that, the performance has encountered a significant drop during the conducted experiments, since the source point cloud was already partially cropped before the execution of the algorithm (due to the reasons explained before to remove the background). Thus, the total number of points of the cropped point set wasn't high enough to lead to a good result in the registration task, and this is the reason why the other approach has finally been taken into account.

Fast Global Registration (FGR). The first approach that has been implemented is the optimization-based chosen candidate, namely Fast Global Registration (*FGR*) [106], following the script provided by the Open3D library [107].
At the beginning of this method, both the input point clouds have been down-sampled with a voxel size of 0.05.

This technique, described in [107], starts from the employment of a structured voxel mesh in order to generate a uniformly down-sampled point set starting from an input one. The procedure is based on a stack of two sequential stages: (1) the points are firstly wheeched into voxels, and secondly (2) each of the obtained voxels spawns just one point by computing the mean value of all the points belonging to the voxel itself. At this point, the normals for every obtained point are calculated by first retrieving the closest points (within a searching radius of 0.1, which corresponds to 10 *cm*, and a maximum nearest neighbors' number of 30 to cut computational costs). Through the usage of covariance analysis, the principal axis of these closest points is therefore quantified.
Worthy of note is that the analysis of covariance is a technique that outputs two converse directions as normal candidates. Both of them could in principle be considered as the right direction if no information regarding the global geometry structure is supplied. This is referred to as the "normal orientation issue": the algorithm brought forth by [107] and followed in this thesis aims at lining up the obtained normal with the native actual one, provided that it is already present. If this is not the case instead, the procedure performs a random prediction.
Fig. 5.10 reports the voxel down-sampling and the normals computation process just described: in particular, Fig. 5.10(a), (b) and (c) display the voxel down-sampling technique applied on the acquired satellite input point cloud, with a voxel size of 0.05, while Fig. 5.10(d), (e) and (f) convey the normal computation. In both down-sampling and normal calculation the figures are shown in an increasing enlargement: Fig. 5.10(c) and (f) illustrate the source point cloud at full zoom, first down-sampled and then with the just quantified normals.



(a) Source point cloud voxel down-sampled with minimum zoom.

(b) Source point cloud voxel down-sampled with medium zoom.

(c) Source point cloud voxel down-sampled with maximum zoom.

(d) Normal Estimation over down-sampled point cloud with minimum zoom.

(e) Normal Estimation over down-sampled point cloud with medium zoom.

(f) Normal Estimation over down-sampled point cloud with maximum zoom.

Figure 5.10: **"Voxel Down-Sampling and Normal Computation of Point Clouds: Satellite Model in TASI Courtesy".** (a), (b) and (c) show the voxel down-sampling process applied to the source input point cloud. Screenshots are taken at different zoom levels. (d), (e) and (f) depict the normals computation starting from the down-sampled point cloud. Screenshots are again taken at different zoom levels.

Finally FPFH [79] features are computed for every point, using a searching radius of 0.25 and a maximum number of nearest neighbors set to 100. Each FPFH can be thought as a feature vector with dimensionality 33, capable of fully understanding and thus representing the local intrinsic characteristics of a single geometric point. The main idea behind its behaviour is that the closest neighbors in the 33-dimensional space to a specific point are all characterized by an analogous geometric structure.
Thus, with the computed features and the down-sampled point clouds previously obtained, the Fast Global Registration [106] algorithm is called using again an Open3D smart implementation [107], obtaining the final alignment. The qualitative and quantitative results of the driven experiments using *FGR* algorithm are discussed in Chapter 6.

FPFH + RANSAC Global Registration (FRR). As a deputy of the feature-learning registration frameworks, FPFH [79] + RANSAC [31] Global Registration (*FRR*) has been selected for its low complexity and yet the high accuracy of its results. Again following the script provided by Open3D library [107], the first part of this algorithm is identical to the one already explained in the previous paragraph for the *FGR* approach. Specifically, after having chosen a voxel size equal to 0.5, both the source and the target point sets are down-sampled and the normals are calculated for every one of their points.
One could have a look at the just described Fig. 5.10 to visually perceive the outcomes of this step, that is indeed analogous to the optimization-based one thoroughly introduced in the previous paragraph.
Starting from the down-sampled point clouds and the normals just obtained, FPFH [79] features are extracted according to the Open3D implementation of the technique, exactly as before. At this point the global RANSAC [31] algorithm based on feature matching is called, with the maximum correspondence points-pair distance set to 0.075, the number of correspondences to fit RANSAC with set to 4 and the "Point-To-Point" transformation as the estimation method.
The most dominant hyper-parameter is the convergence criteria, that indicate the maximum number of iterations and the maximum amount of validation steps that the called method has to perform before stopping. The higher are these two values, the more precise will also be the final outcome, but at the same time the larger will be the execution time employed by the algorithm to converge. In the implemented code they are set to 4000000 and 500 respectively.
At each iteration, 4 random points are gathered from the source point set and, starting from them, the corresponding points belonging to the template point cloud are retrieved. This is achieved by looking at all the closest neighbors in the 33-dimensional feature space that present a similar geometric structure with respect to a given one.
A clipping stage is further performed in order to rapidly discard false pairing prematurely in the pipeline, through the usage of high speed trimming algorithms.
In particular, following the [107] approach, two pruning algorithms are used in this thesis work: (1) *"Correspondence Checker Based On Distance"* tries to understand whether aligned point clouds are adjacent (within a predefined threshold, that in this research has been set to 0.075) or not, and (2) *"Correspondence Checker Based On Edge Length"* examines whether the lengths of two arbitrary lines formed by two vertices separately extracted from source and target matches are alike or not.

Specifically, the implemented code checks that both the following inequalities hold:

$$\begin{aligned} \|\text{edge}_{source}\| &> 0.9 \cdot \|\text{edge}_{target}\| \\ \|\text{edge}_{target}\| &> 0.9 \cdot \|\text{edge}_{source}\| \end{aligned} \tag{5.3}$$

Thus, only those pairings that are compliant with the pruning rules defined by these algorithms are finally employed to estimate the transformation.
FRR is fed only with (densely) down-sampled point sets, thus the obtained outcome may be not well matched. This is the reason why, inspired by the Open3D procedure, a final "Point-to-Point" ICP is additionally employed to refine the esteemed transformation.
Nonetheless, one last note that has to be made is that, as previously anticipated in Subsection 3.2.4, there is no a maximum limit on the time this approach employs to converge. Thus, when the number of iterations is finite and bounded, the solution may be sub-optimal or not particularly accurate. This will be further discussed in next Chapter where the results of *FRR*, compared with the ones of other frameworks, are illustrated.

3DRegNet. The first proposed method for the end-to-end learning-based approaches is *3DRegNet*, introduced in [70]. Following the method described in the relative paper, the network is based on two main modules, one for the classification and one for registration. The classification chunk is fed with a collection of 3D point correspondences between the two scans. Despite the fact that these correspondences could in principle be calculated using any technique, in this thesis work the FPFH [79] features have been used. Each of these computed correlations is then fed to a fully-connected (FC) layer followed by 128 ReLU activation functions. Furthermore, a weight sharing for all the N separate correlations has been implemented, obtaining as output a $N \times 128$ vector, due to the fact that from every point correspondence 128 dimensional features are extracted. The $N \times 128$ output is then further given as input to 4 ResNets, followed at the end by a final FC layer with ReLU and tanh activation functions.
The input to the second registration block is constituted by the features just extracted. A max pooling layer, followed by context normalization, draws out relevant 128×1 features from each of the classification block's layers. These features are then concatenated and fed to a convolutional layer with 8 filters, each presenting a 3×3 kernel with a stride of 1×2. The derived output is finally given as input to two FC layers with 256 filters each, with ReLU activation function between the layers. In this way it is possible to obtain the desirable output, that is composed by the transformation parameters.
In this thesis work the model pre-trained by the authors of the paper [70], publicly available, has been used and slightly modified in order to make it compatible with the input acquired using the PicoFlexx sensor. While adapting it though, tailoring to the needs of this research, a problem emerged: the framework works only when the two input point clouds have the same dimension in terms of the number of points. Besides representing a heavy constraint, this has also caused a serious degradation of the performance obtained in the registration task. Indeed, given that the source point cloud was already cropped before starting the registration procedure (as previously explained at the beginning of Subsection 5.2.3) and that the resolution of the PicoFlexx camera is already low (224×171), forcing one of the two point clouds to always be consistent with the dimension of the other has led to a significant drop of the accuracy. This is the reason why the approach defined in the following paragraph, besides being more straightforward, is also the final one that has been actually used to carry out all the experiments in this thesis.

Feature-Metric Registration (FMR). The last method presented in this section is *FMR* [47], the second candidate that has been selected for the end-to-end learning-based approaches. As described in the corresponding paper, the architecture of the model is based on a simple autoenconder trained either in a semi-supervised or in an unsupervised manner. Specifically, the encoder is based on a stack of two Multi-Layer Perceptron (MLP) layers followed by a max pooling one, and its output is composed by the rotation-attentive extracted features. Starting from these encoded idiosyncratic features, a decoder is employed to track down the original 3D point sets. This decoder module is based on two layers, each followed by a batch normalization and using Leaky ReLU as activation function. A last FC layer is further added, with tanh as activation function. The autoencoder, counting a total amount of just round about 340000 trainable parameters, is trained by using the Chamfer distance loss (see more details in [47] or in the previous Subsection 3.3.6 where the framework is thoroughly explained) in an unsupervised fashion.

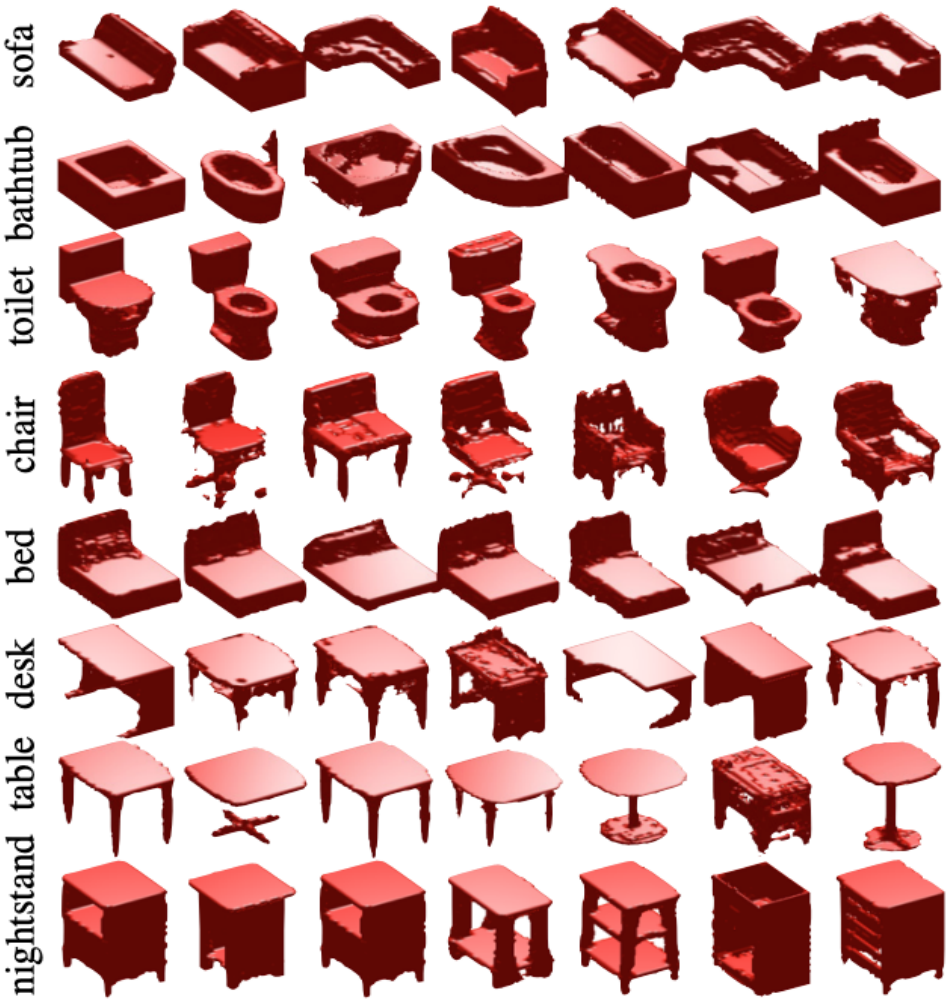
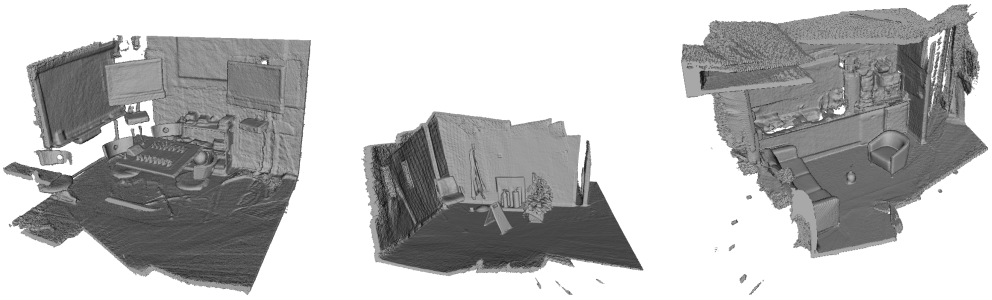


Figure 5.11: **"ModelNet40 Dataset Examples"** (taken from [97]). Some instances extrapolated by ModelNet40, representing a sofa (a), a bathtub (b), a toilet (c), a chair (d), a bed (e), a desk(f), a table (g) and a nightstand (h).

