TUM School of Computation, Information and Technology
Technical University of Munich

Master's Thesis in Robotics, Cognition, Intelligence

Completion, Reconstruction and Refinement of 3D Point Clouds Derived from 2D Sonar Images Using PCTMA-Net

Vervollständigung, Rekonstruktion und Verfeinerung von 3D-Punktwolken aus 2D-Sonarbildern unter Verwendung von PCTMA-Net

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Nico Reeb, M.Sc.
Florian Walter, Dr. rer. nat.

Author Philipp Georg Knestel, B.Sc

Date June 15, 2024 in Munich

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Munich, June 15, 2024 (Philipp Georg Knestel, B.Sc)

Acknowledgement

Completing this thesis required the support and efforts of many dedicated individuals. I am deeply grateful to Prof. Atsushi Yamashita-sensei and Assoc. Prof. Yusheng Wang-sensei of The University of Tokyo for their invaluable organizational and technical supervision. Additionally, I extend my sincere thanks to Nico Reeb of the Technical University of Munich and Dr. rer. nat. Florian Walter of the University of Technology Nuremberg for their organizational assistance and supervision of this work.
I would like to express my gratitude to Mikihiro Ikura-san for establishing contact with Prof. Atsushi Yamashita-sensei. My heartfelt thanks go to the entire Department of Human and Engineered Environmental Studies, and especially to Hayato Terao-san, my contact person and friend, for the warm welcome and seamless integration into the department and life in Tokyo.
Finally, I am profoundly thankful to my girlfriend and my family, whose unwavering help and support made my stay and the completion of this thesis possible.

Abstract

This thesis introduces a method for completing, refining, and reconstructing point clouds derived from 2D sonar images using the Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion (PCTMA-Net). The primary objective is to evaluate the effectiveness of PCTMA-Net in addressing sonar noise, interpolating missing regions, and enhancing the accuracy of 3D reconstructions. This is particularly crucial for applications that require precise 3D models from limited 2D sonar images, such as underwater navigation and mapping. To thoroughly assess the network's adaptability, various data sets are utilized, generated both experimentally in a real-world environment and synthetically using Blender. These data sets simulate different noise levels and data irregularities typical of underwater imaging scenarios. The study generates single-object and multi-object scenes, with PCTMA-Net initially implemented for noise-free single-object reconstruction. The architecture of the PCTMA network is iteratively modified based on interim results that reveal artefacts, ensuring continuous improvement. Pre-trained weights are combined with the generated data sets to train different configurations of the PCTMA-Net. The evaluation compares the network's performance on single-object, multi-object, and combined datasets with various configurations to identify the most effective approaches for managing noise and improving sonar-based 3D completion and refinement tasks. The single-object and multi-object datasets achieve very different results, partly due to artificial bridging in the multi-object dataset results. Ultimately, this study seeks to contribute to the advancement of precise underwater mapping technologies by investigating PCTMA-Net's capabilities in handling noise and discovering potential enhancements for 3D reconstruction from sonar data.

Zusammenfassung

In dieser Arbeit wird eine Methode zur Vervollständigung, Verfeinerung und Rekonstruktion von Punktwolken unter Verwendung des 'Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion' (PCTMA-Net) vorgestellt, die aus 2D-Sonarbildern abgeleitet wurden. Das Hauptziel besteht darin, die Wirksamkeit von 'PCTMA-Net' bei der Behandlung von Sonarrauschen, der Interpolation fehlender Regionen und der Verbesserung der Genauigkeit von 3D-Rekonstruktionen zu bewerten. Dies ist besonders wichtig für Anwendungen, die präzise 3D-Modelle aus begrenzten 2D-Sonarbildern erfordern, wie z. B. Unterwassernavigation und -kartierung. Um die Anpassungsfähigkeit des Netzwerks gründlich zu bewerten, werden verschiedene Datensätze verwendet, die sowohl experimentell in einer realen Umgebung als auch synthetisch mit Blender erzeugt wurden. Diese Datensätze simulieren verschiedene Rauschpegel und Unregelmäßigkeiten, die für Unterwasseraufnahmen typisch sind. In der Studie werden Einzel- und Multiobjekt-Szenen erzeugt, wobei 'PCTMA-Net' zunächst für die Rekonstruktion von rauschfreien Einzelobjekten implementiert wurde. Die Architektur des 'PCTMA-Net' wird auf der Grundlage von Zwischenergebnissen, die Artefakte aufzeigen, iterativ modifiziert, um eine kontinuierliche Verbesserung zu erzielen. Vortrainierte Gewichte werden mit den generierten Datensätzen kombiniert, um verschiedene Konfigurationen des 'PCTMA-Net' zu trainieren. Die Bewertung vergleicht die Leistung des Netzes mit Einzel-, Multiobjekt- und kombinierten Datensätzen mit verschiedenen Konfigurationen, um die effektivsten Ansätze für den Umgang mit Rauschen und die Verbesserung sonarbasierter 3D-Vervollständigungs- und Verfeinerungsaufgaben zu ermitteln. Die Einzel- und Multiobjekt-Datensätze erzielen sehr unterschiedliche Ergebnisse, was teilweise auf künstliche Brückenbildung in den Ergebnissen des Multi-Objekt-Datensatzes zurückzuführen ist. Letztendlich soll diese Studie einen Beitrag zur Weiterentwicklung von präzisen Unterwasserkartierungstechnologien leisten, indem sie die Fähigkeiten von 'PCTMA-Net' im Umgang mit Rauschen untersucht und mögliche Verbesserungen für die 3D-Rekonstruktion aus Sonardaten aufzeigt.

List of Figures

List of Tables

Notation

The following notation style is used for formulas or pseudocode:

Type	Example	Comment
General items		
Scalars	a	lower case, italic
Functions	$\sin(x)$	regular text
Absolute value	$\mid a \mid$	
Concatenation	$\oplus$	
Vectors and matrices		
Vectors	$\boldsymbol{v}$	lower case, italic, bold
Matrices	$\boldsymbol{M}$	upper case, italic, bold
Variable indices	$\boldsymbol{x}_i$	
Static indices	$\boldsymbol{x}_{\min}$	
Unit vectors	$\mathbf{e}_x$	
Transpose	$\boldsymbol{x}^{\mathrm{T}}$	T in superscript and regular font
Vector product (cross product)	$\times$	
Dot product	$\cdot$	
Sets and sequences		
Sets	$A = \{1, 2, 3\}$	upper case, italic
Sequence	$\mathcal{A} = \langle 1, 2, 3 \rangle$	
Set without	$A \backslash \{e\}$	
Unification	$A \cup \{e\}$	
Pseudocode Terms		
Require	`Require List of object groups`	Indicates a precondition or input needed for the algorithm
Ensure	`Ensure Correct visibility`	Indicates a postcondition or output of the algorithm
Function	`Function DenoisePointCloudH5`	Defines a function
Procedure	`Procedure ProcessDirectories`	Defines a procedure or a set of instructions

Table 1: Notation used in this thesis

Glossary

Shortcut	Description
Acronyms	
ANN	Average Nearest Neighbour
BSDF	Bidirectional Scattering Distribution Function
CAD	Computer-Aided Design
CD	Chamfer Distance
EMD	Earth Mover's Distance
FLS	Forward Looking Sonar
FOV	Field of View
HDF5	Hierarchical Data Format version 5
MLP	Multi-layer Perceptron
MNN	Median Nearest Neighbour
MR	Multiple Reflection
PCA	Principal Components Analysis
SAR	Synthetic Aperture Radar
SSIM	Structural Similarity Index
TOF	Time of Flight
Technical Terms	
BatchNorm	Batch Normalization
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
EAE-Net	Elevation Angle Estimation Network
KD-tree	K-Dimensional tree
LBR	Layer-wise Batch Renormalization
MatMul	Matrix Multiplication
NeuSG	Neural Implicit Surface Reconstruction with 3D Gaussian Splatting Guidance
PCTMA-Net	Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion
PointNet	Point Neural Network
ReLU	Rectified Linear Unit
UNet	U-shaped Network

Table 2: Acronyms, technical terms and their descriptions

Contents

Chapter 1

Introduction

Sonar technology has been instrumental in underwater exploration and navigation for centuries. Leonardo da Vinci suggested using underwater tubes to detect approaching ships as early as 1490 [MAG18]. However, significant advancements came after the Titanic disaster when active sonar systems were developed to detect objects and measure distances. During World War I, echosounders were introduced to detect submarines, leading to discoveries about the ocean's topography, including high mountains and deep trenches [Edw+19]. Recent technological improvements in sonar systems, like multibeam echo sounders and synthetic aperture systems, and advances in computers and unmanned platforms have revolutionized seafloor mapping [Edw+19]. Accurately mapping and perceiving the environment in three dimensions is essential for autonomous navigation and task completion, as emphasised by Maddern's study on 3D perception in autonomous systems [Mad+17] and O'Mahony's research on 3D perception in robotics [OMa+19]. Underwater robotics shares this need, although initial conditions differ significantly. Challenges such as distortion, reduced visibility, acoustic interference, and pressure-related issues uniquely affect underwater environments [Kne+]. These conditions hinder the seamless transfer of current 3D information estimation techniques from image-based methods. Sonar images, represented as intensity maps, colourise images based on the backscattered intensity of objects, as noted in related literature [Goo07][KK16, page 492]. A significant challenge in this domain is the generation of accurate 3D models from 2D imaging sources due to speckle noise and sonar artefacts, a common issue in sonar imaging [Kuc07]. This research focuses on reconstructing, refining and completing noisy point clouds generated from 2D sonar images using the Elevation Angle Estimation Network (EAE-Net) [Wan+23]. Despite the effectiveness of this network, the resulting point clouds need further improvement to be usable for navigation tasks. The noise from the sonar images manifests as noisy points in the 3D point clouds. Additionally, this noise complicates the process of elevation angle estimation itself, leading not only to slight inaccuracies but also to significant artefacts and irregularities in the resulting point clouds. To provide reliable environmental representations for autonomous underwater systems, this work introduces an approach for the effective reconstruction, refinement, and completion of the generated point clouds using the Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion (PCTMA-Net) for dense point clouds, as proposed by Lin et al. [Lin+21]. Initially designed for noise-free environments, the trained network can complete single-point clouds even with small correlations sufficient to recognise and complete a shape. This innovative use of PCTMA-Net in underwater robotics marks a significant advancement in research, as its capabilities in noisy single and multi-object scenes are tested and improved. This study aims to thoroughly assess the adaptability and effectiveness of PCTMA-Net in processing sonar-derived point clouds. It will focus on its capability to interpolate missing regions, handle varying levels of sonar noise, single and multi-object scenes, and improve the accuracy of 3D reconstructions from sonar

data. This research aims to answer several crucial questions: How effectively can PCTMA-Net interpolate missing regions in point clouds derived from sonar data? How does it handle sonar noise and artefacts? Do single and multi-object scenes pose different challenges? And to what degree can it enhance the overall accuracy of 3D reconstructions? The answers to these questions will not only confirm the adaptability of PCTMA-Net but also establish a new method in the field of noisy point cloud processing. The significance of this study is not only in its potential to advance practical applications but also in its contribution to scientific understanding. From a practical perspective, this research has the potential to significantly enhance the performance of systems that rely on precise 3D reconstructions, such as those used in autonomous navigation. Scientifically, it introduces a new methodology and provides insights into noisy point cloud processing, thereby highlighting the potential for improved 3D reconstructions in challenging environments. To address the above questions, the theoretical part of this paper begins by illustrating the background on sonar noise, denoising strategies, the EAE-Net, and the PCTMA-Net. The methods section details the implemented pipeline, newly generated datasets, the experimental execution of interim results, the choice of a denoising strategy and subsequent pipeline adaptations, particularly focusing on the PCTMA-Net. The evaluation section compares comprehensive training across different datasets and configurations. Finally, the conclusion is drawn, discussing limitations and potential technical extensions of the system. The experimental part of this work was carried out at the Department of Human and Engineered Environmental Studies of The University of Tokyo. Early results of this thesis were published at the spring conference of the Japan Society for Precision Engineering [Kne+].

Chapter 2

Related Works

This chapter aims to contextualize the pipeline used in this research within the broader field. It addresses elevation angle estimation from sonar images, denoising methods, and the reconstruction of point clouds as distinct pipeline components. Examining each aspect separately addresses their unique contributions and the specific challenges.

Forward-looking sonars (FLS) are indispensable for mapping underwater regions with low visibility. For example, FLS systems have been used to survey reefs where visibility is restricted by water quality. The study by Griffin et al. aims to determine the extent, patchiness, and elevation of these reefs, which are crucial for conservation efforts [Gri+20]. At the same time, these underwater regions are also a challenge for FLS systems and lead to considerable noise impacts as speckle noise and sonar artefacts [Kuc07][HLY20][Goo07]. This complicates processes that use 2D sonar images for further workflows. Since autonomous navigation, in particular, makes 3D information about the environment indispensable, the reliable **estimation of elevation**, i.e. 3D information, is a challenge that is often tackled. Zerr et al. proposed a method that segments a series of sonar images into echo, shadow, and background regions to estimate the elevation of convex objects [ZS96]. Without the limitation of convexity, early research in this field introduced methods that aimed to estimate both the motion of the camera and the corresponding feature points simultaneously [Mai+17] [HK16]. Brahim et al. present a method for creating 3D models from 2D sonar images using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to estimate missing elevation data [Bra+11]. This approach extracts and matches feature points from pairs of 2D images to estimate the camera motion. By pairing these sets of feature points to unique $P_k = (x, y, z)$ points in the scene, geometric constraints are introduced that allow the determination of the elevation angles $\phi_{1,k}$ and $\phi_{2,k}$ of point P_k in the reference frames of the two cameras using the CMA-ES [Bra+11]. Estimating the motion between two images entails further uncertainty besides the elevation estimation itself.

Kim et al. present an underwater-vehicle-based multi-directional scanning method using forward scan sonar (FSS) to enhance 3D point cloud reconstruction of underwater objects. The FSS generates a 3D point cloud by scanning the object in one direction. The 3D point cloud is projected into 2D to extract potential object areas and approximate them with polygons. Based on this information, the next scan path is planned, followed by a 3D reconstruction of the object using a triangular-mesh approach [Kim+20]. Wang et al. introduce a learning-based multi-view stereo method that efficiently estimates 3D information by utilizing an elevation plane sweeping method to generate a depth-azimuth-elevation cost volume, resulting in a volumetric representation of the target [Wan+22b]. The primary drawback of these methods is their requirement for multiple viewpoints, which can be unfeasible in real-world applications.

One approach for underwater simultaneous localization and mapping is using a multibeam imaging sonar for 3D terrain mapping. The method, developed by Wang et al., applies op-

tical flow within bearing-range images for feature tracking and models subsea terrain as a Gaussian Process random field on a Chow-Liu tree to handle degenerate cases [WSE19]. This approach effectively addresses the high levels of noise and the absence of elevation angle information in sonar images, facilitating accurate 3D mapping. Westerman et al. propose a novel method for reconstructing 3D surface points using an FLS. They derive a two-dimensional Fermat flow equation applied to the planes defined by each azimuth angle in the sonar image. This equation helps resolve the surface point's 2D and full 3D location through sensor translation, estimating the spatial gradient of range measurements. Unlike traditional methods, this approach does not depend on image intensity or surface reflectivity [WGK20]. Although the methods mentioned work without multi-viewpoint, they impose clear conditions on the environment, such as convex objects to be restructured. In their work, DeBortoli et al. address the challenge of predicting the missing elevation angle from single 2D underwater sonar images without relying on convex object shapes [DLH19b]. They propose using Convolutional Neural Networks (CNNs) to extract meaningful information despite the high noise levels inherent in these images. Their self-supervised method leverages the physics of the sonar sensor to train the network on real data without requiring ground-truth elevation maps. The network, employing a U-Net-like architecture, treats elevation estimation as a classification problem by discretizing the elevation angle into bins. The method is limited by the necessity of having a human expert first hand-label captured images as high or low-quality, relying on only high-quality images for the dataset.
Qadri et al. present a different approach for dense 3D reconstruction of objects using forward-looking sonar (FLS) by representing geometry as a neural implicit function [QKG23]. This method leverages a differentiable volumetric renderer to model acoustic wave propagation and synthesize sonar measurements. The authors demonstrate that their algorithm reconstructs high-fidelity surface geometry from multi-view FLS images with much higher quality and less memory overhead compared to previous techniques. This approach contrasts with existing methods like NeRF [Mil+21], IDR [Yar+20], and NeuS [Wan+21], which focus on 3D reconstruction using optical sensors. The EAE-Net used in this work utilises acoustic videos and the movement data of the sensor [Wan+23].

Denoising point clouds derived from 3D scanners or other sensors, such as sonar, often encounter significant noise, artefacts, and outliers[Che+19][Rak+20]. Effective denoising of these point clouds is, therefore, an ongoing issue. Rakotosaona et al. present a data-driven method for denoising point clouds, bypassing traditional local surface fitting or statistical noise models [Rak+20]. Using a deep learning architecture adapted from PCPNet, the method classifies and discards outliers and then projects noisy points onto clean surfaces. This approach efficiently handles large, densely sampled point clouds, demonstrating robustness to high noise levels. Luo et al. propose a denoising method for point clouds by increasing the log-likelihood of each point through gradient ascent [LH21]. This approach models the noisy point cloud distribution as a convolution of noise-free samples with a noise model. A neural network estimates the score of the log-probability function, enabling gradient ascent updates for denoising. Hermosilla et al. present an unsupervised method for denoising 3D point clouds directly from noisy data. Extending recent unsupervised image denoising techniques, they address the challenge that 3D point clouds lack a structured pixel grid and are subject to total noise in all coordinates [HRR19]. To overcome this, they introduce a spatial prior term that converges to the unique closest point on the manifold of possible clean points. Their method achieves denoising performance comparable to supervised learning, given enough training examples, without requiring pairs of noisy and clean training data. While these denoising strategies excel in their benchmarks, they often significantly alter the point clouds they aim to refine. In this work, we deal with highly noisy point clouds that

require minimal denoising, as the noise may indicate objects' presence. The goal is to retain as much original data as possible, preserving the essential characteristics while reducing only the most disruptive noise.

Point cloud completion is an active field in recent years and has undergone significant improvements in terms of applicability. This is due, on the one hand, to the improved ability to capture, generate and sense 3D models and, on the other hand, to the profound improvement of deep learning models [Lin+21]. PointNet[Qi+17a] and PointNet++[Qi+17b], which are also integrated into this work, achieved significant results. PointNet pioneered the approach by directly processing raw point clouds to learn global features and perform tasks like classification and segmentation[Qi+17a]. Building on this, PointNet++ extended the framework by introducing hierarchical feature learning that captures local structures at multiple scales, further enhancing the capability to handle complex point cloud data[Qi+17b]. These methods are used in multiple studies like Aoki et al., who introduce a new approach that combines PointNet with the Lucas & Kanade (LK) algorithm to create a trainable recurrent deep neural network for point cloud registration [Aok+19]. Or Ren et al., who present Spiking PointNet [Ren+24]. They tackle optimization and resource challenges using a trained-less but learning-more paradigm, where the model is trained with a single time step but performs better with multiple time steps during inference. Building on this, Peng et al. propose an end-to-end sparse-to-dense multi-encoder neural network (SDME-Net) for uniformly completing unstructured point clouds while preserving shape details [Pen+20]. SDME-Net directly processes incomplete and noisy point clouds without requiring transformations or prior knowledge of shape structures. The first stage generates a sparse but complete point cloud using a bistratal PointNet, and the second stage refines it into a dense, high-fidelity point cloud with PointNet++. The method utilizes distance and repulsion losses to ensure uniform distribution and accurate representation. Nonetheless, complex and/or noisy point clouds cause problems for methods that predict the entire point cloud from a single shape representation [Lin+21]. For this reason, it can be useful to work with attention. A Transformer is a type of neural network architecture introduced by Vaswani et al., which relies entirely on self-attention mechanisms to process input sequences, eliminating the need for recurrent or convolutional networks [Vas+17]. This design allows for highly parallelizable computations, significantly reducing training time while improving model performance. Transformers have shown superior results in tasks like machine translation, achieving state-of-the-art BLEU scores by efficiently capturing long-range dependencies within the data. Yan et al. introduce ShapeFormer, a transformer-based network designed for object shape completion using incomplete and potentially noisy point clouds [Yan+22]. ShapeFormer generates a distribution of plausible object completions, which can be sampled to produce detailed and faithful reconstructions. The network leverages a novel 3D representation called vector quantized deep implicit function (VQDIF), which employs spatial sparsity to approximate 3D shapes with a concise sequence of discrete variables. Chen et al. propose TransSC, a transformer-based shape completion model for robotic grasping [Che+22]. TransSC utilizes a transformer encoder for detailed point-wise feature extraction and a manifold-based decoder for capturing object details from sparse partial point clouds. Experimental results show that TransSC outperforms existing methods in shape completion and improves grasp generation in simulations. However, the methods listed here primarily handle noise-free or minimally noisy point clouds. In contrast, this work introduces a robust pipeline designed to reconstruct, refine, and complete noisy and often challenging-to-recognize point clouds.

Chapter 3

Theoretical Foundations

3.1 Fundamentals of Sonar Sensing

This section explores the principles and functionalities of a 2D forward-looking sonar (FLS) sensor and the methodology employed to create sonar maps. For a 2D FLS, these sonar maps manifest as sonar images, capturing the underwater environment in a two-dimensional representation [KK16, page 492]. Following this, the primary factors contributing to the significant noise characteristics inherent to this sensor are discussed. An in-depth examination of the origins and implications of these noise sources is provided, setting the stage for understanding the challenges and considerations in sonar data processing and analysis.

3.1.1 2D forward-looking Sonar Sensor

A 2D forward-looking sonar, as the name suggests, provides a two-dimensional representation of the underwater environment. It typically operates by emitting acoustic pulses in a fan-shaped pattern or in multiple directions. The system then receives echoes reflected from objects in the water, allowing it to generate a 2D image showing the distribution of objects within its field of view [AN13].

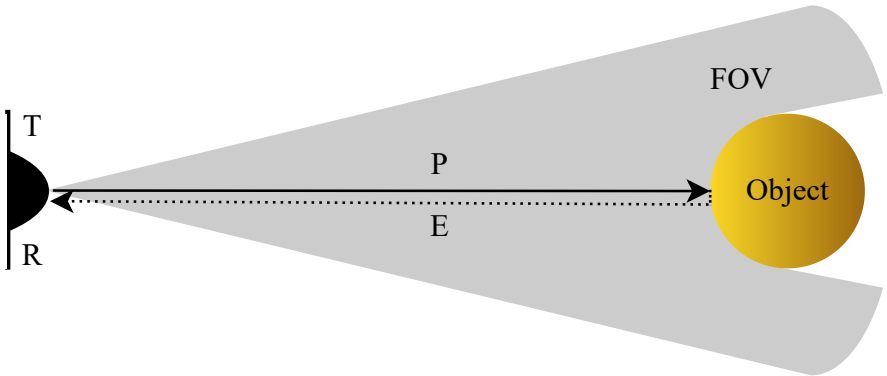

Figure 3.1: Adapted from [KK16, page 492], this figure illustrates a streamlined sonar system. A sonar transducer, denoted T/R, serves as both a transmitter (T) of a probing acoustic pulse (P) and a receiver (R) of echoes (E). The sonar beam's field of view (FOV) is delineated by the shaded area.

An imaging sonar sensor like the 2D forward-looking sonar, which measures the distance to an object using acoustic pulses and their echoes, is a widely used sensor in robotics [FRA21] [Hur+14]. Distance is proportional to the time taken for the echo to travel, assuming a

known acoustic velocity. At ultrasonic frequencies, the sonar beam provides directional information in addition to distance measurements, as the energy is concentrated in a specific direction. The low cost, lightweight, low power consumption, and minimal computational requirements of sonar compared to other range sensors have contributed to its popularity [Che+22]. Sonar can serve as the only practicable sensing modality in certain environments, such as underwater and low visibility [KK16, page 492]. In the system shown in Fig. 3.1, a sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse (P) and a receiver (R) of echoes (E). Specifically, an object within the sonar beams' field of view (FOV) and delineated by the shaded area in Fig. 3.1 reflects the probing pulse. Part of the reflected signal then reaches the transducer, where it is registered as an echo. The Time of Flight (TOF), here referred to as the echo travel time t_o, is determined from the moment the probing pulse is transmitted [KK16, page 492]. In this scenario, the echo waveform reflects the probing pulse.

$$r_{\mathrm{o}} = \frac{c\,t_{\mathrm{o}}}{2} \tag{3.1}$$

Where c is the speed of sound (343 m/s at standard temperature and pressure conditions), the object range r_o is derived from t_o using the formula 3.1. Multiplication by 2 converts the distance travelled in a round trip (P+E) to a measurement of the range to the object. The range of a sonar is limited by the losses due to beam scattering and acoustic absorption [KK16, page 492].

3.1.2 Sonar Map

To map the underwater environment onto a 2D image, a forward-looking sonar (FLS) scans the area ahead using a series of horizontally projected beams [KK16, page 492]. As shown in Fig. 3.2, φ describes the total orientation angle through a series of beams separated by $\Delta\varphi$. These beams are emitted from the sonar transducer and propagate forward to cover a wide field of view in front of the sonar, defined by the range settings $[r_{\min}, r_{\max}]$ and the angles φ and θ as shown in Fig. 3.3 and Fig. 3.2 [AN13].

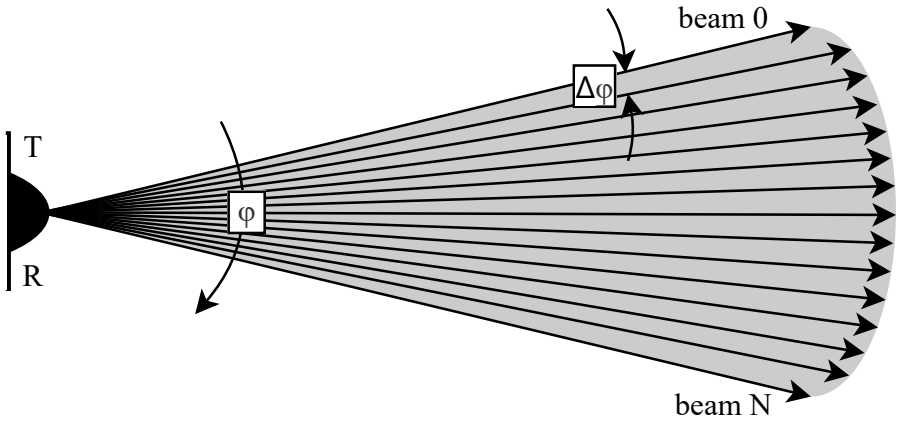

Figure 3.2: Based on [KK16, page 492], this figure shows a two-dimensional sonar mapping method. The sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse and a receiver (R) of echoes. φ indicates the orientation angle while $\Delta\varphi$ shows the angle between each beam of the series. Each series spans from beam 0 to beam N. The shaded area illustrates the FOV of the sonar sensor.

In addition to horizontal scanning, the FLS uses multiple series of beams projected along the vertical axis. Fig. 3.3 illustrates this process [AN13]. θ represents the total vertical angle,

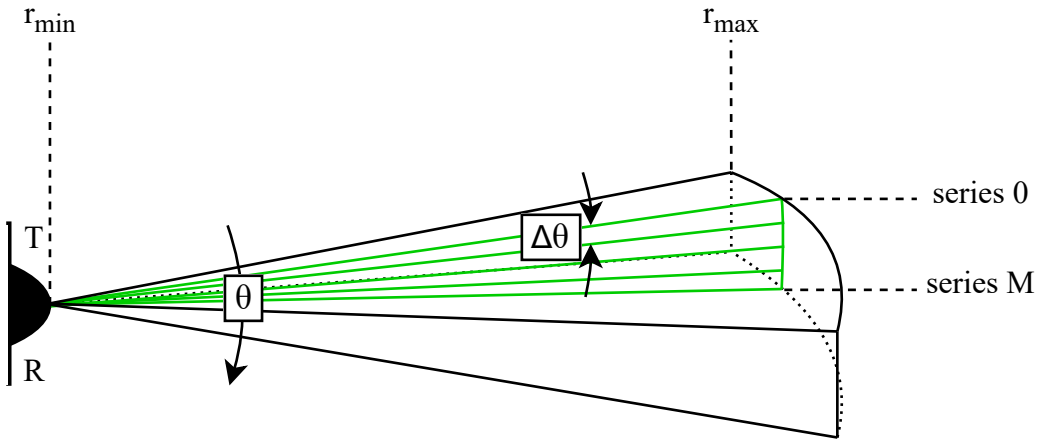

Figure 3.3: Modified from [AN13], this figure shows the vertical rotation of a two-dimensional sonar mapping method. The sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse and a receiver (R) of echoes. θ indicates the orientation angle while $\Delta\theta$ shows the angle between each series. Each series spans from series 0 to series M. The framed area represents the FOV. r_{min} describes the minimal range of the sonar sensor while r_{max} shows the maximum.

and $\Delta\theta$ is the angular distance between different series of beams in the vertical direction. By angling the beams horizontally and vertically, the sonar system effectively creates a grid-like pattern of scanning beams that extends horizontally and vertically. Therefore, θ and φ represent the dimension of the sonar map, while $\Delta\theta$ and $\Delta\varphi$ affect the resolution of the sonar map. Smaller values of $\Delta\theta$ and $\Delta\varphi$ lead to a denser scanning grid, which can produce more detailed images by capturing finer distinctions in the object's features and orientations [KK16, page 493].

By receiving the reflection, the amount of acoustic energy returning is measured, resulting in a pixel value between 0 and 255 [DLH19a, page 2]. 0 represents no echo received (black, indicating no objects or features), and 255 represents the strongest echo (white, indicating the presence of a very close or highly reflective object). Intermediate values represent varying degrees of reflectivity. Only the signals received within the time interval $\left[2\frac{r_{\text{min}}}{c}, 2\frac{r_{\text{max}}}{c}\right]$ are recorded to prevent the frames per second from being unnecessarily delayed [AN13] and to prevent sonar artefacts [Kuc01].
The forward-looking sonar system, therefore, produces a two-dimensional (2D) image of the underwater environment by combining the information from the horizontal and vertical beams. The sonar system captures returning echoes at various angles and distances, forming a polar coordinate system where an angle and a range define each data point [KK16, page 492]. This can be seen in Fig. 3.2 and Fig. 3.3 with φ and θ being the angles and r the range.
The final image appears as shown in Figure 3.4 when displayed on a Cartesian grid. The visualisation clearly shows the inherent polar coordinate system used for data acquisition [Wan+23].

