Collision Distance Estimation for High-dof Robot Systems: A Learning-Based Approach

JIHWAN KIM

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in the
DEPARTMENT OF MECHANICAL ENGINEERING

at
SEOUL NATIONAL UNIVERSITY

February 2025

ABSTRACT

Collision Distance Estimation for High-dof Robot Systems: A Learning-Based Approach

by

Jihwan Kim

Department of Mechanical Engineering
Seoul National University

Collision distance estimation is a critical component in robot path planning and obstacle avoidance. While traditional geometric algorithms have been effective in certain contexts, recent advancements in learning-based robot path planning algorithms and sampling-based planning techniques have highlighted their limitations. These modern approaches often require rapid, parallel computation of collision distances for multiple configurations and their derivatives, which traditional methods struggle to provide efficiently. As a result, these limitations have spurred the development of learning-based approaches for collision distance estimation.

However, existing learning-based methods encounter significant challenges when applied to complex, high-dof robot systems. These challenges include difficulties in dataset construction for the high-dimensional configuration space, the inherent complexity of the collision distance function, and limited generalizability to minor environmental changes. Consequently, there is a pressing need for more sophisticated and adaptable collision distance estimation techniques.

This thesis presents two primary contributions to address these challenges. First, we introduce an active learning strategy for efficient dataset construction in high-dof robot systems. This method focuses on sampling the most informative configurations near collision boundaries, significantly improving the quality of training data and enhancing model performance, particularly in critical regions.

Second, we propose PairwiseNet, an innovative method for pairwise collision distance learning. PairwiseNet focuses on predicting the minimum distance between pairs of elements within the robot system rather than directly estimating the collision distance of the entire system. This approach simplifies the learning task and demonstrates remarkable generalizability across various robot configurations.

Extensive experiments conducted on complex robot systems, including high-dof multi-arm systems and single-arm robots in obstacle-rich environments, validate the effectiveness of our approaches. Results show significant improvements in collision distance regression error, collision checking accuracy, and false positive rates compared to existing methods. Our contributions advance the state-of-the-art in collision distance estimation for complex robot systems, offering more accurate, efficient, and adaptable solutions.

Keywords: Collision distance, machine learning, dataset construction, active learning, pairwise collision distance, path planning, collision avoidance.

Student Number: 2019-24947

Contents

List of Tables

List of Figures

1 Introduction

1.1 Collision Distance Estimation for Path Planning

In the domain of robotics research and applications, accurately estimating the proximity between a robot and its surrounding obstacles plays a crucial role in path planning and obstacle avoidance. This concept, known as *collision distance*, is defined as the minimum distance between a robot and any obstacles in its environment (including potential self-collisions). This distance is essential in various aspects of robot motion planning, from sampling-based methods like Rapidly-exploring Random Trees (RRT) and its variants [2, 3, 4] to potential field methods [5, 6, 7, 8, 9] and graph search algorithms [10, 11, 12, 13].

Traditionally, these calculations rely on geometric algorithms operating on CAD models of the robot and its environment. While these methods have been effective for many robotic applications, recent advancements in learning-based robot path planning algorithms [14, 15, 16, 17] and sampling-based planning techniques [18, 19, 20, 21, 22, 23] have introduced new challenges that highlight the limitations of classical collision

distance calculation methods. These limitations become apparent in three key areas:

- Computational speed: Modern approaches like reinforcement learning and learning from demonstration often require rapid evaluation of numerous configurations, pushing the limits of traditional calculation methods.
- Lack of batch calculation capability: Advanced sampling-based techniques such as model predictive path integral control schemes benefit from simultaneous evaluation of multiple configurations, which is inefficient with sequential processing methods.
- Difficulty in computing derivatives: Many contemporary motion planning and control algorithms, particularly those based on optimization, require efficient access to collision distance derivatives, which is often computationally expensive or infeasible with classical methods.

These limitations make classical collision distance calculation methods less suitable for cutting-edge applications that require processing large numbers of configurations. As robotics research continues to push the boundaries with more sophisticated planning and control algorithms, there is a growing need for more efficient and versatile collision distance estimation techniques.

1.2 Learning-based Collision Distance Estimation

To address the limitations of classical methods, researchers have turned to machine learning approaches for collision distance estimation. Most methods have attempted to learn models that predict the collision distance (or the collision label) from robot configurations. By collecting sufficient data consisting of robot configurations and their corresponding collision distances, various machine learning models have been employed

to learn the collision distance function (2.2.4). These include kernel perceptron models [24, 25, 26], support vector machines [27, 28], Gaussian processes [29], and neural networks [8, 18, 28, 30, 31, 32].

While these data-driven approaches have shown promise in addressing some of the limitations of classical methods, they face three significant challenges. Two of these challenges are particularly pronounced when applied to complex, high-dof robot systems: difficulties in dataset construction and the inherent complexity of the collision distance function. The third challenge, which is not specific to high-dof systems but is nonetheless significant, is the lack of adaptability to minor environmental changes. Even slight changes in the robot base pose or obstacle positions often necessitate retraining of the entire model. These challenges will be elaborated upon in the subsequent sections.

1.2.1 Dataset Construction

Existing learning-based collision distance estimation methods typically generate datasets comprising various joint configurations paired with corresponding collision distances [28, 31, 32] or boolean collision labels [8, 24, 25, 26, 27]. These approaches often utilize datasets of less than a million samples, drawn from a uniform distribution in the joint configuration space. While effective in their specific contexts, most of these methods are limited to simple, low-dof robot systems.

In general machine learning problems, it is desirable for the training dataset to adequately cover the target space – the input data space where accurate prediction of the target value is crucial. In the context of collision distance learning, the target value is the collision distance, and the target space is the joint configuration space, necessitating comprehensive coverage by the training dataset. Moreover, the required density of the training data points in the joint configuration space typically increases when the

collision distance function is complex and exhibits rapid variations.

However, a significant challenge arises as the dof of the system increases: the data requirement to sufficiently cover the joint configuration space grows exponentially. Consider, for instance, a system comprising two 7-dof robot arms. Discretizing this 14-dof joint configuration space at 10-degree intervals (which is still too wide for practical applications) and assuming a 200-degree range for each joint would result in over 100 billion joint configurations ($\sim 20^{14}$) – an unfeasible number to sample and process.

Consequently, for complex, high-dof robot systems, it becomes impractical to generate a training dataset that adequately covers the joint configuration space; a uniformly sampled dataset of one million samples for a 14-dof robot system would provide only about $\sqrt[14]{10^6} \approx 2.68$ samples per joint, which is insufficient for accurate learning. This data sparsity is a key factor contributing to the difficulty of learning collision distance functions for high-dof robot systems. Despite this challenge, many existing methods fail to address the insufficiency of datasets and continue to generate samples from a uniform distribution in the joint configuration space. This approach overlooks the fundamental issue of data sparsity in high-dimensional spaces and limits the effectiveness of these methods for high-dof robot systems.

1.2.2 Inherent Complexity of the Collision Distance Function

The collision distance function is defined as the minimum of all pairwise distances between elements in the system. This definition inherently results in a complex and non-smooth function. Figure 1.1 illustrates this complexity by showing the collision distance plot of a four-arm robot system as it follows a smooth joint trajectory.

