Feature-Aware Point Transformer for Point Cloud Alignment Classification

Pose your pose to FACT

Ludvig Dillén
Abstract

As the demand for 3D maps from LI DAR scanners increases, delivering high-quality maps becomes critical. One way to ensure the quality of such maps is through point cloud alignment classification, which aims to classify the alignment error between two registered point clouds. Specifically, we present the classifier FACT (Feature-Aware Classification Transformer), consisting of two main modules: feature extraction and classification. Descriptive features are extracted from the joint point cloud, which are then processed by a point transformer-based neural network to predict the alignment error class. In a ten-class point cloud alignment classification test, FACT achieved 92.4% accuracy, where the alignment error ranged from zero meters and radians to 0.9 meters and 0.09 radians. Remarkably, the classifier only made one misprediction beyond neighboring classes, exhibiting its ability to detect alignment errors as the classes have an inherent order. Furthermore, when benchmarked on two binary classification tasks, FACT showed significantly superior performance over the baseline and even obtained 100.0% accuracy for the easier of the two tasks. FACT not only detects potential errors in 3D maps but also estimates their magnitude, leading to more reliable 3D maps with quality estimations for each transformation.
Acknowledgments

Firstly, I would like to thank my examiner, Per-Erik Forssén, for his continuous support and guidance throughout this thesis and the preceding year. I also want to thank my supervisor, Johan Edstedt, who promptly answered many of my questions and provided insights into transformers, the original problem formulation, code efficiency, and loss function visualizations. Additionally, I thank everyone at CVL for this fun and enriching past year working at the lab. A special mention goes to my soon-to-be former roommate, Erik Larsson, for paying interest in my work and for the valuable discussions about its machine learning details. Furthermore, I acknowledge the assistance of OpenAI’s ChatGPT [57] and Grammarly for proofreading and creating a more readable report. I would also like to thank ChatGPT for its aid in generating Python code for data visualizations. Most importantly, I would like to thank my family and friends who supported me during the past five years and reminded me to sometimes focus on other things besides my studies.

Linköping, September 2023
Ludvig Dillén
# Contents

Notation xi

1 Introduction 1
   1.1 Motivation .......................................................... 2
   1.2 Aim ................................................................. 4
   1.3 Research Questions .................................................. 4
   1.4 Delimitations ...................................................... 4
   1.5 Contributions .................................................... 4

2 Theory 7
   2.1 Geometry and Representation of Point Clouds ................. 7
      2.1.1 Homogeneous Coordinates .................................. 7
      2.1.2 Rotations .................................................. 8
      2.1.3 Rigid Transformations ..................................... 8
      2.1.4 3D Point Clouds ........................................... 9
      2.1.5 Coordinate Systems ......................................... 9
   2.2 The Point Cloud Registration Problem ......................... 10
      2.2.1 Challenges ................................................ 11
      2.2.2 Registration Frameworks .................................. 11
   2.3 Point Cloud Alignment ........................................... 11
      2.3.1 Visibility of Point Clouds ................................ 15
   2.4 Deep Neural Networks for Point Clouds ....................... 19
      2.4.1 Cross-Entropy Loss ....................................... 20
      2.4.2 One-dimensional Wasserstein Loss ....................... 21
      2.4.3 Evaluation Metrics ....................................... 21
      2.4.4 The Attention Mechanism ................................ 22
      2.4.5 The Point Transformer Layer ............................. 23
      2.4.6 Classification with Point Transformer .................. 24
   2.5 Information Theory ............................................. 24
      2.5.1 Sinkhorn Divergence ..................................... 24
      2.5.2 Differential Entropy ..................................... 27

3 Related Work 29
## Contents

3.1 Overview of Related Work ........................................ 29  
3.2 CorAl ................................................................. 30  

### 4 Method

4.1 Data ................................................................. 33  
4.1.1 Data Generation ................................................. 34  
4.2 Implementing CorAl ................................................ 35  
4.3 Our Point Cloud Alignment Classifier - FACT ..................... 36  
4.3.1 Module 1: Feature Extraction ................................ 36  
4.3.2 Module 2: Point Transformer for Alignment Classification ... 45  
4.4 Tuning Methodology .................................................. 48  
4.4.1 Data Usage and Hardware ...................................... 48  
4.4.2 Parameters To Tune Thoroughly ................................ 48  
4.4.3 Default Settings ................................................... 54

### 5 Results and Analysis

5.1 Ground Truth Leakage .............................................. 55  
5.2 Parameter Tuning and Component Choosing ......................... 56  
5.2.1 Radii Variations .................................................. 56  
5.2.2 Augmentation and Normalization ............................... 58  
5.2.3 Visibility Kernels ................................................. 59  
5.2.4 Loss Functions ..................................................... 60  
5.2.5 Model Scaling ....................................................... 60  
5.2.6 Best Settings ....................................................... 62  
5.3 Additional Improvements ............................................. 62  
5.3.1 Spatial Information ............................................... 62  
5.3.2 Normalization for Features and Positional Embedding ......... 63  
5.3.3 New Augmentation Techniques .................................. 64  
5.4 Feature Channel Ablation ............................................ 66  
5.5 Multinomial Classification .......................................... 67  
5.5.1 Qualitative Analysis .............................................. 68  
5.6 Binary Classification ................................................ 70

### 6 Conclusion

6.1 Future Work .......................................................... 76

### A Proofs and Derivations

A.1 Transformation Functions in Point Cloud Visibility ............... 79  
A.1.1 Spherical Flipping ............................................... 79  
A.1.2 Exponential Inversion ............................................ 80  
A.2 Information Theory .................................................. 82  
A.2.1 Simplified Expression for the Differential Entropy of the Multivariate Normal Distribution .......... 82  
A.3 Point Cloud Densities ............................................... 83  
A.3.1 Deriving the Adaptive Radii .................................... 83  
A.3.2 Surface Densities of Varying Data Positions ................... 85  
A.4 Scene-Dependent Classification Accuracy .......................... 86
B Images 87
  B.1 Point Cloud Pairs ........................................ 87

Bibliography 91
# Notation

## Standard Terminology

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}$</td>
<td>The set of real numbers.</td>
</tr>
<tr>
<td>$\mathbb{R}^+$</td>
<td>The set of real numbers larger than zero.</td>
</tr>
<tr>
<td>$\mathbb{R}^3$</td>
<td>The set of real 3D vectors.</td>
</tr>
<tr>
<td>$D$</td>
<td>The dimension of some space.</td>
</tr>
<tr>
<td>${ \cdot }$</td>
<td>A set of values.</td>
</tr>
<tr>
<td>$[l, u]$</td>
<td>A range of values between the lower limit $l$ and the upper limit $u$ including both boundaries.</td>
</tr>
<tr>
<td>$]l, u[$</td>
<td>A range of values between the lower limit $l$ and the upper limit $u$ excluding both boundaries.</td>
</tr>
<tr>
<td>$</td>
<td>\cdot</td>
</tr>
<tr>
<td>$\odot$</td>
<td>The Hadamard product.</td>
</tr>
</tbody>
</table>

## Terminology for Geometry

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>A point.</td>
</tr>
<tr>
<td>$x_h$</td>
<td>A point in homogeneous coordinates.</td>
</tr>
<tr>
<td>$SO(3)$</td>
<td>The special orthogonal group of 3D rotations.</td>
</tr>
<tr>
<td>$SE(3)$</td>
<td>The special Euclidean transformation group.</td>
</tr>
<tr>
<td>$R$</td>
<td>A rotation matrix in the WCS.</td>
</tr>
<tr>
<td>$R^i$</td>
<td>A rotation matrix in the LCS $i$.</td>
</tr>
<tr>
<td>$t$</td>
<td>A translation vector in the WCS.</td>
</tr>
<tr>
<td>$t^i$</td>
<td>A translation vector in the LCS $i$.</td>
</tr>
<tr>
<td>$T$</td>
<td>A rigid transformation matrix in the WCS.</td>
</tr>
<tr>
<td>$T^i$</td>
<td>A rigid transformation matrix in the LCS $i$.</td>
</tr>
</tbody>
</table>
## Terminology for Point Clouds

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>A 3D point in the WCS.</td>
</tr>
<tr>
<td>$p^i$</td>
<td>A 3D point in the LCS $i$.</td>
</tr>
<tr>
<td>$\mathcal{S}$</td>
<td>A surface or a set of surfaces. Often, the set of surfaces from where point clouds are sampled.</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>A point cloud in the WCS.</td>
</tr>
<tr>
<td>$\mathcal{P}^i$</td>
<td>A point cloud in the LCS $i$.</td>
</tr>
<tr>
<td>$\tilde{i}$</td>
<td>Consider a point cloud pair. If $\mathcal{P}<em>i$ is one of the point clouds, $\mathcal{P}</em>{\tilde{i}}$ is the other one.</td>
</tr>
<tr>
<td>$\mathcal{P}^i_{i,\tilde{i}}$</td>
<td>The concatenation of point cloud $\mathcal{P}^i_i$ and $\mathcal{P}^i_{\tilde{i}}$ in LCS $i$.</td>
</tr>
<tr>
<td>$b_{ij}$</td>
<td>A binary prediction of co-visibility for $p^i_{ij} \in \mathcal{P}^i_i$, $i = 0, 1, j = 0, \ldots, N_i - 1$, where $b_{ij} \in {0, 1}$.</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>A co-visibility confidence of $p^i_{ij} \in \mathcal{P}^i_i$, $i = 0, 1, j = 0, \ldots, N_i - 1$, where $c_{ij} \in [0, 1]$.</td>
</tr>
<tr>
<td>$\mathcal{J}$</td>
<td>The indices to the farthest point sampled point cloud.</td>
</tr>
</tbody>
</table>
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAM</td>
<td>Adaptive Moment Estimation</td>
</tr>
<tr>
<td>AN</td>
<td>Augmentation and Normalization</td>
</tr>
<tr>
<td>BS</td>
<td>Batch Size</td>
</tr>
<tr>
<td>CS</td>
<td>Coordinate System</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>EP</td>
<td>Epochs</td>
</tr>
<tr>
<td>FACT</td>
<td>Feature-Aware Classification Transformer</td>
</tr>
<tr>
<td>FPS</td>
<td>Farthest Point Sampling</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>HPR</td>
<td>Hidden Point Removal</td>
</tr>
<tr>
<td>kNN</td>
<td>k-Nearest Neighbors</td>
</tr>
<tr>
<td>LCS</td>
<td>Local Coordinate System</td>
</tr>
<tr>
<td>LF</td>
<td>Loss Function</td>
</tr>
<tr>
<td>LIDAR</td>
<td>Light Detection and Ranging</td>
</tr>
<tr>
<td>LR</td>
<td>Learning Rate</td>
</tr>
<tr>
<td>LRS</td>
<td>Learning Rate Scheduler</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>MMD</td>
<td>Maximum Mean Discrepancy</td>
</tr>
<tr>
<td>MMN</td>
<td>Min-Max Normalization</td>
</tr>
<tr>
<td>NB</td>
<td>Number of Blocks</td>
</tr>
<tr>
<td>OP</td>
<td>Optimizer</td>
</tr>
<tr>
<td>OT</td>
<td>Optimal Transport</td>
</tr>
<tr>
<td>PCAC</td>
<td>Point Cloud Alignment Classification</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>TD</td>
<td>Transformer Dimensionality</td>
</tr>
<tr>
<td>VRAM</td>
<td>Video Random Access Memory</td>
</tr>
<tr>
<td>VK</td>
<td>Visibility Kernel</td>
</tr>
<tr>
<td>WCS</td>
<td>World Coordinate System</td>
</tr>
<tr>
<td>WD</td>
<td>Weight Decay</td>
</tr>
</tbody>
</table>
Introduction

Today, numerous applications employ LIDAR (light detection and ranging) sensors for various tasks within robotic perception [6]. A LIDAR consists of a transmitter and a receiver [78]. It measures distances to surrounding objects by transmitting light pulses, which, after reflection, are measured by the receiver [78]. In the receiver, a computing device estimates the distance to the hit object by using the known speed of light, a timer, and signal processing [78]. The primary data from a LIDAR is thus distance measurements. With these measurements, partitioned into scans\(^1\), it becomes possible to reconstruct 3D scenes. To do this, rigid transformations are used to merge multiple scans into the same coordinate system, creating a map. The problem of finding the rigid transformation between two point clouds has been extensively studied and can be solved through point cloud registration [11, 37, 63, 73]. As more LIDAR scans are registered (added), the map becomes increasingly detailed. In the multi-view scenario, this problem can be formulated as finding the \(n - 1\) best rigid transformations between an ordered set of \(n\) point clouds. Point cloud data can be useful for environment perception and localization for robotic agents, facilitating advancements in areas including autonomous driving, underground mining, building information modeling, and terrain mapping.

We want to estimate if the rigid estimate between a pair of point clouds is accurate. To solve this, we will study point cloud alignment classification (PCAC). We define point cloud alignment classification as the task of categorizing the results of point cloud registration into \(m\) bins. Each bin represents different rigid transformation errors, with one bin corresponding to an error-free outcome.

There are other methods that study the registration result, but there is no

---

\(^1\) A LIDAR scan, which is also called a point cloud, consists of measurements from one LIDAR revolution. However, a point cloud is defined as a set of points, so it can consist of an arbitrary number of scans.
common problem formulation and known state-of-the-art [6, 11, 16, 19]. Some methods [6–11, 16, 19, 25, 26, 29, 54, 85] try to solve this problem, where the approaches mainly stem from robotic and statistical methods, with fewer elements of the recent promising advancements within deep neural networks (DNNs) and computer vision. The methods studied are often simple, fast, and developed to work integrated in a SLAM (simultaneous localization and mapping) pipeline. Frequently, they are tested on a self-collected private dataset, creating difficulties in comparing different approaches.

Partly because the methods that try to solve PCAC use different datasets and therefore try to solve slightly different tasks, the research on PCAC is fragmented. Another cause for the fragmentation is the lack of naming conventions and code availability. As an example of the lack of naming conventions, we have seen the following names for this problem in literature: "Map Validation", "Map Quality", "Localization Failure", "Failure Detection", "Misalignment Recognition", "Alignment Risk", "Alignment Classification", and more. In addition, for all the methods we have studied, we have not found any publicly available code. This makes it more difficult to compare methods. All of this has also led to that there does not exist any extensive comparison between the most promising solutions to this problem. Among the methods we have studied, we have selected CorAl (Correctly Aligned) by Adolfsson et al. [6, 7] to serve as our baseline throughout the report. An overview of the related work is presented Section 3.

1.1 Motivation

Manual visual inspection is still a common way to analyze map quality today. This method is tedious for large maps and might not detect minor errors, which can propagate to more significant errors later in a sequence. As the advancement of 3D sensors continues and the use of 3D point cloud registration in various applications intensifies [37], there is an increasing need for a reliable quality metric without the reliance on manual visual inspection. If we can assess the quality of a 3D point cloud model, we can understand where registration errors most likely appear in the model. With this quality information, special measures can be taken to repair the point clouds in regions of low quality. This provides a representation of the model’s accuracy, which is beneficial when comparing different 3D models.

Given the limited application of DNNs in PCAC, we aim to investigate if they can be useful. It is also of interest to incorporate hand-crafted features that have been more extensively studied in the existing literature. With these tools, it is intriguing to go beyond previous methods, which mainly have looked at binary PCAC, to look at whether a classifier can discern between different alignment error magnitudes. With several levels of alignment errors, the progress of iterative transformation updates can better be monitored. It also gives further detail into representing a 3D model’s accuracy and can be used when prioritizing transformation corrections.

In the registration problem, a loss function is minimized. However, as loss
1.1 Motivation

Functions typically are non-convex [6, 46] and may not even describe the underlying registration error well [73], the output is simply an estimate of the best rigid transformation [37]. When measuring the quality after alignment, the following ideas should be taken into account, which further motivates solving the PCAC problem:

1. Highly non-linear and non-convex aspects (e.g., visibility [73]) can be considered because the discrimination function does not need to be convex with respect to the pose\(^2\), as it will not be updated. For example, even if the quality metric has many stationary points, the PCAC problem could still be solved as long as the optimal rigid transformation gives a global optimum that is distinguishable from the other local optima. This could possibly be detected using a simple threshold as visualized in Figure 1.1.

2. Since the alignment quality will only be calculated once for each pair of point clouds, the computational cost is allowed to be higher compared to each iteration of a registration algorithm. Thus, more complex methods can be considered.

As many different modules of a solution to the PCAC problem will be developed, it becomes crucial to determine which modules are important. Highlighting this also emphasizes what type of research should be prioritized in future work.

\[\text{Figure 1.1: The graphs illustrate the varied aspects inherent to point cloud registration and alignment classification. The difference in the graphs arises because the classification is performed only once for each estimated transformation, whereas the registration iteratively updates the pose. Our primary intent in presenting this figure is to give an idea of the problem at hand, facilitating the discovery of a plausible solution.}\]

\(^2\)A pose consists of a translation part (position) and a rotation part (orientation).
1.2 Aim

This thesis aims to determine the accuracy of rigid transformation estimates between point cloud pairs. We will investigate hand-crafted features and deep learning techniques to address this issue.

1.3 Research Questions

1. How well can hand-crafted features together with DNNs distinguish between aligned and misaligned point cloud pairs?

2. Is it possible to accurately measure the absolute alignment error between two point clouds?

3. Which modules are important in a point cloud alignment classification pipeline?

1.4 Delimitations

In this thesis, the following delimitations will be made:

i We will only consider classifying pairwise point cloud alignments. Even though we do not perform any explicit limitation on the number of points in the point clouds, concatenations of several point clouds will not be investigated.

ii We will focus on mobile LIDAR sensors in this study; however, the results of the study may be beneficial to other sensors such as Kinect sensors, terrestrial laser scanners, and stereo cameras, even if the point cloud sparsity from the sensors varies.

iii Only same-source registration will be studied. That is, registration using sensor fusion will not be considered.

iv We will introduce controlled perturbations to the data instead of utilizing outputs from a registration pipeline, aiming to simplify the learning problem.

v All experiments will be run on an NVIDIA GeForce GTX 1080 Ti GPU with 11GB RAM [5].

1.5 Contributions

The major contributions of this thesis are:

- We formulate what point cloud alignment is and identify the subtasks necessary to accurately estimate alignment quality.
• We extend the point cloud alignment classification (PCAC) task from a binary yes/no classification to a multinomial classification of the registration error.

• We propose FACT (Feature-Aware Classification Transformer), a method for point cloud alignment classification using feature extraction and deep neural networks, leveraging the point transformer architecture by [88].

• We identify the key features essential for describing alignment in point clouds.

• We compare our approach with the baseline, CorAl [6], for the binary classification task, demonstrating that our method has superior performance.

• We detail the benefits of combining the one-dimensional Wasserstein loss with the cross-entropy loss for multinomial classification tasks with inherent order.

• We present a method to choose neighborhood sizes as a function of two sensor positions, thus accounting for the expected joint point cloud density.

• We show that our multinomial classifiers have excellent performance on the binary classification tasks. These multinomial classifiers are trained on more than the two classes it is tested on.
This chapter outlines the theoretical foundation pivotal for understanding this thesis. Fundamental topics, like rigid transformations, are elaborated in depth due to their importance and recurring application in this work. In contrast, concepts like registration algorithms are only touched upon briefly. Similarly, we assume that the reader has prior rudimentary knowledge of neural networks; thus, we do not spend much time on them. Notations and abbreviations are cataloged in the Notation Index at the thesis’s outset for ease of reference. While complex ideas like the underpinnings of Sinkhorn divergence are introduced to give an enriched background to the interested reader, fully comprehending such topics is not crucial to appreciate the concluding findings.

2.1 Geometry and Representation of Point Clouds

An appropriate representation of geometrical objects becomes important when describing, discussing, explaining, and deriving results. This thesis will primarily study point clouds. To facilitate the argumentation, proper representations of these objects will be introduced.

2.1.1 Homogeneous Coordinates

We start by introducing homogeneous coordinates, a useful concept when applying transformations and constraints on geometric objects [55]. As the point clouds we are studying are in 3D, we will only consider the $\mathbb{R}^3$ space. The canonical form of the homogeneous coordinates of a 3D point, $\mathbf{x}$, in Cartesian coordi-
nates, is defined as
\[ x_h \sim \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}. \]  
(2.1)

### 2.1.2 Rotations
When transforming geometric objects, rotations and a relevant representation are necessary. To start with, we define a rotation matrix acting on elements in \( \mathbb{R}^3 \).

**Definition 2.1 (SO(3)).** The \( \text{SO}(3) \) group consists of every real \( 3 \times 3 \) matrix, \( R \), that satisfies
\[
R^T R = I, \quad \text{and} \quad \det(R) = 1,
\]
under the operation of composition\(^1\) [43, 49, 56].

**Rodrigues’ Rotation Formula**
Rodrigues’ rotation formula can be useful when we want to apply a rotation around a specified axis and map the corresponding rotation to an \( \text{SO}(3) \) matrix.

**Definition 2.2 (Rodrigues’ Formula).** Let the anti-clockwise rotation around the rotation axis \( \hat{n} \) with angle \( \gamma \) be defined according to Rodrigues’ formula as
\[
R(\hat{n}, \gamma) = I_3 + \sin(\gamma)[\hat{n}]_\times + (1 - \cos(\gamma))[\hat{n}]_\times^2,
\]
where \([ \cdot ]_\times\) denotes the cross product operator [55].

### 2.1.3 Rigid Transformations
As this thesis frequently uses rigid transformations to transform point clouds, handle coordinate systems, and pose sequences, we define a rigid transformation in \( \mathbb{R}^3 \).

**Definition 2.3 (Rigid Transformation).** A rigid transformation in \( \mathbb{R}^3 \) given in homogeneous coordinates is represented by a \( 4 \times 4 \) matrix of the form
\[
T \sim \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix},
\]
where \( R \in \text{SO}(3) \) is a rotation matrix and \( t \in \mathbb{R}^3 \) is a translation vector.\(^1\)

A 3D rigid transformation has six degrees of freedom and preserves angles, distances, areas, volumes, and handedness in \( \mathbb{R}^3 \), and needs at least three non-colinear point correspondences prior and posterior to the transformation to be uniquely determined [27, 55]. The group of 3D rigid transformations is also referred to as \( \text{SE}(3) \), the special Euclidean transformation group in \( \mathbb{R}^3 \) [55].