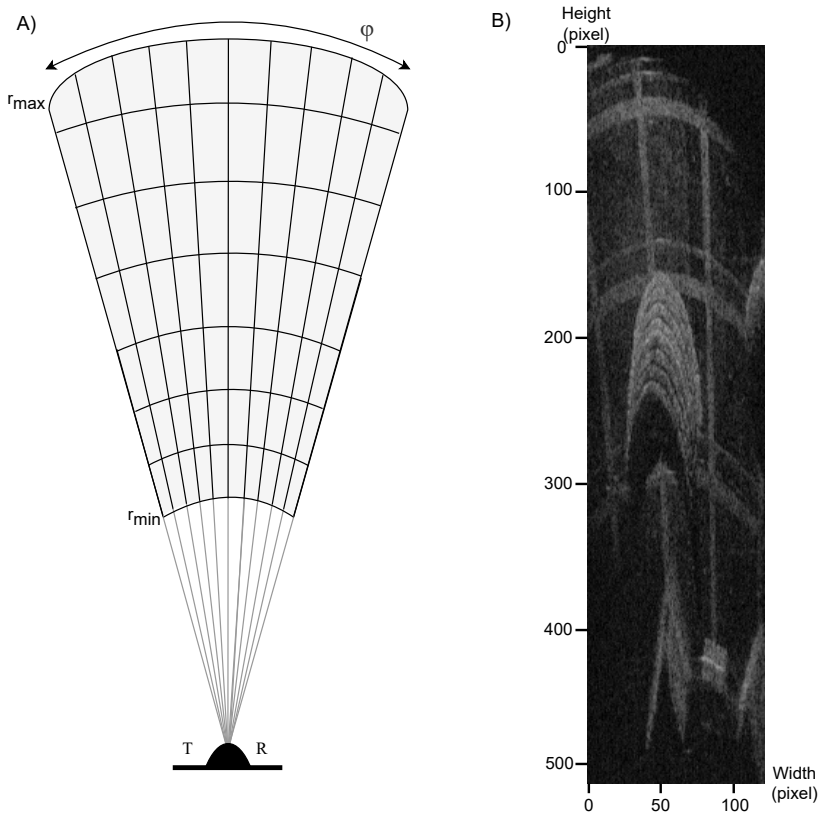

Figure 3.4: (A) Inspired by [Wan+23], this figure shows the projection model and the projection of a 3D point onto a 2D plane by the sonar sensor. The sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse and a receiver (R) of echoes. r_{min} and r_{max} display the range of the sensor while the grid symbolizes the pixels of the image in (B). φ explains the orientation angle, also displayed in 3.2. (B) This figure provides an example of a genuine 2D sonar image captured within an underwater experimental environment by the ARIS Explorer 3000 (Sound Metrics Corp.) sonar sensor. The image unveils the presence of five discernible objects, each positioned atop the floor of a water tank.

3.1.3 Etiology of Acoustic Interference in Sonar Imaging

Sonar Artefact

Smooth surfaces at an angle can cause so-called sonar artefacts. As shown in Figure 3.5, a smooth surface such as a wall can act as a mirror to the sonar, meaning that these smooth surfaces themselves don't produce detectable echoes [Kuc07]. This limitation can have significant consequences, as a robot using sonar to navigate might run into the smooth surface because it can't detect it.

Additionally, the side-lobbing and multiple reflections of the sonar beam can lead to false range readings, indicating the presence of non-existent objects in the environment. Multiple reflection artefacts are caused by delayed echoes produced by a previous probe pulse that exceeds the detection threshold after the current probe pulse has been transmitted [Kuc07][KK16, page 493]. Such artefacts then appear as virtual objects in the near range and obscure the actual objects in the far range in conventional sonars. This effect is visualized in Figure 3.5 by the "Virtual Object 1". Here, a reflection by the smooth surface takes too long to be received within the current detection threshold and is, therefore, received in the next detection threshold as a near-range virtual object. In order to avoid multiple reflection artefacts, most sonars use probing pulse emission periods longer than 50 ms [Kuc07]. Nevertheless, this type of artefact can still appear in complex environments due to a variety of

objects and ranges.
In some cases, slightly roughened or imperfectly smooth surfaces can create a few weak reflections that are not strong enough for reliable detection [HLY20]. This effect can also cause noise to be introduced into the sonar image, making it more challenging to interpret the data.

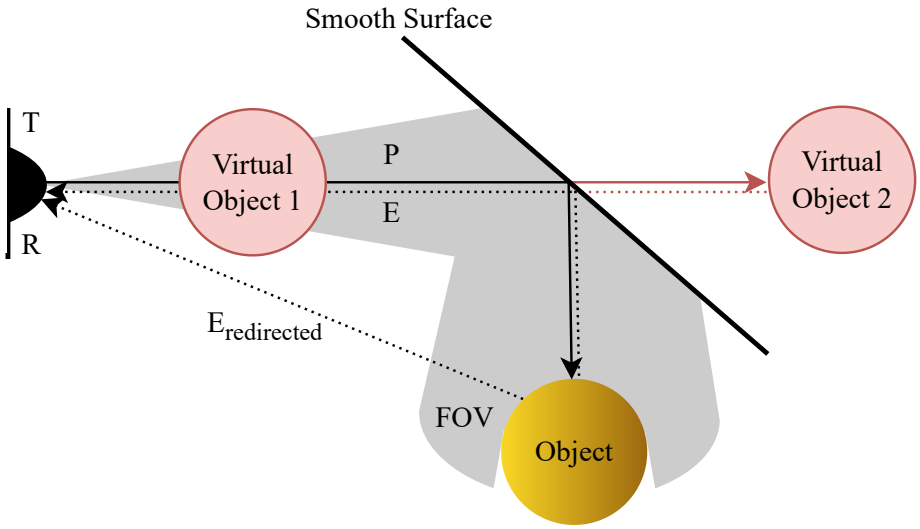

Figure 3.5: Based on [KK16, page 493], this figure displays the impact of a smooth surface redirecting beams (P) and their reflections (E), resulting in sonar artefacts at the position of a virtual object. The sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse and a receiver (R) of echoes. $E_{\text{redirected}}$ shows alternative reflections of the object. The sonar beam's field of view (FOV) is delineated by the shaded area.

A problematic artefact to identify is the non-axial MR artefact, as described in [Kuc01]. This occurs when a smooth surface hit at an oblique angle deflects the sonar beam towards another object that reflects the beam back to the sensor. Despite this, the sonar system will still display a range reading along its primary axis [Kuc07]. Although no object is located at the position shown on the sonar chart, the displayed position can serve as a reliable navigational landmark. Figure 3.5 also shows how the beam is redirected to an object. The reflected echo, redirected by the smooth surface back to the transducer, appears to be coming from the "Virtual Object 2" location, producing a sonar map similar to the one in Figure 3.1. As no real object is located at the virtual object's position, this measurement is an example of a non-axial artefact.

It is also important to note that the acoustic energy in Fig. 3.5 represented by $E_{\text{redirected}}$ can be directly reflected back to the position of the transducer but may not be detected because it falls outside the main beam cone [KK16, page 493]. If $E_{\text{redirected}}$ falls inside the main beam cone, the problem is often exacerbated by the side lobes of the beam that pick up these echoes, resulting again in incorrect short-range readings that are influenced by the orientation of the sonar and variations in echo travel time and amplitude due to inconsistencies in the speed of sound [KK16, page 493].

Speckle Noise

Speckle noise, a formidable challenge in various imaging modalities, including sonar, laser, microscopy, synthetic aperture radar (SAR) and medical ultrasound, is a complex phenomenon. Its presence severely degrades image quality, posing a significant hurdle in data interpretation and analysis [Che+17].

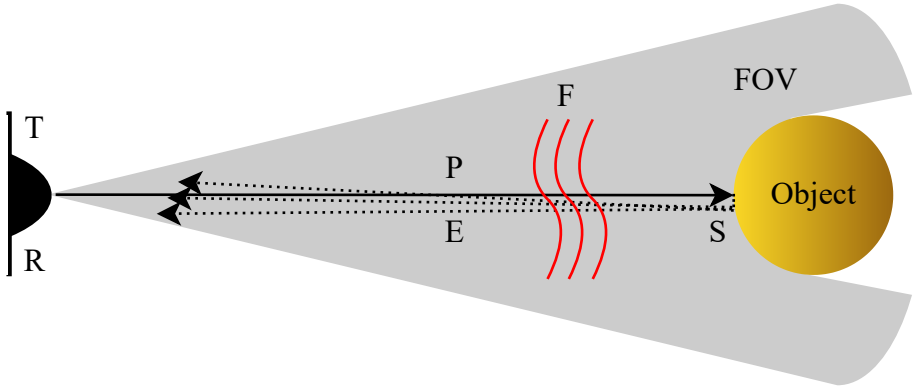

Figure 3.6: Based on [KK16, page 493], this figure displays the impact of thermal fluctuations (F) and a rough surface (S) affecting beams (P) and their reflections (E), resulting in speckle noise. The sonar transducer, denoted T/R, serves a dual purpose as both a transmitter (T) of a probing acoustic pulse and a receiver (R) of echoes. The sonar beam's field of view (FOV) is delineated by the shaded area.

In the context of sonar imaging, speckle noise results from the complex nature of underwater acoustic propagation and the coherent nature of the acquisition process. The propagation characteristics of the acoustic field and the unpredictable underwater environment introduce several artefacts, including radiometric distortion and speckle noise [Che+17] [Che+19]. There are several sources that contribute to the occurrence of speckle noise: As in Figure 3.6 shown, speckle noise, in particular, can be caused by the coherent interference of waves back-scattered by rough surfaces within a resolution cell [BPC18]. This interference occurs when waves reflected from multiple scatterers within the resolution cell interact constructively and destructively. The roughness of the scattering surfaces causes random phase shifts in the reflected waves. When these waves recombine, the different phase relationships cause fluctuations in the intensity of the received signal. Constructive interference produces bright spots, while destructive interference produces dark spots, giving the image a granular or "salt and pepper" appearance [HLY20]. This effect is shown in Fig. 3.6 marked by the scattering "S". The rough surface of the object is a significant factor, introducing random phase shifts in the waves and thereby influencing the direction and intensity of echos.

Another source for speckle noise is thermal fluctuations denoted as "F" in Fig. 3.6. Figure 3.6 illustrates how thermal variations in the water can significantly affect sonar readings. These thermal variations cause the speed of sound to change, which can cause the sound waves to speed up or slow down as they travel. This change in speed affects the travel time of the echoes, resulting in inconsistencies in the timing of the received signals [KK16, page 493]. In addition, thermal variations can cause refraction. This occurs when sound waves bend and change direction as they pass through layers of water at different temperatures. This redirection of the echoes can change the path of the echoes, causing them to arrive at the sonar receiver from an unexpected angle [KK16, page 493]. These combined effects - changes in speed and redirection - cause temporal variations. This means that the echoes

arrive at slightly different times and angles than anticipated. They also cause amplitude variations, where the strength or loudness of the echoes varies. This results in jitter in the range readings, where the distance readings from the sonar become unstable and less reliable, resulting in speckle noise [KK16, page 493].

Also, research as [Mid67] has shown that reverberation is a significant source of speckle noise. Reverberation is the primary background noise in active sonar systems. Reverberation is the sum of scattered waves received from reflections at the seawater boundary and scattering within the seawater volume. The echoes from different scatterers either cancel or amplify each other, causing fluctuations in the amplitudes of the echoes. This phenomenon is manifested as a grainy black-and-white texture on the sonar image, known as speckle noise [HLY20].

3.2 Enhancement Techniques for Sonar Data: Estimation, Denoising, and Reconstruction

This section delves into the techniques used in this work to enhance the usability, quality and reliability of sonar data, focusing on estimation, denoising, and reconstruction methods. The first part introduces the Elevation Angle Estimation Network (EAE-Net), which estimates the elevation angle and derives 3D point clouds from 2D sonar images [Wan+23]. It covers the overall functionality, motion model, architectural components, and the loss function used for optimization. Following this, the focus shifts to point cloud denoising techniques, including Hypergraph methods, Multi-normal-guided methods, and DBSCAN. Each of these techniques is analyzed for their effectiveness in handling different noise types that commonly affect sonar data. Finally, the section explores the Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion (PCTMA-Net) [Lin+21]. This part provides an overview of the network, its architecture, the role of the Transformer Encoder, the Multi-Head-Attention mechanism, and the Morphing-Atlas-Based Point Generator Network. Each of these topics will be thoroughly explained to provide a comprehensive understanding of the methods employed in this work.

3.2.1 Elevation Angle Estimation Net (EAE-Net)

Overview

Motion Degeneracy in Self-Supervised Learning of Elevation Angle Estimation for 2D Forward-Looking Sonar by Yusheng Wang, Yonghoon Ji, Chujie Wu, Hiroshi Tsuchiya, Hajime Asama, and Atsushi Yamashita addresses a significant challenge in underwater robotic perception: estimating the missing elevation angle from 2D forward-looking sonar (FLS) images, resulting in 3D point clouds. This task is crucial for generating accurate 3D reconstructions of underwater environments, which are essential for various applications such as mapping, navigation and object detection [Wan+23]. The results of this method are used as the foundation of this thesis.

Motion degeneracy often challenges traditional methods, which can lead to failures during the training process. Motion degeneracy refers to situations in which the motion patterns of the sonar sensor provide insufficient or ambiguous information, making it difficult to accu-

rately estimate elevation angles in 2D forward-looking sonar imaging [Wan+23]. This phenomenon can occur during pure translational movements, in flat or repetitive environments, or when encountering low-reflectivity surfaces, leading to multiple possible solutions for the elevation data. To overcome this problem, the authors first analyse the motion field of 2D FLS, which is central to understanding the supervision signal required for self-supervised learning. By identifying effective motions that contribute positively to the training process, they propose a method that ensures stable learning without the need for synthetic data [Wan+23]. To achieve that, the proposed method uses the motion of the sonar sensor to enable self-supervised learning, eliminating the need for pre-training with synthetic data. By analysing the motion field, the method identifies effective motions, such as z-axis translations and x-axis rotations, which provide reliable training input. These motions are used to generate synthetic target images from the source images, ensuring sufficient overlap and photometric consistency. This process allows the neural network to learn accurate elevation angles by minimising the difference between the actual and synthetic images, effectively overcoming the challenges of motion degeneracy. Both simulation and real-world experiments validate the effectiveness of this approach [Wan+23].

The data is generated as video in a real-world and synthetic setup. The motion data is recorded together with a video of the sensor. At each time step, a motion and a sonar image are stored. The real-world setup takes place in a water tank with 18 objects placed on the bottom using the ARIS EXPLORER 3000 sonar sensor. The synthetic data were simulated in a congruent Blender environment and recorded by a simulated sonar sensor [Wan+23].
This method uses two adjacent frames from the acoustic video: the target view image I_t and the source image I_s. A synthetic target view image $\tilde{I}_t$ is generated from I_s, the motion $M_{t\to s}$ and the estimated pixel-wise target view elevation angle $E_t = g(I_t)$, where $g(.)$ denotes the neural network. The synthetic image $\tilde{I}_t$ is expressed as $\tilde{I}_t = \pi(I_s, E_t, M_{t\to s})$, where $\pi(.)$ represents the inverse warping process. This process is explained in Section 3.2.1. The general goal is to optimise:

$$\underset{E_t}{\operatorname{argmin}} \mathcal{L}\left(I_t, \tilde{I}_t\right) \tag{3.2}$$

where $\mathcal{L}$ minimises the difference between the original target image and the synthesised image [Wan+23]. Because sonar images are noisy, a pre-trained network generates a signal mask to filter out background noise and multipath reflections, improving performance.

Sonar Image Motion Model

To comprehend the dynamics and principles underlying the motion model for 2D forward-looking sonar, it is essential to examine the basic motion model. The following calculation and description of the motion model is based on the work by Wang et al. [Wan+23], unless otherwise specified. A point in 3D space, represented in the sonar coordinate system, can be described as:

$$\boldsymbol{p}_s = \begin{bmatrix} r\cos\phi\cos\theta \\ r\cos\phi\sin\theta \\ r\sin\phi \end{bmatrix} \tag{3.3}$$

where r represents the range or distance from the sonar to the point. ϕ is the elevation angle, which measures the vertical angle from the horizontal plane. θ is the azimuth angle that measures the horizontal angle from a reference direction [Chu+01]. The corresponding position of this point in 2D sonar image coordinates is:

$$\boldsymbol{s} = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix} \tag{3.4}$$

Here, x_s and y_s are the coordinates of the point in the 2D image plane. Thus, the relationship between p_c and s can be determined.

$$\boldsymbol{s} = \frac{1}{\cos\phi} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{p_s} \tag{3.5}$$

Equation 3.5 as in [Neg13] represents the projection of the 3D point onto the 2D plane by scaling the x and y components with the inverse of $\cos\phi$. This effectively flattens the 3D point into 2D image coordinates. The rate of change of the elevation angle ϕ with respect to time is related to the rate of change of the 3D point $\boldsymbol{p}_s$ with respect to time. This relationship between $\frac{d\phi}{dt}$ and $\frac{dp_s}{dt}$ is given by:

$$\frac{d\phi}{dt} = \frac{1}{r}\begin{bmatrix} -\cos\theta\sin\phi & -\sin\theta\sin\phi & \cos\phi \end{bmatrix} \frac{dp_s}{dt} \tag{3.6}$$

The rate of change of the 3D point coordinates into the rate of change of the elevation angle is converted by the matrix in 3.6. Using the relations 3.5 and 3.6 to analyze the displacement and, therefore, the motion itself, we consider the displacement $\frac{ds}{dt}$ of the pixel positions over time. The rate of change of s with respect to time t is given by:

$$\frac{d\boldsymbol{s}}{dt} = \frac{1}{\cos\phi}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \frac{d\boldsymbol{p}_s}{dt} + \tan\phi\,\boldsymbol{s}\,\frac{d\phi}{dt} \tag{3.7}$$

where $\frac{d\boldsymbol{p}_s}{dt}$ is the change in the 3D coordinate projected onto the 2D plane. When the sonar undergoes a small motion, described by translation $\boldsymbol{t}$ and rotation $\boldsymbol{\omega}$, the displacement of a stationary 3D point in the sonar coordinates follows the rigid body motion [Neg13]:

$$\frac{d\boldsymbol{p}_s}{dt} = -\boldsymbol{\omega} \times \boldsymbol{p}_s - \boldsymbol{t} \tag{3.8}$$

Combining the above equations 3.7 and 3.8, we can express the general form of the relationship between the pixel displacement and the sonar motion, known as the motion field [Wan+23] [Neg13]:

$$\begin{bmatrix} \frac{dx_s}{dt} \\ \frac{dy_s}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{t_x}{\cos\phi} - \left(\frac{t_z\sin\phi}{r}\right)x_s - \omega_z y_s + \left(\frac{\sin\phi\tan\phi\, t_x}{r^2}\right)x_s^2 \\ +\left(\frac{\sin\phi\tan\phi\, t_y}{r^2} + \frac{\tan\phi\,\omega_x}{r}\right)x_s y_s + \left(\frac{\tan\phi\,\omega_y}{r}\right)y_s^2 \\ -\frac{t_y}{\cos\phi} - \left(\frac{t_z\sin\phi}{r}\right)y_s + \omega_z x_s + \left(\frac{\sin\phi\tan\phi\, t_y}{r}\right)y_s^2 \\ +\left(\frac{\sin\phi\tan\phi\, t_x}{r^2} - \frac{\tan\phi\,\omega_y}{r}\right)x_s y_s - \left(\frac{\tan\phi\,\omega_x}{r}\right)x_s^2 \end{bmatrix} \tag{3.9}$$

The translations t_x, t_y and t_z and the rotations w_x, w_y and w_z along the x, y and z axis of this equation are displayed in Figure 3.7.

Considering the specific sonar system, the ARIS EXPLORER 3000, this general term 3.9 can be simplified based on the specific properties of the sensor.

With an elevation angle range of $[7°, -7°]$ as displayed in Fig 3.7, certain approximations can simplify the calculations.

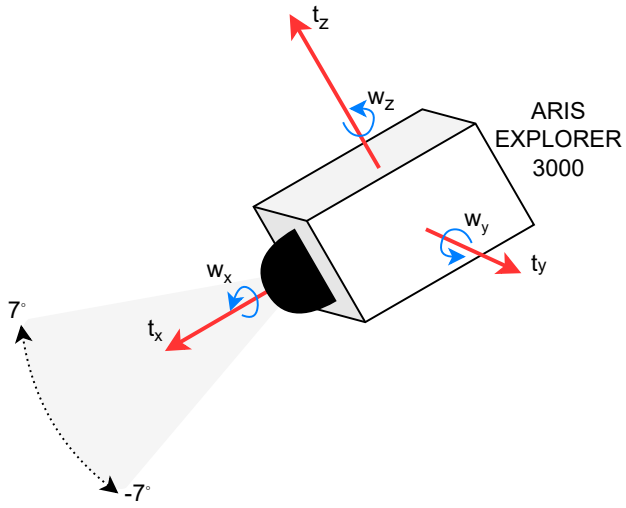

Figure 3.7: Based on [Wan+23], this figure displays the different types of motion the sonar sensor can perform. Translations along the x, y, and z axes are defined as t_x, t_y, and t_z. Rotations along the x, y, and z axes are defined as w_x, w_y, and w_z. The aperture angle ranges from $[7°, -7°]$.

- The range of $\cos\varphi$ is about $[0.9925, 1]$. An approximation $\cos\varphi \approx 1$ can be made since this is very close to 1.
- The range of $\sin\varphi$ is around $[-0.1219, 0.1219]$.
- The range of $\tan\varphi$ is about $[-0.1228, 0.1228]$.
- The range of $\sin\varphi \tan\varphi$ is $[0, 0.015]$, which is small enough to be approximated as zero.

Resulting in the simplified equation:

$$\begin{bmatrix} \frac{dx_s}{dt} \\ \frac{dy_s}{dt} \end{bmatrix} = \begin{bmatrix} -t_x - \left(\frac{t_z \sin\phi}{r}\right) x_s - \omega_z y_s + \left(\frac{\tan\phi\omega_x}{r}\right) x_s y_s + \left(\frac{\tan\phi\omega_y}{r}\right) y_s^2 \\ -t_y - \left(\frac{t_z \sin\phi}{r}\right) y_s + \omega_z x_s - \left(\frac{\tan\phi\omega_y}{r}\right) x_s y_s - \left(\frac{\tan\phi\omega_x}{r}\right) x_s^2 \end{bmatrix} \tag{3.10}$$

Using the simplified motion model 3.10 the following insights are experimentally derived:

- Basic motions such as rotation around the x-axis and translation along the z-axis are sensitive to the elevation angle φ. This sensitivity means that changes in φ significantly affect the motion field, making these motions particularly effective for tasks like 3D reconstruction where detectable changes in the images between time steps are needed.
- Horizontal motions, including translations along the x and y axes (t_x and t_y) and rotation around the z-axis (ω_z), are independent of the elevation angle φ in the motion field. As a result, these motions do not provide useful information for 3D reconstruction, as they do not affect the vertical position of points in the sonar image.
- The displacement resulting from rotation around the y-axis (ω_y) is relatively small. This small displacement is difficult to detect with sonar imaging due to the resolution, making ω_y less useful for applications that require precise elevation angle estimation.

Therefore, for effective dataset generation, it is beneficial to focus on basic motions such as rotation around the x-axis and translation along the z-axis. These motions are more sensitive to changes in the elevation angle and provide more significant contributions to 3D reconstruction accuracy [Wan+23].

Model Architecture

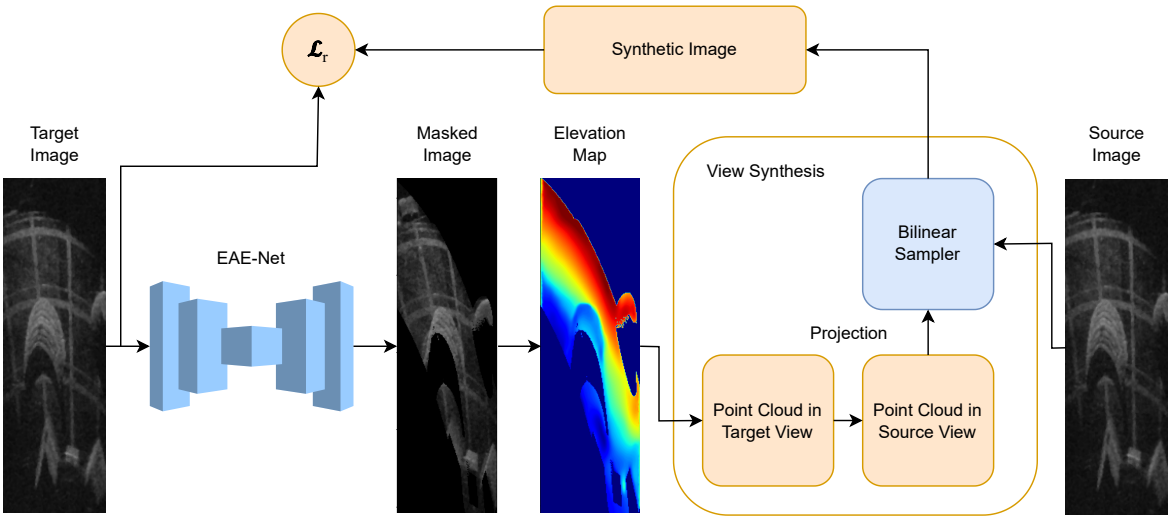

Figure 3.8: Adapted from [Wan+23], this figure illustrates the self-supervised learning framework in the training setup. The Elevation Angle Estimation Network (EAE-Net) trains the estimation of the elevation angle. The Masked Image is derived by a pre-trained UNet structure for binary segmentation.

The architecture of the EAE-Net (Elevation Angle Estimation Network) [Wan+23], developed to estimate elevation angles, is shown in Figure 3.8. The subsequent description of the architecture is primarily based on the work of Wang et al. [Wan+23], unless indicated otherwise. EAE-Net is based on a UNet architecture known for its effectiveness in image segmentation tasks [Li+21]. The network structure includes an encoder-decoder configuration where the encoder progressively reduces the spatial dimensions of the input while increasing the number of feature channels, and the decoder progressively restores the spatial dimensions to the original input size. This architecture allows both local and global features to be captured efficiently [Li+21].

To ensure that the output values of the elevation map are between 0 and 1, a sigmoid activation layer is placed before the final output. The elevation maps produced by EAE-Net are linearly mapped to the area defined by the aperture angle. This is referred to as $\varphi_{\text{aperture}}$ as one can see in Figure 3.2 and 3.4. Specifically, this mapping translates the values to the interval $\left[-\frac{\varphi_{\text{aperture}}}{2}, \frac{\varphi_{\text{aperture}}}{2}\right]$.

One of the main challenges addressed by this framework is the lack of 3D ground truth data for monitoring. To overcome this problem, two consecutive frames from an acoustic video are used in the training process. The first step is to estimate the target view elevation map E_t, which is then converted into a point cloud representation by mapping each pixel to its corresponding 3D coordinates using the estimated elevation (see Figure 3.8).

Known motion between frames, represented by the transformation matrix $M_{t\to s}$, is applied to the point cloud, transforming it to source coordinates. The synthetic target image $\tilde{I}_t$ is generated from the transformed point cloud by sampling the source image I_s. Bilinear sampling ensures that the generation of $\tilde{I}_t$ is smooth and differentiable, essential for backpropagation during training. During image transformation, bilinear sampling calculates the value of a new pixel by taking a weighted average of the four closest pixels in the original image. Compared to more straightforward methods, such as nearest neighbour interpolation, this method ensures smooth transitions and preserves image detail.

Optimisation of the network is achieved by minimising the difference between the actual target image I_t and the synthetic target image $\tilde{I}_t$. This difference is quantified using a loss function $\mathcal{L}$, which guides the network in adjusting its parameters to improve accuracy.

In Forward-Looking Sonar (FLS), an elevation map and known motion transformations synthesize a target image from a source image. The algorithm uses the previously derived field of motion to achieve this synthesis.

- The inputs to the algorithm are the target view elevation map E_t, the source image I_s, and the motion transformation $M_{t\to s}$.
- The output is a synthetic target view image $\tilde{I}_t$ generated from the source image.

For each point (r_t, θ_t, ϕ_t) in the elevation map E_t, the point is first transformed into Euclidean coordinates p_t. To obtain the new point p_s, the motion transformation $M_{t\to s}$ is then applied. This point p_s is projected back into image coordinates, producing pairs (r_s, θ_s). Using bilinear sampling, which ensures smooth and differentiable interpolation, the intensity i_s of each point is calculated. This computed intensity i_s is then assigned to the corresponding coordinates (r_t, θ_t) in the synthetic target image $\tilde{I}_t$. Once all the points have been processed, the algorithm returns the entirely constructed synthetic target image $\tilde{I}_t$. This method ensures that the synthetic target image closely matches the actual target image by accurately transforming and projecting points based on the elevation map and known motion [Wan+23].

