Utrecht University

Faculty of Science
Graduate School of Natural Sciences

Mathematical Sciences Master's Thesis

Probabilistic Methods for Point Cloud Registration Problem

Supervisor:
Prof. Dr. Tristan van Leeuwen

Author:
Liang Xu

Second reader:
Dr. Sjoerd Dirksen

Internship:
Computational Imaging Group
Centrum Wiskunde & Informatica

July 7, 2024

Abstract

Over the past few decades, probabilistic methods have been one of the popular approaches for registering point clouds generated by LiDAR systems. Such methods consist of two major steps, i.e., using probabilistic models to represent point clouds and finding optimal transformation to align point clouds with the help of some statistical distances. In this thesis, a theoretical framework of probabilistic methods is studied, which provides a foundation for understanding and implementing point cloud registration with some specific probabilistic models and distance measures. Specifically, the concepts of Gaussian Mixture Models and Kernel Density Estimation are explored, with a detailed discussion of their practical implementation. Furthermore, Kullback-Leibler divergence and Wasserstein distance, including the computation of Wasserstein distance through Kantorovich-Rubinstein duality, are also studied. An approximation of Wasserstein distance that preserves differentiability through linear programming techniques is proposed, which enables the use of gradient-based methods on Wasserstein distance to find the optimal transformation that aligns two point clouds. The algorithms that contribute to a complete procedure for solving point cloud registration problem with our proposed methods are also discussed and demonstrated.

Acknowledgements

First and foremost, I am extremely grateful to my first supervisor, Prof. Dr. Tristan van Leeuwen, for his invaluable advice, continuous support, and patience during my research project. His expertise and encouragement have been instrumental in my research and professional development, and I am thankful for the opportunity to intern at Centrum Wiskunde & Informatica (CWI) under his guidance.

I would also like to thank my second supervisor, Dr. Sjoerd Dirksen, for being second reader of my thesis and for attending my final presentation.

I am deeply appreciative of my office mates, Vladyslav Andriiashen, Adriaan Graas, Hamid Fathi, and Xinyuan Wang. Their camaraderie and assistance have made our office a pleasant and productive place to work. My sincere thanks go to Dr. Alexander Skorikov for his support in programming. His guidance has been pivotal in the technical aspects of my research. Additionally, I am grateful to my other colleagues at CWI, including but not limited to Marcos Obando, Dr. Richard Schoonhoven, Dr. Felix Lucka, Dr. Ezgi Demircan Türeyen, and Dirk Schut. Their insights and suggestions have greatly enriched my research experience.

A special thanks to Centrum Wiskunde & Informatica for offering the internship and providing a stimulating research environment.

I extend my heartfelt thanks to those who have provided advice and support for my studies and life abroad, especially Dr. Paul Zegeling, my tutor, for helping me create a comprehensive study plan. His direction has been invaluable in navigating my master's academic journey. I would also like to thank Tianyuan Wang, Yeyang Hu, and my parents. Their encouragement and understanding have been a source of strength throughout this journey.

Lastly, I wish to thank everyone who has contributed, directly or indirectly, to my research and personal growth during this period. Your support has been invaluable, and I am truly grateful.

Contents

1 Introduction **1**
1.1 Related research 2
1.1.1 Feature-based methods 3
1.1.2 Optimization-based methods 4
1.1.2.1 ICP and variants 4
1.1.2.2 Semi-definite registration 5
1.1.2.3 Probabilistic methods 6
1.1.3 Other methods 10
1.2 Outline and contributions 11

2 Theory **13**
2.1 Gaussian Mixture Model 13
2.1.1 E-M algorithm 13
2.1.2 Estimate number of components 14
2.2 Kernel Density Estimation 15
2.2.1 Kernel functions 15
2.2.2 Choosing bandwidth 16
2.3 K-L divergence 16
2.4 Wasserstein distance 17
2.4.1 Kantorovich-Rubinstein Duality 19
2.5 Gradient-based methods to optimize the transformation 23
2.5.1 Problem formulation 23
2.5.2 The existence of gradients and the feasibility of optimization 24

3 Algorithms and implementation **29**
3.1 Modelling point clouds 29
3.2 Finding optimal transformation 30
3.3 Full procedure 32

4 Experiments **35**
4.1 Dataset 35

4.2 Probability models 36
4.2.1 Gaussian Mixture Model 36
4.2.2 Kernel density estimation 38
4.3 Different number of basis functions 38
4.4 Comparing approximate Wasserstein distance with K-L divergence 41
4.5 Point cloud registration with approximate Wasserstein distance 43

5 Conclusion and outlook **47**

A Appendix **49**
A.1 LBFGS Optimizer 49
A.2 Reference LBFGS algorithm embedded with **c** update 50
A.3 Code 52
A.4 Notations and conventions 52

Bibliography **53**

1. Introduction

In recent years, advancements in sensor technology, including Lidar, have paved the way for innovative applications in various fields. A Lidar system, which functions as a radar using light, employs a laser to aim at objects, and times the return journey of the reflected light to calculate the distance between itself and the target. In detail, a Lidar sensor consists of a transmitter that emits laser with wavelengths between 250 to 1600 nanometres and a receiver which can gather, analyse, and compute the reflected signal [1]. As introduced in paper [2], Lidar sensors are possible to be equipped by smartphones, and it is already possible to reconstruct small real-world objects and even indoor environments using Lidar sensors on handheld devices. Three-dimensional reconstruction of indoor objects can be used in many interesting and useful fields, such as augmented reality (AR), indoor navigation, and blind assistance.

A Lidar system usually can only emit one laser beam at a time, and the scanning range is expanded by fine-tuning the angle of the laser emission. The process of obtaining all the information within its field of view (FOV) through such multiple laser emission and reception processes is considered "a scan". Such a technique enables the acquisition of high-resolution point clouds representing (parts of) the objects. However, a single scan may not provide comprehensive data about an object. For instance, the FOV may be too narrow to scan an object fully, and because of noise and measurement error, the point cloud obtained from a single scan is less accurate than the point cloud obtained from multiple scans followed by noise reduction. Apart from this, to capture points from all six faces of a cube, intuitively a minimum of two scans from a single Lidar system would be required.

An important step in reconstructing 3D objects from point clouds produced by a Lidar system is point cloud registration. In this step, the point clouds are matched and aligned with each other. The essential concept underlying point cloud registration is achieving pose alignment between the source and target point clouds by finding the best transformation [3]. In this project, we focus exclusively on same-source point cloud registration, indicating that the point clouds under consideration originate from a single sensor. Consequently, these point clouds are likely to share specific attributes, such as comparable density and similar error distribution patterns. In addition to these assumptions, we also assume that the Lidar system can only obtain depth information for every point. In other words, it cannot obtain colour information, nor can it obtain information from other sensors such as gyroscopes and accelerometers. Such assumptions can significantly influence the choice of methods. Intuitively, on one hand, having more detailed information generally makes solving a problem easier. However, on the other hand, accumulating more information can also make the problem more complicated.

The goal of point cloud registration is to find accurate point-wise correspondences between two point clouds. The correspondence refers to the relationship between individual points in one point cloud and their equivalent points in another, i.e., the relationship between corresponding points in two point clouds, representing the same location on an object or in a scene. A usual approach is to fix one point cloud and transform the other one so that the potential corresponding points of the two point clouds are close enough that the nearest neighbour method can be used to find the correspondences. For example, as shown in Figure 1.1, given two sets of points $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ and $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$, every point in one set pairs with its nearest neighbour from the other set. Such an approach forms a well-known chicken-and-egg problem: the optimal transformation can be calculated easily if the accurate correspondences are known; in contrast, the correspondences can also be readily found if the optimal transformation is given [4].

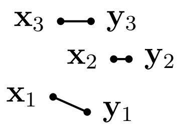


Figure 1.1: Pairs of nearest neighbours

In the following section, we will have a general understanding of effective strategies for resolving the chicken-and-egg paradox in point cloud registration. Specifically, a brief overview on how to transform a point cloud to align with other one despite not knowing the point-wise correspondences, or how to identify the correspondences without first aligning the clouds.

1.1 Related research

In general, currently there are numbers of categories of research direction on solving the point cloud registration problem. Specifically, we will provide overviews of feature-based methods, optimization-based methods, and other approaches, including those that utilize neural networks.

1.1.1 Feature-based methods

Feature-based registration is a crucial approach in point cloud registration, typically used when aligning two or more 3D point sets. This method can be broken down into three primary steps: feature detection, feature matching, and transformation estimation.

Some of the well-known related approaches are 3D SIFT [5], Fast Point Feature Histograms (FPHF) [6] and Signature of Histograms of Orientations (SHOT) [7].

3D SIFT extends a famous 2D image feature extraction algorithm, called Scale Invariant Feature Transform (SIFT), to three dimensions. The orientation invariance and feature selectivity are improved. The modified algorithm is particularly suited to analysing complex medical imagery where precise, robust feature matching is critical. FPHF is specifically designed to handle the challenges of surface registration in the context of robotic manipulation tasks. It provides a fast and accurate method for matching 3D point clouds. SHOT encodes local geometric properties by capturing the distribution of normal vectors within a local neighbourhood. It is not only designed for applications in matching and comparing 3D models, but also suitable for applications like object recognition and scene understanding.

[6] also demonstrates a full process of applying a feature-based method on 3D registration. For each point in the point cloud, it starts by computing a simplified version of the feature histogram (SPFH) based on the relationships between the point and its immediate neighbours within a specified radius. This includes calculating angular variations between the normals of the point and its neighbours. Then, FPFH for each point is obtained by aggregating its SPFH with those of its neighbours. This aggregation is weighted by the inverse of the distance between neighbours, effectively smoothing the feature histogram over a local neighbourhood and capturing more global contextual information. In the feature matching and transformation estimation step, FPFH features are used to quickly find correspondences between different sets of point clouds by matching similar histograms. These correspondences serve as initial guesses for aligning two datasets. Sample Consensus Initial Alignment (SAC-IA) algorithm further refines the initial alignment. It starts by randomly selecting a subset of points from the first point cloud. For each of the sampled points,

SAC-IA searches for corresponding points in the second point cloud based on the similarity of their FPFH descriptors. Once correspondences are established between pairs of points, SAC-IA computes the rigid transformation (including rotation and translation) that best aligns these matching point pairs. Furthermore, the process of selecting samples, finding correspondences, and calculating transformations is repeated multiple times to improve accuracy.

1.1.2 Optimization-based methods

For the majority of the optimization-based methods, the objective of the optimization problem is some distance between two point clouds. Given dataset of two point clouds, the distance can be viewed as some function of the transformation. The goal of the optimization problem is to find an optimal transformation that minimizes the distance. The mathematical representation of the optimization problem is

$$\underset{\mathbf{R}\in SO(3),\mathbf{t}\in\mathbb{R}^3}{\arg\min} \ \|d(\mathcal{P}, T_{\mathbf{R},\mathbf{t}}(\mathcal{Q}))\|^2, \tag{1.1}$$

where $\mathcal{P} \in \mathbb{R}^{n_{\mathcal{P}}\times 3}$ and $\mathcal{Q} \in \mathbb{R}^{n_{\mathcal{Q}}\times 3}$ are matrices denotes the two point clouds ($n_{\mathcal{P}}$ and $n_{\mathcal{Q}}$ denote numbers of points in $\mathcal{P}$ and $\mathcal{Q}$ respectively), T denotes the transformation on a point cloud and parametrized by rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$, while d is some metric that measures the "distance" between two point clouds. For example, the average Euclidean distance between points in $\mathcal{P}$ and the geometric centre of $\mathcal{Q}$. When $n_{\mathcal{P}} = n_{\mathcal{Q}} = n$, another example of the distance can be

$$d(\mathcal{P}, T_{\mathbf{R},\mathbf{t}}(\mathcal{Q})) = \sum_{i=1}^{n} \|\mathbf{x}_i - (\mathbf{R}\mathbf{y}_i + \mathbf{t})\|_2, \tag{1.2}$$

where $\mathbf{x}_i \in \mathcal{P}$ and $\mathbf{y}_i \in \mathcal{Q}$ is a pair of corresponding points. Note that d is also called projection error in some articles, e.g., [4]. Usually the moving point cloud $\mathcal{Q}$, is called source, while the fixed point cloud $\mathcal{P}$ is called target.

Notice that such an optimization problem also meets the chicken-or-egg paradox. However, the following methods can break the self-contradictory chain.

1.1.2.1 ICP and variants

Iterative closest point method (ICP) [8] is one of the most classic and popular optimization-based point cloud registration methods. It breaks the self-contradictory chain by iteratively giving better and better guesses on the relationships between points in two point clouds to achieve better and better transformation. Given an initial guess on the transformation, ICP iteratively finds a better transformation to align two point clouds. During each iteration, the algorithm identifies the closest points in the target set for each point in the source set. It then computes the optimal transformation that minimizes the distance between these paired points. This transformation is applied to the source point set to adjust its position and orientation. The process is repeated iteratively, with the source point cloud gradually being moved to better align with the target point cloud until the changes in the transformation parameters fall below a predefined threshold or after a set number of iterations.

Different from the original ICP algorithm that uses point-to-point distance, people also tried other distances, such as point-to-plane and plane-to-plane distances. For example, in [9], the point-to-plane distance evaluates the orthogonal distance between a point of a point cloud and the surface normal at a point in the other point cloud. Estimating the surface normal at a point involves its neighbouring points. Furthermore, plane-to-plane distance makes use of surface normals of both point clouds [10].

1.1.2.2 Semi-definite registration

Finding the optimal correspondences is a quadratic assignment problem (QAP) when considering paired correspondences constraint [4]. QAP is proved to be strongly NP-hard [11], which indicates that it is intractable to directly solve when the number of points is large. However, by projecting the problem to a semi-definite optimization problem (SDP), an approximation to the global optima is possible and relatively easier to achieve. For example, the correspondence problem is formulated as an optimization problem over permutation matrices which are difficult to solve directly due to their combinatorial nature. Then, the problem is "relaxed" into an SDP by replacing the permutation constraints with semi-definite constraints, making it solvable with convex optimization techniques. Specifically, in [12], the original problem is finding a permutation matrix X that best aligns the points in point cloud P to those in point cloud Q. The optimization problem can be formulated as:

$$\max_{X} \ \operatorname{trace}(AX) \tag{1.3}$$

subject to

$$X \in \{0,1\}^{N\times N}, \quad X\mathbf{1} = \mathbf{1}, \quad \mathbf{1}^T X = \mathbf{1}^T, \tag{1.4}$$

where A is a matrix of matching potentials, and the constraints ensure that X is a valid permutation matrix. Specifically, each entry X_{ij} in the matrix X represents the correspondence between the i-th point in P and the j-th point in Q; an entry of 1 indicates those two points are matched, and an entry of 0 indicates they are not matched.

This combinatorial problem is then relaxed into an SDP by lifting it into a higher-dimensional space and replacing the binary constraints on X with semi-definite constraints. The relaxed problem can be written as:

$$\max_{X,Y} \ \text{trace}(AY) \tag{1.5}$$

subject to

$$\begin{cases} Y - XX^T \succeq 0, \\ X \in [0,1]^{N\times N}, \\ \sum_i X_{ij} \leq 1, \\ \sum_j X_{ij} \leq 1, \\ \text{trace}(Y) = m, \end{cases} \tag{1.6}$$

where $Y = XX^T$ and m is the number of correspondences. The semi-definite constraint $Y - XX^T \succeq 0$ ensures that Y is a valid semi-definite matrix.

While the QAP remains a challenging combinatorial problem, the use of SDP relaxations provides a powerful tool for approximating solutions in a computationally feasible manner. Recent advancements have introduced more complex relaxation techniques that further improve the quality of the solution and the computational efficiency. For example, the use of dual decomposition, cutting planes, and other advanced methods has been shown to enhance the performance of SDP relaxations.

1.1.2.3 Probabilistic methods

The probabilistic methods essentially model point clouds using probability distributions. In this way, one can employ some "distance measure" (such as divergence, entropy, likelihood, etc.) in probability theory to evaluate the difference between two probability distributions representing two point clouds to avoid knowing the correct point-wise correspondence in advance. The goal of such methods is to find an optimal transformation such that the difference between those two distributions is the smallest. For example, point clouds can be modelled by Gaussian Mixture Models (GMM). One can use the likelihood function of GMM to measure whether the positions of X and $T(Y)$ are close enough. Given point clouds $\mathcal{P}$ and $\mathcal{Q}$, the registration from $\mathcal{Q}$ to $\mathcal{P}$ can be transformed into 2-step optimization problem [13]:

$$\textbf{Fitting:} \begin{cases} \Theta^*_{\mathcal{P}} = \arg\max_{\Theta} \mathbb{P}_{\mathbb{M}}(\mathcal{P} \mid \Theta) \\ \Theta^*_{\mathcal{Q}} = \arg\max_{\Theta} \mathbb{P}_{\mathbb{M}}(\mathcal{Q} \mid \Theta) \end{cases}; \tag{1.7}$$

$$\textbf{Registration:}\ T^* = \arg\min_{T} d[\mathbb{M}(\Theta^*_{\mathcal{P}}), \mathbb{M}_T(\Theta^*_{\mathcal{Q}})]; \tag{1.8}$$

where Θ denotes the tuple of probability model parameters, $\mathbb{P}_{\mathbb{M}}(\cdot \mid \Theta)$ denotes likelihood function of probability model $\mathbb{M}$ given the tuple of parameters Θ, $\mathbb{M}(\Theta)$ denotes a probability model with some certain tuple of parameters Θ (in other words, $\mathbb{M}(\Theta)$ is some certain probability distribution), d denotes some "distance measure" between two probability distribution, and T denotes transformation on probability distribution, i.e.,

$$\mathbb{P}_{\mathbb{M}_T}(\mathbf{x} \mid \Theta) = \mathbb{P}_{\mathbb{M}}(\mathbf{R}\mathbf{x} + \mathbf{t} \mid \Theta), \tag{1.9}$$

where $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation vector, which will be described in detail in latter chapter.