\(^1\)This means that if \( R_0, R_1 \in \text{SO}(3) \), then \( R_0 R_1 \in \text{SO}(3) \).
2.1 Geometry and Representation of Point Clouds

2.1.4 3D Point Clouds

A 3D point cloud, \( P \), is an unordered set of \( n \) 3D points, where each point consists of the \((x, y, z)\) coordinate vector. Even though we will regularly refer to specific points with indices, such as \( p_j \), it does not mean we impose a specific order on the point set. Additional features like intensity can be associated with each point to describe the data further. It is also possible to add features describing the point’s neighborhood, e.g., a surface normal. The neighborhood can be parameterized in various ways, including a radius defining a sphere, a grid size for voxels, or by \( k \)-nearest neighbors (kNN). Furthermore, since the point cloud is an unordered set, it is invariant to permutations \[61\]. The spatial points are also assumed not to be isolated\(^2\), and hence, the interaction between neighboring points is easily handled (e.g., distances are easily comparable). Moreover, point clouds can have varying density or constant density, whereas point clouds acquired from a LIDAR sensor naturally exhibit varying density.

When studying pairs of point clouds sampled at different points in time, we can estimate corresponding regions in the point clouds. Some corresponding regions might have changed position while others have not. Let us introduce terminology to distinguish between the two cases clearly.

**Definition 2.4 (Dynamic Point).** If a point \( p \) changes position between two time points \( t_0 \) and \( t_1 \), it is *dynamic*; otherwise, it is *static*.

2.1.5 Coordinate Systems

Upon working with sensor measurements of moving agents or employing multimodal approaches, handling different coordinate systems (CS) becomes inevitable. For instance, we establish one local coordinate system (LCS) after every LIDAR revolution. We start by introducing relevant notation for this. Let \( T \) denote a pose given in the world coordinate system (WCS) and let \( T^i \) denote a pose in the LCS of sensing device \( i \) or at time point \( i \). We furthermore extend this notation to rotations, \( R^i \), and translations, \( t^i \). Note that the WCS is chosen by the engineer and then fixed for the remainder of the use case. Frequently, the WCS’s origin is placed in the origin of the first measurement of the sensing device (in the same-source case). Additionally, we extend this notation to point clouds, \( P \), and their points, \( p \).

Often, one is given a vehicle pose \( (T_v) \) expressed in the WCS and a sensing device (e.g., LIDAR) pose \( (T_s^i) \) in the LCS of the vehicle. The chain of transformations required to obtain the sensing device’s pose in the WCS is given by:

\[
T_s = T_v T_s^i = \begin{bmatrix} R_v^i & t_v^i \\ 0^i & 1 \end{bmatrix} \begin{bmatrix} R_s^i & t_s^i \\ 0^i & 1 \end{bmatrix} = \begin{bmatrix} R_v R_s^i & R_v t_s^i + t_v^i \\ 0^i & 1 \end{bmatrix}.
\]

(2.5)

If we want to align two point clouds \( (P_0^0 \) and \( P_1^1 \) ) with their respective poses \( (T_0 \) and \( T_1 \) ), these steps can be carried out:

\(^2\)This means that the distance between all points in the set is finite.
1. First, we map $P_1$ to the WCS, thus obtaining $P_1$.

2. Secondly, we map $P_1$ to the LCS of $P_0$, thus obtaining $P_0^1$.

If $P_0^0 = P_0$ (i.e., we let the first point cloud’s pose denote the origin of the WCS), then it is sufficient to perform the first step. Mathematically, we write

$$
\begin{bmatrix}
    p_{1j}^0 \\
    1
\end{bmatrix}
= T_{0}^{-1} T_1
\begin{bmatrix}
    p_{1j}^1 \\
    1
\end{bmatrix}
= \begin{bmatrix}
    R_0^T & -R_0^T t_0 \\
    0^T & 1
\end{bmatrix}
\begin{bmatrix}
    R_1 \\
    t_1
\end{bmatrix}
\begin{bmatrix}
    p_{1j}^1 \\
    1
\end{bmatrix}
= \begin{bmatrix}
    R_0^T R_1 & R_0^T (t_1 - t_0) \\
    0^T & 1
\end{bmatrix}
\begin{bmatrix}
    p_{1j}^1 \\
    1
\end{bmatrix}
$$

(2.6)

for each $p_{1j}^1 \in P_1$. Now, we have aligned the point clouds as we have $P_0^0, P_1^0$ in the same coordinate system. It is important to note that the pose of the second point cloud in the LCS of the first point cloud is, as previously mentioned, $T_{1}^0 = T_{0}^{-1} T_1$.

Furthermore, let us now denote the joint point cloud and its points as $P_{0,1}^0, p_{j}^0$ while still remembering each point’s originating point cloud using $\pi_j$, where:

$$
\pi_j = \begin{cases}
    0 & \text{if } p_j \text{ originates from } P_0, \\
    1 & \text{if } p_j \text{ originates from } P_1.
\end{cases}
$$

(2.7)

This makes it possible to keep track of which point cloud each point in the joint point cloud originates from.

### 2.2 The Point Cloud Registration Problem

The objective in point cloud registration is to find the rigid transformation that best aligns a pair of point clouds [37]. It can be addressed through same-source and cross-source approaches. Cross-source (also referred to as multimodal) systems are more complex but often advantageously fuse information from different data sources [37]. This thesis will not delve further into the multimodal methods for point cloud registration. Instead, we direct the interested reader to papers like [7, 37, 70].

Moreover, the registration problem consists of two main components: the registration loss and a method that minimizes the registration loss [73]. Huang et al. [37] explain the point cloud registration problem as follows: given $K$ correspondence pairs between point cloud $P_0^0$ and $P_1^1$, the objective is to find the rigid transformation that best aligns the point clouds. They formulate it as

$$
\arg \min_{R_0^0 \in SO(3), t_0^1 \in \mathbb{R}^3} \left\| d(P_0^0, P_1^1) \right\|^2_2, \text{ where } d(P_0^0, P_1^1) = \sum_{j=0}^{K-1} \left\| p_{ij}^0 - (R_1^0 p_{ij}^1 + t_1^0) \right\|^2_2,
$$

(2.8)

where $p_{ij}$ is point $j$ in point cloud $i \in \{0, 1\}$. However, a limitation here is that point correspondences have to be estimated. Perhaps, this is impossible or makes
aligning the two point clouds more difficult than necessary. This problem formulation implies that both correspondences and the transformation must be estimated when in practice, only the transformation is desired to acquire. Theoretically, when one of the two quantities is correctly found, the other can be derived optimally [37]. Nonetheless, the joint problem requires more effort to be solved.

2.2.1 Challenges

Within the context of same-source point cloud registration, challenges include but are not limited to:

1. Sensor noise: The sensor noise increases with the distance to the sensor [79].

2. Sampling density: As the distance between an agent and an object varies over two scans, the scans’ sample densities of the object will vary, causing difficulties. For example, one-to-one correspondences between the point clouds can no longer be expected.

3. Partial overlap: The scene’s visibility may change as agents or objects move during the data acquisition process. Consequently, identifying the areas where point clouds overlap frequently becomes a necessary subtask to address. See Proposition 2.6 for why partial overlap occurs.

4. Dynamic scenes: If the scene is dynamic, the best rigid transformation cannot correctly align both the static and the dynamic regions (see Proof 2).

2.2.2 Registration Frameworks

Numerous point cloud registration frameworks have been proposed over the years. Huang et al. [37] divide the various methods into three categories as presented in Table 2.1. Here, we limit our presentation of registration frameworks; however, this can be a starting point for possible approaches when solving the point cloud alignment classification task.

2.3 Point Cloud Alignment

Throughout this thesis, the concept of point cloud alignment is extensively discussed. Hence, it is appropriate to explain the idea adequately. First, we study Figure 2.1, demonstrating the distinction between aligned and misaligned point cloud pairs in practice. It is essential to acknowledge that achieving perfect alignment between two point clouds is difficult due to sensor noise, different point cloud densities, occlusions, and a limited field of view. Instead, we direct the discussion to the rigid transformation yielding the best possible alignment. Figure 2.1a illustrates alignment, as the blue and the green points occupy similar

3 The correspondences must either be static points, or dynamic points where their translations are known.
Table 2.1: This table summarizes the three point cloud registration categories explained by Huang et al. [37]. The table is partially adapted from the mentioned paper.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization-based</td>
<td>Often based on solving the correspondence problem and the transformation estimation jointly.</td>
<td>1. Requires no training data. 2. Have theoretical guarantees of convergence.</td>
<td>1. Handling special cases requires complex optimization formulations. 2. Does not leverage DNNs.</td>
</tr>
<tr>
<td>Feature learning</td>
<td>Find descriptive features using DNNs, followed by correspondence search and transformation estimation.</td>
<td>1. Can generate robust and accurate correspondences. 2. Once correspondences are established, it is fast.</td>
<td>1. Need much training data. 2. Does not handle out-of-distribution well. 3. Not end-to-end.</td>
</tr>
<tr>
<td>End-to-end learning</td>
<td>Two point clouds are fed into a DNN and the output is the transformation estimate.</td>
<td>1. End-to-end. 2. The network optimizes parameters to solve the registration task directly.</td>
<td>1. Black box. 2. Sensitive to noise and density differences.</td>
</tr>
</tbody>
</table>

positions. In practice, point clouds are considered aligned if they align sufficiently well. This introduces the necessity to find a carefully calibrated decision boundary to separate aligned and misaligned point cloud pairs, which should be determined based on the application specifics.

![Figure 2.1: An example of how aligned and misaligned point clouds might look in practice with an Ouster OS-1 LiDAR sensor [58] acquired in the project Smart Twins for Forest Environments [72]. In Figure 2.1b, the blue and red point clouds have similar structures but appear shifted, suggesting a misalignment.](image)

While we have demonstrated alignment through a practical example, a rigorous mathematical representation is essential for measuring it without resorting to laborious manual visual inspection. We start by considering an arbitrary point \( p_{ij} \in P_i \). We now define the point’s neighborhood, \( \Omega_{ij} \subseteq P_i \), as

\[
\Omega_{ij} = \{ p_{ik} : \| p_{ik} - p_{ij} \|_2 < r, p_{ik} \in P_i \}, i \in \{0, 1, (0, 1)\},
\]

where \( r \) is some radius suitable for defining the neighborhood. For simplicity, we here on denote \( \Omega_{(0,1)j} \) as \( \Omega_j \). One could argue that if a point cloud is aligned, all \( \Omega_j \) should approximately have equal numbers of \( \pi \) values being 0 and 1. At
least this reasoning makes sense when studying Figure 2.1. Imagine randomly choosing a point in either figure and considering the resulting distribution in the point’s local neighborhood. This will greatly vary between the left and right figures. To ascertain whether the number of neighbors is equal from both point clouds, we define the variable

\[ \Phi_{ij} = \frac{|\Omega_{ij}|}{|\Omega_j|}, \quad (2.10) \]

where \(|\cdot|\) denotes the cardinality of the set. According to the argumentation, a point cloud pair is aligned if \(\Phi_{ij} = 0.5\). Moreover, this condition is not sufficient to define alignment, even if it can indicate alignment. Let us prove why (see below). This condition is not even mathematically necessary due to different point cloud densities, for example.

**Proof:** In Figure 2.2, we see that \(\Phi_{ij} = 0.5\) holds for all points, but the point sets are not aligned. Even though this plot is in \(\mathbb{R}^2\), we can still have points spread on a 2D surface in \(\mathbb{R}^3\), generalizing the proof to \(\mathbb{R}^3\).

Although we disproved the previous suggestion of alignment, the idea remains useful; nonetheless, it should be expanded upon. The value of the radius, \(r\), is imperative for the usage of the variable \(\Phi_{ij}\). A too small value will not adequately capture the neighborhood. For instance, \(\Phi_{ij} \to 1\) if \(r \to 0\), consistently indicating misalignment. Conversely, an excessively large value will increase the computational complexity and capture more than the immediate neighborhood.
For instance, \( \Phi_{ij} \to 0.5 \) if \( r \to \infty \), given that \(|P_0| = |P_1|\), thereby always indicating alignment. Perfect alignment occurs when the points from both point clouds occupy the exact same space in the WCS. As previously mentioned, achieving this is practically infeasible. This is because of the following:

1. Due to the movements of the agent and the presence of dynamic object regions, all regions of the joint point cloud \( P_{0,1} \) will not be co-visible (visible from both sensor poses). Hence, these regions cannot obtain \( \Phi_{ij} = 0.5 \). This topic is further investigated in Section 2.3.1 and Section 4.3.1.

2. The best rigid transformation will not perfectly align both static and dynamic points.

**Proof:** According to Definition 2.4, a point that is dynamic over \( P_0 \) and \( P_1 \) (the point clouds are assumed to be sampled at different points in time) changes position in the WCS whereas a static point does not. As rigid transformations preserve angles and distances, the angles and distances must be the same between all points \( P_1 \) as in \( P_0 \). However, since the relative position between a dynamic and static point cannot be the same over \( P_0 \) and \( P_1 \), the rigid transformation cannot perfectly align both points in \( P_0 \) to their corresponding points in \( P_1 \).

Hence, precisely estimating the quality of a transformation estimate necessitates the handling of dynamic objects. While this topic is beyond the scope of this thesis, we strongly advocate for future research in this area. One potential approach could be scene flow estimation using the method by Zhang et al., where the translation of corresponding points is estimated [87].

3. The sample density of surfaces captured by a mobile LIDAR can vary considerably. Factors such as the agent’s movement, the presence of dynamic object regions, and occlusion all influence sampling differences. Given these dynamics, we cannot expect consistent densities in regions affected by these factors, making the value \( \Phi_{ij} = 0.5 \) typically unrealistic in practice. Studying sampling differences in more detail would likely increase the chances of correctly assessing alignment quality. Nevertheless, it is left for future research. However, we do, in the subsection The Neighborhood Size under Section 4.4.2, provide some details on how surface densities vary with the distance between two sensors and an object which can facilitate the research on handling point cloud density differences.

Finally, we propose what practical point cloud alignment is in Proposition 2.5 as an expansion of the variable \( \Phi_{ij} \).

**Proposition 2.5 (Practical Point Cloud Alignment).** Given two point clouds \( P_0, P_1 \) and an estimated transformation \( T_{0,1} \), the joint point cloud \( P_{0,1} \) is practically aligned if

1. when we only consider co-visible regions,
2.3 Point Cloud Alignment

**Figure 2.3:** An image of a point cloud synthetically generated with Point-E [52] visualized from two viewpoints. It is apparent that some regions are co-visible while others are not.

2. remove the dynamic objects,

3. adjust for the difference in sampling density, and

4. compensate for sensor noise,

in local neighborhoods, $\Omega_j$, obtain Wasserstein distances between the point clouds close to 0.

### 2.3.1 Visibility of Point Clouds

In Figure 2.3, we see a complete 3D point cloud of a yellow car from two viewpoints generated synthetically with Point-E [52]. It is clear that the visible points from the poses are not the same. When studying the alignment of point clouds registered from different poses and/or moments in time, the notion of local alignment only makes sense in co-visible regions. In other regions, there will not be samples from both point clouds, and then it is impossible to have local alignment, even if the estimated transformation is accurate. In the co-visible regions, the relative surfaces as functions of the transformation estimate can be compared between the poses. If the estimated transformation is correct, the co-visible areas will, for the set of static points, align nearly perfectly. In other words, the two estimations of the position of the surface are the same. However, if the estimated transformation is incorrect, the co-visible regions will not align, as seen in Figure 2.1b.

The notion of co-visibility of point cloud pairs is to estimate if a point $p_{ij} \in \mathcal{P}_i^j$, $i = 0, 1$, $j = 0, \ldots, N_i - 1$ is visible from both poses $T_j$, $i = 0, 1$. The prediction can either be in the form of a binary prediction $b_{ij} \in \{0, 1\}$ or a co-visibility confidence $c_{ij} \in [0, 1]$. We express the co-visibility task as:
Co-visibility task: For a given point $p_{ij}$, determine its degree of co-visibility from two distinct viewpoints, represented by $b_{ij}$ or $c_{ij}$.

When noise is disregarded, all points captured by a LIDAR will necessarily be visible from the viewpoint of the sensor. However, when studying the visibility of points from another viewpoint, there may be points that are hidden due to several reasons. We start by explaining in which cases such points may appear in Proposition 2.6.

**Proposition 2.6 (Causes of Hidden Points).** A point $p_{ij}$ is hidden from a sensor due to one or more of the following reasons:

- it is occluded
- it is not in the field of view of the sensor
- it is not within the range of the sensor

We divide up the co-visibility problem into two parts:

I First, studying the point clouds separately: Given the two viewpoints, which points are co-visible?

II Second, we investigate the joint point cloud $P_{0,1}$: Given the two viewpoints, which points are co-visible?

For a point $p_{ij}$ to be classified as co-visible, it must be considered co-visible in both case I and II. As we go into details, we will see why case I is weaker given this condition and implicitly covered by case II. To motivate this claim, we present two useful statements:

1. If $p_{ij}$ is visible from $T_{\tilde{i}}$, it will remain visible when removing (other) points from $P_{0,1}$, where $\tilde{i}$ denotes the other point cloud than $i$.

2. All $p_{ij}$ are visible from $T_i$.

The first point holds as removing other points from a point cloud will not by Proposition 2.6 affect any of the causes of why a point, $p_{ij}$, is hidden. Hence, $p_{ij}$ will remain visible.

Now, we can realize that problem I is implicitly solved by solving problem II. Note that it is, by the second point, not necessary to consider if $p_{ij}$ is co-visible from $T_i$. Assume $p_{ij}$ is considered co-visible according to case I. Then, it might be co-visible or not in the joint case, as adding points to the point cloud can create occlusions. However, if $p_{ij}$ is considered co-visible in case II, it must be co-visible in case I by the first point. This is because a point that is considered visible in the joint case, must remain visible in the separate case as removing points from the joint point cloud will not affect its visibility. Thus, co-visibility in II implies

\[\text{It is important to note that this concerns point clouds sampled from a viewpoint. This does not hold for complete point clouds such as the one in Figure 2.3.}\]
co-visibility in I. If we require a point to co-visible in both case I and II to be considered visible, it suffices checking case II.

An interesting remark is that if a point is hidden, according to some operator, from its own viewpoint, there must be something wrong. This can either be that the transformation estimate is wrong, there exist dynamic points, or that there is sensor noise. Consequently, whether a point is visible from its own viewpoint could serve as a feature to discriminate between correct and incorrect transformation estimates.

### The "Hidden" Point Removal Operator

Katz et al. [41] introduces the "Hidden" Point Removal (HPR) operator to determine which points of a point cloud are visible from a given viewpoint. Theoretically, the operator they propose has various use cases and is not limited to utilization in any point cloud dimension, density, or viewpoint. Their operator does not try to reconstruct the underlying surface but instead finds the hidden points directly from the raw point cloud.

To obtain useful terminology, we present a few relevant mathematical statements from Katz et al. [41]. As a point cloud constitutes a set of sampled surfaces, the notion of sample density is fundamental.

**Definition 2.7. (Sample Density)**

A sample \( P \subseteq S \) is a \( \rho \)-sample from surface \( S \) if \( \forall q \in S, \exists p \in P \) s.t. \( \|q - p\| < \rho \).

Almost identically as Katz et al., we define what an \( \varepsilon \)-visible point is in \( \mathbb{R}^D \) (see Definition 2.8). Intuitively, if \( P \) is 0-visible, then all points in \( P \) are indeed visible.

**Definition 2.8. \( \varepsilon \)-visible**

A point \( p \in P \) is \( \varepsilon \)-visible from \( T \) if \( \exists q \in \mathbb{R}^D \) s.t. \( \|q - p\| < \varepsilon \) and \( q \) is visible from \( T \). In other words, moving \( p \) a distance shorter than \( \varepsilon \) will make it visible.

Given a point cloud and a viewpoint, we want to determine all visible (or, equivalently, hidden) points. This can be solved with the method developed by Katz et al. consisting of two main steps:

1. Transform the points to a dual domain through inversion.

2. Compute the convex hull of the point set in the dual domain.

An operator that performs these two steps is what we call an HPR operator. The only thing to select freely is the kernel function. To explain what kernel function can be used, let the viewpoint given by the position of \( T \) define the origin of the coordinate system without loss of generality, and let us define a general transformation function, as

\[
F_f(p_j) = \begin{cases} 
\frac{p_j}{\|p_j\|} f\left(\|p_j\|\right), & \|p_j\| \neq 0, \\
0, & \|p_j\| = 0,
\end{cases}
\]  

(2.11)
where \( f : \mathbb{R}^+ \rightarrow \mathbb{R}^+ \) and \( F_f : \mathbb{R}^D \rightarrow \mathbb{R}^D \) [40]. Katz and Tal show that a transformation function \( f(\cdot) \) that is assumed to be invertible, satisfying the following properties

1. \( f'(\|p_j\|) < 0, \)
2. \( f(\|p_j\|) > 0, \)
3. If \( f(\cdot) \) is \( \gamma \)-controlled, it must hold that for any pair \( \|p_j\|, \|p_k\| \in \mathbb{R}^+ \) s.t. \( \|p_j\| > \|p_k\| \), and for any \( 0 < \epsilon < 1, \exists \) a \( \gamma \) s.t.

\[
1 - \epsilon < \frac{f(\|p_j\|)}{f(\|p_k\|)} < 1
\]

constitute a valid transformation function [40].\(^5\) Both kernels introduced in (2.13) and (2.14) are \( \gamma \)-controlled which means that they are controlled by a parameter \((2R\) and \( \gamma \), respectively).

One example of such a function is the spherical flipping function where

\[
f(\|p_j\|) = \|p_j\| + 2(R - \|p_j\|).
\]

(2.13)

If the radius \( R > \|p_j\| \), the function mirrors \( p_j \) in the circle’s boundary to a point outside the circle. This function clearly passes the three conditions set by Katz and Tal given that \( 2R > \max_{p_j \in P} \|p_j\| \). For the proof, see Appendix A.1.1.

Another example of a common transformation function is the exponential inversion function

\[
f(\|p_j\|) = \|p_j\|^{\gamma}, \quad \gamma < 0.
\]

(2.14)

We prove that the three conditions proposed by [40] are met for this kernel in Appendix A.1.2.

After the transformation to the dual domain with a suitable\(^6\) function, the convex hull should be calculated. This hull is derived from the union of the points in the dual domain and the viewpoint [41]. For the spherical flipping function, Katz et al. [41] presents the following lemma.