Loss Function

This description of the loss function of the EAE-Net follows the framework established by Wang et al. [Wan+23], except where noted. A combination of L1 loss and Structural Similarity Index (SSIM) loss is used, collectively referred to as the reconstruction loss $\mathcal{L}_r$. The SSIM is defined as:

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha}[c(\mathbf{x}, \mathbf{y})]^{\beta}[s(\mathbf{x}, \mathbf{y})]^{\gamma}, \tag{3.11}$$

where $\alpha > 0, \beta > 0$ and $\gamma > 0$ control the relative significance of each of the three terms of the index. The three terms in the Structural Similarity Index (SSIM) together estimate the visual impact of different types of image degradation. These terms assess the impact of shifts in the image's luminance, changes in the contrast, and any other residual differences that can be categorised as structural changes. By assessing these aspects, SSIM measures perceived image quality, capturing both small and large distortions that affect the overall visual experience [DY11]. The reconstruction loss is expressed as:

$$\mathcal{L}_r = \beta \mathcal{M}_t \left(1 - \text{SSIM}\left(I_t, \tilde{I}_t\right)\right) + (1-\beta)\mathcal{M}_t \left\|I_t - \tilde{I}_t\right\|_1, \tag{3.12}$$

Here β is a hyperparameter that controls the balance between the SSIM loss and the L1 loss. The term $\mathcal{M}_t$ represents a mask that is applied to the target image in order to focus the loss calculation on certain regions. In this framework, β is set to 0.3. The SSIM loss component measures the structural similarity between the real image and the synthetic image, emphasising perceptual quality, while the L1 loss measures the absolute differences between the images, ensuring accuracy on a pixel-by-pixel basis. In addition to the reconstruction loss, a smoothing loss $\mathcal{L}_s$ is applied to the elevation maps in the target view E_t as illustrated in Figure 3.8. This edge-aware smoothness loss is designed to ensure that the elevation maps remain smooth in regions where there is little or no texture, while still allowing for sharp edges where there are significant changes in the image. The smoothness loss is expressed as:

$$\mathcal{L}_s = |\partial r \mathcal{M}_t E_t| e^{-|\partial r I_t|} + |\partial \theta \mathcal{M}_t E_t| e^{-|\partial \theta I_t|}. \tag{3.13}$$

In this equation, ∂r and $\partial \theta$ represent the partial derivatives with respect to distance and angle, respectively. The exponential terms $e^{-|\partial r I_t|}$ and $e^{-|\partial \theta I_t|}$ act as edge detectors. They reduce the influence of smoothness loss in regions with high texture (i.e. sharp edges) and enforce smoothness in flatter regions.
The total loss function $\mathcal{L}$ used to optimise the network is a weighted sum of the reconstructed and smoothed loss:

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_s \mathcal{L}_s, \tag{3.14}$$

where λ_r and λ_s are weights that balance the contribution of each loss term to the total loss. In this framework, λ_r is set to 2 and λ_s is set to 1, indicating a greater emphasis on reconstructing loss compared to smoothing loss. The combination of these loss functions guides the network to generate elevation maps that not only closely match the target imagery in terms of both pixel accuracy and perceptual quality, but also remain smooth in areas of low texture. This dual focus helps to achieve high-quality, realistic elevation estimates suitable for 3D reconstruction tasks.

3.2.2 Point Cloud Denoising

Point clouds are among the most popular representations of 3D objects and environments, along with meshes and RGB-D images [Zha+19]. A point cloud is a dense collection of discrete, unstructured points in a three-dimensional coordinate system that is an approximation of the geometry of 3D data. Formally, a point cloud Q with n points can be represented as $Q = \{q_i \mid i = 1, ..., n\}$, where each point $q_i \in \mathbb{R}^3$ in 3D space is associated with various attributes such as position (x_i, y_i, z_i) and colour (r_i, g_i, b_i) [Zho+22]. In spite of their advantages, the point clouds acquired by these sensors often suffer from varying levels of noise and outliers due to measurement errors [Jav+17]. Noise in point clouds can come from a variety of sources, including the acquisition device, the limitations of the sensor, and the reflective or lighting conditions of the surfaces being scanned [Zho+22]. In the case of this thesis, the sources of noise are explained in Section 3.1.3.

The presence of noise not only degrades the quality of point clouds. It also hinders subsequent geometry processing tasks [Jav+17]. Therefore, denoising techniques are essential to improve the quality of point clouds while preserving as many essential geometric features as possible [Jav+17]. It is imperative to choose denoising algorithms and their parameters carefully, given the variety of noise types and the specific requirements of future processing tasks. In this Section, different denoising and noise reduction algorithms are examined. Their strengths and limitations are discussed.

Hypergraph

Graph-based denoising methods interpret the input point cloud as a graph signal and denoise using selected graph filters. This approach takes advantage of the inherent relationships between the graph and point cloud features, such as the correlation between smoothness over a graph and the flatness of surfaces in point clouds. The success of graph signal processing techniques in point cloud processing is attributed to the unique ability to capture the underlying geometric structures of point clouds [ZCD21]. A weighted graph $G = (V, E, W)$ is typically defined by two sets: a node-set V of cardinality $|V| = n$ and an edge set E, as shown in Figure

3.9. Nodes in a graph represent entities (e.g., points in a point cloud). Edges represent relationships between these entities. The matrix W is a weighted adjacency matrix, where each weight $w_{i,j} \in W$ assigned to an edge $e_{ij} \in E$ reflects the degree of pairwise similarity between the node v_i and the node v_j [Zho+22].

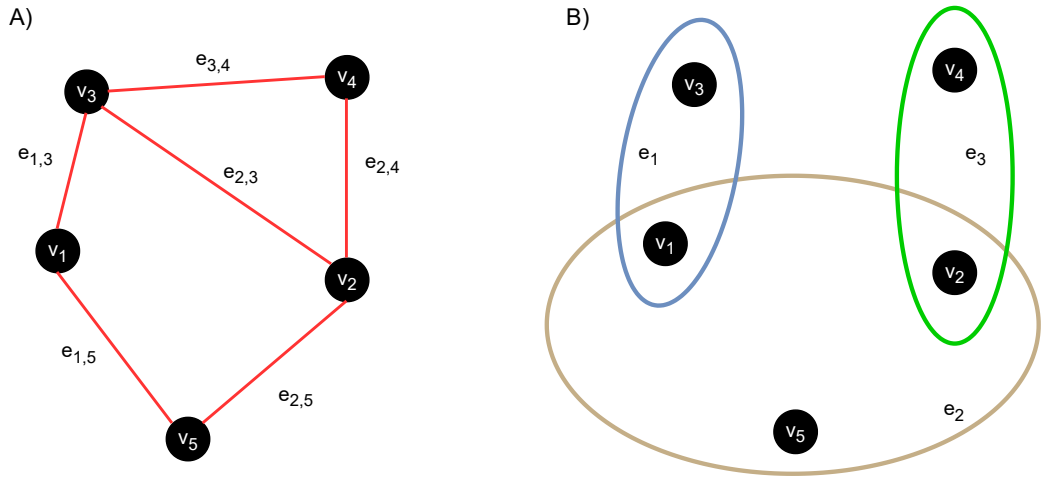

Figure 3.9: Based on [Zho+22]. (A) A graph $G = (V, E)$ with nodes $V = \{v_1, v_2, v_3, v_4, v_5\}$ and edges $E = \{e_{1,5}, e_{1,3}, e_{2,3}, e_{2,4}, e_{2,5}, e_{3,4}\}$. (B) A hypergraph $H = \{V, E\}$ with nodes $V = \{v_1, v_2, v_3, v_4, v_5\}$ and hyperedges $E = \{e_1, e_2, e_2\}$.

Graph-based methods can be categorised based on the approach to graph construction, which are:

- **Point-based methods**: These methods construct graphs directly from the points in the point cloud. Each point is treated as a node, and edges are formed based on spatial proximity or other similarity metrics. This direct approach, critical for effective denoising, ensures that each point's local neighbourhood is captured [Zho+22].
- **Patch-based methods**: These methods group points into patches before constructing the graph. Each patch can be considered a minor, coherent point cloud region, often with similar geometry characteristics. By working with patches, the collective information from a patch provides a more robust basis for denoising, allowing these methods to deal with noise and outliers more effectively [Zho+22].
- **Hypergraph-based methods**: Unlike traditional graphs, hypergraphs allow edges (hyperedges) to connect more than two nodes, as illustrated in Figure 3.9. This allows higher-order relationships between points to be captured. As a result, hypergraph-based methods can more effectively model complex geometric structures, making them particularly suitable for denoising tasks where the preservation of intricate detail is critical [Zho+22].

The problem of denoising a signal on a hypergraph is formulated as a convex minimisation problem. The constraint is that the denoised signals should be smooth over the hypergraph.

Hypergraph signal processing uses a tensor-based framework. A tensor can be interpreted as a multidimensional array. The tensor outer product between a tensor of order m $\mathbf{U} \in \mathbf{R}^{I_1 \times \ldots \times I_m}$ with entries $u_{i_1 \ldots i_m}$ and a tensor of order n th order tensor $\mathbf{V} \in \mathbb{R}^{J_1 \times \ldots \times J_n}$ with entries $v_{j_1 \ldots j_n}$ can be defined as:

$$\mathbf{O} = \mathbf{U} \cdot \mathbf{V} \tag{3.15}$$

where $\mathbf{O} \in \mathbf{R}^{I_1 \times \dot{\times} I_m \times J_1 \times \dot{\times} J_n}$, and $o_{i_1 \cdot i_m j_1 \cdot j_n} = u_{i_1 \cdot i_m} \ldots v_{j_1 \ldots j_n}$ [Zho+22].

The noisy point cloud with n nodes is denoted by a location matrix $\mathbf{s} = [s_1\ s_2\ \ldots\ s_n]^{\mathrm{T}}$, where s_i denotes the coordinates of the i-th point. Given a hypergraph with n nodes and the longest hyperedge connecting m nodes, it can be represented by an m-order n dimension representing a tensor $\mathbf{A} = \left(a_{i_1 i_2 \ldots i_m}\right) \in \mathbf{R}^{n^n}$, whose entry at position $(i_1, i_2, \ldots, i_m)$ is labelled as $a_{i_1 i_2 \ldots i_m}$. This adjacency tensor, called the representation tensor $\mathbf{A}$, indicates whether nodes $v_1, v_2, \ldots, v_m$ are connected in the hyperedges. The representation tensor $\mathbf{A}$ can be decomposed as follows:

$$\mathbf{A} \approx \sum_{r=1}^{n} \lambda_r \cdot \underbrace{\mathbf{f}_r \cdot \quad \ldots \quad \cdot \mathbf{f}_r}_{m \text{ times}}, \tag{3.16}$$

where $\mathbf{f}_r$'s are orthonormal basis vectors called spectral components, and λ_r are frequency coefficients related to the hypergraph frequency [ZCD21].

For clean point clouds, the spectrum components $\mathbf{f}_r$ based on the hypergraph stationary process and optimally determined their frequency coefficients λ_r based on smoothness to recover the original hypergraph structure. Given the original signal $\mathbf{s} = [s_1 \ldots s_n]^{\mathrm{T}}$, the hypergraph signal is defined as the $(m-1)$-times tensor outer product of $\mathbf{s}$, i.e [Zho+22]:

$$\mathbf{s}^{[m-1]} = \underbrace{\mathbf{s} \cdot \mathbf{s} \cdot \quad \ldots \quad \cdot \mathbf{s}}_{m-1 \text{ times}}. \tag{3.17}$$

Using the hypergraph signal $\mathbf{s}^{[m-1]}$ and the representation tensor $\mathbf{A}$, they jointly estimated hypergraph spectral pairs $(\mathbf{f}_r, \lambda_r)$ and denoised the noisy point clouds [Zho+22].

Multi-normal-guided Method

Normal-guided point cloud denoising methods take into account not only the positions of the points but also the point normals as guidance signals. The estimated normals are filtered iteratively. The updated normal field from the previous iteration is used as a guide. Finally, the positions of the points are adjusted to be aligned with the estimated normals [Han+18]. One of the notable approaches in this category is the iterative guidance normal filter. The iterative process can be described by the following equations [Han+18]:

$$\mathbf{n}_i^{k+1} = \frac{1}{K_i} \sum_{p_j \in Y(p_i)} w_d\left(\left\|p_{ij} - p_i\right\|\right) w_n\left(\left\|\mathbf{n}_i^k - \mathbf{n}_j^k\right\|\right) \cdot \mathbf{n}_j^k, \tag{3.18}$$

where:

$$K_i^k = \sum_{p_v \in \mathcal{N}(p_i)} w_d\left(\left\|p_{ij} - p_i\right\|\right) w_n\left(\left\|\mathbf{n}_i^k - \mathbf{n}_j^k\right\|\right). \tag{3.19}$$

where:
- $\mathbf{n}_i^{k+1}$ is the updated normal at point p_i in the $(k+1)$-th iteration.
- $\mathbf{n}_i^k$ and $\mathbf{n}_j^k$ are the normals at points p_i and p_j in the k-th iteration, respectively.
- $Y(p_i)$ is the neighborhood of point p_i.
- $w_d\left(\left\|p_{ij}-p_i\right\|\right)$ is a spatial weighting function based on the distance between points p_i and p_j.
- $w_n\left(\left\|\mathbf{n}_i^k-\mathbf{n}_j^k\right\|\right)$ is a normal weighting function based on the difference between normals $\mathbf{n}_i^k$ and $\mathbf{n}_j^k$.
- K_i is a normalization factor ensuring the sum of the weights is 1.

The initial normal $\mathbf{n}_i^0$ for each point p_i is estimated using Principal Components Analysis (PCA). In the k-th iteration, $\mathbf{n}_i^{k+1}$ is computed in a bilateral filtering form with respect to the normal $\{\mathbf{n}_i^k\}$ from the previous iteration. The normal field from the previous iteration is used as a guide in the above equation to filter the newly adjusted normal field. After obtaining filtered normal fields, each point is adjusted to match its estimated normal field by extending the iterative point updating scheme proposed by Sun et al [Sun+07]. However, this isotropic point updating process handles each point uniformly, which may lead to the smoothing of sharp areas such as corners and edges.

In this paper, Zheng et al. extend the guidance normal filter, originally used for mesh normal smoothing, to point clouds using a multiple normal strategy [Zhe+17]. In this method, multiple normals are estimated for each feature point by partitioning its k-nearest neighbours (k-NN) into piecewise smooth patches. Each patch corresponds to one normal. The guide normal for each point is then evaluated by computing the average normal of its most consistent patch. This guided normal is then used to filter the normal field, which results in a piecewise smooth normal field.

While most denoising methods focus on removing noise while preserving geometric features, Zheng et al. took a different approach by using a multi-normal strategy to remove different scales of geometric features from noisy point clouds [Zhe+18]. They extended the mesh rolling guidance normal filter to process the normal field on the point cloud. To deal with normal discontinuities along sharp features, they applied the multi-normal strategy during the point position update. This approach is particularly robust concerning the removal of small-scale geometric features [Zho+22]. Building on this, Sun et al. used the rolling guidance normal filter to suppress multiscale textures while preserving prominent structures in textured point clouds [Sun+19]. However, this method may inadvertently filter out detailed features containing crucial semantic information [Zho+22].

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm. It identifies clusters based on an estimate of the minimum density level. This method uses two main parameters: The minimum number of points (minPts) required to form a dense region and a radius (ε) that defines the neighbourhood around each point [Sch+17].

- **Core points**: If a point has at least minPts neighbors within the radius ε, it is considered a core point. The neighbours include the point itself.
- **Direct density reachability**: All points are considered part of the same cluster within the core point's radius ε.

- **Density reachability**: The neighbourhoods of a core point are transitively included in the cluster if a neighbour of the core point is also a core point.
- **Boundary points**: Points within the ε radius of a core point but do not have enough neighbours of their own to be a core point are called boundary points.
- **Noise points**: Points that do not have a density reachable from any core point are considered noise. They do not belong to any cluster.

The basic intuition behind DBSCAN is to find areas of high density separated by areas of lower density. This allows the algorithm to find clusters of any shape and effectively discriminate between noise and clusters [Sch+17]. For efficiency, DBSCAN avoids direct density estimation between points. Instead, it focuses on neighbours within the ε radius of the core points to form clusters.

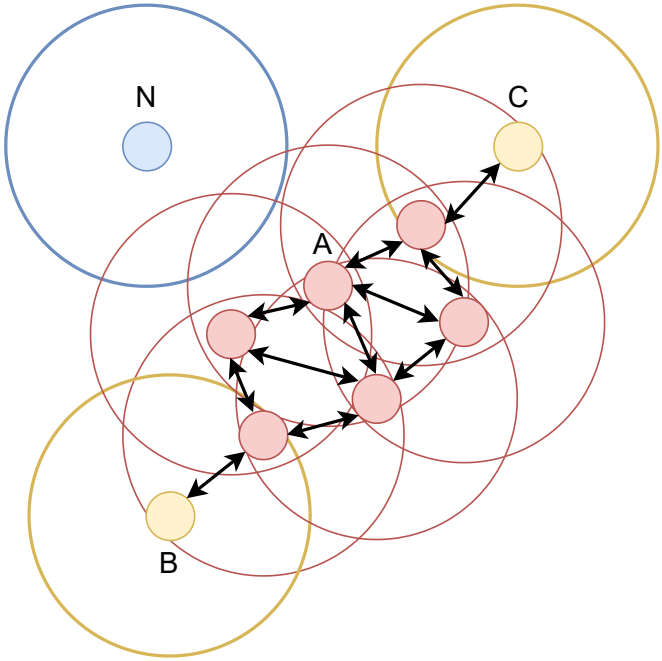

Figure 3.10: The DBSCAN cluster model inspired by [Est+96]. The dots symbolise the points, and the circles around them display the range ε of each point. N is not a density-reachable noise point. B and C are density-reachable by the cluster around the core point A. Arrows indicate direct density accessibility.

Figure 3.10 illustrates the concepts of DBSCAN with minPts $= 4$ and ε represented by circles. Points B and C are density-connected because both are density accessible from A. Point N is not density accessible and is therefore considered noise.

DBSCAN clusters data by linearly scanning the database for unprocessed objects. Non-core points are classified as noise. When a core point is identified, its neighbours are iteratively expanded and are part of the cluster. Objects assigned to a cluster are skipped in subsequent scans. Only points initially labelled as undefined are subjected to neighbourhood queries. If a query is performed, the point is subsequently labelled as either clustered or noise. A point's label is only changed from noise to a cluster label if it is included in a cluster during the expansion process [Est+96] [Sch+17].

The algorithm achieves reasonable efficiency by ensuring that each point is processed only once. The runtime complexity is $O\left(n \cdot Q + \sum_i r_i\right)$, where Q is the complexity of the 'Range-Query' function and r_i is the result size of the i-th query [Est+96]. DBSCAN is not restricted

to Euclidean distances [Est+96]. Appropriate distance measures can be applied to various data types, including geographic data and polygons, as in [Sch+17] and [Xu+07].

Although the order of cluster discovery can affect cluster labels, the DBSCAN algorithm produces deterministic results. Border points, which may be densely accessible from more than one cluster, are assigned to the first cluster from which they are accessible. This design choice simplifies cluster assignment and minimises memory usage [Est+96]. The DBSCAN algorithm is a robust and versatile clustering method. It is capable of identifying clusters of arbitrary shape while effectively dealing with noise.

Analyzing Denoising Techniques for Different Noise Types

Sonar images are affected by noise from various sources, resulting in complex noise behaviour [BPC18] [Kuc07]. This noise significantly impacts the performance of the EAE-Net and, consequently, the reconstruction process of the PCTMA-Net. Section 3.1.3 provides a detailed examination of the sources and effects of this noise for sonar images. This noise in the sonar images has a direct impact on the point clouds generated by the EAE-Net, as artefacts, geometric distortions, speckle noise, and random noise play a role in the elevation estimation [Wan+23]. The following Table 3.1 summarizes the noise sources for the sonar images and point clouds, along with effective denoising methods for each type.

Hypergraph-based methods: These methods are particularly effective at removing geometric distortions and artefacts caused by inaccuracies in the imaging process. Hypergraph methods capture complex relationships within the data, allowing distorted geometries to be corrected. These methods help to understand and mitigate the noise patterns inherent in the systematic errors of the conversion process. While they may alter a significant number of points to correct the overall shape and structure, they aim to preserve as much of the integrity of the original data as possible [ZCD21].

Multi-normal guided methods: These methods are designed to smooth out noise while preserving essential structure and edges. They slightly adjust many points to smooth the random noise, ensuring that the point cloud's crucial features and geometric details remain intact. This method is effective in reducing background noise that lacks a specific pattern [Han+18] [Zhe+18].

DBSCAN: This efficient clustering algorithm identifies and removes systematic errors by grouping points into clusters and discarding those that do not belong to any cluster (i.e. noise). DBSCAN tends to remove isolated points while preserving the overall structure, thus primarily affecting points identified as noise [Sch+17].

Overall , Hypergraph-based techniques may adjust a significant number of points to correct overall geometric distortions [ZCD21], while others, like DBSCAN, focus on selectively removing or adjusting points identified as noise [Sch+17]. Smoothing filters and multi-normal guided methods make more subtle adjustments to many points, aiming to smooth the noise while preserving essential features and structures [Zhe+18].

Noise	Cause	Denoising Strategies	Method
Geometric Distortions and Artefacts	Inaccuracies in the imaging process lead to distorted shapes and structures in the point cloud (Section 3.1.3).	Hypergraph methods capture complex relationships and correct distorted geometries [ZCD21].	Hypergraph
Systematic Noise from 2D to 3D (EAE-Net)	Systematic errors during sonar imaging or 2D to 3D conversion [Wan+23].	Hypergraph-based methods help understand noise patterns [ZCD21]. DBSCAN eliminates systematic errors of lower density [Sch+17].	Hypergraph / DBSCAN
Speckle Noise	Interference of sound waves reflecting from objects, leading to contrast variations (Section 3.1.3).	Smoothing filters and statistical outlier removal. The multi-normal guided normal filter preserves edges and large structures [Zhe+18].	Multi-normal-guided method
Random Noise	Background noise without a specific pattern, possibly from environmental factors or random errors [Che+17].	Multi-normal guided methods leverage normal vectors to preserve geometric details while smoothing noise [Han+18].	Multi-normal-guided method
Outliers	Erroneous points are significantly distant from the actual surface, possibly due to reflections or transient objects. Misinterpretations by the EAE-Net can lead to outliers, especially with increasing distance to the sensor [Wan+23].	Outlier detection methods like DBSCAN for clustering or statistical approaches [Sch+17].	DBSCAN

Table 3.1: Sources of Noise and Corresponding Denoising Strategies

3.2.3 Point Cloud Transformer with Morphing Atlas-based Point Generation Network for Dense Point Cloud Completion (PCTMA-Net)

Overview

This Section introduces the PCTMA-Net framework by Jianjie Lin, Markus Rickert, Alexander Perzylo and Alois Knoll [Lin+21], specifically designed to tackle the complex challenge of 3D point cloud completion. This framework, which is the culmination of this thesis's pipeline, is a crucial tool in overcoming the limitations of sensor resolution, occlusion, and camera angles that often result in incomplete point clouds. The completion of these point clouds is of utmost importance in various computer vision and robotics applications, including object recognition, autonomous driving, and robotic grasping. Point cloud completion is the inference of the missing parts of a 3D shape from an incomplete point cloud. This task is particularly challenging due to the irregular nature of the point cloud and the need to capture the detailed geometric structure of the object. Traditional methods, which rely on global features, often need more structural details, resulting in a less accurate reconstruction [Lin+21].

To overcome these challenges, PCTMA-Net introduces a sophisticated architecture that combines the strengths of transformer models with a novel morphing atlas-based point generation network. The framework consists of two main components:

1. **Point Cloud Transformer Encoder:** The Transformer encoder, adapted from natural language processing, is used for semantic affinity learning within a partial point cloud. The encoder processes the point cloud to extract detailed features using the attention mechanism, which effectively captures the local context within the point cloud [Lin+21]. The attention mechanism in the transformer is permutation invariant, which makes it well suited to learning from unordered point cloud data. This mechanism updates the weights between points based on their global context, improving the network's ability to preserve structural detail.
2. **Morphing Atlas-Based Point Generation Network:** In order to predict the missing regions of the shape, the decoder component of PCTMA-Net uses the features extracted from the transformer encoder. This is achieved by a morphing atlas-based network that divides the shape's surface into multiple charts [Lin+21]. Each chart represents a local region of the manifold, and the network generates these charts to reconstruct the full 3D shape. Multiple charts allow the network to produce high-resolution and fine-grained detail.

Architecture of PCTMA-Net

The architecture of the PCTMA network, as described by Lin et al. [Lin+21], is presented below, unless otherwise specified. The overall structure of the PCTMA-Net is shown in Figure 3.11. Given a partial point cloud $\mathcal{P}$ with N_{in} points [(a) in Figure 3.11], where each point is represented by 3D coordinates $\mathbf{x} = [x_i, y_i, z_i]$, the first step is to convert this input into a feature vector $\mathbf{F}_0$ using a PointNet.
PointNet is a neural network specifically designed to directly process 3D point clouds by extracting features from the raw input data and converting the raw 3D coordinates of the partial point cloud into a structured feature vector. The network consists of three key modules:

- A max-pooling layer.
- A structure for combining local and global information.
- Two joint alignment networks that align both input points and point features.

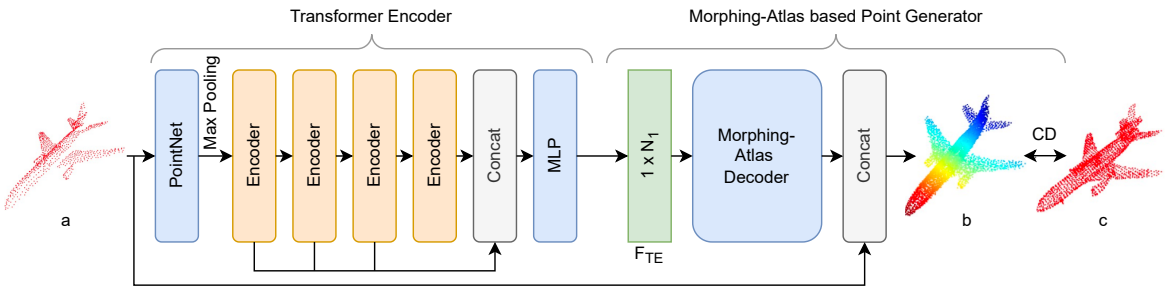

Figure 3.11: The basic model of the PCTMA-Net based on [Lin+21]. The structure consists of a Transformer encoder and the morphing-atlas-based point generation decoder. Using N x stacked encoder layer, the Transformer extracts features from the input point cloud (a), serving as input for a concatenation operation (Concat). $1xN_1$ represents this concatenated feature vector, where N_1 is the number of combined features. $\mathbf{F}_{\mathrm{TE}} = \mathrm{LBR}(\mathrm{LBR}(\mathbf{F}_{\mathrm{e}}))$. The Chamfer Distance (CD) is used to evaluate the results (b) with the ground truth (c).

The max-pooling layer acts as a symmetric function, aggregating information from all point clouds. This aggregation ensures permutation invariance, meaning that the output is independent of the order in which the points are input.

The combination of local and global information allows PointNet to capture detailed features from individual points while understanding the overall structure of the point cloud [Qi+17a]. This dual approach helps create a comprehensive feature representation of the input data. The two joint alignment networks are responsible for aligning the input points and the extracted features. The classification network in PointNet takes n points as input. The network applies transformations to both the input points and the features and then aggregates the point features using max-pooling. This process results in classification scores for k classes. The segmentation network extends this by combining global and local features to output scores for each point, indicating its classification within the 3D shape [Qi+17a].

The PointNet derived feature vector $\mathbf{F}_0$ is then processed by the transformer encoder. The encoder learns a detailed and discriminative representation of the input using an attention mechanism to capture the local context within the point cloud. This process will be described in more detail in Section 3.2.3. This step enhances the network's ability to understand and exploit the structural details of the point cloud, unlike traditional methods that rely solely on global features.
Through the layers of the transform encoder, the network learns semantic relationships within the partial clouds. This allows the network to infer correlations and structures that are critical to accurately completing the shape by updating weights between points based on their global context.
The features from each encoder layer are concatenated, combining information from multiple levels of the network. This means they are combined into a single representation, integrating information from multiple levels of the network to enhance the overall feature set. The concatenated features are processed by a multi-layer perceptron (MLP), which further refines the feature representation. The output from the MLP is a feature vector with dimensions $1 \times N_1$, representing the combined and refined features. The feature vector serves as the input to the morphing atlas-based point generation network, which is explained in Section 3.2.3. This decoder component divides the surface of the shape into multiple maps, each representing a local region of the manifold. By generating and combining these maps, the network reconstructs the full 3D shape. This approach ensures high resolution and fine-grained detail in the final shape by allowing the model to focus on smaller, more manageable Sections of the

surface. The generated points are concatenated with the existing input points (a) to form a complete point cloud (b). This reconstructed shape retains the detailed topology and structure inferred by the mesh, resulting in a high-fidelity representation.

The Chamfer Distance (CD) is used as a quantitative evaluation metric because of its efficient computation compared to the earth mover's distance. Chamfer Distance measures the mean distance between each point in one point cloud and its nearest neighbour in another point cloud.
Given a ground truth point cloud $S_G = [x_i, y_i, z_i]_i^{n_G}$ and a reconstructed point cloud $S_R = [x_i, y_i, z_i]_i^{n_R}$, where n_G and n_R are the number of points in S_G and S_R, respectively, the Chamfer Distance d_{CD} is calculated as [FSG17]:

$$d_{\text{CD}} = \frac{1}{n_R} \sum_{x \in S_R} \min_{y \in S_G} \|x - y\|^2 + \frac{1}{n_G} \sum_{y \in S_G} \min_{x \in S_R} \|y - x\|^2 \tag{3.20}$$

In this formula:
- S_G: The ground truth point cloud, consisting of n_G points.
- S_R: The reconstructed point cloud, consisting of n_R points.
- $\min_{y \in S_G} \|x - y\|^2$: The minimum squared Euclidean distance ($L2$ norm) from point x in S_R to any point y in S_G.
- $\min_{x \in S_R} \|y - x\|^2$: The minimum squared Euclidean distance ($L2$ norm) from point y in S_G to any point x in S_R.

The Chamfer Distance is the sum of two terms: the mean distance from each point in the reconstructed point cloud to its nearest neighbour in the ground truth point cloud and the mean distance from each point in the ground truth point cloud to its nearest neighbour in the reconstructed point cloud. This metric ensures that both point clouds are close to each other in terms of their geometric shape [FSG17].