Modelling Point clouds

The most popular probabilistic model in point cloud registration problem is Gaussian Mixture Model (GMM). A GMM is a weighted sum of K Gaussian distributions,

$$p(\mathbf{x} \mid \Theta) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{1.10}$$

where $\mathbf{x}$ is a D-dimensional vector representing the point ($\mathbf{x} \in \mathbb{R}^3$ in 3D case), w_k are the mixture weights that satisfy $\sum_k w_k = 1$ and $w_k \geq 0$, and each $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is a component that denotes a Gaussian distribution with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$ [14]:

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\intercal \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right\}. \tag{1.11}$$

In general, since Equation 1.10 is a non-linear function of the parameters, it is impossible to directly maximize the likelihood using given samples [14]. A common approach to estimate the parameters of GMM is Expectation-Maximization (E-M) algorithm [15], which iteratively finding the optimal parameters. The fundamental idea of E-M algorithm is by introducing latent variables, one can

1. Develop a function that calculates the expected value of the log-likelihood based on the current estimated parameters during the E-step.
2. Update the parameters to optimize the log-likelihood function in the M-step.

Given an initial value, and then continuously repeating E-step and M-step, the likelihood and parameters will converge to their optimal.

Moreover, one has to choose the number of components for GMM. On the one hand, too few components will make the GMM unable to reflect the characteristics of the given point cloud well; on the other hand, too many components will make the GMM overfit the point cloud. Recently, many model selection criteria have been proposed to determine the optimal number of components (K_{OPT}) for controlling over-fitting and providing a standardized way to balance sensitivity and specificity via log-likelihood functions with simple penalties [16]. For example, Bayesian information criterion (BIC) [17] and Akaike information criterion (AIC) [18] are two wildly used criteria to determine K_{OPT}.

Another possible choice for modeling point clouds is Kernel Density Estimation (KDE). Compared to GMM, KDE is a non-parametric method to estimate the probability density function. In other words, the probability density can be estimated without knowing what kind of distribution the population is, which is exactly the case for the point cloud registration problem. KDE can be expressed as

$$\hat{p}(x) = \frac{1}{nh^D} \sum_{i=1}^{n} \mathcal{K}\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \tag{1.12}$$

where $\hat{p}$ is the estimated density function for an unknown distribution with n samples denoted by $\mathbf{x}_i \in \mathbb{R}^D$, $\mathcal{K} : \mathbb{R}^D \mapsto \mathbb{R}$ is a smooth function as known as the kernel function, and $h > 0$ denotes the smoothing bandwidth [19].

A commonly used kernel is Gaussian kernel, which is the probability density function for Gaussian distribution,

$$\mathcal{K}_{\mathcal{N}} = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{1.13}$$

with a usually choice of the parameters $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$.

Distances for probability distributions

Up to now, numerous metrics have been developed to measure the discrepancy between two probability distributions. For examples, likelihood, Kullback-Leibler divergence (K-L divergence) and its variants, Bhattacharyya distance, and Wasserstein distance.

We can prove that likelihood and K-L divergence are equivalent in the point cloud registration problem, which will be discussed in a later chapter. Given probability density functions $p(\mathbf{x})$ and $q(\mathbf{x})$ for two multi-dimensional continues probability distributions P and Q, the K-L divergence is

$$\mathrm{KLD}(P, Q) = \int_{\mathbf{x}\in\mathbb{R}^D} p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) \mathrm{d}V, \tag{1.14}$$

In general, the Kullback-Leibler divergence and its variants measure the difference between two probability distributions by integrating the relative entropy (logarithmic ratio) over their support. Since K-L divergence is non-symmetric, Jensen-Shannon Divergence is preferred over it when symmetric property is required:

$$\mathrm{JSD}(P, Q) = \frac{1}{2}\mathrm{KLD}(P, M) + \frac{1}{2}\mathrm{KLD}(Q, M), \tag{1.15}$$

where $M = \frac{1}{2}(P + Q)$ is a mixture distribution of P and Q.

Wasserstein distances are metrics on probability distributions inspired by the problem of optimal mass transportation [20]. Research on it has become popular in recent years due to active studies on machine learning and neural networks, for example, Wasserstein GAN [20]. Unlike K-L divergence and many other distance measures, Wasserstein distance is an optimization problem:

$$W_p(\mu, \nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \left(\int_{\mathbb{R}^D\times\mathbb{R}^D} d(\mathbf{x}, \mathbf{y})^p \, \mathrm{d}\gamma(\mathbf{x}, \mathbf{y})\right)^{1/p}, \tag{1.16}$$

where μ and ν are probability measures on $\mathbb{R}^D$, $p \in [1, +\infty]$, $\Gamma(\mu, \nu)$ denotes the set of all couplings of μ and ν, and d is some metric defined on $\mathbb{R}^D$. A coupling is joint probability distributions that connect two given marginal probability distributions, which means that the marginal distribution of $\gamma \in \Gamma$

over its first factor is μ, while the marginal distribution over its second factor is ν.

Wasserstein distance for $p = 1$ is also known as Earth Mover Distance (EMD). Intuitively, $d\gamma(\mathbf{x}, \mathbf{y})$ can be interpreted as a small shovel of earth moving from $\mathbf{x}$ to $\mathbf{y}$. Using Euclidean distance, $d(\mathbf{x}, \mathbf{y})$ quantifies the distance over which the mass is moved. Integrating these differential transfers over all possible pairs of source and target locations provides a measure of the total work required to transport mass distribution μ to mass distribution ν. Consequently, the objective of the optimization is to determine the transportation plan that minimizes this total work.

However, although there is already a closed-form solution for Wasserstein distance in one-dimensional probability measures, it is much more complex to obtain solutions in multidimensional cases. There are rare cases where closed-form solutions exist for it. Evaluating multidimensional Wasserstein distance is numerically intractable in general [21]. Some approximations to it are studied, such as wavelet EMD [22], sliced-Wasserstein distance and its variants [21], and neural network approximation [23].

Different combinations of probabilistic models and distance measures

Table 1.1 shows some key corresponding works on some popular probabilistic models and some distance measures.

Articles / probabilistic model / Distance measure	GMM	KDE	Other models
Log-likelihood	[13], [24]	[25]	
K-L divergence	[26]	[27]	
Wasserstein distance			[28]

Table 1.1: Different choices for probabilistic models and distance measures

1.1.3 Other methods

There has been substantial research on the problem of 2D image registration, leading to the development of numerous effective 2D registration algorithms. Therefore, researchers have begun exploring methods to project 3D point clouds onto 2D spaces so that 2D image registration algorithms can be applied. For example, [29] proposed a method that transforms the point clouds into 2D bearing angle images and then uses the 2D feature-based matching method, Speeded-Up Robust Features (SURF) [30], to identify corresponding pixel pairs between the resulting 2D images. The optimal transformation to align the 3D point clouds can be inferred from the established 2D image pixel correspondences.

Methods based on end-to-end neural networks would be more straightforward for users. Different from previous approaches, the inputs of the neural network are point clouds, and the outputs are optimal transformations that align the input point clouds [31] or correspondences of (key) points [32].

There are also researchers looking for hybrid methods that complement each others strengths and weaknesses. For instance, some hybrid methods integrate neural network-based feature extraction with traditional optimization techniques. This combination allows for efficient feature learning while leveraging the robustness of traditional optimization algorithms. For example, [33] uses AlexNet to learn features and find key points, then match the key points and calculate the transformation with some other methods like random sample consensus (RANSAC) algorithm [34]. [35] uses neural networks to evaluate "feature-metric" that measures the proximity of point clouds' location and position. This hybrid approach can enhance the accuracy and efficiency of the correspondence problem-solving process. Another kind popular hybrid methods involve integrating end-to-end neural networks with traditional registration methods. The neural network provides an initial estimate of the transformation or correspondences, which are then refined using traditional optimization techniques.

1.2 Outline and contributions

In this thesis, we focus on probabilistic methods for point cloud registration problem. A concise introduction to the point cloud registration problem and its relevant research has already been presented in chapter 1. In chapter 2, relevant theories of probabilistic methods will be introduced. First, two probabilistic models for point cloud modelling are discussed, specifically the Gaussian Mixture Model and the Kernel Density Estimation. Algorithms for estimating the parameters and the number of components in the Gaussian Mixture Model will also be addressed. For Kernel Density Estimation, we utilize Gaussian kernel and introduce a methodology for setting the bandwidth. Subsequently, the theoretical foundations of Kullback-Leibler (K-L) divergence and Wasserstein distance will be introduced. We propose that minimizing the K-L divergence in point cloud registration is equivalent to maximizing the likelihood function, providing an intuitive proof of the intrinsic connection between them. Next, relevant theories of the Wasserstein distance and its dual problem, namely the Kantorovich-Rubinstein Duality, will be introduced along with their mathematical formulations. An approach to approximate Wasserstein distance by transforming Kantorovich-Rubinstein Duality into a linear programming problem is proposed. We also prove that such approach this approach preserves the gradients with respect to the transformation, and thus gradient-based methods can be applied.

In chapter 3, discussions on relevant algorithms and their implementation details will be presented. The chapter 4 showcases some representative experiments, from which conclusions are drawn that the proposed method is effective when point clouds are relatively close and have limited degrees of freedom (one or two degrees of freedom) in transformation. The experiments empirically demonstrate that the method preserves gradients and can correctly compute them, as proved in the theoretic part. However, due to the limitations of the proposed basis functions, the problem is not strictly convex and possesses many stationary points (local optima), making the proposed algorithm ineffective when the point clouds are more distant or when there are more degrees of freedom.

Overall, in this thesis, by addressing the practical issue of the point cloud registration problem, we proposed a novel method that approximates the Wasserstein distance between two probability measures while preserving its differentiability with respect to transformation on one of the probability measure. Both theoretical analysis and practical experiments demonstrate the effectiveness of this approach to a certain extent. Furthermore, we hypothesize that the quality of approximation and its applicability could be enhanced by identifying better basis functions.

2. Theory

2.1 Gaussian Mixture Model

2.1.1 E-M algorithm

Recall the Gaussian Mixture Model in subsubsection 1.1.2.3, an iterative algorithm called Expectation-maximization algorithm is commonly applied to it to determine parameters for the components. The algorithm iterates through two main steps to find the parameters that maximize the likelihood of the observed data:

1. **Expectation Step (E-step)**: Calculate the expected value of the log-likelihood function with respect to the conditional distribution of the latent variables given the observed data and the current estimate of the parameters. This involves computing the responsibility $\gamma(z_{nk})$ that component k has for observation n:

$$\gamma(z_{nk}) = \frac{w_k \mathcal{N}(\mathbf{x}_n | \mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} w_j \mathcal{N}(\mathbf{x}_n | \mu_j, \mathbf{\Sigma}_j)}, \tag{2.1}$$

   where $\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)$ is the probability density of the n-th data point given the k-th component. γ is also called responsibility, which reflects how likely a data point belongs to a particular component of the mixture.

2. **Maximization Step (M-step)**: Update the parameters to maximize the expected log-likelihood found in the E-step. The new parameters are calculated as follows:
   - Update the mixing weights:

$$w_k^{new} = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk}). \tag{2.2}$$

   - Update the means:

$$\boldsymbol{\mu}_k^{new} = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}. \tag{2.3}$$

- Update the covariances:

$$\mathbf{\Sigma}_k^{new} = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^T}{\sum_{n=1}^{N} \gamma(z_{nk})}. \tag{2.4}$$

These steps are repeated until the convergence criteria are met, usually when the change in log-likelihood or in the parameter values falls below a threshold.

2.1.2 Estimate number of components

As discussed before, many model selection criteria have been proposed to determine the optimal number of components K_{OPT}, such as AIC and BIC.

An alternative criterion is silhouette score [36], which is usually used for cluster quality analysis. The silhouette score is simply the mean of silhouette coefficient of all samples. For a point i, the silhouette coefficient can be calculated by

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \tag{2.5}$$

where $a(i)$ is mean intra-cluster distance for point i and $b(i)$ is the mean nearest-cluster distance for point i [36], supposing the point i is assigned to cluster A. The detailed explanation of $a(i)$ and $b(i)$ can be seen in [37].

Then, the silhouette score of a GMM fitted by point cloud $\mathcal{P}$ can be expressed as

$$sc(\text{GMM}_{\mathcal{P}}) = \frac{1}{n} \sum_{i=1}^{n} s(i), \tag{2.6}$$

where n is the number of points in $\mathcal{P}$. Note that in the concept of clustering in $\text{GMM}_{\mathcal{P}}$, the centers of clusters are determined by $\boldsymbol{\mu}$, and the points are registered to their closest clusters.

To determine the optimal K using the silhouette score, one effective method is to perform a linear search across all possible K values and select the one with the highest silhouette score as the optimal value. The algorithm is shown in Algorithm 1. This method is simple and effective, but has a potential disadvantage: the computation time depends on the time consumed by the E-M algorithm of GMM, which can become time-consuming as the dataset grows larger.

2.2 Kernel Density Estimation

In this section, we will first briefly explore the selection of kernel functions for Kernel Density Estimation. Then, we will select the standard Gaussian kernel and determine how to set the bandwidth, especially using the standard Gaussian kernel.

2.2.1 Kernel functions

[38] mentioned some commonly used kernel functions in the one-dimensional case, as shown in Table 2.1.

Kernel	$\mathcal{K}(x)$
Uniform	$\frac{1}{2} \quad (\lvert x\rvert \leq 1)$
Triangle	$(1-\lvert x\rvert) \quad (\lvert x\rvert \leq 1)$
Epanechnikov	$\frac{3}{4}(1-x^2) \quad (\lvert x\rvert \leq 1)$
Biweight	$\frac{15}{16}(1-x^2)^2 \quad (\lvert x\rvert \leq 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$

Table 2.1: Some common choices of the kernel [38]

Gaussian kernel is a popular choice [38]. For multidimensional cases, the formula for it is shown as follow

$$\mathcal{K}_{\mathcal{N}}(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}}} \exp\left(-\frac{\|\mathbf{x}\|_2^2}{2}\right). \tag{2.7}$$

There are multiple reasons that we also use the Gaussian kernel in our approach. One of the reasons is that the support of the Gaussian kernel is $\mathbb{R}^D$ without any constraint, which is different from other kernels in Table 2.1. Such property makes it easier to implement in practice. Another reason is that it is well studied and there is a good result that is beneficial to use in choosing bandwidth practically. Furthermore, by using Gaussian kernel, KDE and GMM arrive at the same destination through different paths. KDE can be regarded as a special GMM, or vice versa. More preciously, KDE can be viewed as the GMM that has the same number of components as the number of points in point cloud, where each component is a standard Gaussian distribution scaled by a factor.

2.2.2 Choosing bandwidth

The quality of the estimator depends more on the choice of the bandwidth than the choice of the kernel [38]. A common criterion used to select bandwidth is the mean integrated squared error:

$$\mathrm{MISE}(\hat{p}) = \mathbb{E}\left[\int_{\mathbb{R}^D} (\hat{p}(\mathbf{x}) - p(\mathbf{x}))^2 \,\mathrm{d}V\right], \tag{2.8}$$

where $\hat{p}$ is the estimated density function and p is the real (but usually unknown) density function.

Referring to [38], one can derive that by minimizing the asymptotic MISE (AMISE) which is related to MISE. The optimal bandwidth would be

$$h_{\text{opt}} = n^{-\frac{1}{5}} \left[\int \|\mathbf{x}\|^2 \mathcal{K}(\mathbf{x}) \,\mathrm{d}V\right]^{-\frac{2}{5}} \left[\int \mathcal{K}(\mathbf{x})^2 \,\mathrm{d}V\right]^{\frac{1}{5}} \left[\int \|p''(\mathbf{x})\|^2 \,\mathrm{d}V\right]^{-\frac{1}{5}}, \tag{2.9}$$

where n denotes the number of samples.