**Lemma 2.9.** The point set marked visible by the HPR operator is equal to the true visible point

\[
\begin{cases}
\text{as } R \to \infty \text{ and} \\
\text{all } p \in P \text{ is a } 0\text{-sample,} \\
\text{if } L = \inf\{\|p_j - C\|; \ p_j \in P\} > 0, \text{ where } C \text{ is the position of the pose } T \ [41].
\end{cases}
\]

Lemma 2.9 suggests that points close to the viewpoint can be problematic for the operator to handle. Another crucial aspect to consider when implementing the operator is that as \( R \) increases, the number of points marked visible by the operator also increases [41]. This implies that there will be more true positives

\(^5\)According to Katz and Tal, these are sufficient conditions, whereas identifying the necessary conditions remains a potential topic for future research [40].

\(^6\)What a suitable transformation function is, likely depends on the use case and is not investigated in the study of [40].
and false positives. The opposite effect occurs for small radii, suggesting that the radius must be carefully selected, which is also emphasized by Katz et al..

When implementing the HPR operator using the spherical flipping kernel, the result shown in Figure 2.4 can be achieved. The complete point cloud has been transformed into a sampled version from the given viewpoint. After the filtration, it is much easier to discern the direction the bunny is facing. It is also evident that the operator works well but that some artifacts, particularly close to silhouettes (e.g., by the right ear), do appear.

![Figure 2.4: The left point cloud is the Stanford Bunny from [3] scanned with the zipper technique from [75]. The right point cloud is our implementation of the HPR operator applied on the bunny with the spherical flipping kernel.](image)

2.4 Deep Neural Networks for Point Clouds

The field of machine learning has, over the past years, largely been dominated by the concept of neural networks. For sensor data applications, deep neural networks have played a crucial role, primarily enabled by the increasing availability of training data and computational resources, both regarding software and hardware [33]. Neural networks are inspired by the human brain structure, with the research roots going back to the mid-20th century with Rosenblatt [69] introducing the perceptron. However, it was not until 2012 that Krizhevsky et al. [42] introduced the AlexNet architecture for image classification, which drastically improved the performance in the ImageNet [23] LSVRC-2010 contest. Since then, convolutional neural networks have been one of the main methods for classification tasks. Over the previous years, the attention mechanism has gotten
massive recognition, initially through the paper "Attention Is All You Need" by Vaswani et al. [76], which incorporated it into their proposed architecture, the Transformer, for natural language processing. The success of the transformer architecture has not been limited to natural language processing tasks, as seen in products like ChatGPT [17]. It has also been adapted to, for example, image [24] and point cloud data [34, 88].

Basic knowledge of deep learning is assumed to be known by the reader. However, for a refresher on selected parts, we refer to the following books [14, 33, 38]. Especially to fully grasp the discussed architectures of the thesis, concepts such as multilayer perceptrons (MLPs), activation functions, residual networks, pooling, and loss functions should be familiar.

2.4.1 Cross-Entropy Loss

In a neural classification network, the outputs are called logits, which are non-normalized predictions. To turn these into class probabilities, the softmax activation function [38] is often used such that

\[
p_{ij} = \text{Softmax}_{\text{col}}(x_{ij}) = \frac{e^{x_{ij}}}{\sum_{k=0}^{m-1} e^{x_{ik}}},
\]

(2.15)

where

- \(x_{ij}\) is the logit for sample \(i\) and class \(j\),
- \(p_{ij}\) is the class probability for sample \(i\) and class \(j\),
- \(m\) is the number of classes.

Here, the softmax function is applied column-wise. To apply it row-wise, it should be changed to

\[
p_{ij} = \text{Softmax}_{\text{row}}(x_{ij}) = \frac{e^{x_{ij}}}{\sum_{k=0}^{n-1} e^{x_{kj}}},
\]

(2.16)

where \(n\) is the number of rows. With the column-wise softmax operator, the cross-entropy loss can then be written as

\[
H(y, p) = -\frac{1}{n} \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} y_{ik} \ln p_{ik},
\]

(2.17)

where \(n\) is the batch size [38]. For each sample \(i\), \(y_i\) is its one-hot vector, where all entries are zero except for the correct label, which is set to one. It is common to use the cross-entropy loss in neural networks, as minimizing the cross-entropy is equivalent to performing maximum likelihood under the model parameters [33].
A special case of this is the binary cross-entropy loss written as

\[ H(y, p) = -\frac{1}{n} \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} y_{ik} \ln p_{ik} = -\frac{1}{n} \sum_{i=0}^{n-1} (y_{i0} \ln p_{i0} + y_{i1} \ln p_{i1}) \]

\[ = -\frac{1}{n} \sum_{i=0}^{n-1} (y_{i0} \ln p_{i0} + (1 - y_{i0}) \ln(1 - p_{i0})). \]  

(2.18)

### 2.4.2 One-dimensional Wasserstein Loss

Even though the cross-entropy loss has benefits, including a solid theoretical foundation and simplicity, it does not account for the potential inherent order between classes [36]. This is handled by the Wasserstein loss, where the implementation and computational complexity become remarkably low in one dimension [83]. The formula for the discrete Wasserstein loss in one dimension is

\[ \mathcal{L}_{W}^{1}(y, p) = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{k=0}^{m-1} |Y_{ik} - P_{ik}|, \]

(2.19)

where \( Y_{ik} \) and \( P_{ik} \) are the cumulative distribution functions of \( y_{ik} \) and \( p_{ik} \), respectively, such that \( Y_{ik} = \sum_{l=0}^{k} y_{il} \) and \( P_{ik} = \sum_{l=0}^{k} p_{il} \) [83]. The Wasserstein distance is further explained in the context of information theory in Section 2.5.1.

### 2.4.3 Evaluation Metrics

We will evaluate the model using accuracy and confusion matrices. Both metrics are described below, with Figure 2.5 as an illustration of a confusion matrix in multinomial classification.

Accuracy is probably the most common metric in classification. It quantifies how well the classifier performs across the entire dataset and is defined as [64]:

\[ \text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}. \]

(2.20)

For an individual class \( i \), this can be expressed as

\[ \text{Accuracy}_i = \frac{\text{Number of Correct Predictions for Class } i}{\text{Total Number of Predictions for Class } i}. \]

(2.21)

From the confusion matrix (see Figure 2.5), we deduce:

\[ \text{Accuracy} = 0.440 \text{ and Accuracy}_{0.3} = \{13/20, 7/26, 7/24, 17/30\}. \]

A confusion matrix in multinomial classification helps us understand what happens when the classifier makes a mistake. Ideally, if neighboring classes (e.g., class 0 and 1) are more similar than non-neighboring classes (e.g., class 0 and 3), we would prefer misclassifications between neighboring classes rather than non-neighboring ones. This is also what we observe in practice (see Figure 2.5).
2.4.4 The Attention Mechanism

In image analysis, the attention mechanism is used to find the most important parts of an image while neglecting irrelevant data [35]. Vaswani et al. [76] introduced a scalar attention called "Scaled Dot-Product Attention" which they mathematically formulated as

$$
\text{Attention}(Q, K, V) = \text{Softmax}_{\text{row}} \left( \frac{Q K^T}{\sqrt{D_k}} \right) V,
$$

(2.22)

where

- $Q$ is a matrix of query vectors on the rows (e.g., the embedding of words).
- $K$ is a matrix of key vectors on the rows (e.g., the embedding of the context of the words).
- $V$ is a matrix of value vectors on the rows associated with one key vector such that every value vector determines the relevance of the associated key vector. If we disregard the softmax operator (see (2.16)), we can illustrate this with

$$
K^T V = \begin{bmatrix} k_0^T & \ldots & k_{n-1}^T \end{bmatrix} \begin{bmatrix} v_0 \\ \vdots \\ v_{n-1} \end{bmatrix} = \sum_{i=0}^{n-1} k_i^T v_i.
$$

(2.23)

- $D_k$ is the dimension of the vectors.

The scalar product for each query and key vector $q_i k_j^T$, $i, j \in \{0, \ldots, n - 1\}$, gives the similarity of the embedded representations. To prevent the softmax operator from having too small gradients, the scale factor $\frac{1}{\sqrt{D_k}}$ was proposed in the paper. Lastly, the softmax operator row-wise maps the scaled similarity matrix to a set of probability distributions which are the weights for the values $V$ mapped to the output.

Figure 2.5: An example of how a confusion matrix might appear for four classes. Class 0 represents the error-free bin, while the error magnitude in subsequent bins increases with their class number.
2.4.5 The Point Transformer Layer

In this work, we want to leverage the advancements of the transformer architecture adapted to point clouds. This has been done in [34, 88] with promising results. Zhao et al. [88] claim that the self-attention operator should be particularly useful for point cloud data as the operator is invariant to permutations and the cardinality of the input. Note that there is a follow-up paper on [88] by Wu et al. with some updates and improved performance [81]. However, the performance gains are relatively small. Hence, due to increased complexity and a less tested code base, this paper is not investigated thoroughly but should be reconsidered in future work.

Now, let \( \mathcal{X} \in \mathbb{R}^{D \times N} \) be a set of \( N \) feature vectors of dimension \( D \). The authors Zhao et al. propose to use vector self-attention formulated as

\[
y_i = \sum_{x_j \in \mathcal{X}(i)} \rho \left( \gamma(\varphi(x_i) - \psi(x_j) + \delta(p_i, p_j)) \right) \odot (\alpha(x_j) + \delta(p_i, p_j)),
\]

where

- \( \mathcal{X}(i) \subseteq \mathcal{X} \) is the set of feature vectors corresponding to the \( k \)-nearest neighbors of \( p_i \) (including the point itself),

- \( \rho(\cdot) \) is some normalization operator such as the softmax operator with matrix normalization as in (2.22),

- \( \gamma(\cdot) \) is an MLP with two linear layers and one ReLU activation function,

- \( \varphi(\cdot) \) is the query,

- \( \psi(\cdot) \) is the key,

- \( \delta(p_i, p_j) = \theta(p_i - p_j) \) is the position encoding\(^7\) where \( \theta(\cdot) \) is an MLP with two linear layers and one ReLU activation function,

- \( \odot \) denotes the Hadamard product,

- \( \alpha(\cdot) \) is the value.

Differently from (2.22), (2.24) calculates similarity through subtraction, \( \gamma(\cdot) \), and positional embedding instead of a scalar product. This produces a point transformer layer that transforms the input set of feature vectors \( \mathcal{X} \) into an output \( \mathcal{Y} \in \mathbb{R}^{D \times N} \), maintaining the input’s dimension.

---

\(^7\)Encoding positional information through relative positions is merely one among numerous approaches for incorporating spatial awareness. Zhao et al. conducted an ablation study on position encoding for semantic segmentation, concluding that relative position encoding is superior to other variants, such as absolute position encoding or the absence of position encoding [88].
2.4.6 Classification with Point Transformer

Utilizing the point transformer layer, Zhao et al. [88] construct a classification network and test it on ModelNet40 [82]. At the time of the publication, September 26, 2021, the network achieved state-of-the-art performance, considering both overall and mean accuracy. Their proposed architecture can be seen in Figure 2.6. The point transformer block, as well as the transition down block, are thoroughly explained by Zhao et al. [88]. For more information on the code from the authors, see [1] or another implementation [86], which we have studied more extensively for its simplicity.\(^8\)

Briefly, the point transformer block consists of a linear layer, the point transformer layer, another linear layer, and a residual connection from the input to the output. Thus, in this block, the dimensionality of the feature data is kept. Secondly, the transition down block consists of farthest point sampling (FPS\(^9\)) (with a rate of four), a kNN graph (based on the input point cloud to this block) fed into an MLP with a linear layer, batch normalization, and ReLU, succeeded by max pooling of all the $k$-nearest neighbors. This results in a downsampling factor of four for the block. Finally, all the features are fed through a global average pooling layer (acting on each feature channel separately) followed by an MLP with three linear layers and two ReLU activations, which outputs class scores, where the max score is chosen as the class prediction.

2.5 Information Theory

In this section, we discuss methods for measuring information within point cloud neighborhoods and relationships among them. These approaches can be helpful in scene classification of point clouds as they offer a compact description of neighborhoods while still capturing a significant amount of information.

2.5.1 Sinkhorn Divergence

When analyzing continuous densities or discrete distributions, it is essential to have a notion of distance between pairs of measures. For instance, one might want to determine the distance between point sets, as shown in Figure 2.7. Approaches using Kullback-Lieber relative entropy are unsuitable as they do not consider the distance between two measures with non-overlapping support [28]. For example, Dirac deltas will have the same relative entropy at varying distances. Naturally, this can make it difficult for iterative approaches to converge since progress cannot be measured. Two metrics that overcome this obstacle are the Optimal Transport (OT) distances and the Maximum Mean Discrepancies (MMD

---

\(^8\)The official code in [1] is not the exact one used in the paper by Zhao et al.. Therefore, we primarily used another implementation [86] because it was simple to use and had documented performance. Consequently, the exact layout of the network may differ. This was, for example, noted in the different implementations of the MLP.

\(^9\)The method aims to find a well-spread fixed size subset of points from a point cloud such that the points are as far away from each other as possible. This implementation is from Qi et al. [62].
Figure 2.6: A representation of the classification architecture presented by Zhao et al. [88]. The input is a point cloud of dimension $D$ (besides the $(x, y, z)$ vectors, more features can be associated with the points) for a set of $N$ points. Then, the first five double-blocks work as the backbone, encoding features and progressively downsampling the point cloud. On top of the arrows, we have $(N_i, D_i)$, which denotes the number of points and feature channels after block $i$. Finally, global average pooling gives one value per channel. All channels are then input to an MLP, which outputs logits for the given classes. This visualization is largely inspired by the corresponding one in [88].

The OT distance between a pair of probability measures can be defined as the Wasserstein distance

$$W(\alpha, \beta) = \min_{\pi \in \Pi(\alpha, \beta)} \int_{X \times Y} c(x, y) d\pi(x, y),$$

(2.25)

where

- $\alpha, \beta$ are probability measures,
- $c(x, y)$ is the cost of moving a unit of mass from source $x$ to target $y$,
- $\pi(x, y)$ is the amount of mass moved from $x$ to $y$, and
- $\Pi(\alpha, \beta)$ denotes the set of transport plans between the measures, ensuring all mass is transferred from source to target without negativity [32, 47].

Unfortunately, the computational and sample complexity of this method (as can be seen in Table 2.2) results in lacking scalability to larger feature sets, both regarding cardinality and dimensionality. Beyond computational and statistical issues, the OT approach is unstable, non-smooth, and challenging to handle in high dimensions [31]. Even if MMD is significantly faster and is beneficial in several computational aspects, it lacks the desirable geometric nature inherent in the OT framework [28, 31].

Fortunately, in the year 2013, Cuturi presented an efficient implementation of an entropic regularization approach to the OT problem using Sinkhorn’s matrix scaling [21]. This definition and description of the Sinkhorn distance have since
Table 2.2: The table presents computational and sample complexities for OT and MMD as detailed in [32]. Regarding computational complexity, $n$ is the number of drawn samples from the two distributions. For the sample complexity, which specifies the error when estimating the distances between measures using samples, $n$ denotes the size of the sample sets, while $D$ represents the dimensionality of the space.

<table>
<thead>
<tr>
<th>Computational Complexity</th>
<th>MMD</th>
<th>OT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample Complexity</td>
<td>$O\left(\frac{1}{\sqrt{n}}\right)$</td>
<td>$O\left(n^{-1/D}\right)$</td>
</tr>
</tbody>
</table>

been widely used in the field of OT. The formulation of entropic regularization can be defined as

$$W_{\epsilon}(\alpha, \beta) = \min_{\pi \in \Pi(\alpha, \beta)} \int_{X \times Y} \left( c(x, y) + \epsilon \ln \left( \frac{d\pi(x, y)}{d\alpha(x) d\beta(y)} \right) \right) d\pi(x, y), \quad (2.26)$$

where $\epsilon > 0$ ensures that the second term aims to maximize the relative entropy, yielding a smoother solution with both computational and sample complexities substantially lower than those of the classical Wasserstein distance [21, 28, 32].

Even though the entropic regularization of the OT problem could be seen as an improvement from its predecessor, it entails an entropic bias [28]. In essence, the regularized transport plan operates under the assumption that points in the target distribution are affected by Gaussian noise [68]. Consequently, the gradient of $W_{\epsilon}$ causes the support of the source measure to be smaller than that of the target measure [28, 68]. This can be advantageous in cases where data is anticipated to be affected by Gaussian observation noise that can be quantified. Successfully estimating the expected noise allows for a proper choice of $\epsilon$. However, the presence of a bias where $W_{\epsilon}(\alpha, \alpha) \neq 0$ also calls for a better method to address optimal transport without such bias [28]. A solution to this problem is suggested by Genevay et al. [31] in the form of Sinkhorn divergence

$$S_{\epsilon}(\alpha, \beta) = W_{\epsilon}(\alpha, \beta) - \frac{1}{2} W_{\epsilon}(\alpha, \alpha) - \frac{1}{2} W_{\epsilon}(\beta, \beta), \quad (2.27)$$

which (given a few requirements, see Theorem 1 in [28]) ensures that

$$0 = S_{\epsilon}(\alpha, \alpha) \leq S_{\epsilon}(\alpha, \beta), \quad (2.28)$$

$$\alpha = \beta \iff S_{\epsilon}(\alpha, \beta) = 0, \quad (2.29)$$

$$\alpha_n \to \alpha \iff S_{\epsilon}(\alpha_n, \alpha) \to 0. \quad (2.30)$$

In fact, this method interpolates between OT ($\epsilon = 0$) and MMD ($\epsilon = \infty$) [28, 32, 65]. Figure 2.7 shows the Sinkhorn divergence which has been computed with the efficient GPU (Graphics Processing Unit) implementation geomloss [28], where $c(x, y) = \frac{1}{2}||x - y||^2_2$ and $\epsilon = 0.05$ is used.
2.5 Information Theory

2.5.2 Differential Entropy

We define what differential entropy is based on [20]. Let $X = X_0, X_1, \ldots, X_{D-1}$ be a sequence of random variables with density $f(x) = \{x_0, x_1, \ldots, x_{D-1}\}$. Noteworthy is that when $X_i$ is continuous, Shannon entropy is called differential entropy [33]. We can now define what differential entropy is.

**Definition 2.10 (Differential Entropy).** The differential entropy of a set $X$ of random variables with density $f(x)$ is defined as

$$h(X) = - \int f(x) \log (f(x)) \, dx.$$  

Continuing, we state the theorem of the differential entropy of a multivariate normal distribution followed by proving it (for the proof, see Appendix A.2.1), partially based on [20]. This is essential to understand in order to fully grasp the alignment quality measure by Adolfsson et al. [6].

---

Note that there is a typo in [6]'s definition of the differential entropy later corrected in [7].
**Theorem 2.11.** The differential entropy of a multivariate normal distribution for a set $X$ with mean $\mu$ and covariance matrix $\Sigma$ is

$$h(X) = \frac{1}{2} \ln \left( (2\pi e)^D \det(\Sigma) \right) \text{ nats.} \quad (2.32)$$

Additionally, a few other important properties of differential entropy are stated.

1. Differential entropy is not always non-negative and can therefore not be a direct measurement of information as entropy can [30].

2. Differential entropy is shift invariant: $h(X + c) = h(X)$, $c \in \mathbb{R}^D$ [20, 59].

3. Differential entropy is not scale invariant: $h(AX) = h(X) + D \ln \det(A)$, $A \in \mathbb{R}^{D \times D}$ [20, 59].
Evaluation of map quality constitutes a critical component of SLAM. As demonstrated by [25, 26], evaluating map quality is vital when comparing 3D reconstruction models. They use the mean map entropy metric to argue why their method is superior to other ones. The motivation for employing this metric is its ability to measure the sharpness or crispness of a map. However, the assumption that a crisper map invariably equals a higher-quality map is not always accurate. This statement is supported by the study of Adolfsson et al. [6], which illustrates the insufficiency of the mean map entropy metric as a comprehensive measure of map quality. Adolfsson et al. introduce a method called CorAl, inspired by mean map entropy and superior to it. Because of that and because the differential entropy feature will be included in our feature map, CorAl will be our baseline. To start with, we give an overview of the related work. Afterward, CorAl is presented in detail.

### 3.1 Overview of Related Work

The complexity of the techniques encountered in the related work varies significantly. Some methods, [6, 7, 11, 16, 25, 26, 29, 54, 85], use relatively simple features measuring alignment quality. In contrast, other methods, such as [9, 10, 19], adopt more complex approaches.

Among the simpler approaches, [11, 16, 54] focus on one or a few features, such as spatial overlap or projection of 3D points to virtual image planes, to quickly estimate alignment quality. Other methods input their features into a logistic regression model to classify a point cloud pair as aligned or misaligned [6, 7, 29, 85]. The works of [6, 7, 25, 26] utilize entropy-based features to assess point cloud alignment quality.

Of the more complex methods, [9, 10] examine residual errors between points
and the map to which they belong. They study this in the context of Markov random fields with fully connected latent variables to detect localization failures. Conversely, Camous et al. [19] compare two DNN architectures inspired by kernel point convolution [74] and point-pillar encoders [45] to tackle the binary PCAC problem.

### 3.2 CorAl

We calculate the alignment quality measure proposed by Adolfsson et al. [6] based on differential entropy measurements. Now, we define the alignment quality measure, the difference in differential entropy\(^1\), as

\[
Q(P_0^0, P_1^0) = H_{\text{joint}}(P_{0,1}^0) - H_{\text{sep}}(P_0^0, P_1^0),
\]

where

\[
H_{\text{joint}}(P_{0,1}^0) = \frac{H_0(P_{0,1}^0)}{|P_{0,1}^0|},
\]

\[
H_{\text{sep}}(P_0^0, P_1^0) = \frac{H_0(P_0^0) + H_1(P_1^0)}{|P_0^0| + |P_1^0|} = \frac{H_0(P_0^0) + H_1(P_1^0)}{|P_{0,1}^0|}
\]

\[\approx \frac{1}{2} \left( \frac{H_0(P_0^0)}{|P_0^0|} + \frac{H_0(P_1^0)}{|P_1^0|} \right),\]

with equality if \(|P_0| = |P_1|\). The sum of differential entropies, \(H_i(P_i)\), is calculated as

\[
H_i(P_i) = \sum_{\Omega_{ij} \subseteq P_i} h_i(\Omega_{ij}),
\]

for \(i = \{0, 1, (0, 1)\}\). Lastly, we obtain the pointwise differential entropy for a multivariate Gaussian distribution as presented in Theorem 2.11 and proven in Appendix A.2.1. We have

\[
h_i(\Omega_{ij}) = \frac{1}{2} \ln \left( (2\pi e)^D \det(\Sigma(\Omega_{ij})) \right),
\]

where \(\Sigma(\Omega_{ij})\) is the sample covariance of the point set \(\Omega_{ij}\). Note that \(D = 3\) since we work with three-dimensional spatial data. According to the definition of the sample covariance [84], \(\Sigma(\Omega_{ij})\) is then calculated as

\[
\Sigma(\Omega_{ij}) = \frac{1}{|\Omega_{ij}| - 1} \sum_{\mathbf{p}_{il} \in \Omega_{ij}} (\mathbf{p}_{il} - \bar{\mathbf{p}}_{ij})(\mathbf{p}_{il} - \bar{\mathbf{p}}_{ij})^T,
\]

where \(\bar{\mathbf{p}}_{ij} = \frac{1}{|\Omega_{ij}|} \sum_{\mathbf{p}_{il} \in \Omega_{ij}} \mathbf{p}_{il}\) is the sample mean.