Transformer Encoder

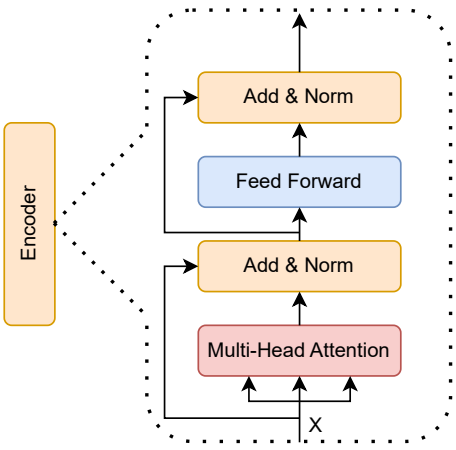

Figure 3.12: The structure of the Transformer Encoder based on [Lin+21]. The Multi-Head Attention focuses on different parts of the input X, enhancing feature representation. The Add & Norm layers add residual connections and normalize the data for stability. The Feed Forward layer further refines features, capturing complex patterns.

The Transformer encoder in PCTMA-Net is crucial in transforming an incomplete point cloud into a feature space, enabling the network to learn detailed and discriminative representations for shape completion [Lin+21]. The extracted feature $\mathbf{F}_0$ is processed through $N\times$

stacked encoder layers, each designed to learn a discriminative representation for each point in the point cloud [Lin+21]. The process is displayed in Figure 3.12.

$$\mathbf{F}_i = \mathrm{F}_{\mathrm{encoder}_i}(\mathbf{F}_{i-1}), \quad i = [1, \ldots, N]. \tag{3.21}$$

$$\mathrm{F}_{\mathrm{encoder}_i}(\mathbf{F}_{i-1}) = \mathrm{FFN}_i(\,\mathrm{attention}_i\,(\mathbf{F}_{i-1})), \tag{3.22}$$

Each encoder layer $\mathrm{F}_{\mathrm{encoder}_i}$ contains two sublayers: a Multi-Head-Attention mechanism $\mathrm{attention}_i(\mathbf{F}_{i-1})$ and a Feed Forward Network (FFN_i) as in 3.22. A FeedForward Network (FFN) is a type of artificial neural network where information flows in one direction, from the input layer X, through one or more hidden layers H, to the output layer Y, without any feedback loops or cycles. The dimension of the hidden layers, denoted as d_{hidden}, determines the number of neurons in each layer, which directly influences the network's capacity to approximate complex functions $f : X \to Y$. The FFN is defined by the equation [Gev+20]:

$$\mathrm{FF}(\mathbf{x}) = f\left(\mathbf{x} \cdot K^{\top}\right) \cdot V \tag{3.23}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input vector, $K \in \mathbb{R}^{d_m \times d}$ and $V \in \mathbb{R}^{d_m \times d}$ are parameter matrices, d is the input dimension, d_m is the intermediate dimension, and f is a non-linearity such as ReLU. The dimension of a FeedForward Network refers to the size of its hidden layers. Adjusting this dimension can significantly affect the network's performance and behaviour. Increasing the number of neurons enhances the network's ability to learn complex functions and capture intricate patterns in the data, as it provides more parameters (weights and biases) for the network to adjust during training [Gev+20]. The Multi-Head-attention mechanism computes attention scores to focus on different parts of the input X and is further explained in the next Section 3.2.3. The FFN processes the features to capture complex patterns and is defined as [Lin+21]:

$$\mathrm{FFN}_i(\mathbf{x}) = \mathrm{LBR}_{i,1}(\mathrm{LBR}_{i,0}(\mathbf{x})) + \mathbf{x}, \tag{3.24}$$

where $\mathrm{LBR}_{i,0}$ and $\mathrm{LBR}_{i,1}$ represent Linear, BatchNorm, and ReLU layers. This setup ensures that the network can efficiently capture and preserve important features from the input data. After passing through the encoder layers, the features from each layer are concatenated to form a comprehensive global feature, as visible in Figure 3.11. This concatenated feature is then processed by two cascaded LBR layers to produce an effective global feature representation [Lin+21]:

$$\mathbf{F}_{\mathrm{e}} = \mathrm{BatchNorm}(\mathbf{F}_i \oplus \cdots \oplus \mathbf{F}_N) \tag{3.25}$$

$$\mathbf{F}_{\mathrm{TE}} = \mathrm{LBR}(\mathrm{LBR}(\mathbf{F}_{\mathrm{e}})) \tag{3.26}$$

Here, $\mathbf{F}_i \in \mathcal{R}^{d_{\mathrm{model}}}$, $\mathbf{F}_e \in \mathcal{R}^{N \times d_{\mathrm{model}}}$, and $\mathbf{F}_{\mathrm{TE}} \in \mathcal{R}^{d_{\mathrm{model}}}$. The operator $\oplus$ denotes concatenation, visible in the notation Table 1. The dimension d_{model} is a hyperparameter that defines the d_{model} -dimensional embedding feature $\mathbf{F}_0 \in \mathcal{R}^{d_{\mathrm{model}}}$ and, therefore, the feature description [Lin+21]. The En_{channels} and De_{channels} number is directly linked to the d_{model} -dimension and must therefore be adjusted simultaneously.

Multi-Head-Attention Mechanism

Self-attention is a mechanism that computes semantic relationships between elements within a data sequence. In point cloud processing, attention is used to build weights between

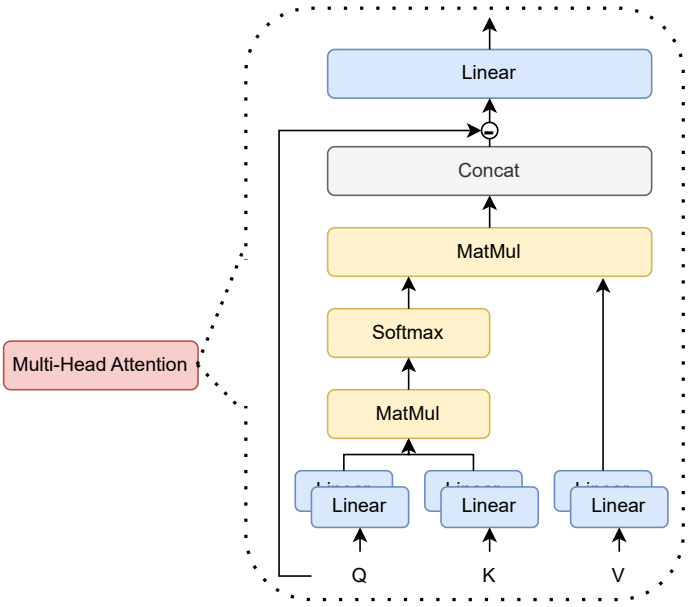

Figure 3.13: The structure of the Multi-Head Attention Mechanism based on [Lin+21] and [Vas+17]. Q representing queries for attention scores. K representing keys for comparing with queries. V contains the values to be aggregated, weighted by the attention scores. The Linear module is used to transform the input features into the query, key, and value vectors.

any two positions in feature space. The attention mechanism's permutation invariant property makes it suitable for disordered, irregular data representations such as point clouds [Lin+21]. The structure of the implemented Multi-Head-Attention mechanism is illustrated in Figure 3.13.
Given a partial point cloud, the input features are first transformed into a query ($\mathbf{Q}$), key ($\mathbf{K}$) and value ($\mathbf{V}$) vectors using the linear modules [Lin+21]:

$$\hat{\mathbf{Q}} = \text{Linear}(\mathbf{Q}), \quad \hat{\mathbf{K}} = \text{Linear}(\mathbf{K}), \quad \hat{\mathbf{V}} = \text{Linear}(\mathbf{V}) \tag{3.27}$$

Here, the Linear module applies a linear transformation to the input features. The attention scores are computed by performing a matrix multiplication (MatMul in Figure 3.13) between the query matrix $\mathbf{Q}$ and the transpose of the key matrix $\mathbf{K}$, followed by a softmax normalization:

$$\mathbf{A} = \text{softmax}\left(\frac{\hat{\mathbf{Q}}\hat{\mathbf{K}}^T}{\sqrt{d_k}}\right) \tag{3.28}$$

where d_k is the dimensionality of the keys [Vas+17]. This step results in an attention score matrix $\mathbf{A}$, which indicates the similarity between queries and keys.
The attention scores are then, in the next MatMul module, used to compute a weighted sum of the value vectors [Lin+21]:

$$\mathbf{F}_{\text{head}_i} = \mathbf{A}\hat{\mathbf{V}} \tag{3.29}$$

Each element in the output is a weighted sum of the values, where the attention scores give the weights.
Multiple sets of $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ (heads) are used to capture different aspects of the data. This number of heads can be used as a hyperparameter. The outputs of these heads are concatenated and linearly transformed [Lin+21]:

$$\mathbf{F}_{\text{sa}} = \text{Linear}\left(\mathbf{F}_{\text{head}_1} \oplus \cdots \oplus \mathbf{F}_{\text{head}_h}\right) \tag{3.30}$$

This allows the model to jointly attend to information from different representational subspaces at different locations [Vas+17]. Therefore, the attention mechanism incorporates the idea of a Laplacian matrix $L = D - E$, where E is the adjacency matrix and D is the diagonal matrix. The attention mechanism, based on this Laplacian matrix, is structured as follows [Lin+21]:

$$\begin{aligned} \mathbf{F}_{\text{sa,out}} &= \text{attention}\left(\mathbf{F}_{\text{sa,in}}\right) \\ &= \text{LBR}\left(\mathbf{F}_{\text{sa,in}} - \mathbf{F}_{\text{sa}}\right) + \mathbf{F}_{\text{in}} \end{aligned} \tag{3.31}$$

The formula 3.31 describes how the output of the self-attention mechanism is computed. The term $\mathbf{F}_{\text{sa,in}} - \mathbf{F}_{\text{sa}}$ represents the difference between the input features and the self-attention features, which is then passed through an LBR layer. The result is added to the original input features $\mathbf{F}_{\text{in}}$ to produce the final output.

Morphing-Atlas-Based Point Generator Network

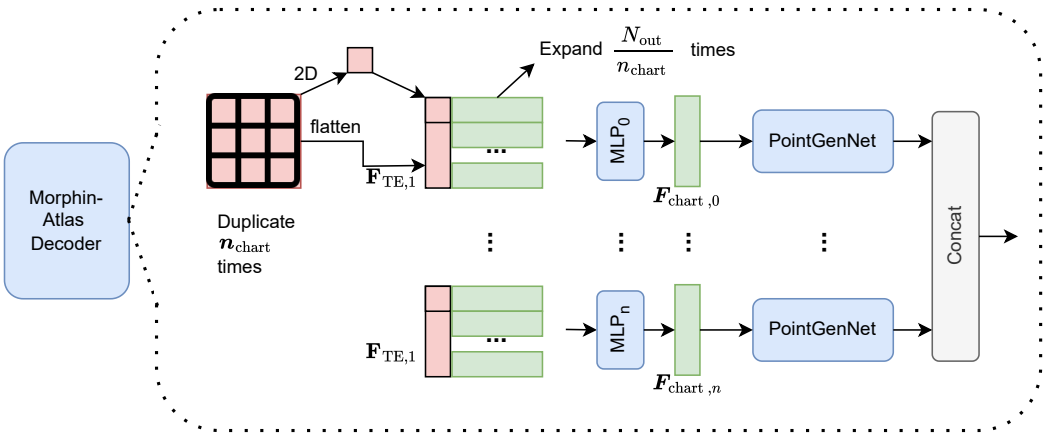

Figure 3.14: The structure of the Morphing-Atlas Decoder based on [Lin+21]. The decoder generates point clouds by morphing an atlas of point patches into the target shape. This involves transforming the features extracted by the encoder into 3D coordinates and refining the output through multiple layers to capture detailed geometric structures.

The extracted features $\mathbf{F}_{\text{TE}}$ of the Transformer encoder are fed into a morphing atlas-based point generator that predicts continuous and smooth 3D shapes.
The concept of an atlas comes from topology, where it is used to describe a manifold. An atlas is composed of multiple charts, each representing a local region of the manifold [Lin+21].

A subset $\mathcal{S}$ of $\mathbb{R}^3$ is considered a 2-manifold if, for each point $\mathbf{p} \in \mathcal{S}$, there exist open sets U in $\mathbb{R}^2$ and W in $\mathbb{R}^3$ containing $\mathbf{p}$ such that $\mathcal{S} \cap W$ is homeomorphic to U. The collection of homeomorphisms from $\mathcal{S} \cap W$ to U is known as a chart, and its inverse is termed a parameterization [Gro+18]. An atlas is a complete set of charts whose images collectively cover the 2-manifold. We consider a local parameterization of a 2-manifold, specifically a 2-manifold $\mathcal{S}$, a point $\mathbf{p} \in \mathcal{S}$, and a parameterization φ of $\mathcal{S}$ in a local neighborhood of $\mathbf{p}$. This parameterization is defined on the open unit square $]0,1[^2$ by first restricting φ to an open neighbourhood of $\varphi^{-1}(\mathbf{p})$ with the disc topology and then mapping this neighbourhood onto the unit square [Gro+18]. The problem of learning to generate the local 2-manifold involves finding parameterizations $\varphi_\theta(\mathbf{x})$ with parameters θ that map the open 2D unit square $]0,1[^2$ to a good approximation of the desired 2-manifold $\mathcal{S}_{\text{loc}}$. Specifically, we seek parameters θ that minimize the objective function:

$$\min_{\theta} \mathcal{L}(\mathcal{S}_\theta, \mathcal{S}_{\text{loc}}) + \lambda \mathcal{R}(\theta) \tag{3.32}$$

where $\mathcal{L}$ is a loss over 2 -manifolds, $\mathcal{R}$ is a regularization function over the parameters θ, and λ is a scalar weight [Gro+18].

Several such charts are needed to generate a complete surface. Specifically, N trainable parameterisations ϕ_{θ_i} for $i \in \{1,\ldots,N\}$ are used [Gro+18]. For 3D shapes, the manifold can be thought of as the shaped surface, and a complete 3D shape can be represented by combining all these charts. This process involves using multiple learnable charts $\mathcal{C}_i$, each represented by a Multi-Layer Perceptron (MLP) with ReLU activations, denoted φ_θ with parameters θ. Each MLP, a designed decoder $\mathcal{D}_i$, maps points in $\mathbb{R}^2$ (a 2D plane) to points in $\mathbb{R}^3$ (the 3D surface) [Lin+21]. This process is shown in Figure 3.14 on the left side.

To train these MLP parameters θ_i, it is necessary to measure the distance between the generated and target surfaces and to include the shape feature $\mathbf{F}_{\text{TE}}$ in the MLP. Points can be sampled from the target surface regardless of the available representation. Let $\mathcal{A}$ be a set of points sampled from the unit square $[0,1]^2$ (the 2D plane) and $\mathcal{S}^\star$ be a set of points sampled from the target surface. The shape feature $\mathbf{F}_{\text{TE}}$ is incorporated by concatenating it with the sampled point coordinates $\mathbf{p} \in \mathcal{A}$ before passing it as input to the MLPs. This means that each input to an MLP consists of both the 2D coordinates from $\mathcal{A}$ and the shape feature $\mathbf{F}_{\text{TE}}$. MLPs are not explicitly constrained to encode different regions of space, but their collective output should cover the entire shape. They learn to specialise in different regions without explicit bias, although they depend on their random initialisation, similar to convolutional filter weights. The training objective is to minimise the Chamfer loss between the set of generated 3D points and the target surface points $\mathcal{S}^\star$ [Gro+18] [Lin+21].

A hyperparameter $n_{\text{primitives}}$ is introduced to control the number of charts defined for a shape, ensuring the prediction of a smooth and high-resolution shape. The global function $\mathbf{F}_{\text{TE}} \in \mathcal{R}^{d_{\text{model}}}$ is duplicated $\frac{N_{\text{out}}}{n_{\text{cllatt}}}$ times and concatenated with a mesh grid, resulting in a linear projection of $\mathbf{F}_{\text{TE},1}$ with different learned linear projections (see Figure 3.14). This approach is similar to multi-head attention, allowing the model to capture shape features from different representational subspaces at different locations. Consequently, $\mathbf{F}_{\text{TE},1}$ is duplicated n_{chart} times, and each $\mathbf{F}_{\text{TE},1}$ is fed into a MLP layer. This MLP layer produces a new hidden code called $\mathbf{F}_{\text{chart},i} \in \mathcal{R}^{(d_{\text{motel}}+2)\times\left(\frac{N_{\text{curt}}}{n_{\text{chart}}}\right)}, i \in [1,\ldots,n_{\text{chart}}]$. For each chart, $\mathbf{F}_{\text{chart},i}$ is entered into a PointGenNetwork, which has the same structure as [Gro+18]. All the charts are then concatenated to form the complete 3D shape [Lin+21]. This step is essential to generate a coherent point cloud.

This theoretical part of the work has explained the background of noise in sonar images and its effects. This was followed by an overview of the EAE-Net and how it is used and functions. The output of the EAE-Net consists of ground truth and elevation-estimation point clouds. The estimated point clouds are also affected by sonar and structural noise. For this reason, various denoising methods were presented, compared and assigned to the noise types of the point clouds. Finally, the architecture and functionality of the PCTMA-Net was explained. Special attention was paid to the explanation of hyperparameters within the Transformer encoder and the Morphing-Atlas-based point generator network. The topics discussed here will serve as the basis for the implemented pipeline in the next Chapter 4, the methodology.

Chapter 4

Methodology

This chapter details the approach taken to adapt the PCTMA-Net for completing point clouds derived from 2D sonar images. The methodology focuses on the pipeline, experimental setup, and specific adaptations to handle sonar noise and data irregularities. The approach aims to investigate how effectively PCTMA-Net can adapt to and process the additional complexity introduced by sonar noise, interpolate missing regions, and enhance 3D reconstruction accuracy.

4.1 Pipeline

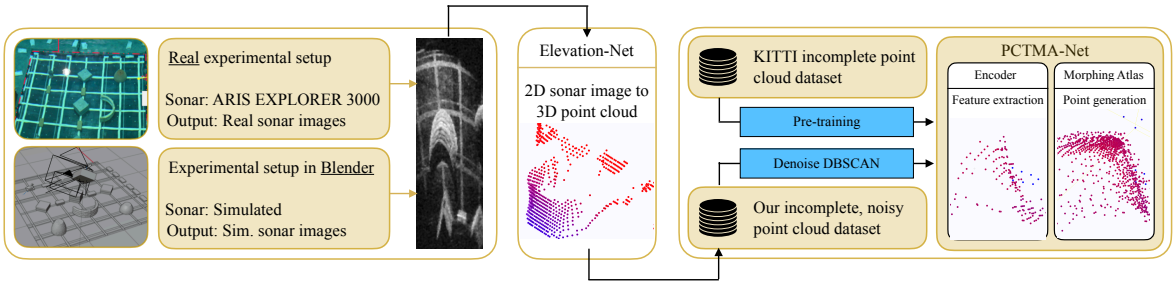

Figure 4.1: This by [Kne+] pipeline shows the real and the Blender dataset generation, the transformation of 2D sonar images into 3D point clouds using the EAE-Net, the denoising of point clouds, and their completion and refinement using the PCTMA-Net.

As can be seen in Figure 4.1, the datasets are generated from two different sources. From the real experimental setup (explained in Section 4.2) as well as from the simulated setup in Blender. A total of seven additional datasets are added to the existing two datasets (1615 samples [real] and 1615 samples [synthetic] by [Wan+23]), resulting in nine datasets comprising 14,535 samples with various objects and noise levels. The EAE-Net generates 3D point clouds from the 2D sonar images (see Section 4.3). To reduce noise at the edges of objects and eliminate general outliers of the point clouds, density-based spatial clustering of applications with noise (DBSCAN) is applied (section 4.4). PCTMA-Net is then employed to reconstruct, refine, and denoise the generated point clouds, deeper explained in Section 4.5. Initially, a pre-training phase is conducted using the Completion3D dataset from ShapeNet, which consists of 28,000 incomplete point clouds to establish pre-trained weights [Cha+15]. The Completion3D contains incomplete but noise-free samples of single object point clouds. Following this, PCTMA-Net is trained, starting with these pre-trained weights, using the previously generated noisy point clouds. Since the original PCTMA-Net is not designed for noisy

point clouds, hyperparameters are adjusted to handle the added complexity [Lin+21]. Ultimately, we obtain completed, refined, and reconstructed 3D point clouds (see Section 5.3).

4.2 Experimental Setup

The sonar images utilized as input for the EAE-Net are obtained through both experimental and synthetic setups. The following sections provide a detailed explanation of the experimental environments in which these images are recorded, as well as a comprehensive overview of the structure of all the datasets generated from these setups.

4.2.1 Real-World Data Acquisition

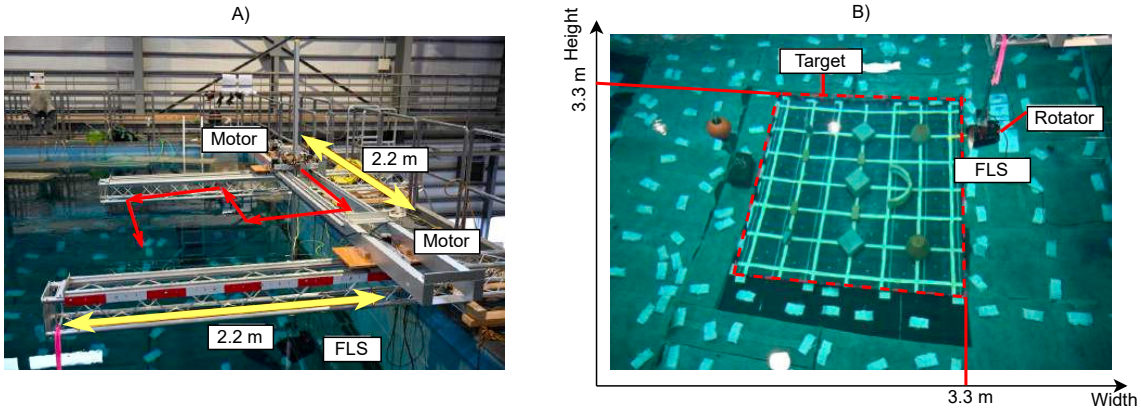

Figure 4.2: These photographs by [Wan+23] show the experimental setup to capture sonar images. (A) The structure which moves the sensor along the red-marked line. (B) The target area at the bottom of the water tank with 18 objects. The sonar sensor (FLS) is mounted on the rotator.

The real sonar image dataset is built in a large water tank, by the Wakachiku Const. Co. Ltd. Tokyo, Japan, to capture the real sonar images within the work of Wang et al. [Wan+23]. 18 objects are placed within a target area measuring 3.3m by 3.3m. The underwater experimental setup can be seen in Figure 4.2[B]. These objects include eight rectangular blocks, two additional stacked rectangular blocks, a rabbit, a duck, two quadratic blocks, a cube, a round wall, and two vases. The ARIS EXPLORER 3000 is used in 3.0 MHz mode. The forward-looking sonar (FLS) is moved by a positioning device and carried to 51 positions, as shown in Figure 4.2[A]. At each position, an AR2 rotator generates a ω_x motion that rotates the FLS along the x-axis from $-35°$ to $35°$ [Wan+23]. These basic motions are, according to the motion model in Section 3.2.1, sensitive to the elevation angle φ. Images from 33 positions are used to train the network, 9 positions for validation and 9 positions for testing. This setup provides 1,060 images for training, 272 images for validation, and 283 for testing. Triplets are used to train the network. The roll angle for the target image is denoted as $\alpha°$, while the roll angles for the two source images are approximately $(\alpha-10)°$ and $(\alpha+10)°$, respectively [Wan+23]. The target scene is shown in Figure 4.2[B]. The 3D ground truth of the objects in the scene is a measurement in a terrestrial environment [Wan+22c]. These objects are subsequently arranged in the water tank according to a predetermined configuration. This controlled setup enables the acquisition of a CAD model of the scene, which is crucial for the synthetic experimental setup, which will be elaborated upon in the following section. The

determination of the relative pose between the scene and the FLS is essential. The initial pose is measured manually by a diver with the aid of a ruler, and the accurate result is obtained by comparing the synthetic image with the real image in the edge domain [Wan+22a]. Once a pose is accurately captured, the remaining poses are calculated using the control input for the positioning device and rotator.

4.2.2 Synthetic Data Generation

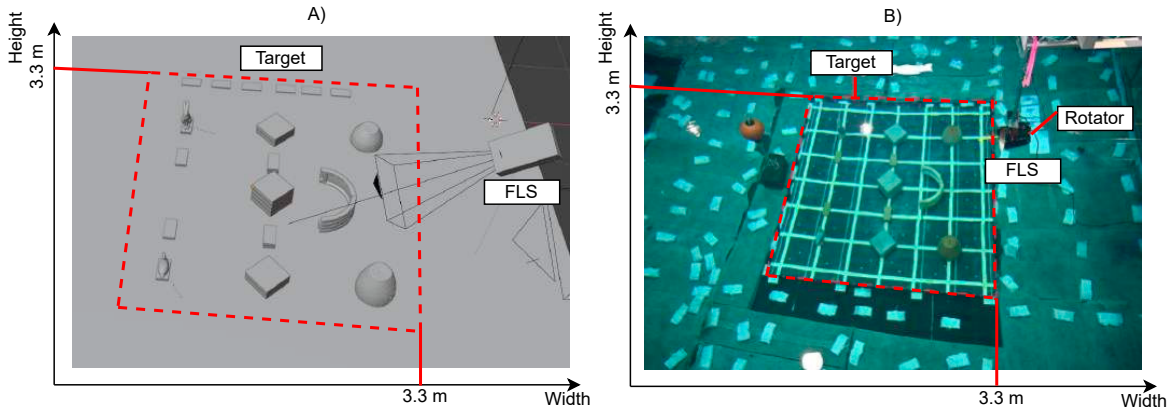

Figure 4.3: (A) The accurately recreated scene in Blender with a synthetic sonar sensor. (B) Photograph by [Wan+23] shows the experimental setup to capture real sonar images. The target area at the bottom of the water tank with 18 objects. The sonar sensor (FLS) is mounted on the rotator.

Since the configuration of the real-world experiment is known, the CAD model can be transferred directly into Blender (see Figure 4.3[A]). The Blender script automates the generation of 2D sonar images using Blender. Initially written by Yusheng Wang [Wan+23], later modified for this work. The process starts by configuring the Blender scene to render depth images, greyscale images and normal maps, which are linked to output nodes. The scene is then rendered, generating the necessary images.

The script applies a Diffuse Bidirectional Scattering Distribution Function (BSDF) shader to the objects in the scene to accurately simulate the sonar reflections. This shader is essential for generating accurate sonar data and helps to create realistic surface reflections. The Diffuse BSDF shader provides a realistic simulation of sonar reflections off different materials by modelling how light (or sonar waves) diffuses across surfaces [Ble24]. The script also automates the camera movement within the Blender scene. To simulate the sonar scanning process, it moves the camera to predefined positions and orientations. The camera's position and orientation are incrementally adjusted to capture images from different angles and distances. Specifically, the camera performs the following movements:

- The camera starts from an initial position and is rotated along the x-axis from -35° to 35° employing an AR2 rotator.
- It is moved horizontally and vertically within the scene to simulate different scan angles and distances.

By rendering the scene from different positions, the script generates a series of images representing the sonar's perspective at different FOV. These images are then processed to extract relevant data such as depth information and normal vectors. Depth images,

normal maps and greyscale images are generated at a resolution of 1200 x 720 pixels. Once rendering is complete, the script processes the rendered images. The grayscale images are normalised and scaled, while the depth images are processed to convert distance values into depth values. The normal maps are processed to extract normal vectors. The script resizes the images as necessary and applies smoothing to correct any aliasing problems. It also generates fan-shaped images from the processed data to represent the sonar scan results more accurately. The generated output can be summarized as:

- **Depth images:**
  - Represents the distance from the sonar sensor to surfaces in the scene.
  - Depth values are calculated based on the distance from the camera and converted to depth.
  - Values are normalised to a scale defined by the maximum and minimum ranges.
- **Normal maps:**
  - Represents the orientation of surfaces in the scene.
  - Normal vectors are extracted for each pixel, indicating the direction of the surface.
  - Values are normalised to ensure consistent intensity across the image.
- **Greyscale images:**
  - Represents the intensity of sonar reflections.
  - Grayscale values are normalised and scaled to provide a consistent intensity display.
  - Used to represent the sonar's perspective of the scene, showing variations in reflection intensity.
  - The greyscale images are initially generated in Cartesian coordinates, but the script generates fan-shaped (polar coordinate) representations to sufficiently simulate the sonar scan results.
- **Motion data:**
  - Documented translation and rotation between each frame.

Seven additional datasets, based on the original one, were generated using Blender to increase the sample size (Figure 4.4). These datasets exhibit several differences:

- The original PCTMA-Net is designed to reconstruct single-object and noise-free point clouds. In our scenario, we have up to 18 different objects lying on the bottom of a water tank. The floor acts as a connecting element to create a coherent point cloud of the 18 objects and the floor. Four corresponding datasets without the bottom were generated to test the model's ability to handle multiple free-floating objects (such as fish). The objects, camera movement and rotation, and positioning of the objects remain consistent with those in the corresponding datasets, including the ground.
- The level of complexity of the datasets also varies. Firstly, the newly generated datasets differ from the original ones regarding object positioning to create new scenarios. Secondly, different levels of complexity are introduced to test the model's ability to reconstruct under various levels of complexity, which may produce different results. Complexity is varied by changing the number of objects, their spatial distribution, and the presence of occlusions or overlapping objects. This helps to assess how well the model can reconstruct scenes of increasing complexity.