However, since p is unknown, it is difficult to analytically obtain h_{opt} via this approach in practice. Refers to [39], under the assumption that the kernel is Gaussian, the optimal bandwidth h_{opt} for KDE can be approximated by the following formula:

$$h_{\text{opt}} = 1.06 n^{-\frac{1}{5}} \hat{\Sigma}, \tag{2.10}$$

where $\hat{\Sigma}$ represents the observed standard deviation of the sample data. For multidimensional data, $\hat{\Sigma}$ can take the average of standard deviation of each dimension.

2.3 K-L divergence

As mentioned in subsubsection 1.1.2.3, we claim that

Claim 1. *Kullback-Leibler divergence and likelihood are equivalent in the context of probabilistic methods for point cloud registration problem.*

Proof. Suppose we have two point clouds $\mathcal{P}$ and $\mathcal{Q}$, the i-th point in the point clouds is denoted by $\mathbf{x}_i \in \mathcal{P}$ or $\mathbf{y}_i \in \mathcal{Q}$. The probability density of some probabilistic model given some parameter θ can be denoted by $\mathbb{P}(\cdot \mid \theta)$. Then the problem of registering $\mathcal{Q}$ to $\mathcal{P}$ can be expressed as finding the best transformation $T : \mathbb{R}^D \mapsto \mathbb{R}^D$ such that $\mathcal{Q}$ is as similar to $\mathcal{P}$ as possible after applying the transformation to all points in $\mathcal{Q}$.

Therefore, given $\theta_{\mathcal{P}}$ the parameter that modelling $\mathcal{P}$, we have the following two optimization problems.

1. Using likelihood:

$$\hat{T} = \arg\max_T \frac{1}{N} \sum_{i=1}^{N} \mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}}). \tag{2.11}$$

2. Using K-L divergence:

$$\hat{T} = \arg\min_T \sum_{i=1}^{N} \mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}}) \log \frac{\mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}})}{\mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}})}. \tag{2.12}$$

Notice that

$$\begin{align}
\hat{T} &= \arg\max_T \frac{1}{N} \sum_{i=1}^{N} \mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}}), \tag{2.13}\\
&= \arg\max_T \frac{1}{N} \sum_{i=1}^{N} \mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}}), \tag{2.14}\\
&= \arg\max_T \left[\frac{1}{N} \sum_{i=1}^{N} \log \mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}}) - \frac{1}{N} \sum_{i=1}^{N} \log \mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}}) \right], \tag{2.15}\\
&= \arg\max_T \frac{1}{N} \sum_{i=1}^{N} \log \frac{\mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}})}{\mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}})}, \tag{2.16}\\
&= \arg\max_T \sum_{i=1}^{N} \mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}}) \log \frac{\mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}})}{\mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}})}, \tag{2.17}\\
&= \arg\min_T \sum_{i=1}^{N} \mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}}) \log \frac{\mathbb{P}(\mathbf{x}_i \mid \theta_{\mathcal{P}})}{\mathbb{P}(T(\mathbf{y}_i) \mid \theta_{\mathcal{P}})}, \tag{2.18}
\end{align}$$

thus via either likelihood or K-L divergence, the estimated optimal transformation $\hat{T}$ remains the same. □

Therefore, since calculating likelihood involves either using samples from the dataset or sampling from probability distributions, using K-L divergence can potentially reduce computational complexity if there are analytical forms for the probability density functions.

2.4 Wasserstein distance

Consider a metric space (M, d) that is also a Radon space. Let $p \in [1, +\infty]$ be a real number. Given two probability measures μ and ν on M, the Wasserstein

1-distance between μ and ν, denoted as $W_1(\mu,\nu)$, is defined as

$$W_1(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \int_{M\times M} d(x,y)\, d\gamma(x,y), \tag{2.19}$$

where $\Gamma(\mu,\nu)$ is the set of all couplings between μ and ν. A coupling γ is a joint probability measure on $M \times M$ such that its marginals on the first and second factors are μ and ν, respectively. Formally, a measure γ is a coupling if it satisfies

$$\int_M \gamma(x,y)\, \mathrm{d}y = \mu(x), \tag{2.20}$$

$$\int_M \gamma(x,y)\, \mathrm{d}x = \nu(y). \tag{2.21}$$

In the limit as p approaches infinity, the Wasserstein distance $W_\infty(\mu,\nu)$ is defined as

$$W_\infty(\mu,\nu) = \lim_{p\to\infty} W_p(\mu,\nu), \tag{2.22}$$

which can be interpreted as a supremum norm.

In the one-dimensional case and $d(x,y) := |x-y|$, the Wasserstein metric has the following closed form solution ([28]):

$$W_p(\mu,\nu) = \left(\int_0^1 |F_\mu^{-1}(t) - F_\nu^{-1}(t)|^p\, dt\right)^{\frac{1}{p}}, \tag{2.23}$$

where F_μ and F_ν are cumulative distribution functions for μ and ν respectively.

However, evaluating Wasserstein distance between multidimensional measures is numerically intractable in general [21]. In higher dimensional cases, an approach called sliced-Wasserstein distance (SWD) projects the problem to 1D and use the closed form solution. The concept of SWD involves projecting the target probability distributions, μ and v, in a specific direction, θ, on the unit sphere, resulting in two projected measures: $\pi^\theta\mu$ and $\pi^\theta v$. Specifically, the SWD averages Wasserstein distances of these projections over every possible θ. When $p \geq 1$, the sliced Wasserstein distance of order p can be expressed as:

$$SW_p(\mu,\nu) = \left(\int_{S^{n-1}} W_p^p(\pi^\theta\mu, \pi^\theta\nu) d\theta\right)^{\frac{1}{p}} \tag{2.24}$$

One way to approach Equation 2.19 is using Monte Carlo simulation for the integral, combining with gradient descent for the infimum. However, the computational complexity will become very high with this approach.

2.4.1 Kantorovich-Rubinstein Duality

Kantorovich-Rubinstein Duality [40] provides a better tool for calculating the high dimensional Wasserstein distance for $d(x, y) = ||x - y||_2$ and $p = 1$:

$$W_1(\mu, \nu) = \sup_{||h||_L \leq 1} \left[\underset{\mathbf{x}\sim\mu}{\mathbb{E}} [h(\mathbf{x})] - \underset{\mathbf{x}\sim\nu}{\mathbb{E}} [h(\mathbf{x})] \right], \tag{2.25}$$

where $||h||_L$ denotes the best Lipschitz constant for h and $||h||_L \leq 1$ implies h being 1-Lipschitz.

The problem now is to find an optimal 1-Lipschitz function that maximizes the objective. However, this remains infeasible because it is impossible to search over all 1-Lipschitz functions.

Below, we propose a method for approximating W_1 using a parameterized approximation of 1-Lipschitz functions.

Since 1-Lipschitz functions are continues, we can approximate them using sum of some basis functions $\phi_i(x)$:

$$h(\mathbf{x}) \approx \sum_i c_i \phi_i(\mathbf{x}). \tag{2.26}$$

Note that we have many options for choosing the basis functions. For example, when considering one-dimensional case, some of the common choices are shown in Table 2.2.

$\phi_i(x)$	Description
x^i	Taylor series
$sin(x)$ and/or $cos(x)$	Fourier series
Chebyshev polynomials	Chebyshev approximation
...	

Table 2.2: Some of the common choices for basis functions in 1D

In a multidimensional case, constructing the basis function is a more complex task. With the help of good properties of the sin function, we are able to construct the basis functions and impose constraints on the coefficients to guarantee that the approximating function belongs to a set of 1-Lipschitz functions. As the starting point, define

$$\phi_{ijk}(\mathbf{x}) = \sin(2\pi i x_1) \sin(2\pi j x_2) \sin(2\pi k x_3), \tag{2.27}$$

as the basis function that maps $\mathbb{R}^3$ to $[-1, 1]$, where i, j, k are indexes and (x_1, x_2, x_3) denotes the Euclidean coordinates of the 3D point $\mathbf{x}$. Note that the gradient of such $\phi_{ijk}(\mathbf{x})$ is bounded, i.e.,

$$\nabla\phi_{ijk}(\mathbf{x}) \in [-1,1] \times [-1,1] \times [-1,1].$$

Then, assume that we approximate the 1-Lipschitz function with a weighted sum of basis functions, i.e.,

$$\hat{h}(\mathbf{x}) := \sum_{i+j+k\leq s} c_{ijk} \sin(2\pi i x_1)\sin(2\pi j x_2)\sin(2\pi k x_3), \tag{2.28}$$

where $\sum_{i+j+k\leq s}$ denotes sum over all possible combinations of i, j, k where $i, j, k \in \mathbb{N}$, $s \in \mathbb{N}^+$ and $i + j + k \leq s$. By giving different s, we could have different numbers of terms in the sum. Although intuitively, the more terms there are, the better the approximation, this claim does not always hold and depends on the approach to encode the constraints of the optimization problem, which will be discussed in the following text.

The partial derivative is bounded as

$$\begin{aligned}\frac{\partial\hat{h}}{\partial x_1} &= \sum_{i+j+k\leq s} 2\pi i c_{ijk}\cos(2\pi i x_1)\sin(2\pi j x_2)\sin(2\pi k x_3), &(2.29)\\ &\leq \left[\max_{i+j+k\leq s} c_{ijk}\right]\cdot \sum_{i+j+k\leq s} 2\pi i\cos(2\pi i x_1)\sin(2\pi j x_2)\sin(2\pi k x_3). &(2.30)\end{aligned}$$

There are $\frac{1}{6}(s+1)(s+2)(s+3)$ terms in the sum. Notice that if $i > (s-2)$, then either j or k or both will be 0, resulting in making the term $2\pi i\cos(2\pi i x_1) sin(2\pi j x_2)\sin(2\pi k x_3)$ equals to 0. Combining with the fact that sine and cosine functions are bounded by $[-1, 1]$, we have

$$2\pi i\cos(2\pi i x_1) sin(2\pi j x_2)\sin(2\pi k x_3) \leq 2\pi(s-2). \tag{2.31}$$

Since there are totally $\frac{1}{6}(s+1)(s+2)(s+3)$ number of i, j, k combinations, continue on Equation 2.30, we have

$$\frac{\partial\hat{h}}{\partial x_1} \leq \left[\max_{i+j+k\leq s} c_{ijk}\right]\cdot\frac{1}{6}(s+1)(s+2)(s+3)\cdot 2\pi(s-2). \tag{2.32}$$

Rearranging Equation 2.32 we have

$$\frac{\partial\hat{h}}{\partial x_1} \leq \frac{\pi(s+1)(s+2)(s+3)(s-2)}{3}\cdot\max_{i+j+k\leq s} c_{ijk}. \tag{2.33}$$

With similar derivations we could also have

$$\frac{\partial \hat{h}}{\partial x_2} \leq \frac{\pi(s+1)(s+2)(s+3)(s-2)}{3} \cdot \max_{i+j+k\leq s} c_{ijk}, \tag{2.34}$$

$$\frac{\partial \hat{h}}{\partial x_3} \leq \frac{\pi(s+1)(s+2)(s+3)(s-2)}{3} \cdot \max_{i+j+k\leq s} c_{ijk}. \tag{2.35}$$

Note that with the definition of 1-Lipschitz function,

$$||\nabla \hat{h}(\mathbf{x})|| \leq 1 \tag{2.36}$$

should holds for every $\mathbf{x}$.

Thus, if we could make sure $\left|\frac{\partial \hat{h}}{\partial x_d}\right| \leq \frac{\sqrt{3}}{3}$ for $d = 1, 2, 3$, i.e.,

$$\max_{i+j+k\leq s} |c_{ijk}| \leq \frac{\sqrt{3}}{\pi(s+1)(s+2)(s+3)(s-2)}, \tag{2.37}$$

$$\Rightarrow |c_{ijk}| \leq \frac{\sqrt{3}}{\pi(s+1)(s+2)(s+3)(s-2)} \quad \forall i, j, k, \tag{2.38}$$

then we could ensure that $\hat{h}(\mathbf{x})$ being 1-Lipschitz and write

$$h_{||h||_L\leq 1}(\mathbf{x}) \approx \hat{h}(\mathbf{x}) = \sum_{i+j+k\leq s} c_{ijk} \sin(2\pi i x_1)\sin(2\pi j x_2)\sin(2\pi k x_3). \tag{2.39}$$

Now, recall the Kantorovich-Rubinstein Duality (Equation 2.25), let $p(\mathbf{x})$ and $q(\mathbf{x})$ denotes probability density functions (PDF) of μ and ν respectively, we can write

$$\mathbb{E}_{\mathbf{x}\sim p}[h(\mathbf{x})] = \int_{\mathbb{R}^3} p(\mathbf{x})h(\mathbf{x})\,\mathrm{d}V \tag{2.40}$$

$$\approx \int_{\mathbb{R}^3} p(\mathbf{x})\hat{h}(\mathbf{x})\,\mathrm{d}V \tag{2.41}$$

$$= \int_{\mathbb{R}^3} p(\mathbf{x}) \sum_{i+j+k\leq s} c_{ijk}\phi_{ijk}(\mathbf{x})\,\mathrm{d}V \tag{2.42}$$

$$= \sum_{i+j+k\leq s} c_{ijk} \int_{\mathbb{R}^3} p(\mathbf{x})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V, \tag{2.43}$$

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim q_T}[h(\mathbf{x})] &= \int_{\mathbb{R}^3} q(\mathbf{x})h(\mathbf{x})\,\mathrm{d}V && (2.44)\\
&\approx \int_{\mathbb{R}^3} q(\mathbf{x})\hat{h}(\mathbf{x})\,\mathrm{d}V && (2.45)\\
&= \int_{\mathbb{R}^3} q(\mathbf{x}) \sum_{i+j+k\le s} c_{ijk}\phi_{ijk}(\mathbf{x})\,\mathrm{d}V && (2.46)\\
&= \sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} q(\mathbf{x})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V. && (2.47)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim\mu}[h(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\nu}[h(\mathbf{x})] \approx & \sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} p(\mathbf{x})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V \\
& - \sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} q(\mathbf{x})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V, && (2.48)\\
= & \sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{x})\right]\phi_{ijk}(\mathbf{x})\,\mathrm{d}V. && (2.49)
\end{aligned}$$

Together with Equation 2.38 the constrains on coefficients, the problem in (2.25) now becomes

$$\begin{aligned}
& W(\mu,\nu) \approx \max_{\mathbf{c}} \left[\sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{x})\right]\phi_{ijk}(\mathbf{x})\,\mathrm{d}V \right], \\
& \text{subject to:} \quad |c_{ijk}| \le \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi} \quad \forall i,j,k,
\end{aligned} \qquad (2.50)$$

where $\mathbf{c}$ denotes a vector contains c_{ijk} with some certain order and the constraint of $i+j+k \le s$, i.e.

$$\mathbf{c} = [c_{0\ 0\ 0}, c_{0\ 0\ 1}, c_{0\ 0\ 2}, \ldots, c_{0\ 0\ s}, c_{0\ 1\ 0}, \ldots, c_{s\ 0\ 0}]^T. \qquad (2.51)$$

Assume that $p(\mathbf{x})$ and $q(\mathbf{x})$ are known, then optimization problem (2.50) is a linear programming, where c_{ijk} are decision variables and $\int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{x})\right]\phi_{ijk}(\mathbf{x})\,\mathrm{d}V$ are coefficients.

2.5 Gradient-based methods to optimize the transformation

2.5.1 Problem formulation

In our method, Euler angles are used to describe rigid body rotations, rotations around axes x, y and z with angle θ can be represented by the following rotation matrices:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{2.52}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{2.53}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.54}$$

Then, multiply them to get the complete rotation matrix around the origin (i.e. the point $[0,0,0]$):

$$\mathbf{R}(\alpha,\beta,\gamma) = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) \tag{2.55}$$

$$= \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \tag{2.56}$$

$$= \begin{bmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{bmatrix}. \tag{2.57}$$

For a given vector $\mathbf{x} \in \mathbb{R}^3$, using the right-hand rule to determine positive direction, first rotate it around the x-axis by an angle of α, then rotate around the y-axis by an angle of β, and finally rotate around the z-axis by an angle of γ can be represented by

$$\mathbf{R}(\alpha,\beta,\gamma)\mathbf{x}. \tag{2.58}$$

For simplicity, we denote $\mathbf{R}(\alpha, \beta, \gamma)$ and $\mathbf{R}(\hat{\alpha}, \hat{\beta}, \hat{\gamma})$ by $\mathbf{R}$ and $\hat{\mathbf{R}}$ respectively.

Furthermore, for a vector $\mathbf{x} \in \mathbb{R}^3$, translate it with the direction and distance of $\mathbf{t} \in \mathbb{R}^3$ can be represented by $\mathbf{x} + \mathbf{t}$.