Additionally, Adolfsson et al. add a few extra considerations to their method to handle problematic scenarios. These considerations are summarized below.

---

\(^1\)Note that difference in differential entropy is similar to Jensen-Shannon divergence [44, 53].
1. It is not good to implement (3.4) as it is since it can be numerically unstable. This will depend on the point cloud density, which is especially different when working with a mobile LIDAR compared to a terrestrial laser scanner. For mobile LIDAR, Adolfsson et al. adds an offset to the argument of the logarithm since it otherwise might be zero (or very close to zero) for which the logarithm is undefined (or unstable). The recommended implementation is instead

$$h_i(\Omega_{ij}) = \frac{1}{2} \ln \left( (2\pi e)^D \det(\Sigma(\Omega_{ij})) + \epsilon \right).$$

(3.6)

Note that the covariance matrix is symmetric positive semidefinite [22]. Thus, the determinant of the covariance matrix is non-negative. Because the product of the eigenvalues gives the determinant, degenerate cases appear when we have 2D spread data in 3D space, which frequently happens (e.g., surfaces are often flat). However, adding this positive offset is enough to handle this degenerate case. Also, note that some neighborhoods, $\Omega_{ij}$, will only contain one point, meaning the covariance is undefined. For that case, we set

$$h_i(\Omega_{ij} | |\Omega_{ij}| = 1) = \frac{1}{2} \ln (\epsilon) \leq \frac{1}{2} \ln \left( (2\pi e)^D \det(\Sigma(\Omega_{ij})) + \epsilon \right)$$

(3.7)

since $\ln(\cdot)$ is strictly increasing and $(2\pi e)^D \det(\Sigma(\Omega_{ij})) \geq 0$.

2. Adolfsson et al. [6] choose the radius dynamically based on the distance from the sensor, at least for the mobile LIDAR setup. This approach is adopted to account for the varying point cloud density of LIDAR scans. As a result, a shorter radius can be used while maintaining a sufficient representation of the neighborhood. The radius is chosen as

$$r = \begin{cases} 
  d \sin(\alpha), & \text{if } r_{\min} < d \sin(\alpha) < r_{\max}, \\
  r_{\min}, & \text{if } d \sin(\alpha) \leq r_{\min}, \\
  r_{\max}, & \text{if } r_{\max} \leq d \sin(\alpha), 
\end{cases}$$

(3.8)

where $\alpha$ is the vertical angular resolution of the sensor, and $d$ is the distance between a point and its corresponding sensor.

3. Furthermore, the authors suggest removing the $E_{\text{reject}}$ percent of points with the least entropy. This leads to degenerate cases having no impact.

4. Adolfsson et al. set a threshold such that if the overlap between the point clouds is lower than 10%, the pair gets classified as misaligned due to lack of evidence to measure alignment [6].

The parameters to tune are $\epsilon$, $r_{\min}$, $r_{\max}$, $\alpha^2$, and $E_{\text{reject}}$. To obtain predictions based on the differential entropy features, a logistic regression model for binary

\footnote{Selecting $\alpha$ directly is problematic as the resolution of the sensor is fixed. We will instead choose to scale $\alpha$ with some factor $l$.}
classification [38] is used where

\[ z = \beta_0 + \beta_1 x_1 + \beta_2 x_2, \quad p = \frac{1}{1 + e^{-z}}, \quad y_{\text{pred}} = \begin{cases} \text{aligned} & \text{if } p \geq t_h, \\ \text{misaligned} & \text{if } p < t_h, \end{cases} \]  

(3.9)

and \( x_1 = H_{\text{joint}}(P_0^0, 1), x_2 = H_{\text{sep}}(P_0^0, P_1^0) \), \( p \) is the class probability, and \( t_h \) is the chosen threshold. The suitable choice for the threshold is application dependent and, in particular, dependent on the relative importance of the precision and recall metrics. Finally, they test their alignment metric both in an indoor warehouse environment using the Velodyne HDL32E sensor as well as on the ETH registration dataset measured with a terrestrial laser scanner [6, 60].
The main focus of this chapter is to introduce our multinomial point cloud alignment classifier, FACT (Feature-Aware Classification Transformer). We begin by presenting the data that will be used for the evaluation. Secondly, we discuss some details in implementing the baseline CorAl for the binary classification task. Subsequently, we explain the pipeline of FACT, including two modules: feature extraction and classification. Finally, we go through the tuning methodology for the parameters of FACT.

4.1 Data

During this thesis, we have utilized the nuScenes dataset [18], released in March 2019, as the primary data source to develop and test the various point cloud alignment classifiers. nuScenes contains 1,000 scenes, of which 850 are designated for training and validation, while the remaining 150 are for testing. All scenes, except test scenes, have ground truth data. As we need ground truth in this project, 850 scenes are useful. By driving around Boston and Singapore, they have aimed to acquire a diverse dataset regarding class distribution, weather conditions, locations, and times of day. Each scene consists of 20 seconds of data capture, with the LIDAR operating at a scan frequency of 20 Hz. This implies that the useful dataset includes approximately $20 \times 20 \times 850 = 340,000$ LIDAR scans. Due to extensive computational requirements, we cannot exploit all this data. However, given our computational limitations, we will ensure to sample from as many scenes as possible to obtain as diverse a distribution as possible. Obviously, we do not sample training, validation, and test data from the same scenes.

The LIDAR sensor that is used is the Velodyne HDL32E which has a vertical angular resolution of $\alpha \approx 1.33^\circ$, a horizontal field of view of $360^\circ$, and a vertical field of view of $+10.67^\circ$ to $-30.67^\circ$ [2]. Each LIDAR scan has 32 channels in the
vertical direction and $1,080 \pm 10$ points per ring, resulting in each scan having around $32 \cdot 1,080 = 34,560$ 3D points [18].

In the dataset, a Renault Zoe car is utilized [18] with dimensions $4.084 \times 1.730$ (length×width) [4]. The LiDAR is mounted on top of the car and, based on our visual estimation, $0.375 \cdot 4.084 = 1.5315 \approx 1.5$ meters from the car’s rear. For the point cloud registration task, it is advisable to preprocess the point cloud by removing all points belonging to the car. Thus, excluding points within a radius of 1.5 meters seems reasonable, where we have seen that car data points are largely eliminated while the surrounding environment remains intact. There is no need to cover the car’s front as it seems to be occluded, and doing so risks removing important nearby static objects.

4.1.1 Data Generation

To distinguish between aligned and misaligned point clouds, we must systematically create misaligned point cloud pairs. Aligned point clouds are easily created using the ground truth transformations and the point clouds given in the WCS and LCS, respectively. These are then inserted into (2.6). Two alternative ways to create misaligned point clouds are:

1. Use a suitable registration method to estimate the pose transformation between the point clouds. If the error is beyond a certain threshold from the ground truth transformation, classify it as misaligned while remembering the magnitude of the error, as it should be predicted. Otherwise, repeat the procedure with a different starting solution to obtain a new result.

2. Add an offset to the ground truth transformation.

The first method is appealing because loss functions often become stuck in local minima with respect to the pose transformations, which are presumably more challenging to detect than random offsets. It is also appealing to test on such data as that is how the classifier will be used in practice. However, it is a more uncontrolled procedure, implying that producing useful misaligned data takes more time and effort with a considerable misaligned data distribution support.

This approach is intriguing but is not investigated further in this thesis. Instead, automatically offsetting the data in a controlled manner allows us to set up a multinomial classification task with a balanced class distribution, easily evaluated and benchmarked.

Automatic Misalignment Strategy

To create a useful dataset using data from nuScenes [18] and to remain consistent with the approach of Adolfsson et al. [6, 7], we automatically create misaligned point cloud pairs by perturbing one of the point clouds after alignment, as done in their work. An alternative would be to follow Camous et al. [19] and perturb the data according to a normal distribution such that the resulting error magnitudes will have continuous support instead of adhering to $m$ classes. While this approach may lead to a more diverse dataset, it would shift the problem from a
4.2 Implementing CorAl

In the absence of explicit details from Adolfsson et al. [6] regarding the loss function and the optimizer they use, we have opted to use the binary cross entropy loss (see (2.18)) and the stochastic gradient descent optimizer. To implement the logistic regression classifier, we use the MIT licensed code by Loevlie [48] as a starting point.

It is not stated in the papers [6, 7] how to define \( d \) in (3.8) when it concerns the joint point cloud. Nevertheless, we choose to use the same radius for each point in the separate and joint point cloud to get a suitable comparison between \( H_{\text{joint}}(\cdot) \) and \( H_{\text{sep}}(\cdot) \). Although, this might not be the best way to do it since the radius will only be a function of the density of one of the point clouds. We also incorporate a larger radius than used in CorAl. Even though they use the same LIDAR (Velodyne HDL32E) for one of their experiments, it takes place indoors where the mean and max distances to objects are shorter [6]. Thus, to get enough points in our neighborhoods, it is suitable to have more voluminous neighborhoods.

For every class we want to use, we specify an offset. Consider, for example, six classes with the following perturbations:

\[
(\gamma, e_d) \in \{(0, 0), (0.02, 0.2), (0.04, 0.4), (0.06, 0.6), (0.08, 0.8), (0.10, 1.0)\}.
\]

We make sure to have an expected balanced class distribution. To create an angular offset around the vertical axis, we use Rodrigues’ formula in (2.3) with \( \hat{n} = e_3 \).

Then we obtain

\[
R(\hat{n}, \gamma) = I_3 + \sin(\gamma)[\hat{n}]_x + (1 - \cos(\gamma))[\hat{n}]_x^2
\]

\[
= I_3 + \sin(\gamma)\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + (1 - \cos(\gamma))\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

\[
= \begin{bmatrix} 1 & -\sin(\gamma) & 0 \\ \sin(\gamma) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \cos(\gamma) - 1 & 0 & 0 \\ 0 & \cos(\gamma) - 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

\[
= \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\]

4.2 Implementing CorAl

In the absence of explicit details from Adolfsson et al. [6] regarding the loss function and the optimizer they use, we have opted to use the binary cross entropy loss (see (2.18)) and the stochastic gradient descent optimizer. To implement the logistic regression classifier, we use the MIT licensed code by Loevlie [48] as a starting point.

It is not stated in the papers [6, 7] how to define \( d \) in (3.8) when it concerns the joint point cloud. Nevertheless, we choose to use the same radius for each point in the separate and joint point cloud to get a suitable comparison between \( H_{\text{joint}}(\cdot) \) and \( H_{\text{sep}}(\cdot) \). Although, this might not be the best way to do it since the radius will only be a function of the density of one of the point clouds. We also incorporate a larger radius than used in CorAl. Even though they use the same LIDAR (Velodyne HDL32E) for one of their experiments, it takes place indoors where the mean and max distances to objects are shorter [6]. Thus, to get enough points in our neighborhoods, it is suitable to have more voluminous neighborhoods.

For every class we want to use, we specify an offset. Consider, for example, six classes with the following perturbations:

\[
(\gamma, e_d) \in \{(0, 0), (0.02, 0.2), (0.04, 0.4), (0.06, 0.6), (0.08, 0.8), (0.10, 1.0)\}.
\]

We make sure to have an expected balanced class distribution. To create an angular offset around the vertical axis, we use Rodrigues’ formula in (2.3) with \( \hat{n} = e_3 \).

Then we obtain

\[
R(\hat{n}, \gamma) = I_3 + \sin(\gamma)[\hat{n}]_x + (1 - \cos(\gamma))[\hat{n}]_x^2
\]

\[
= I_3 + \sin(\gamma)\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + (1 - \cos(\gamma))\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

\[
= \begin{bmatrix} 1 & -\sin(\gamma) & 0 \\ \sin(\gamma) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \cos(\gamma) - 1 & 0 & 0 \\ 0 & \cos(\gamma) - 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

\[
= \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\]
When implementing the CorAl method, the following parameters have to be set:

\[
\{r_{\text{min}}, r_{\text{max}}, \epsilon, l, E_{\text{reject}}\},
\]

where \( l \) is the factor we scale the vertical resolution \( \alpha \) with. These parameters have been tuned partially using Bayesian optimization through Ax [66, 67], which is powered by BoTorch [12]. We found the following settings giving the best performance:

\[
\{r_{\text{min}} = 1, r_{\text{max}} = 5, \epsilon = e^{-18}, l = 5, E_{\text{reject}} = 0.20\}.
\]

### 4.3 Our Point Cloud Alignment Classifier - FACT

This section introduces our point cloud alignment classifier FACT, whose name summarizes its two main modules: feature extraction (Feature-Aware) and classification (Classification Transformer). The flowchart in Figure 4.1 gives an overview of the computer program implemented in this thesis.

#### 4.3.1 Module 1: Feature Extraction

The overall algorithm for the feature extraction part is presented in Algorithm 1. We then follow up by explaining its components in the subsequent sections. The function `covisible_points()` assigns a weight \( c_j \in [0, 1] \) to each point, representing its co-visibility score. Moreover, with the information \( P_{0,1}, i_{P_0} \) (\( i_{P_0} \) denotes the
indices to the first point cloud in the joint point cloud), and $T_1^0$, we can represent the two separate point clouds, the joint point cloud, and the transformation estimate.

**Algorithm 1** Overview of Feature Extraction for One Point Cloud Pair

1. **Input:** $P_0^0, P_1^1, T_1^0, N_{FPS}$
2. **Output:** $G_{FPS}$ (Feature Map)
3. $P_0^0, P_1^1 = \text{preprocess}(P_0^0, P_1^1)$
4. $\hat{P}_{0,1}, i_{\hat{P}_0} = \text{get_joint_point_cloud}(P_0^0, P_1^1, T_1^0)$
5. $\hat{P}_{0,1}, c, i_{\hat{P}_0} = \text{covisible_points}(\hat{P}_{0,1}, i_{\hat{P}_0}, T_1^0) \# c_j \in [0, 1]$;
6. $i_{FPS} = \text{farthest_point_sampling}(\hat{P}_{0,1}, i_{\hat{P}_0}, N_{FPS})$
7. $\pi_{FPS} = \text{set_labels}(i_{FPS}) \# \pi_j \in \{0, 1\}$
8. $P_{batches}, i_{batches} = \text{get_batches}(\hat{P}_{0,1}, i_{FPS})$
9. $F_{FPS} = []$
10. $W_{FPS} = []$
11. $k = 0$
12. for $P_{batch}, i_{batch}$ in zip($P_{batches}, i_{batches}$):
13. $N_{batch} = \text{len}(P_{batch}) \# \text{batch size}$
14. $b = \{k \cdot N_{batch}, \ldots, (k + 1) \cdot N_{batch} - 1\} \# \text{indices to batch samples}$
15. $w_b = \text{get_joint_cardinality_ratio}(P_{batch}, \hat{P}_{0,1}) \# \frac{|\hat{\Omega}_b|}{N_{0,1}}$
16. $w_{ib} = \text{get_sep_cardinality_ratio}(P_{batch}, \hat{P}_{0,1}, i_{batch}, i_{\hat{P}_0}) \# \frac{|\hat{\Omega}_{ib}|}{N_i}$
17. $W_{FPS}.append([w_b, w_{ib}])$
18. $F_{FPS}.append([h_b, h_{ib}, S_{ib}])$
19. $k = k + 1$
20. $W_{FPS} = \text{concat}(W_{FPS})$
21. $G_{FPS} = \text{concat}(F_{FPS}, \pi_{FPS}, W_{FPS})$
22. return $G_{FPS}$
Classical Preprocessing of Point Clouds

We preprocess the point clouds by:

1. Removing close points\(^1\)
   (a) These are points that belong to the car (as argued in Section 4.1, a 1.5m threshold is suitable).
   (b) This makes everything run faster.
   (c) This is in line with Lemma 2.9. The closest points will be further away, meaning the validity of the lemma strengthens. In practice, it holds for lower \(R\) values, which is desirable.

2. Subsampling using farthest point sampling
   (a) Use farthest point sampling to select a subset of points for feature calculation. When calculating features for a point based on its neighborhood, we consider the full neighborhood (i.e., all points in the original point cloud). While the farthest point sampling may take a little time, calculating features for only a subset of points can result in a dramatic speed-up.
   (b) For the classifier FACT, large point clouds cannot be considered during training due to limited VRAM (Video Random Access Memory) memory on our GPU. Consequently, reducing the memory requirement becomes a necessity.

Co-visibility for Point Clouds

In this project, we use the HPR operator to remove points estimated not to be visible while keeping co-visibility scores, \(c\), for the rest of the points. The motivation is that comparing not co-visible areas will not help describe misalignments as reasoned in Section 2.3.1. In the theory section, we introduced two visibility kernel with one parameter each. These two will be tuned and compared. For more information on this, see Section 4.4. Additionally, removing the points marked as hidden increases the computational speed of the feature extraction. When using the co-visibility estimate as a weight for each point, several problems must be dealt with. The estimation of the co-visibility might be wrong in itself due to sampling artifacts, concavities, and noise. In fact, the most problematic scenario for the spherical flipping operator to handle is deep concavities and silhouettes [41]. Katz and Tal [40] provide a criterion for up to what degree of curvature the general\(^2\) HPR operator still can find visible points. As the estimation of co-visibility is dependent on the estimated transformation, the estimated co-visible points will be wrong if the transformation estimate is wrong, which is the problem we want to investigate.

---

\(^1\)This will also be done for our baseline CorAI as it increases its performance and is a critical preprocessing specific to this dataset.

\(^2\)The general HPR operator is an extension of the HPR operator [40].
Studying case II, delineated in Section 2.3.1 as the only relevant case, we start investigating same surface estimations. Observing Figure 2.1, we discern an example of an aligned point cloud pair (a) and a misaligned point cloud pair (b). Notably, in the misaligned instance, the house surfaces (see the upper middle part of the image) do not align. Studying the joint point cloud scenario, the red point cloud would presumably be considered co-visible given that the poses of the sensors are on the same side of the house, which would be most likely for consecutive point clouds. Conversely, the blue point cloud might not be co-visible, resulting in minimal co-visible weights or even exclusions. In this setup, a red point on the house will not have blue points with high weights in its neighborhood, leading to significant differences between the distributions. This disparity allows a classifier to easily conclude that there are local signs of misalignment. Regarding the case of the blue and the green point cloud, most points will be marked co-visible. Thus, we will measure similar local distributions and, therefore, get local signs of alignment. This means that estimating co-visibility helps a classifier distinguish between aligned and misaligned point cloud pairs.

Examining Figure 4.2, we see that the binary co-visibility estimation works rather well for the Stanford Bunny. However, a few of the points that should be visible are not displayed as visible.
Method (a) Bunny first viewpoint. 

(b) Bunny second viewpoint. 

(c) Approximation of the ground truth of the co-visible points from the first and the second viewpoint. 

(d) Estimation of the co-visible points from the first and the second viewpoint. 

**Figure 4.2:** In this figure, co-visible point estimation using [41] on the Stanford Bunny [3] is shown. Two viewpoints are chosen to maintain a wide baseline while ensuring a sufficiently large co-visible region. As the true co-visible region shrinks, the task of solving co-visibility likely becomes more difficult. Figure 4.2c uses [41] with known one-to-one correspondences between the viewpoints, simplifying the task since co-visible points are those visible in both views. In contrast, the result visualized in Figure 4.2d is based on the estimated co-visible points in the joint point cloud. The exponential inversion kernel with $\gamma = -0.0001$ was used.
To determine the degree of co-visibility, we incorporate visibility scores, inspired by Katz and Tal [40], detailed in Algorithm 2\textsuperscript{3}. The input to the algorithm is the convex hull\textsuperscript{4} of the point cloud in the dual domain (the visible points) and the viewpoint $C$. The reasoning behind our algorithm comes from the following ideas where we use Figure 4.3 as reference (see the notation in the figure):

1. If the green point lies on the convex hull of the points in the transformed domain, we investigate its degree of visibility.

2. The green point is more likely to be visible if the following holds
   
   \begin{itemize}
   \item It lies in front of its neighbors in the original domain.
   \item If $\beta$ is small.
   \item If $\gamma$ is large.
   \item $\gamma - \beta$ are on average large when considering the green point and all its neighbors $\iff$ \text{mean}(\gamma - \beta) is large
   \end{itemize}

In Figure 4.4, Algorithm 2 is applied to point cloud data with multiple objects. The figure suggests that the algorithm works well in practice and has no problem handling several point cloud objects. An important observation is that the choice of kernel for the HPR operator need not influence the construction of a visibility score. Since the visible points invariably lie on the convex hull in the transformed domain, a visibility score should primarily gauge "by how large of a margin" a point either lies on or off the convex hull. Another approach for determining visibility scores could involve calculating how far a point needs to be moved to either lie on or off the hull. For instance, if a point on the convex hull has to be moved a significant distance to leave the hull, then it is highly visible and should receive a high visibility score. Conversely, if a point inside the convex hull has to be moved a significant distance to join the hull, it should receive a low visibility score. A similar idea is presented by Mehra et al. [51], which relaxes the conditions of points lying on the convex hull to include nearby points. Thus, they formulate a robust visibility operator tailored for noisy point cloud data.

\textsuperscript{3}Note that the algorithm provided is a concise simplification of the actual implementation.

\textsuperscript{4}The convex hull is calculated with \texttt{scipy.spatial.ConvexHull} [77].
Figure 4.3: An image displaying what happens with two points after spherical flipping (mirroring in the circle’s boundary). The spherical flipping kernel is chosen here as it is more obvious what happens after applying it to a point set, compared to using the exponential inversion kernel.