Shadowing the authors' specifics, the datasets used for training are ModelNet40 [97] and 7Scene [82]. Specifically, ModelNet40 includes 40 distinct categories of CAD models, which are divided into two separate fractions: 20 classes are used for training and testing within the same class, while the remaining 20 are employed for testing across different categories. In the former tests the dataset is further divided, following [47]'s procedure, into 8 : 2, where 80% of the data is dedicated to training and the remaining 20% to testing. Some samples extracted from this dataset can be seen by looking at Fig. 5.11 (taken from [97]), representing respectively a sofa (a), a bathtub (b), a toilet (c), a chair (d), a bed (e), a desk (f), a table (g) and a nightstand (h).
7Scene is instead a dataset for indoor environment extensively employed as a benchmark in registrations tasks. It is composed by seven scenes, namely Chess, Fires, Heads, Office, Pumpkin, Redkitchen and Stairs, for a total amount of 353 screenings. They are split into 296 for training and 57 for testing. For the training procedure, the depth images are cast onto point clouds, and depths belonging to multiple frames are merged through truncated signed distance function (*TSDF*) fusion [102]. Some scans taken from this dataset are available at Fig. 5.12, where (a) shows a chess scene, (b) a fire one and finally (c) a pumpkin scene.



(a) Chess example scene. (b) Fire example scene. (c) Pumpkin example scene.

Figure 5.12: **"7Scene Dataset Examples"** (taken from [66]). (a), (b) and (c) show some example scenes extracted from the 7Scene Dataset [82], representing chess, fire and pumpkin respectively.

The registration problem is then tackled by using the inverse compositional (IC) algorithm to minimize a feature-metric projection error and predict the final transformation. Experiments conducted by the authors have demonstrated that *FMR* is usually able to obtain the best performance within 5 iterations of the IC algorithm, thus this is the number utilized in this thesis work during the implementation. Following [47], the network is coded in PyTorch and re-trained using Google Colaboratory's [14] GPU, a 12GB NVIDIA Tesla K80 with 4.1 TFLOPS.
The model trained on 7Scene Dataset allowed to achieve the best performance, thus this is finally used for the assessment phase. Qualitative and quantitative results of the experiments performed in this thesis about *FMR* are exposed in the next Chapter, where this framework is compared with all the other proposed approaches.

Chapter 6

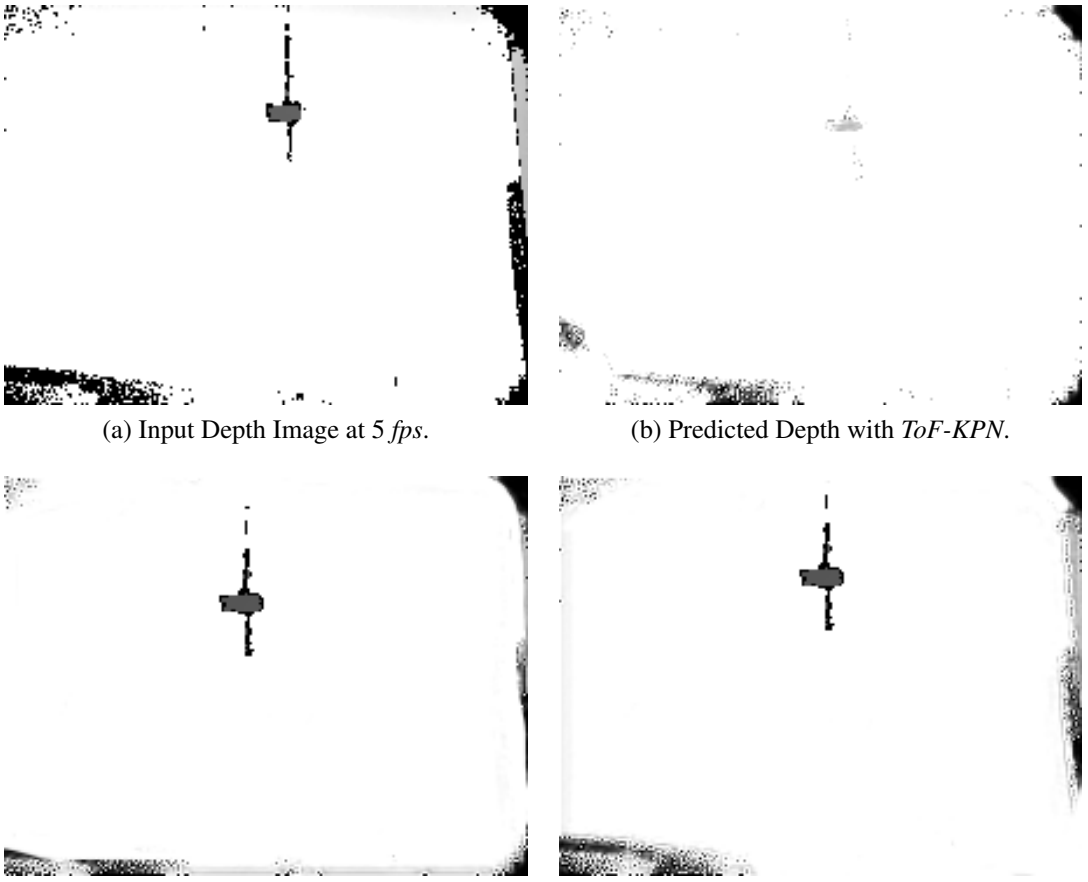
Results

In this final Chapter of the thesis, all the implemented approaches are compared, both from a qualitative and a quantitative point of view. In particular, Section 6.1 refers to *SHARP-Net* and its variants thoroughly described in the previous Chapter (Subsection 5.2.2): the performance of the proposed networks are evaluated with inputs acquired at 4 different frame rates (5, 10, 15 and 25 *fps* respectively). Section 6.2 focuses instead on the point cloud registration task, by taking into account the three introduced frameworks working with target point clouds of different size (20000, 30000, 50000 and 100000 points respectively) and with different initial poses for the source one, always acquired at 5 *fps*.

6.1 MPI and Shot Denoising Results

In this section the performance of the implemented *MPI* and shot denosing approaches (*SHARP-Net* [27] and its variants, namely *ToF-KPN* and *SHARP-Net* without Residual Fusion and Depth Refinement modules) are evaluated both qualitatively and quantitatively and compared one another.

Qualitative Analysis. Figs. 6.1, 6.2, 6.3 and 6.4 display visual comparison between the proposed methods with input point clouds acquired at different frame rate, 5, 10, 15 and 25 *fps* respectively. In particular, these figure show (using OpenCV [16] tools) the depth maps acquired and denoised through the employment of the implemented approaches. As it is possible to see, *SHARP-Net* and its variant without the Residual Fusion and the Depth Refinement modules generally perform similarly, while *ToF-KPN* always brings the worst results. This could be explained given that the latter, built upon a simple U-Net architecture, may be too straightforward to extract relevant features and to understand the inner structure of the point cloud at hand. In fact, as depicted in all the figures, this method is still able to correctly process the input data, partially removing the noise present, but it indiscriminately discards both the inliers and the noisy outliers in the point clouds.
The similarity of the performance between *SHARP-Net* in its original version and the alternative without two building blocks (the Residual Fusion module and the Depth Refinement one) could be interpreted according to the characteristics of the sensor itself. The PMD Camboard PicoFlexx camera is indeed very different with respect to the ones used for building the datasets utilized for the experiments conducted by the authors of [27]: as outlined in Chapter 4, the former isn't affected by the wiggling phenomenon that is instead



(a) Input Depth Image at 5 *fps*.

(b) Predicted Depth with *ToF-KPN*.

(c) Predicted Depth with *SHARP-Net* with no Refine Fusion.

(d) Predicted Depth with *SHARP-Net*.

Figure 6.1: **"Results of MPI and Shot Denoising at 5 fps: Satellite Model in TASI Courtesy"**. Depth Map acquired at 5 *fps*(a) and predicted depth maps with the usage of *ToF-KPN*(b), *SHARP-Net* with no Refine Fusion(c) and *SHARP-Net*(d).

common in many other ToF sensors, thus is could be considered as particularly resistant to shot and especially *MPI* noise. The Depth Refinement module of *SHARP-Net* could therefore be removed without losing almost nothing in performance. Moreover it has to be said that, being the resolution of the PicoFlexx extremely low (224×171, for a total amount of 38304 points), this may lead to a coarse depth map estimation (performed by the *SHARP-Net*'s Residual Regression module) that is better than the one experimented by the authors of the paper this method is taken from. Thus, the contribution of the Residual Fusion block is nearly irrelevant for the final performance of the network.

More in detail, Fig. 6.1 shows the results of the denoising of an input acquired at 5 *fps*: (a) depicts the input depth image, (b) the result obtained applying the simple *ToF-KPN*, (c) the predicted depth map using *SHARP-Net* without the Residual Fusion and the Depth Refinement modules, and finally (d) the outcome achieved with the original *SHARP-Net* as described in [27]. As it is possible to see, the starting point cloud is already almost free of noise. However, *SHARP-Net* with and without the last two modules is able to correctly denoise the input data, visibly reducing shot and *MPI* noise, while *ToF-KPN* removes

(a) Input Depth Image at 10 *fps*.

(b) Predicted Depth with *ToF-KPN*.

(c) Predicted Depth with *SHARP-Net* with no Refine Fusion.

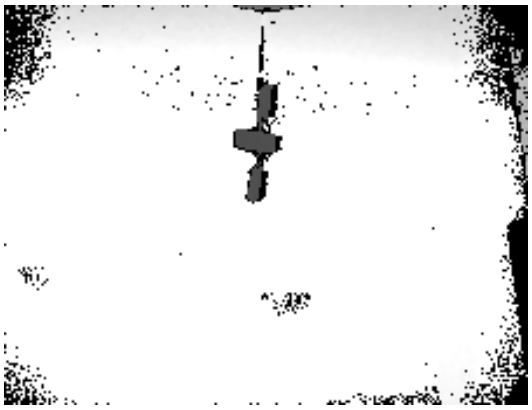
(d) Predicted Depth with *SHARP-Net*.

Figure 6.2: **"Results of MPI and Shot Denoising at 10 fps: Satellite Model in TASI Courtesy"**. Depth Map acquired at 10 *fps*(a) and predicted depth maps with the usage of *ToF-KPN*(b), *SHARP-Net* with no Refine Fusion(c) and *SHARP-Net*(d).

both inliers and outliers compromising also the shape of the satellite itself.