Recall the two-step optimization problem (1.7) and (1.8). In our approach, the registration step (1.8) can be described as:

Given two probability density functions $p(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$ and $q(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$, we want to find the optimal rotation matrix $\hat{\mathbf{R}}$ and optimal translation vector $\hat{\mathbf{t}}$ such that

$$p(\mathbf{x}) = q(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})$$

holds for every $\mathbf{x} \in \mathbb{R}$. For simplicity, we also denote $q(\mathbf{R}\mathbf{x} + \mathbf{t})$ by $q_T(\mathbf{x})$.

Finally we have the following optimization problems to find optimal transformation:

1. **With K-L divergence:** recall the Equation 1.14 for K-L divergence between two probability distributions, we have the following optimization problem:

$$\operatorname*{arg\,min}_{\alpha,\beta,\gamma,\mathbf{t}} \int_{\mathbf{x}\in\mathbb{R}^3} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{R}(\alpha,\beta,\gamma)\mathbf{x} + \mathbf{t})} \, \mathrm{d}V \tag{2.59}$$

2. **With Wasserstein distance:** recall Equation 2.50 the approximation of Wasserstein distance, to find the optimal rotation degrees $\hat{\alpha}, \hat{\beta}, \hat{\gamma}$ and optimal translation vector $\hat{\mathbf{t}}$, we have to solve the following optimization problem:

$$\begin{aligned} &\operatorname*{arg\,min}_{\alpha,\beta,\gamma,\mathbf{t}} \; \max_{\mathbf{c}} \left[\sum_{i+j+k\le s} c_{ijk} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{R}(\alpha,\beta,\gamma)\mathbf{x} + \mathbf{t}) \right] \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V \right], \\ &\text{subject to:} \quad |c_{ijk}| \le \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi} \quad \forall i, j, k. \end{aligned} \tag{2.60}$$

2.5.2 The existence of gradients and the feasibility of optimization

Note that in the registration step, we assume that $p(\mathbf{x})$ and $q(\mathbf{x})$ are known and their analytical expressions exist. Suppose that $p(\mathbf{x})$ and $q(\mathbf{c})$ are smooth, calculating the divergence of the objective function in the optimization problem with K-L divergence (Equation 2.59) is intuitive. Therefore, it is not elaborated upon in the following. For the optimization problem with Wasserstein distance (Equation 2.60), we have the following two claims.

Claim 2.

Given $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$, the inner layer problem, i.e.

$$\begin{aligned} &\max_{\mathbf{c}} \left[\sum_{i+j+k\leq s} c_{ijk} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}}) \right] \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V \right], \\ &\text{subject to:} \quad |c_{ijk}| \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi} \quad \forall i,j,k, \end{aligned} \tag{2.61}$$

is a linear programming.

proof. Given $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$, the integrals are not related to $\mathbf{c}$. Thus, let

$$I_{ijk} = \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}}) \right] \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V, \tag{2.62}$$

and then the objective can be written as

$$\sum_{i+j+k\leq s} I_{ijk} c_{ijk}, \tag{2.63}$$

which is a linear function of c_{ijk}. The constraints can be expressed as

$$c_{ijk} \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi}, \tag{2.64}$$

$$-c_{ijk} \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi}, \tag{2.65}$$

for all c_{ijk}.

Therefore, the inner layer problem (2.61) is a linear programming. □

Claim 3.

Given $\hat{c}_{ijk}(\alpha, \beta, \gamma, \mathbf{t})$, the objective of outer layer problem, i.e.

$$\sum_{i+j+k\leq s} \hat{c}_{ijk} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{R}(\alpha,\beta,\gamma)\mathbf{x} + \mathbf{t}) \right] \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V, \tag{2.66}$$

is differentiable w.r.t α, β, γ and $\mathbf{t}$, and the gradient can be calculated accordingly.

proof. The optimization problem (2.60) can be abstracted as

$$\min_{\theta, \mathbf{c}} \mathcal{J}(\theta, \mathbf{c}) = \mathcal{J}(\theta, \mathbf{c}) + r_2(\mathbf{c}), \tag{2.67}$$

where θ denotes a tuple of transformation parameters $\mathbf{t}$, α, β and γ, $\mathcal{J}(\theta, \mathbf{c})$

corresponds to the negative of objective function in original problem 2.60 since the original problem contains a maximization problem, r_2 is some penalty function encoding the constraints $|c_{ijk}| \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi}\ \forall i,j,k$.

As proved by [41], for finite-dimensional separable optimization problems of the form

$$\min_{\theta,\mathbf{c}} J(\theta,\mathbf{c}) = \mathcal{J}(\theta,\mathbf{c}) + r_2(\mathbf{c}), \tag{2.68}$$

where $\mathcal{J}$ smoothly couples $(\theta,\mathbf{c})$ but may be non-convex, while r_2 encode additional constraints or regularizers. Consider $g(\theta) = \min_{\mathbf{c}} J(\theta,\mathbf{c})$, then $\nabla g(\theta) = \nabla_\theta \mathcal{J}(\theta,\mathbf{c})|_{\mathbf{c}=\arg\min_{\mathbf{c}} J(\theta,\mathbf{c})}$ holds if we can prove the objective satisfies two conditions below:

A1: $\nabla_\theta \mathcal{J}$ and $\nabla_{\mathbf{c}} \mathcal{J}$ exist and are Lipschitz-continuous for all $(\theta,\mathbf{c})$:

$$\|\nabla_\theta \mathcal{J}(\theta,\mathbf{c}) - \nabla_\theta \mathcal{J}(\theta',\mathbf{c})\| \leq L_{\theta\theta}\|\theta-\theta'\|, \tag{2.69}$$

$$\|\nabla_\theta \mathcal{J}(\theta,\mathbf{c}) - \nabla_\theta \mathcal{J}(\theta,\mathbf{c}')\| \leq L_{\theta\mathbf{c}}\|\mathbf{c}-\mathbf{c}'\|, \tag{2.70}$$

$$\|\nabla_{\mathbf{c}} \mathcal{J}(\theta,\mathbf{c}) - \nabla_{\mathbf{c}} \mathcal{J}(\theta',\mathbf{c})\| \leq L_{\mathbf{c}\theta}\|\theta-\theta'\|, \tag{2.71}$$

$$\|\nabla_{\mathbf{c}} \mathcal{J}(\theta,\mathbf{c}) - \nabla_{\mathbf{c}} \mathcal{J}(\theta,\mathbf{c}')\| \leq L_{\mathbf{cc}}\|\mathbf{c}-\mathbf{c}'\|; \tag{2.72}$$

A2: $J(\theta,\mathbf{c})$ is μ-strongly convex in $\mathbf{c}$ for all θ.

Since $\mathbf{R}(\alpha,\beta,\gamma)$ is a smooth projection maps $\mathbb{R}\times\mathbb{R}\times\mathbb{R}$ to $\mathbb{R}^{3\times3}$, we can consider $\mathbf{R}$ as a whole and temporarily ignore α,β, and γ. In the following we decompress $\mathbf{R}$ and $\mathbf{t}$ from θ in order to let the notations in formulas be consistent with the previous sections. For example, when discussing $\nabla_\theta\mathcal{J}$ we will discuss $\nabla_{\mathbf{R}}\mathcal{J}$ and $\nabla_{\mathbf{t}}\mathcal{J}$ separately.

Beginning with $\nabla_\theta\mathcal{J}$, i.e., $\frac{\partial\mathcal{J}}{\partial\mathbf{R}}$ and $\frac{\partial\mathcal{J}}{\partial\mathbf{t}}$:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{R}} = \frac{\partial}{\partial\mathbf{R}}\left[\sum_{i+j+k\leq s} c_{ijk}\cdot\int_{\mathbb{R}^3} -q(\mathbf{R}\mathbf{x}+\mathbf{t})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V\right] \tag{2.73}$$

Since the integral in Equation 2.73 is on $\mathbf{x}$, we can directly interchange the partial derivative and integral, i.e.,

$$\frac{\partial\mathcal{J}}{\partial\mathbf{R}} = \sum_{i+j+k\leq s} c_{ijk}\cdot\int_{\mathbb{R}^3} -\frac{\partial}{\partial\mathbf{R}} q(\mathbf{R}\mathbf{x}+\mathbf{t})\phi_{ijk}(\mathbf{x})\,\mathrm{d}V. \tag{2.74}$$

Equation 2.74 is linear w.r.t $\mathbf{c}$, therefore it is Lipschitz-continuous for all $\mathbf{c}$. With similar derivations, we could also prove that $\frac{\partial\mathcal{J}}{\partial\mathbf{t}}$ is linear w.r.t $\mathbf{c}$. Hence $\nabla_\theta\mathcal{J}$ is Lipschitz-continuous w.r.t $\mathbf{c}$, and thus inequality (2.70) holds.

To prove $\nabla_\theta\mathcal{J}$ is Lipschitz-continuous for all θ, recall an approach to prove a function being Lipschitz-continuous is to prove its derivative is bounded.

Recall Equation 2.74, notice that in order to check whether the derivative of $\nabla_\theta \mathcal{J}$ is bounded, the key is to check whether the derivatives of $\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{R}}$ and $\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{t}}$ are bounded. Therefore, at first we can write the formula for $\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{R}}$ in detail:

$$\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{R}} = \nabla_{\mathbf{Rx}+\mathbf{t}} q(\mathbf{Rx}+\mathbf{t}) \cdot \nabla_{\mathbf{R}} (\mathbf{Rx}+\mathbf{t})^T, \tag{2.75}$$

$$= \left[-\frac{1}{nh^2} \sum_{i=1}^{n} \frac{\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i}{(2\pi h^2)^{3/2}} \exp\left(-\frac{1}{2h^2} \|\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i\|^2 \right) \right] \cdot \mathbf{x}^T. \tag{2.76}$$

Then we need to explore the properties of its derivatives w.r.t $\mathbf{R}$ and $\mathbf{t}$ respectively. First, its derivative w.r.t $\mathbf{R}$ is

$$\begin{aligned} \frac{\partial^2 q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{R}^2} = & -\frac{1}{nh^2} \sum_{i=1}^{n} \left[\frac{\mathbf{x}(\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i)^T}{(2\pi h^2)^{3/2}} \exp\left(-\frac{1}{2h^2} \|\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i\|^2 \right) \right. \\ & \left. - \frac{(\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i)(\mathbf{x})^T \mathbf{x}(\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i)^T}{h^4 (2\pi h^2)^{3/2}} \exp\left(-\frac{1}{2h^2} \|\mathbf{Rx}+\mathbf{t}-\mathbf{x}_i\|^2 \right) \right] . \end{aligned} \tag{2.77}$$

The properties of exp(·) terms make $\frac{\partial^2 q}{\partial R^2}$ bounded w.r.t $\mathbf{x}$ for all $\mathbf{R}$. Hence the integral of it over $\mathbf{x}$ is bounded, i.e.

$$\sum_{i+j+k \le s} c_{ijk} \cdot \int_{\mathbb{R}^3} -\frac{\partial^2}{\partial \mathbf{R}^2} [q(\mathbf{Rx}+\mathbf{t})] \, \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V \tag{2.78}$$

is bounded. With similar derivations, we could prove that $\frac{\partial^2 q}{\partial \mathbf{R} \partial \mathbf{t}}$ is also bounded. Therefore, $\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{R}}$ is Lipschitz-continuous w.r.t $\mathbf{R}$ and $\mathbf{t}$. Also with similar derivations we could prove that $\frac{\partial q(\mathbf{Rx}+\mathbf{t})}{\partial \mathbf{t}}$ is Lipschitz-continuous w.r.t $\mathbf{R}$ and $\mathbf{t}$. Thus, we can conclude that $\nabla_\theta \mathcal{J}$ is Lipschitz-continuous w.r.t θ. Hence, inequality (2.69) holds.

To prove $\nabla_{\mathbf{c}} \mathcal{J}$ is Lipschitz-continuous:

$$\nabla_{\mathbf{c}} \mathcal{J} = \begin{bmatrix} \int_{\mathbb{R}^3} [p(\mathbf{x}) - q(\mathbf{Rx}+\mathbf{t})] \, \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V \\ \vdots \\ \int_{\mathbb{R}^3} [p(\mathbf{x}) - q(\mathbf{Rx}+\mathbf{t})] \, \phi_{ijk}(\mathbf{x}) \, \mathrm{d}V \end{bmatrix} \tag{2.79}$$

It is obviously Lipschitz-continuous w.r.t $\mathbf{c}$ since $\mathbf{c}$ is eliminated. Therefore, inequality (2.72) holds.

Furthermore, recall the above derivations, we already know that $\frac{\partial q}{\partial \mathbf{R}}$ and

$\frac{\partial q}{\partial \mathbf{t}}$ are bounded and tend to $\mathbf{0}$ when $\|\mathbf{x}\| \to \infty$, hence

$$\frac{\partial}{\partial \mathbf{R}} \int_{\mathbb{R}^3} \left[p(\mathbf{x}) - q(\mathbf{R}\mathbf{x} + \mathbf{t})\right] \phi_{ijk}(\mathbf{x}) \,\mathrm{d}V = -\int_{\mathbb{R}^3} \frac{\partial}{\partial \mathbf{R}} q(\mathbf{R}\mathbf{x} + \mathbf{t}) \phi_{ijk}(\mathbf{x}) \,\mathrm{d}V$$

is bounded w.r.t $\mathbf{R}$. Again, with similar derivations we could prove that it is bounded w.r.t $\mathbf{t}$. Hence, $\nabla_{\mathbf{c}}\mathcal{J}$ is Lipschitz-continuous w.r.t θ and thus inequality (2.71) holds.

So far, we have already given the proof to four inequalities in **A1**. Recall the condition **A2**. From the above derivations, known that the Hessian of $\mathcal{J}$ w.r.t $\mathbf{c}$ is $\mathbf{0}$, since the first derivatives (Equation 2.79) are already independent on $\mathbf{c}$. Hence, condition **A2** is satisfied by ensuring $r_2(\mathbf{c})$, which encodes constraints on $\mathbf{c}$, being μ-strongly convex. $\square$

Therefore, with the help of Claim 2 and Claim 3, we are able to update $\mathbf{c}$ and transformation parameters iteratively and separately: given initial guess on transformation, which is denoted by θ_0, we could solve the linear programming problem

$$\begin{aligned} \mathbf{c}_0 &= \arg\max_{\mathbf{c}} \mathcal{J}(\theta_0, \mathbf{c}), \\ \text{subject to:} \quad |c_{ijk}| &\leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi} \quad \forall i, j, k. \end{aligned} \tag{2.80}$$

Then, with $\mathbf{c}_0$ we could update the transformation

$$\theta_1 = \arg\max_{\theta} \mathcal{J}(\theta, \mathbf{c}_0). \tag{2.81}$$

In subsequent iterations, we continue this process to obtain $\mathbf{c}_i$ and θ_{i+1} in the i-th iteration.

3. Algorithms and implementation

In this chapter, algorithms from modelling point clouds to calculating optimal transformation will be introduced and discussed in detail. An algorithm for selecting the number of components of GMM will be given, as well as complete algorithms for modelling point clouds using GMM and KDE to obtain the corresponding density functions. Additionally, an algorithm, which is based on Wasserstein distance and LBFGS, outputs optimal transformation on the moving point cloud given density functions representing two point clouds will be proposed. A full procedure for solving the point cloud registration problem with proposed probabilistic methods will be shown in the end. The procedure begins with two point clouds and ends with an optimal transformation of the moving point cloud.

3.1 Modelling point clouds

Given a set of points, fitting GMM with the E-M algorithm is a well-studied and reliable approach. [42] provides detailed information on the Gaussian mixture model and EM algorithm, as well as necessary background knowledge. It is worth mentioning the algorithm used to determine the optimal number of components. As shown in Algorithm 1, a linear search on all possible numbers of components is performed. Note that since two point clouds obtained from the same object share similar properties in terms of geometry, we could run the algorithm once on one point cloud and directly apply the result when modelling another point cloud. Where silhouette score is the average of silhouette coefficient (Equation 2.5) of all samples. Furthermore, this algorithm includes GMM fitting. Therefore, once the optimal number of components is determined, the corresponding optimal GMM is also established.

Algorithm 1 Estimate number of components of GMM using Silhouette score

```
1: Input: Point cloud $\mathcal{P}$; Boundaries of $K$: $K_{\min}$ and $K_{\max}$
2: Output: $K_{\text{opt}}$ (optimal number of components)
3: $K_{\text{opt}} \leftarrow 0$
4: $sc_{\text{opt}} \leftarrow -1$
5: for $K \leftarrow K_{\min}$ to $K_{\max}$ do ▷ Update GMM$_{\mathcal{P}}$ by performing GMM fitting with $K$ components
6:     GMM$_{\mathcal{P}} \leftarrow$ fit GMM to $\mathcal{P}$ with $K$ components
7:     $sc \leftarrow$ Silhouette score of GMM$_{\mathcal{P}}$
8:     if $sc > sc_{\text{opt}}$ then
9:         $sc_{\text{opt}} \leftarrow sc$
10:        $K_{\text{opt}} \leftarrow K$
11: return $K_{\text{opt}}$
```

Compared to the Gaussian mixture model, the implementation of kernel density estimation is more straightforward. Given a set of points, the density function of KDE with Gaussian kernel can be directly obtained from the combination of Equation 1.12, Equation 2.7 and Equation 2.10. Specifically, if given a 3-dimensional point cloud $\mathcal{P}$, the density function is $\hat{p}$ as below:

$$\hat{p}(\mathbf{x}) = \frac{1}{nh^3} \sum_{\mathbf{x}_i \in \mathcal{P}} \frac{1}{(2\pi)^{\frac{3}{2}}} \exp\left(-\frac{1}{2} \left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|_2^2 \right), \tag{3.1}$$

where n denotes number of points in the point cloud, $h = 1.06n^{-\frac{1}{5}}\hat{\Sigma}$ and $\hat{\Sigma}$ is the average of standard derivation of each dimension of $\mathcal{P}$.