By incorporating these variations, we aim to thoroughly evaluate the PCTMA network's robustness and adaptability under different realistic conditions. This approach not only tests the model's performance in ideal scenarios but also in challenging situations that

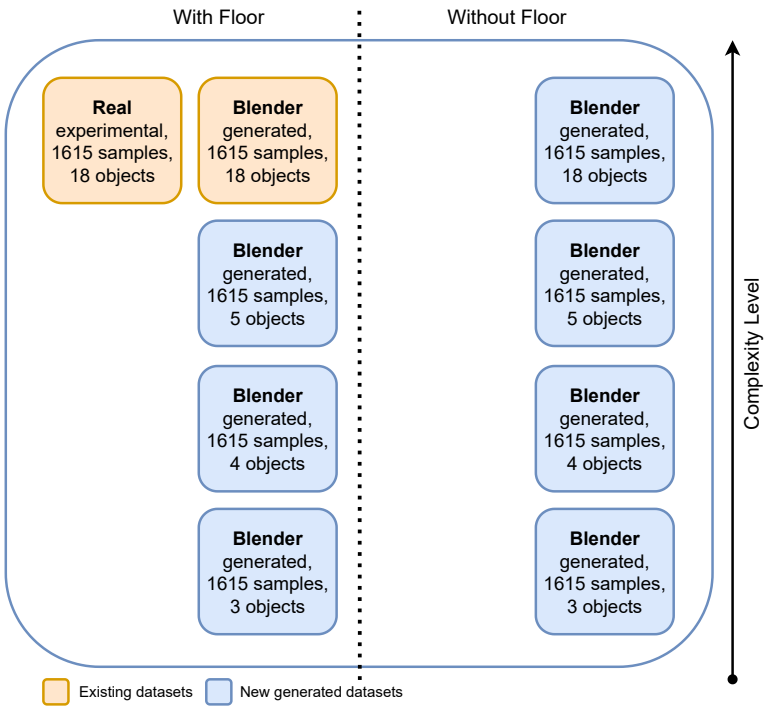

Figure 4.4: Two already existing datasets (orange) and seven newly generated datasets (blue). Separated into with and without floor and sorted by rising complexity level.

better reflect real-world applications. To achieve this, seven distinct Blender scenes were constructed across the categories mentioned in Figure 4.4. The pivotal role of the Blender scripts in generating the datasets is underscored by their ability to alter the visibility of objects. A dedicated section of the script is responsible for selectively displaying different groups of objects during the dataset generation process, thereby ensuring a diverse range of object combinations and scenarios.

Object Groups: The objects are categorized into predefined groups, with each group comprising one or more objects. These groups are then amalgamated into unique triples (or larger sets, depending on the dataset) to guarantee that each set of visible objects is distinct.

As shown in algorithm 1, for each iteration, the script:

- hides all the objects by unlinking them from the set of visible objects and linking them to the set of hidden objects (Alg. 1 line 7).
- chooses a new combination of object groups and makes the objects in these groups visible by linking them back to the visible collection and de-linking them from the hidden collection (Alg. 1 line 10).
- updates the camera position and rotation based on predefined lists, renders the scene, and saves the resulting images and associated data to specified directories (Alg. 1 line 13).

This process ensures that each dataset contains a unique combination of objects and camera views, facilitating comprehensive training and testing scenarios. All datasets, whether they contain three, four, or five objects, have the same functionality. The differences lie in the size of the visible groups and the objects' specific positions, which are adjusted to create the desired variety in each dataset.

Algorithm 1 Manage Object Visibility in Blender

Require: List of object groups, all objects, and collections.
Ensure: Correct visibility of objects based on group triples.

```
Initialize object groups and flatten to all objects.
Generate unique triples from object groups.
for each id in id_mid_lists do
    for each count in id_mid_lists[id] do
        Determine the current triple and flatten it to current groups.
        for each object in all objects do
            Hide object if it is in the visible collection.
        end for
        for each object in current groups do
            Show object if it is not in the visible collection.
            : Ensure the object is not in the hidden collection.
        end for
        Reposition and rotate the camera based on the current id and count.
    end for
end for
```

These datasets form the basis of this research and serve as the primary input to the elevation angle estimation network. Two original sets are added and seven new datasets are generated, totalling 14,535 samples. These datasets were designed to contain different noise levels due to the varying number of objects present in each sample. This variability ensures that the model can handle a wide range of real-world scenarios of varying complexity. The no-floor datasets consist of 6,460 samples, while the floor datasets consist of 8,075 samples. This distinction allows for the evaluation of the model's performance in different environmental contexts and ensures that it can accurately estimate elevation angles in both simple and complex scenes. The collected samples play a pivotal role in the training and testing of the elevation estimation network and, subsequently, the PCTMA-Net. They offer a rich and diverse dataset, enabling the model to learn and accurately predict elevation angles in a multitude of scenarios.

4.3 Elevation Angle Estimation Network Implementation

The EAE-Net (Elevation Angle Estimation Network) converts 2D sonar images into 3D point clouds by estimating the elevation angle. The theoretical analysis of Wang et al.'s [Wan+23] is discussed in Section 4.3. Several updates and changes have been implemented to improve the performance and robustness of the model and to use the network for the further pipeline. These changes address the need for a comprehensive evaluation of the model across multiple metrics and improve its overall accuracy.
Firstly, all new Blender-generated datasets were trained and tested on the original EAE-Net to ensure functionality. Afterwards, an AllData dataset was used without a separate test set to obtain the complete dataset as a point cloud output. The newly generated datasets were treated separately to better recognise and measure errors or underperformance in individual sets. AllData, therefore, merely stands for rebalancing between training, test, and validation sets. By using the full dataset without separating a test set, the model can use more data for training, improving its generalization and performance. Additionally, every sample is used

to generate a point cloud output; due to the structure of the original script, this step was needed to get as many point clouds as output as possible. Secondly, the number of training epochs was increased from 25 to 30. This change was driven by observed improvements in key performance metrics. The additional epochs led to better results in the Hausdorff distance, which improved from 0.247 to 0.198, indicating a closer match between predicted and actual point sets. In addition, the Chamfer Distance improved from 147.2 to 134.5 (the Chamfer Distance is multiplied by 10^4 to ensure comparability with the PCTMA-Net [Lin+21]), reflecting a reduction in the average distance between points in the predicted and actual sets. These values are average values from all datasets, with and without the floor. Finally, several improvements were made to the evaluation metrics. The calculation of F-score, precision, and recall for two different thresholds (0.001 and 0.003) has been added. The F-score is a harmonic mean of precision and recall, providing a balanced measure of the accuracy of the model. Precision is defined as the ratio of correctly predicted positive observations to the total predicted positives, while recall is the ratio of correctly predicted positive observations to all observations in the actual class. The formulas for precision and recall are:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP+FP} \\ \text{Recall} &= \frac{TP}{TP+FN} \end{aligned} \tag{4.1}$$

where TP is the number of true positives, FP is the number of false positives, and FN is the number of false negatives [Der16]. The F-score is calculated as:

$$F\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.2}$$

The use of multiple thresholds allows for a more nuanced assessment of performance at different levels of tolerance. In addition, the Hausdorff distance, which measures the maximum distance between points in the predicted and actual sets, has been added to the evaluation metrics [HKR93]. The Hausdorff distance is calculated as:

$$d_H(A,B) = \max\left\{ \sup_{a\in A} \inf_{b\in B} \|a-b\|, \sup_{b\in B} \inf_{a\in A} \|b-a\| \right\} \tag{4.3}$$

Here, $\|a-b\|$ denotes the distance between points a and b [HKR93]. The term sup represents the supremum (or least upper bound) of the set of distances, and inf represents the infimum (or greatest lower bound) of the set of distances. This metric ensures that the worst-case performance of the model is taken into account, providing a more rigorous measure of alignment accuracy. The total output from the EAE-Net amounts to nine datasets with 14,535 samples. A sample here consists of a ground truth and an elevation angle estimated point cloud in the `.ply` file format. The different noise levels are still subdivided into the nine datasets. The no-floor and floor datasets are also retained. Therefore, the dataset structure of Figure 4.4 remains intact. Each dataset contains a `ground truth` and a `particle` folder to store the data accordingly. A significant performance difference between the datasets could not be determined. This means that the subsequent performance of the PCTMA network can be assessed comparably across all datasets.

4.4 Denoising

Since the original PCTMA-Net was designed for single noise-free objects [Lin+21], and the output of the EAE-Net contains a high amount of noise [Wan+23], the next step was to eliminate strong outliers to minimize the distraction of the network. Objects further away from the sonar sensor tend to be very noisy and barely recognizable. However, these structures are essential reference points for later reconstruction and should be allowed. Only strong outliers resulting from misinterpretation by the EAE-Net should be removed. In particular, this refers to negative elevations that do not occur in the original experimental environment (visible in Section 5.2). These negative elevations occurred in comparatively low densities within the individual point clouds. To eliminate this effect, the use of three denoising methods was considered, which are explained theoretically in Section 3.2.2. Table 3.1 in the theoretical part of this work, shows the various sources of noise in sonar imaging, how they occur, and the denoising strategies used to deal with them. Different noise sources occur due to various factors inherent in sonar imaging (Section 3.1.3) and converting 2D to 3D point clouds using the EAE-Net. Inaccuracies in the imaging process result in geometric distortions and artefacts [Kuc07]. Errors in converting from 2D images to 3D point clouds result in systematic noise from the EAE-Net conversion process [Wan+23]. Speckle noise is caused by the interference of sound waves that reflect off objects and back to the sonar, resulting in variations in contrast (Section 3.1.3). Random noise is caused by environmental factors or random errors in data acquisition [Che+17]. Outliers are false points significantly distant from the actual surface or structure. They are often caused by reflections or transient objects and misinterpretation by the EAE network [Wan+23]. The following three denoising methods are further explained in Section 3.2.2.

- **Hypergraph methods**: Can model higher-order interactions, they are effective in capturing complex relationships and correcting distorted geometry [ZCD21].
- **DBSCAN**: Useful for identifying and eliminating outliers by clustering data points based on their density, making it effective for systematic errors and isolated outliers [Sch+17].
- **Multi-Normal Guided Methods**: Use normal vectors to preserve geometric detail while smoothing irregular noise, making them suitable for both speckle and random noise [Han+18].

To enable an effective reconstruction of the point clouds, obtaining as many points as possible in the usable regions of the point clouds and deleting interfering outliers is essential. Preserving the majority of the data is, therefore, a set goal. The output of the EAE-Net is generally very noisy and sometimes shows different point densities within a point cloud [Wan+23]. However, misinterpretations and general outliers are characterised by a conspicuously low density or a significant distance. Therefore, general denoising of the point clouds is not the goal, as many noisy points could still allow conclusions to be drawn about the objects to be reconstructed. This eliminates multi-normal guided methods, as these would smoothen the entire point cloud [Zhe+18]. Hypergraph methods might also not be ideal for excluding only real outliers without altering the rest of the point cloud because they can over-smooth data, change subtle features, and modify valid data points while modelling complex relationships [ZCD21]. The hypergraph method can also be unusable if the point clouds are too noisy, and it also involves higher computational complexity, which may be unnecessary for outlier removal [Han+18]. In contrast, DBSCAN focuses on density-based clustering, effectively identifying and eliminating outliers without significantly affecting the rest of the point cloud [Sch+17]. Unlike smoothing methods that can blur important details,

DBSCAN selectively eliminates points that fail to meet the density criteria, ensuring the quality of the reconstructed point cloud remains intact. This makes it a superior choice for addressing general outliers and systematic noise introduced by the EAE network.

The implementation of the DBSCAN algorithm is described in the following section. To work with the data according to the PCTMA-Net input file format, the samples in `.ply` format are converted into the `.h5` format. The HDF5 (Hierarchical Data Format version 5) file format is a versatile data model that allows for the storage and management of large and complex data collections [Iqb22].

Algorithm 2 Denoise Point Cloud in HDF5 Files

```
1:  function DenoisePointCloudH5(file_path, eps, min_samples)
2:      Load point cloud from HDF5 file
3:      Apply DBSCAN clustering with eps and min_samples
4:      Extract points with noise labels
5:      return denoised point cloud
6:  end function
7:  procedure ProcessDirectories(root_dir)
8:      for all HDF5 files in 'particle' directories do
9:          Call DenoisePointCloudH5 with eps=0.06 and min_samples=10
10:         Save denoised point cloud back to file
11:     end for
12: end procedure
```

The implemented denoising Script 2 consists of two main parts: a function and a procedure. The DenoisePointCloudH5 function (lines 1-6) is responsible for loading the point cloud data from an HDF5 file (line 2). It then applies DBSCAN clustering with the specified `eps` and `min_samples` parameters (line 3). The DBSCAN algorithm assigns a label to each point, indicating whether it belongs to a cluster or is considered noise. The function extracts the points with labels different from -1 (indicating they belong to clusters) and returns this denoised point cloud (line 4). The ProcessDirectories procedure (lines 7-12) iterates through all directories and files within the specified root directory (line 7). It checks if the directory name is 'particle' and processes each HDF5 file within those directories (lines 8-10). For each file, it calls the DenoisePointCloudH5 function (line 9) with `eps=0.06` and `min_samples=10`, denoises the point cloud, and then saves the denoised point cloud back to the original file (line 10). An example of this process can be seen in Figure 4.5.

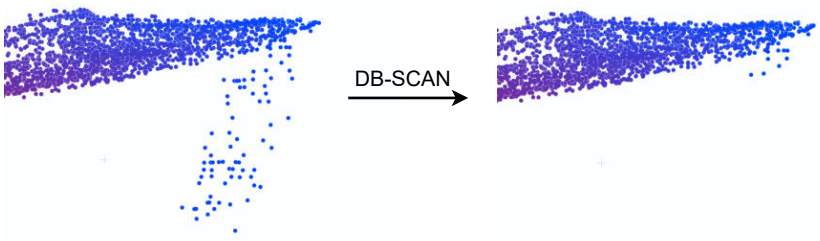

Figure 4.5: Illustration of the functionality of the DBSCAN algorithm with `eps` = 0.06 and `min_samples` = 10.

In the DBSCAN algorithm, `eps` (epsilon) is the maximum distance between two points for them to be considered part of the same neighbourhood. `min_samples` is the minimum number of points required within an `eps` radius for a point to be classified as a core point,

forming the core of a cluster (see Section 3.2.2). These parameters determine the cluster density: `eps` controls the size of the neighbourhood, and `min_samples` ensures enough points to form a dense region [Sch+17]. Together, they influence how clusters are identified and how noise or outliers are detected. During implementation, a higher `eps` value was chosen (0.06) to accommodate the low density of the point clouds, ensuring that fewer points were deleted compared to an `eps` value of 0.03. In total following parameters were tested with $[\text{min}_{\text{samples}}, \text{eps}]$: $[50, 0.08]$, $[30, 0.08]$, $[20, 0.08]$, $[20, 0.05]$, $[20, 0.06]$, $[10, 0.06]$. Where $[10, 0.06]$ archived exactly the result needed, by only deleting extreme outliers without touching the rest of the point cloud significantly (see Figure 4.5). The change in Chamfer Distance observed is only around 1%. This relatively small change is likely because the Chamfer Distance, which measures the average distance between points in two point clouds, is not significantly affected by removing a small number of extreme outliers, especially when the rest of the point cloud remains quite noisy [FSG17]. The primary goal of this step is to remove outliers that could potentially disrupt the PCTMA-Net in subsequent processing stages. By eliminating these extreme outliers, the risk of distraction during the reconstruction process is minimized, ensuring more reliable results from the PCTMA-Net.

4.5 PCTMA-Net

Up to this point, 2D sonar images and ground truth data were created in various datasets and converted into 3D point clouds using the EAE-Net. This output was converted to .h5 format and denoised using the DBSCAN algorithm to remove extreme outliers. The datasets are further categorised as described in section 4.2.2 to allow comparisons in performance. The PCTMA-Net, initially designed for the reconstruction of individual noise-free objects [Lin+21], is now to be tested for the first time without changing the initial architecture or adding pre-training.

4.5.1 Preprocessing

To ensure the PCTMA-Net operates effectively, several preprocessing steps were required to harmonise the output structure of the EAE-Net with the input specifications of the PCTMA-Net. These preprocessing steps are outlined as follows and visualized in Figure 4.6.

1. The `copy_h5_to_database.py` script, pseudocode attached in the appendix A.0.1, **copies and organizes point cloud files** stored in HDF5 format from a source directory into **training, validation, and test directories** for each dataset. It processes files located in "gt" (ground truth) directories and their corresponding "particle" files. Files are assigned to training, validation, or test sets based on their numeric filenames: files with numbers less than 1294 are placed in the training set, numbers between 1294 and 1456 go to the validation set, and numbers 1456 and above are assigned to the test set (in total 1615 files for each dataset). The script renames the files sequentially within each set and ensures that both "gt" and "particle" files are copied to their respective destinations. For test files, only the "particle" files are processed and copied to the test directory.

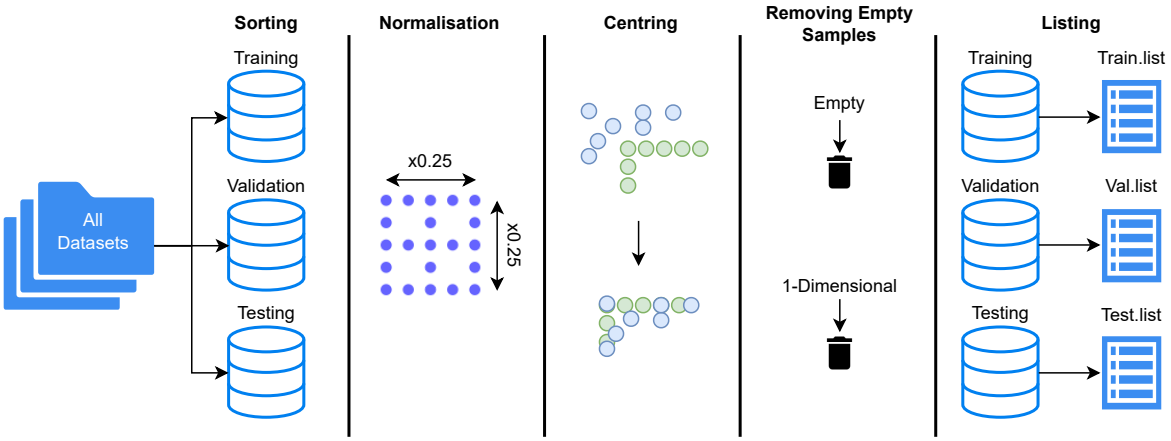

Figure 4.6: Visualization of the five preprocessing steps: sorting, normalization, centring, removing empty samples, and listing the samples.

2. It turned out that the point clouds of the EAE mesh were not normalised as intended by the initial architecture of the PCTMA-Net [Lin+21]. As a result, the point cloud was not reconstructed, and points were generated in the empty space from it. To prevent this, the **point clouds are normalised** in the next step. The `bound_box_test.py` script, pseudocode shown in the appendix A.0.1, normalises point cloud data stored in HDF5 files by applying a scale factor and translation. For each file in a specified directory, it reads the point cloud data, normalises it using the given scale factor $[0.25, 0.25, 0.25]$ and translation $[0.25, 0.25, 0.25]$, and then saves the normalised point cloud back to the file. The normalisation process, therefore, scales the point cloud by 0.25 and then translates it by 0.25 units in each dimension. The script iterates over all HDF5 files in the directory, applying these transformations to each file's point cloud data. This process ensures that the point clouds are uniformly scaled and positioned, facilitating further processing.
3. A slight shift has been observed between the ground truth point clouds and the EAE-Net output point clouds. To avoid reconstruction errors, the `center_to_gt.py` script **centres the EAE-Net output point clouds to the ground truth**. Detailed in Appendix A.0.1, the script calculates the centroid of the ground truth point cloud and subtracts it from each point in both the ground truth and EAE-Net output point clouds. The script's key functions are centroid calculation (reading a point cloud from an HDF5 file and determining its mean position), centring (aligning a point cloud with a reference centroid), pair processing (centring both ground truth and particle point clouds using the ground truth centroid), and directory processing (iterating through a root directory to identify and process pairs of ground truth and particle HDF5 files). This alignment ensures the EAE-Net output matches the ground truth, preventing reconstruction errors.
4. It was discovered that some samples were empty point clouds, causing errors in the PCTMA-Nets training process. The script `find_del_empty_or_zero.py`, stated in Appendix A.0.1, was therefore required to **search for, identify and remove these empty point clouds** from the datasets. The script works by iterating through all the HDF5 files in the specified directory. It checks each file to see if it contains an empty dataset or if the point cloud dataset is a 1D array. If either condition is met, the file is flagged for deletion. The script then deletes the identified files and their corresponding counterparts (e.g. if a file in the 'particle' directory is deleted, the corresponding file in the 'gt' directory is also deleted). By removing these problematic samples, the script will help maintain the data set's quality and reliability and allow the PCTMA-Net to function

correctly.

5. For the PCTMA-Net to efficiently iterate over samples during processing, samples must be stored in lists. The `write_list.py` script (Appendix A.0.1) **creates three lists (train, validation and test)** by extracting and organising file names from specified directories. The `create_list` function creates a list of file numbers. It constructs the path to the subfolder containing the partial data, checks if the directory exists and iterates through all the files with a `.h5` extension. The numeric part of each file name is extracted, converted to an integer for sorting and stored in a list. After sorting the list numerically, the script writes each number, optionally prefixed, to the specified list file. This process ensures that the PCTMA-Net can access and process the samples in an organised and systematic manner.

4.5.2 Restructuring

With the datasets now correctly formatted, structured, and cleaned to prevent errors, it is essential to reorganize the PCTMA-Net by Lin et al. [Lin+21] partially to accommodate the new datasets. This restructuring process is detailed below, following the sequence of script calls within the architecture at the start of training. Figure 4.7 provides an overview of the proposed restructuring. The original implementation of the PCTMA-Net uses the 3DCompletion Shapenet dataset [Cha+15]. All mentioned scripts of this section are initially implemented by [Lin+21]. The step-by-step description outlines how the architecture must be modified to process the new datasets and train the model effectively. The scripts mentioned here can be found in the GitHub repository, which is provided in the Appendix A.0.2. Due to the complexity of the scripts, this section will focus solely on the significant changes that have been made. Scripts and functions are explicitly named and referenced throughout the documentation to ensure ease of subsequent implementation. This approach aims to provide clear guidance and improve the overall understanding of the modifications and their context within the broader framework.

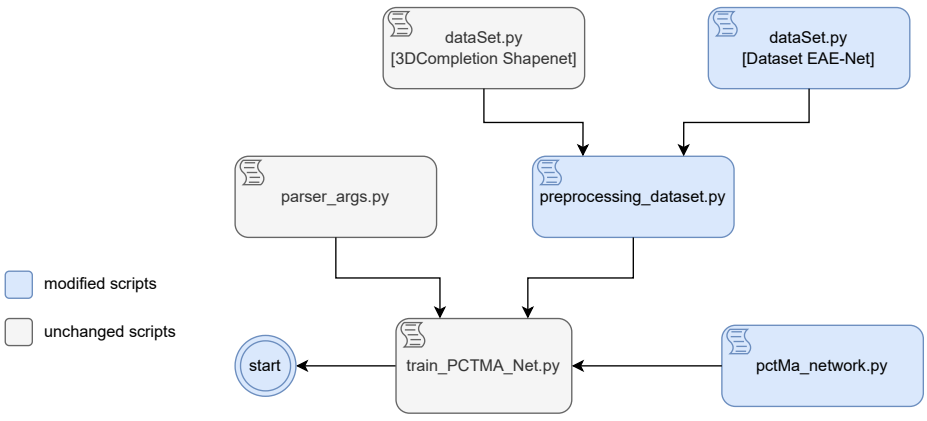

Figure 4.7: Overview of the essential scripts of the PCTMA-Net [Lin+21] needed to implement the new database.

1. **train_PCTMA_Net.py** The command: `python ../[...]/train_PCTMA_Net.py -batch_size 48 -epochs 300 -checkpoint_name best_model.pth -best_name best_model.pth -load_checkPoints False -train True -evaluate False` calls the `train_PCTMA_Net.py` script and provides the batch size, the number of epochs, the name of the checkpoints, and the name of the best model. It also specifies whether a checkpoint should be loaded (this allows pre-trained

weights to be used) and whether it is a training of the model or an evaluation of this model. The `train_PCTMA_Net.py` initializes the training or evaluation process and imports the scripts shown in Figure 4.7 to load the training data and methods to do so.

2. **preprocessing_dataset.py** The `train_PCTMA_Net.py` imports the used dataset via `import load_data` using the `preprocessing_dataset.py` script. This script begins by importing necessary modules and components, including the `EAENetData` dataset class from `pointComNet.pytorch_utils.components.dataSet`. To handle data loading appropriately, it is necessary to set up a new `load_data(args)` function within the script specifically for the `EAENetData`. The `load_data` function is defined to load the data required for training or evaluation. Depending on the `train` flag, the function creates a `DataLoader` for the dataset, specifying parameters such as the number of points, number of workers, batch size, shuffle, and drop_last options. This setup ensures that the dataset is appropriately loaded for both training and evaluation purposes. Thus, the dataset `EAENetData` is imported via the `dataSet.py` and set up to facilitate the loading of the data for training and evaluation.
3. **dataSet.py [Dataset EAE-Net]** The general purpose of the `dataSet.py` script is to handle the loading, preprocessing, and augmentation of datasets required for training and evaluating point cloud completion networks. It handles the used datasets, providing necessary functionalities to process and prepare data for the PCTMA-Net and is initially implemented for the Completion3D of the Shapenet dataset (28,000 incomplete, noise-free samples) [Cha+15]. The Completion3D dataset consists of eight different classes of samples [”*Plane*”,”*Cabinet*”,”*Car*”,”*Chair*”,”*Lamp*”,”*Couch*”,”*Table*”,”*Watercraft*”]. Since this thesis's generated datasets (defined in Section 4.2.2) do not have eight different classes but multi-object scenes, the script defines a new class for the EAE-Net dataset. For both datasets, the script provides functionalities for reading point cloud data from HDF5 files, applying various preprocessing steps such as downsampling and rotation, and organizing the data into training, validation, and test sets, with specific implementations for different datasets. The script ensures that both dataset samples are in the correct format and structure needed for the subsequent training and evaluation processes in the PCTMA-Net. The main difference between the `EAENetData` and the `Shapenet` dataset classes is the category handling is very different between ShapeNet and EAE-Net datasets. The `PointCompletionShapeNet` class manages the ShapeNet-specific categories through several methods. The `get_category_file()` method reads category information from a file and maps it to specific labels used within the dataset. The `label_to_category()` method converts label indices back to their respective category names, allowing reference to different object categories. The `get_with_shapeName()` and `get_with_shapeNames()` methods filter and retrieve data based on specific shape names, allowing the dataset to be queried for particular objects within ShapeNet's extensive collection.
   In contrast, the `EAE-Net` class in the EAE-Net dataset handles categories in a more simplified way. It contains similar category management methods but is tailored to the specific needs of EAE-Net. The `get_category_file()` method reads from the `synsetoffset2category.txt` file, which maps categories particular to the EAE-Net dataset. The label handling methods in `EAE-Net` are designed to map these category labels to dataset-specific identifiers. Unlike ShapeNet, which deals with multiple categories, EAE-Net simplifies this process by labelling the entire dataset as a single category, reducing unneeded complexity.
4. **pctMa_network.py** Directly imported by the `train_PCTMA_Net.py` script, the `pctMa_network.py` script implements the structure and core functionalities of the ar-

chitecture of the PCTMA-Net. It initializes the encoder and decoder networks, sets up logging, and manages checkpoints. During the forward pass, the encoder extracts features from input point clouds, and the decoder reconstructs them. The script calculates losses using Chamfer Distance (CD) and Earth Mover's Distance (EMD) to guide training. It defines an epoch-based training loop, updating model weights using an Adam optimizer. The evaluation process assesses model performance on test data, logs results, and saves reconstructed point clouds optionally. Checkpointing functionality allows saving and loading model states [Lin+21]. The script handles two datasets, EAE-Net and ShapeNet, with specific configurations for each evaluation method to process the different categories. EAE-Net focuses its evaluation on the 'ground' class, appending results to this single category. In contrast, ShapeNet evaluates the eight object classes, logging results separately for each. This approach provides a detailed insight into the model's performance on the range of objects. Another key difference is the logging of evaluation metrics. EAE-Net calculates and logs the total elapsed time and the average time per sample during evaluation. This provides a clear understanding of the computational efficiency and performance of the model when dealing with EAE-Net data. Given the potential application of the model in underwater navigation, an analysis of the computational time is of particular interest.

4.5.3 Changed Hyperparameter

The PCTMA-Net was then trained with the nine data sets (Section 4.2.2) to generate **interim results** and provide conclusions for possible changes. For this purpose, 300 epochs with a batch size of 48 were used as in the original paper[Lin+21]. The results of the first full training of the PCTMA network with all nine datasets listed in Section 5.2 showed initially satisfactory performance in terms of Chamfer Distance but also two problems. The first problem is that artefacts in the form of dot planes have formed, as seen in Section 5.2. These point plane artefacts are added very close to some ground truth points. The hypothesis is that these artefacts potentially contribute to the high recall and, consequently, the high F-score and Chamfer Distance. Although these metrics indicate good performance, the artefacts do not reconstruct the input point cloud as accurately as intended, leading to a less desirable reconstruction result, visible in Figure 5.2. The second problem is a difference in the density of the reconstructed point clouds within the individual point clouds. In the lower dense regions, single point planes generated by the Morphin-Atlas-based point generator (see Section 3.2.3) are distinguishable from other point planes close to them. These optically recognisable point planes and the artefacts near some ground truth points allow the hypothesis that the model has problems reconstructing the point clouds as desired due to the noise in the input. The result of the original paper in which neither artefacts nor conspicuously different densities within a point cloud nor recognisable point planes occurred can be noted for this [Lin+21].

The significantly different results are hypothetically attributed to two main contrasts between the original paper and the method presented here. Firstly, at around 28,000 samples, the 3Dcompletion Shapenet is twice as large as the datasets generated in this paper [Cha+15]. Secondly, the 3DCompletion Shapenet only contains single object and, above all, noise-free incomplete point clouds.

The first conclusion drawn in this work is to **increase the sample size**, not by further generation of noisy point clouds but by **pre-training** using the 3DCompletion Shapenet [Cha+15]. After the pre-training, the noise-free pre-trained weights are used to train on the generated

noisy nine datasets with 14.535 samples. The second conclusion from the hypothesis that noise causes problems is the **change in the architecture** of the PCTRMA network. More specifically, noise is expected to be more complex than noise-free point clouds. Thus, another hypothesis is that the network must have an **increased complexity** to perform a successful reconstruction. It should be noted that this is associated with increased computational capacity and computational time. There is also a risk of overfitting. This occurs when the model becomes too complex relative to the amount and quality of training data. Overfitting can lead to poor generalisation of new, unseen data. For this reason, the changes described later were considered and kept within manageable limits. In addition, the network has implemented methods to prevent overfitting, such as a dropout rate of 0.1. It should be mentioned that the division into pre-training and training results in a significant and complete data exchange, which should not encourage overfitting. The specific changes are summarised in Table 4.1.