As the robot arms move along a trajectory, the closest pair of elements continuously

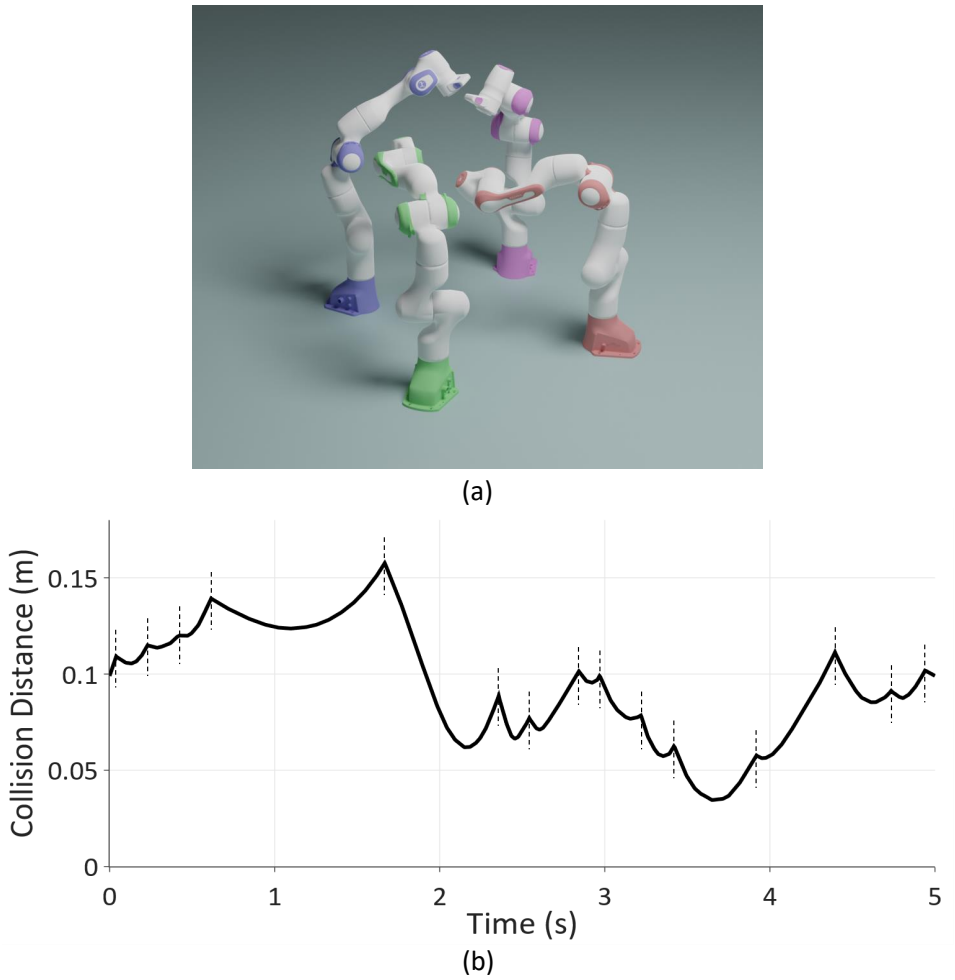

Figure 1.1: Illustration of the complex nature of collision distance function. (a) A four-arm robot system. (b) Collision distance plot as the system follows a smooth joint trajectory. The black dashed lines indicate points where the closest pair changes, demonstrating the non-smooth and abruptly varying nature of the collision distance.

changes, leading to non-smoothness and abrupt variations in the collision distance function even when the robots follow a smooth path. In Figure 1.1(b), the plot is divided by black dashed lines, with each section corresponding to the minimum distance of a specific element pair. These divisions indicate the points at which the closest pair changes.

This intrinsic non-smooth nature of the collision distance function poses a significant challenge for learning approaches. Simple learnable models, which are inherently smooth functions, struggle to accurately capture these abrupt changes and discontinuities in the collision distance landscape. Consequently, more sophisticated modeling techniques are required to effectively represent and predict collision distances in complex robot systems.

Existing methods for learning collision distance functions often rely on simplistic input representations and models. These approaches typically use basic input representations such as raw joint configurations [8, 28, 29, 30, 31], concatenated vectors of Cartesian coordinates for joint positions [26, 27, 30], or positional embedding vectors of joint configurations [18].

Moreover, the models employed in these methods fall into two main categories. The first category includes kernel-based models such as kernel perceptrons [24, 25, 26], support vector machines [27, 28], and Gaussian processes [29]. While these models can be effective for certain applications, they often struggle to scale efficiently when dealing with the large datasets required for learning collision distance functions in high-dof robot systems.

The second category comprises neural network-based approaches [8, 18, 28, 30, 31]. However, these methods frequently employ simple, fully-connected architectures. Such architectures, being inherently smooth functions, may be ill-suited for capturing the complex, non-smooth nature of collision distance functions. An exception is the work of [32], which utilizes an advanced graph neural network structure that represents the

geometric shapes of robots and environments as graphs. Nevertheless, this approach's efficiency is still limited to low-dof robot systems, and its inference speed is not only sensitive to the complexity of the graph but also significantly slower compared to methods utilizing simple fully-connected neural network structures.

This reliance on simplistic representations and models presents a significant limitation in accurately learning and representing collision distance functions, particularly for complex, high-dof robotic systems.

1.2.3 Generalizability to Minor Environmental Changes

A significant limitation of existing learning-based methods is their sensitivity to minor environmental changes. Slight modifications, such as adjustments to the robot's base position or obstacle locations, can result in a substantially different collision distance function. Consequently, many of these methods require a complete retraining process, consisting of both data collection and model training, to adapt to these changes. This lack of flexibility poses a significant challenge for deploying these systems in real-world environments.

While some approaches, like that proposed by [26], offer strategies for efficient model updates in dynamic environments, they are often limited in scope, typically applying only to low-dof robot systems and still necessitating additional training procedures. The model proposed by [32] demonstrates generalizability to environments with random obstacles by employing graph representations of the environment as inputs, but its applicability is constrained to toy examples involving low-dof, 2D planar robot systems. This challenge underscores the need for more adaptable and robust collision distance estimation methods that can readily accommodate environmental variations without extensive retraining.

1.3 Contribution

This thesis addresses the aforementioned challenges in learning-based collision distance estimation for complex robot systems. Our work makes several key contributions to the field, advancing the state-of-the-art in both the methodology and practical application of collision distance estimation. We propose novel approaches that significantly improve the accuracy, efficiency, and adaptability of collision distance estimation, particularly for complex, high-dof robot systems. The following subsections detail our main contributions, each addressing a critical aspect of the collision distance estimation problem.

1.3.1 Active Learning of the Collision Distance Function

To address the challenges of dataset construction for high-dof robot systems, we propose a novel active learning strategy [33]. Our approach is based on a key insight: given limited computational resources, particularly in terms of dataset size, the distribution of data points should be concentrated in the most informative regions of the joint configuration space. In the context of collision distance learning, these critical regions are found near the collision boundaries – areas where the collision distance is zero. By focusing on these boundaries, our strategy efficiently utilizes the limited data to capture the most relevant information for accurate collision distance estimation.

Our active learning-based training procedure iteratively refines the collision distance estimation model. Beginning with an initial dataset, the process involves repeated cycles of model training, generating new data points (joint configurations) near the estimated collision boundaries, calculating collision distances of these points, and updating the dataset with these newly sampled informative points. This iterative approach progressively enhances the model's accuracy, particularly near collision boundaries. Consequently, it proves highly effective for complex, high-dof robot systems where sampling

near the true collision boundaries is often impractical.

Extensive experiments conducted on high-dof robot systems demonstrate that our active learning-based training procedure enhances the performance of baseline models. The improvements are particularly notable in the critical regions near collision boundaries, where accurate distance estimation is crucial for safe and efficient motion planning. These results underscore the effectiveness of our strategy in addressing the challenges of collision distance estimation in complex, high-dof robot systems.

1.3.2 PairwiseNet: Pairwise Collision Distance Learning

Existing methods often employ simplistic input representations and model structures, which struggle to capture the complex collision distance function of high-dof robot systems. To address this limitation, we initially explored a link $SE(3)$ configuration space representation [33], a more informative input representation compared to raw joint configurations or concatenated vectors of joint positions.

The pose $T \in SE(3)$ of each link is described by a 3×3 rotation matrix R and a translation vector $q \in \mathbb{R}^3$, totaling 12 elements. The proposed representation concatenates these 12 elements for all links. By incorporating additional rotational information of links, this approach demonstrates significant performance improvements over baselines using simpler input representations.

While our link $SE(3)$ configuration space representation enhances learning performance, the model architecture remains relatively simple and still faces challenges in learning the complex, abruptly varying collision distance function. This limitation underscores the need for a more comprehensive approach beyond merely modifying input representations. To address this, we propose a novel approach: instead of directly learning the collision distance for the entire robot system – which we term the *global*

collision distance to distinguish it from pairwise distances – we focus on learning the *pairwise* collision distances between specific elements. The global collision distance is inherently non-smooth as it is defined as the minimum of these pairwise collision distances. By shifting our focus to the pairwise distances, we effectively circumvent the difficulties associated with learning the complex, non-smooth global collision distance function.

Building upon our insights into pairwise collision distance learning, we introduce PairwiseNet [34], a novel method for estimating collision distances in complex robot systems. Unlike conventional approaches that directly estimate the global collision distance, PairwiseNet focuses on predicting the minimum distance between pairs of elements within the robot system.