3.2 Finding optimal transformation

Since in subsection 2.5.2 we have proved that the approximate Wasserstein distance is differentiable w.r.t transformation and actually compute it, we are able to design a gradient-based algorithm to find optimal transformation iteratively. A simple reference algorithm is provided in Algorithm 2. When using K-L divergence, the calculation of gradient is intuitive and easy to implement with third-party packages such as autograd from PyTorch. Therefore, the algorithm for computing optimal transformation with K-L divergence and gradient-based method is not elaborated further in the thesis. In practice, we used LBFGS (Algorithm 3) to find optimal transformation. Again, some modifications should be made when using Wasserstein distance because of its two-layer nature. $\hat{\mathbf{c}}$ should be updated inside the loop. The complete algorithm of LBFGS with Wasserstein distance could be found in Algorithm 4.

Algorithm 2 Algorithm to find optimal rotation and translation with approximate Wasserstein distance

```
1: **Input:** $p, q$ (probability density functions);
              $s$ (upper bound for $i + j + k$);
              $\alpha_0, \beta_0, \gamma_0, \mathbf{t}_0$ (initial values for rotation degrees and translation);
              $max\_iter$ (maximum number of iterations);
              $\epsilon$ (convergence threshold for Wasserstein distance);
              $\mu$ (step size);
2: **Output:** $\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\mathbf{t}}$ (optimal values for rotation degrees and translation)
3: $n \leftarrow 0$
4: while $n < max\_iter$ do
5:     for $i \leftarrow 0$ to $s$ do
6:         for $j \leftarrow 0$ to $s - i$ do
7:             for $k \leftarrow 0$ to $s - i - j$ do
8:                 $I_{ijk} \leftarrow \int [p(\mathbf{x}) - q(\mathbf{R}(\alpha_n, \beta_n, \gamma_n)\mathbf{x} + \mathbf{t}_n)] \phi_{ijk}(\mathbf{x})\, dV$ ▷ Loop over all possible combinations of $i, j, k$
9:     $\mathbf{\Gamma} \leftarrow [I_{000}, I_{001}, \ldots, I_{ijk}, \ldots, I_{s00}]$ ▷ Construct vector $\mathbf{\Gamma}$ from $I_{ijk}$ with a certain order
10:    $\hat{\mathbf{c}} \leftarrow \arg\max_{\mathbf{c}} \mathbf{\Gamma c}$ subject to $|c_{ijk}| \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi}$ $\forall i, j, k$ ▷ Solve the inner layer problem (linear programming)
11:    define $\mathcal{J}_{\hat{\mathbf{c}}}(\alpha, \beta, \gamma, \mathbf{t}) = \sum_{i+j+k \leq s} \hat{c}_{ijk} \int_{\mathbb{R}^3} [p(\mathbf{x}) - q(\mathbf{R}(\alpha, \beta, \gamma)\mathbf{x} + \mathbf{t})] \phi_{ijk}(\mathbf{x})\, \mathrm{d}V$ ▷ Define the objective for outer layer problem
12:    $d_n \leftarrow \mathcal{J}_{\hat{\mathbf{c}}}(\alpha_n, \beta_n, \gamma_n, \mathbf{t}_n)$
13:    if $n == 0$ or $|d_n - d_{n-1}| > \epsilon$ then    ▷ Solve the outer layer problem
14:        $n \leftarrow n + 1$
15:        $\mathbf{g} \leftarrow \nabla \mathcal{J}_{\hat{\mathbf{c}}}(\alpha_{n-1}, \beta_{n-1}, \gamma_{n-1}, \mathbf{t}_{n-1})$    ▷ Compute the gradient
16:        $[\alpha_n, \beta_n, \gamma_n, \mathbf{t}_n] \leftarrow [\alpha_{n-1}, \beta_{n-1}, \gamma_{n-1}, \mathbf{t}_{n-1}] - \mu \mathbf{g}$
17:    else
18:        break    ▷ If converge, break the loop and return optimal values
19: return $\alpha_n, \beta_n, \gamma_n, \mathbf{t}_n$
```

Note that in the algorithms,

- $\mathbf{\Gamma}$ and $\mathbf{c}$ are vectors containing I_{ijk} and c_{ijk} respectively, with some certain order and the constraint of $i + j + k \leq s$, e.g.,

$$\mathbf{\Gamma} = [I_{0\ 0\ 0}, I_{0\ 0\ 1}, I_{0\ 0\ 2}, \ldots, I_{0\ 0\ s}, I_{0\ 1\ 0}, \ldots, I_{s\ 0\ 0}], \tag{3.2}$$

$$\mathbf{c} = [c_{0\ 0\ 0}, c_{0\ 0\ 1}, c_{0\ 0\ 2}, \ldots, c_{0\ 0\ s}, c_{0\ 1\ 0}, \ldots, c_{s\ 0\ 0}]^T; \tag{3.3}$$

- I_6 denotes an identity matrix of size 6;
- Since α, β and γ are scalars and $\mathbf{t}$ is a 3×1 vector, the independent variables of $\mathcal{J}$ are a mixture of scalars and a vector. Therefore, we define

the gradient of $\mathcal{J}$ as

$$\nabla\mathcal{J}(\alpha,\beta,\gamma,\mathbf{t}) := \begin{bmatrix} \frac{\partial\mathcal{J}}{\partial\alpha} \\ \frac{\partial\mathcal{J}}{\partial\beta} \\ \frac{\partial\mathcal{J}}{\partial\gamma} \\ \nabla_{\mathbf{t}}\mathcal{J} \end{bmatrix} := \begin{bmatrix} \frac{\partial\mathcal{J}}{\partial\alpha} \\ \frac{\partial\mathcal{J}}{\partial\beta} \\ \frac{\partial\mathcal{J}}{\partial\gamma} \\ \frac{\partial\mathcal{J}}{\partial t_x} \\ \frac{\partial\mathcal{J}}{\partial t_y} \\ \frac{\partial\mathcal{J}}{\partial t_z} \end{bmatrix}. \tag{3.4}$$

- To compute the integral, a "quasi trapezoidal rule" is applied, i.e.,

$$\int_{\mathbb{R}^3} f(\mathbf{x})\,\mathrm{d}V = \int_D f([x,y,z]^T)\,\mathrm{d}V = \Delta x\Delta y\Delta z \sum_{i=0}^{N_x}\sum_{j=0}^{N_y}\sum_{k=0}^{N_z} f([x_i,y_j,z_k]^T), \tag{3.5}$$

  where
  - $D = \{(x,y,z) \mid x\in[x_{\min},x_{\max}], y\in[y_{\min},y_{\max}], z\in[z_{\min},z_{\max}]\}$ being a proper cubic integral area such that f is approximately 0 on the bonds of D and outside D;
  - $x_0 = x_{\min},\ x_{N_x} = x_{\max},\ y_0 = y_{\min},\ y_{N_y} = y_{\max},\ z_0 = z_{\min},\ z_{N_z} = z_{\max}$;
  - $x_{i+1} - x_i = \Delta x\ \forall i \in [N_x - 1]$, $y_{j+1} - y_j = \Delta y\ \forall j \in [N_y - 1]$, $z_{k+1} - z_k = \Delta z\ \forall k \in [N_z - 1]$.

Additionally, due to the limitation of floating-point arithmetic in computers, in practice, a tiny amount will be recognized as 0. Then, when using K-L divergence, this will raise undefined errors, e.g., log(0) and $\frac{\cdot}{0}$ when $p(\mathbf{x}) \to 0$ or $q(\mathbf{x}) \to 0$ for some $\mathbf{x}$. Therefore, denote machine epsilon by *eps*, we have the practical version of Equation 2.59 as follow:

$$\operatorname*{arg\,min}_{\alpha,\beta,\gamma,\mathbf{t}} \int_{\mathbf{x}\in\mathbb{R}^3} p(\mathbf{x}) \log\max\left[\frac{p(\mathbf{x})}{\max\left[q(\mathbf{R}(\alpha,\beta,\gamma)\mathbf{x}+\mathbf{t}), eps\right]}, eps\right] \mathrm{d}V. \tag{3.6}$$

Note that with this approach, gradient may be 0 when two point clouds are far from each other.

3.3 Full procedure

In this section, a full procedure for solving the point cloud registration problem with proposed probabilistic methods will be shown. Recall formulas (1.7) and (1.8), there are two major steps to solve the problem. For the fitting step, Gaussian Mixture Model and Kernel Density Estimation are introduced and discussed in detail in section 2.1 and section 2.2 respectively. For registration step, K-L divergence and Wasserstein distance are introduced as criteria for judging whether two point clouds are close enough. The capability to use gradient-based methods is discussed in section 2.5. Therefore, combining all of them, we have the following probabilistic method for the point cloud registration problem:

- First, in fitting step: given two point clouds $\mathcal{P}$ and $\mathcal{Q}$ in the form of two sets of 3D points, we use either GMM or KDE to model the point cloud and get two probability density functions p and q corresponding to those two point clouds.
  - GMM: performing on one point cloud, use the method outlined in Algorithm 1 along with the Expectation-Maximization algorithm, detailed in subsection 2.1.1, to determine the optimal number of components. Corresponding optimal GMM will be also established during this process, which yields a probability density function. Perform the same procedure on another point cloud so that we can get two probability density functions, namely p and q representing $\mathcal{P}$ and $\mathcal{Q}$, respectively.
  - KDE: first, calculate the average of the standard derivation of each dimension of one point cloud, namely $\hat{\Sigma}$. Then, substitute data points to the formula in Equation 3.1 with $h = 1.06n^{-\frac{1}{5}}\hat{\Sigma}$. Perform the same procedure on another point cloud. This results in two probability density functions, denoted as p and q.
- Then, in the registration step, given two probability density functions p and q, perform gradient-based methods to optimize the transformation.
  - K-L divergence: solve the optimization problem (2.59) with gradient-based methods. For example, the LBFGS algorithm (see Algorithm 3 for reference). Output of the algorithm will be the optimal transformation on the moving point cloud $\mathcal{Q}$ that aligns it with the target point cloud $\mathcal{P}$.
  - Approximate Wasserstein distance: solve the optimization problem (2.60) using Algorithm 2, specifically LBFGS algorithm can be applied (see Algorithm 4 for reference). Output of the algorithm will be the optimal transformation on the moving point cloud $\mathcal{Q}$ that aligns it with the target point cloud $\mathcal{P}$.
- Finally, after obtaining the optimal transformation, apply this transformation to the moving point cloud $\mathcal{Q}$. Then, establish the correspon-

dences between points in two point clouds using the nearest neighbour method. In other words, given a set

$$S = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in \mathcal{P} \wedge \mathbf{y} \in \mathcal{Q}\}, \tag{3.7}$$

then, every element in the following set represents a pair of points between two point clouds,

$$S_{\text{co}} = \left\{ (\mathbf{x}, \mathbf{y}) \in S \mid \mathbf{x} = \operatorname*{arg\,min}_{\mathbf{x}' \in \mathcal{P}} d(\mathbf{x}', \mathbf{y}) \wedge \mathbf{y} = \operatorname*{arg\,min}_{\mathbf{y}' \in \mathcal{Q}} d(\mathbf{x}, \mathbf{y}') \right\}. \tag{3.8}$$

Note that in practice, due to noise, sampling errors, inaccurate transformation output by the registration algorithm, and the varying number of points in different point clouds, a point in one point cloud does not always find its corresponding point in another point cloud.

4. Experiments

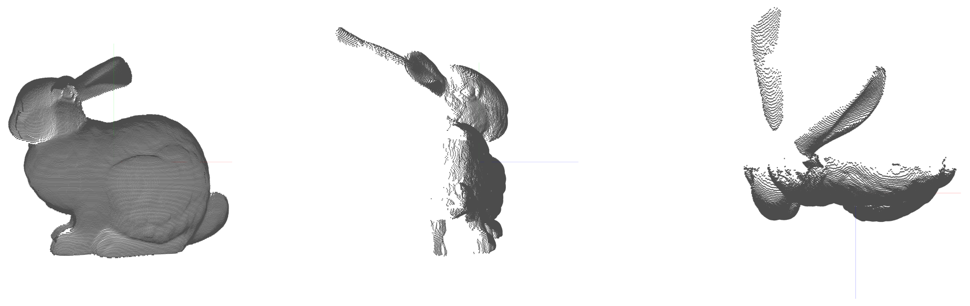
In this chapter, experiments will be conducted focusing on the fitting step and the registration step, respectively. First, we will model the point clouds using Gaussian Mixture Models and Kernel Density Estimation to compare their differences and determine which model is more appropriate regarding to a given dataset. Subsequently, we will apply the Kullback-Leibler divergence and the approximate Wasserstein distance proposed in this thesis to the registration step, starting with one degree of freedom in transformation, progressing to two degrees and six degrees of freedom. These experiments aim to ascertain whether our approximate Wasserstein distance algorithm can provide correct gradients with respect to transformations and whether it is applicable for point cloud registration.

4.1 Dataset

The experiment dataset comes from "Stanford Bunny" [43] with scan angle 000. This dataset has been widely used in computer graphics and computational geometry research, making it a standard benchmark for testing and comparing point cloud processing algorithms, including point cloud registration algorithms. The dataset includes complex surface details with rich geometric features, which are challenging to process. Moreover, its asymmetry enables a better understanding of the behaviour of the proposed algorithms.

Figure 4.1 shows the Stanford Bunny point cloud in front, left and top views. In the figures, the red, green, and blue lines indicate the positive direction of the x-axis, y-axis and z-axis respectively. For example, in the front view (Figure 4.1a), the positive x-axis direction is pointing right, the positive y-axis is pointing up, and the positive z-axis is pointing out of the page ($\odot$). There are totally 40256 points in the point cloud, and the dimensions are

$$\begin{aligned} x &\in [-0.071, 0.085]; \\ y &\in [-0.061, 0.091]; \\ z &\in [-0.094, 0.023]. \end{aligned} \tag{4.1}$$



(a) Front **(b)** Left **(c)** Top

Figure 4.1: Stanford Bunny

4.2 Probability models

In this section, experiments on modelling point clouds with Gaussian Mixture Model and kernel density estimation will be conducted and the corresponding 3D probability density functions will be displayed in the form of slices. This allows us to intuitively compare differences between the two probabilistic models and point out that KDE might be a more suitable probabilistic model.

4.2.1 Gaussian Mixture Model

Algorithm 1 provides a linear search algorithm to find an optimal number of components for GMM. In this experiment, the Stanford Bunny point cloud is down-sampled to 1408 points with the voxel grid downsampling method in order to speed up the computation. This means that the point cloud is divided into a 3D grid of voxels, and points that fall into the same voxel are approximated by a single point, typically the centroid of the points in that voxel. Silhouette scores (Equation 2.6) of different numbers of components when modelling the point cloud with GMM are shown in Figure 4.2, and the output is $K_{\text{opt}} = 229$.