Hyperparameter	**Initial Configuration**	**Updated Configuration**
Num_Encoder	4	6
num_head	8	12
d_ff	2048	2400
d_model	1024	1200
En_channels	[1024]	[1200, 1200]
De_channels	[1024, 2048, 2048]	[1200, 2400, 2400]
use_cd	True	False
use_emd	True	True
n_primitives	32	48

Table 4.1: Comparison of hyperparameters between the initial [Lin+21] and updated configurations of the PCTMA-Net

Number of encoder layers (Num_Encoder). The encoder layer of the PCTMA-Net is explained in Section 3.2.3. The ability of a neural network to handle noisy input point clouds can be significantly improved by increasing the number of encoder layers in the network. With more layers, the network gains a greater capacity to learn complex and nuanced representations. The network can progressively filter and abstract the input data with each additional layer, capturing the intricate features and hierarchical patterns within the point cloud data [Alz+21].

Number of attention heads (num_head). The Multi-Head-Attention mechanism is described in Section 3.2.3. Increasing the number of attention heads allows the model to simultaneously focus on different parts of the input, enhancing its performance by capturing more nuanced features. This is especially useful in point cloud processing, where detailed topology and structure are crucial, as in the noisy EAE-Net output. The offset self-attention mechanism, referenced in the theory part of this work (see Section 3.2.3), leverages a larger receptive field and permutation-invariant properties to handle disordered data effectively.

Feed-forward network dimension (d_ff). As described in Section 3.2.3, increasing the feed-forward network dimension (d_{ff}) in a neural network can significantly enhance the model's ability to learn complex functions, leading to improved performance. A larger d_{ff} provides the network with more parameters and a greater capacity to represent intricate patterns and relationships in the data. While lower layers capture more basic patterns, upper layers learn more abstract and semantic features, which is particularly beneficial for tasks requiring detailed feature extraction and complex decision-making [Gev+20].

Model Dimension (d_model), Encoder channels (En_channels) and Decoder Channels (De_channels). As explained in Section 3.2.3, increasing the model dimension (d_{model}), encoder channels (En_channels), and decoder channels (De_channels) in a neural network can enhance its ability to learn and represent complex data [Wu+16]. A larger d_{model} allows for richer representations, capturing intricate patterns and relationships within the data, which improves model accuracy. More encoder channels improve feature extraction by processing different aspects of the input data simultaneously, leading to a more comprehensive understanding. Similarly, additional and larger decoder channels enhance the network's ability to reconstruct detailed and precise outputs. These enhancements significantly increase computational requirements, including memory usage and processing power, resulting in longer training times and higher resource consumption. Balancing these improvements with computational efficiency is crucial for effective neural network design[Wu+16][Alz+21].

Earth Mover's Distance (EMD) instead of Chamfer Distance (use_cd). Chamfer Distance is a metric that measures the similarity between two point clouds [PW09]. Not using Chamfer Distance in the second configuration could lead to differences in how the model optimizes and evaluates the point cloud reconstruction. The Earth Mover's Distance (EMD) between two distributions P and Q is defined as [PW09]:

$$\text{EMD}(P,Q) = \min_{f_{ij}} \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d(p_i, q_j) \tag{4.4}$$

subject to:

- $f_{ij} \geq 0$
- $\sum_{j=1}^{n} f_{ij} = w_i$ for all i
- $\sum_{i=1}^{m} f_{ij} = v_j$ for all j

where $d(p_i, q_j)$ is the distance between points p_i and q_j, and f_{ij} is the flow from p_i to q_j. The Earth Mover's Distance (EMD) is more robust for reconstructing noisy point clouds. EMD considers the optimal transport of all points in P to all points in Q, distributing the total cost of transforming one distribution into another. This mitigates the influence of outliers, as the cost is spread across the entire distribution rather than being dominated by individual point distances, unlike Chamfer Distance, which relies on nearest neighbours. Additionally, EMD smooths the influence of individual points, resulting in a more stable similarity measure [PW09]. This change is expected to serve as a counter against different density levels within a point cloud, as observed in the interim results (Section 5.2).

Number of primitives (n_primitives) As defined in Section 3.2.3, increasing the number of primitives used in point cloud reconstruction can improve the model's ability to capture finer details. Each primitive acts as a basic shape to approximate the overall structure, and more primitives will allow for a more detailed representation. Research like [Sha+22] shows that using more primitives improves the quality of reconstructed point clouds, capturing

intricate details and subtle features required for precise applications such as 3D modelling and CAD.

The new configuration could improve the model's ability to learn and represent complex point cloud data, with more encoder layers, attention heads and larger dimensions for both encoder and decoder channels. This can lead to improvements in accuracy, as the model can more effectively capture and reconstruct finer details. However, this added complexity increases the computational requirements. This requires more memory and processing power, resulting in longer training times and higher resource costs. In addition, the optimisation behaviour of the model can be altered, resulting in different reconstruction characteristics, if the Chamfer Distance is not used as a loss metric. In addition, a more detailed and accurate representation of the original point cloud is achieved by increasing the number of primitives.

4.5.4 Postprocessing

Removing Input Points

Algorithm 3 Delete Input Points

```
1: function RemoveInputPointsFromOutput(input_file, output_file, result_file)
2:     Convert both point clouds to NumPy arrays
3:     Identify and keep unique points in the output point cloud not present in the input point cloud
4:     Create a new point cloud from these unique points and save it
5: end function
6: function ProcessDirectory(input_dir, output_dir)
7:     Retrieve list of all input files in input_dir
8:     for all input_file in list of input files do
9:         Determine corresponding output_file and ground_truth_file paths
10:        Call RemoveInputPointsFromOutput with input_file, output_file, and result_file
11:    end for
12: end function
13: Example Usage:
14: Call ProcessDirectory with input_dir and output_dir
```

The last change is the post-processing of the PCTMA-Net's output data. Since the PCTMA-Net only adds points and does not remove any points, the resulting point clouds are still distorted by the noise in the input point clouds and do not show the actual result. The deletion was not necessary for the original work as the point clouds did not contain any noise [Lin+21]. Therefore, a script was implemented to delete the input points. The functionality can be described as in the pseudocode 3. The process consists of two main functions and a general procedure for dealing with directories of point cloud files. The `RemoveInputPointsFromOutput` function is designed to remove points from the output point cloud that are present in the input point cloud. It begins by loading the input and output point clouds from the specified files and converting them to NumPy arrays to make manipulating easier. The function then identifies points in the output point cloud that are not present in the input point cloud. These unique points are retained. A new point cloud is created from these unique points and the resulting point cloud is saved to a specified result file. The `ProcessDirectory` function handles the mass processing

of point cloud files within a directory. It first checks for the existence of the output directory and creates it if necessary. The function retrieves a list of all input files in the input directory and processes each file. For each input file, the function determines the corresponding paths for the output and ground truth files and sets the paths for the resulting files in the output directory. It calls the `RemoveInputPointsFromOutput` function to process the point clouds and removes the points from the output that are present in the input.

In addition, the Chamfer Distance (Section 4.5), the Earth Mover's Distance (Section 4.5.3), the F-score (Section 4.3), the Hausdorff distance (Section4.3) and two other metrics were implemented to evaluate the data from various perspectives. The two new metrics are presented below.

The Average Nearest Neighbour Distance (ANN distance) is a metric used to quantify the density of a point cloud. It measures the average distance from each point in the point cloud to its nearest neighbour in the cloud [BA72]. This metric provides a straightforward way to assess how densely packed the points are within a given spatial dataset. ANN Distance is closely related to the concept of K-Nearest Neighbours (KNN), a fundamental method in data analysis and machine learning [MD15]. In the context of ANN distance, the KNN method is used to find the nearest neighbour (k=10) for each point in the point cloud. The nearest neighbour search in this implementation is performed using a KD tree. This efficiently organises the data to find the nearest point quickly. The ANN distance is calculated using the following steps and formula. First, the nearest neighbour is determined for each point in the point cloud. For each point p_i in the point cloud P, find its nearest neighbour p_j. Compute the Euclidean distance $d(p_i, p_j)$ between p_i and p_j. Then, calculate the mean of these distances[MM99]:

$$\text{ANN distance} = \frac{1}{N}\sum_{i=1}^{N} d(p_i, p_{NN(i)}) \tag{4.5}$$

where N is the total number of points in the point cloud, p_i is the ith point in the point cloud, $p_{NN(i)}$ is the nearest neighbour of p_i, and $d(p_i, p_{NN(i)})$ is the Euclidean distance between p_i and its nearest neighbour $p_{NN(i)}$. The nearest neighbour search for each point p_i in the point cloud identifies the nearest neighbour $p_{NN(i)}$ using a nearest neighbour search algorithm such as KD-tree. This step is essential to determine the closest point to p_i. The Euclidean distance between each point p_i and its nearest neighbour $p_{NN(i)}$ is then calculated. The Euclidean distance in a 3D space is given by[MD15]:

$$d(p_i, p_{NN(i)}) = \sqrt{(x_i - x_{NN(i)})^2 + (y_i - y_{NN(i)})^2 + (z_i - z_{NN(i)})^2} \tag{4.6}$$

Finally, by averaging these distances, the ANN distance measures the average distance between points in the point cloud.

The Median Nearest Neighbour Distance (MNN distance) is used to assess density in a point cloud by calculating the median distance to its k nearest neighbours for each point. This metric provides a robust measure of the density of a point cloud that is less likely to be affected by outliers when compared to the ANN distance. The calculation begins by finding the k nearest neighbours as in the ANN distance. The median of these distances is then determined for each point:

$$\text{Median Distance}(p_i) = \text{median}\left\{d\left(p_i, p_{j1}\right), d\left(p_i, p_{j2}\right), \ldots, d\left(p_i, p_{jk}\right)\right\} \tag{4.7}$$

Finally, the overall median distance metric for the point cloud is calculated as the mean of these median distances:

$$\text{Mean Median Distance} = \text{mean}\{\text{Median Distance}(p_1), \ldots, \text{Median Distance}(p_N)\} \quad (4.8)$$

where N is the total number of points in the point cloud. The interim results inspired the idea of calculating both the ANN and the MNN distance. As Section 5.2 describes, the point plane artefacts are expected to strongly influence the measured values, such as the Chamfer Distance or F-score. The metrics now implemented are intended to show the predicted difference in density between the point clouds of the interim results and the final results.

4.6 Experimental Training Configurations

The pipeline described in this chapter and its adaptation based on the hypotheses established by interim results will now be evaluated in detail using new test results. The following training configurations, described in Table 4.2, will be conducted to obtain a comprehensive and comparable overview of the effectiveness and performance of the PCTMA-Net in the different data sets.

Datasets (Samplesize)	**Pre-Training**	**DBSCAN Denoising**	**Updated Hyperparameter**
AllData (14,535)	Yes	Yes	No
AllData (14,535)	No	Yes	Yes
AllData (14,535)	Yes	Yes	Yes
Floor (8,075)	No	Yes	Yes
Floor (8,075)	Yes	Yes	Yes
NoFloor (6,460)	No	Yes	Yes
NoFloor (6,460)	Yes	Yes	Yes

Table 4.2: Training configurations for various datasets, indicating whether pre-training, DBSCAN denoising, and updated hyperparameters were applied.

The configurations include combinations of pre-training, DBSCAN denoising and updated hyperparameters. Three different configurations were used for AllData: One with all three methods, one with pre-training and DBSCAN denoising but without updated hyperparameters, and one without pre-training but with DBSCAN denoising and updated hyperparameters. Similarly, the Floor and NoFloor datasets were each evaluated in two configurations: one without pre-training but with DBSCAN denoising and updated hyperparameters and one with all three methods applied. This approach ensures a comprehensive evaluation of model performance under different training conditions, allowing a detailed comparison of the impact of both pre-training and updated hyperparameters on overall model performance. In addition, the distinction between Floor and NoFloor will be used to test how the model can handle multi-object point clouds without connecting the individual objects. All models were trained for 300 epochs with a batch size of 48. The training was conducted on a system running Ubuntu 18.04.1. The hardware configuration included an 11th Gen Intel(R) Core(TM) i9-11900K CPU @ 3.50GHz and an NVIDIA GeForce RTX 3090 GPU with 24GB of memory,

utilizing CUDA compilation tools, release 11.6, V11.6.55. It is important to note that the initial setup, as required by the PCTMA-Net paper, specified CUDA 10.1 [Lin+21]. That setup was not possible due to compatibility issues with the drivers for the RTX 3090 GPU. This necessitated the use of CUDA 11.6 to ensure proper functionality and performance during the training and evaluation of the model. The results are presented in the Evaluation Chapter 5 and compared to the interim results (Section 5.2).

Chapter 5

Evaluation

5.1 Data Description

The nature and form of the data resulting from the PCTMA network are described before the interim and final results are evaluated. For all datasets, the average point count of the ground truth remained constant at 2048 points, with no variability as both the minimum and maximum point counts were also 2048. This is because the ground truth is downsampled and upsampled to 2048 points by the PCTMA-Net pre-processing, providing a standardised basis for comparison [Lin+21]. The reconstructed point clouds showed intriguing variability depending on whether the hyperparameters were updated. When the hyperparameters were updated, the number of points in the reconstructed point clouds was 8112, with no deviation in the minimum and maximum values. In contrast, without updated hyperparameters, the number of points was higher at 8192, again with no variability. As a result, the minimum and maximum values were 8112 and 8192, respectively. This slight variability in the reconstruction methods between the datasets, as reflected in the point cloud counts, could potentially impact the accuracy and precision of the reconstructed point clouds. The input file format for the point clouds in the PCTMA-Net is `.h5`, while the output is generated in `.ply` format.

The overall average values of the bounding box in every dimension based on the provided dataset are:

- **Average ground truth bounding box dimensions (in meter):**
  - X: 1.5688
  - Y: 1.5257
  - Z: 0.9420 (height)
- **Average reconstructed bounding box dimensions (in meter):**
  - X: 1.6125
  - Y: 1.5499
  - Z: 1.0324 (height)

These values represent the average extent of the bounding boxes in the x, y, and z dimensions for both ground truth and reconstructed point clouds across all the datasets.

5.2 Interim Results

The PCTMA-Net was initially trained and tested in its original form, as described in [Lin+21], with the two existing data sets and the seven newly generated datasets (see Section 4.2.2). In total, the dataset comprises nine datasets, amounting to 14,535 samples. This includes 6,460 samples without floor data and 8,075 samples with floor data.

Category	Value	Description
Num_Encoder	4	Number of layers in the encoder (see Section 3.2.3).
num_head	8	Number of attention heads (see Section 3.2.3).
d_ff	2048	Dimension of the feed-forward network.
d_model	1024	Size of the hidden states, allowing the model to capture complex patterns.
En_channels	1024	Number of channels in the encoder layers.
De_channels	1024, 2048, 2048	Number of channels in the decoder layers.
use_cd	False	Chamfer Distance (CD) loss is used.
n_primitives	32	Primitives used in the model, leading to finer shape representation.
use_emd	True	Earth Mover's Distance (EMD) is used as a loss function.
w_cd	10000	Weight of the Chamfer Distance loss.
w_emd	0.00001	Weight of the Earth Mover's Distance loss.
use_atlas	True	AtlasNet module for generating point clouds.

Table 5.1: Configuration and descriptions of the initial PCTMA-Net [Lin+21].

The dataset was meticulously divided into training, validation, and test sets to ensure a robust evaluation and training process. Specifically, 11,637 samples (approximately 80.07%) were allocated for training, 1,458 samples (approximately 10.03%) for validation, and 1,440 samples (approximately 9.90%) for testing. Accordingly, the real, experimentally obtained data set and a synthetic, Blender-generated data set were used to obtain the ground truth point clouds. The original average **Chamfer Distance** of the output of the EAE-Net is **134.5**. In Table 5.1, the used PCTMA-Net configuration of hyperparameters is stated, according to [Lin+21]. To assess its performance under the original conditions, the model was trained for 300 epochs with a batch size of 48. The training was conducted on a system running Ubuntu 18.04.1. The hardware configuration included an 11th Gen Intel(R) Core(TM) i9-11900K CPU @ 3.50GHz and an NVIDIA GeForce RTX 3090 GPU with 24GB of memory, utilizing CUDA compilation tools, release 11.6, V11.6.55. The initial setup specified in the PCTMA-Net paper required CUDA 10.1 [Lin+21]. However, this setup was incompatible with the drivers for the RTX 3090 GPU, necessitating the use of CUDA 11.6 to ensure proper functionality and performance during the training and evaluation of the model. This evaluation was conducted without altering any hyperparameters, generating new datasets,

or utilizing pre-trained weights. The objective was to observe the model's baseline results in their unmodified state. The resulting **average loss of 37.7249**, determined by the Chamfer Distance, signifies a substantial improvement over the previous 134.5 reported for the EAE-Net. The Chamfer Distance for the PCTMA-Net is scaled by 10^4. In contrast, the original EAE-Net script scales the Chamfer Distance by 500. To ensure comparability between the two models, the EAE-Net value is further scaled by 20. Figure 5.1 displays the input point cloud, the ground truth point cloud, and the resulting reconstruction point cloud. A closer examination of the point cloud itself (see Figure 5.1) reveals additional insights and first conclusions.

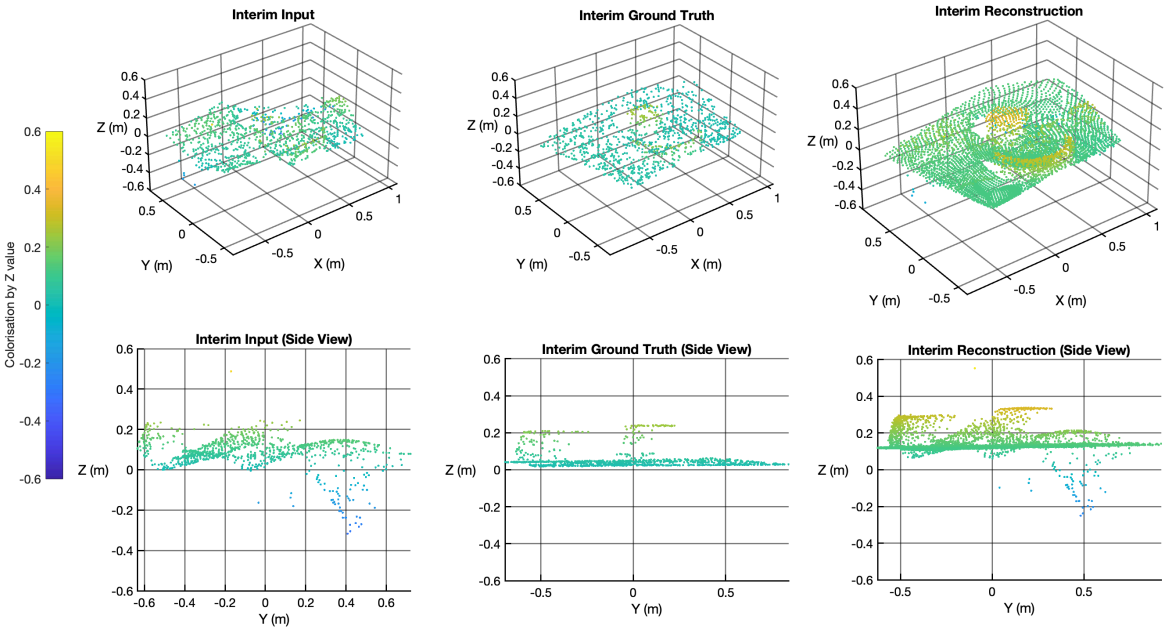

Figure 5.1: Illustrating an example of the interim input point cloud, the interim ground truth point cloud, and the resulting interim reconstruction, presented and colorized from two different viewpoints.

In the context of point cloud reconstruction, the F-score is a measure of accuracy that takes into account both precision and recall. The precision is the percentage of the ground truth points with a corresponding reconstruction point within a certain distance (threshold value) in the reconstructed point cloud. This indicates how well the reconstruction captures the actual ground truth points. The recall is the percentage of reconstruction points within the specified distance of any ground truth point. This indicates how many of the reconstruction points added during the reconstruction are correct or relevant. For the method without pre-training and specific hyperparameters, 78.3388 % of the ground truth points have corresponding reconstruction points in the reconstructed cloud, signifying a high level of accuracy. Moreover, 62.5904 % of the reconstruction points are in close proximity to the ground truth points, demonstrating recall. This observation suggests that the method adeptly encompasses the existing ground truth points while introducing fewer reconstruction points away from the ground truth, resulting in a more balanced distribution. Consequently, the method attains a relatively high overall accuracy, as evidenced by the F-score.

As shown in Figure 5.2, point plane artefacts are added very close to some ground truth points. Although these metrics indicate good performance, the artefacts do not reconstruct the input point cloud as accurately as intended, leading to a less desirable reconstruction result. In addition, the density of reconstruction differs greatly within a point cloud.

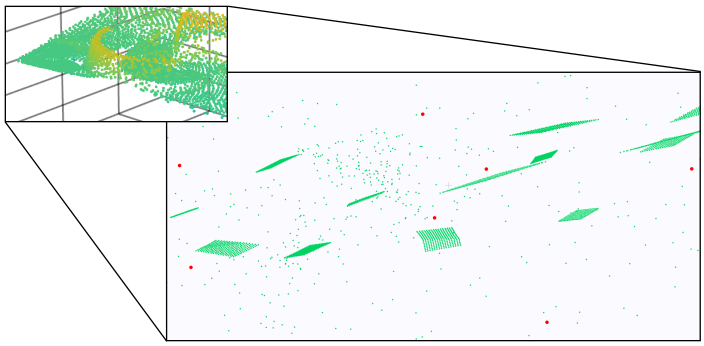

Figure 5.2: A zoomed-in view of the interim reconstruction point cloud after the interim training, highlighting visible point plane artefacts. The green points are the reconstructed points. The red points represent the lower dense ground truth.

5.3 Results

5.3.1 Combined Datasets (AllData)

The results of the final training sessions, meticulously measured using six metrics, are presented in the following section. The F-score, a key metric, is further divided (combined precision and recall) for a comprehensive evaluation. Table 5.2 showcases the measured results of all nine datasets combined (see Section 4.2.2). Figure 5.3 presents these metrics in a more meaningful graphical form. While Figure 5.4 illustrates an example of the dataset, presenting the point cloud input, ground truth, and reconstruction.

Evaluation Metric	**Hyper Denoise**	**PreTrain Denoise**	**PreTrain Hyper Denoise**
Chamfer Distance	39.8653	50.5123	40.5007
Earth Mover's Distance	0.2186	0.2278	0.2131
Hausdorff Distance	0.4176	0.3614	0.3741
F-Score Combined < 0.02 (%)	63.0970	46.6730	62.1463
F-Score Precision < 0.02 (%)	89.3146	72.3683	88.2144
F-Score Recall < 0.02 (%)	52.2506	38.3299	51.1031
Average Nearest Neighbour (ANN)	0.0190	0.0205	0.0188
Median Nearest Neighbour (MNN)	0.0179	0.0200	0.0177

Table 5.2: Evaluation of the trained models with all nine datasets combined in different configurations, truncated to the fourth decimal place. In each row, the green cell represents the best and the red cell the worst value.

It is important to understand the different configurations and analyze their impact on the evaluation results. *PreTrain* indicates whether the pre-trained weights from the 3DCompletion ShapeNet [Cha+15] were included.
Hyper specifies whether the updated hyperparameters were used. *Denoise* denotes whether the DBSCAN algorithm was applied to the data before using it with the PCTMA-Net. Although denoising was performed on all datasets, this label is included for the sake of completeness. Importantly, all configurations utilized the DBSCAN algorithm.

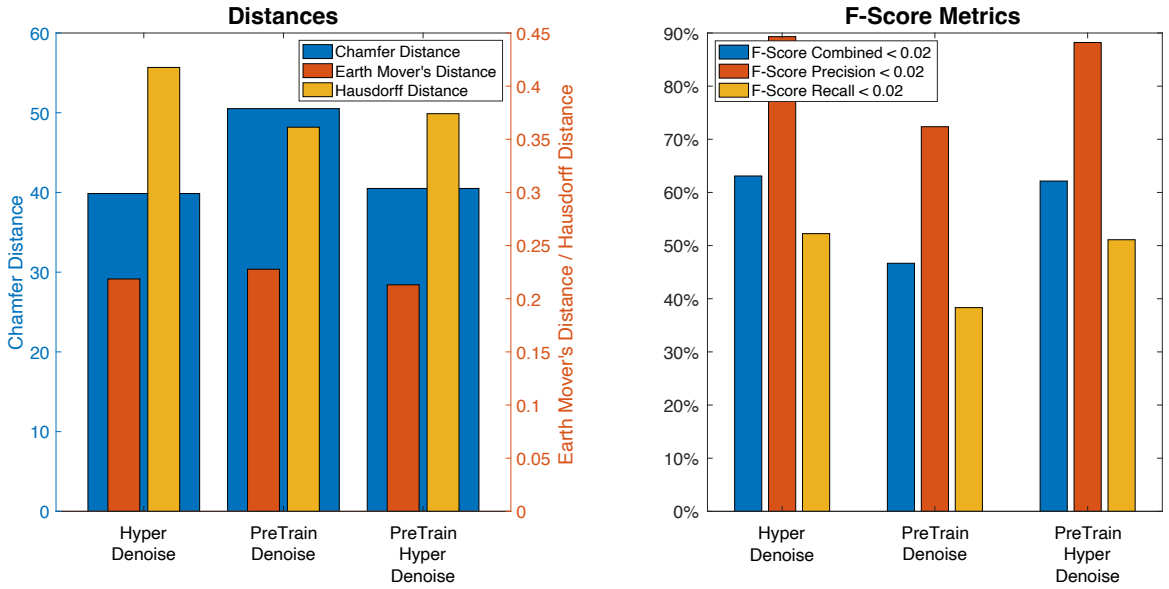

Figure 5.3: An illustration of the meaningful metrics from Table 5.2 of all *combined* datasets. The left diagram shows the Chamfer Distance (blue) as well as the Earth Mover's (red) and the Hausdorff (orange) distance of the three training configurations. The bar chart on the left visualises the three F-score values (Combined, Precision, and Recall) of the three training configurations.

The Chamfer Distance, a key metric in point cloud reconstruction, reveals the average squared distance between points in the predicted and ground truth point clouds. Lower values signify superior performance [FSG17]. Notably, the *Hyper-Denoise* configuration, which leverages updated hyperparameters without pre-trained weights, demonstrated the best performance. However, the difference to the *PreTrain-Hyper-Denoise* configuration is negligible (see Figure 5.4). In contrast, the *PreTrain-Denoise* configuration, which lacks updated hyperparameters, fared the worst. This stark contrast underscores the crucial role of hyperparameters in achieving accurate point cloud reconstruction.

The Earth Mover's distance measures the minimum cost of transforming one point cloud into another [PW09]. According to Table 5.2, the *PreTrain-Hyper-Denoise* configuration performed best, while the *PreTrain-Denoise* configuration performed worst. This suggests that combining pre-trained weights and updated hyperparameters results in the most accurate point cloud transformation regarding the Earth Mover's distance. The PCTMA-Net used the Earth Mover's distance as the evaluation metric and, therefore, for calculating the loss function. Due to that, this metric serves as the primary indicator for the success of the training here.

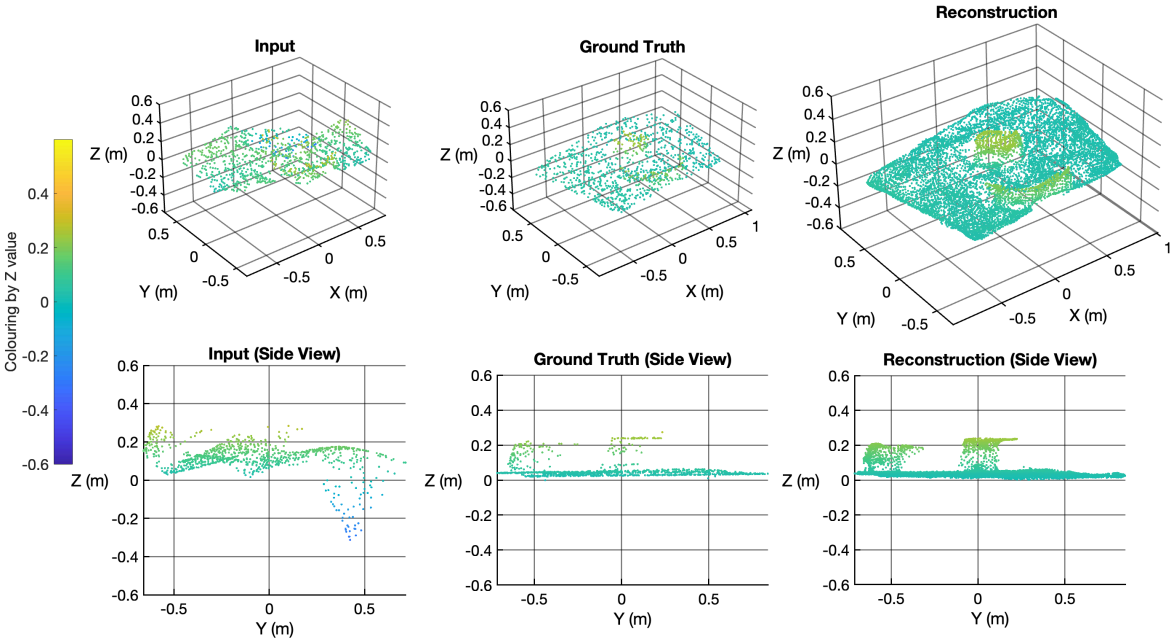

Figure 5.4: An example illustrating the input point cloud, the ground truth point cloud, and the resulting reconstruction of the *combined* dataset (AllData) is presented from two different viewpoints.