PairwiseNet is designed to input point cloud data of two geometric shapes along with their relative transformation, outputting the minimum distance between these shapes. To compute the global collision distance, PairwiseNet first estimates the pairwise distances for all possible element pairs in the system, then selects the minimum value among these estimates. This approach leverages the efficient parallel batch computation capabilities of neural networks, enabling rapid distance predictions for multiple element pairs simultaneously.

A key advantage of PairwiseNet lies in its ability to simplify the learning task. The pairwise collision distance function is inherently less complex and more manageable to learn compared to the complex, non-smooth global collision distance function. By decomposing the challenging global problem into more tractable sub-problems of pairwise distance learning, PairwiseNet achieves significant performance improvements, especially for high-dof robot systems.

Moreover, PairwiseNet exhibits remarkable generalizability to minor environmental changes, such as changes in the position of robot bases and obstacles. While the global

collision distance function changes significantly even with minor environmental modifications, necessitating most existing methods to retrain their models for every change, PairwiseNet offers a more adaptable solution. Once trained on a dataset of pairwise distances for a set of shape elements, PairwiseNet can be applied to any system composed of these trained elements without requiring additional training or modifications. This flexibility allows PairwiseNet to adapt to various robot configurations, including systems with multiple arms or changed base positions, as long as the constituent shape elements remain consistent with those used during training.

We have evaluated PairwiseNet across a range of scenarios, including high-dof multi-arm systems (from 14-dof two-arm to 28-dof four-arm configurations) and single-arm robots navigating obstacle-rich environments. Our results demonstrate PairwiseNet's superior performance over existing learning-based methods in terms of collision distance regression error, collision checking accuracy, and notably, the False Positive Rate with a collision-free guaranteed threshold (Safe-FPR). Importantly, PairwiseNet maintains its high performance even when a single trained model is applied across diverse multi-arm systems, underscoring its versatility and robustness.

1.4 Organization

Chapter 2 provides a comprehensive background on collision distance, laying the foundation for the subsequent chapters. It introduces the concept of collision distance and its critical importance in robotics, particularly in path planning and obstacle avoidance. The chapter delves into classical methods for collision distance calculation, discussing their principles and limitations. We also discuss how the advent of learning-based planning approaches has introduced new demands in collision distance estimation. Specifically, we highlight the growing importance of rapid calculation, batch processing capabilities,

and efficient derivative computation in modern robotics applications. These emerging requirements, which are often difficult to meet with classical methods, set the stage for the need for more advanced techniques in collision distance estimation.

Chapter 3 presents our novel active learning approach for collision distance function learning, based on [33]. This chapter introduces the concept of active learning in the context of collision distance estimation and explains how it addresses the data efficiency challenges in high-dof systems. We detail our methodology for efficiently sampling informative configurations in high-dimensional joint spaces, with a particular focus on regions near collision boundaries. We also present our link $SE(3)$ configuration space representation, demonstrating its effectiveness in improving learning performance for high-dof robot systems. The chapter includes thorough experimental results that validate the superiority of our approach over existing methods.

Chapter 4 introduces PairwiseNet, our innovative method for pairwise collision distance learning, based on [34]. We provide a detailed explanation of PairwiseNet's architecture and training process, highlighting how it simplifies the complex task of global collision distance estimation by focusing on pairwise distances. We discuss the remarkable generalizability of PairwiseNet to various robot configurations and present comprehensive performance analyses in different scenarios, including multi-arm systems and environments with obstacles.

Chapter 5 explores the practical applications of our collision distance estimation methods in robot path planning. We demonstrate how our approaches can be integrated into various planning algorithms, including offline trajectory optimization and real-time collision avoidance. This chapter showcases the effectiveness of our methods in real-world planning scenarios, particularly emphasizing their performance in complex, high-dof robotic systems.

Throughout these chapters, we provide extensive experimental results and comparative analyses. These evaluations rigorously validate our approaches and demonstrate their superiority over existing methods in collision distance estimation for complex, high-DOF robot systems.

2

Preliminaries: Collision Distance

2.1 Introduction

In the field of robotics, particularly in path planning and collision avoidance, collision distance is a crucial concept. This chapter explores the details of collision distance, its importance in various robot applications, and the methods used to compute it.

We start by defining collision distance as the minimum distance between a robot and any obstacles in its environment. This fundamental measure is essential in many aspects of robot motion planning, from sampling-based and graph-based path planners to reactive collision avoidance strategies and trajectory optimization methods. The frequent use of collision distance calculations in these applications highlights its importance as a key element of safe and efficient robot planning.

The chapter first outlines a general approach for calculating collision distance in robot systems. We present a three-step process that involves finding the position and orientation of system elements, computing minimum distances between element pairs, and identifying the overall minimum distance. We express this process mathematically,

providing a clear understanding of how collision distance is derived from the system's generalized coordinates. An important insight explored in this chapter is the relationship between the system's generalized coordinates and the resulting collision distance. This relationship forms the basis for learning-based approaches to collision distance estimation.

After this foundational discussion, we examine classical methods for collision distance calculation. These include well-known algorithms like Gilbert-Johnson-Keerthi (GJK), the Expanding Polytope Algorithm (EPA), and various Bounding Volume (BV) techniques. We discuss their principles and uses, as well as their key limitations, particularly their lack of batch processing capability and the challenges they present in calculating derivatives of collision distances.

By providing this comprehensive overview, we aim to establish a clear understanding of both the importance of collision distance in robotics and the challenges involved in its efficient computation. This foundation will be crucial as we explore more advanced and novel approaches to collision distance estimation in later sections.

2.2 Collision Distance

Collision distance is the minimum distance between a robot and any obstacles in its environment, including potential self-collisions where the robot's own parts are considered as obstacles. This calculation is crucial in various aspects of path planning. For instance, sampling-based path planners [2, 3] require collision distance calculations for numerous random configurations, while graph-based path planners [12, 13] need these calculations for every node. Furthermore, certain reactive collision avoidance strategies [5, 6, 7, 8, 9, 27, 28] and trajectory optimization methods [26] not only rely on collision distance but also require its derivative. Given its widespread use and

critical role in ensuring safe robot movement, the calculation of collision distance stands as one of the most fundamental processes in robot path planning.

To calculate the collision distance in robot systems, we assume access to a simulation environment of the target system. This environment includes the robot's kinematic parameters, obstacle positions, and the geometric shapes of both robot links and obstacles. In this context, we define an *element* as an individual rigid body in the system, such as a single link of a robot arm or a discrete obstacle. To ensure computational efficiency and maintain convexity, complex shapes are often decomposed into sets of simpler elements. The collision distance calculation for the target system follows a three-step process:

1. Determine the position and orientation of each element in the system.
2. Compute the minimum distance between all pairs of elements in the system.
3. Identify the smallest value among these computed minimum distances.

The mathematical representation of this calculation procedure is as follows:

$$\text{1.} \quad \{T_i\}_{i=1}^{M} = \texttt{ForwardKinematics}(q) \tag{2.2.1}$$

$$\text{2.} \quad d_{ij} = \texttt{Distance}(\mathcal{M}_i, \mathcal{M}_j, T_i, T_j), \quad \forall i, j \in \{1, \ldots, M\},\ i < j \tag{2.2.2}$$

$$\text{3.} \quad d_{\text{col}} = \min_{(i,j)} \{d_{ij}\} \tag{2.2.3}$$

Here:

- $q \in \mathbb{R}^n$ represents the generalized coordinate of the system where n denotes degree-of-freedom (dof).
- $T_i \in SE(3)$ represents the position and orientation of the i-th element
- M denotes the total number of elements

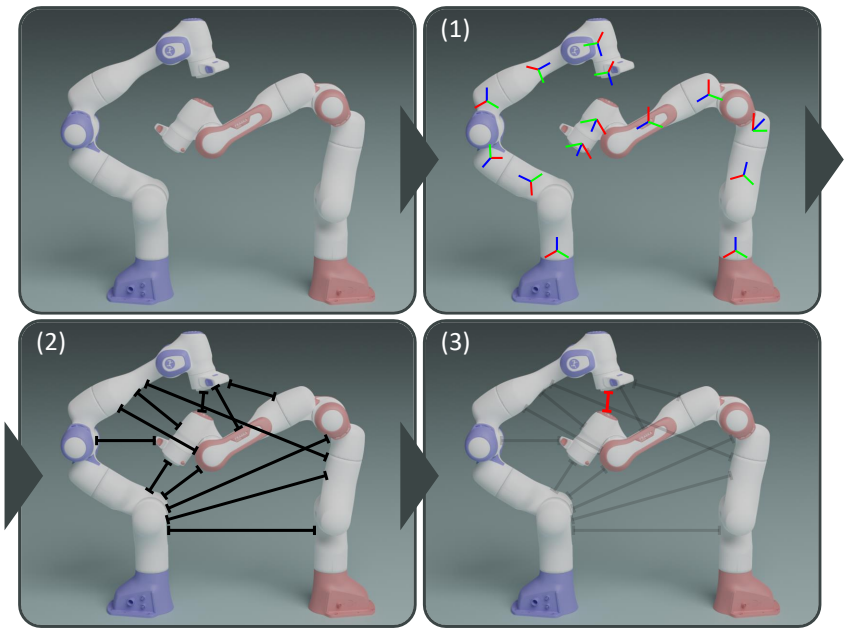

Figure 2.1: Illustration of the collision distance calculation process. The steps are: (1) Determine the position and orientation of each element. (2) Compute the minimum distances between all element pairs. (3) Estimate the collision distance as the minimum of these computed values.