After applying E-M algorithm, we are able to obtain GMM probability density functions. Figure 4.3 shows six heat-map slices of the estimated probability density function. The slices are parallel to the xy-plane. It can be observed that the Gaussian mixture model captures certain features of the point cloud, such as the shape of the bunny's body (Figure 4.3b and Figure 4.3c) and its long ears (Figure 4.3e, Figure 4.3d and Figure 4.3f). However, due to the relatively small estimated variance of each component and the tendency of the E-M algorithm to concentrate and cluster components in high-density areas,

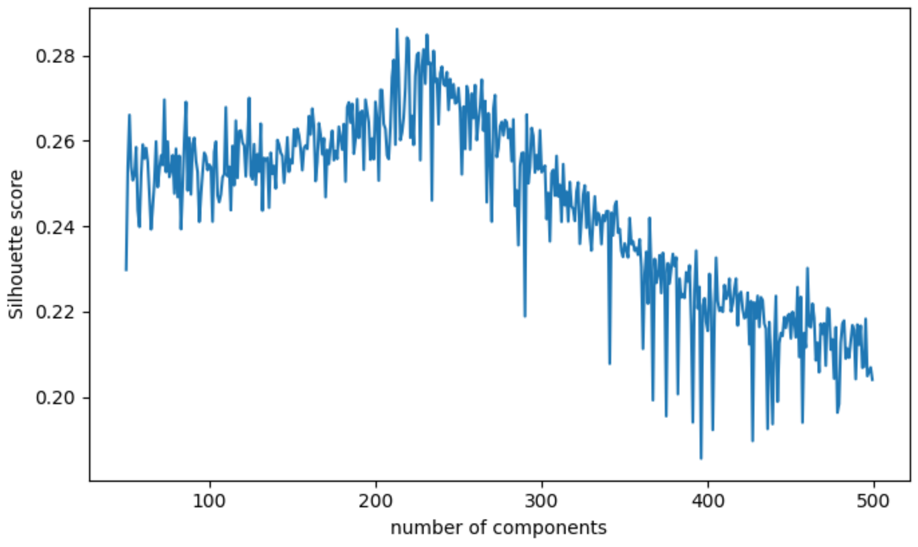
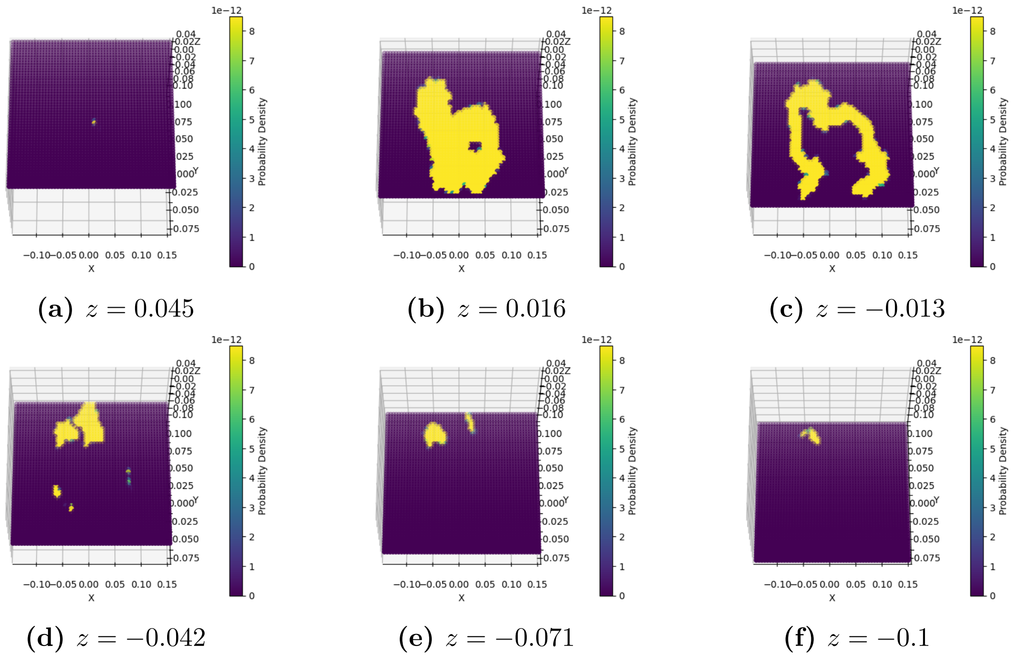


Figure 4.2: Silhouette scores of different number of components

the resulting probability density function exhibits significant variability around the bunny's surface. This is shown in the image as sharp and clear boundaries, which may affect the registration step. For instance, it might perform worse when handling point clouds with substantial random noise.



(a) $z = 0.045$ **(b)** $z = 0.016$ **(c)** $z = -0.013$

(d) $z = -0.042$ **(e)** $z = -0.071$ **(f)** $z = -0.1$

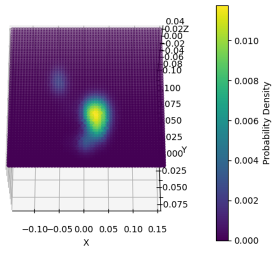
Figure 4.3: Slices of GMM probability density function

4.2.2 Kernel density estimation

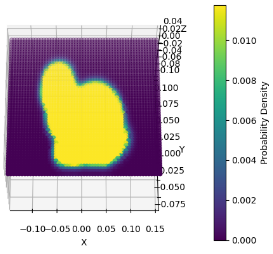
As stated in section 3.1 to estimate the bandwidth for KDE, the standard derivations of each dimension of the 1408-points point cloud (as the downsampled point cloud from the previous experiment) are 0.04153521, 0.04034828 and 0.02181502 respectively. Therefore, their average, $\hat{\Sigma} = 0.03456617$. Then, we have the bandwidth

$$h = 1.06n^{-\frac{1}{5}}\hat{\Sigma} = 0.0085948. \tag{4.2}$$

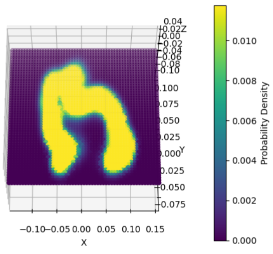
Applying Equation 3.1, we can obtain a KDE probability density function modelling the point cloud. Figure 4.4 shows slices of the probability density function. The features such as the shape of the bunny's body (Figure 4.4b and Figure 4.4c) and its ears (Figure 4.4d, Figure 4.4e and Figure 4.4f) are also visible. In contrast to GMM (subsection 4.2.1), KDE provides a smoother probability density function, which may help the registration step.



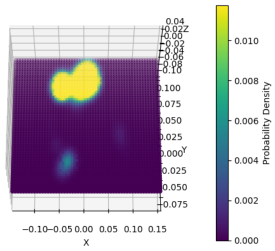
(a) $z = 0.045$



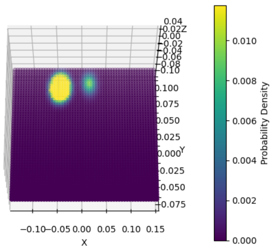
(b) $z = 0.016$



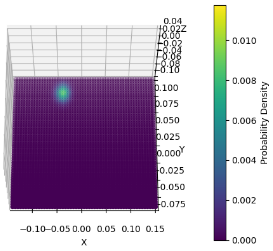
(c) $z = -0.013$



(d) $z = -0.042$



(e) $z = -0.071$



(f) $z = -0.1$

Figure 4.4: Slices of KDE probability density function

4.3 Approximating Wasserstein distance with different number of basis functions

Recall Equation 2.28, by changing s, the number of basis equations can be changed. Therefore, in this experiment, the effect of the choice of s on the

Wasserstein distance approximation will be explored. In the experiments, the Stanford Bunny point cloud is down-sampled to 23 points with the voxel grid downsampling method. Then, the 23-point point cloud is duplicated, and one of them is used as the fixed point cloud (i.e., target point cloud) and the other as the moving point cloud (i.e., source point cloud). KDE will be applied to model these two point clouds in the experiments.

In the first experiment, the moving point cloud is translated along x-axis, y-axis, and z-axis respectively. For example, when translating along x-axis, the moving point cloud is translated to coordinate $(x, 0, 0)$ where $x \in [-4, 4]$. The corresponding approximate Wasserstein distances with different s are shown in Figure 4.5a, Figure 4.5c, and Figure 4.5e. In the second experiment, the moving point cloud is rotated around its z-axis, y-axis, and z-axis respectively. The corresponding approximate Wasserstein distances with different s are shown in Figure 4.5b, Figure 4.5d, and Figure 4.5f. Intuitively, the more basis functions, the better to approximate all 1-Lipschitz, and thus the closer to the supremum in Equation 2.25. However, from the results of the experiments, we observe that the statement does not hold for some cases. As s increases, the curve becomes higher and higher, but when s is greater than some certain value, the curve gradually decreases. The reason for that comes from the imperfect constraints of the linear programming scheme. The constraints (i.e., formula 2.38) are sufficient but not necessary conditions for $\hat{h}$ to be 1-Lipschitz. In other words, suppose the best Lipschitz constant of $\hat{h}$ is k', then constructing $\hat{h}$ with such constraints will make

$$k' < 1, \tag{4.3}$$

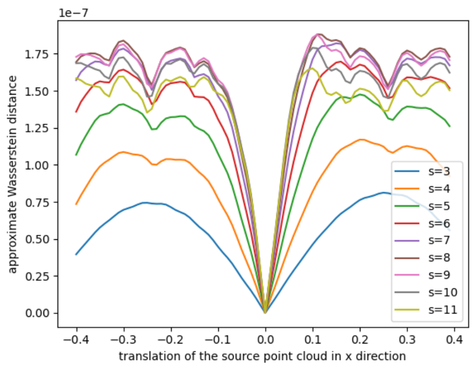
and thus it can not approximate all 1-Lipschitz functions in principle. [44] states that in Equation 2.25, if consider h to be k-Lipschitz instead of 1-Lipschitz, then the supremum of the formula will equals to k times Wasserstein distance, i.e.,

$$kW_1(\mu, \nu) = \sup_{||h||_L \leq k} \left[\mathop{\mathbb{E}}_{\mathbf{x} \sim \mu} [h(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{x} \sim \nu} [h(\mathbf{x})] \right]. \tag{4.4}$$

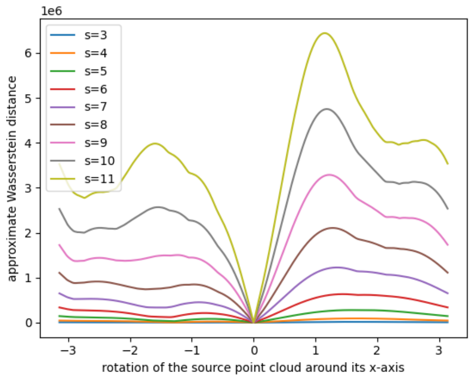
Therefore, denotes the optimal solution of linear programming (2.61) by $\hat{W}_1$, then we have

$$\hat{W}_1 \approx k' W_1 < W_1. \tag{4.5}$$

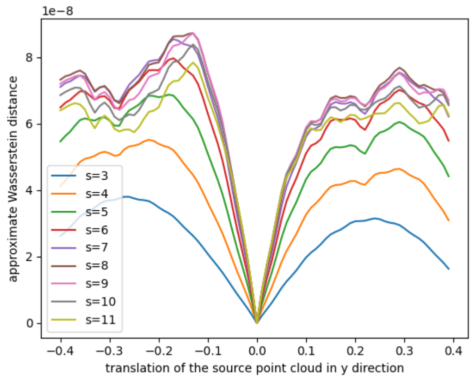
Note that k' is an unknown function of s, and it is not guaranteed to increase as s increases. Therefore, as shown in Figure 4.5, the curve may decrease when s increases. However, the optimal solution remains the same as the real Wasserstein distance, and Figure 4.5 shows that the approximation scheme can provide correct descent direction. Therefore, choosing different s will not make a huge impact on optimization. Considering the computation speed, a small but not too small value of s should be selected. Unless otherwise specified, $s = 6$ is selected by default in subsequent experiments.



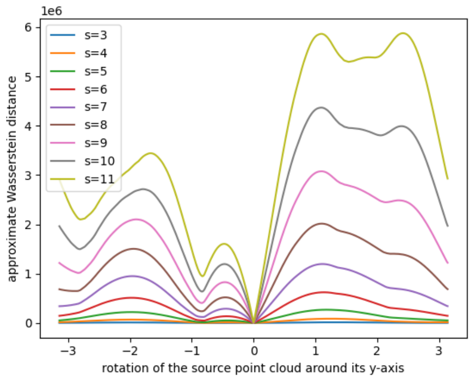
(a) Translate in x direction



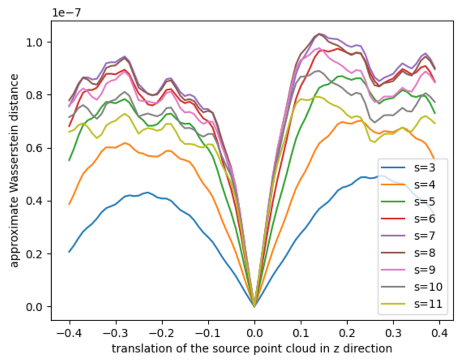
(b) Rotate around x-axis



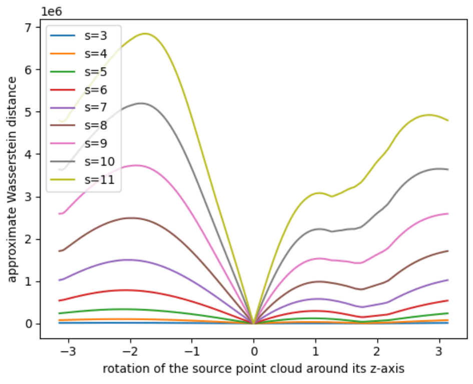
(c) Translate in y direction



(d) Rotate around y-axis



(e) Translate in z direction

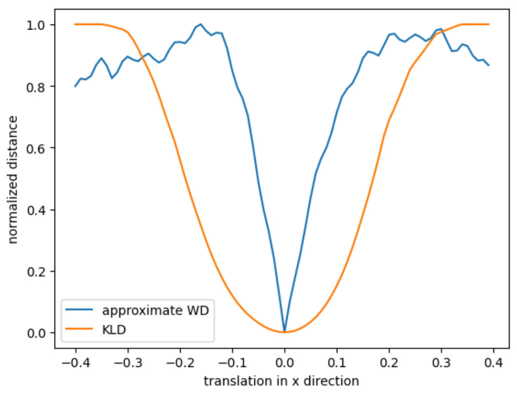


(f) Rotate around z-axis

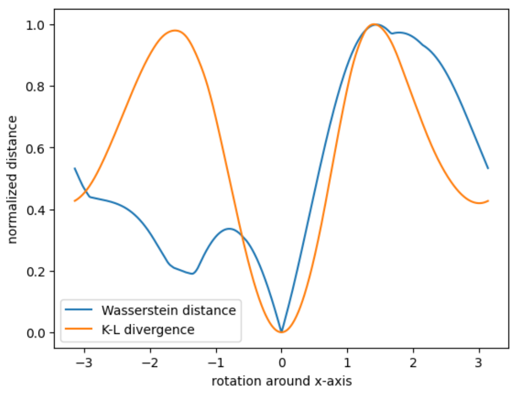
Figure 4.5: Approximated Wasserstein distance with different s

4.4 Comparing approximate Wasserstein distance with K-L divergence

In the experiments, similar to section 4.3, the moving point cloud is modelled by kernel density estimation, and it has 1 degree of freedom in terms of transformation. The corresponding normalized approximate Wasserstein distance and K-L divergence are shown in Figure 4.6. The figures show that on the 23-point Stanford Bunny point cloud, compared to the approximate Wasserstein distance, K-L divergence has a shape that is more conducive to the use of gradient-based optimization methods in most cases. However, when two point clouds are far from each other, the gradient (derivative in this 1-degree-freedom case) of K-L divergence tends to be 0 as expected (recall Equation 3.6). When two point clouds are close enough (e.g., less than 0.1 in Figure 4.6a, but still far enough considering the size of the point cloud), approximate Wasserstein distance can also provide correct descend direction, and it decreases faster than K-L divergence. Similar situations can be observed in the cases of rotation. The rotation angle is represented in radians. The observed decreases when approaching π or $-\pi$ arise from the fact that the downsampled Stanford Bunny point cloud is relatively thin, allowing the data points to be approximated as residing on a single plane. Therefore, due to its symmetry, the period of the approximate Wasserstein distance during rotation is π. When the moving point cloud is subjected to a small angle of rotation, Wasserstein distance can provide correct descend direction, while its decrease speed is at least not worse and even faster in some cases (Figure 4.6d and Figure 4.6f).