Fig. 6.2 is composed by the same structure of the previous one, with (a) representing the input data and (b), (c) and (d) referring to the three implemented methods, *ToF-KPN*, *SHARP-Net* without the last two modules and the whole *SHARP-Net*, respectively. This time, the input depth image is acquired at 10 *fps*, and in fact it results slightly noisier with respect to the one acquired at 5 *fps* shown in 6.1(a). The outcomes of the denoising task are consistent with what was previously said about the performance of the three approaches, with *ToF-KPN* achieving the worst results out of the three methods. In this case though, the absence of the two last blocks of *SHARP-Net* architecture (c) seems to lead to a slightly better solution than (d).

Fig. 6.3 is again representing the input depth image (a) acquired at a frame rate of 15 *fps* and the three predicted depths in (b), (c) and (d), following the same order of the two previous figures. This time the noise in the depth image is more visible, as one could see by looking at the area around the satellite. *ToF-KPN* (b) is able to remove the noise and partially retain the inliers belonging to the satellite structure, but the result is



(a) Input Depth Image at 15 *fps*.

(b) Predicted Depth with *ToF-KPN*.

(c) Predicted Depth with *SHARP-Net* with no Refine Fusion.

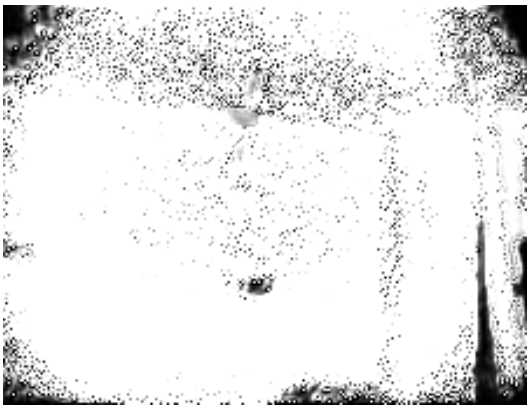
(d) Predicted Depth with *SHARP-Net*.

Figure 6.3: **"Results of MPI and Shot Denoising at 15 fps: Satellite Model in TASI Courtesy"**. Depth Map acquired at 15 *fps*(a) and predicted depth maps with the usage of *ToF-KPN*(b), *SHARP-Net* with no Refine Fusion(c) and *SHARP-Net*(d).

still unsatisfactory. On the other hand, (c) and (d) are able to successfully remove noisy points, depicting a cleaner depth map. Specifically, (c) shows a predicted depth map in which shot and *MPI* noise is for sure attenuated, but also some points belonging to the satellite shape are deleted. (d), depicting the original *SHARP-Net,* is instead very precise in polishing the point cloud while keeping the detected satellite model almost untouched.

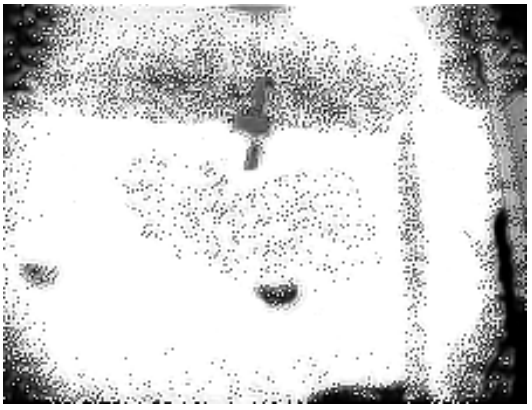
Finally, Fig. 6.4 reports the depth image captured at 25 *fps* (a) and the predicted depths by means of *ToF-KPN* (b), *SHARP-Net* without the last two modules (c) and *SHARP-Net* in its original version (d). As already said, increasing the frame rate at which the input is acquired leads to a significant increase also of the presence of noise in the depth image itself. Indeed, (a) shows a very noisy environment in which the satellite in the foreground is still visible, but the wall in the background and the other (few) objects present in the room hanging on it contribute in making the final acquisition "dirty". *ToF-KPN* approach (b) is as always the worst one, incapable of recognizing whether a point is an inlier or an outlier. *SHARP-Net* with and without the Residual Fusion and the Depth Refinement blocks perform very similarly. Most likely (c) achieves a slightly better result in terms of

(a) Input Depth Image at 25 *fps*.



(b) Predicted Depth with *ToF-KPN*.

(c) Predicted Depth with *SHARP-Net* with no Refine Fusion.



(d) Predicted Depth with *SHARP-Net*.

Figure 6.4: **"Results of MPI and Shot Denoising at 25 fps: Satellite Model in TASI Courtesy"**. Depth Map acquired at 25 *fps*(a) and predicted depth maps with the usage of *ToF-KPN*N(b), *SHARP-Net* with no Refine Fusion(c) and *SHARP-Net*(d).

denoising and outliers removal, but it partially loses some of the satellite's points, while (d) is a little less efficient in removing outliers but moderately more precise in keeping the shape of the satellite untouched.

To conclude this paragraph, Fig. 6.5 illustrates the point clouds (through the usage of Open3D [107] tools) after the employment of the implemented frameworks. In particular, (e) to (h) show the point sets obtained by means of *ToF-KPN* for the considered frame rates (5, 10, 15 and 25 *fps* respectively), (i) to (l) illustrate the results achieved with the "reduced" alternative of *SHARP-Net*, deprived of its last two blocks, while (m) to (p) finally depict the outcomes attained through the original version of *SHARP-Net*. As it is possible to notice, the results are consistent with what has been seen until now: *ToF-KPN* always achieves the worst performance (Fig. 6.5(e), (f), (g), (h)), not being able to correctly distinguish between points to be denoised and not to, whereas *SHARP-Net* with and without the final modules accomplishes good results for every considered frame rate.

Quantitative Analysis. In order to quantitatively measure the performance of the proposed approaches, different metrics have been taken into account. Unfortunately, unlike

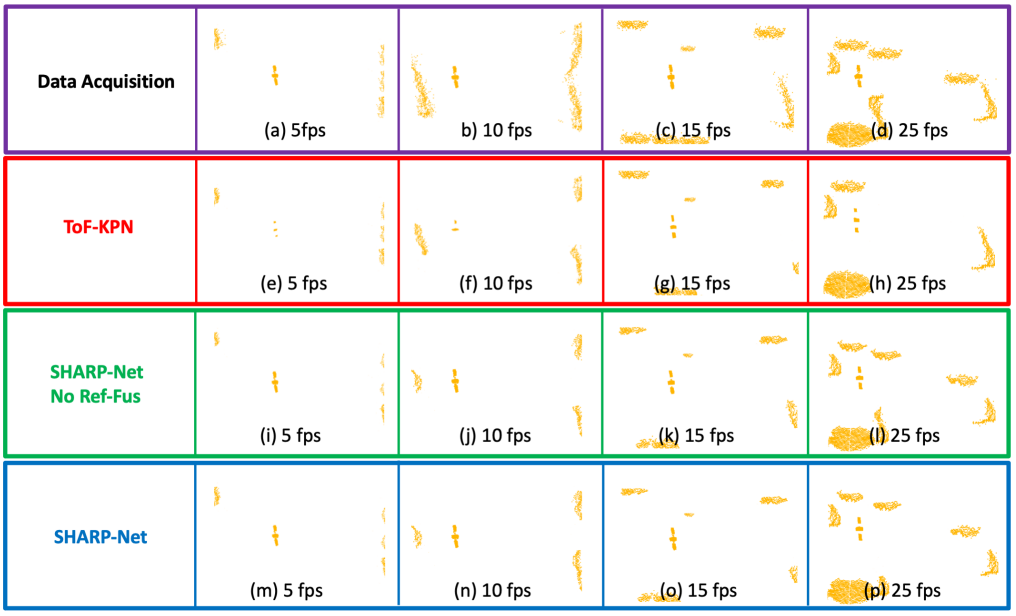


Figure 6.5: **"Point Clouds after Denoising Phase: Satellite Model in TASI Courtesy"**. Point clouds acquired at 5 *fps* (a), 10 *fps* (b), 15 *fps* (c) and 25 *fps* (d). (e) to (p) represent the denoising point sets obtained through the employment of the implemented approaches: specifically, (e) to (h) depict the results of *ToF-KPN* for the considered frame rates, (i) to (l) show the outcomes of *SHARP-Net* without the two final modules, while (m) to (p) lastly illustrate the point clouds derived by means of the original *SHARP-Net*.

all the experiments conducted in the papers reported in Chapter 2 that were able to compare the output of the denoising algorithm with a ground truth value, in this case there isn't any ground truth value to match up the denoised point clouds to, since the only starting point is the input point cloud directly acquired with the PicoFlexx sensor. Thus, the main idea behind the performance evaluation conducted in the following is that of computing the percentage reduction in the number of outliers (which means, the percentage increase in the number of inliers) of the denoised point cloud with respect to the input one. Inliers are all the points of the point cloud belonging to the satellite model, while outliers refer to all the noisy points, either belonging to the background or to every other object but the satellite.
Therefore, first and foremost the relative percentage of inliers has been computed for the depth input point clouds and for the denoised ones. In particular, the RI (Relative Inliers) index is defined as follows:

$$RI = \frac{N_{Inliers}}{N_{Inliers} + N_{Outliers}} \tag{6.1}$$

where $N_{Inliers}$ refers to the number of inliers points belonging to the volume of the detected satellite, while $N_{Outliers}$ refers to the noisy outliers.
Vice versa, obviously, it can be defined the RO (Relative Outliers) index as:

$$RO = \frac{N_{Outliers}}{N_{Inliers} + N_{Outliers}} \tag{6.2}$$

with the clear constraint that $RI + RO = 1$.

Moreover, the following metrics have also been considered in this Section:

$$\begin{aligned} MRI\ (Mean\ RI) &= \frac{1}{N}\sum_{i=1}^{N} RI_i \\ MRO\ (Mean\ RO) &= \frac{1}{N}\sum_{i=1}^{N} RO_i \end{aligned} \tag{6.3}$$

where N is the number of acquired input depth maps.
Starting from these metrics, other indicators have further been implemented. Specifically, the Relative MAE and the Relative RMSE metrics are described as follows:

$$\begin{aligned} Relative\ MAE\ (RMAE) &= \frac{1}{N}\sum_{i=1}^{N} |RO_{i_{denoised}} - RO_{i_{acquired}}| \\ Relative\ RMSE\ (RRMSE) &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} (RO_{i_{denoised}} - RO_{i_{acquired}})^2} \end{aligned} \tag{6.4}$$

where N is again the number of input acquisitions, $RO_{i_{denoised}}$ is the relative number of outliers in each denoised point cloud while $RO_{i_{acquired}}$ is the relative number of outliers in the input acquired ones, with $i \in [1,N]$. Being "denoising" methods, it must always hold that $RO_{i_{denoised}}$ is lower or equal to $RO_{i_{acquired}}$.
Given that these metrics represent the number of outliers of the denoised depth map with respect to the number of outliers in the original input, the higher they are the better it is, since it means that the lower will also be the amount of noisy points present in the predicted scene. Indeed, the perfect situation will be that of having in the esteemed depth map all the points belonging to the satellite models (inliers), thus $RO_{denoised} = 0$ for each input acquisition, which exactly leads to have the maximum difference between the computed *RO*s, therefore the highest value for the metrics.

Tables 6.1 and 6.2 summarize the performance of the proposed approaches. Specifically, Table 6.1 concentrates on the execution times of the algorithms while Table 6.2 focuses on the metrics just described. As reported in Table 6.1, execution times are different for the three proposed approaches, as one could have expected. In particular, *ToF-KPN* is able to achieve the fastest execution, since it is the simplest model out of the three.

Execution Time of Denoising Approaches				
Model	**Frame Rate for Depth Input**			
	5 fps	**10 fps**	**15 fps**	**25 fps**
ToF-KPN	0.375 s	0.380 s	0.374 s	**0.369 s**
SHARP-Net no Refine Fusion	**0.570 s**	0.579 s	0.590 s	0.581 s
SHARP-Net	**0.915 s**	0.918 s	0.922 s	0.919 s

Table 6.1: **"Execution Time of Denoising Approaches".** Execution time calculated for the implementation of the proposed architectures, namely *SHARP-Net*, *SHARP-Net* without Refine and Fusion blocks and *ToF-KPN*. The value is the computed as the average out of 5 different input depth images acquired per each frame rate. The values marked in **bold** denote the best performance according to each implemented method.