- d_{ij} is the minimum distance between the i-th and j-th elements
- $\mathcal{M}_i$ represents the geometric shape data (e.g., mesh data) of the i-th element
- d_{col} is the resulting collision distance
- `ForwardKinematics` is a function that computes $\{T_i\}$ from q, incorporating forward kinematics of the robot
- `Distance` is a function that calculates the minimum distance between two geometric shapes

It's important to note that the position and orientation of each element are functions of q alone. Consequently, if the geometric shapes remain constant, the minimum distance between each pair of elements is also solely a function of q. Therefore, the collision distance of the system can be expressed as a function of the generalized coordinate q:

$$d_{\text{col}}(q) = \min_{(i,j)} \{d_{ij}(q)\}. \tag{2.2.4}$$

Thus, the collision distance of the system depends exclusively on the generalized coordinate q, making it feasible to learn a function that takes the generalized coordinate q as input and outputs the collision distance. In this thesis, we focus on environments where the dof are solely in the robot arms, so the generalized coordinate q directly corresponds to the joint configuration of the robot arm.

2.3 Classical Collision Distance Calculation

Classical methods for collision distance calculation have been widely used in robotics and computer graphics. Most of these methods rely on directly calculating the minimum distance from geometric shape information, such as triangular meshes or shape primitives.

The Gilbert-Johnson-Keerthi (GJK) algorithm [35] and the Expanding Polytope Algorithm (EPA) [36] are widely used techniques for estimating the minimum distance and penetration depth between convex shapes. These algorithms work by iteratively constructing a polytope that encloses the origin and is closest to both objects, continuing until the closest points between the polytope and the shapes converge. However, these methods are computationally expensive, particularly for complex, high-dof systems which contain numerous geometric shapes. Furthermore, they are not well-suited for batch calculations of collision distances and their derivatives.

Bounding Volume (BV) algorithms are another commonly used approach for collision distance estimation. These methods calculate the minimum distance between bounding volumes composed of simpler shape primitives, such as boxes [37], spheres [27], and capsules [8, 38]. By using these simpler shapes instead of complex geometric forms, BV algorithms simplify the calculation of minimum distances. However, these methods often struggle to find a balance between the computational efficiency and avoiding oversimplification of geometric shapes. Moreover, for complex, high-dof robot systems with numerous elements, calculating collision distances and their derivatives often remains challenging, even with these simplified approaches.

The Flexible Collision Library (FCL) [39] offers a comprehensive toolkit for collision distance calculation by unifying various algorithms such as GJK, EPA, BV, and other related methods. This versatile set of tools enables researchers and developers to address collision-related problems across diverse scenarios and geometric complexities. However, FCL lacks support for derivative calculations and does not provide GPU parallelization capabilities. These shortcomings restrict its applicability in scenarios requiring gradient-based optimization or efficient batch processing of collision distances.

The aforementioned classical methods, despite their widespread use, face significant challenges in addressing the demands of modern robotics applications. To summarize,

the key limitations include:

- Lack of batch calculation capability: These methods typically process one configuration at a time, making it inefficient to compute collision distances for multiple robot configurations simultaneously. This limitation is particularly problematic in applications that require evaluating numerous configurations, such as sampling-based path planning.

- Challenges with derivative calculation: Computing the derivative of the collision distance with respect to the robot's configuration is either impossible or computationally expensive with these classical methods. This limitation is crucial because many advanced motion planning and control algorithms, including gradient-based trajectory optimization and some reactive collision avoidance strategies, require efficient access to these derivatives.

These limitations make classical methods less suitable for applications that require processing large numbers of configurations and derivative calculations, especially in complex, high-dof robot systems.

3

Active learning of the Collision Distance Function

3.1 Introduction

For typical industrial robots operating in simple, static structured environments, motion planning has now almost been reduced to a black box technology. Sampling-based methods like RRT and its many variants [2, 3, 4] potential field methods [5, 6, 7, 8, 9] and related hybrid approaches involving graph search and optimization [10, 11] are well-established and widely used in many application settings.

At a minimum, these methods require a function that, given a robot's pose and information about obstacles in the environment, can determine whether the robot pose is collision-free or not. To do so, CAD models of the robot, obstacles, and the environment together with a simulation environment are usually assumed available. Typically the links and obstacles are approximated by boxes [37], spheres [27], capsules [38] and

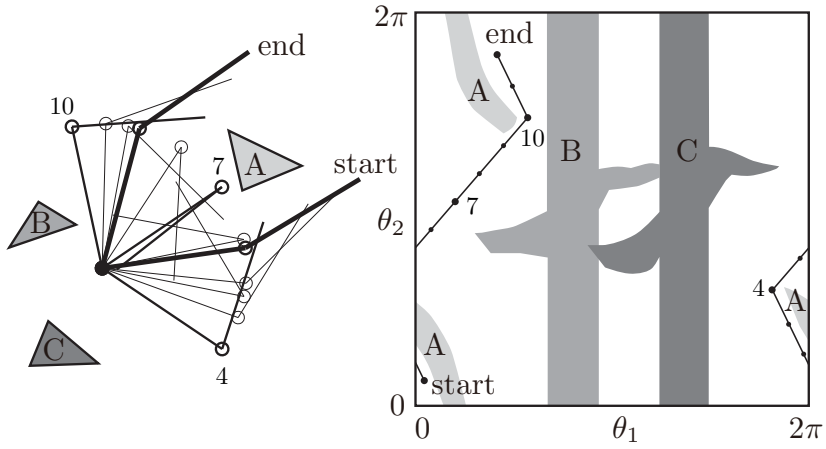

Figure 3.1: The configuration space for a 2R planar robot [1]: The robot with workspace obstacles (left), and corresponding collision regions in the configuration space (right).

other simple convex shapes [40] that simplify collision calculations, and standard computational geometric algorithms for collision checking are employed. For typical low-dof robots operating in static structured environments, these methods work well enough.

For more complex robots, however, e.g., those with higher degrees of freedom, or robots with more complex topologies like parallel robots or multiple arms manipulating a common object, even the most basic form of the motion planning problem – finding a collision-free path – can pose a formidable challenge. Usually, more information beyond a simple collision checking function is needed in order to make the problem computationally tractable, i.e., the distance between the robot and the nearest obstacle (including distances between robot link pairs, to avoid self-collisions) together with its derivative may be needed.

Evaluating the collision distance is a highly computation-intensive task that involves

finding the minimum distance between an obstacle and each of a robot's links, with distances computed in either the joint configuration space or the task space. Even for a simple planar two-link robot, the collision-free configuration space can become very complicated as shown in Figure 3.1; for robots with multi-arm or closed chain topologies, computing pairwise distances between the links and obstacles quickly becomes computationally intractable, especially as the robot degrees of freedom and the number of obstacles increases.

Motivated in part by the remarkable success of machine learning methods in robot grasping, researchers have attempted to leverage similar machine learning methods to learn the collision distance function. Given sufficient data in the form of robot configurations that are in-collision and collision-free, distance functions have been learned using a variety of machine learning models, e.g., support vector machine (SVM) models [27, 28], kernel perceptron models [26], and neural network (NN) models [8, 28, 30, 18, 31, 32]. Once the correct distance function is learned, whether or not a configuration is collision-free can now be inferred in real-time.