Figure 4.4: Synthetically generated point clouds with Point-E [52] studied from a certain viewpoint (pink dot). To a large extent, we see that the HPR operator finds the visible points from the point clouds of multiple objects and that the visibility score fairly well captures the degree of visibility of the points. The points marked as "hidden" by the operator are assigned a zero co-visibility weight. Note that the point size is bigger in (b) to avoid mixed-up points on the visible and hidden surfaces when plotting. The exponential inversion kernel with $\gamma = -0.001$ was used.
Algorithm 2 Visibility Scores from Convex Hull

1: **Input:** $P_{CH}, C$ # transformed point cloud, viewpoint
2: **Output:** $c$
3: 
4: def *calculate_angle_diff*(C, $p_{CH}$-neighbors, $p_{CH}$):
5:   # Calculate two of the triangle angles using the law of cosines
6:   $a, b, c = \|p_{CH} - p_{CH}$-neighbors$\|_2, \|C - p_{CH}$-neighbors$\|_2, \|C - p_{CH}\|_2$
7:   $\beta = \frac{a^2 + c^2 - b^2}{2ac}$
8:   $\gamma = \frac{a^2 + b^2 - c^2}{2ab}$
9:   # Calc. the diff. between the angles and divide by the number of neighbors
10:  $N_{angles} = \text{len}(\gamma)$ # equals the number of neighbors
11:  $\alpha = \frac{\gamma - \beta}{N_{angles}}$
12:  return $\alpha$
13: 
14: def *angle_diff_to_score*($\alpha$):
15:   # Normalize score to $[0, 1]$
16:  $c = \alpha - \alpha_{\text{min}}$
17:  $c = \left(\frac{c}{c_{\text{max}}}\right)$
18:  return $c$
19: 
20: $\alpha = []$
21: for $p_{CH}$ in $P_{CH}$:
22:  $p_{CH}$-neighbors = *find_neighbors_vertices*($p_{CH}, P_{CH}$)
23:  $\alpha = \text{calculate_angle_diff}(C, p_{CH}$-neighbors, $p_{CH})$
24:  $\alpha$.append($\alpha$)
25:  $c = \text{angle_diff_to_score}(\alpha)$
26: return $c$
Useful Local Features for Point Cloud Alignment Classification

The local features separate entropy, joint entropy, and Sinkhorn divergence are all calculated in neighborhoods around each point of interest. The neighborhood is defined by (2.9) with the radius in (3.8) (the choice of radii is discussed more thoroughly in Section 4.4.2) and the points of interest are obtained from farthest point sampling. To speed up the calculations, the various points of interest are processed simultaneously on a GPU where the VRAM capacity affects the possibility for parallelization.

Unlike CorAl, introduced in Section 3.2, the separate and joint differential entropy are not aggregated over the entire point cloud. Instead, they are retained as features for the points of interest. As a result, there is no need to reject any points using $E_{\text{reject}}$ or to examine the overlap between the point clouds.

Regarding the Sinkhorn divergence, the weight for each point is set to one divided by the number of points in its separate neighborhood. Thus, all points in the same separate neighborhood will have the same weight, where the weights sum up to one. There is no reason to assign higher weights to denser areas as more samples are already there. Furthermore, when studying a pair of neighborhoods’ Sinkhorn divergence, one of the neighborhoods might be empty. In that case, the divergence is set to a large value. For the parameters of the Sinkhorn divergence, exploratory analysis showed that $ \epsilon(x, y) = \frac{1}{2} \|x - y\|_2^2$ as cost function and $\epsilon = 0.05$ as blur parameter gives a well-behaved metric for the efficient PyTorch implementation: geomloss [28].

Feature Weights

Each point $p_j$ will initially be assigned three weights. At a later stage (in Section 5.4), adding the weight $\Phi_{ij}$ will additionally be considered.

- $c_j \in [0, 1]$
  - **Explanation**: The co-visibility score, where zero means not co-visible while close to one means co-visible.
  - **Motivation**: Can be useful for a classifier to know the degree of co-visibility of points to assign reliability to their features. For a more thorough motivation, see Section 4.3.1.

- $w_j = \frac{|\Omega_j|}{|P_{0,1}|} \in [0, 1], \quad w_{ij} = \frac{|\Omega_{ij}|}{|P_i|} \in [0, 1]$
  - **Explanation**: The number of points in the joint/separate neighborhood relative to the joint/separate point cloud.
  - **Motivation**: It can be good for the classifier to know what ratio of the joint/separate point cloud is used in the local feature extraction when assigning reliability to a point’s feature vector.

- $\Phi_{ij} = \frac{|\Omega_{ij}|}{|P_i|} \in [0, 1]$
  - **Explanation**: The weight for the Sinkhorn divergence.
4.3 Our Point Cloud Alignment Classifier - FACT

- **Explanation**: The ratio of points in the separate and joint point cloud neighborhoods.

- **Motivation**: This gives an idea of the relative local density between the separate and the joint point cloud. This weight can help discriminate between aligned and misaligned point clouds. At least to some extent, the expected number of samples from both point clouds should in neighborhoods be quite similar for aligned point clouds and not necessarily for misaligned point clouds, as explained in Section 2.3. It is true that if the two point clouds contain different numbers of samples, this expectation becomes less reliable. Fortunately, $w_j$, $w_{ij}$, and $\Phi_{ij}$ can together represent the ratio $\frac{|P_i|}{|P_{0,1}|}$ since

$$\frac{|P_i|}{|P_{0,1}|} = \frac{|\Omega_{ij}| / w_{ij}}{|\Omega_j| / w_j} = \frac{\Phi_{ij} w_j}{w_{ij}},$$

thus compensating for this.

This results in the following weight vector $W_j = \{c_j, w_j, w_{ij}\}$.

### 4.3.2 Module 2: Point Transformer for Alignment Classification

To our knowledge, no one has investigated the usage of transformers for the alignment classification task. Thus, we propose our own problem formulation based on the classification model proposed in [88] described in Section 2.4.6. We will study the joint point cloud but remember the originating point cloud for each point with the $\pi_j$ variables introduced in Section 2.1.5. Then, each point will be assigned a set of $L$ features $F_j = \{f_0, \ldots, f_{L-1}\}$. These features include

- $h_{0,1}(\Omega_j)$ (local joint differential entropy),
- $h_i(\Omega_{ij})$ (local separate differential entropy),
- $S_{\epsilon}(\Omega_{0j}, \Omega_{1j})$ (local Sinkhorn divergence),

where all features are extracted after co-visibility filtration. Concatenating these features results in

$$F_j = \{h_{0,1}(\Omega_j), h_i(\Omega_{ij}), S_{\epsilon}(\Omega_{0j}, \Omega_{1j})\}.$$  \hspace{1cm} (4.4)

Now, we extend our feature map with

- $\pi_j$ variables,
- feature weights $W_j$,
- an $(x, y, z)$ vector.
This yields a comprehensive feature map, $G_j$, for each point:

$$G_j = \{F_j, \pi_j, W_j, p_j\} = \{h_{0,1}(\Omega_j), h_{i}(\Omega_{ij}), S_{c}(\Omega_{0j}, \Omega_{1j}), \pi_j, \epsilon_j, w_j, w_{ij}, x_j, y_j, z_j\}. \tag{4.5}$$

From this point forward, all these channels will be referred to as features, creating a total of $L = 10$ feature channels. On top of this, we will have positional embedding, explained in Section 2.4.5, to supply the transformer information about the relative 3D coordinates. Finally, this problem will be formulated as a scene classification problem with the following $m$ classes as explained in Section 4.1.1

$$(\gamma, e_d) \in \{(0, 0), (0.01, 0.1), \ldots, (m - 2) \cdot (0.01, 0.1), (m - 1) \cdot (0.01, 0.1)\}. \tag{4.6}$$

We will also perform binary classification to compare with the baseline method, CorAl, which only supports binary classification.

**Learning Parameters**

Based on the MIT licensed code implemented by You [86], our classification network already has pre-set learning parameters specifically for the ModelNet40 classification task. Therefore, a few of these parameters might not be suitable for our purposes. Yet, they serve as a good starting point. The following network parameters can be modified:

- batch size (BS),
- epochs (EP),
- learning rate (LR),
- learning rate scheduler (LRS),
- optimizer (OP),
- weight decay (WD),
- $k$-nearest neighbors in the transformer network ($k$NN),
- number of transformer-transition blocks (cyan & yellow in Figure 2.6) (NB),
- transformer dimensionality (TD),
- farthest point sampling (FPS),
- loss function (LF).

We will not delve into the details of all of these parameters. Nonetheless, here are some short comments on the learning parameters. Due to memory constraints on the GPU’s VRAM, we must modify some of the following parameters: batch size, number of neighbors, transformer dimensionality, and number of points in
farthest point sampling. Tuning of these will be presented in Section 5.2. The
epochs are chosen to ensure that the network has sufficient iterations to learn
from the training data without causing overfitting or continuing beyond a point
of no performance gain. We choose a suitable learning rate to obtain a smooth
learning process. Additionally, we incorporate a learning rate scheduler decreas-
ing the learning rate every Nth epoch, enabling finer refinements in later stages.
In the original code, the ADAM (Adaptive Moment Estimation) optimizer is used
with 0.0001 as weight decay, and we do not modify it. Also, changing the num-
ber of blocks in the network will not be investigated to maintain a focused scope
for our study. Moreover, in the original work, a cross-entropy loss function is
employed. However, using the cross-entropy alone may not be appropriate, as
penalizing minor classification errors (e.g., mixing classes 2 and 3) should differ
from penalizing major classification errors (e.g., mixing classes 0 and 3) regard-
ing loss magnitude. Thus, we will investigate a new loss formulation.

Evaluation Methodology

The learning procedure will require that we test our classifier on a different
dataset than the one on which it is trained. Monitoring the training progress to
make informed decisions will also be crucial. For instance, decisions about learn-
ing rate decay, early stopping, and finding the optimal parameter settings for the
classifier require monitoring. Therefore, we will have a validation set whose pri-
mary purpose is to help analyze whether the network overfits and to identify the
configuration that achieves the best performance on the validation data. Hence,
the learning procedure will follow Algorithm 3, where $D_i$ are the datasets and $\Theta$
are the network weights.

**Algorithm 3 Overview of Datasets Usage in the Learning Procedure**

1: **Input:** $D_{\text{train}}$, $D_{\text{val}}$, $D_{\text{test}}$, $N_{\text{epochs}}$, $\Theta_{\text{init}}$, $N_{\text{batch\_size}}$
2: **Output:** accuracy$_{\text{test}}$
3:
4: $\Theta = \Theta_{\text{init}}$
5: $\Theta_{\text{best}} = \Theta_{\text{init}}$
6: accuracy$_{\text{best}} = 0$
7: for $i$ in range($N_{\text{epochs}}$):
8: $D_{\text{train\_batches}} = \text{get\_batches}(D_{\text{train}}, N_{\text{batch\_size}})$
9: for $D_{\text{train\_batch}}$ in $D_{\text{train\_batches}}$:  
10: accuracy$_{\text{train}} = \text{classifier}(D_{\text{train\_batch}}, \Theta)$
11: $\Theta = \text{update\_params}(\Theta)$
12: accuracy$_{\text{val}} = \text{classifier}(D_{\text{val}}, \Theta)$
13: if better\_result(accuracy$_{\text{val}}$, accuracy$_{\text{best}}$):
14: $\Theta_{\text{best}} = \Theta$
15: accuracy$_{\text{best}} = \text{accuracy\_val}$
16: accuracy$_{\text{test}} = \text{classifier}(D_{\text{test}}, \Theta_{\text{best}})$
17: return $\Theta_{\text{best}}$, accuracy$_{\text{test}}$
4.4 Tuning Methodology

When tuning the classifier, its task will be to predict the correct class out of ten classes with the following offsets (for more information on $\gamma$ and $e_d$, see Section 4.1.1):

$$(\gamma, e_d) \in \{(0, 0), (0.01, 0.1), \ldots, (0.08, 0.8), (0.09, 0.9)\}.$$  (4.7)

A default network will be used to compare with the various parameter configurations. First, we go through the experiment setup and present parameters we will tune more thoroughly. Finally, we present our default parameter settings.

4.4.1 Data Usage and Hardware

Deciding on the amount of data for comparison is a trade-off between the confidence gained from extensive data and the ability to test multiple parameters using minimal data. We will balance these two extremes using 3,350 point cloud pairs divided into 2,010 train, 500 validation, and 840 test samples. These will be sampled from 335 of the 850 available scenes.\(^5\) For later experiments, we will sample from all scenes and use more data as well, but keep the data split: 60% training, 15% validation, and 25% test data. Using the default settings, this takes around six hours of feature extraction and four hours of training on an NVIDIA GeForce GTX 1080 Ti GPU with 11GB RAM.\(^6\) This is the GPU that we use for all our experiments. For some of the parameters we will test, there is no need to re-run the feature extraction. However, it will be necessary to re-run the classifier training in all cases.

4.4.2 Parameters To Tune Thoroughly

As previously stated, the default parameters were derived from small experiments not included in the report. During these experiments, we have gained a better intuition of which parameters might be interesting to tune more thoroughly; nevertheless, we motivate it here.

The Neighborhood Size

Because all features and weights are calculated in neighborhoods around points of interest, determining a good neighborhood size becomes essential. Therefore, we will try three alternative neighborhood definitions. The default neighborhood definition is as described in (3.8) based on the vertical angular resolution of the sensor and the distance from the sensor to the point. The varying radius parameter is thus $k = d \sin(5\alpha \frac{\pi}{180}) \approx 0.116d$, where $\alpha \approx 1.33^\circ$. Accordingly, the default

---

\(^5\)In the beginning of the thesis, we did not have enough storage space for all 850 scenes. This issue was later solved.

\(^6\)Naturally, parameters such as batch size and epochs significantly affect this time estimation.
varying radius is

\[
    r = \begin{cases} 
        k, & \text{if } r_{\text{min}} < k < r_{\text{max}}, \\
        r_{\text{min}}, & \text{if } k \leq r_{\text{min}}, \\
        r_{\text{max}}, & \text{if } r_{\text{max}} \leq k, 
    \end{cases} \tag{4.8}
\]

where \( r_{\text{min}} = 1 \) and \( r_{\text{max}} = 5 \), which we denote by \( r \in [1, 5] \).\(^7\) This means that when \( d \leq \frac{r_{\text{min}}}{0.116} \approx 8.6 \) meters, the radius is \( r_{\text{min}} \), and when \( d \geq \frac{r_{\text{max}}}{0.116} \approx 43.2 \) meters, the radius is \( r_{\text{max}} \). An alternative method would be to expand this range to account for a larger degree of varying densities. Hence, as our first alternative approach, let \( r_{\text{min}} = 0.5 \) meters and \( r_{\text{max}} = 7.5 \) meters. This will give us varying radii for the range \( d \in [r_{\text{min}}, r_{\text{max}}] = [0.116, 64.8] \) meters and two fixed radii (\( r_{\text{min}} \) and \( r_{\text{max}} \)) for the range \( d \in [0, 4.3] \cup [64.8, \infty] \).

When considering the joint neighborhood, choosing the radius dependent on both sensor positions is more relevant, as this will determine the joint density. Accordingly, we could let \( k \) be replaced by \( k_{\text{joint}} = \frac{d_0 + d_1}{2} \sin(5\alpha \frac{\pi}{180}) \) in (4.8). This will be our second alternative approach where we use \( r_{\text{min}} = 0.5 \) and \( r_{\text{max}} = 7.5 \).

Even though \( k_{\text{joint}} \) might work better than the original expression, it has drawbacks. Assume we have the following two cases:

1. \( d_0 = d_1 = d \),
2. \( d_0 \neq d_1 \),

under the following condition \( d = \frac{d_0 + d_1}{2} \). Then \( k_{\text{joint}} \) would be the same for both cases; however, the expected density would not be the same (a proof of a simplified case is given later). Our third approach aims to handle this; consequently, the reasoning will be more involved. We refer to Appendix A.3.1 for a derivation of our last alternative for \( k \). The result is to choose:

\[
    k_{\text{adaptive}} = \frac{\sqrt{2}d_0d_1 \sin(5\alpha \frac{\pi}{180})}{\sqrt{d_0^2 + d_1^2}} \tag{4.9}
\]

and plug it into (4.8) with default \( r_{\text{min}} = 0.5 \) and \( r_{\text{max}} = 7.5 \).

To compare \( k_{\text{joint}} \) and \( k_{\text{adaptive}} \), we have illustrated the variation in radii as a function of sensor distances in Figure 4.5. When \( d_0 \) and \( d_1 \) are similar, the radii derived from both methods converge. However, as the disparity in distances increases, so does the divergence in radii. This difference becomes especially pronounced in sequences with large expected sensor distances, such as in global registration or with low-frequency sensors. Consequently, our \( k_{\text{adaptive}} \) offers a versatile choice of radii suitable for a diverse range of applications.

**Augmentation and Normalization**

To regularize the network and enhance generalization capabilities, we aim to use augmentation and normalization as a part of the learning pipeline. Both

\(^7\)Compared to CorAl, we have increased \( r_{\text{min}}, r_{\text{max}}, \) and \( \alpha \) by a factor five to obtain more voluminous neighborhoods as argued for in Section 4.2.
modules were already implemented in the code by You [86], which is why we will test if these methods remain suitable for our application. The augmentation/normalization is performed in the following order:

1. **Point Cloud Normalization**: Center the point cloud by removing the mean in each spatial direction, followed by scaling the point cloud such that the most distant point lies on the unit ball.

2. **Point Dropout**: Draw $\alpha_j \sim U(0, 1)$ (uniform distribution). For each $p_j \in \mathcal{P}$, reassign $p_j$ to the first point in the point cloud if $\alpha_j > 0.4375$. We keep the point to maintain an equal number of points in each point cloud. However, as outlined in Section 2.4.6, the transition down block of the classification network uses farthest point sampling. Consequently, most of the dropout samples will be neglected in subsequent layers. All in all, this reduces the dependence on specific points in the point cloud.

3. **Point Cloud Scaling**: Draw $\alpha \sim U(0.8, 1.25)$ and scale the entire point cloud with $\alpha$.

4. **Point Cloud Shift**: Let $x = [x_0 \ x_1 \ x_2]^T \in \mathbb{R}^3$. Draw $x_i \sim U(-0.1, 0.1)$ for $i = \{0, 1, 2\}$. Shift the point cloud such that $\forall p_j \in \mathcal{P}$ update $p_j := p_j + x$.

Two other augmentation techniques we will try later are:
1. **Point Cloud Rotation**: Draw $\alpha \sim U[0, 2\pi]$. Use (4.2) to rotate each point in the point cloud $\alpha$ radians around the $z$-axis.

2. **Point Cloud Jitter**: Perturb each $p_j \in P$ according to

$$q_{jk} = \begin{cases} 
    b, & \text{if } s_{jk} > b, \\
    -b, & \text{if } s_{jk} < -b, \\
    s_{jk}, & \text{else,}
\end{cases} \quad (4.10)$$

where $s_{jk} \sim N(0, \sigma), k \in \{0, 1, 2\}$. As a result, we update all our points with $p_j := p_j + q_j$.

**Selecting Hidden Point Operator**

Furthermore, we aim to find a good visibility kernel with suitable parameter settings. To do that, we compare the spherical flipping kernel with the exponential inversion kernel. Parameter values we aim to test thoroughly applied on a sample in the nuScenes dataset are shown in Figure 4.6 and Figure 4.7. When choosing a suitable parameter value, there is a trade-off between an operator that detects all the hidden points and an operator that is more restrictive in each prediction but more accurate when it marks a point as hidden. For the spherical flipping kernel, two good examples are obtained for the $R$-values $8.25$ and $3.25$, while for the exponential inversion kernel, $\gamma$-values $-0.0005$ and $-0.0001$ are satisfactory.

**The Loss Function**

We also aim to explore a new loss function as mentioned in Section 4.3.2. The default network uses the cross-entropy loss we presented in Section 2.4.1. To penalize classification errors with our knowledge of the classes’ inherent order, we use the one-dimensional Wasserstein loss detailed in Section 2.4.2. Given the advantages of both loss functions, a combination may prove beneficial. The combined loss function would then be

$$L_{\text{FACT}}(y, p, \lambda) = \lambda H(y, p) + (1 - \lambda)L_{W1}(y, p), \lambda \in [0, 1]. \quad (4.11)$$

This loss formulation will be tested for different $\lambda$-values.\(^9\)

**VRAM Intensive Parameters during Classification**

Four critical parameters of the classification network that affect the GPU’s VRAM usage are the batch size ($bs$), the transformer dimensionality ($td$), the number of nearest neighbors ($kNN$), and the number of points we farthest point sample to. For example, some tensors created during inference are of shape ($bs \times td \times kNN \times \text{FPS}$). With the default settings, we obtain a tensor of shape ($20 \times 128 \times 16 \times 2048$),

---

\(^8\)The actual radius is proportional to $10^R$, where $R$ is the value presented here.

\(^9\)Earlier, the loss formulation was: $L_{\text{FACT}}(y, p, \lambda) = \lambda_0 H(y, p) + \lambda_1 L_{W1}(y, p), \lambda_0, \lambda_1 \geq 0$, resulting in a slightly different loss for some of the experiments.
Figure 4.6: Here, we color code the co-visibility of one sample from the nuScenes dataset [18]. Each point is assigned a co-visibility score according to the spherical flipping operator. On the upper row, we visualize two aligned scenarios, whereas on the bottom row, we visualize two misaligned scenarios. The R-parameter value varies from left to right, where a high R-value marks more points as visible. The misaligned cases’ offsets are $(\gamma, e_d) = (0.05, 0.5)$. 
Figure 4.7: Here, we color code the co-visibility of one sample from the nuScenes dataset [18]. Each point is assigned a co-visibility score according to the exponential inversion kernel. On the upper row, we visualize two aligned scenarios, whereas on the bottom row, we visualize two misaligned scenarios. The $\gamma$-parameter value varies from left to right, where a high $\gamma$-value marks more points as visible. The misaligned cases’ offsets are $(\gamma, e_d) = (0.05, 0.5)$. 
which has 83,886,080 elements. If using torch.float32, this matrix takes up approximately 336 MB VRAM space (roughly 3% of the available VRAM). Besides the four factors affecting memory consumption and computation speed, we give some additional aspects to consider in Table 4.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Small Value</th>
<th>Large Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>Has a regularizing effect [80]. Weight updates become noisy [33].</td>
<td>Offers accurate gradient estimations [33].</td>
</tr>
<tr>
<td>Transformer dimensionality</td>
<td>Low-dimensional embeddings lead to important connections being missed.</td>
<td>Focusing on irrelevant details results in overfitting and speed issues.</td>
</tr>
<tr>
<td>Number of nearest neighbors (kNN)</td>
<td>Lacking knowledge of the point’s neighborhood [88].</td>
<td>Include irrelevant points too far away [88].</td>
</tr>
<tr>
<td>Farthest point sampling</td>
<td>Limited input to the classifier. The point subset is well-spread and thus efficiently represents the point cloud structure.</td>
<td>It boosts information but may restate details as close points have similar neighborhoods. Slow feature extraction and training.</td>
</tr>
</tbody>
</table>

*Table 4.1: Impact of different model parameters affecting the GPU’s VRAM usage.*

### 4.4.3 Default Settings

The settings in the default network are the following:

1. BS: 20,
2. EP: 150,
3. LR: 0.00005,
4. LRS: $N = 10, \gamma = 0.85$,
5. OP:adam,
6. WD: 0.0001,
7. kNN : 16,
8. NB: 4,
9. TD: 128,
10. FPS: 2048,
11. LF: $\mathcal{L}_{\text{FACT}}(y,p,1)$,
12. AN: Norm, Drop, Scale,
13. VK: $\gamma = -0.0001$,
14. $r : [1, 5]$.