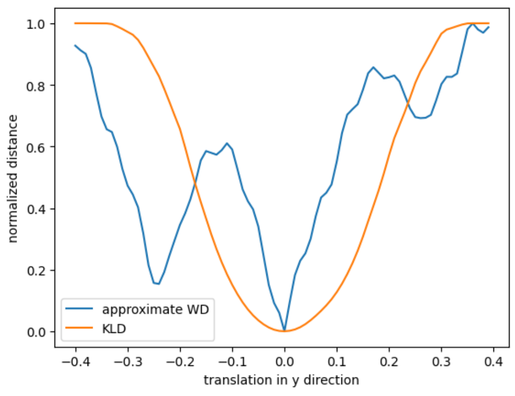


(a) Translate in x direction

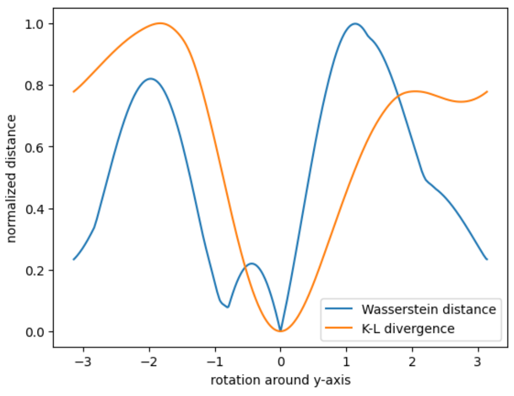


(b) Rotate around x-axis

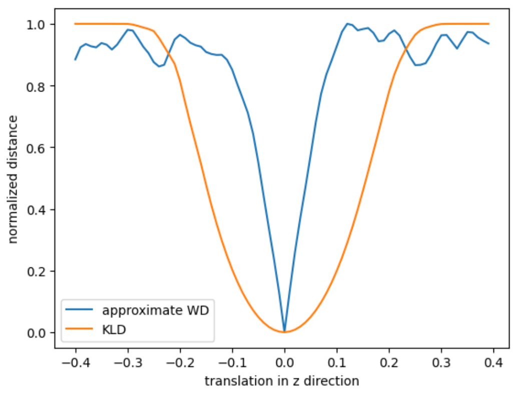
Figure 4.6: Approximate Wasserstein distance (WD) and KullbackLeibler divergence (KLD) with different translation or rotation on source point cloud



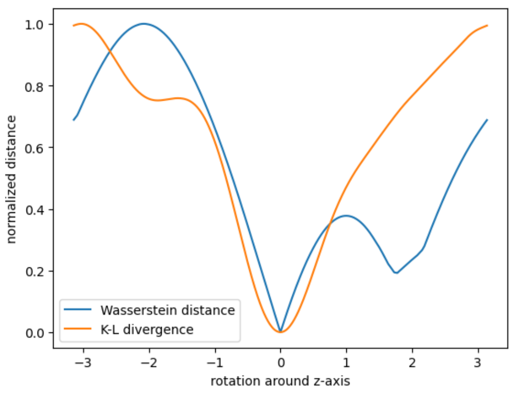
(c) Translate in y direction



(d) Rotate around y-axis



(e) Translate in z direction

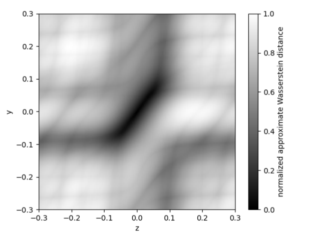


(f) Rotate around z-axis

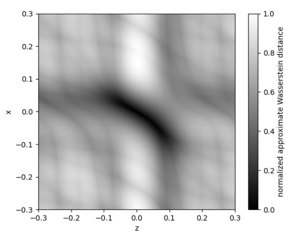
Figure 4.6: Approximate Wasserstein distance (WD) and KullbackLeibler divergence (KLD) with different translation or rotation on source point cloud

Next, the moving point cloud has 2 degree of freedom in translation. Corresponding grayscale images for normalized approximate Wasserstein distance are shown in Figure 4.7 with black representing 0 and white representing 1. For example, in Figure 4.7a, the grayscale value of a pixel located in (z, y) corresponds to the normalized approximate Wasserstein distance between the fixed point cloud and moving point cloud, where the moving point cloud is translated to position $(0, y, z)$.

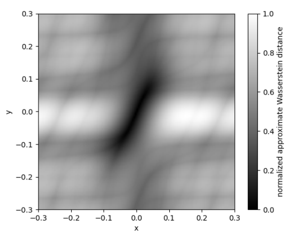
The figures show that the global minimum of approximate Wasserstein distance is obtained around $(0, 0, 0)$ as expected and desired. However, multiple local optima exist, which may prevent gradient-based optimization algorithms from reaching the global optimal solution. In contrast, as shown in Figure 4.8, K-L divergence has a better shape in this case.



(a) Translate in z-y plane, $x = 0$

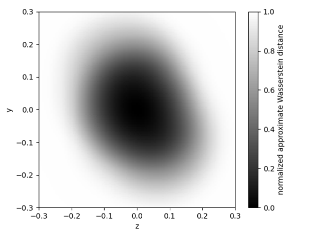


(b) Translate in z-x plane, $y = 0$

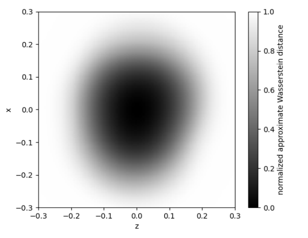


(c) Translate in x-y plane, $z = 0$

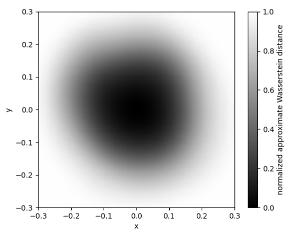
Figure 4.7: Grayscale images for normalized approximate Wasserstein distance



(a) Translate in z-y plane, $x = 0$



(b) Translate in z-x plane, $y = 0$



(c) Translate in x-y plane, $z = 0$

Figure 4.8: Grayscale images for normalized K-L divergence

4.5 Point cloud registration with approximate Wasserstein distance

The experiments begin with 1-degree-of-freedom transformation, by allowing the moving point cloud to only translate along the z-axis, with constraints set to $x = 0$, $y = 0$. Then, the moving point cloud is allowed to rotate around its z-axis, with its centre being always the same as the target point cloud. Subsequently, an experiment involving 2 degrees of freedom will be conducted, allowing the moving point cloud to translate on the $x - z$ plane. Finally, experiments featuring six degrees of freedom, which include translations in $\mathbb{R}^3$ and rotations in $SO(3)$, will be conducted. Additionally, a comparison of success rates with the registration algorithm that utilizes K-L divergence will be included.

In the first experiment, for Figure 4.9a, the centre of moving point cloud is translated to $(0, 0, z)$ and the point cloud registration experiments was performed using the approximate Wasserstein distance proposed in this thesis. Assume that the optimal translation estimated by the algorithm that aligns the two point clouds, which acts on the moving point cloud, is $(0, 0, -\hat{z})$. Then, the formula for calculating the relative error with respect to the point cloud size is

$$\text{relative error} = \frac{|z - \hat{z}|}{L_z}, \tag{4.6}$$

where L_z denotes the length of the point cloud in z direction, i.e.,

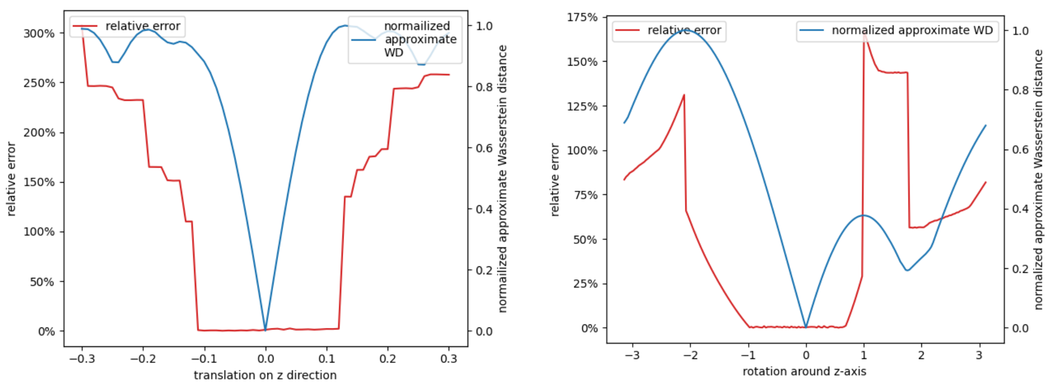
$$L_z = 0.023 - (-0.094) = 0.117$$

for the Stanford Bunny (refers to Equation 4.1).

For Figure 4.9b, the centre of moving point cloud remains aligned with that of the target point cloud, but the moving point cloud is allowed to rotate around its z-axis. The angle of rotation is expressed in radians, and it is known that when the angle is 0, the two point clouds are perfectly aligned. Similarly, the formula for calculating the relative error is

$$\text{relative error} = \frac{|\gamma - \hat{\gamma}|}{\pi}, \tag{4.7}$$

where γ denotes the actual rotation angle (around z-axis) on the moving point cloud and $\hat{\gamma}$ denotes the estimated one.



(a) Point cloud registration results with translation along z-axis

(b) Point cloud registration results with rotation around z-axis

Figure 4.9: Point cloud registration results with 1 degree of freedom

Figure 4.9 shows that when the two point clouds are close enough, i.e., when approximate Wasserstein distance can provide the correct gradient descent direction, the registration result is good and relative errors are nearly 0. When the two point clouds are far apart enough to be in a region with local optima, the gradient-based method will fall into the local optima.

In the second experiment, as results shown in Figure 4.10, the centre of moving point cloud is translated to $(x, 0, z)$, where $x \in [-0.09, 0.09]$ and $z \in [-0.3, 0.3]$. The formula of relative error is

$$\text{relative error} = \frac{\|(x, 0, z) - (\hat{x}, 0, \hat{z})\|}{\|(L_x, 0, L_z)\|}. \tag{4.8}$$

Figure 4.10 shows that for some small translations applied to the moving point cloud, our proposed algorithm can return the correct translation to register the point clouds. However, the problem arising from local optima (refers to Figure 4.7b) prevents the algorithm from returning correct translation when two point clouds are far from each other on z and y directions.

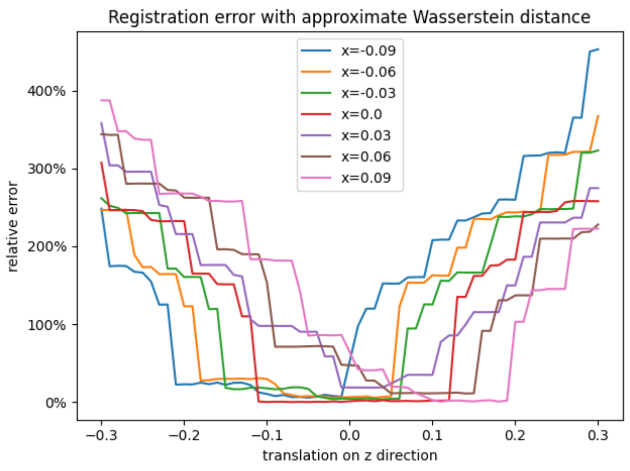


Figure 4.10: Point cloud registration results with translation on $x - z$ plane

In the third experiment, Gaussian noise with deviations $(0.01L_x, 0.01L_y, 0.01L_z)$ is added to the points in the moving point cloud. Then, the moving point cloud will be translated and rotated to random positions. Let (x, y, z) be the tuple of random variables that denotes the coordinate of the centre of the moving point cloud after translating. Let α, β, γ be the random variables that denote the rotation angles around the x, y, z axes passing through the centre of the moving point cloud. Then the random variables follow the following distribution:

$$\begin{cases} x \sim U([-0.3, -0.01] \cup [0.01, 0.3]), \\ y \sim U([-0.3, -0.01] \cup [0.01, 0.3]), \\ z \sim U([-0.3, -0.01] \cup [0.01, 0.3]), \\ \alpha \sim U\left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \\ \beta \sim U\left(-\frac{\pi}{2}, \frac{\pi}{2}\right), \\ \gamma \sim U\left(-\frac{\pi}{2}, \frac{\pi}{2}\right). \end{cases} \tag{4.9}$$

After applying the transformation on the moving point cloud, we use the proposed algorithm to align two point clouds. Recall chapter 1, nearest neighbour method is applied to estimate the correspondences. Since the target point cloud (the fixed one) and the source point cloud (the moving one) come from exactly the same data set, the ground truth of the correspondence information is already known. Compare the estimated correspondences with the ground truth, if 50% of the correspondences are correct, then the registration is considered to be successful. Table 4.1 shows the comparison of success rate of the algorithm using our proposed approximate Wasserstein distance with the one using K-L divergence.

"Distance" metric	Success rate
Approximate Wasserstein distance	0.7%
KullbackLeibler divergence	78.9%

Table 4.1: Success rates of proposed point cloud registration algorithm with approximated Wasserstein distance and KullbackLeibler divergence

This experiment shows that in the 6-degree-of-freedom case, the algorithm can not work for most of the time. As discussed in previous experiments, the reason for the low success rate is that the objective (refers to Equation 2.60) is not convex enough when using the Stanford Bunny dataset and the proposed basis functions Equation 2.27.

5. Conclusion and outlook

In the beginning, we introduced the Lidar system and the assumptions regarding the point clouds, specifically that only depth information is available. We also discussed various methods for addressing the 3D point cloud registration problem, underscoring the challenges and techniques relevant to spatial data handling.

Following this, the probabilistic method and its two steps, namely the fitting step and the registration step, are discussed. Specifically, we focused on applying the Gaussian Mixture Model and Kernel Density Estimation to the fitting step and employing K-L divergence and Wasserstein distance to the registration step. Then, we proposed using linear programming techniques to approximate the Wasserstein-1 distance that preserves differentiability through Kantorovich-Rubinstein Duality.

The experiments demonstrate that the gradient-based registration algorithm, which utilizes the proposed approximate Wasserstein distance, performs effectively when the point clouds are close together and when restricted to 1 or 2 degrees of freedom. These experiment results not only highlight the potential of employing a linear programming approach to approximate Wasserstein distance via Kantorovich-Rubinstein Duality but also validate our theoretical framework regarding the differentiability w.r.t transformation on the probability measure.

However, subsequent experiments show limitations in this approximation approach, specifically the non-strictly convexity of the objective. The existence of local optima prevents gradient-based algorithms from being effective in most cases. These findings suggest that while the proposed method shows promise under some conditions, its applicability in broader contexts requires further enhancement on basis functions to overcome the challenges of local optima.

The first area for future exploration is improving the probabilistic models used to represent point clouds. Gaussian Mixture Model and Kernel Density Estimation approaches currently employed offer a solid foundation, but GMM has limitations in capturing the full complexity of point cloud data, while KDE is relatively slow in terms of computational speed. Research could focus on developing more sophisticated models that better capture the details of spatial distributions while ensuring fast computation. This could lead to more robust and accurate point cloud representations and speed up the registration step.

Another promising research direction is the exploration of alternative basis functions for approximating the Wasserstein distance. The current basis functions (Equation 2.27), characterized by trigonometric functions, are symmetric about the origin and exhibit periodicity. This presents certain challenges, such as its inadequacy in accurately approximating asymmetric or non-periodic Lipschitz functions.

Further, there is significant potential in exploring the integration of the current methods with other advanced topics such as feature-based methods, neural networks, and deep learning architectures. The proposed scheme of using linear programming for approximating the Wasserstein distance points towards a new possible direction for obtaining a metric to measure the similarity between two probability measures while preserving differentiability with respect to transformations of the probability measures. Such a metric potentially benefits other research areas, such as Image Processing, Machine Learning, Computational Neuroscience, Finance, and Climate Science.