On the other side, *SHARP-Net* in its original version is the slowest one, since the presence of the three modules highly influences the online performance of the network. The alternative of *SHARP-Net* without the last two blocks (Residual Fusion and Depth Refinement) stands between the two previous frameworks, presenting good results (better than the ones provided by *ToF-KPN* and comparable with the *SHARP-Net* ones) at low execution time (almost half of the time required by *SHARP-Net*). The outcomes are computed by taking the mean value over 5 different input depth images captured per each frame rate, using a 2.7 GHz Intel Core *i*5 dual-core.

Table 6.2 quantitatively compares the proposed approaches using the indicators previously described. As it is possible to notice, the results are consistent with what is depicted in Figs. 6.1, 6.2, 6.3, 6.4 and 6.5. Indeed, *SHARP-Net*'s outcomes, with and without the last two modules, present a MRI metric generally higher with respect to the input data (thus, a lower MRO). Specifically, at 15 *fps*, *SHARP-Net* achieves an increase of nearly 12% with respect to the input noisy data (i.e., the number of inliers in the predicted depth map is 12% greater than the number of inliers in the input), while its variant without the Residual Fusion and the Depth Refinement blocks is able to get an improvement of almost 14% with the depth image acquired at 25 *fps*.

Performance Evaluation of Denoising Approaches					
Performance Metrics		**Frame Rate for Depth Input**			
		5 fps	**10 fps**	**15 fps**	**25 fps**
Input Data	MRI	0.6951	0.6724	0.5833	0.3409
	MRO	0.3049	0.3276	0.4167	0.6591
ToF-KPN	MRI	0.2123	0.2242	0.3594	0.2031
	MRO	0.7881	0.7761	0.6413	0.7978
	RMAE	0.4830	0.4484	0.2253	0.1378
	RRMSE	0.4832	0.4485	0.2257	0.1381
SHARP-Net No Ref/Fus	MRI	**0.7162**	0.7012	0.5912	**0.3898**
	MRO	**0.2841**	0.2993	0.4097	**0.6115**
	RMAE	**0.0208**	0.0286	0.0066	**0.0479**
	RRMSE	**0.0222**	0.0299	0.0115	**0.0492**
SHARP-Net	MRI	0.7154	**0.7061**	**0.6492**	0.3778
	MRO	0.2855	**0.2945**	**0.3511**	0.6237
	RMAE	0.0199	**0.0337**	**0.0645**	0.0360
	RRMSE	0.0220	**0.0349**	**0.0657**	0.0369

Table 6.2: **"Performance Evaluation of Denoising Approaches".** MRI and MRO calculated for the input depth image and for the denoised ones, along with the two newly introduced metrics (RMAE and RRMSE) computed for the three denoising approaches. Values are obtained taking $N = 5$ input depth images. The values marked in **bold** denote the best performance according to each frame rate acquisition.

6.2 Point Cloud Registration Results

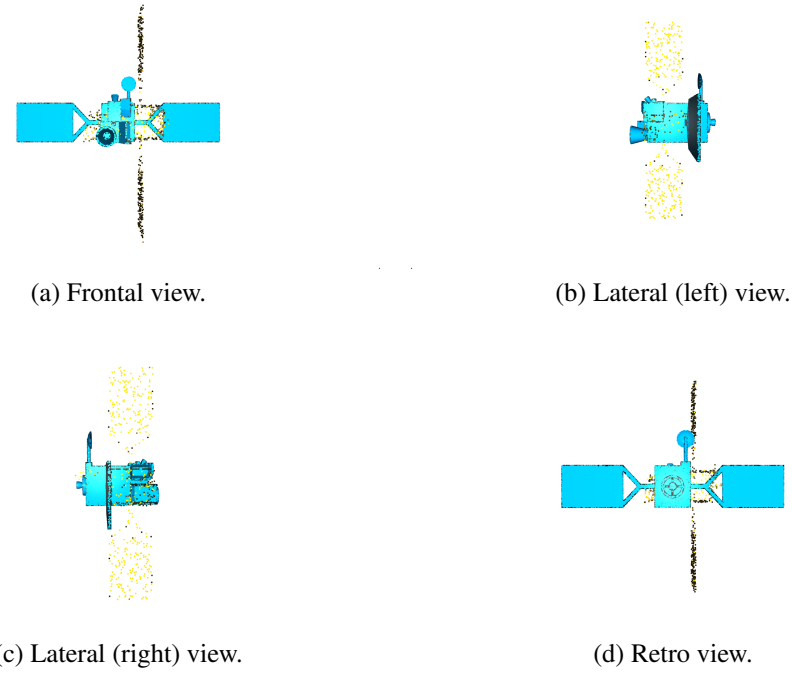
This Section finally evaluates the performance of the point cloud registration frameworks implemented in this thesis work:

1. *FGR*: For the classic optimization-based methods, the fastest among them is taken into account, Fast Global Registration [106]

2. *FRR*: For the feature-learning approaches the simplest yet very effective FPFH + RANSAC (+ Refinement ICP) is chosen [79, 31, 107]

3. *FMR*: Lastly, for the end-to-end learning-based frameworks the Feature-Metric Registration [47] is considered in these experiments.

Specifically, these approaches are compared both qualitatively and quantitatively using 4 different target point cloud's dimensions (20000, 30000, 50000 and 100000 points respectively), while the input is always captured at 5 *fps*, since this use case leads to the best and cleanest acquisitions as previously shown. As one can see by looking at Fig. 6.6, the two initial point clouds (the denoised and cropped source input and the target CAD model) are rotated and translated one another. Specifically, (a) depicts the point clouds from a frontal view, (d) from the posterior view while (b) and (c) from the lateral views, left and right respectively. The aim of the point cloud registration methods is that of finding the transformation that best aligns the two input point clouds.



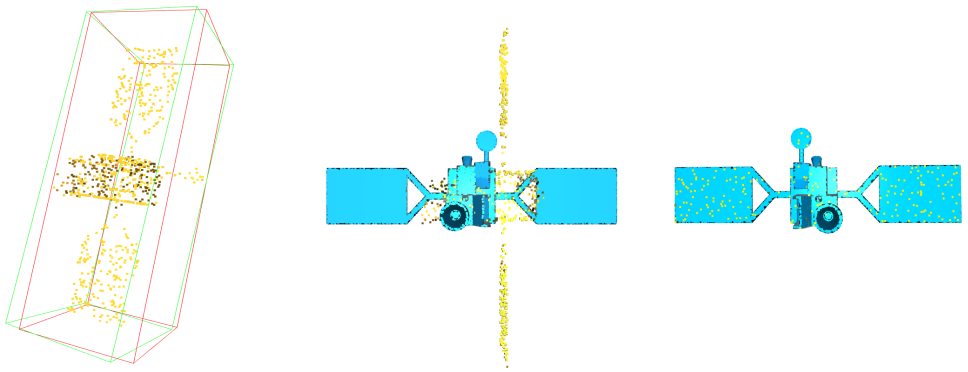
(a) Frontal view.

(b) Lateral (left) view.

(c) Lateral (right) view.

(d) Retro view.

Figure 6.6: **"Initial Alignment of the two point clouds: Satellite Model in TASI Courtesy".** Frontal view of the two point clouds (a), lateral views, left (b) and right (c) respectively, and lastly retro view (d).



(a) Object Detection. (b) Initial Alignment. (c) Final Alignment.

Figure 6.7: **"Point Cloud Registration Workflow: Satellite Model in TASI Courtesy".** The workflow of the implemented end-to-end point cloud registration framework is as follows: (a) the object of the source point cloud is detected and enclosed in bounding boxes, (b) the two input point clouds are initially aligned, and lastly (c) the algorithm finds the rigid transformation that best aligns the two point sets.

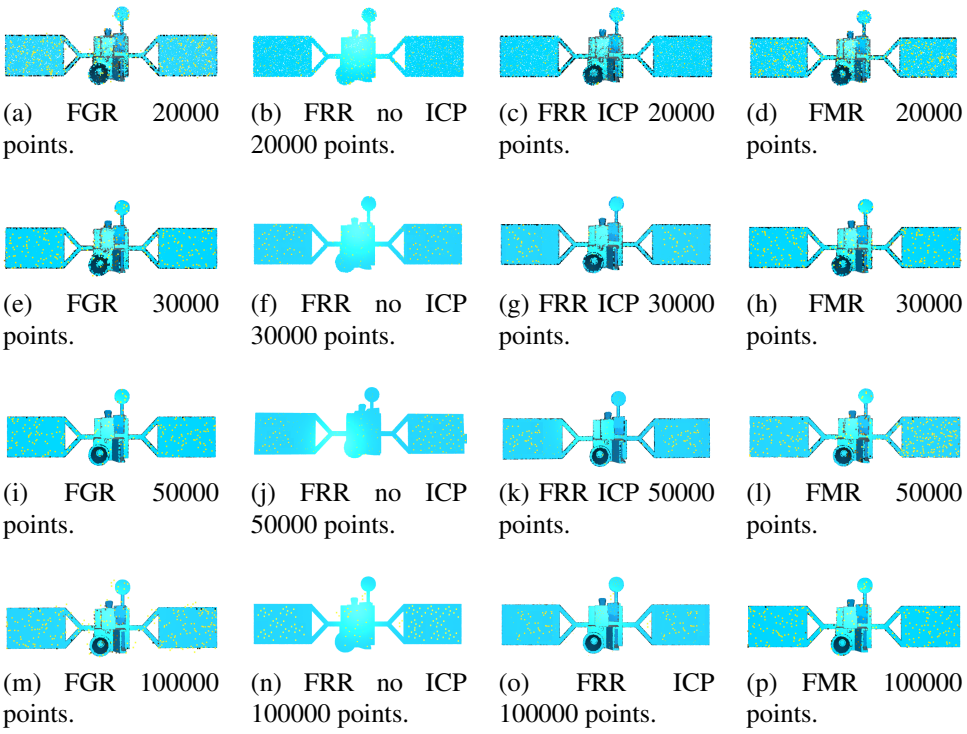
Thus, the workflow that one has to expect from these frameworks is the following, as show in Fig. 6.7:

1. First, the object in the source input point cloud is detected and enclosed in bounding boxes (Fig. 6.7(a)).

2. Second, the two point clouds are overlapped and their initial transformation (i.e., their original alignment) is estimated (Fig. 6.7(b)).

3. Third, the registration algorithm finds the best transformation able to align the two point clouds, allowing to obtain a (nearly) perfect match (Fig. 6.7(c).

Qualitative Analysis. Fig. 6.8 visually compares all the proposed approaches, namely *FGR*, *FRR* (with and without ICP refinement) and *FMR*, with 4 different target point cloud's dimensions (with 20000, 30000, 50000 and 100000 points respectively). As it is possible to see, all the approaches are able to correctly align the two point clouds at the end of their execution, both for low and high-dimensional target point clouds. In particular, *FGR* and *FMR* always get the best results, being able to nearly perfectly align the source to target, even though in *FGR* results some points of the source point clouds remain not precisely overlapped (as can be seen in Figs. 6.8(a), (e), (i) and (m)). Moreover, when increasing the size of the target point cloud, *FGR* makes the convergence harder (presenting some points of the source point set which are not perfectly aligned with the template), while *FMR* seems to be very robust to these changes (Figs. 6.8(d), (h), (l) and (p)). On the other hand, *FRR* without ICP achieves the worst results (Figs. 6.8(b), (f), (j) and (n)), while still being able to correctly align them. With the usage of ICP as refinement algorithm though (Figs. 6.8(c), (g), (k) and (o)), the outcomes of *FRR* are almost comparable with the ones of *FGR* and *FMR*, even if its execution time becomes much larger.



(a) FGR 20000 points. (b) FRR no ICP 20000 points. (c) FRR ICP 20000 points. (d) FMR 20000 points.

(e) FGR 30000 points. (f) FRR no ICP 30000 points. (g) FRR ICP 30000 points. (h) FMR 30000 points.

(i) FGR 50000 points. (j) FRR no ICP 50000 points. (k) FRR ICP 50000 points. (l) FMR 50000 points.

(m) FGR 100000 points. (n) FRR no ICP 100000 points. (o) FRR ICP 100000 points. (p) FMR 100000 points.

Figure 6.8: **"Point Cloud Registration Results: Satellite Model in TASI Courtesy".** Comparison between the final alignment obtained by means of the proposed methods, with target point clouds of 20000, 30000, 50000 and 100000 points respectively.