The Hausdorff distance quantifies the maximum distance from a point in one point cloud to the nearest point in the other, reflecting the worst-case scenario [HKR93]. Table 5.2 shows that the *PreTrain-Denoise* configuration had the best performance, while the *Hyper-Denoise* configuration had the worst. This indicates that pre-trained weights alone help to reduce the maximum error between point clouds for this metric, highlighting their impact on minimizing outlier distances.

The F-Score, a comprehensive measure of a model's performance that combines precision and recall, serves as an important indicator in this evaluation. Precision measures the proportion of true positive predictions out of all positive predictions, while recall measures the proportion of true positive predictions out of all true positive cases [Der16]. For a threshold of 0.02, the *Hyper-Denoise* configuration, which incorporates updated hyperparameters, showed slightly better results in terms of precision, recall, and the combined F-score compared to the other configurations. The *PreTrain-Hyper-Denoise* configuration values were nearly the same (see Table 5.2), indicating that updated hyperparameters significantly enhance both the accuracy and completeness of point cloud reconstructions. In contrast, the *PreTrain-Denoise* configuration, which lacks updated hyperparameters, performed the worst in all these metrics.

ANN calculates the average distance from a point in the predicted point cloud to its nearest ten neighbours [MM99], while MNN measures the median distance. Lower values indicate better performance for both metrics. As shown in Table 5.2, the *PreTrain-Hyper-Denoise* configuration performed best in both ANN and MNN, while the *PreTrain-Denoise* configuration performed worst. This indicates that pre-training combined with hyperparameter updates produces the closest and most uniformly distributed points to the ground truth. The combined interpretation of ANN and MNN suggests that the points are not only close on average but also consistently close, indicating a high-quality and uniformly distributed point cloud reconstruction without point concentrations (see Figure 5.4).

Summarising, the results of the trained models on the *combined* datasets (see Table 5.2) provide several insights into the effectiveness of the different configurations. The *Hyper-Denoise* configuration consistently achieved the best performance in Chamfer Distance, F-Score metrics, and recall, indicating that updated hyperparameters significantly improve the accuracy and completeness of point cloud reconstructions. The *PreTrain-Hyper-Denoise* configuration, which performed best in Earth Mover's Distance (EMD), ANN, and MNN metrics, suggests that combining pre-trained weights and updated hyperparameters leads to accurate and uniformly distributed point cloud reconstructions. Given that EMD was used as the loss function during training, its better performance in this configuration underscores the effectiveness of pre-training and hyperparameter optimization in improving model accuracy. The *PreTrain-Denoise* configuration, which performed the worst across most metrics, serves as a stark reminder of the limited impact of pre-trained weights alone. This underscores the crucial role of hyperparameter optimization in achieving the best reconstruction performance.

5.3.2 Comparison Between Single- and Multiobject Datasets

As already explained in the methodology in Section 4.2.2, the single-object datasets *Floor* and multi-object datasets *NoFloor* differ neither in the number of objects nor in the arrangement or other experimental configurations. Only the floor as an object was removed. This floor served as a connecting object, which consequently united the point cloud into a coherent point cloud. The *NoFloor* datasets represent unconnected multi-object scenes. As already described in the theory in Section 4.5, the PCTMA-Net was initially designed to reconstruct single object point clouds [Lin+21]. Since free-floating objects are not uncommon underwater, the structure implemented in this work was trimmed and tested for such scenes. The results of all implemented metrics in direct comparison to the normal *Floor* dataset configurations are shown in Table 5.3. Figure 5.5 presents these distance and F-score metrics in a graphical form. While Figure 5.6 illustrates an example of the *NoFloor* dataset in different training configurations, presenting the point cloud input, ground truth, and reconstruction.

The Chamfer Distance for the *Floor* datasets shows that both configurations (*Hyper-Denoise* and *PreTrain-Hyper-Denoise*) perform similarly with values around 42.5. In contrast, the *NoFloor* datasets show a significant difference between the configurations. The *PreTrain-Hyper-Denoise* of the *NoFloor* dataset configuration had the best performance overall, while the *Hyper-Denoise* configuration had the worst of all values. **The Earth Mover's distance** for the *Floor* datasets shows that the *PreTrain-Hyper-Denoise* configuration achieves the best performance, slightly better than the *Hyper-Denoise* configuration. For the *NoFloor* datasets again, the *Hyper-Denoise* configuration performs the worst, while the *PreTrain-Hyper-Denoise* configuration performs better. But for this metric, the *Floor* dataset configurations both outperform the *NoFloor* dataset configurations. **The Hausdorff distance** for the *Floor* datasets shows that the *PreTrain-Hyper-Denoise* configuration performed best, while the *Hyper-Denoise* configuration had a higher value. For the *NoFloor* datasets, the *Hyper-Denoise* configuration performed the worst, while the *PreTrain-Hyper-Denoise* configuration, in direct comparison, performed better. Also, the *Floor* dataset configurations both outperform the *NoFloor* dataset configurations. These differences in performance are shown separately in Figure 5.5. Taken together, these distance metrics show that pre-trained weights significantly improve performance when the floor is removed.

Pre-training consistently reduces average and maximum distances in unconnected scenes with multiple objects. However, the *Floor* and *NoFloor* dataset configurations differ significantly.

Evaluation Metric	Floor Hyper Denoise	Floor PreTrain Hyper Denoise	NoFloor Hyper Denoise	NoFloor PreTrain Hyper Denoise
Chamfer Distance	42.4771	42.5391	53.4981	37.8924
Earth Mover's Distance	0.1559	0.1543	0.4005	0.2884
Hausdorff Distance	0.3800	0.1744	0.9511	0.6296
F-Score Combined < 0.02 (%)	47.0438	46.6483	81.3003	81.9779
F-Score Precision < 0.02 (%)	87.3044	86.2587	89.8561	90.7172
F-Score Recall < 0.02 (%)	32.2331	32.0024	74.8304	75.5447
Average Nearest Neighbour (ANN)	0.0244	0.0241	0.0128	0.0119
Median Nearest Neighbour (MNN)	0.0242	0.0240	0.0095	0.0096

Table 5.3: Evaluation of the trained models with Floor and NoFloor datasets in different configurations, truncated to the fourth decimal place. In each row, the green cell represents the best and the red cell the worst value.

The F-Score metrics in Table 5.3 for the *Floor* datasets show that the *Hyper-Denoise* and *PreTrain-Hyper-Denoise* configurations perform similarly, with combined F-Score values of around 47%. However, for the *NoFloor* datasets, the *PreTrain-Hyper-Denoise* configuration achieved the best performance across all F-Score metrics (Combined, Precision, and Recall). The *Hyper-Denoise* configuration without pre-training had lower values but still higher ones than the *Floor* dataset configurations, especially for the Recall metric. This suggests that pre-trained weights combined with updated hyperparameters significantly improve both precision and recall in unconnected multi-object scenarios. This indicates that multi-object scenes (*NoFloor*) outperform single-object scenes (*Floor*) when considering the F-Score metrics (see Figure 5.5).

The Average and Median Nearest Neighbour (ANN and MNN) for the *Floor* datasets show that both configurations (*Hyper-Denoise* and *PreTrain-Hyper-Denoise*) perform similarly with values around 0.024. For the *NoFloor* dataset configurations, both configurations behave comparably in ANN and MNN. This suggests that whether pre-training is combined with updated hyperparameters or not, it does not significantly affect the density of the resulting point clouds. The *Floor* and *NoFloor* dataset configurations differ here by a factor of about 2. The significantly concentrated objects of the multi-object scene can also be seen in Figure 5.6. The *NoFloor* dataset configurations are likely to benefit from the fact that the area to be reconstructed with points is significantly smaller than in the *Floor* dataset configurations. Nevertheless, the same number of points should be added in all configurations. This increases the density of objects in multi-object scenes.

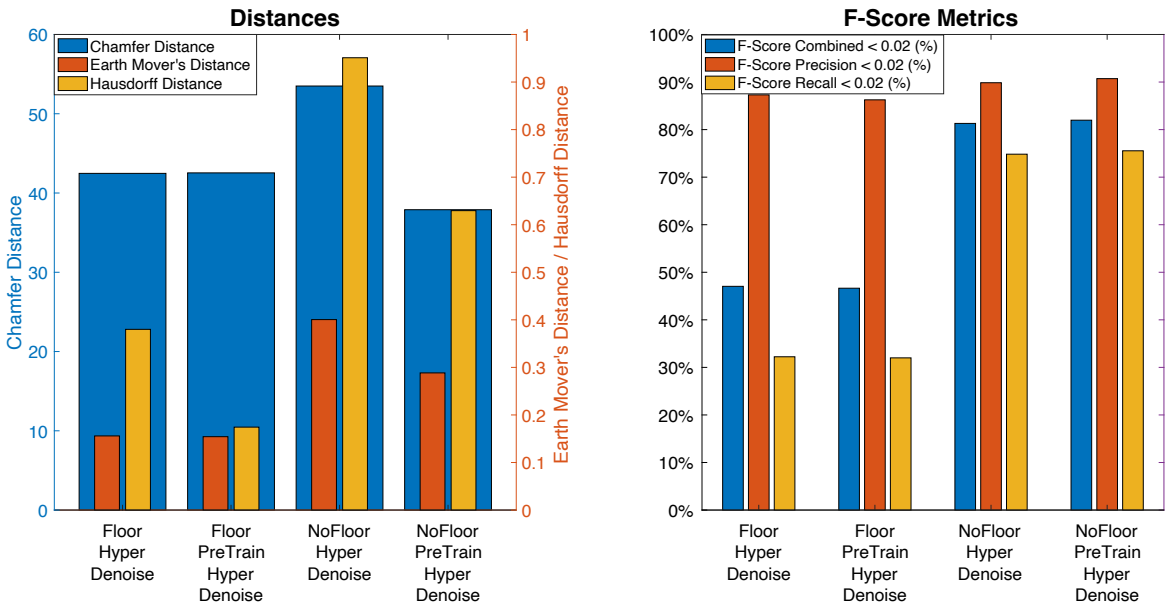

Figure 5.5: An illustration of the meaningful metrics from Table 5.3 of the Floor and NoFloor datasets. The left diagram shows the Chamfer Distance (blue) as well as the Earth Mover's (red) and the Hausdorff (orange) distance of the three training configurations. The bar chart on the left visualises the three F-score values in % (Combined, Precision, and Recall) of the three training configurations.

Artificial bridge formations can be observed in the *NoFloor* and multi-object scenes. While the primary objects are reconstructed with high density, bridge-like point formations are generated between these objects, as shown in Figure 5.6. This phenomenon could explain the significant discrepancy between the performance of the F-Score and the distance metrics. Since the density of these artificial bridges is low, their impact on a relative metric such as the F-Score is minimal. This hypothesis is supported by the fact that the ANN value for the *NoFloor* dataset configurations is higher than the MNN (see Table 5.3). This difference represents an unevenly distributed density within the point cloud. However, the distances of these generated bridges to the ground truth are likely substantial, which elevates the distance metric values.

Summarising, the evaluation results reveal the intricate and multifaceted relationship between dataset structure and reconstruction accuracy. Including pre-trained weights and updated hyperparameters improves performance, especially in floorless scenarios. The *NoFloor* datasets benefit significantly from the *PreTrain-Hyper-Denoise* configuration, demonstrating the effectiveness of pre-training and hyper-parameter optimization in handling unconnected multi-object scenes. However, the differences between the *Floor* and *NoFloor* datasets, as highlighted by the F-Scores and density metrics, suggest that the presence of a coherent connecting object can significantly impact the overall performance. Additionally, the presence of artificial bridging in the *NoFloor* and multi-object scenes, where bridge-like point formations are generated between objects, may explain the discrepancy between the F-Scores and distance metrics. While these bridges have minimal impact on F-Scores due to their low density, they significantly affect distance metrics, emphasizing the complex dynamics of point cloud reconstruction in different scenarios.

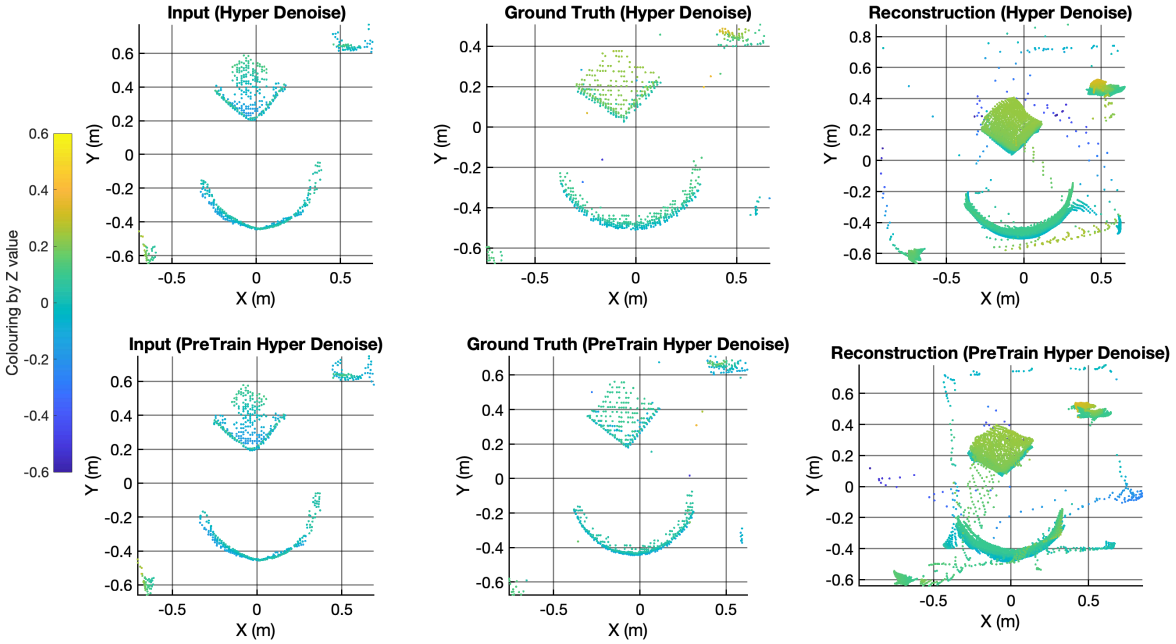

Figure 5.6: An example of the *Hyper-Denoise* and *PreTrain-Hyper-Denoise* training illustrated by the input point cloud, the ground truth point cloud, and the resulting reconstruction of both models.

5.4 Comparison with Interim Results

This section provides a detailed comparison between the *interim results* and the newly generated training configurations: *Hyper-Denoise*, *PreTrain-Denoise*, and *PreTrain-Hyper-Denoise*. By examining the performance across the same set of metrics in Table 5.4, this section aims to highlight the specific differences between the *interim results* and the other configurations, revealing the impact of each configuration on the overall results.

Evaluation Metric	Hyper Denoise	PreTrain Denoise	PreTrain Hyper Denoise	Interim Results
Chamfer Distance	39.8653	50.5123	40.5007	37.7250
Earth Mover's Distance	0.2186	0.2278	0.2131	1.0652
Hausdorff Distance	0.4176	0.3614	0.3741	0.3362
F-Score Combined < 0.02 (%)	63.0970	46.6730	62.1463	66.3716
F-Score Precision < 0.02 (%)	89.3146	72.3683	88.2144	78.3388
F-Score Recall < 0.02 (%)	52.2506	38.3299	51.1031	62.5904

Table 5.4: Evaluation of the trained models with all nine datasets combined in different configurations in comparison with the interim results, truncated to the fourth decimal place. Comparing the Chamfer, Earth Mover's, and Hausdorff distance as well as the F-score (Combined, Precision, and Recall). In each row, the green cell represents the best and the red cell the worst value.

The *interim results* present a unique performance profile compared to the final configurations (*Hyper-Denoise*, *PreTrain-Denoise*, and *PreTrain-Hyper-Denoise*). This analysis delves into the specific metrics and highlights the influence of point plane artefacts on the *interim results*.

The Chamfer Distance for the *interim results* is 37.7250, the best among all configurations. This suggests that the interim model produces point clouds that are, on average, closer to the ground truth. However, this might be misleading due to the dense point plane artefacts, which artificially lower the Chamfer Distance by placing many points close to the ground truth points.

The Earth Mover's distance for the *interim results* is 1.0652, significantly higher than the other configurations. This stark difference is a clear indication of the detrimental impact of the dense point plane artefacts. These artefacts, while they may be close to the ground truth points and lower the Chamfer Distance, increase the EMD because EMD measures the minimum cost of transforming one point cloud into another. The dense planes mean more points need to be moved, which inflates the EMD value. This highlights a significant flaw in the *interim results*.

The Hausdorff Distance for the *interim results* is 0.3362, again the best among all configurations. This metric measures the maximum distance from a point in one point cloud to the nearest point in the other. The dense point planes close to the ground truth help achieve a low maximum distance, as these artefacts ensure that no point is too far from the ground truth.

F-Score metrics for the *interim results* show a combined F-Score of 66.3716, the best among the configurations. This suggests high precision and recall due to the proximity of the dense artefacts to the ground truth. With a precision of 78.3388, the *interim results* perform worse than *Hyper-Denoise* and *PreTrain-Hyper-Denoise* but better than *PreTrain-Denoise*. This lower accuracy indicates that no reconstructed point was generated in the area around multiple ground truth points. The recall for the *interim results* is 62.5904, which is the highest. The dense point planes ensure that most reconstruction points have nearby ground truth points, boosting recall. However, this high recall does not reflect the actual quality of the reconstruction.

Summarising, the *interim results* display a mixed performance profile compared to the final configurations. The hypothesis is that dense point plane artefacts close to the ground truth significantly impact the metrics. These artefacts artificially improve the Chamfer Distance, Hausdorff Distance, and F-Score metrics while severely degrading the Earth Mover's Distance. The high EMD highlights the substantial cost required to transform the artefact-laden *interim results* into the ground truth. While the *interim results* may appear favourable in some metrics, they do not represent a genuinely accurate reconstruction due to the unwanted artefacts. The newly trained configurations, with updated hyperparameters and/or pre-training, provide a more reliable and accurate reconstruction of the point clouds.
Referring to Table 5.5, the average and median densities reveal the influence of artefacts on the interim results (see Section 5.2). The *Average 10 Nearest Neighbour* (ANN) and *Median 10 Nearest Neighbour* (MNN) metrics provide insight into the density and density concentration of the point clouds. The *interim results* exhibit the lowest ANN and MNN values (0.0116 and 0.0020, respectively) among all configurations, indicating an unusually high density of points. The discrepancy between the average and median shows a few very concentrated artefacts in these point clouds, significantly reducing the MNN distance as illustrated in Figure 5.2. While the ANN distance represents a dense point cloud, the MNN distance can show an unbalanced distribution. Similarly, for the *Average 100 Nearest Neighbour* (ANN) and *Median 100 Nearest Neighbour* (MNN) metrics, the *interim results* again show the lowest

Evaluation Metric	Hyper Denoise	PreTrain Denoise	PreTrain Hyper Denoise	Interim Results
Average 10 Nearest Neighbour (ANN)	0.0190	0.0205	0.0188	0.0116
Median 10 Nearest Neighbour (MNN)	0.0179	0.0200	0.0177	0.0020
Average 100 Nearest Neighbour (ANN)	0.0544	0.0529	0.0541	0.0324
Median 100 Nearest Neighbour (MNN)	0.0516	0.0522	0.0513	0.0059

Table 5.5: Evaluation of the trained models with all nine datasets combined in different configurations, truncated to the fourth decimal place. Comparing the ANN and MNN of the 10 and 100 nearest neighbours. In each row, the green cell represents the lowest and the red cell the highest value.

values (0.0324 and 0.0059, respectively), further highlighting the presence of dense, large artefacts. In contrast, the *PreTrain-Denoise* configuration consistently shows the highest ANN and MNN values, suggesting a less dense point cloud. No significant discrepancy between ANN and MNN can be recognised in all the newly trained configurations. Likewise, no point plane artefacts were detected during manual observation. The lack of discrepancy between ANN and MNN now supports this observation. Compared to the other configurations, the significantly lower ANN and MNN values in the interim results indicate the strong influence of artefacts. These dense point planes lower the average and median distances between nearest neighbours, creating an illusion of higher point cloud density. This finding is crucial as it highlights the need to be cautious of artefacts that can skew the metrics, a key consideration in this work on point cloud reconstruction. Overall, the average and median density metrics effectively capture the impact of dense point plane artefacts, demonstrating their influence on the point cloud's structure and the resulting evaluation metrics.

5.5 Discussion

The results presented in the previous section are systematically categorized in the following discussion. This section addresses the initial research questions of this study and evaluates the hypotheses formulated after analyzing the interim results. Each finding is thoroughly examined to provide a comprehensive understanding of the outcomes. The analysis of the interim results in Section 4.5.3 led to several hypotheses, each of which will now be discussed in turn to assess their validity.

The artefacts in the form of dot planes potentially contribute to the high recall and, consequently, the high F-score and Chamfer Distance, but they do not reconstruct the input point cloud accurately. Although the interim results largely achieved the best performance in distance and F-score metrics, a notable discrepancy exists between the ANN and MNN for the 10 and 100 nearest neighbours. This discrepancy indicates a strong concentration of points in a small area (Table 5.5). Consequently, these artefacts are present in a measurable form, highlighting a critical area for further investigation. To address the hypothesis regarding artefacts,

it was concluded that increasing the sample size through pre-training with the 3DCompletion Shapenet dataset would help. The pre-trained weights from this noise-free dataset are then used to train on the noisy datasets. Table 5.4 clearly shows that the *PreTrain-Denoise* configuration of the *combined* dataset does not exhibit a significant discrepancy between ANN and MNN. As a result, it neither visually nor metrically shows detectable artefacts. Additionally, the *PreTrain-Denoise* configuration achieves a substantial improvement in Chamfer Distance, reducing it to 50.5123 compared to the original EAE-Net output of 134.5. Therefore, it can be concluded that the hypothetically implemented pre-training achieved the expected effect.

The difference in density within the reconstructed point clouds is due to noise in the input data, causing the model to struggle with accurate reconstruction. Since the discrepancy between ANN and MNN is influenced by both artefacts and other variations in density within the point cloud, it is important to clarify that the different density distributions could not be measured independently of the artefacts. However, the apparent discrepancy between ANN and MNN in the interim results (Table 5.4) indicates an imbalanced distribution across the point clouds. To tackle the hypothesis that noise causes reconstruction issues, it was concluded that the architecture of the PCTMA-Net needs to be more complex to handle the noisy data effectively, despite the increased computational requirements. This is evidenced by the fact that the *combined* dataset's *Hyper-Denoise* configuration achieved a Chamfer Distance of 39.8653, which is close to the 37.725 obtained in the interim results. Unlike the interim results, the *Hyper-Denoise* configuration does not show a significant discrepancy between ANN and MNN, indicating a balanced distribution of points across the point cloud (Table 5.5). Updating the hyperparameters had the desired effect, yielding performance that even surpassed the PreTrain-Denoise configuration in almost all measured metrics.

The significantly different results compared to the original PCTMA-Net paper are due to the smaller sample size and the presence of noise in the datasets, as opposed to the larger, noise-free 3DCompletion Shapenet dataset. These led to the hypothesis that pre-training on a larger, noise-free dataset like 3DCompletion Shapenet, in combination with updated hyperparameters, would enhance the network's performance on the smaller, noisy datasets. The *PreTrain-Hyper-Denoise* configuration of the *combined* datasets achieves results comparable to the interim results in both distance and F-score metrics, significantly outperforming the original EAE-Net output. Notably, the model achieved markedly better results in Earth Mover's Distance and F-score precision, with an almost equal distribution of ANN and MNN over the 10 and 100 nearest neighbours. While the *Hyper-Denoise* configuration slightly outperformed Earth Mover's Distance and F-score precision, the difference is negligible. Therefore, it can be concluded that the actions taken based on the hypotheses met expectations, thereby validating the hypotheses.

The introductory question of this work was: How effectively can the PCTMA-Net interpolate missing regions in point clouds derived from sonar data? The substantial reduction in Chamfer Distance from the original 134.5 to 40.5007 demonstrates the effectiveness of the PCTMA network in the *PreTrain-Hyper-Denoise* configuration. Additionally, the *Hyper-Denoise* and *PreTrain-Denoise* configurations achieved nearly the same or only slightly higher values. This indicates that the PCTMA-Net, with the adjustments described in the methodology (Section 4.5.3, can achieve satisfactory results and successfully interpolate the missing regions.

How does the PCTMA-Net handle sonar noise and artefacts? The types of sonar noise described in the theory section, especially speckle noise and sonar artefacts, significantly affect the performance of the EAE-Net [Wan+23]. However, the metrics recorded in this thesis do not clearly indicate the specific influence each type of noise has on the models. Despite this, the overall performance and improvement of the various configurations across all metrics suggest that the modified PCTMA-Net can effectively handle the input noise from the EAE-Net. It is important to note that pre-training seems to have less influence than the changed hyperparameters when considering the *combined* datasets in all metrics from Table 5.4. The *PreTrain-Denoise* configuration does not achieve the best value in any metric and shows poorer results in Chamfer Distance, and all F-scores metrics. In summary, the modified PCTMA network appears to effectively handle sonar noise and potential artefacts, even though the individual influences cannot be clearly distinguished.

Do single and multi-object scenes pose different challenges for PCTMA-Net? Table 5.3 reveals apparent discrepancies between the *Floor* and *NoFloor* dataset configurations. The *Hyper-Denoise* configurations of the respective datasets show underperformance in the *NoFloor* datasets, while the F-score metrics indicate a significantly higher recall for the *NoFloor* datasets. This behaviour is related to the significantly different requirements of the multi-object scenes in the *NoFloor* datasets. The higher recall is attributed to the increased density, as points are primarily generated on the objects to be reconstructed, resulting in a closer and denser match to the ground truth points. However, the artificial bridging of generated points impacts the distance metrics significantly. Although the artificial bridges are statistically less relevant within the *PreTrain-Hyper-Denoise* configuration, they remain visible in the current pipeline and can be interpreted as errors. These artificially generated bridge-like constructions originate from the Morphing-Atlas-Based Point Generator Network. As Lin et al. stated and Section 3.2.3 explains, "All charts are concatenated to form a complete shape" [Lin+21]. This point generator inherently counteracts a multi-object scene, as demonstrated in Figure 5.6. Accordingly, it can be clearly stated that single and multi-object scenes achieve different challenges and results. However, the reconstruction in both data sets can be seen as a significant improvement on the original EAE-Net results.

To summarize , the PCTMA-Net achieves significant improvements across all configurations and datasets compared to the EAE-Net results. The PreTrain-Hyper-Denoise configuration, while not achieving the best results in any single metric, proves to be the most consistent option across all metrics. Its results are negligibly close to the best outcomes, with no underperformance outliers in any metric. However, it is also evident that the Hyper-Denoise configuration achieves similar results, except for the Hausdorff distance.
Underscoring the importance of the Hausdorff distance metric in the context of point cloud reconstruction, it serves as a crucial tool for calculating the worst-case scenario of the conversion from one point cloud to another. By focusing on the most significant deviation, this metric plays a vital role in ensuring the accurate reconstruction of even the most challenging parts of the point cloud. A significantly better Hausdorff distance, therefore, indicates that the model can handle extreme differences and outliers more effectively, leading to a more robust and consistent performance [HKR93].
In this context, the PreTrain-Hyper-Denoise configuration's superior performance in the Hausdorff distance suggests it better manages the worst-case scenarios and outliers, providing a more reliable reconstruction in challenging areas. Thus, while the Hyper-Denoise configuration also offers overall consistency, the PreTrain-Hyper-Denoise configuration excels in being consistent and handling the most challenging aspects of the point cloud reconstruction.

Chapter 6

Conclusion

6.1 Limitations

While this study has made significant strides and yielded promising results, it's crucial to recognize the limitations that could potentially affect the overall conclusions and generalizability of the findings. These constraints serve as guideposts for areas where the methodology and experimental setup could be further refined and optimized. The following points delve into the specific limitations observed during this work.

1. A notable aspect is the absence of completely modified scenes in the synthetic Blender dataset generation. This means that the same experimental environment was used, with only the number, position, and grouping of objects being altered. The creation of new datasets with significantly different environments could potentially enhance the model for real-world applications, opening up avenues for future research.
2. The trained models were not tested with new, unknown environments and objects. The test sets of the models were tested from the existing datasets. However, pretraining with the 3D Completion Shapenet could enable the reconstruction of objects that were not present in the original experimental environment. Since the 3D Completion Shapenet consists of eight different classes of samples: *["Plane", "Cabinet", "Car", "Chair", "Lamp", "Couch", "Table", "Watercraft"]* [Cha+15]. However, this was not evaluated in this work and, therefore, cannot be confirmed.
3. No attempt was made to improve the performance of the EAE-Net further. Only minor changes were made, such as increasing the number of epochs. Apart from additional metrics, the other architecture remained unchanged. However, improved input point clouds could further enhance the final result.
4. The output ground truth of the EAE-Net has a higher resolution than the input ground truth of the PCTMA-Net (average of 4354 points), but the *dataSet.py* of the PCTMA-Net forced a downsampling to 2048 points. Changing this value led to considerable problems and could not be solved due to a lack of time. However, an increased resolution of the ground truth could further improve the results.
5. For the training of the PCTMA-Net, it is crucial to have ground truth point clouds for the samples used in the datasets. Relying solely on real-world data is insufficient, as obtaining real ground truth point clouds is typically not feasible. Therefore, the generation of synthetic ground truth point clouds is necessary for this architecture.
6. The *Floor* and *NoFloor* datasets were not trained with a configuration without changed hyperparameters like the *AllData* dataset. It would have been interesting to recognize the different effects here. However, this was no longer possible due to time and, ultimately, organizational reasons.

7. The artificially generated bridges in the *NoFloor* dataset configurations could be denoised by the significantly different densities of the points to free these results from the disadvantages of the Morphing-Atlas-Based Point Generator Network. In addition to the primary objects, a denoising method other than the DBSCAN algorithm would be necessary here. These small objects are not so densely completed and are sometimes reconstructed in the background.