The main drawback with existing methods, and a significant one, is the explosion in data requirements as the degrees of freedom and number of obstacles increase. To illustrate the case of two seven-dof arms operating in a shared workspace, discretizing the joint configuration space for the combined 14-dof robot system at 10-degree intervals (which is clearly insufficient in realistic settings), and assuming a 200-degree joint range for each joint, this coarse sampling would result in more than 100 billion robot configurations ($\sim 20^{14}$).

A somewhat obvious observation is that uniform sampling of the configuration space is inefficient and in most cases unnecessary; it is much better to sample as many points as possible near the collision space boundary. The collision space boundary is of course

not known a priori, but the boundary must necessarily lie in those parts of the configuration space in which the in-collision configurations are adjacent to collision-free configurations.

Based on this observation, an active learning strategy based on sampling points near the boundary is proposed in [26]. Key to their approach is a kernel perceptron model for estimating collision distances: by using an active learning-based update strategy, the model can quickly adapt to dynamic changes in the environment. Results for low-dof systems in simple structured environments show reasonably good performance. For higher-dof robots, however, the performance of the kernel perceptron model is considerably diminished. The underlying reason is that the computational complexity of training a kernel perceptron model is $O(N^2)$, where N denotes the number of training data. Since the training data requirements for high-dof robot systems increase substantially, kernel perceptron models are not well-suited for such systems from both a computational and memory requirement perspective.

In this paper, we propose an active learning strategy for high-dof multi-arm robot systems that overcomes these limitations of existing learning-based methods. The proposed active learning-based boundary data sampling strategy samples near-boundary configurations, even in high-dimensional configuration spaces. A training dataset of fixed size is updated with these sampled configurations so that the dataset eventually consists mostly of points near the collision boundary. Another key feature of the proposed algorithm, and one that at first sight may seem somewhat paradoxical, is that by calculating distances not in the robot's configuration space or the task space, but in the larger space of link $SE(3)$ configurations – for an n-dof robot system consisting of n articulated links, this space corresponds to n copies of $SE(3)$ – the learned collision distance function is more accurate, and learning is also more robust and efficient. The redundancy in representation appears to significantly enhance the ability of a neural network to learn

the distance function.

In summary, our contributions are:

- We propose an active learning-based training method for high-dof robot systems. This method ensures that the training dataset focuses more on the near-collision boundary configurations.
- We propose a link $SE(3)$ configuration space representation, a redundant representation of the joint configuration that enables more accurate model training.
- Experimental evaluations involving high-dof robot systems verify that our approach achieves significant performance improvements compared to existing state-of-the-art methods.

In the remainder of this paper, we first discuss related works (Section 3.2) followed by the problem formulation of collision distance learning and details of the neural network model (Section 3.3). Next, we describe our novel methods (Section 3.4). To validate the proposed methodology, numerical evaluations are performed for two high-dof multi-arm robot systems (Section 3.5). We also conducted a real-world experiment using a 7-dof robot arm and obstacles to further validate the effectiveness of our approach (Section 3.6).

3.2 Related Works

A variety of techniques based on machine learning have been proposed to estimate collision distances and their derivatives. In [8], SVM classifiers are used to classify the safe or dangerous self-collision status of a humanoid robot's part pairs, and minimum distances for only the dangerous pairs are estimated using a capsule bounding volume algorithm. A similar approach is proposed in [27], in which an SVM classifier is used

to predict collision labels for a 14-dof dual-arm robot manipulation system. A neural network model was trained in [30] to predict the self-collision cost by inputting the Cartesian joint positions as a concatenated vector. Similarly, the neural network model employed in [18] utilizes the positional encoding vector of the joint configuration as its input. In [28], neural network models are trained to predict collision labels of a humanoid manipulation system. Ten sub-models are employed, each corresponding to a separate collision classifier model trained for specific sub-part pairs, such as the left arm and right leg. In [31], they take an extended configuration, which includes both joint and workspace configurations, as inputs. GraphDistNet [32] utilizes a Graph Neural Network (GNN) model to estimate collision distances, in which the shape information of the manipulator links and obstacles are represented as graphs, and the geometric relationship between the two graphs is utilized to predict the collision distance.

Similar to the approach proposed in this paper, DiffCo [26] utilizes an active learning strategy that modifies the trained collision score function to adapt to dynamic updates in the environment. DiffCo generates both the collision score and its derivative as a collision classifier model based on the kernel perceptron. A Forward Kinematics (FK) kernel that calculates distances as the Euclidean distance between points on the robot body is employed. Their active learning approach generates a dataset to address environmental changes, such as moving obstacles, by combining *exploitation* points and *exploration* points: *exploitation* points are sampled from a Gaussian distribution located near the support point of the trained kernel perceptron model, while *exploration* points are sampled from a uniform distribution within the robot's joint range limits. The effectiveness of the majority of existing methods is mostly limited to low-dof robot systems and struggles with accurately estimating collision distances for high-dof systems. A key feature of the proposed approach is that collision distance learning performance is improved through the utilization of the link $SE(3)$ configuration space representation as

an input vector. The proposed method also samples new data points near the decision boundaries of the trained model. As noted earlier, kernel perceptron models cannot handle large datasets (i.e., those on the order of a million data points), and as such these methods are not well-suited to high-dof systems with their larger training dataset requirements.

3.3 Collision Distance Learning

3.3.1 Problem Formulation

Our objective is to train a collision distance estimator model for a given target robot system, represented as a parameterized function f_θ. Initially, we assume that we have access to the ground-truth collision distance function d_{col} of the target system, which uses all the necessary information such as the robot kinematics and the geometric shapes of links and obstacles, to calculate the collision distances $d_i \in \mathbb{R}$ from joint configurations $q_i \in \mathbb{R}^N$, i.e., $d_i = d_{\text{col}}(q_i)$. The d_{col} function encompasses the forward kinematics of the robot, 3D shape information of all objects in the system, and minimum distance calculation between them (clearly, calculating $d_{\text{col}}(q)$ is computationally expensive and impractical for path planning tasks, but it is feasible for generating the dataset offline).

Next, we collect a dataset $\mathcal{D} = \{(q_i, d_i)_i\}$, containing joint configurations q_i and corresponding collision distances d_i, computed by the ground-truth collision distance function d_{col}. Finally, we optimize the parameters θ of the model f_θ by minimizing the mean-squared error (MSE) loss function L, which measures the difference between the predicted collision distances $f_\theta(q_i)$ and the actual collision distances d_i, as shown in

Equations (3.3.1) and (3.3.2).

$$\theta^* = \arg\min_{\theta} L(\mathcal{D}, f_\theta) \tag{3.3.1}$$

$$= \arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(q_i,d_i)\in\mathcal{D}} ||f_\theta(q_i) - d_i||^2 \tag{3.3.2}$$

The trained collision distance estimator model f_θ can be utilized for collision checking by setting a safety threshold ϵ. If the estimated collision distance $f_\theta(q)$ is less than ϵ, then the configuration q is regarded as in-collision, while if $f_\theta(q)$ is greater than ϵ, then the configuration q is collision-free. In the next section, we describe the structure of the proposed collision distance estimator model f_θ (Section 3.3.2), the input representation of the model (Section 3.4.1), and the active learning-based training method (Section 3.4.2).

3.3.2 Neural Network Model

When it comes to designing the form of the parametric function f_θ, there are several options to consider. Among them, neural network models are primarily preferred for f_θ in the majority of recent works, mainly because of their potential to harness big data, enable fast inference times, and their recent success in various fields. To implement the proposed approach, which are based on a novel input representation and training procedure, we have chosen to use a simple neural network architecture with fully connected layers. By keeping the neural network structure simple, we are able to focus on the benefits of the proposed approach without introducing additional complexity.

A fully connected layer, also known as a dense layer, is a layer in a neural network where each neuron is connected to every neuron in the previous and subsequent layers. The purpose of a fully connected layer is to combine the features learned in the previous layers and perform a linear transformation followed by a non-linear activation function

to generate the output of the layer. Mathematically, the output of a fully connected layer can be represented as:

$$y = \phi(Wx + b) \tag{3.3.3}$$

where x is the input vector of size n, representing the activations of the previous layer, W is the weight matrix of size $m \times n$, where m is the number of neurons in the fully connected layer and n is the number of neurons in the previous layer, b is the bias vector of size m, containing the biases for each neuron in the fully connected layer, ϕ is the activation function applied element-wise, such as the Rectified Linear Unit (ReLU) or Sigmoid function, and y is the output vector of size m, representing the activations of the fully connected layer.