In order to finally qualitatively evaluate the robustness of the implemented approaches, their performance is compared taking into account different starting poses of the source point cloud. Specifically, Fig. 6.9 shows the behaviour of these methods when both the initial pose of the source point set and the number of points of the target one vary.
Figs. 6.9 from (a) to (p) depict the case reported above, while Figs. 6.9 from (q) to (F) and Figs. 6.9 from (G) to (V) introduce two new initial locations. As one could imagine, the outcomes are consistent with what was previously shown, with generally *FMR* working better and *FRR* (with ICP) achieving the worst results.
In particular, all the proposed frameworks are still able to correctly align the point clouds one another, but with more difficult starting positions and with a larger size of the target point cloud *FGR* and *FRR* slightly suffer in the estimation of the best rigid transformation. On the other hand, *FMR* is always capable of perfectly aligning the two input point sets, regardless of the initial position of the source and the dimensionality of the target.
Specifically, Figs. 6.9 from (u) to (x) and from (K) to (N) demonstrate the lack of precision presented by *FGR* while trying to align the left "wing" of the two input point clouds. Figs. 6.9 from (y) to (B) report the issues encountered by *FRR* (coupled with ICP as final refinement) in aligning the main body of the satellite source and target point sets, whereas Figs. 6.9 from (O) to (R) illustrate the scarcity of accuracy experienced by the same method when trying to align the right "wing" of the satellite models. Figs. 6.9 from (C) to (F) and from (S) to (V) finally proclaim the clear supremacy of the last approach, *FMR*, which always achieves the best performance, no matter the initial starting pose.

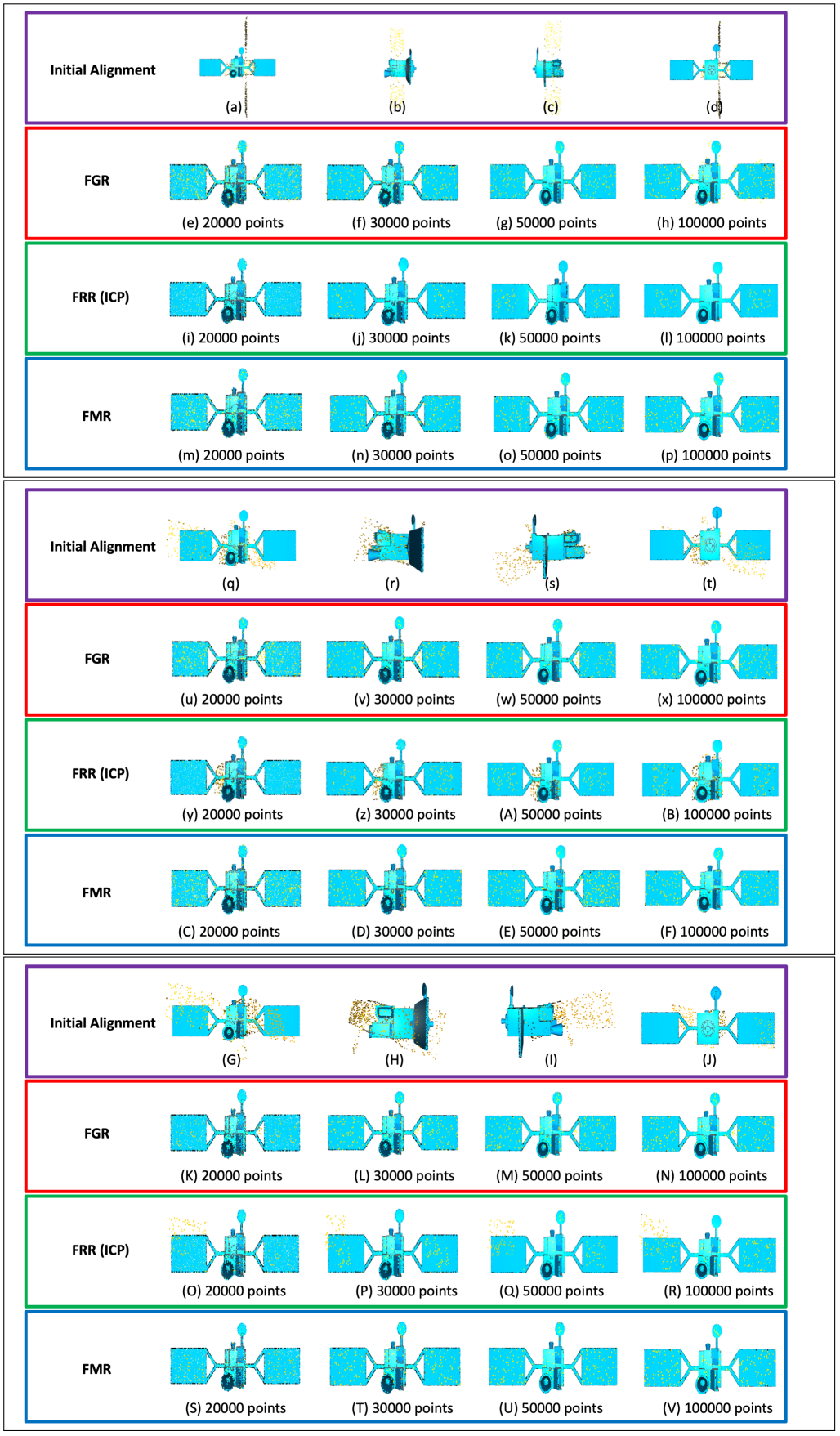


Figure 6.9: **"Point clouds alignment starting from different source poses and target sizes: Satellite Model in TASI Courtesy".**

Quantitative Analysis. As already stated in the previous Section for what concerns ToF denoising approaches, also in this case the ground truth value of the transformation wasn't available, thus the performance of the algorithms couldn't be assessed through its usage, as it is done in most of the literature of the topic nowadays present. Hence, being partially inspired by the indicators used in [107]'s implementation of registration pipelines, from a quantitative point of view the evaluation is performed by calculating two main metrics, namely Hit-Rate and Inlier RMSE.
The first, denoted in the following as HR, measures the fraction of source points being correctly aligned by computing the number of inlier correspondences over the total amount of points in the source point set. In formulas this corresponds to:

$$\textit{Hit-Rate (HR)} = \frac{N_{S_{Inliers}}}{N_S} \tag{6.5}$$

where N_S exactly indicates the cardinality of the source point cloud and $N_{S_{Inliers}}$ the number of its points correctly aligned to the target reference. Of course it holds that the higher is the HR metric the better it is, since it means that the more points are correctly aligned to the template point set.
Inlier RMSE quantifies instead the RMSE of all the retrieved inlier correspondences. Thus, this presupposes that the lower this indicator is the better is the final performance.

Table 6.3 summarizes the results according to these metrics by taking into account *FGR*, *FRR* without ICP, *FRR* with ICP and *FMR*. The results reported here are dependent both on the size of the target point cloud and on the initial alignment between the two: in particular, the classic 4 different sizes are considered for the target point set, and the three starting poses depicted in Fig. 6.9 are chosen for the source point cloud's initial position. As it is possible to observe, *FMR* achieves the best outcomes, being therefore consistent with what is shown in Figs. 6.8 and 6.9. Specifically, for the first case, when the dimensionality of the target point cloud increases, the HR of the *FGR* slightly decreases, obtaining worst outcomes even with respect to *FRR* (with ICP as refinement). *FMR* is instead perfectly capable of accomplishing state-of-the-art performance, both of what concerns HR and Inlier RMSE metrics, regardless of the target point set size.
On the other hand, considering the different starting poses of the source point cloud, it appears evident that *FMR* is always able to achieve the highest performance, both in terms of HR and Inlier RMSE, without loosing in accuracy as the initial position becomes more difficult. This highly proves its capability in understanding rotation-attentive features from the input point clouds. Nonetheless, the same cannot be said for *FGR* and *FRR*: both encounter some troubles when dealing with more strenuous situations. The former is still capable of getting very high results, without losing too much in precision with respect to the first case, while the latter sees its performance significantly decay, especially for what concerns the second alignment. So, while coming (nearly) always close to *FMR*'s results, *FGR* never fully manages to surpass it, neither in the Hit Rate metric nor in the Inlier RMSE one, not to mention *FRR*, with or without ICP.

Lastly, the execution times of all the implemented methods are taken into account to have another important metric in order to establish the performance of these algorithm from a 360° point of view. The only initial position of the source point cloud that has been taken into account for this "time cost" evaluation is the one displayed at Fig. 6.6, since the time needed for these algorithms to complete is independent from the starting pose.

Performance evaluation of pcd registration approaches						
Performance Metrics			**Target Point Cloud Dimension**			
			20000 points	**30000 points**	**50000 points**	**100000 points**
First Alignment	**FGR**	HR	0.9621	0.9630	0.9539	0.9534
		Inlier RMSE	0.0106	0.0106	0.0106	0.0105
	FRR NoICP	HR	0.9214	0.9231	0.9220	0.9295
		Inlier RMSE	0.0153	0.0168	0.0159	0.0151
	FRR ICP	HR	0.9531	0.9534	0.9634	0.9644
		Inlier RMSE	0.0144	0.0155	0.0151	0.0142
	FMR	HR	**0.9705**	**0.9706**	**0.9708**	**0.9708**
		Inlier RMSE	**0.0101**	**0.0100**	**0.0100**	**0.0099**
Second Alignment	**FGR**	HR	0.9244	0.9212	0.9239	0.9221
		Inlier RMSE	0.0203	0.0204	0.0204	0.0204
	FRR NoICP	HR	0.7679	0.7601	0.7690	0.7681
		Inlier RMSE	0.0391	0.0390	0.0394	0.0389
	FRR ICP	HR	0.8111	0.8107	0.8103	0.8109
		Inlier RMSE	0.0281	0.0286	0.0283	0.0284
	FMR	HR	**0.9704**	**0.9705**	**0.9705**	**0.9703**
		Inlier RMSE	**0.0102**	**0.0101**	**0.0102**	**0.0102**
Third Alignment	**FGR**	HR	0.9312	0.9308	0.9310	0.9311
		Inlier RMSE	0.0182	0.0185	0.0184	0.0184
	FRR NoICP	HR	0.8361	0.8314	0.8344	0.8359
		Inlier RMSE	0.0272	0.0281	0.0279	0.0277
	FRR ICP	HR	0.8801	0.8814	0.8810	0.8799
		Inlier RMSE	0.0244	0.0251	0.0239	0.0257
	FMR	HR	**0.9721**	**0.9722**	**0.9722**	**0.9722**
		Inlier RMSE	**0.0098**	**0.0098**	**0.0099**	**0.0097**

Table 6.3: **"Performance Evaluation of Point Cloud Registration Approaches".** Hit-Rate and Inlier RMSE metrics for evaluating the performance of the proposed approaches calculated as the average value of 5 different runs, considering target point cloud with growing sizes (20000, 30000, 50000 and 100000 points respectively) and different starting poses for the source one (as depicted in Fig. 6.9). The values marked in **bold** denote the best performance according to each target point cloud dimension and initial alignment.

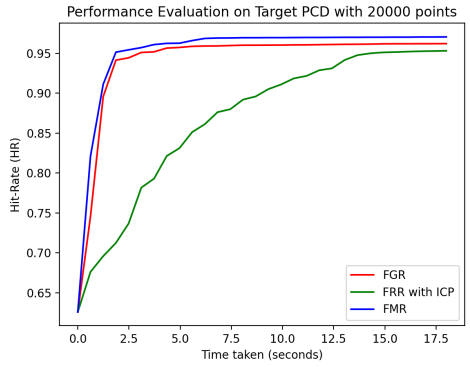
Table 6.4 outlines the results of the performed experiments, where again several target point clouds with different dimensions are considered. It is possible to notice that *FMR* is not only the method that allows to achieve the best performance, but it is also the fastest, thus being perfectly consistent with the initial requirements of this thesis work. *FGR* also achieves good results with a very low time cost (proving to be even faster than *FMR* sometimes), while *FRR* with or without the usage of the refinement ICP results to be incredibly slower with respect to the others.