6.2 Future Outlook

In addition to the aforementioned limitations of this work, another approach also offers scope for further pursuit of this topic in the future. A sonar image's pixel colouring represents the sonar beams' received intensity. This intensity indicates the distance [AN13], but also the type of material hit [MZ95]. Different materials result in slightly different amplitudes and a slightly different received intensity [MZ95]. This thesis presents a pipeline that uses the depth information of a sonar image to create and reconstruct 3D point clouds. The information about the material, i.e. the intensity adjusted for distance, needs to be recovered. In order to recreate a 3D sonar, it would be necessary to feed these intensity values back into the reconstructed point cloud. This would represent the depth information as well as the material information in three-dimensional space. One idea to realise this is using NeuSG (Neural Implicit Surface Reconstruction with 3D Gaussian Splatting Guidance) [CLL23].

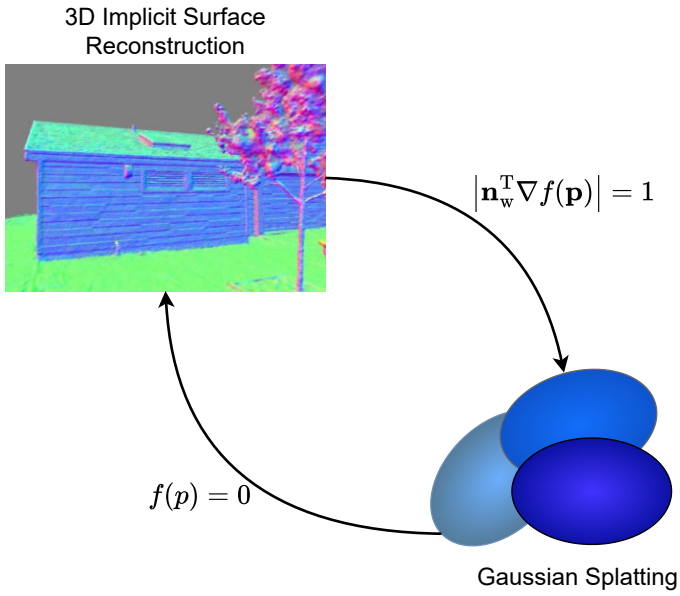

Figure 6.1: Based on [CLL23], this figure illustrates the optimization of implicit surface reconstruction using Gaussian Splatting, demonstrating how this method can incorporate intensity values into the 3D space.

The paper "NeuSG: Neural Implicit Surface Reconstruction with 3D Gaussian Splatting Guidance" by Zhihao Wang, Jiabin Li, Kai Zhang, Zheng Chen, and Jingyi Yu proposes a novel approach to 3D surface reconstruction that combines neural implicit models with 3D Gaussian splatting to recover highly detailed surfaces from multi-view images [CLL23]. Traditional methods often rely on depth maps or point clouds, which can lack fine detail. The proposed method, NeuSG, integrates 3D Gaussian splatting, which generates dense point clouds, with

neural implicit models to enhance surface detail. The authors introduce a scale regularizer to align the points generated by Gaussian splatting with the surface and use normal priors from neural implicit models to refine these points. Experiments on the Tanks and Temples dataset show that NeuSG significantly outperforms existing methods in terms of surface reconstruction quality, demonstrating its ability to produce complete surfaces with intricate details efficiently[CLL23]. Gaussian splatting is a technique used to generate dense point clouds by representing 3D scenes with 3D Gaussians. Each Gaussian is defined by a covariance matrix and a centre point, representing the Gaussian distribution's mean. The advantage of Gaussian splatting lies in its ability to produce highly detailed and dense point clouds, which are crucial for regions with intricate details. However, the naive use of Gaussian splatting can result in points not aligning perfectly with the surface, as they are often located inside the surface. To address this, the authors propose a scale regularizer that flattens the 3D Gaussians into thin shapes, pulling the centre points closer to the surface. This refinement process ensures that the dense point clouds generated by Gaussian splatting provide accurate geometric constraints for surface reconstruction[CLL23]. The 3D Gaussians would, therefore, create a closed surface, which, as in reality, would now have to be coloured with the intensity values of the sonar images. Since the colouring of the sonar images is a combination of distance and intensity, a further network would be necessary here, which does not estimate the elevation angle, as the EAE-Net does, but the nature of the material at a particular position. This material information would then have to be transferred to the 3D Gaussians in combination with the generation of the depth information. As a result, a 3D intensity map with a closed surface could be generated, thus imitating a 3D sonar as in Figure 6.1.

6.3 Summary

This thesis's primary objective was to develop and rigorously evaluate a novel approach for dense point cloud reconstruction. The intricate challenges this work aimed to tackle are those associated with reconstructing noisy, complex single and multi-object scenes. These challenges include dealing with occlusions, handling varying object shapes and sizes, and managing the high density of points in the cloud. This work focussed on overcoming these challenges in the demanding context of underwater environments and their perception using a sonar sensor. By systematically experimenting with various configurations and parameters, this research has uncovered several key findings that advance the state of knowledge in 3D reconstruction using a transformer network and have significant practical implications for real-world applications. Through a series of experiments and new synthetically generated datasets, the research demonstrated that adjusting hyperparameters significantly enhances the accuracy and quality of point cloud reconstructions while avoiding point plane artefacts. It was also found that pre-training on a diverse dataset, such as the 3D Completion Shapenet [Cha+15], is crucial for improving performance in scenarios with free-floating objects. Despite these advancements, the research also highlighted several limitations. The synthetic datasets used were confined to a consistent experimental environment, and the models were not tested with entirely new and unknown environments. Future work should incorporate more varied training datasets, explore advanced denoising techniques, and test the models in more diverse and realistic settings. By addressing these limitations, further improvements can be made to enhance the applicability and accuracy of point cloud reconstruction in underwater and other challenging environments. In conclusion, this thesis provides significant contributions to point cloud reconstruction, offering methodological advancements and practical insights. The findings lay a solid foundation for future research and development, aiming to achieve more accurate and reliable 3D reconstructions in complex environments.

Appendix A

Appendix

A.0.1 Preprocessing

Database restructuring This pseudocode illustrates the core functionality of the `copy_h5_to_database.py` script, which organizes all provided datasets into the structure required by the PCTMA-Net.

Algorithm 4 Copy Files and Counterparts

```
function CopyFilesAndCounterparts(input_dir, train_dir, val_dir, test_dir)
    Initialize counters: train_counter, val_counter, test_counter ← 1, 1, 1
    for all directories and files in input_dir do
        if directory name is "gt" then
            for all .h5 files in directory do
                Determine file number from the file name
                if file_number < 1294 then
                    dest_dir ← train_dir
                    Increment train_counter
                else if file_number < 1456 then
                    dest_dir ← val_dir
                    Increment val_counter
                else
                    dest_dir ← test_dir
                    Increment test_counter
                    if corresponding particle file exists then
                        Copy particle file to dest_dir
                    end if
                    Continue to the next file
                end if
                Copy GT file to dest_dir
                if corresponding particle file exists then
                    Copy particle file to dest_dir
                end if
            end for
        end if
    end for
end function
```

The script 4 organizes point cloud HDF5 files from a specified input directory into training, validation, and test directories. It processes files located in the "gt" (ground truth) directory,

assigns each file to one of the three sets based on its numeric filename, and renames them sequentially. It copies both the ground truth and corresponding particle files to their new locations. Files with numbers less than 1294 go to the training set, numbers between 1294 and 1456 go to the validation set, and numbers 1456 and above go to the test set. For test files, only the particle files are processed and copied.

Normalization The `bound_box_test.py` script, shown in script 5, normalizes point cloud data stored in HDF5 files by applying a specified scale factor and translation. For each HDF5 file in a given directory, it reads the point cloud data, normalizes it using the provided scale factor and translation, and then saves the normalized data back to the file. The normalization process scales and translates the point cloud to ensure uniformity across the dataset. A `dataset_key` in an HDF5 file is a string that identifies a specific dataset within the file, allowing for the storage and retrieval of structured data, such as arrays or point cloud data, under that key.

Algorithm 5 Normalize Point Clouds

```
function NormalizePointCloud(point_cloud, scale_factor, translation)
    return point_cloud × scale_factor + translation
end function
function ProcessFile(file_path, scale_factor, translation, dataset_key)
    Open HDF5 file in read/write mode
    if dataset_key exists in file then
        Load point cloud from dataset_key
        normalized_point_cloud ← NormalizePointCloud(point_cloud, scale_factor, translation)
        Delete existing dataset_key
        Create new dataset_key with normalized_point_cloud
        Print "Normalized" file_path
    end if
    Close HDF5 file
end function
function ProcessDirectory(directory, dataset_key)
    avg_scale_factor ← [0.25, 0.25, 0.25]
    avg_translation ← [0.25, 0.25, 0.25]
    for all .h5 files in directory do
        file_path ← join(directory, filename)
        ProcessFile(file_path, avg_scale_factor, avg_translation, dataset_key)
    end for
end function
ProcessDirectory(test_directory, dataset_key)
```

Centering the Point Cloud Data This pseudo-code 6 outlines the process of centring point clouds from the EAE-Net output relative to the ground truth point clouds in order to avoid reconstruction errors, made in the `center_to_gt.py`. The `CenterPointCloud` function subtracts the reference centroid from the point cloud. The centroid of the ground truth point cloud is calculated using the `GetCentroid` function. Using the calculated centroid, the `ProcessPair` function centres both the ground truth point cloud and the corresponding

Algorithm 6 Center Point Clouds

```
function CenterPointCloud(point_cloud, reference_centroid)
    return point_cloud - reference_centroid
end function
function GetCentroid(file, dataset_key)
    Open file in read mode
    if dataset_key exists then
        Load point cloud from dataset_key
        return mean of point cloud (centroid)
    end if
    return None
end function
function ProcessPair(reference_file, target_file, dataset_key)
    reference_centroid ← GetCentroid(reference_file, dataset_key)
    if reference_centroid is not None then
        Open reference file in read/write mode
        if dataset_key exists then
            Load and center reference point cloud
            Save centred point cloud
        end if
        Open target file in read/write mode
        if dataset_key exists then
            Load and center target point cloud
            Save centred point cloud
        end if
    end if
end function
function ProcessDirectory(directory, dataset_key)
    Get list of (reference_file, target_file) pairs in directory
    for all (reference_file, target_file) in list do
        if target_file exists then
            ProcessPair(reference_file, target_file, dataset_key)
        end if
    end for
end function
Example Usage:
directory ← "./"
dataset_key ← 'point_cloud'
ProcessDirectory(directory, dataset_key)
```

particle point cloud. The `ProcessDirectory` function iterates through the root directory, identifying and processing pairs of ground truth and particle HDF5 files to ensure alignment.

Removing Empty Point Clouds The pseudo-code 7 outlines a process for identifying and deleting empty point cloud datasets in HDF5 files. The `IsH5FileToBeDeleted` function checks if an HDF5 file contains empty datasets or if the point cloud dataset is a 1D array. If any of these conditions are true, then the file will be marked for deletion. The `FindAndDeleteH5Files` function scans the specified directory for HDF5 files, using the `IsH5FileToBeDeleted` function to identify files to be removed. If a file is flagged, its corresponding point cloud will be deleted. This procedure ensures that only valid point clouds remain in the dataset, preventing errors in the PCTMA network.

Algorithm 7 Find and Delete Empty Point Clouds

```
function IsH5FileToBeDeleted(h5_file_path)
    Open file in read mode
    for all datasets in file do
        if dataset is empty or point_cloud is 1D array then
            return True
        end if
    end for
    return False
    Handle any exceptions by printing error and returning False
end function
function FindAndDeleteH5Files(input_dir)
    Get list of all .h5 files in input_dir
    for all files in list do
        if IsH5FileToBeDeleted(file_path) then
            Print "Deleting file: " file_path
            Delete file_path
            Determine counterpart file_path
            if counterpart_file exists then
                Delete counterpart_file_path
            end if
        end if
    end for
end function
Example Usage:
FindAndDeleteH5Files(input_directory)
```

Creating Sample Lists for PCTMA-Net The pseudocode 8 describes the `write_list.py` function, `CreateList`, which creates a list of sample files for the PCTMA-Net. The function constructs a path to a data subdirectory, initialises an empty list of file numbers and checks if the subdirectory exists. If the subdirectory exists, it will iterate through all its files, checking if the file extension is `.h5`. It extracts the numerical part of the file name for each valid file and adds it to the list. After collecting all file numbers, the list will be sorted numerically. The function then opens a list file for writing and iterates through the sorted list of file numbers, writing each number with an optional prefix to the list file. Once all the numbers have been written, the list file is closed. This process is repeated for different directories to create lists for test, training and validation samples, with appropriate prefixes added for the training

Algorithm 8 Create List of Sample Files

```
function CreateList(data_dir, subfolder, list_file_path, prefix)
    Construct path to data partial directory
    Initialize empty list for file numbers
    if partial directory exists then
        for all files in directory do
            if file extension is ".h5" then
                Extract numerical part of the filename
                Add number to list
            end if
        end for
    end if
    Sort list of file numbers numerically
    Open list file for writing
    for all number in list do
        Write prefix and number to list file
    end for
    Close list file
end function
Example Usage:
Create a list for the test directory
Create a list for the train directory with a prefix
Create a list for the validation directory with a prefix
```

and validation samples. The generated lists allow the PCTMA-Net to access and process the samples systematically during its operation.

A.0.2 GitHub Repository

The code and data used in this thesis are available in the following GitHub repository:
https://github.com/phknestel/PCTMA_Net_Sonar

This repository includes:

- Scripts and the implemented architecture of the PCTMA-Net.
- Notebooks used for the EAE-Net are stored in *elevation_net_notebooks*.
- The preprocessing scripts are stored in the *data* folder.
- The DBSCAN script is stored in the *data* folder.
- The postprocessing scripts are stored in the *Finished_models_gen_files* folder.

Due to their large file sizes, the generated and utilized datasets are not included in the repository. However, they can be made available upon request.
For any issues or questions regarding the repository, please contact me at: philipp.knestel@tum.de

Bibliography

[Alz+21] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamarıa, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of big Data* 8 (2021), pp. 1–74.

[Aok+19] Aoki, Y., Goforth, H., Srivatsan, R. A., and Lucey, S. "Pointnetlk: Robust & efficient point cloud registration using pointnet". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 7163–7172.

[AN13] Aykin, M. D. and Negahdaripour, S. "Forward-look 2-D sonar image formation and 3-D reconstruction". In: *2013 OCEANS - San Diego*. 2013, pp. 1–10. DOI: 10.23919/OCEANS.2013.6741270.

[BA72] Bansal, P. and Ardell, A. J. "Average nearest-neighbor distances between uniformly distributed finite particles". In: *Metallography* 5.2 (1972), pp. 97–111.

[Ble24] Blender. *Diffuse BSDF - Blender 4.1 Manual*. Apr. 2024. URL: https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/diffuse.html.

[Bra+11] Brahim, N., Guériot, D., Daniel, S., and Solaiman, B. "3D reconstruction of underwater scenes using DIDSON acoustic sonar image sequences through evolutionary algorithms". In: *OCEANS 2011 IEEE-Spain*. IEEE. 2011, pp. 1–6.

[BPC18] Bruna, M. A., Pate, D. J., and Cook, D. "Synthetic aperture sonar speckle noise reduction performance evaluation". In: *The Journal of the Acoustical Society of America* 143.3_Supplement (2018), pp. 1856–1856.

[Cha+15] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[Che+17] Chen, D., Chu, X., Ma, F., and Teng, X. "A variational approach for adaptive underwater sonar image denoising". In: *2017 4th International Conference on Transportation Information and Safety (ICTIS)*. 2017, pp. 1177–1181. DOI: 10.1109/ICTIS.2017.8047920.

[CLL23] Chen, H., Li, C., and Lee, G. H. "Neusg: Neural implicit surface reconstruction with 3d gaussian splatting guidance". In: *arXiv preprint arXiv:2312.00846* (2023).

[Che+19] Chen, W., Gu, K., Lin, W., Xia, Z., Le Callet, P., and Cheng, E. "Reference-Free Quality Assessment of Sonar Images via Contour Degradation Measurement". In: *IEEE Transactions on Image Processing* 28.11 (2019), pp. 5336–5351. DOI: 10.1109/TIP.2019.2910666.

[Che+22] Chen, W., Liang, H., Chen, Z., Sun, F., and Zhang, J. "Improving object grasp performance via transformer-based sparse shape completion". In: *Journal of Intelligent & Robotic Systems* 104.3 (2022), p. 45.

[Chu+01] Chu, D., Baldwin, K., Foote, K., Li, Y., Mayer, L., and Melvin, G. "Multibeam sonar calibration: target localization in azimuth". In: *MTS/IEEE Oceans 2001. An Ocean Odyssey. Conference Proceedings (IEEE Cat. No.01CH37295)*. Vol. 4. 2001, 2506–2510 vol.4. DOI: 10.1109/OCEANS.2001.968395.

[DLH19a] DeBortoli, R., Li, F., and Hollinger, G. A. "Elevatenet: A convolutional neural network for estimating the missing dimension in 2d underwater sonar images". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 8040–8047.

[DLH19b] DeBortoli, R., Li, F., and Hollinger, G. A. "ElevateNet: A Convolutional Neural Network for Estimating the Missing Dimension in 2D Underwater Sonar Images". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 8040–8047. DOI: 10.1109/IROS40897.2019.8968121.

[Der16] Derczynski, L. "Complementarity, F-score, and NLP Evaluation". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Ed. by Calzolari, N., Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 261–266. URL: https://aclanthology.org/L16-1040.

[DY11] Dosselmann, R. and Yang, X. D. "A comprehensive assessment of the structural similarity index". In: *Signal, Image and Video Processing* 5 (2011), pp. 81–91.

[Edw+19] Edwards, M., Levy, J., Baghdady, M. M., and Sternlicht, D. D. "100 Years of Seafloor Discoveries and the Technologies that Made them Possible". In: *AGU Fall Meeting Abstracts*. Vol. 2019. Dec. 2019, OS41A-08, OS41A–08.

[Est+96] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[FSG17] Fan, H., Su, H., and Guibas, L. J. "A point set generation network for 3d object reconstruction from a single image". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[FRA21] Franchi, M., Ridolfi, A., and Allotta, B. "Underwater navigation with 2D forward looking SONAR: An adaptive unscented Kalman filter-based strategy for AUVs". In: *Journal of Field Robotics* 38.3 (2021), pp. 355–385.

[Gev+20] Geva, M., Schuster, R., Berant, J., and Levy, O. "Transformer feed-forward layers are key-value memories". In: *arXiv preprint arXiv:2012.14913* (2020).

[Goo07] Goodman, J. W. *Speckle phenomena in optics: theory and applications*. Roberts and Company Publishers, 2007.

[Gri+20] Griffin, R. A., Jones, R. E., Lough, N. E., Lindenbaum, C. P., Alvarez, M. C., Clark, K. A., Griffiths, J. D., and Clabburn, P. A. "Effectiveness of acoustic cameras as tools for assessing biogenic structures formed by Sabellaria in highly turbid environments". In: *Aquatic Conservation: Marine and Freshwater Ecosystems* 30.6 (2020), pp. 1121–1136.

[Gro+18] Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. "A papier-mâché approach to learning 3d surface generation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 216–224.

[Han+18] Han, X.-F., Jin, J. S., Wang, M.-J., and Jiang, W. "Iterative guidance normal filter for point cloud". In: *Multimedia Tools and Applications* 77 (2018), pp. 16887–16902.

[HRR19] Hermosilla, P., Ritschel, T., and Ropinski, T. “Total denoising: Unsupervised learning of 3D point cloud cleaning”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 52–60.

[HK16] Huang, T. A. and Kaess, M. “Incremental data association for acoustic structure from motion”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1334–1341.

[HLY20] Huang, Y., Li, W., and Yuan, F. “Speckle noise reduction in sonar image based on adaptive redundant dictionary”. In: *Journal of marine science and engineering* 8.10 (2020), p. 761.

[Hur+14] Hurtós, N., Nagappa, S., Palomeras, N., and Salvi, J. “Real-time mosaicing with two-dimensional forward-looking sonar”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 601–606.

[HKR93] Huttenlocher, D., Klanderman, G., and Rucklidge, W. “Comparing images using the Hausdorff distance”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.9 (1993), pp. 850–863. DOI: 10.1109/34.232073.

[Iqb22] Iqbal, K. *H5 File - Hierarchical Data Format 5 file*. Jan. 2022. URL: https://docs.fileformat.com/misc/h5/.

[Jav+17] Javaheri, A., Brites, C., Pereira, F., and Ascenso, J. “Subjective and objective quality evaluation of 3D point cloud denoising algorithms”. In: *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 2017, pp. 1–6. DOI: 10.1109/ICMEW.2017.8026263.

[Kim+20] Kim, B., Kim, J., Cho, H., Kim, J., and Yu, S.-C. “AUV-Based Multi-View Scanning Method for 3-D Reconstruction of Underwater Object Using Forward Scan Sonar”. In: *IEEE Sensors Journal* 20.3 (2020), pp. 1592–1606. DOI: 10.1109/JSEN.2019.2946587.

[KK16] Kleeman, L. and Kuc, R. “Sonar sensing”. In: *Springer handbook of robotics* (2016), pp. 753–782.

[Kne+] Knestel, P., Wang, Y., Reeb, N., Karimi, N., Walter, F., Tsuchiya, H., Ota, J., Asama, H., An, Q., Knoll, A., et al. “Completion of 3D Point Clouds Derived from 2D Sonar Images Using PCTMA-Net”. In: ().

[Kuc01] Kuc, R. “Pseudo-amplitude scan sonar maps”. In: *IEEE Transactions on Robotics and Automation* 17.5 (2001), pp. 767–770.

[Kuc07] Kuc, R. “Biomimetic Sonar and Neuromorphic Processing Eliminate Reverberation Artifacts”. In: *IEEE Sensors Journal* 7.3 (2007), pp. 361–369. DOI: 10.1109/JSEN.2006.890126.

[Li+21] Li, X., Qian, W., Xu, D., and Liu, C. “Image segmentation based on improved unet”. In: *Journal of Physics: Conference Series*. Vol. 1815. 1. IOP Publishing. 2021, p. 012018.

[Lin+21] Lin, J., Rickert, M., Perzylo, A., and Knoll, A. “PCTMA-Net: Point cloud transformer with morphing atlas-based point generation network for dense point cloud completion”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5657–5663.

[LH21] Luo, S. and Hu, W. “Score-based point cloud denoising”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 4583–4592.

[MZ95] MacKay, M. and Zisk, S. *Radar and Sonar - a primer for the geophysically-challenged | Wat on Earth*. May 1995. URL: https://uwaterloo.ca/wat-on-earth/news/radar-and-sonar-primer-geophysically-challenged.

[Mad+17] Maddern, W., Pascoe, G., Linegar, C., and Newman, P. "1 year, 1000 km: The oxford robotcar dataset". In: *The International Journal of Robotics Research* 36.1 (2017), pp. 3–15.

[Mai+17] Mai, N. T., Woo, H., Ji, Y., Tamura, Y., Yamashita, A., and Asama, H. "3-d reconstruction of underwater object based on extended kalman filter by using acoustic camera images". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 1043–1049.

[MAG18] Menna, F., Agrafiotis, P., and Georgopoulos, A. "State of the art and applications in archaeological underwater 3D recording and mapping". In: *Journal of Cultural Heritage* 33 (2018), pp. 231–248.

[Mid67] Middleton, D. "A Statistical Theory of Reverberation". In: *The Journal of the Acoustical Society of America* 42.5_Supplement (1967), pp. 1200–1200.

[Mil+21] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[MM99] Mitchell, A. and Mitchell, A. *The ESRI guide to GIS analysis*. Vol. 1. ESRI press Redlands, CA, 1999.

[MD15] Murty, M. N. and Devi, V. S. *Introduction to pattern recognition and machine learning*. Vol. 5. World Scientific, 2015.

[Neg13] Negahdaripour, S. "On 3-D Motion Estimation From Feature Tracks in 2-D FS Sonar Video". In: *IEEE Transactions on Robotics* 29.4 (2013), pp. 1016–1030. DOI: 10.1109/TRO.2013.2260952.

[OMa+19] O'Mahony, N., Campbell, S., Carvalho, A., Krpalkova, L., Riordan, D., and Walsh, J. "3D vision for precision dairy farming". In: *IFAC-PapersOnLine* 52.30 (2019), pp. 312–317.

[PW09] Pele, O. and Werman, M. "Fast and robust Earth Mover's Distances". In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 460–467. DOI: 10.1109/ICCV.2009.5459199.

[Pen+20] Peng, Y., Chang, M., Wang, Q., Qian, Y., Zhang, Y., Wei, M., and Liao, X. "Sparse-to-Dense Multi-Encoder Shape Completion of Unstructured Point Cloud". In: *IEEE Access* 8 (2020), pp. 30969–30978. DOI: 10.1109/ACCESS.2020.2973003.

[QKG23] Qadri, M., Kaess, M., and Gkioulekas, I. "Neural Implicit Surface Reconstruction using Imaging Sonar". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1040–1047. DOI: 10.1109/ICRA48891.2023.10161206.

[Qi+17a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[Qi+17b] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in neural information processing systems* 30 (2017).

[Rak+20] Rakotosaona, M.-J., La Barbera, V., Guerrero, P., Mitra, N. J., and Ovsjanikov, M. "Pointcleannet: Learning to denoise and remove outliers from dense point clouds". In: *Computer graphics forum*. Vol. 39. 1. Wiley Online Library. 2020, pp. 185–203.

[Ren+24] Ren, D., Ma, Z., Chen, Y., Peng, W., Liu, X., Zhang, Y., and Guo, Y. "Spiking PointNet: Spiking Neural Networks for Point Clouds". In: *Advances in Neural Information Processing Systems* 36 (2024).

[Sch+17] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Trans. Database Syst.* 42.3 (July 2017). ISSN: 0362-5915. DOI: 10.1145/3068335. URL: https://doi.org/10.1145/3068335.

[Sha+22] Sharma, G., Dash, B., RoyChowdhury, A., Gadelha, M., Loizou, M., Cao, L., Wang, R., Learned-Miller, E. G., Maji, S., and Kalogerakis, E. "Prifit: Learning to fit primitives improves few shot point cloud segmentation". In: *Computer Graphics Forum*. Vol. 41. 5. Wiley Online Library. 2022, pp. 39–50.

[Sun+07] Sun, X., Rosin, P. L., Martin, R., and Langbein, F. "Fast and effective feature-preserving mesh denoising". In: *IEEE transactions on visualization and computer graphics* 13.5 (2007), pp. 925–938.

[Sun+19] Sun, Y., Chen, H., Qin, J., Li, H., Wei, M., and Zong, H. "Reliable Rolling-guided Point Normal Filtering for Surface Texture Removal". In: *Computer graphics forum*. Vol. 38. 7. Wiley Online Library. 2019, pp. 721–732.

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[WSE19] Wang, J., Shan, T., and Englot, B. "Underwater Terrain Reconstruction from Forward-Looking Sonar Imagery". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 3471–3477. DOI: 10.1109/ICRA.2019.8794473.

[Wan+21] Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., and Wang, W. "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction". In: *arXiv preprint arXiv:2106.10689* (2021).

[Wan+22a] Wang, Y., Ji, Y., Liu, D., Tsuchiya, H., Yamashita, A., and Asama, H. "Simulator-aided edge-based acoustic camera pose estimation". In: *OCEANS 2022-Chennai*. IEEE. 2022, pp. 1–4.

[Wan+22b] Wang, Y., Ji, Y., Tsuchiya, H., Asama, H., and Yamashita, A. "Learning Pseudo Front Depth for 2D Forward-Looking Sonar-based Multi-view Stereo". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 8730–8737. DOI: 10.1109/IROS47612.2022.9982049.

[Wan+22c] Wang, Y., Ji, Y., Tsuchiya, H., Asama, H., and Yamashita, A. "Learning pseudo front depth for 2d forward-looking sonar-based multi-view stereo". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 8730–8737.

[Wan+23] Wang, Y., Ji, Y., Wu, C., Tsuchiya, H., Asama, H., and Yamashita, A. "Motion Degeneracy in Self-supervised Learning of Elevation Angle Estimation for 2D Forward-Looking Sonar". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 6133–6140.

[WGK20] Westman, E., Gkioulekas, I., and Kaess, M. "A Theory of Fermat Paths for 3D Imaging Sonar Reconstruction". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 5082–5088. DOI: 10.1109/IROS45743.2020.9341613.

[Wu+16] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[Xu+07] Xu, X., Yuruk, N., Feng, Z., and Schweiger, T. A. "Scan: a structural clustering algorithm for networks". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007, pp. 824–833.

[Yan+22] Yan, X., Lin, L., Mitra, N. J., Lischinski, D., Cohen-Or, D., and Huang, H. "Shapeformer: Transformer-based shape completion via sparse representation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 6239–6249.

[Yar+20] Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., and Lipman, Y. "Multiview neural surface reconstruction by disentangling geometry and appearance". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 2492–2502.

[ZS96] Zerr, B. and Stage, B. "Three-dimensional reconstruction of underwater objects from a sequence of sonar images". In: *Proceedings of 3rd IEEE International Conference on Image Processing*. Vol. 3. 1996, 927–930 vol.3. DOI: 10.1109/ICIP.1996.560946.

[Zha+19] Zhang, J., Zhao, X., Chen, Z., and Lu, Z. "A review of deep learning-based semantic segmentation for point cloud". In: *IEEE access* 7 (2019), pp. 179118–179133.

[ZCD21] Zhang, S., Cui, S., and Ding, Z. "Hypergraph Spectral Analysis and Processing in 3D Point Cloud". In: *IEEE Transactions on Image Processing* 30 (2021), pp. 1193–1206. DOI: 10.1109/TIP.2020.3042088.

[Zhe+17] Zheng, Y., Li, G., Wu, S., Liu, Y., and Gao, Y. "Guided point cloud denoising via sharp feature skeletons". In: *The Visual Computer* 33 (2017), pp. 857–867.

[Zhe+18] Zheng, Y., Li, G., Xu, X., Wu, S., and Nie, Y. "Rolling normal filtering for point clouds". In: *Computer Aided Geometric Design* 62 (2018), pp. 16–28.

[Zho+22] Zhou, L., Sun, G., Li, Y., Li, W., and Su, Z. "Point cloud denoising review: from classical to deep learning-based approaches". In: *Graphical Models* 121 (2022), p. 101140.