A. Appendix

A.1 LBFGS Optimizer

Algorithm 3 LBFGS Optimizer

```
1: Input: $\mathcal{J}(\alpha, \beta, \gamma, \mathbf{t})$ (the objective function);
          $\alpha_0, \beta_0, \gamma_0, \mathbf{t}_0$ (initial values of the parameters);
          $m$ (history size of previous updates);
          $\epsilon$ (convergence threshold);
          $max\_iter$ (maximum number of iterations);
          $B_0$ (initial value for inverse Hessian matrix).
2: Output: $\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\mathbf{t}}$ (optimal rotation and translation parameters).
3: $g_0 \leftarrow \nabla\mathcal{J}(\alpha_0, \beta_0, \gamma_0, \mathbf{t}_0)$
4: $k \leftarrow 0$
5: while $\|g_k\| > \epsilon$ and $k < max\_iter$ do
6:                           ▷ Apply the two-loop recursion to compute $B_k g_k$.
7:     if $k > 0$ then
8:         $r \leftarrow g_k$
9:         for $i \leftarrow k-1$ down to $\max(0, k-m)$ do
10:            $\rho_i \leftarrow 1/(y_i^\top s_i)$
11:            $a_i \leftarrow \rho_i s_i^\top r$
12:            $r \leftarrow r - a_i y_i$
13:        $\mu_k = \frac{s_{k-1}^\top y_{k-1}}{y_{k-1}^\top y_{k-1}}$
14:        $B_0 \leftarrow \mu_k I$
15:        $z \leftarrow B_0 r$
16:        for $i \leftarrow \max(0, k-m)$ to $k-1$ do
17:            $b \leftarrow \rho_i y_i^\top z$
18:            $z \leftarrow z + s_i(a_i - b)$
19:    else
20:        $z \leftarrow B_0 g_0$
21:                                                      ▷ Descent direction
22:    $p_k \leftarrow -z$
23:                                                      ▷ Perform a line search
24:    $\lambda_k \leftarrow \arg\min_\lambda \mathcal{J}(\alpha_k + \lambda p_k^\alpha, \beta_k + \lambda p_k^\beta, \gamma_k + \lambda p_k^\gamma, \mathbf{t}_k + \lambda p_k^\mathbf{t})$
```

```
25:                                        ▷ Update $\alpha, \beta, \gamma, \mathbf{t}$
26:     $\alpha_{k+1} \leftarrow \alpha_k + \lambda_k p_k^{\alpha}$
27:     $\beta_{k+1} \leftarrow \beta_k + \lambda_k p_k^{\beta}$
28:     $\gamma_{k+1} \leftarrow \gamma_k + \lambda_k p_k^{\gamma}$
29:     $\mathbf{t}_{k+1} \leftarrow \mathbf{t}_k + \lambda_k p_k^{\mathbf{t}}$
30:     $s_k \leftarrow \lambda_k p_k$
31:     $y_k \leftarrow \nabla\mathcal{J}(\alpha_{k+1}, \beta_{k+1}, \gamma_{k+1}, \mathbf{t}_{k+1}) - g_k$
32:     $g_{k+1} \leftarrow \nabla\mathcal{J}(\alpha_{k+1}, \beta_{k+1}, \gamma_{k+1}, \mathbf{t}_{k+1})$
33:     $k \leftarrow k + 1$
34:     if $k > m$ then
35:                                        ▷ Delete early historical values
36:         delete $y_{k-m-1}$
37:         delete $s_{k-m-1}$
38:     delete other unnecessary values
39: return $\alpha_k, \beta_k, \gamma_k, \mathbf{t}_k$
```

A.2 Reference LBFGS algorithm embedded with c update

Algorithm 4 Optimization with LBFGS, update **c** inside

```
1: Input: $p, q$ (probability density functions);
          $s$ (upper bound for $i + j + k$);
          $\alpha_0, \beta_0, \gamma_0, \mathbf{t}_0$ (initial values for rotation degrees and translation);
          $max\_iter$ (maximum number of iterations);
          $m$ (history size of previous updates);
          $\epsilon$ (convergence threshold);
          $B_0$ (initial value for inverse Hessian).
2: Output: $\hat{\alpha}, \hat{\beta}, \hat{\gamma}, \hat{\mathbf{t}}$ (optimal values for rotation degrees and translation).
3: $l \leftarrow 0$
4: for $i \leftarrow 0$ to $s$ do
5:     for $j \leftarrow 0$ to $s - i$ do
6:         for $k \leftarrow 0$ to $s - i - j$ do
7:             $I_{ijk} \leftarrow \int_{\mathbb{R}^3} [p(\mathbf{x}) - q(\mathbf{R}(\alpha_l, \beta_l, \gamma_l)\mathbf{x} + \mathbf{t}_l)]\phi_{ijk}(\mathbf{x})\, dV$
8: $\mathbf{\Gamma} \leftarrow [I_{000}, I_{001}, \ldots, I_{ijk}, \ldots, I_{s00}]$
9: $\hat{\mathbf{c}} \leftarrow \arg\max_{\mathbf{c}} \mathbf{\Gamma}\mathbf{c}$ subject to $|c_{ijk}| \leq \frac{\sqrt{3}}{(s+1)(s+2)(s+3)(s-2)\pi}\ \forall i, j, k$
10: define $\mathcal{J}_{\hat{\mathbf{c}}}(\alpha, \beta, \gamma, \mathbf{t}) = \sum_{i+j+k \leq s} \hat{c}_{ijk} \int_{\mathbb{R}^3} [p(\mathbf{x}) - q(\mathbf{R}(\alpha, \beta, \gamma)\mathbf{x} +$
    $\mathbf{t})]\phi_{ijk}(\mathbf{x})\, dV$
```

```
11: $g_0 \leftarrow \nabla \mathcal{J}_{\hat{\mathbf{c}}}(\alpha_l, \beta_l, \gamma_l, \mathbf{t}_l)$
12: while $\|g_l\| > \epsilon$ and $l < max\_iter$ do
13:     if $l > 0$ then                ▷ Apply the two-loop recursion to compute $B_l g_l$.
14:         $r \leftarrow g_l$
15:         for $i \leftarrow l-1$ down to $\max(0, l-m)$ do
16:             $\rho_i \leftarrow 1/(y_i^\top s_i)$
17:             $a_i \leftarrow \rho_i s_i^\top r$
18:             $r \leftarrow r - a_i y_i$
19:         $\mu_l \leftarrow \frac{s_{l-1}^\top y_{l-1}}{y_{l-1}^\top y_{l-1}}$
20:         $B_0 \leftarrow \mu_l I$
21:         $z \leftarrow B_0 r$
22:         for $i \leftarrow \max(0, l-m)$ to $l-1$ do
23:             $b \leftarrow \rho_i y_i^\top z$
24:             $z \leftarrow z + s_i(a_i - b)$
25:     else
26:         $z \leftarrow B_0 g_0$
27:     $p_l \leftarrow -z$                ▷ Descent direction
28:     $\lambda_l \leftarrow \arg\min_\lambda \mathcal{J}(\alpha_l + \lambda p_l^\alpha, \beta_l + \lambda p_l^\beta, \gamma_l + \lambda p_l^\gamma, \mathbf{t}_l + \lambda p_l^{\mathbf{t}})$ ▷ Perform a
        line search to solve
29:     $\alpha_{l+1} \leftarrow \alpha_l + \lambda_l p_l^\alpha$
30:     $\beta_{l+1} \leftarrow \beta_l + \lambda_l p_l^\beta$
31:     $\gamma_{l+1} \leftarrow \gamma_l + \lambda_l p_l^\gamma$
32:     $\mathbf{t}_{l+1} \leftarrow \mathbf{t}_l + \lambda_l p_l^{\mathbf{t}}$                ▷ Update $\alpha, \beta, \gamma, \mathbf{t}$
33:     $s_l \leftarrow \lambda_l p_l$
34:     $y_l \leftarrow \nabla \mathcal{J}_{\hat{\mathbf{c}}}(\alpha_{l+1}, \beta_{l+1}, \gamma_{l+1}, \mathbf{t}_{l+1}) - g_l$
35:     $g_{l+1} \leftarrow \nabla \mathcal{J}_{\hat{\mathbf{c}}}(\alpha_{l+1}, \beta_{l+1}, \gamma_{l+1}, \mathbf{t}_{l+1})$
36:     $l \leftarrow l + 1$
37:     if $l > m$ then                ▷ Delete early historical values
38:         delete $y_{l-m-1}$
39:         delete $s_{l-m-1}$
40:     delete other unnecessary values
41: return $\alpha_l, \beta_l, \gamma_l, \mathbf{t}_l$
```

A.3 Code

The reference implementation with Python for the algorithms is available at `https://github.com/Li4ngXu/ProbMethodForPointCloudReg`.

A.4 Notations and conventions

Mathematical expression	Meaning
D	dimension
$\mathbb{R}$	the set of real numbers
$\mathbf{x}$	D-dimensional point
$\mathbf{y}$	D-dimensional point
$\mathcal{P}$	point cloud
$\mathcal{Q}$	point cloud
T	transformation
$\mathcal{SE}(D)$	D-dimensional special Euclidean group
$\mathcal{SO}(D)$	D-dimensional rotation group
$\mathbf{R}$	rotation matrix
$\mathbf{t}$	translation vector
n	number of points in a point cloud
$\mathbb{M}$	probability model
Θ	tuple of parameters of some probability model
$\mathbb{P}$	probability
K	number of components (of GMM)
$\mathcal{K}$	kernel function
$\mathcal{N}$	Gaussian (Normal) distribution
$\boldsymbol{\mu}$	mean vector (for Gaussian distribution)
$\boldsymbol{\Sigma}$	covariance matrix (for Gaussian distribution)
$\mathbf{I}$	identity matrix
sc	silhouette score

Table A.1: Notations

Bibliography

[1] Ricardo Roriz, Jorge Cabral, and Tiago Gomes. “Automotive LiDAR Technology: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (July 2022), pp. 6282–6297. ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3086804.

[2] Swadhin Pradhan et al. “Smartphone-based Acoustic Indoor Space Mapping”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.2 (July 5, 2018), 75:1–75:26. DOI: 10.1145/3214278.

[3] Xiang Li et al. “Gaussian Mixture Model-Based Registration Network for Point Clouds with Partial Overlap”. en. In: *Artificial Neural Networks and Machine Learning ICANN 2022*. Ed. by Elias Pimenidis et al. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2022, pp. 405–416. ISBN: 978-3-031-15934-3. DOI: 10.1007/978-3-031-15934-3_34.

[4] Xiaoshui Huang et al. *A comprehensive survey on point cloud registration*. en. Mar. 2021.

[5] Stephane Allaire et al. “Full orientation invariance and improved feature selectivity of 3D SIFT with application to medical image analysis”. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. June 2008, pp. 1–8. DOI: 10.1109/CVPRW.2008.4563023.

[6] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation. May 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[7] Samuele Salti, Federico Tombari, and Luigi Di Stefano. “SHOT: Unique signatures of histograms for surface and texture description”. In: *Computer Vision and Image Understanding* 125 (Aug. 1, 2014), pp. 251–264. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2014.04.011.

[8] P.J. Besl and Neil D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 1939-3539. DOI: 10.1109/34.121791.

[9] Yang Chen and Gérard Medioni. “Object modelling by registration of multiple range images”. In: *Image and Vision Computing*. Range Image Understanding 10.3 (Apr. 1, 1992), pp. 145–155. ISSN: 0262-8856. DOI: 10.1016/0262-8856(92)90066-C.

[10] C. Brenner, C. Dold, and N. Ripperda. “Coarse orientation of terrestrial laser scans in urban environments”. In: *ISPRS Journal of*

Photogrammetry and Remote Sensing. Theme Issue: Terrestrial Laser Scanning 63.1 (Jan. 1, 2008), pp. 4–18. ISSN: 0924-2716. DOI: `10.1016/j.isprsjprs.2007.05.002`.

[11] E. Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Springer Science & Business Media, Mar. 14, 2013. 296 pp. ISBN: 978-1-4757-2787-6.

[12] Huu Le et al. *SDRSAC: Semidefinite-Based Randomized Approach for Robust Point Cloud Registration without Correspondences*. Apr. 14, 2019. arXiv: `1904.03483[cs]`.

[13] Wentao Yuan et al. "DeepGMR: Learning Latent Gaussian Mixture Models for Registration". en. In: *Computer Vision ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 733–750. ISBN: 978-3-030-58558-7. DOI: `10.1007/978-3-030-58558-7_43`.

[14] Douglas Reynolds. "Gaussian Mixture Models". In: *Encyclopedia of Biometrics*. Ed. by Stan Z. Li and Anil K. Jain. Boston, MA: Springer US, 2015, pp. 827–832. ISBN: 978-1-4899-7488-4. DOI: `10.1007/978-1-4899-7488-4_196`.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data Via the *EM* Algorithm". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 39.1 (Sept. 1, 1977), pp. 1–22. ISSN: 1369-7412, 1467-9868. DOI: `10.1111/j.2517-6161.1977.tb01600.x`.

[16] Shuping Sun et al. "An adaptive optimization method for estimating the number of components in a Gaussian mixture model". In: *Journal of Computational Science* 64 (Oct. 1, 2022), p. 101874. ISSN: 1877-7503. DOI: `10.1016/j.jocs.2022.101874`.

[17] Jouni Kuha. "AIC and BIC: Comparisons of Assumptions and Performance". en. In: *Sociological Methods & Research* 33.2 (Nov. 1, 2004), pp. 188–229. ISSN: 0049-1241. DOI: `10.1177/0049124103262065`.

[18] H. Akaike. "A New Look at the Statistical Model Identification". In: *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. ISSN: 0018-9286. DOI: `10.1109/TAC.1974.1100705`.

[19] Yen-Chi Chen. "A tutorial on kernel density estimation and recent advances". In: *Biostatistics & Epidemiology* 1.1 (Jan. 1, 2017), pp. 161–187. ISSN: 2470-9360. DOI: `10.1080/24709360.2017.1396742`.

[20] Victor M. Panaretos and Yoav Zemel. "Statistical Aspects of Wasserstein Distances". In: *Annual Review of Statistics and Its Application* 6 (Volume 6, 2019 Mar. 7, 2019), pp. 405–431. ISSN: 2326-8298, 2326-831X. DOI: `10.1146/annurev-statistics-030718-104938`.

[21] Soheil Kolouri et al. "Generalized Sliced Wasserstein Distances". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[22] Sameer Shirdhonkar and David W. Jacobs. "Approximate earth movers distance in linear time". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008 IEEE Conference on Computer

Vision and Pattern Recognition. June 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587662.

[23] Samantha Chen and Yusu Wang. *Neural approximation of Wasserstein distance via a universal architecture for symmetric and factor-wise group invariant functions.* Nov. 17, 2023. arXiv: 2308.00273[cs].

[24] Ben Eckart et al. "MLMD: Maximum Likelihood Mixture Decoupling for Fast and Accurate Point Cloud Registration". In: *2015 International Conference on 3D Vision.* 2015 International Conference on 3D Vision. Oct. 2015, pp. 241–249. DOI: 10.1109/3DV.2015.34.

[25] Hanchen Xiong, Sandor Szedmak, and Justus Piater. "A Study of Point Cloud Registration with Probability Product Kernel Functions". In: *2013 International Conference on 3D Vision - 3DV 2013.* 2013 International Conference on 3D Vision - 3DV 2013. June 2013, pp. 207–214. DOI: 10.1109/3DV.2013.35.

[26] Guangfu Qu and Won Hyung Lee. "Point Set Registration Based on Improved KL Divergence". en. In: *Scientific Programming* 2021 (Oct. 6, 2021). Ed. by Bai Yuan Ding, pp. 1–8. ISSN: 1875-919X, 1058-9244. DOI: 10.1155/2021/1207569.

[27] Lin Hongbin and Liu Bin. "Research on a novel 3D point cloud robust registration algorithm". In: *2010 International Conference On Computer Design and Applications.* 2010 International Conference On Computer Design and Applications. Vol. 5. June 2010, pp. V5–441–V5–445. DOI: 10.1109/ICCDA.2010.5540928.

[28] Trung Nguyen et al. "Point-set Distances for Learning Representations of 3D Point Clouds". en. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV).* 2021 IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada: IEEE, Oct. 2021, pp. 10458–10467. ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.01031.

[29] Chien-Chou Lin et al. "A novel point cloud registration using 2D image features". In: *EURASIP Journal on Advances in Signal Processing* 2017.1 (Jan. 7, 2017), p. 5. ISSN: 1687-6180. DOI: 10.1186/s13634-016-0435-y.

[30] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision ECCV 2006.* Ed. by Ale Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer, 2006, pp. 404–417. ISBN: 978-3-540-33833-8. DOI: 10.1007/11744023_32.

[31] Lingjing Wang et al. *Non-Rigid Point Set Registration Networks.* Apr. 2, 2019. DOI: 10.48550/arXiv.1904.01428. arXiv: 1904.01428[cs].

[32] Weixin Lu et al. "DeepVCP: An End-to-End Deep Neural Network for Point Cloud Registration". In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 12–21.

[33] Andy Zeng et al. “3DMatch: Learning Local Geometric Descriptors From RGB-D Reconstructions”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017, pp. 1802–1811.

[34] Martin A. Fischler and Robert C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Communications of the ACM* 24.6 (June 1, 1981), pp. 381–395. ISSN: 0001-0782. DOI: `10.1145/358669.358692`.

[35] Xiaoshui Huang, Guofeng Mei, and Jian Zhang. “Feature-Metric Registration: A Fast Semi-Supervised Approach for Robust Point Cloud Registration Without Correspondences”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 11366–11374.

[36] Ketan Rajshekhar Shahapure and Charles Nicholas. “Cluster Quality Analysis Using Silhouette Score”. In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA). Oct. 2020, pp. 747–748. DOI: `10.1109/DSAA49011.2020.00096`.

[37] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1, 1987), pp. 53–65. ISSN: 0377-0427. DOI: `10.1016/0377-0427(87)90125-7`.

[38] Onrina Chandra. “Choice of the Bandwidth in Kernel Density Estimation”. In: 9.9 (2018).

[39] Adrian G. Bors and Nikolaos Nasios. “Kernel Bandwidth Estimation for Nonparametric Modeling”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.6 (Dec. 2009), pp. 1543–1555. ISSN: 1941-0492. DOI: `10.1109/TSMCB.2009.2020688`.

[40] Cédric Villani. “Cyclical monotonicity and Kantorovich duality”. In: *Optimal Transport: Old and New*. Ed. by Cédric Villani. Berlin, Heidelberg: Springer, 2009, pp. 51–92. ISBN: 978-3-540-71050-9. DOI: `10.1007/978-3-540-71050-9_5`.

[41] Tristan van Leeuwen and Aleksandr Aravkin. *Non-smooth Variable Projection*. Nov. 20, 2020. arXiv: `1601.05011[math,stat]`.

[42] Christopher M. Bishop. *Pattern recognition and machine learning*. en. Information science and statistics. New York: Springer, 2006. 738 pp. ISBN: 978-0-387-31073-2.

[43] Stanford University Computer Graphics Laboratory. *The Stanford 3D Scanning Repository*. `https://graphics.stanford.edu/data/3Dscanrep/`.

[44] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. Dec. 6, 2017. DOI: `10.48550/arXiv.1701.07875`. arXiv: `1701.07875[cs,stat]`.