Fig. 6.10 and Fig. 6.11 finally depict the trend of the proposed metrics, HR and Inlier RMSE respectively, for each of the implemented methods, considering the four different sizes of the target point cloud taken into account so far. Once more, also in this case as previously explained for the results collected in Table 6.3, the metrics are computed by only taking into account the position of the source point cloud illustrated in Fig. 6.6, since this is the situation in which the results of the three methods are closest one another,

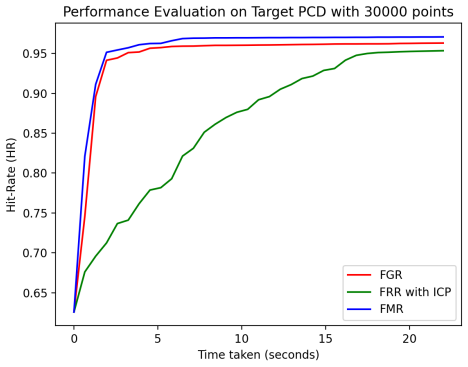
Execution Time of Point Cloud Registration Approaches				
Model	**Target Point Cloud Dimension**			
	20000 points	**30000 points**	**50000 points**	**100000 points**
FGR	0.431 s	0.748 s	**1.997 s**	5.441 s
FRR	16.840 s	21.121 s	24.968 s	29.914 s
FRR (with Ref. ICP)	17.012 s	21.511 s	25.098 s	30.115 s
FMR	**0.392 s**	**0.694 s**	2.130 s	**5.012 s**

Table 6.4: **"Execution Time of Point Cloud Registration Approaches".** Execution time of the proposed approaches, calculated as a mean of 5 different runs, considering target point cloud with growing dimensions, 20000, 30000, 50000 and 100000 points respectively. The values marked in **bold** denote the best performance according to each target point cloud size.

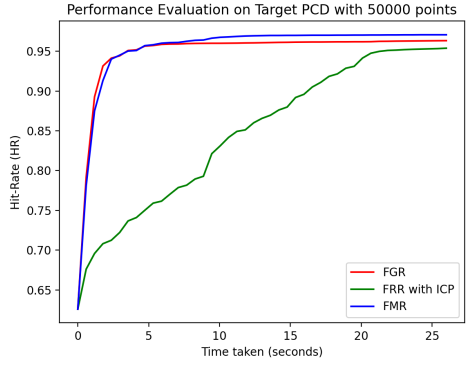
regardless of the target point cloud's dimensionality.



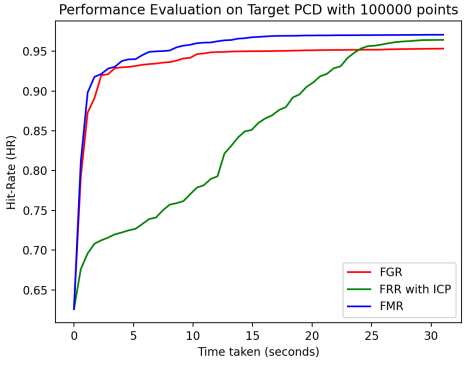
(a) Target PCD with 20000 points



(b) Target PCD with 30000 points.



(c) Target PCD with 50000 points.

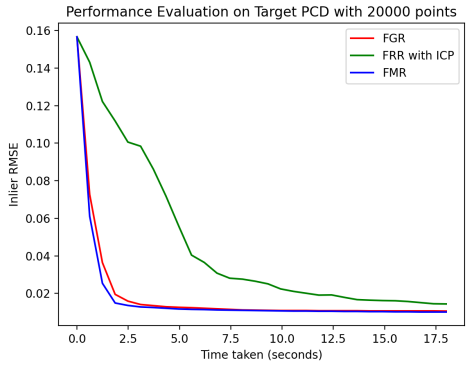


(d) Target PCD with 100000 points.

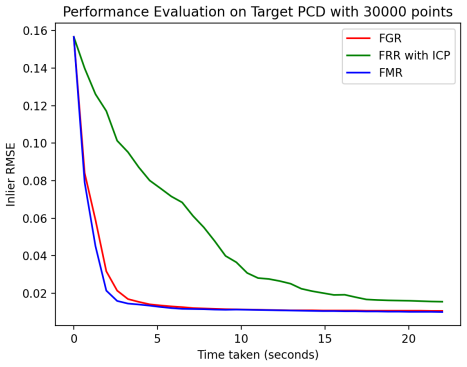
Figure 6.10: **"Hit Rate metric of the Point Cloud Registration Approaches".** Hit rate trend of the proposed approaches in function of time spent to converge (in seconds) and number of points in the target point set. Specifically, (a) considers 20000 points, (b) 30000, (c) 50000 and finally (d) 100000. Results are consistent with what shown in Tables 6.3 and 6.4.

Therefore, it has been decided not to show their graphs in the following, but to have a quantitative comparison one can simply have a look at Table 6.3 thoroughly described before. In particular, Fig. 6.10 clearly shows that *FMR* method is always capable of achieving the best results in the lowest amount of time, regardless of the size of the target point set (or the initial starting pose of the source one, for what matters), thus proving the outcomes highlighted in Tables 6.3 and 6.4. *FGR*, while being slightly faster than *FMR* in getting a good HR in the case of a target point set with 50000 points (Fig.6.3(c)), gets worst but yet very satisfactory performance, always better than the one obtained through the employment of *FRR*, except for the very last example (with a 100000-points target point cloud). As already shown in Table 6.4 *FRR* proves to be extremely slow in getting satisfactory results, which in any case are nearly always worst than the ones of the other methods.

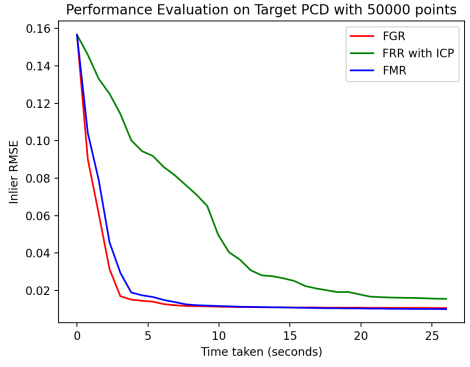
Fig. 6.11 illustrates instead the trend of the Inlier RMSE metric as function of the time needed by the algorithms to converge. Being defined as the RMSE of all the retrieved inlier correspondences, the lower is its value the better is the performance of the method.



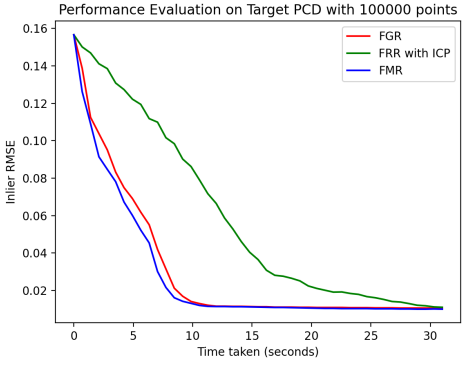
(a) Target PCD with 20000 points



(b) Target PCD with 30000 points.



(c) Target PCD with 50000 points.



(d) Target PCD with 100000 points.

Figure 6.11: **"Inlier RMSE metric of the Point Cloud Registration Approaches".** Inlier RMSE trend of the proposed approaches in function of time spent to converge (in seconds) and number of points in the target point set. Specifically, (a) considers 20000 points, (b) 30000, (c) 50000 and finally (d) 100000. Results are consistent with what shown in Tables 6.3 and 6.4.

Again, *FMR* proves to be the most robust framework, presenting the best outcomes in every conducted experiment, as also outlined in Table 6.3.
FGR also presents good results in an execution time that is nearly comparable to the one exhibited by *FMR*. On the other side, *FRR* (with and without the ICP algorithm as final refinement) turns out to be much slower with respect to the first two methods, achieving good results in terms of Inlier RMSE (sometimes even very close to the ones obtained by means of *FMR*) only in a too high number of seconds. This unfortunately is definitely not in line with the real-time requirements which are at the basis of this thesis work, as previously stated several times.

It is possible to conclude that the implemented end-to-end learning-based method, *FMR*, consistently outperforms the classic optimization-based and the feature-learning ones, both for what concerns qualitative and quantitative results according to the considered metrics (HR, Inlier RMSE and time cost).
As authors of [47] suggest, this could be explained by considering the strength of the unsupervised part of the framework that offers a feature extraction network which is truly capable of embedding peculiar features in order to intrinsically understand the point cloud geometry. Thus, the *FMR* approach provides an effective and very efficient way to tackle the 3D point cloud registration problem, with an overall limited complexity and most importantly capable of working in (near-)real-time, especially when the point clouds have a limited amount of points (like the ones produced by the PicoFlexx camera).
On the other hand, both *FGR* and *FRR* (with and without ICP) suffer from some issues either related to a lack of accuracy when the starting pose isn't straightforward or the size of one point cloud is particularly large (the former), or attributed to the huge amount of time required by the algorithm to converge to a good enough solution (the latter).
The goal of this thesis work, which was that of presenting a promising NN-based approach for 6 *DoF* pose estimation and 3D point cloud registration, has therefore been highly demonstrated.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this thesis work the problem of Object Detection and 6 *DoF* pose estimation with input data acquired with a ToF camera has been addressed through the usage of a NN-based approach.
One of the main contributions of this research is that of having provided a complete and exhaustive survey of (almost) all the state-of-the-art methods both for what concerns the denoising of ToF raw data (for systematic and non-systematic errors) and the 3D point cloud registration issue. Specifically, ten methods for the former task (the ToF denoising one, Chapter 2) and fourteen for the latter (the 6 *DoF* pose estimation, Chapter 3) have been introduced and thoroughly explained, presenting pros and cons for each of these frameworks, also in relation with the specifics and the requirements of this work. Indeed, having in mind a space application for the tasks at hand, the complexity and the (near-)real-time performance of the proposed approaches have always played a key role in every choice performed throughout the whole research.
Another important contribution is that of having worked with a ToF sensor relatively new, the PMD Camboard PicoFlexx camera introduced in Chapter 4, fully exploiting its potential in a large series of different circumstances and calibration procedures and being, to the best of my knowledge, the very first to ever employ this device to fulfil a point cloud registration task.
Most importantly this thesis has aimed at building an end-to-end framework which takes as input the raw noisy data captured from the PicoFlexx camera (in the form of amplitude images, depth maps and point clouds), denoises them through the usage of *SHARP-Net* (along with its variants) implemented and meticulously described in Chapter 5, and finally returns as output, by applying the learning-based *FMR* (or one of the alternatives introduced in Chapter 5, namely the optimization-based *FGR* or the feature-learning *FRR*) the rigid esteemed transformation that is able to best align the source denoised point cloud to a target reference.
Extensive experiments conducted by taking into account different frame rates for the acquisition, different starting poses of the source point cloud and different dimensionalities of the target one have exhaustively proved that the approach presented in this work is perfectly capable of tackling the problem at the very basis of this thesis, by providing a promising solution that meets all the requirements of paramount importance for a robotic

application (but also valuable in avionics) defined at the beginning. Thus, the original aim of the project of designing an approach able to demonstrate its capability to detect a part of a satellite (such as a gripping interface) and estimate its 6 *DoF* pose to support in-orbit servicing missions has been achieved, paving the way for important new research in this area.

7.2 Future Works

Besides the great advantages achieved in this thesis work, there is still a lot of room for improvement. First and foremost, due to the lack of time some experiments haven't been performed, but could potentially have been useful to better assess the performance of the implemented methods. Specifically, on one hand, covering the satellite model with some reflective material could have helped in evaluating the robustness of denoising frameworks.
On the other hand, considering other different starting poses for the source point cloud or knowing ground truth information about translation and rotation could outstandingly have led to a more comprehensive analysis of the proposed point cloud registration solutions.
Moreover, a dataset specifically designed for the PicoFlexx camera could be taken into account to enlarge the performance obtained by the denoising approaches. Indeed, while *SHARP-Net* and its variants achieve good results, they are trained on a dataset containing scenes captured by a device which is very different from the PMD Camboard PicoFlexx sensor this research is done with. These differences regard both the resolution of the acquired data and also the intrinsic characteristics of the camera itself, which inevitably affect the acquisition phase. It goes without saying that a dataset thought "ad hoc" for this sensor could greatly increase the final results, since the methods shouldn't have to be somehow adapted.
Furthermore, due to the inherent structure of ToF PMD sensor leveraging on sinusoidal signals, one could think about using SIREN activation function, described in [83], that is introduced to exploit periodic activation functions to represent elaborated natural signals and, most importantly, their derivatives. This could in some way better delineate the behaviour of the input impulse and perhaps help in better understanding its characteristics.
Finally, to achieve a relevant speed-up on representative avionics, it may be possible to optimize the written code using either C or C++ which are significantly faster than Python for this kind of applications. In this way, while maintaining the same complexity for the presented approaches, one could remarkably gain in real-time performance which, as it has been said several times, is one of the major constraints in robotic (and space above all) applications.