Our model consists of five fully connected layers with hidden neurons of 128, each followed by a ReLU activation function. The model can be formally expressed as:

$$f_\theta(q) = (NN_\theta \circ g)(q) \tag{3.3.4}$$

$$NN_\theta(x) = W^{(5)}h^{(4)} + b^{(5)} \tag{3.3.5}$$

$$h^{(i)} = \phi(W^{(i)}h^{(i-1)} + b^{(i)}),\ i = 2, 3, 4 \tag{3.3.6}$$

$$h^{(1)} = \phi(W^{(1)}x + b^{(1)}) \tag{3.3.7}$$

where g denotes the input representation mapping, and $h^{(i)} \in \mathbb{R}^{128}$ denotes the hidden neurons of the i^{th} layer. Thus, the learnable parameter θ includes weight matrices $W^{(i)}$ and biases $b^{(i)}$. The input representation mapping g depends on user choice (e.g., the joint configuration representation [8, 28] or the position vector representation of joints [27, 30]). Our choice is detailed in Section 3.4.1.

3.4 Methods

3.4.1 Link SE(3) Configuration Space Representation

Rather than using the joint configuration as the input representation, we use the link $SE(3)$ configuration space representation for each of the links for more effective learning of the collision distance function. The mapping $g : \mathbb{R}^N \mapsto \mathbb{R}^{12N}$ represents the forward kinematics from the joint configuration space to the link $SE(3)$ configuration space for each of the N links as shown in Figure 3.2 (recall that $T \in SE(3)$ consists of a 3×3 rotation matrix R and a three-dimensional translation vector p, resulting in 12 (dependent) elements). Given a joint pose $q \in \mathbb{R}^N$, we calculate the link frame $T \in SE(3)$ for each link, then arrange each R and p for all links into a single vector $g(q) \in \mathbb{R}^{12N}$. In mathematical notation, the input vector $g(q)$ is given by

$$g(q)_{12(n-1)+4(i-1)+j} = T_{ij}^{(n)}(q). \tag{3.4.8}$$

Here, $g(q)_k$ refers to the k^{th} element of the input vector $g(q)$, while $T_{ij}^{(n)}$ represents the element located at the i^{th} row and j^{th} column of the n^{th} link frame $T^{(n)}$. The variables are defined such that n ranges from 1 to N, and i and j each range from 1 to 3 and 4, respectively.

Clearly, this representation is redundant (e.g., the third column of each R could be fully determined by the first and second columns), but we use this redundant representation instead of other alternatives, e.g., three-parameter representations or quaternion representations for the rotation matrix R. This choice is motivated by its simplicity in implementation and its superior collision distance learning performance. The link $SE(3)$ configuration space representation only necessitates a `flatten()` operation on the link frame T, thereby making the mapping g and its derivative ∇g simple to implement. Furthermore, our ablation study reveals that the proposed redundant representation shows

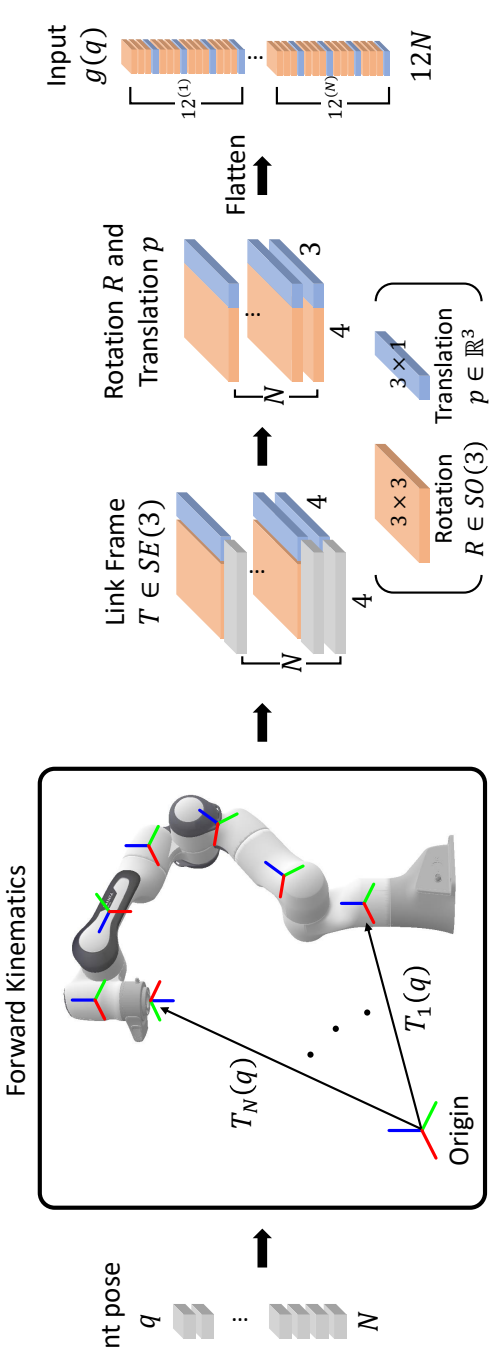

Figure 3.2: An illustration of the link $SE(3)$ configuration space representation mapping g. The joint configuration q is mapped to the collection of link frames $T_i(q)$, and transformed to an input vector $g(q)$

better learning performance than other non-redundant representations (Section 3.5.4).

3.4.2 Active Learning-based Training with Boundary Data Sampling

In this section, we describe the proposed active learning-based training method. We employ an iterative procedure to enhance the model's performance in estimating collision distances as illustrated in Figure 3.3. Starting with an initial dataset, we use this data to train the model, adjusting its parameters to minimize the loss function. Once trained, we generate new data points near the trained collision boundary, a region where $f_\theta(q)$ is close to 0. We then determine the true collision distances for the newly sampled data points using the ground-truth collision distance function, d_{col}. These new data points, along with their true collision distances, are then integrated into the current dataset. This entire process, from training the model to updating the dataset, is repeated N_{active} times. This iterative procedure enables the model to estimate collision distances more accurately as it encounters new data points near the collision boundary. Our active learning-based training procedure is detailed in Algorithm 1.

The overall process of the proposed active learning-based training method can be easily understood through a toy experiment of a 2R planar robot system, as illustrated in Figure 3.4. We create a robot environment consisting of a 2R planar robot and obstacles, along with the ground-truth collision distance function d_{col} for this system (Figure 3.4 (a)). We begin by initializing an initial dataset containing 500 data points. The initial dataset $\mathcal{D}_0$ is uniformly sampled in the joint configuration space. We then train the first model $f_\theta^{(1)}$ using this dataset. While the first model demonstrates good collision check accuracy for the training data points, it still shows an inaccurate collision boundary, as shown in Figure 3.4 (b). To improve the model, we employ an active learning-based training method that involves boundary data sampling and dataset update procedures.

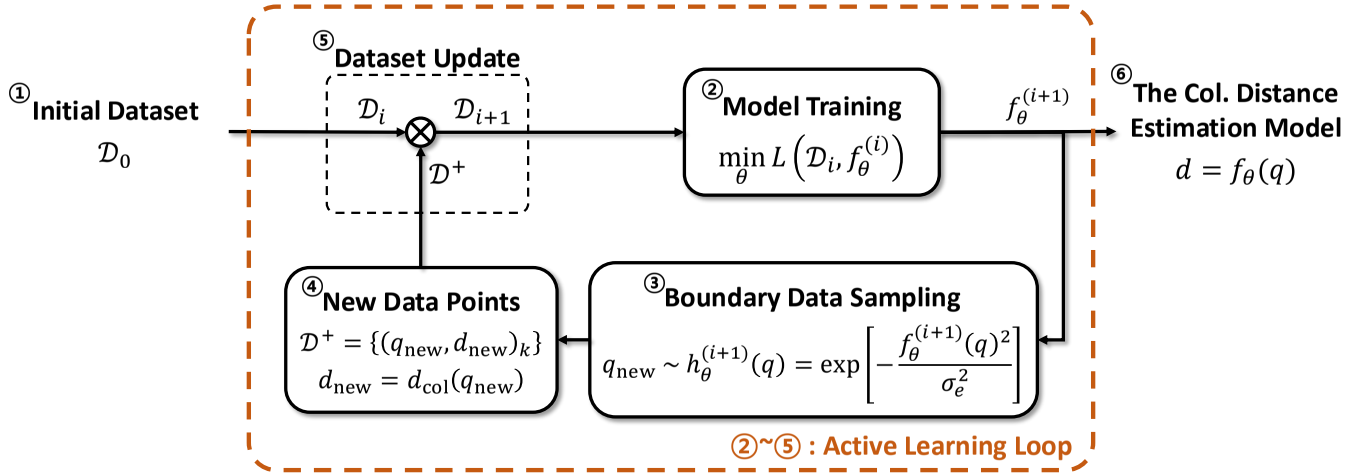

Figure 3.3: An illustration of the active learning-based training procedure. The process starts with (1) an initial dataset and proceeds to (2) train the model using this data. Once trained, (3) the model generates new data points near the trained collision boundary. (4) The true collision distances of these new data points are determined using the ground-truth collision distance function. (5) The current dataset is then updated with the newly sampled data points. By repeating this loop of training (2) and dataset updating (5), the model's performance in estimating collision distances is progressively refined, allowing for greater accuracy in the proximity of the collision boundary.