Throughout this thesis, various parameters are defined. For ease of reference:

- Parameters 1-11, designated as learning parameters, are detailed in Section 4.3.2.
- Parameter 12, concerning augmentation and normalization, is detailed in Section 4.4.2.
- Parameter 13, representing the kernel function, is detailed in Section 2.3.1.
- Parameter 14, defining possible neighborhood radii, is detailed in (3.8).

The choices behind the current parameters stem from ad-hoc testing and the previous settings in the implementation by You [86].
This chapter presents the results of our classifiers on both binary and multinomial classification tasks. First, parameter tuning is presented. Second, a few additional improvements are tested. Third, an ablation study on our proposed feature channels is performed. Finally, we test our best classifiers on both binary and multinomial classification tasks, where we benchmark against the baseline CorAl. Note that a bug explained in Section 5.1 affects all results presented Section 5.2 and Section 5.3.

5.1 Ground Truth Leakage

Unfortunately, after running all the experiments in Section 5.2 and Section 5.3, we discovered a bug in our implementation where ground truth data was, to a certain extent, used in the feature extraction. Specifically, to determine which points in the joint point cloud are in the neighborhood of the second point cloud, distances between the two point clouds were calculated. This is represented by the following expression: \( \text{dist}(P_{0,1}^0, P_{1}^i) \). Here, \( P_{1}^0 \) is, by mistake, the correctly aligned point cloud, not the intended perturbed one. This leads to some ground truth data leaking into features such as \( w_j = |\Omega_j|/|P_{0,1}| \) and \( w_{ij} = |\Omega_{ij}|/|P_i| \) which can be utilized by the classifier.

This change affects the performance of the classifier, which is shown in Table 5.1. The performance drops by eight to ten percentage points, depending on the settings and amount of data used. We believe that most of the experiments in Section 5.2 and Section 5.3 are still valid in the sense that the conclusions regarding parameter choices hold. One exception for this is the radii variation ablation highlighted in Table 5.2, as it directly concerns neighborhoods. Given that the bug influences the content of the neighborhoods, it indirectly affects many other components to varying degrees as well. The best thing here would
be to redo all experiments in Section 5.2 and Section 5.3. Re-running all those experiments would approximately take 320 hours (12 \cdot 6 = 72h (feature extraction), 62 \cdot 4 = 248h (classification)) if no breaks between runs were made and no problems appeared. In practice, perhaps three weeks would be enough to re-run the experiments. Therefore, we refrain from doing this. Similarly, we have not repeated all experiments \( N \) times to account for variations stemming from the stochastic nature of a neural network. This underlines the challenge we face when drawing conclusions about the input or the stochastic system conditioned on the output.

\[
\begin{array}{|c|c|}
\hline
\text{Configuration} & \text{Accuracy} \\
\hline
335 \text{ scenes, } 3,350 \text{ samples, } BS = 32, \text{ EP } = 150 & \\
1. \text{ Bug present} & 90.5 \\
2. \text{ Bug solved} & 80.4 \\
335 \text{ scenes, } 3,350 \text{ samples, } BS = 16, \text{ EP } = 150 & \\
3. \text{ Bug present} & 87.6 \\
4. \text{ Bug solved} & 76.8 \\
850 \text{ scenes, } 4,250 \text{ samples, } BS = 16, \text{ EP } = 150 & \\
5. \text{ Bug present} & 90.0 \\
6. \text{ Bug solved} & 82.9 \\
\hline
\end{array}
\]

\textbf{Table 5.1}: The table shows the result before and after solving a ground truth leakage bug found during the later part of the experiment phase.

### 5.2 Parameter Tuning and Component Choosing

The performance of the default network is presented in Figure 5.1, with an accuracy of 89.0%. In Figure 5.1a, we see some overfitting, suggesting that more data is needed. Because the data distribution is the same in expectation for the train, validation, and test set, drawing more samples implies an empirical distribution that is more similar over the sets. Positively, we see in Figure 5.1b that the classifier does not make any extreme misclassifications. The only misclassifications we observe are neighboring class predictions.

First, parameter tuning is performed as detailed in Section 4.4. The results from the tuning are shown in Table 5.2. We divide up the parameter tuning into six parts. One part for each subsection of Section 4.4.2, and one for three potential best settings configurations.

#### 5.2.1 Radii Variations

Increasing the radii range from \( r \in [1, 5] \) to \( r \in [0.5, 7.5] \) seems to improve accuracy moderately. Additionally, the feature extraction becomes about 21% faster
### 5.2 Parameter Tuning and Component Choosing

<table>
<thead>
<tr>
<th>Parameter Tuning</th>
<th>Accuracy</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default Configuration</strong></td>
<td>89.0</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td><strong>Radii Variations ($r$)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. $r \in [0.5, 7.5]$, $k = \frac{d_0 + d_1}{2} \sin(5\pi \frac{r}{2})$</td>
<td>89.6</td>
<td>4.615 + 0.016 = 4.631</td>
</tr>
<tr>
<td>2. $r \in [0.5, 7.5]$</td>
<td>89.9</td>
<td>4.997 + 0.015 = 5.012</td>
</tr>
<tr>
<td>3. $r \in [0.5, 7.5]$, $k = \sqrt{d_0 d_1} \sin(5\pi \frac{r}{2}) / \sqrt{\frac{d_0}{d_1}}$</td>
<td>90.1</td>
<td>4.675 + 0.015 = 4.690</td>
</tr>
<tr>
<td><strong>Augmentation and Normalization (AN)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1. Norm, Drop</td>
<td>85.2</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>2.2. Norm, Drop, Shift</td>
<td>89.4</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>2.3. Norm, Drop, Scale, Shift</td>
<td>90.2</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td><strong>Visibility Kernels (VK)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1. $\gamma = -0.0005$</td>
<td>85.2</td>
<td>3.463 + 0.015 = 3.478</td>
</tr>
<tr>
<td>3.2. $R = 3.25$</td>
<td>88.3</td>
<td>6.501 + 0.015 = 6.516</td>
</tr>
<tr>
<td>3.3. $R = 2.75$</td>
<td>90.0</td>
<td>6.537 + 0.015 = 6.552</td>
</tr>
<tr>
<td><strong>Loss Functions (LF, EP)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1. $L_{FACT}(y, p, 0)$</td>
<td>37.4</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>4.2. $5L_{FACT}(y, p, 0.2)$</td>
<td>84.9</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>4.3. $1.25L_{FACT}(y, p, 0.8)$</td>
<td>89.3</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>4.4. $2L_{FACT}(y, p, 0.5)$, $EP = 200$</td>
<td>89.6</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>4.5. $2L_{FACT}(y, p, 0.5)$</td>
<td>91.0</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td><strong>Model Scaling (BS, kNN, TD, FPS, EP)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1. $2L_{FACT}(y, p, 0.5)$, $BS = 64$, $kNN = 8$, $TD = 64$</td>
<td>84.0</td>
<td>6.322 + 0.017 = 6.339</td>
</tr>
<tr>
<td>5.2. $2L_{FACT}(y, p, 0.5)$, $BS = 64$, $TD = 64$</td>
<td>85.4</td>
<td>6.322 + 0.010 = 6.332</td>
</tr>
<tr>
<td>5.3. $2L_{FACT}(y, p, 0.5)$, $BS = 16$, $kNN = 32$</td>
<td>87.0</td>
<td>6.322 + 0.021 = 6.343</td>
</tr>
<tr>
<td>5.4. $2L_{FACT}(y, p, 0.5)$, $BS = 32$, $TD = 256$</td>
<td>87.1</td>
<td>6.322 + 0.021 = 6.342</td>
</tr>
<tr>
<td>5.5. $2L_{FACT}(y, p, 0.5)$, $BS = 16$, $TD = 256$</td>
<td>88.4</td>
<td>6.322 + 0.021 = 6.343</td>
</tr>
<tr>
<td>5.6. $2L_{FACT}(y, p, 0.5)$, $BS = 32$, $kNN = 8$, $TD = 256$</td>
<td>88.7</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>5.7. $2L_{FACT}(y, p, 0.5)$, $BS = 32$, $kNN = 8$, $TD = 256$, $EP = 225$</td>
<td>89.9</td>
<td>6.322 + 0.014 = 6.336</td>
</tr>
<tr>
<td>5.8. $2L_{FACT}(y, p, 0.5)$, $BS = 32$</td>
<td>89.9</td>
<td>6.322 + 0.013 = 6.335</td>
</tr>
<tr>
<td>5.9. $2L_{FACT}(y, p, 0.5)$, $BS = 16$, $TD = 256$, $EP = 200$</td>
<td>90.0</td>
<td>6.322 + 0.021 = 6.343</td>
</tr>
<tr>
<td>5.10. $2L_{FACT}(y, p, 0.5)$, $BS = 16$, $FPS = 4096$</td>
<td>91.8</td>
<td>10.961 + 0.040 = 11.001</td>
</tr>
<tr>
<td><strong>Best Settings ($r$, AN, VK, LF, BS, TD, FPS, EP)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.1. $r$: #3, AN: #3, VK: #3, LF: $L_{FACT}(y, p, 0.5)$, $BS = 32$</td>
<td>86.0</td>
<td>4.604 + 0.014 = 4.618</td>
</tr>
<tr>
<td>6.2. $r$: #3, AN: #3, VK: #3, LF: $L_{FACT}(y, p, 0.5)$, $BS = 16$, $TD = 256$</td>
<td>91.9</td>
<td>4.604 + 0.025 = 4.629</td>
</tr>
<tr>
<td>6.3. $r$: #3, AN: #3, VK: #3, LF: $L_{FACT}(y, p, 0.5)$, $BS = 16$, $FPS = 4096$</td>
<td>90.0</td>
<td>6.603 + 0.039 = 6.642</td>
</tr>
</tbody>
</table>

**Table 5.2**: This table showcases parameter tuning of key parameters of the FACT pipeline on the data and hardware specified in Section 4.4.1. The default parameter configuration can be found in Section 4.4.3. We adjusted a subset of these parameters for our experiments. This subset’s default configuration has the following settings: $L_{FACT}(y, p, 1)$, $r \in [1, 5]$, Norm, Drop, Scale, $\gamma = -0.0001$, $BS = 20$, $TD = 128$, $EP = 150$, $kNN = 16$, $FPS = 2048$. If no value for a specific parameter is shown, it retains the default value. Due to a late change in our loss function formulation, the loss function used in the table may have been scaled with a constant, as can be seen. For the best parameter configurations, we trained the models for 200 epochs. The displayed timings indicate the duration for feature extraction and inference of a single sample.
Figure 5.1: The figure shows a run on 3,350 point cloud pairs drawn from 335 scenes in the nuScenes dataset with ten different perturbation classes with an expected balanced class distribution. The classifier is trained on 2,010 samples, validated on 500 samples, and tested on 840 samples. On the test data, the accuracy is 89.0%.

and more stable\(^1\). The reason for the speed-up is that the maximum number of points within neighborhoods radically gets decreased as these often are observed for nearby neighborhoods whose volumes are \(r^3_{\text{min,old}} - r^3_{\text{min,new}}\) smaller. This is also the reason for the observed improved stability. Furthermore, the density aware \(k\)-parameter, \#1.3 under radii variations, outperforms the other radii alternatives. It better accounts for the expected density in the neighborhoods and together with \#1.1 give additional speed improvements.

### 5.2.2 Augmentation and Normalization

The augmentation and normalization that is used affect the data that the network trains on. We believe that using dropout is more reasonable than using normalization, shift augmentation, and scale augmentation. At least for the positional embedding, normalization of the point cloud destroys the notion of the observed distances. This can be seen as a problem as the LIDAR’s data points live in an observed 3D space. On the other hand, when studying synthetic datasets like ModelNet40 based on CAD models [82], it does make sense to normalize as there is no coordinate system with the same scale for the different data samples. The same reasoning applies to the usage of scale augmentation.

Concerning shift augmentation, this can also be seen as problematic as the entire point cloud is shifted, resulting in the expected density of the point cloud not

\(^1\)In this context, stable refers to a program with fewer crashes. Since adjusting the radii range and addressing a subsequent bug, no crashes have been observed.
decreasing quadratically with the distance from the origin. Augmenting the data so that it leaves the distribution inherent to the sensor is learning an invariance that will never be used.

Nevertheless, we do obtain superior performance with normalization and augmentation. One possible reason that the augmentation is useful is that the training data will be varied during the training process, making the network less prone to overfitting. This overfitting can stem from the \((x, y, z)\) vector, which is used both in the position encoding and as a feature. The network might learn that some positions should be highly weighted, even though that might just be true for the training data. It is important to note here that the different neighborhoods in the network do not "communicate" during training except through the relative position encoding. Thus, having the \((x, y, z)\) vector only makes sense as a reliability weight of different parts in the 3D space and not as a feature relating neighborhoods beyond the extent of the relative position encoding. As a reliability weight, it can be used to model the anticipated distance or divergence distributions across various alignment classes. That is because the spatial offset between distributions typically increases with the distance to the origin, as a rotation translates points further away from the origin more than points close to the origin.

Encoding information about 3D space, apart from the position encoding, should not require the full \((x, y, z)\) vector. It is reasonable to assume the network should pay equal attention to the \(x\) and \(y\) directions, while the \(z\) direction might have a different role. For the \(x\) and \(y\) directions, it should suffice to encode the distance from the origin instead of the \(x\) and \(y\) values. This is to weigh regions differently based on their expected density. Even though there might be some distinct information between the \(x\) and \(y\) channels, such as in the automotive settings where houses more often appear on the side of a car than in front of it, we think this is close to negligible for the classification task. However, from the \(z\) channel, the network might understand which points are more likely to belong to the ground and which are more likely to be associated with houses. As it can be more challenging to detect misalignment on the ground than on other surfaces like houses (see Figure 2.1), weighing these points differently might make sense. To some extent, this reasoning assumes that the ground is flat and without slope.

Because of all this, we will investigate further what positional information, beyond the position encoding, is helpful for the classifier. Even if the \((x, y, z)\) vector would not be used as a feature, augmentation will still be necessary for position encoding. This will be studied more later. We will also study other types of augmentation that might be more suitable for this specific task.

### 5.2.3 Visibility Kernels

The spherical flipping kernel with \(R = 2.75\) is the most accurate among the visibility kernels. On the other hand, the exponential inversion kernel with \(\gamma = -0.0005\) leads to faster processing. This is because many points are assigned a co-visibility weight of zero (as seen in Figure 4.7c), resulting in their removal from the point cloud. If all points in a neighborhood from one of the point clouds
are removed due to occlusion, the joint and separate differential entropy for that neighborhood would be the same. Typically, when the joint and separate differential entropy is the same, it indicates alignment, even though they might be severely misaligned. It is possible that keeping these points would have enhanced performance, especially for $\gamma = -0.0005$, but at the expense of slower processing. However, this potential misalignment is indirectly encoded in the features. If a neighborhood is co-visible, then it should contain points from both point clouds. This would not be the case if the points were removed, so the misalignment could still be detected and also at a faster processing speed.

5.2.4 Loss Functions

Studying the loss function, it is clear that the 1D Wasserstein loss ($L_{W1}(y,p) = L_{\text{fact}}(y,p,0)$) in itself has deficient performance, while it improves the performance of default model when combined with the cross-entropy loss. $L_{W1}$ does take the classes’ inherent order into account, making it a suitable loss for this task. To demonstrate how the loss functions $L_{W1}(y,p), H(y,p)$, and $L_{\text{fact}}(y,p,0.5)$, behave for an overconfident and an underconfident network, we refer to Figure 5.2. Figure 5.3 further visualizes what happens with the gradients with respect to the logits for these cases. When the network exhibits overconfidence, the norm of the gradients of the Wasserstein loss with respect to the logits gets very small (notice the scale $10^{-11}$ on the $y$-axis), posing severe challenges for a network to learn. Conversely, as depicted in the second column, this does not occur for the cross-entropy loss. However, a limitation with cross-entropy loss is that the gradients of the loss with respect to the logits are almost the same for all misclassifications. This is where the Wasserstein loss comes in. It can help the network steer the logits in the correct direction by understanding which misclassification should have the largest and smallest gradients, creating a loss function that models the underlying relationship between classes. Thus, when we combine the losses in our proposed $L_{\text{fact}}(y,p,0.5)$ loss, we get the best of both worlds: A loss function that understands the underlying relationship between the classes without suffering from the vanishing gradient problem.

In Table 5.2, a few combinations of the losses were tested. However, equally weighing the losses seems to give the highest performance. Thus, we can conclude that from a quantitative and a qualitative standpoint, our proposed loss function with $\lambda = 0.5$ demonstrates superior performance to the cross-entropy loss and one-dimensional Wasserstein loss.

5.2.5 Model Scaling

Additionally, we test a subset of the GPU VRAM parameters. We decided to use the best identified loss function for all experiments because there may be a non-negligible connection between the loss function and the parameters, such as the batch size. As we mainly vary four parameters where every parameter can attain different values, the set of possible combinations is enormous. Hence, we restricted the parameter search to a limited subset. As the network is stochastic,
5.2 Parameter Tuning and Component Choosing

Figure 5.2: In the left column, two sets of loss functions for two distinct prediction sets are visualized: an overconfident set (upper) and an underconfident set (lower). Studying the right column, it is evident that the example of a predicted distribution for the top row exudes high confidence, while the distribution in the bottom row demonstrates significantly less confidence.

Figure 5.3: Here, the average gradients with respect to the logits for the three loss functions in Figure 5.2 are shown. The top and bottom rows in this figure correspond to the top and bottom rows in Figure 5.2, respectively.
we cannot entirely rely on the results. It is also difficult to draw definitive con-
clusions about a specific parameter as the different parameters affect each other.
We can conclude that a high batch size does not necessarily perform better, ex-
cept in terms of inference speed. Having 16 nearest neighbors seems like a good
balance between fully grasping the neighborhood and without including too dis-
 tant neighbors. A transformer dimensionality of 128 or 256 gives the highest
accuracy. Finally, we see that using more points from the farthest point sampling
further improves the results, but at the cost of significantly longer feature extrac-
tion time.

5.2.6 Best Settings
The best parameters should be tested together to see how well they interact. We
test three different settings and find that configuration #6.2 performs the best.
Therefore, we will, for the most part, continue to focus on this variant. However,
increasing the batch size speeds up training considerably, which might be use-
ful for later experiments. We also believe having more points from the farthest
sampling should benefit the classification. The relatively low accuracy obtained
for this particular run might be explained by the network’s stochasticity, even if
the size of the transformer might have a meaningful impact. One reason to be-
lieve the randomness induced by the network has a considerable impact is the
surprisingly low performance of the first best setting (#6.1).

5.3 Additional Improvements
In this section, we test some additional changes to get some potential improve-
ments.

5.3.1 Spatial Information
The transformer architecture we use for scene classification of point clouds incor-
porates spatial information through relative position encoding. The network uses
this information to find the $k$-nearest neighbors, which define the neighborhood
in the transformer. Besides this, it can be useful to provide additional spatial
information to the network to allow for weighing point cloud regions differently.
This is more thoroughly discussed in Section 2.4.5, Section 2.4.6, and Section 5.2.

We conducted an exploratory study in Table 5.3 to determine what additional
spatial information might be useful. There, we compared the usage of the $(x, y, z)$
vector, the distance to the origin $(\|p\|_2)$, the height channel $(z)$, and the usage of
augmentation. We see from the first section of the table that using either the
$(x, y, z)$ channels, $\|p\|_2$, or $z$, together with augmentation, gives the best perfor-
ance. It is also clear that augmentation is most important for the $(x, y, z)$ vec-
tor, which was expected as three channels more easily cause spatial overfitting
than one channel. Considering all three sub-experiments, it hard to determine
between using the $(x, y, z)$ channels, $\|p\|_2$, or $z$. Nevertheless, using one spatial
channel seems to give a network less prone to spatial overfitting at the same time
as its performance is on par with the \((x, y, z)\) vector. Following Occam’s razor \([15]\), we choose to continue with one spatial channel. Between \(\|p\|_2\) and \(z\), we choose \(\|p\|_2\) since it has a slightly higher average accuracy and because it better can help anticipate the distance and divergence between the points sets for the various alignment classes.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Accuracy ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>335 scenes, 1,675 samples, BS = 32, EP = 100</td>
<td></td>
</tr>
<tr>
<td>1.1.</td>
<td>75.4</td>
</tr>
<tr>
<td>1.2. aug</td>
<td>81.4</td>
</tr>
<tr>
<td>1.3. (z)</td>
<td>76.6</td>
</tr>
<tr>
<td>1.4. (z), aug</td>
<td>86.4</td>
</tr>
<tr>
<td>1.5. (|p|_2)</td>
<td>71.1</td>
</tr>
<tr>
<td>1.6. (|p|_2), aug</td>
<td>86.4</td>
</tr>
<tr>
<td>1.7. (|p|_2), (z)</td>
<td>74.9</td>
</tr>
<tr>
<td>1.8. (|p|_2), (z), aug</td>
<td>82.8</td>
</tr>
<tr>
<td>1.9. ((x, y, z))</td>
<td>60.4</td>
</tr>
<tr>
<td>1.10. ((x, y, z)), aug</td>
<td>82.3</td>
</tr>
<tr>
<td>335 scenes, 3,350 samples, BS = 16, EP = 200</td>
<td></td>
</tr>
<tr>
<td>2.1. (z), aug</td>
<td>88.8, 88.8</td>
</tr>
<tr>
<td>2.2. (|p|_2), aug</td>
<td>89.0, 91.7</td>
</tr>
<tr>
<td>2.3. ((x, y, z)), aug</td>
<td>91.9, 90.6</td>
</tr>
<tr>
<td>335 scenes, 3,350 samples, BS = 32, EP = 200</td>
<td></td>
</tr>
<tr>
<td>3.1. (z), aug</td>
<td>89.4</td>
</tr>
<tr>
<td>3.2. (|p|_2), aug</td>
<td>88.2</td>
</tr>
<tr>
<td>3.3. ((x, y, z)), aug</td>
<td>89.8</td>
</tr>
</tbody>
</table>

*Table 5.3:* This table displays the accuracy for different feature combinations of the parameters \(\|p\|_2\), \(z\), aug, and \((x, y, z)\). These parameters represent the norm of the points, the \(z\) coordinate of the points, the usage of augmentation (which includes normalization, shift, and scale, with dropout implemented in all cases.), and the \((x, y, z)\) vector. For each row, only the parameters set to true are given (e.g., for row three, only the \(z\) channel is used). The run time for each setting is quite similar and has, therefore, not been included in the table. The settings used here are the best observed settings from Table 5.2, where we varied the batch size, number of epochs, and sample size. Because \#1.10 in section two of this table was already evaluated in Table 5.2, the value here is simply a restatement. To obtain better estimates of the fluctuations between runs, we repeated the experiments in section two.