Bibliography

[1] MARTÍN ABADI, ASHISH AGARWAL, PAUL BARHAM, EUGENE BREVDO, ZHIFENG CHEN, CRAIG CITRO, GREG S. CORRADO, ANDY DAVIS, JEFFREY DEAN, MATTHIEU DEVIN, SANJAY GHEMAWAT, IAN GOODFELLOW, ANDREW HARP, GEOFFREY IRVING, MICHAEL ISARD, YANGQING JIA, RAFAL JOZEFOWICZ, LUKASZ KAISER, MANJUNATH KUDLUR, JOSH LEVENBERG, DANDELION MANÉ, RAJAT MONGA, SHERRY MOORE, DEREK MURRAY, CHRIS OLAH, MIKE SCHUSTER, JONATHON SHLENS, BENOIT STEINER, ILYA SUTSKEVER, KUNAL TALWAR, PAUL TUCKER, VINCENT VANHOUCKE, VIJAY VASUDEVAN, FERNANDA VIÉGAS, ORIOL VINYALS, PETE WARDEN, MARTIN WATTENBERG, MARTIN WICKE, YUAN YU, AND XIAOQIANG ZHENG, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] GIANLUCA AGRESTI, HENRIK SCHAEFER, PIERGIORGIO SARTOR, AND PIETRO ZANUTTIGH, *Unsupervised domain adaptation for tof data denoising with adversarial learning*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 5579–5586.

[3] GIANLUCA AGRESTI AND PIETRO ZANUTTIGH, *Deep learning for multi-path error removal in tof sensors*, in Computer Vision – ECCV 2018 Workshops, Laura Leal-Taixé and Stefan Roth, eds., Cham, 2019, Springer International Publishing, pp. 410–426.

[4] ALEXANDER ALSPACH, KUNIMATSU HASHIMOTO, NAVEEN SURESH KUPPUSWAMY, AND RUSS TEDRAKE, *Soft-bubble: A highly compliant dense geometry tactile sensor for robot manipulation*, 2019 2nd IEEE International Conference on Soft Robotics (RoboSoft), (2019), pp. 597–604.

[5] YASUHIRO AOKI, HUNTER GOFORTH, RANGAPRASAD ARUN SRIVATSAN, AND SIMON LUCEY, *Pointnetlk: Robust & efficient point cloud registration using pointnet*, 2019.

[6] PABLO ARBELÁEZ, MICHAEL MAIRE, CHARLESS FOWLKES, AND JITENDRA MALIK, *Contour detection and hierarchical image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 33 (2011), pp. 898–916.

[7] K. S. ARUN, T. HUANG, AND STEVEN D. BLOSTEIN, *Least-squares fitting of two 3-d point sets*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9 (1987), pp. 698–700.

[8] ASHESKNIGHT, *Tof mpi remove*. https://github.com/ashesknight/tof-mpi-remove, 2021. [Online; accessed 01-November-2021].

[9] MATVEEV V B AND SALLE M A, *Darboux Transformations and Solitons*, Springer-Verlag, 1991.

[10] SIMON BAKER AND IAIN MATTHEWS, *Lucas-kanade 20 years on: A unifying framework*, 2004.

[11] STEVE BAKO, THIJS VOGELS, BRIAN MCWILLIAMS, MARK MEYER, JAN NOVÁK, ALEX HARVILL, PRADEEP SEN, TONY DEROSE, AND FABRICE ROUSSELLE, *Kernel-predicting convolutional networks for denoising monte carlo renderings*, ACM Trans. Graph., 36 (2017).

[12] P. BESL AND N. MCKAY, *A method for registration of 3-d shapes*, IEEE Trans. Pattern Anal. Mach. Intell., 14 (1992), pp. 239–256.

[13] TOLGA BIRDAL AND SLOBODAN ILIC, *Point pair features based object detection and pose estimation revisited*, 2015 International Conference on 3D Vision, (2015), pp. 527–535.

[14] EKABA BISONG, *Google Colaboratory*, Apress, Berkeley, CA, 2019, pp. 59–64.

[15] M. BLACK AND A. RANGARAJAN, *On the unification of line processes, outlier rejection, and robust statistics with applications in early vision*, International Journal of Computer Vision, 19 (1996), pp. 57–92.

[16] G. BRADSKI, *The OpenCV Library*, Dr. Dobb's Journal of Software Tools, (2000).

[17] ANDERS GLENT BUCH, LILITA KIFORENKO, AND DIRK KRAFT, *Rotational subgroup voting and pose clustering for robust 3d object recognition*, 2017 IEEE International Conference on Computer Vision (ICCV), (2017).

[18] ENRICO BURATTO, ADRIANO SIMONETTO, GIANLUCA AGRESTI, HENRIK SCHÄFER, AND P. ZANUTTIGH, *Deep learning for transient image reconstruction from tof data*, Sensors (Basel, Switzerland), 21 (2021).

[19] JOHN CANNY, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8 (1986), pp. 679–698.

[20] XIAOTIAN CHEN, X. CHEN, AND ZHENGJUN ZHA, *Structure-aware residual pyramid network for monocular depth estimation*, in IJCAI, 2019.

[21] YAN CHEN, JIMMY REN, XUANYE CHENG, KEYUAN QIAN, LUYANG WANG, AND JINWEI GU, *Very power efficient neural time-of-flight*, in 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), 2020, pp. 2246–2255.

[22] FRANÇOIS CHOLLET ET AL., *Keras*. https://github.com/fchollet/keras, 2015.

[23] CHRISTOPHER CHOY, WEI DONG, AND VLADLEN KOLTUN, *Deep global registration*, 2020.

[24] CHRISTOPHER BONGSOO CHOY, JAESIK PARK, AND VLADLEN KOLTUN, *Fully convolutional geometric features*, 2019 IEEE/CVF International Conference on Computer Vision (ICCV), (2019), pp. 8957–8965.

[25] HAOWEN DENG, TOLGA BIRDAL, AND SLOBODAN ILIC, *Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors*, in Computer Vision – ECCV 2018, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, eds., Cham, 2018, Springer International Publishing, pp. 620–638.

[26] HAOWEN DENG, TOLGA BIRDAL, AND SLOBODAN ILIC, *Ppfnet: Global context aware local features for robust 3d point matching*, 2018.

[27] G. DONG, Y. ZHANG, AND ZHIWEI XIONG, *Spatial hierarchy aware residual pyramid network for time-of-flight depth denoising*, in ECCV, 2020.

[28] A. A. DORRINGTON, J. P. GODBAZ, M. J. CREE, A. D. PAYNE, AND L. V. STREETER, *Separating true range measurements from multi-path and scattering*

interference in commercial range cameras, in Three-Dimensional Imaging, Interaction, and Measurement, J. Angelo Beraldin, Geraldine S. Cheok, Michael B. McCarthy, Ulrich Neuschaefer-Rube, Ian E. McDowall, Margaret Dolinsky, and Atilla M. Baskurt, eds., vol. 7864, International Society for Optics and Photonics, SPIE, 2011, pp. 37 – 46.

[29] ESA, *Introducing mtg*. https://www.esa.int/Applications/Observing_the_Earth/Meteorological_missions/meteosat_third_generation/Introducing_MTG, 2021. [Online; accessed 01-November-2021].

[30] PHILIPP FISCHER, ALEXEY DOSOVITSKIY, EDDY ILG, PHILIP HÄUSSER, CANER HAZIRBAŞ, VLADIMIR GOLKOV, PATRICK VAN DER SMAGT, DANIEL CREMERS, AND THOMAS BROX, *Flownet: Learning optical flow with convolutional networks*, 2015.

[31] M. FISCHLER AND R. BOLLES, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Commun. ACM, 24 (1981), pp. 381–395.

[32] A. FITZGIBBON, *Robust registration of 2d and 3d point sets*, in BMVC, 2001.

[33] STEFAN FUCHS AND GERD HIRZINGER, *Extrinsic and depth calibration of tof-cameras*, in 2008 IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1–6.

[34] XAVIER GLOROT AND YOSHUA BENGIO, *Understanding the difficulty of training deep feedforward neural networks*, in AISTATS, 2010.

[35] GOOGLE, *Coral edge tpu accelerator*. https://coral.ai/products/accelerator/, 2021. [Online; accessed 30-October-2021].

[36] GOOGLE AI BLOG, *Introducing the next generation of on-device vision models: Mobilenetv3 and mobilenetedgetpu*. https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html, 2021. [Online; accessed 30-October-2021].

[37] JOHN C. GOWER, *Generalized procrustes analysis*, Psychometrika, 40 (1975), pp. 33–51.

[38] THIBAULT GROUEIX, MATTHEW FISHER, VLADIMIR G. KIM, BRYAN C. RUSSELL, AND MATHIEU AUBRY, *Atlasnet: A papier-mâché approach to learning 3d surface generation*, 2018.

[39] Q. GUO, I. FROSIO, ORAZIO GALLO, TODD E. ZICKLER, AND J. KAUTZ, *Tackling 3d tof artifacts through learning and the flat dataset*, in ECCV, 2018.

[40] KAPIL GUPTA AND YANWEN XU, *Denoising 3d time-of-flight data*, in ECCV, 2019.

[41] SUYOG GUPTA AND BERKIN AKIN, *Accelerator-aware neural network design using automl*, 2020.

[42] FREDERIK HAGELSKJÆR AND ANDERS GLENT BUCH, *Pointvotenet: Accurate object detection and 6 dof pose estimation in point clouds*, 2020.

[43] MILES HANSARD, SEUNGKYU LEE, OUK CHOI, AND RADU HORAUD, *Time of Flight Cameras: Principles, Methods, and Applications*, SpringerBriefs in Computer Science, Springer, Oct. 2012.

[44] AMMAR HATTAB AND GABRIEL TAUBIN, *3d rigid registration of cad point-clouds*, in 2018 International Conference on Computing Sciences and Engineering (ICCSE), 2018, pp. 1–6.

[45] KAIMING HE, X. ZHANG, SHAOQING REN, AND JIAN SUN, *Deep residual learning for image recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2016), pp. 770–778.

[46] ZAIXING HE, WUXI FENG, XINYUE ZHAO, AND YONGFENG LV, *6d pose estimation of objects: Recent technologies and challenges*, Applied Sciences, 11 (2021).

[47] XIAOSHUI HUANG, GUOFENG MEI, AND JIAN ZHANG, *Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences*, in The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[48] XIAOSHUI HUANG, GUOFENG MEI, JIAN ZHANG, AND RANA ABBAS, *A comprehensive survey on point cloud registration*, 2021.

[49] CHING-HSIANG HUNG AND THOMAS L. MARKHAM, *The moore-penrose inverse of a sum of matrices*, Journal of the Australian Mathematical Society, 24 (1977), p. 385–392.

[50] JASON BROWNLEE, *A gentle introduction to pix2pix generative adversarial network*. https://machinelearningmastery.com/a-gentle-introduction-to-pix2pix-generative-adversarial-network/, 2019. [Online; accessed 02-December-2021].

[51] AHMED EL KHAZARI, YUE QUE, THAI LEANG SUNG, AND HYO JONG LEE, *Deep global features for point cloud alignment*, Sensors, 20 (2020).

[52] DIEDERIK P. KINGMA AND JIMMY BA, *Adam: A method for stochastic optimization*, 2017.

[53] ANDREAS KOLB, ERHARDT BARTH, REINHARD KOCH, AND RASMUS LARSEN, *Time-of-Flight Sensors in Computer Graphics*, in Eurographics 2009 - State of the Art Reports, M. Pauly and G. Greiner, eds., The Eurographics Association, 2009.