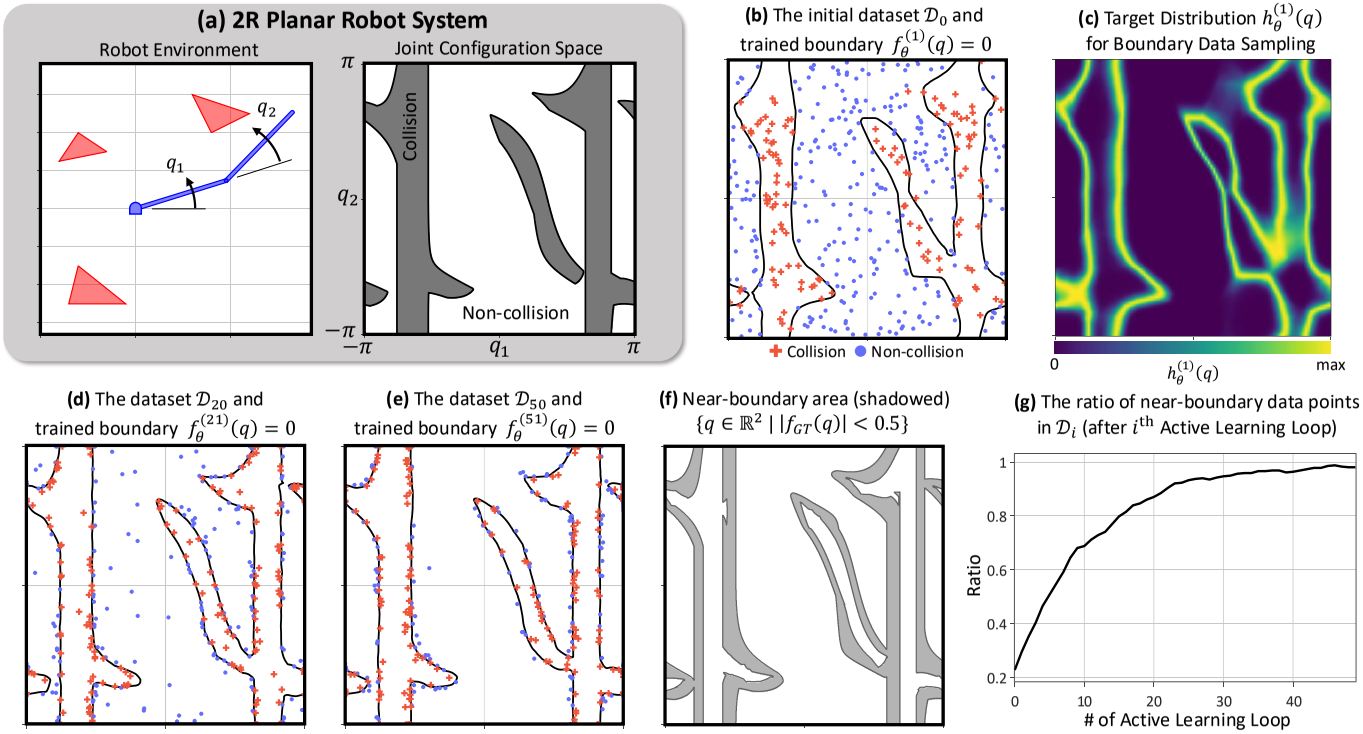

Figure 3.4: (a) A 2R planar robot system and its joint configuration space. The grey region indicates the collision area, while the remaining space represents the non-collision area. (b) The initial dataset $\mathcal{D}_0$ and the collision boundary (black line) of the model trained using the initial dataset. (c) The target distribution $h_\theta^{(1)}(q)$ for boundary data sampling. (d) and (e) depict the updated dataset and the collision boundary of the trained model after 20 and 50 iterations of the active learning loop, respectively. (g) The ratio of near-boundary data points, which are the data points in the area shown in (f), in the dataset increases with each iteration of the active learning loop.

The target distribution $h_\theta^{(1)}(q)$ for boundary data sampling of the first model is illustrated in Figure 3.4 (c), with a high sampling probability near the collision boundary of the model $f_\theta^{(1)}$. The updated dataset and trained model after 20 and 50 iterations of the active learning loop are shown in Figure 3.4 (d) and (e), respectively. As the active learning loop continues, the updated dataset increasingly includes more near-collision data points. The plot in Figure 3.4 (g) represents the ratio of near-boundary data points in the dataset, with the near-boundary region displayed in Figure 3.4 (f). Initially, only 23% of the data points are in the near-boundary region. However, this ratio converges to nearly 100% as the active learning loop progresses. Once the trained collision boundary closely approximates the ground-truth collision boundary and the majority of the dataset comprises near-boundary data points, the active learning loop no longer makes significant changes to the dataset or the model. At this stage, the data distribution of the dataset remains unchanged, and the performance of the collision distance estimation model converges. The active learning loop concludes, resulting in a model that accurately estimates collision distances near the true collision boundary. We have set the hyperparameter N_{active} to a sufficiently large value to ensure that this active learning loop can converge satisfactorily.

Boundary Data Sampling. We define a target distribution, h_θ, an unnormalized probability distribution for sampling new data points near the collision boundary:

$$q_{\text{new}} \sim h_\theta(q) = \exp\left[-\frac{|f_\theta(q)|^2}{\sigma_e^2} \right] \tag{3.4.9}$$

As shown in Figure 3.4 (c), h_θ is highest at the boundary, decreasing with distance from it. σ_e is a hyperparameter that dictates the rate of decrease in the probability with increasing distance from the boundary. We then utilize Markov Chain Monte Carlo

Algorithm 1 Active learning-based training method

Given an initial dataset $\mathcal{D}_0 = \{(q_i, d_i)_i\}$, learning rate η, updating dataset size k
Initialize $\theta \to f_\theta^{(0)}$
for $i = 0, \ldots, N_{\text{active}}$ **do**
    for $j = 1, \ldots, N_{\text{epoch}}$ **do** ▷ Model Training
        Calculate $L(\mathcal{D}_i, f_\theta^{(i)})$
        Update $\theta_{j+1} \leftarrow \theta_j - \eta \nabla_\theta L$
    end for
    Sample $q_{\text{new}} \leftarrow \text{BOUNDARYSAMPLING}(f_\theta^{(i+1)})$ ▷ Boundary Data Sampling
    Label $d_{\text{new}} \leftarrow$ collision distance calculation $d_{\text{col}}(q_{\text{new}})$
    $\mathcal{D}^+ \leftarrow \{(q_{\text{new}}, d_{\text{new}})_k\}$ ▷ Dataset Update
    $\mathcal{D}^- \leftarrow$ A subset of $\mathcal{D}_i$, consisting of k randomly chosen elements
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}^+ \cup (\mathcal{D}_i - \mathcal{D}^-)$
end for

Algorithm 2 Boundary data sampling (MCMC sampling)

function BOUNDARYSAMPLING(f_θ)
    Given a target distribution $h_\theta(q) = \exp[-\frac{|f_\theta(q)|^2}{\sigma_e^2}]$
    Initialize $q \leftarrow$ sample from UNIFORM($q_{\text{min}}, q_{\text{max}}$)
    $p_q \leftarrow h_\theta(q)$
    for $i = 1, \ldots, max\_step$ **do**
        $q^+ \leftarrow q + \epsilon, \epsilon \sim N(0, \sigma)$ ▷ Random walk
        $p_{q^+} \leftarrow h_\theta(q^+)$
        $u \leftarrow$ sample from UNIFORM($u_{\text{min}}, 1$) ▷ Relaxed rejection
        if $\frac{p_{q^+}}{p_q} > u$ **then**
            $q \leftarrow q^+$
        end if
    end for
    return q
end function

(MCMC) [41], a powerful and widely-used method for sampling from probability distributions. It is often used when we can't directly sample from a desired distribution, but we can calculate some function proportional to its density (in our case, h_θ). MCMC methods build a Markov chain of samples, where the next sample is generated based on the current one, and the chain is constructed such that its stationary distribution is the desired distribution we want to sample from. There are many variants of MCMC, here we use Metropolis-Hastings (MH) algorithm [42].