### 5.3.2 Normalization for Features and Positional Embedding

Data normalization has been shown to increase the performance of machine learning algorithms frequently \([39, 71]\). We study feature and positional embedding
normalization using the min-max normalization zero (MMN₀) method such that

\[ x'_{lj} = \frac{x_{lj} - \min_{j \in J}(x_{lj})}{\max_{j \in J}(x_{lj}) - \min_{j \in J}(x_{lj})}, \quad \forall l \in 0, \ldots, L - 1, \]  

(5.1)

where \( L \) is the number of features and \( J \) is the set of points in the farthest point sampled joint point cloud [71]. This maps every feature to the \([0, 1]\) range. This was done for single samples (individual joint point clouds) and batches of samples. For the batch method, we also set the min values of the differential entropies to \( \frac{1}{2} \ln(e)^2 \) for every batch and the max value of the Sinkhorn divergence to 1,000 as this was the distance we used when there was no valid Sinkhorn divergence between the two separate neighborhoods (this happened when one of the neighborhoods was empty\(^3\)). The result from the normalization study is shown in Table 5.4. While feature normalization deteriorates performance, normalization of positional embedding yields the best results when applied to single samples.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Accuracy ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>335 scenes, 3,350 samples, BS = 32, EP = 150 (Single Samples)</td>
<td></td>
</tr>
<tr>
<td>1. Non-Normalized Features, Non-Normalized Positional Embedding</td>
<td>87.7</td>
</tr>
<tr>
<td>2. Non-Normalized Features, Normalized Positional Embedding</td>
<td>90.5</td>
</tr>
<tr>
<td>3. Normalized Features, Non-Normalized Positional Embedding</td>
<td>56.5</td>
</tr>
<tr>
<td>4. Normalized Features, Normalized Positional Embedding</td>
<td>71.5</td>
</tr>
<tr>
<td>335 scenes, 3,350 samples, BS = 32, EP = 150 (Batch-wise)</td>
<td></td>
</tr>
<tr>
<td>5. Non-Normalized Features, Normalized Positional Embedding</td>
<td>87.0</td>
</tr>
<tr>
<td>6. Normalized Features, Non-Normalized Positional Embedding</td>
<td>34.0</td>
</tr>
<tr>
<td>7. Normalized Features, Normalized Positional Embedding</td>
<td>31.4</td>
</tr>
</tbody>
</table>

Table 5.4: Using the best settings from Table 5.2 with batch size 32 and 150 epochs, this table varies the normalization of features and positional embedding over single samples and batches.

### 5.3.3 New Augmentation Techniques

As mentioned in Section 5.2, the scale and shift augmentation disrupt the representation of actual observed distances. The scale augmentation should be extra problematic as shrinking or expanding the point cloud causes the alignment error to decrease or increase. This is not good because the augmentation is done after the feature extraction. However, the ratio of the distances to the origin between any two points remains unchanged, implying that \( \|p\|_2 \) still can weigh the relative reliability of the points similarly. Moreover, the relative distance between

\(^2\)This is the lowest possible value for the differential entropy, see (3.7).

\(^3\)Note that only one of the neighborhoods could be empty at most. The other neighborhood at least contains the point of interest.
points will also be scaled, leading to a different input to the MLP used in the relative positional embedding. When the input is scaled, the weights of the linear mapping have to be scaled inversely to obtain the same output for the same initial (before scaling) input. Thus, the gradient of the loss function with respect to linear mapping’s weights becomes different for the same initial input. This can be beneficial if a suboptimal local minimum is reached. Yet, it can also lead to difficulties in converging to optimal weights.

Incorporating augmentation techniques that do not destroy the notion of actual observed distances could be beneficial. At the same time, it would be advantageous if all other features remained invariant to the introduced augmentation. For example, differential entropy is not scale invariant, so scale augmentation after feature extraction should not be good. Moreover, a technique that meets these criteria is rotation around the z-axis (see Augmentation and Normalization under Section 4.4.2 for a description of all augmentation and normalization techniques discussed). This ensures all distances to the origin are preserved before and after the transformation, and the distances between all points remain the same. If the augmentation were to be applied before feature extraction, the poses would also need to be rotated. Otherwise, problems would arise; for instance, the distances between the points and the pose not located at the origin will differ, resulting in new radii for the neighborhoods. We propose a rotational augmentation where the point cloud is randomly rotated $\alpha \in [0, 2\pi]$ radians around the z-axis.

Another augmentation technique examined is jittering. A small jittering of points does not modify the misalignment class but can mitigate overfitting to specific positions. We have experimented with two different jittering settings: $(\sigma, b) \in \{(0.02, 0.05), (0.06, 0.15)\}$. This entails a max deviation for each component of the $(x, y, z)$ vector of 5 and 15 centimeters, with approximately an average absolute deviation of 1.6 and 4.8 centimeters, respectively.

The results of our two proposed augmentation methods are presented in Table 5.5. Unfortunately, none of them improved the performance. One reason could be that more augmentation might necessitate extended training. When increasing the number of epochs from 150 to 300, a slight increase in accuracy was obtained. These results indicate that jittering is not particularly effective. A potential reason might be that the network does not overfit to centimeter-level adjustments but instead weighs larger regions of the point cloud differently. Another possible reason why jittering was not successful is its interference with dropout. When jittering is applied after dropout, points that have been set to the first point will be affected by the jittering. Consequently, these points might not be removed during farthest point sampling. This change in the point cloud geometry can cause unnecessary misclassifications. Applying jittering before dropout thus seems like a better strategy.

Furthermore, it remains unclear why the rotational augmentation is not beneficial for the network. While we had expected that preserving distances within the data through methods like rotational augmentation would be beneficial, our results do not support this assumption. The drop in performance after applying the rotational transformation suggests that certain inherent features in the spatial data might be more important for FACT than initially thought. Thus, we will
not use jittering or rotational augmentation going forward.

<table>
<thead>
<tr>
<th>New Augmentation Techniques</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>335 scenes, 3,350 samples, BS = 32, EP = 150</strong></td>
<td></td>
</tr>
<tr>
<td>1.1. Best Normalization, Dropout, Shift, Scale</td>
<td>90.5</td>
</tr>
<tr>
<td>1.2. Best Normalization, Dropout, Shift, Scale, Jittering ± 5cm</td>
<td>88.5</td>
</tr>
<tr>
<td>1.3. Best Normalization, Dropout, Shift, Scale, Jittering ± 15cm</td>
<td>65.4</td>
</tr>
<tr>
<td>1.4. Best Normalization, Dropout, Shift, Scale, Rotation</td>
<td>83.2</td>
</tr>
</tbody>
</table>

| **335 scenes, 3,350 samples, BS = 32, EP = 300**                  |          |
| 2.1. Best Normalization, Dropout, Shift, Scale, Jittering ± 5cm   | 89.9     |
| 2.2. Best Normalization, Dropout, Shift, Scale, Jittering ± 15cm  | 66.7     |
| 2.3. Best Normalization, Dropout, Shift, Scale, Rotation         | 85.1     |

Table 5.5: The table gives the performance of FACT using two new augmentation techniques: jittering and rotation around the z-axis.

5.4 Feature Channel Ablation

To deduce which features are important, we perform a feature channel ablation study. We remove one feature at a time and then observe the performance change. This is the first time we test \( \Phi_{ij} \) as a feature. To be clear, \( \Phi_{ij} \) has not been included in any of the above tests. For additional information on \( \Phi_{ij} \), see Section 4.3.1. The results from the feature channel ablation study are presented in Table 5.6. It is relatively clear from both a theoretical and empirical perspective that features 1, 2, 3, and 4 are needed. Since \( \Phi_{ij} \) significantly decreases the accuracy, we remove it. The reason that \( \Phi_{ij} \) decreases performance could be that it correlates with other features and thus just takes up unworthy attention from the classifier. However, the magnitude of the performance change is challenging to justify. The correlation with other features is because

\[
\Phi_{ij} = \frac{|\Omega_{ij}|}{|\Omega_j|} = \frac{|P_i|w_{ij}}{|P_{0,1}|w_j} = a \frac{w_{ij}}{w_j},
\]

where \( a \) is constant for each point cloud pair. Either this information is not so important, as the ratio most often should be close to 0.5, or it can be learned implicitly through other features. Nevertheless, it is logical that \( \Phi_{ij} \) has limited contribution, given its substantial information overlap with \( w_j \) and \( w_{ij} \).

Now, we continue with the ablation study of features 5, 6, 7, and 9. It is clear from Table 5.6 that removing any additional feature decreases the performance by at least 2.5 percentage points (84.0 → 81.5). Thus, we conclude that only \( \Phi_{ij} \) should be removed. Moreover, if a speed-up in the feature extraction is desired, the \( c_j \) channel can be removed as it is expensive to compute.
5.5 Multinomial Classification

Here, we will present the performance of two classifiers using a total of 17,000 point cloud pairs. The first classifier is called FACToptimal which is relatively fast but primarily an accurate classifier. For this classifier, the following settings were used: \{\text{LR} = 0.0001\) (the previous LR doubled), \text{LRS} : (0.825, 10), \text{BS} = 16, \text{FPS} = 2048, \text{TD} = 256, \text{EP} = 250\}, and the concluded best settings from before. These settings are \#6.2 in Table 5.2, with \#1.8 in Table 5.6, and \|\text{p}\|_2 as the spatial feature channel instead of the \((x, y, z)\) vector. The second classifier we present is FACTfast, which has the same settings as the optimal classifier apart from the following: \{\text{LRS} : (0.800, 10), \text{BS} = 32, \text{FPS} = 1024, \text{T}_{\text{close}} = 2.5, \text{EP} = 200, \text{remove} c_j\}.

Studying Figure 5.4, we observe the confusion matrices for both FACToptimal and FACTfast. For FACToptimal, there is only one misclassification that is not one of the neighboring classes, while this happens two times for FACTfast. As the test set is of size 4,240 samples, both classifiers seem very trustworthy.

The performance of the classifiers is given in Table 5.7. As expected, the optimal classifier achieves a high accuracy whereas the fast classifier is executed on 33\% of the time of FACToptimal.

### Ablation Study of Feature Channels

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. All features</td>
<td>81.3</td>
</tr>
<tr>
<td>1. Label: (\pi_j)</td>
<td>79.5</td>
</tr>
<tr>
<td>1.2. Joint differential entropy: (h_{0,1}(\Omega_j))</td>
<td>79.8</td>
</tr>
<tr>
<td>1.3. Separate differential entropy: (h_i(\Omega_{ij}))</td>
<td>80.1</td>
</tr>
<tr>
<td>1.4. Sinkhorn divergence: (S_{\varepsilon}(\Omega_{0j}, \Omega_{1j}))</td>
<td>79.9</td>
</tr>
<tr>
<td>1.5. Co-visibility weight: (c_j)</td>
<td>81.8</td>
</tr>
<tr>
<td>1.6. Joint neighborhood weight: (w_j)</td>
<td>82.6</td>
</tr>
<tr>
<td>1.7. Separate neighborhood weight: (w_{ij})</td>
<td>81.2</td>
</tr>
<tr>
<td>1.8. Separate-joint neighborhood weight: (\Phi_{ij})</td>
<td>84.0</td>
</tr>
<tr>
<td>1.9. Distance to origin: (|\text{p}|_2)</td>
<td>81.6</td>
</tr>
<tr>
<td>2.5. (\Phi_{ij}, c_j)</td>
<td>81.2</td>
</tr>
<tr>
<td>2.6. (\Phi_{ij}, w_j)</td>
<td>81.5</td>
</tr>
<tr>
<td>2.7. (\Phi_{ij}, w_{ij})</td>
<td>80.7</td>
</tr>
<tr>
<td>2.9. (\Phi_{ij}, |\text{p}|_2)</td>
<td>81.0</td>
</tr>
</tbody>
</table>

**Table 5.6:** This table contains the performance for a feature channel ablation study. Each row (excluding the first) details the removed feature or features, while all other features are included in that particular run.
5.5.1 Qualitative Analysis

To assess the classifier’s (FACT$\text{optimal}$) performance and to understand occasional mispredictions, we examine the underlying point clouds from which features are extracted. For each perturbation class, two examples are shown: one correct and one incorrect classifier prediction. This gives a total of 20 images, which are displayed\(^4\) in Appendix B.1.

Although we have detailed the offset classes mathematically, studying the figures in Appendix B.1 gives a better understanding of how accurate FACT$\text{optimal}$ is. From the figures, it is harder to distinguish between classes 8 and 9 than classes 0 and 1. One conclusion from this could be that the classifier should be penalized less when mixing up classes 8 and 9 than classes 0 and 1. This is presumably also the type of classifier that is desired in applications – a classifier that accurately discerns between small errors but is less precise when distinguishing between larger errors. However, in Figure 5.4a, it is evident that FACT$\text{optimal}$ is more accu-

\(^4\)The point clouds depicted are the raw point clouds after all points within a radius of 1.5 meters are removed.
rate for the bottom classes 0-4 (94.8% accuracy) than classes 5-9 (90.0% accuracy), so an extra penalization might not be necessary. If an extra penalization were to be added, the used evaluation metric should be adjusted accordingly.

While analyzing 20 examples does not give a comprehensive understanding of occasional incorrect predictions, it can yield insightful observations. Our primary theories for why the classifier occasionally fails are:

1. Insufficient sample size
2. Presence of dynamic objects
3. Erroneous ground truth data
4. Point clouds being partially invariant to certain transformations
5. Intermingled points

where one interesting sub-topic is how specific scenes influence the classifier’s accuracy. Now, we will delve into these theories.

As a LIDAR has a limited range, it will not necessarily acquire $64 \cdot 1080$ samples for each scan. This leads to point clouds having varying numbers of samples. Naturally, if no samples are measured for a scan, estimating alignment is impossible. Similarly, if a point cloud consists of few samples, the classifier will have less evidence to support its prediction. Observations in Appendix B.1 support this, where many of the mispredictions seem to be for point clouds with fewer samples than for the correct classifications.

Observing the right image in Figure B.1, the ground truth class is zero, which means that the point clouds should be aligned. Yet, many of the lines in the image suggest otherwise. Three reasons for this discrepancy are the presence of dynamic objects, erroneous ground truth data, or noisy measurements. The idea of noise being the reason for the systematic offset can be ruled out as LIDAR noise typically is random rather than systematic, as seen in the image. While dynamic objects can introduce systematic offsets, the observed widespread offset makes it less likely. Consequently, erroneous ground truth data is the most probable reason for the offset. If erroneous ground truth data exists for many point cloud pairs, it will seem like the classifier is worse than it actually is.

To illustrate point 4, consider a simple example where a LIDAR is placed on the floor in the middle of a completely cylinder-shaped building. Then, rotating the sensor around its z-axis will result in almost identical measurements as before. Thus, estimating the alignment error becomes infeasible. While this specific scenario might be rare in real-world applications, similar situations can happen. In the upper right image of Figure B.2, some lines on the left side are perpendicular to the origin. When a rotation offset around the z-axis is applied, the lines will largely still be aligned, potentially leading to an incorrect prediction.

When there is a relatively large perturbation offset, as seen in classes 5-9, and the point clouds are grouped into clusters with significant dispersion, there is considerable overlap both before and after perturbation. Conversely, when points are restricted to 2D surfaces or uniformly spread out in space, the overlap is smaller.
In the former scenario, determining the alignment quality becomes challenging because the intermingling of points can make it difficult to distinguish between two relatively large perturbation offsets. This issue seems to appear in some of the images in Appendix B.1.

Finally, there will be scenes that are more challenging to determine alignment quality for compared to other scenes. It is reasonable to assume that the expected success rate of a sample is dependent on which scene it is sampled from. To analyze this further, we conduct hypothesis testing in Appendix A.4 over the test set to conclude with confidence 100% that the expected success rate of a sample is dependent on which scene it is sampled from.

### 5.6 Binary Classification

Figure 5.5 demonstrates CorAl’s performance on two binary classification tasks. Evidently, the classifier’s performance increases as the perturbation offset is larger. Based on this figure, we can state that the joint and separate differential entropy is not descriptive enough to discriminate between aligned and misaligned point clouds with high accuracy.

To solve the binary classification task, we propose the following three classifiers:

- **FACT\textsubscript{binary}:** trained and tested on 4,250 samples with only the two binary classes in the training set.
5.6 Binary Classification

- **FACT\textsubscript{multinomial}**: trained and tested on 4,250 samples with all the ten classes in the training set.

- **FACT\textsubscript{optimal}**: trained and tested on 17,000 samples with all the ten classes in the training set.

Other than this, the same settings were used for all of the three classifiers (see Section 5.5 for these settings). Figure 5.6 compares the confusion matrices for the baseline classifier, CorAl, with three proposed classifiers. **FACT\textsubscript{multinomial}** is the best classifier from Table 5.6 and **FACT\textsubscript{optimal}** is presented in Section 5.5. By a large margin, **FACT\textsubscript{binary}** outperforms the baseline for both classification tasks. Regarding the multinomial classifiers\footnote{Both **FACT\textsubscript{multinomial}** and **FACT\textsubscript{optimal}** are multinomial classifiers.}, the analysis is more involved. First of all, multinomial classifiers are able to handle more classes than binary classifiers, making them more general. Secondly, **FACT\textsubscript{multinomial}** is trained on equally many samples as the binary classifiers and thus has seen fewer samples of the two tested classes. Thirdly, by mapping the ten predictions of the multinomial classifiers to the two test classes, performance increases. We denote the multinomial classifiers with the mapping from ten to two classes by **FACT**. The mapping rule can be defined differently, but we simply map the prediction to the closest of the two binary classes. If the distances to the two classes are equal, we distribute the prediction equally among the binary classes. Fourth and finally, the multinomial classifiers enable us to easily incorporate confidence in the predictions. For example, as the mapping distance becomes larger, the confidence decreases. As an example, for the task \((\gamma, e_d) = (0.03, 0.3)\) and class 3 as the true label, if the prediction is class 2 in the multinomial case, then the prediction is mapped to class 1 in the binary case. As the prediction in the multinomial case has to be mapped to one class above (class 2 \(\rightarrow\) class 3), a decreased confidence, compared to the true prediction of class 3, can be assigned.

In Table 5.8, we observe that **FACT\textsubscript{binary}** is the most accurate classifier closely followed by **FACT\textsubscript{multinomial}** for classifiers trained on 4,250 samples. However, **FACT\textsubscript{optimal}**, which is trained with 17,000 samples, shows exceptional performance and outperforms all other classifiers. It even has 100.0% accuracy for the easier task. Because of all the beneficial properties a multinomial classifier entails, we believe that **FACT\textsubscript{multinomial}** or **FACT\textsubscript{optimal}** are the most suitable classifiers to use for binary classification tasks. To further describe the confidence of the predictions, more classes between the two binary classes can be added. We conclude that all our proposed classifiers achieve high performance and outperform CorAl when used correctly.
Results and Analysis

The figure shows the confusion matrices for four classifiers on two tasks: $(\gamma, e_d) = (0.01, 0.1)$ (left) and $(\gamma, e_d) = (0.03, 0.3)$ (right). The multinomial classifiers are trained on ten classes but tested on two of them.
### Binary Classification Comparison

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>850 scenes, 4,250/17,000 samples, ((\gamma, e_d) = (0.01, 0.1))</td>
<td></td>
</tr>
<tr>
<td>1.1. CorAl</td>
<td>75.3</td>
</tr>
<tr>
<td>1.2. (\text{FACT}_{\text{binary}})</td>
<td>96.3</td>
</tr>
<tr>
<td>1.3. (\text{FACT}_{\text{multinomial}})</td>
<td>90.8</td>
</tr>
<tr>
<td>1.4. (\text{FACT}_{\text{multinomial}})</td>
<td>94.2</td>
</tr>
<tr>
<td>1.5. (\text{FACT}_{\text{optimal}})</td>
<td>96.3</td>
</tr>
<tr>
<td>1.6. (\text{FACT}_{\text{optimal}})</td>
<td>97.4</td>
</tr>
<tr>
<td>850 scenes, 4,250/17,000 samples, ((\gamma, e_d) = (0.03, 0.3))</td>
<td></td>
</tr>
<tr>
<td>2.1. CorAl</td>
<td>95.6</td>
</tr>
<tr>
<td>2.2. (\text{FACT}_{\text{binary}})</td>
<td>99.6</td>
</tr>
<tr>
<td>2.3. (\text{FACT}_{\text{multinomial}})</td>
<td>87.0</td>
</tr>
<tr>
<td>2.4. (\text{FACT}_{\text{multinomial}})</td>
<td>99.6</td>
</tr>
<tr>
<td>2.5. (\text{FACT}_{\text{optimal}})</td>
<td>94.7</td>
</tr>
<tr>
<td>2.6. (\text{FACT}_{\text{optimal}})</td>
<td>100.0</td>
</tr>
</tbody>
</table>

**Table 5.8:** This table presents the performance for the six classifiers: CorAl, \(\text{FACT}_{\text{binary}}\), \(\text{FACT}_{\text{multinomial}}\), \(\text{FACT}_{\text{multinomial}}\), \(\text{FACT}_{\text{optimal}}\), and \(\text{FACT}_{\text{optimal}}\) for two binary classification tasks. The four first classifiers have been trained with 4,250 samples, while the two latter have been trained with 17,000 samples.