[54] KONRADPCOP, *New metrics for industrial depth sensors evaluation for precise robotic applications*.

[55] LAURENCE MORONEY (GOOGLE) AND SIDDARTH SHARMA (NVIDIA), *Announcing tensorrt integration with tensorflow 1.7*. https://developers.googleblog.com/2018/03/tensorrt-integration-with-tensorflow.html, 2018. [Online; accessed 27-November-2021].

[56] E. L. LAWLER AND D. E. WOOD, *Branch-and-bound methods: A survey*, Operations Research, 14 (1966), pp. 699–719.

[57] JIAHAO LI, CHANGHAO ZHANG, ZIYAO XU, HANGNING ZHOU, AND CHI ZHANG, *Iterative distance-aware similarity matrix convolution with mutual-supervised point elimination for efficient point cloud registration*, 2020.

[58] MARVIN LINDNER AND ANDREAS KOLB, *Lateral and depth calibration of pmd-distance sensors*, in Proceedings of the Second International Conference on Advances in Visual Computing - Volume Part II, ISVC'06, Berlin, Heidelberg, 2006, Springer-Verlag, p. 524–533.

[59] MING-YU LIU, ONCEL TUZEL, AND YUICHI TAGUCHI, *Joint geodesic upsampling of depth images*, in 2013 IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 169–176.

[60] YONGCHENG LIU, BIN FAN, SHIMING XIANG, AND CHUNHONG PAN, *Relation-shape convolutional neural network for point cloud analysis*, 2019.

[61] BRUCE D. LUCAS AND TAKEO KANADE, *An iterative image registration technique with an application to stereo vision*, in In IJCAI81, 1981, pp. 674–679.

[62] FABIO MANGANARO, *Riconoscimento di gesti per l'interazione uomo-veicolo mediante sensori infrarossi e 3d*, 2017-18.

[63] JULIO MARCO, QUERCUS HERNANDEZ, A. MUÑOZ, YUE DONG, A. JARABO, M. KIM, XIN TONG, AND D. GUTIERREZ, *Deeptof: off-the-shelf real-time correction of multipath interference in time-of-flight imaging*, ArXiv, abs/1805.09305 (2017).

[64] STEFAN MAY, DAVID DROESCHEL, STEFAN FUCHS, DIRK HOLZ, AND ANDREAS NÜCHTER, *Robust 3d-mapping with time-of-flight cameras*, in 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 1673–1678.

[65] STEFAN MAY, BJORN WERNER, HARTMUT SURMANN, AND KAI PERVOLZ, *3d time-of-flight cameras for mobile robotics*, in 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 790–795.

[66] MICROSOFT, *Rgb-d dataset 7-scenes*. https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/, 2013. [Online; accessed 30-November-2021].

[67] BEN MILDENHALL, JONATHAN T. BARRON, JIAWEN CHEN, DILLON SHARLET, REN NG, AND ROBERT CARROLL, *Burst denoising with kernel prediction networks*, 2018.

[68] KAICHUN MO, PAUL GUERRERO, LI YI, HAO SU, PETER WONKA, NILOY MITRA, AND LEONIDAS J. GUIBAS, *Structurenet: Hierarchical graph networks for 3d shape generation*, 2019.

[69] NIELSEN NORMAN GROUP, *Response times: The 3 important limits*. https://www.nngroup.com/articles/response-times-3-important-limits/, 1993. [Online; accessed 30-October-2021].

[70] G. DIAS PAIS, SRIKUMAR RAMALINGAM, VENU MADHAV GOVINDU, JACINTO C. NASCIMENTO, RAMA CHELLAPPA, AND PEDRO MIRALDO, *3dregnet: A deep neural network for 3d point registration*, 2020.

[71] SIMONE PASINETTI, M. MUNEEB HASSAN, JÖRG EBERHARDT, MATTEO LANCINI, FRANCO DOCCHIO, AND GIOVANNA SANSONI, *Performance analysis of the pmd camboard picoflexx time-of-flight camera for markerless motion capture applications*, IEEE Transactions on Instrumentation and Measurement, 68 (2019), pp. 4456–4471.

[72] ADAM PASZKE, SAM GROSS, FRANCISCO MASSA, ADAM LERER, JAMES BRADBURY, GREGORY CHANAN, TREVOR KILLEEN, ZEMING LIN, NATALIA GIMELSHEIN, LUCA ANTIGA, ALBAN DESMAISON, ANDREAS KOPF, EDWARD YANG, ZACHARY DEVITO, MARTIN RAISON, ALYKHAN TEJANI, SASANK CHILAMKURTHY, BENOIT STEINER, LU FANG, JUNJIE BAI, AND SOUMITH CHINTALA, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[73] PMDTEC.COM, *Pmd picoflexx*. https://pmdtec.com/picofamily/flexx/, 2021. [Online; accessed 6-October-2021].

[74] FRANÇOIS POMERLEAU, FRANCIS COLAS, AND ROLAND SIEGWART, *A Review of Point Cloud Registration Algorithms for Mobile Robotics*, Now Foundations and Trends, 2015.

[75] WILLIAM H. PRESS, SAUL A. TEUKOLSKY, WILLIAM T. VETTERLING, AND BRIAN P. FLANNERY, *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*, Cambridge University Press, USA, 1992.

[76] C. QI, HAO SU, KAICHUN MO, AND LEONIDAS J. GUIBAS, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2017), pp. 77–85.

[77] DI QIU, JIAHAO PANG, WENXIU SUN, AND CHENGXI YANG, *Deep end-to-end alignment and refinement for time-of-flight rgb-d module*, 2019.

[78] OLAF RONNEBERGER, PHILIPP FISCHER, AND THOMAS BROX, *U-net: Convolutional networks for biomedical image segmentation*, 2015.

[79] R. RUSU, NICO BLODOW, AND M. BEETZ, *Fast point feature histograms (fpfh) for 3d registration*, 2009 IEEE International Conference on Robotics and Automation, (2009), pp. 3212–3217.

[80] RADU BOGDAN RUSU AND STEVE B. COUSINS, *3d is here: Point cloud library (pcl)*, 2011 IEEE International Conference on Robotics and Automation, (2011), pp. 1–4.

[81] RADU BOGDAN RUSU, ZOLTÁN-CSABA MÁRTON, NICO BLODOW, AND MICHAEL BEETZ, *Learning informative point classes for the acquisition of object model maps*, 2008 10th International Conference on Control, Automation, Robotics and Vision, (2008), pp. 643–650.

[82] JAMIE SHOTTON, BEN GLOCKER, CHRISTOPHER ZACH, SHAHRAM IZADI, ANTONIO CRIMINISI, AND ANDREW FITZGIBBON, *Scene coordinate regression forests for camera relocalization in rgb-d images*, in Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, June 2013.

[83] VINCENT SITZMANN, JULIEN N.P. MARTEL, ALEXANDER W. BERGMAN, DAVID B. LINDELL, AND GORDON WETZSTEIN, *Implicit neural representations with periodic activation functions*, in Proc. NeurIPS, 2020.

[84] KILHO SON, MING-YU LIU, AND YUICHI TAGUCHI, *Learning to remove multipath distortions in time-of-flight range images for a robotic arm setup*, in 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 3390–3397.

[85] SHUOCHEN SU, FELIX HEIDE, GORDON WETZSTEIN, AND WOLFGANG HEIDRICH, *Deep end-to-end time-of-flight imaging*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 6383–6392.

[86] TIRIAS RESEARCH, *Plaster: A framework for deep learning performance*. https://www.tiriasresearch.com/wp-content/uploads/2018/05/TIRIAS-Research-NVIDIA-PLASTER-Deep-Learning-Framework.pdf, 2018. [Online; accessed 30-October-2021].

[87] GUIDO VAN ROSSUM AND FRED L DRAKE JR, *Python reference manual*, Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[88] ASHISH VASWANI, NOAM SHAZEER, NIKI PARMAR, JAKOB USZKOREIT, LLION JONES, AIDAN N. GOMEZ, LUKASZ KAISER, AND ILLIA POLOSUKHIN, *Attention is all you need*, 2017.

[89] MASSIMO VIOLANTE, *Lecture notes in technologies for autonomous vehicles, 01sqhov*, March 2021.

[90] XUANQUAN WANG, PING SONG, AND WUYANG ZHANG, *An improved calibration method for photonic mixer device solid-state array lidars based on electrical analog delay*, Sensors, 20 (2020).

[91] YUE WANG AND JUSTIN M. SOLOMON, *Deep closest point: Learning representations for point cloud registration*, CoRR, abs/1905.03304 (2019).

[92] YUE WANG, YONGBIN SUN, ZIWEI LIU, SANJAY E. SARMA, MICHAEL M. BRONSTEIN, AND JUSTIN M. SOLOMON, *Dynamic graph cnn for learning on point clouds*, 2019.

[93] WHOIS IDCPRIVACY SERVICE, *Ransac - advantages and disadvantages*. https://www.liquisearch.com/ransac, 2021. [Online; accessed 4-October-2021].

[94] WIKIPEDIA, *B-spline*. https://en.wikipedia.org/wiki/B-spline, 2021. [Online; accessed 17-November-2021].

[95] ——, *Tikhonov regularization*. https://en.wikipedia.org/wiki/Tikhonov_regularization, 2021. [Online; accessed 28-October-2021].

[96] ZONGHAN WU, SHIRUI PAN, FENGWEN CHEN, GUODONG LONG, CHENGQI ZHANG, AND PHILIP S. YU, *A comprehensive survey on graph neural networks*, IEEE Transactions on Neural Networks and Learning Systems, 32 (2021), p. 4–24.

[97] ZHIRONG WU, SHURAN SONG, ADITYA KHOSLA, FISHER YU, LINGUANG ZHANG, XIAOOU TANG, AND JIANXIONG XIAO, *3d shapenets: A deep representation for volumetric shapes*, 2015.

[98] JIANXIONG XIAO, ANDREW OWENS, AND ANTONIO TORRALBA, *Sun3d: A database of big spaces reconstructed using sfm and object labels*, in 2013 IEEE International Conference on Computer Vision, 2013, pp. 1625–1632.

[99] JIAOLONG YANG, HONGDONG LI, DYLAN CAMPBELL, AND YUNDE JIA, *Go-icp: A globally optimal solution to 3d icp point-set registration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 38 (2016), p. 2241–2254.

[100] AMIR YAZDANBAKHSH, KIRAN SESHADRI, BERKIN AKIN, JAMES LAUDON, AND RAVI NARAYANASWAMI, *An evaluation of edge tpu accelerators for convolutional neural networks*, 2021.

[101] CEM YUKSEL, *Sample elimination for generating poisson disk sample sets*, Computer Graphics Forum (Proceedings of EUROGRAPHICS 2015), 34 (2015), pp. 25–32.

[102] ANDY ZENG, SHURAN SONG, MATTHIAS NIESSNER, MATTHEW FISHER, JIANXIONG XIAO, AND THOMAS FUNKHOUSER, *3dmatch: Learning local geometric descriptors from rgb-d reconstructions*, 2017.

[103] YAYU ZHAI, PING SONG, AND XIAOXIAO CHEN, *A fast calibration method for photonic mixer device solid-state array lidars*, Sensors, 19 (2019).

[104] F. ZHANG, T. LEI, J. LI, XINGQUAN CAI, X. SHAO, JIAN CHANG, AND FENG TIAN, *Real-time calibration and registration method for indoor scene with joint depth and color camera*, Int. J. Pattern Recognit. Artif. Intell., 32 (2018),

pp. 1854021:1–1854021:26.

[105] Z. ZHANG, *A flexible new technique for camera calibration*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 1330–1334.

[106] QIAN-YI ZHOU, JAESIK PARK, AND V. KOLTUN, *Fast global registration*, in ECCV, 2016.

[107] QIAN-YI ZHOU, JAESIK PARK, AND VLADLEN KOLTUN, *Open3D: A modern library for 3D data processing*, arXiv:1801.09847, (2018).

[108] JUN-YAN ZHU, TAESUNG PARK, PHILLIP ISOLA, AND ALEXEI A. EFROS, *Unpaired image-to-image translation using cycle-consistent adversarial networks*, 2020.