Given the simple classification boundary of CorAl, adding more training data is unlikely to yield a significant performance improvement. Besides the changing classification boundary, additional features seem to be necessary to separate the classes, as some points of varying classes occupy the same position in the current two-dimensional feature space. The fact that CorAl does not require a lot of data can be seen as positive. In contrast, \(\text{FACT}\) becomes better and better when increasing the size of the training set, as \(\text{FACT}_{\text{optimal}}\) clearly outperforms \(\text{FACT}_{\text{multinomial}}\).

A problem with CorAl is that if we have an estimated transformation that is so inaccurate that the point clouds’ neighborhoods do not overlap at all, then both the joint and separate differential entropy values will be the same. This would incorrectly suggest that the point clouds are aligned when they are, in fact, significantly misaligned. As mentioned in Section 3.2, the authors have set a threshold in their method to detect whether the point clouds overlap sufficiently. If the point clouds overlap less than 10%, the point cloud pair is classified as misaligned. Although this might be an extreme case, there is an issue where transformed points leave the neighborhood of a point of interest, thus not affecting its differential entropy score. This is a drawback that is challenging to address with this approach. One potential workaround could be enlarging the neighborhoods’ radii. However, this strategy would exponentially amplify computational demands, making it impractical in extreme situations.

Another issue with the method is its use of all points in the feature calculation. In reality, using fewer points would nearly suffice, as points close to each other almost have identical neighborhoods. This inefficiency can easily be addressed us-
ing farthest point sampling to choose which points to calculate features for. Furthermore, relying on logistic regression entails limitations. Logistic regression only gives decision boundaries that are affine (linear plus an intercept), which may not be sufficient to separate classes.
We start this chapter by answering the research questions. Then, we wrap up this thesis by presenting eleven ideas for future work within point cloud alignment classification. We restate the research questions:

1. How well can hand-crafted features along with DNNs distinguish between aligned and misaligned point cloud pairs?

2. Is it possible to accurately measure the absolute alignment error between two point clouds?

3. Which modules are important in a point cloud alignment classification pipeline?

As our method, based on hand-crafted features and DNNs, shows superior performance on two binary classification tasks compared with the baseline CorAl and even obtains 100.0% accuracy on one of the tasks, we can conclude that hand-crafted features along DNNs effectively can be used to distinguish between aligned and misaligned point cloud pairs.

Beyond binary classification, our method distinguishes between ten classes of misalignment with 92.4% accuracy. Our best classifier, FACT\text{optimal}, only once in 4,240 samples makes a misprediction outside the neighboring classes, which further demonstrates its competitiveness. Therefore, we find it is possible to accurately measure the absolute alignment error between two point clouds.

Lastly, we have argued why co-visibility estimation, dynamic region estimation, and density adaptation constitute key modules of a point cloud alignment classification pipeline. We demonstrated that features from optimal transport (Sinkhorn divergence) and information theory (differential entropy) satisfactorily describe relationships within a point cloud pair, playing a crucial role in measuring alignment. Furthermore, by representing the reliability of neighborhoods,
the feature weights have been helpful in measuring alignment quality. We have also shown that a classifier capable of finding complicated patterns in feature data is able to solve the PCAC task.

### 6.1 Future Work

1. **Learned descriptors**: Instead of using hand-crafted features, a DNN could be employed to learn descriptors from the data.

2. **Improved co-visibility**: For co-visibility estimation, use [51], to better handle noisy point clouds, or [13], to deal with varying point cloud densities.

3. **Incorporate dynamic region estimation**: Find the dynamic regions by solving the scene flow problem, for example, using the method by [87].

4. **Perform density adaptation**: Adapt the density of one of the point clouds to better resemble the density in the other point cloud, potentially creating more reliable OT and differential entropy measurements.

5. **Generate misaligned data differently**: Generate misaligned point cloud pairs using registration techniques as described in Section 4.1.1.

6. **The impact in the registration field**: Investigate to what extent point cloud alignment classification can be used to improve the performance of registration algorithms.

7. **Perform regression**: Convert this classification task to a regression task.

8. **Use $k$NN to define neighborhoods**: This can drastically speed up feature extraction as no padding needs to be done. It is also easier to measure distances between distributions if no empty neighborhoods appear. However, $k$NN can be harmful to use when the distance between the sensors is large, resulting in two very different point cloud distributions. With $k$NN, a large distance between two distributions could be a result of both misalignment and varying densities between the point clouds.

9. **Keep hidden points**: Try not removing points after co-visibility estimation but instead, assign the co-visibility weight zero.

10. **Handle degenerate cases better for differential entropy**: In (3.6), replace $(2\pi\epsilon)^D \det(\Sigma(\Omega_{ij})) + \epsilon$ with $(2\pi\epsilon)^D \det(\Sigma(\Omega_{ij}) + \epsilon \mathbf{I}_3)$. This can distinguish between different degenerate cases when the determinant is zero.

11. **New evaluation metric**: Use the following distance as an evaluation metric to measure the degree of the misclassification errors

\[
\frac{\sum_{y \in D_l} |y_l - y|}{|D_l|},
\]

where $y_l$ is the target label and $D_l$ is the set of predictions for that label.
Appendix
Proofs and Derivations

This chapter presents various proofs for some of the mathematical claims in this thesis.

A.1 Transformation Functions in Point Cloud Visibility

A.1.1 Spherical Flipping

Proof: The first condition holds:

\[
\frac{\partial f(\|p_j\|)}{\partial \|p_j\|} = 1 - 2 = -1 < 0. \tag{A.1}
\]

We check the second condition as well

\[
f(\|p_j\|) = \|p_j\| + 2(R - \|p_j\|) = 2R - \|p_j\| > 0, \tag{A.2}
\]

which holds if \(2R > \max_{p_j \in P} \|p_j\|\). Now, we check the third property. With \(\gamma = 2R\), the kernel can be written as

\[
f(\|p_j\|) = \gamma - \|p_j\|. \tag{A.3}
\]

Then, for \(\|p_j\| > \|p_k\|\), we have that

\[
\frac{f(\|p_j\|)}{f(\|p_k\|)} = \frac{\gamma - \|p_j\|}{\gamma - \|p_k\|} < \frac{\gamma - \|p_k\|}{\gamma - \|p_k\|} = 1. \tag{A.4}
\]
Similarly, for $0 < \epsilon < 1$, we have that
\[
\frac{f(||p_j||)}{f(||p_k||)} = \frac{\gamma - ||p_j||}{\gamma - ||p_k||} = \frac{\gamma - ||p_k||}{\gamma - ||p_k||} = 1 - \frac{||p_j|| - ||p_k||}{\gamma - ||p_k||} \tag{A.5}
\]
where
\[
1 - \frac{||p_j|| - ||p_k||}{\gamma - ||p_k||} > 1 - \epsilon \iff \frac{||p_j|| - ||p_k||}{\gamma - ||p_k||} < \epsilon \iff \begin{cases} 
\gamma > \max_{p_i \in P} ||p_i|| \text{ from Equation (A.2)} \end{cases} \iff ||p_j|| - ||p_k|| < \epsilon (\gamma - ||p_k||) \tag{A.6}
\]
which gives that
\[
\gamma \in \left[ ||p_j|| - ||p_k|| + ||p_k||, \infty \right] = \left[ ||p_j||, \infty \right]. \tag{A.7}
\]
Hence, we can conclude that for any pair $||p_j||, ||p_k|| \in \mathbb{R}^+$ s.t. $||p_j|| > ||p_k||$, and for any $0 < \epsilon < 1$, there does exist a $\gamma$ s.t.
\[
1 - \epsilon < \frac{f(||p_j||)}{f(||p_k||)} < 1.
\]
In all cases, $\gamma > \max_{p_j \in P} ||p_j||$ must hold. \qed

### A.1.2 Exponential Inversion

**Proof:** Firstly,
\[
\frac{\partial f(||p_j||)}{\partial ||p_j||} = \gamma ||p_j||^{\gamma - 1} < 0, \text{ when } \gamma < 0. \tag{A.8}
\]
We check the second condition as well
\[
f(||p_j||) = ||p_j||^\gamma > 0, \text{ when } \gamma < 0. \tag{A.9}
\]
Finally for $||p_j|| > ||p_k||$,
\[
\frac{f(||p_j||)}{f(||p_k||)} = \left( \frac{||p_j||}{||p_k||} \right)^\gamma < 1^\gamma = 1,
\]
\[
\frac{f(||p_j||)}{f(||p_k||)} = \left( \frac{||p_j||}{||p_k||} \right)^\gamma = \left( \frac{||p_j||}{||p_k||} + \frac{||p_j|| - ||p_k||}{||p_k||} \right)^\gamma = (1 + \xi)^\gamma \tag{A.10}
\]
Now, we want to show that there exists some $\gamma < 0$ s.t. $(1 + \xi)^\gamma > 1 - \epsilon$ where $\xi > 0$ and $0 < \epsilon < 1$. We write,
\[
(1 + \xi)^\gamma > 1 - \epsilon \iff \gamma > \frac{\ln(1 - \epsilon)}{\ln(1 + \xi)}. \tag{A.11}
\]
We have that
\[
\begin{cases}
-\infty < \ln(1 - \epsilon) < 0, \\
0 < \ln(1 + \xi) < \infty,
\end{cases}
\implies -\infty < \frac{\ln(1 - \epsilon)}{\ln(1 + \xi)} < 0. 
\tag{A.12}
\]

Thus, all \( \gamma \in \left[ \frac{\ln(1 - \epsilon)}{\ln(1 + \xi)}, 0 \right] \) suffices. However, the choice of \( \gamma \) should hold for all pairs in the point set so the final condition becomes \( \gamma \in \left[ \frac{\ln(1 - \epsilon)}{\ln(\max_{p_j \in P} \|p_j\|) - \ln(\min_{p_k \in P} \|p_k\|)}, 0 \right] \). Hence, we can conclude that for any pair \( \|p_j\|, \|p_k\| \in \mathbb{R}^+ \) s.t. \( \|p_j\| > \|p_k\| \), and for any \( 0 < \epsilon < 1 \), there does exist a \( \gamma \) s.t.
\[
1 - \epsilon < \frac{f_t(\|p_j\|)}{f_t(\|p_k\|)} < 1.
\]

In all cases, \( \gamma < 0 \) must hold. \( \square \)
A.2 Information Theory

A.2.1 Simplified Expression for the Differential Entropy of the Multivariate Normal Distribution

Proof: The probability density of $X$ is

$$ f(x) = \frac{1}{(\sqrt{2\pi})^D (\det(\Sigma))^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right). \quad (A.13) $$

Using Definition 2.10 with the natural logarithm, we can now write

$$ h(X) = -\int f(x) \ln \left(\frac{1}{(\sqrt{2\pi})^D (\det(\Sigma))^{1/2}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)\right) dx $$

$$ = \ln \left(\left(\sqrt{2\pi}\right)^D (\det(\Sigma))^{1/2}\right) \int f(x) dx + \int f(x) \left(\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right) dx, \quad (A.14) $$

where we utilize the fact that the integral of a probability density function over its entire support is one and the definition of the expected value. Now, we continue by writing

$$ h(X) = \ln \left(\left(\sqrt{2\pi}\right)^D (\det(\Sigma))^{1/2}\right) + \frac{1}{2} \mathbb{E} \left[(x - \mu)^T \Sigma^{-1} (x - \mu)\right] $$

$$ = \frac{1}{2} \ln \left(2\pi^D \det(\Sigma)\right) + \frac{1}{2} \mathbb{E} \left[\text{Tr} \left((x - \mu)^T \Sigma^{-1} (x - \mu)\right)\right] \quad (A.15) $$

$$ = \frac{1}{2} \left(\ln \left(2\pi^D \det(\Sigma)\right) + \mathbb{E} \left[\text{Tr} \left(\Sigma^{-1} (x - \mu)(x - \mu)^T\right)\right]\right), $$

where we exploited the property that a scalar is equal to the trace of the scalar and then used the cyclic property of the trace operator.

$$ h(X) = \frac{1}{2} \left(\ln \left(2\pi^D \det(\Sigma)\right) + \text{Tr} \left(\Sigma^{-1} \mathbb{E} \left[(x - \mu)(x - \mu)^T\right]\right)\right) $$

$$ = \frac{1}{2} \left(\ln \left(2\pi^D \det(\Sigma)\right) + \text{Tr} \left(\Sigma^{-1} \mathbb{E} \left[(x - \mu)(x - \mu)^T\right]\right)\right) $$

$$ = \frac{1}{2} \ln \left(2\pi^D \det(\Sigma)\right) + \frac{1}{2} \left(\ln \left(2\pi^D \det(\Sigma)\right) + \ln(e^D)\right) $$

$$ = \frac{1}{2} \ln \left(2\pi e^D \det(\Sigma)\right) \text{nats}. \quad (A.16) $$

\qed
A.3 Point Cloud Densities

A.3.1 Deriving the Adaptive Radii

Assume we have two 2-spheres of radius $d_0$ and $d_1$, respectively. In the origin of these spheres, one LIDAR is located each. Then, for both spheres, area elements made up of the area between any set of four adjacent points can be found. The distances between the points on the spheres are determined by the sensor resolutions $\alpha$ (vertical resolution) and $\beta$ (horizontal resolution). Additionally, we make the following assumptions:

- We disregard occlusions as this makes modeling expected densities significantly more involved.

- We assume the small area elements’ curvature on the 2-spheres are neglectable (if this holds well in practice depends on the size of the examined area).

Studying case 1, the minimal length in the horizontal direction between two adjacent points at the same distance, $d$, from a sensor is $\beta d$ where $\beta \approx \frac{360}{1080}$. Similarly, the minimal length in the vertical direction would be $\alpha d$ where $\alpha \approx \frac{\pi}{180}$. Hence, the minimal area element connecting neighboring points at a distance $d$ from the LIDAR would be $A_0(d) \approx \alpha \beta d^2$. If we start expanding the area to include $n_0 \cdot m_0$ points, this has $(n_0 - 1)(m_0 - 1)$ area elements. So, the density would approximately be

$$D_0(d) = \frac{\mathbb{E}[P_0(A_0(d))]}{A_0(d)} \approx \frac{n_0 m_0}{(n_0-1)(m_0-1)} \cdot \frac{1}{\alpha \beta d^2}. \quad (A.17)$$

Considering both LIDARS, the joint density would be

$$D_{0,1}(d) = 2 \mathbb{E}[P_0(A_0(d))] \approx \frac{2n_0 m_0}{(n_0-1)(m_0-1)\alpha \beta d^2}. \quad (A.18)$$

If $d_0 > d_1$ in case 2, there would be one larger area $\tilde{A}_0(d_0)$ and one smaller area $\tilde{A}_1(d_1)$. The densities for the larger and smaller areas would be

$$\tilde{D}_i(d_i) = \frac{\mathbb{E}[\tilde{P}_i(\tilde{A}_i(d_i))]}{\tilde{A}_i(d_i)} \approx \frac{\tilde{n}_i \tilde{m}_i}{(\tilde{n}_i-1)(\tilde{m}_i-1)} \cdot \frac{1}{\alpha \beta d_i^2}, \quad i \in \{0, 1\}. \quad (A.19)$$

This gives the joint density

$$\tilde{D}_{0,1}(d_0, d_1) \approx \frac{\tilde{n}_0 \tilde{m}_0}{(\tilde{n}_0-1)(\tilde{m}_0-1)} \cdot \frac{1}{\alpha \beta d_0^2} + \frac{\tilde{n}_1 \tilde{m}_1}{(\tilde{n}_1-1)(\tilde{m}_1-1)} \cdot \frac{1}{\alpha \beta d_1^2} = \frac{\tilde{n}_0 \tilde{m}_0}{(\tilde{n}_0-1)(\tilde{m}_0-1)d_0^2} + \frac{\tilde{n}_1 \tilde{m}_1}{(\tilde{n}_1-1)(\tilde{m}_1-1)d_1^2}. \quad (A.20)$$
As \( n \) and \( m \) become large \( \frac{mn}{(m-1)(n-1)} \rightarrow 1 \). If we set the quotient to one, we obtain

\[
D_{0,1}(d) \approx \frac{2}{\alpha \beta d^2},
\]

\[
\tilde{D}_{0,1}(d_0, d_1) \approx \frac{1}{\alpha^2} + \frac{1}{\beta^2} = \frac{d_0^2 + d_1^2}{\alpha \beta (d_0 d_1)^2},
\]

(A.21)

where we have the vital property \( \tilde{D}_{0,1}(d, d) = D_{0,1}(d) \). In Appendix A.3.2, we show a second vital property, namely that \( \tilde{D}_{0,1}(d_0, d_1) > D_{0,1}(d) \) for \( \alpha, \beta, d_0, d_1 > 0, d_0 \neq d_1 \). Now, we can map approximated densities \( \tilde{D}_{0,1}(d_0, d_1) \) to different radii using a suitable function \( g(\cdot) \). The radius should increase as the density decreases. To obtain a function that scales linearly with the combined magnitude of \( d_0 \) and \( d_1 \), we write the following:

\[
g(\tilde{D}_{0,1}(d_0, d_1)) = \frac{1}{\sqrt{\tilde{D}_{0,1}}} = \sqrt{\frac{\alpha \beta (d_0 d_1)^2}{d_0^2 + d_1^2}} \propto \frac{\sqrt{2d_0 d_1 \sin(5\alpha \frac{\pi}{180})}}{\sqrt{d_0^2 + d_1^2}}
\]

(A.22)

where

\[
\lim_{d_0 \rightarrow d_1} \frac{\sqrt{2d_0 d_1 \sin(5\alpha \frac{\pi}{180})}}{\sqrt{d_0^2 + d_1^2}} = \frac{\sqrt{2d^2 \sin(5\alpha \frac{\pi}{180})}}{\sqrt{2d^2}} = d \sin\left(5\alpha \frac{\pi}{180}\right).
\]

(A.23)

This is a third vital property as we get back the default radii criterion.
A.3.2 Surface Densities of Varying Data Positions

Let \( 2d = d_0 + d_1 \) and

\[
D_{0,1}(d) = \frac{2}{\alpha \beta d^2}, \quad \tilde{D}_{0,1}(d_0, d_1) = \frac{d_0^2 + d_1^2}{\alpha \beta (d_0 d_1)^2}
\]

where \( \alpha, \beta, d_0, d_1 > 0, d_0 \neq d_1 \). Let us show that \( \tilde{D}_{0,1}(d_0, d_1) > D_{0,1}(d) \).

**Proof:** We write

\[
\tilde{D}_{0,1}(d_0, d_1) > D_{0,1}(d) \iff \frac{d_0^2 + d_1^2}{(d_0 d_1)^2} > \frac{2}{d^2} \iff \left\{ d^2 = \frac{(d_0 + d_1)^2}{4} \right\}
\]

(A.25)

Expanding the L.H.S., we obtain

\[
(d_0^2 + d_1^2)(d_0 + d_1)^2 = (d_0^2 + d_1^2)(d_0^2 + 2d_0 d_1 + d_1^2)
\]

\[
= d_0^4 + 2d_0^3 d_1 + 2d_0^2 d_1^2 + 2d_0 d_1^3 + d_1^4.
\]

(A.26)

Inserted into the original inequality, we get

\[
d_0^4 + 2d_0^3 d_1 + 2d_0^2 d_1^2 + d_1^4 > 6d_0^2 d_1^2.
\]

(A.27)

Now, we can write \( (d_0^2 - d_1^2)^2 > 0 \iff d_0^4 + d_1^4 > 2d_0^2 d_1^2 \). Similarly, we get

\[
2d_0^3 d_1 + 2d_0^2 d_1^3 = 2d_0 d_1 (d_0^2 + d_1^2) > 2d_0 d_1 (2d_0 d_1) = 4d_0^2 d_1^2.
\]

(A.28)

Finally, we conclude that \( \tilde{D}_{0,1}(d_0, d_1) > D_{0,1}(d) \) for \( \alpha, \beta, d_0, d_1 > 0, d_0 \neq d_1 \). \( \square \)
A.4 Scene-Dependent Classification Accuracy

We conduct a hypothesis test to determine if the expected number of correct classifications is equal for all scenes. The test is constructed as follows on all 212 test scenes:

\[ H_0 : P(Z = k | Z \in Y_j) = P(Z = k), \]
\[ H_a : P(Z = k | Z \in Y_j) \neq P(Z = k), \]  

(A.29)

where \( Z \) denotes the number of correct classifications in an unspecified scene and \( Y_j \) denotes scene \( j \). To test the hypothesis, a \( \chi^2 \)-test is performed. Without information about from which scene a sample is drawn, the expected accuracy is \( p = 0.924 \) (see Table 5.7). Assuming the samples are scene-independent, the expected probability of obtaining \( k \) correct classifications for \( n = 20 \) samples is

\[ P(Z = k) = \binom{n}{k} p^k (1 - p)^{n-k}. \]  

(A.30)

Hence, the expected number of scenes with \( k \) correct classifications is \( z_k = np(1 - p)^{n-k} \). For the assumptions of a \( \chi^2 \)-test to hold, at least 80% of the expected values should be more than 5 [50]. Thus, we join categories 0-15. With the data visualized in Figure A.1, we calculate the \( \chi^2 \)-statistic with the observed and expected values \( m_k \) and \( z_k \):

\[ X^2 = \sum_{k=15}^{n} \frac{(m_k - z_k)^2}{z_k} = 37.29, \]  

(A.31)

With \( n - 15 = 5 \) degrees of freedom, the p-value is \( 5.23 \cdot 10^{-7} \). Hence, we reject \( H_0 \), deducing that there is variation in classification difficulty across scenes.

Figure A.1: Observed and expected number of correct classifications.
This chapter contains additional images.

**B.1 Point Cloud Pairs**

The following describes the figures displayed in this section. Each row shows both a correct and an incorrect prediction made by the classifier FACT\(_{\text{optimal}}\). In each plot, the first separate point cloud is colored blue, while the second one is colored red. Every figure has a title with the prediction made by the classifier and the ground truth label. All raw data comes from nuScenes [18].

Figure B.1: Class 0, see Appendix B.1 for the remaining figure text.
Figure B.2: Class 1 to 3, see Appendix B.1 for the remaining figure text.
Prediction: class 4, GT: class 4
Prediction: class 5, GT: class 4
Prediction: class 5, GT: class 5
Prediction: class 6, GT: class 5
Prediction: class 6, GT: class 6
Prediction: class 7, GT: class 6

*Figure B.3:* Class 4 to 6, see Appendix B.1 for the remaining figure text.
Figure B.4: Class 7 to 9, see Appendix B.1 for the remaining figure text.
Bibliography


[82] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for vol-