The Metropolis-Hastings algorithm starts with an arbitrary point from the joint configuration space. A new point is proposed from an easily sampled distribution, e.g., Gaussian distribution. The algorithm then decides to accept this point, based on an acceptance probability computed from the ratio of the probability densities at the proposed and current points under the target distribution h_θ. If the proposed point is less likely than the current one, it's accepted with a probability equal to this ratio. A random number u is drawn to determine whether to accept or reject the proposed point. This process repeats, resulting in a sequence of points converging to the target distribution h_θ, which is data points near the boundary. The boundary data sampling procedure is detailed in Algorithm 2.

Several critical hyperparameters influence the boundary data sampling process, necessitating careful tuning. For example, σ_e controls the target distribution shape - smaller values result in more focused sampling near boundaries, but overly narrow distributions can make sampling computationally challenging. Since the optimal hyperparameters vary with collision boundary complexity, appropriate adjustment is required for each environment. We tuned these hyperparameters by verifying that the distribution of estimated collision distances $f_\theta(q)$ from sampled points approximates the intended zero-mean Gaussian distribution for both target robot systems. Detailed tuning results are provided in Appendix A.1.

Dataset Update. After generating the new data points $\mathcal{D}^+$ with a size of k, we update the training dataset $\mathcal{D}_i$ for the next training process. We randomly select data points $\mathcal{D}^-$ from the training dataset, also with a size of k, to be removed. Subsequently, we replace $\mathcal{D}^-$ with $\mathcal{D}^+$ in order to preserve the overall size of the next training dataset $\mathcal{D}_{i+1}$. This replacement strategy is employed to prevent the continuous growth of training time in each active learning loop. By incorporating near-boundary data points and this dataset update strategy, the training dataset gradually focuses more on the near-boundary area, as demonstrated in Figure 3.4. Once the majority of training data points are located in the near-boundary area (Figure 3.4 (e)), updating the dataset with new data points does not yield substantial changes to the distribution of the training dataset. At this stage, we can conclude the active learning loop and proceed to obtain the final trained model.

Using larger values of k incorporates more new data points into the dataset, leading to faster concentration of data near boundaries and potentially accelerating performance convergence across active learning iterations. However, this increased data usage results in higher computational costs, making it important to tune k appropriately. Our ablation study in Appendix A.1 shows that while increasing k leads to modest performance improvements and slightly faster convergence, these gains appear to be merely a result of using more data points. This approach of improving performance simply by using more data diverges from our paper's focus on finding better data distributions under limited computational resources. Therefore, we did not use higher k values in our main experiments. Nevertheless, if the goal is to achieve better performance and more computational resources are available, using higher k values remains a viable option.

3.5 Learning Performance Evaluation

3.5.1 Evaluation Setting

Proposed Methods and Baselines. In this section, we train and compare various models for collision distance learning. These models include the methods proposed in this paper as well as existing collision distance learning methods used as the baseline. The trained models are as follows:

- **JointNN**: A neural network model that directly takes the joint configuration as inputs (the input representation used in [8, 28, 30]).
- **PosNN**: A neural network model that takes the Cartesian coordinates of the joint positions as inputs (the input representation used in [27, 30]).
- **ClearanceNet** [31]: A neural network model composed of two fully-connected layers with each followed by a dropout layer, taking the joint configuration as inputs.
- **DiffCo** [26]: A kernel perceptron model with FK kernel, which inputs the joint configuration and outputs the collision score.
- **SE3NN** (ours): A neural network model that takes the link $SE(3)$ space representation as inputs.

To compare the effectiveness of the proposed link $SE(3)$ configuration space representation with other input representations, we implement JointNN and PosNN with the same network structure as our model (SE3NN) as described earlier in Section 3.3.2. DiffCo and ClearanceNet, which are existing learning-based collision estimator models, are also used as baselines. Details on implementing DiffCo and ClearanceNet are consistent with those specified in [26, 31].

We employ two distinct training procedures for the neural network models: the training procedure denoted as *active* is the proposed active learning-based training procedure described in Section 3.4.2, while the training procedure denoted *none* corresponds to the standard training procedure using only the initial training dataset (without active learning). DiffCo also employs an active learning-based boundary data sampling strategy that is specific to the kernel perceptron model. The active learning-based training procedure of DiffCo models that we implement are denoted *DiffCo*; these are similar to our framework, the key difference being that the boundary data sampling strategy is replaced with DiffCo's strategy.

Target Environment and Datasets. We select two high-dof multi-arm robot manipulation systems as our target environment as shown in Figure 3.5. The first system features two Franka-Emika Panda robot arms, each with 7-dof, placed 0.6 meters apart and facing each other. The second system features three Panda robot arms arranged equidistantly on a circle of radius 0.5 meters centered at the origin, each facing towards the center.

For each target environment, we generate a collision distance dataset $\mathcal{D} = \{(q_i, d_i)_i\}$ by randomly sampling joint configurations q_i from a uniform distribution within the joint limits, and calculating the corresponding collision distance d_i using mesh-based collision distance calculation methods [35]. Using the collision distance calculation algorithm implemented in PyBullet [43], a Python wrapper for the Bullet physics engine library, we generate 1 million data points for both the training and test dataset. For training the DiffCo model, we use 75,000 data points with boolean collision labels (-1 for collision-free and 1 for collision). This dataset size is the maximum manageable dataset size for kernel perceptron training on our computing hardware (AMD Ryzen Threadripper 3960X, 256GB RAM, and NVIDIA GeForce RTX4090 with 24GB VRAM).

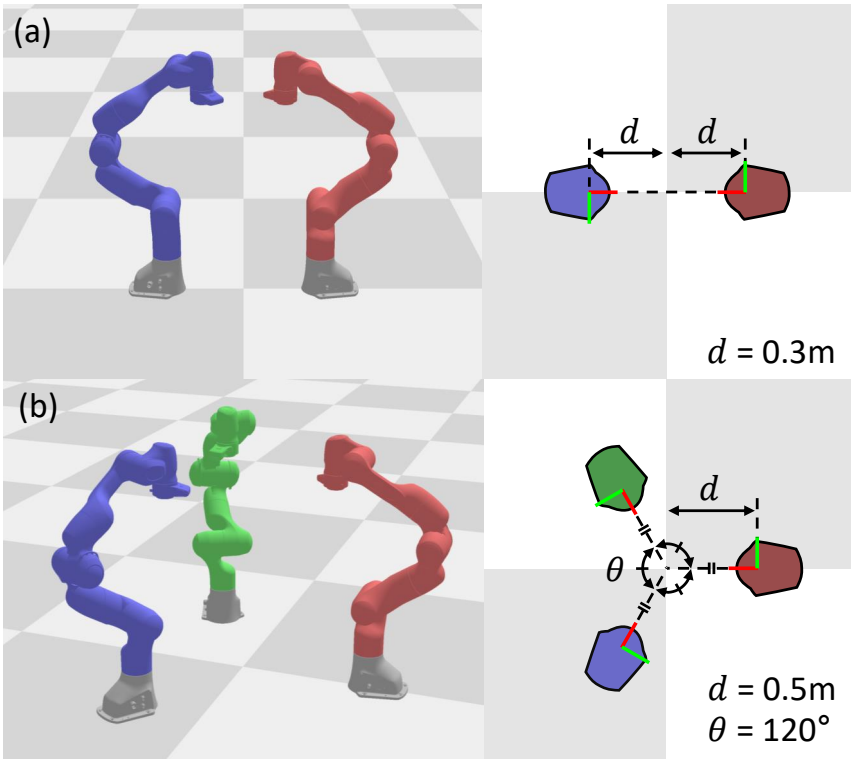

Figure 3.5: An illustration of the target multi-arm systems. (a) Two 7-dof Franka-Emika Panda robot arms resulting in a 14-dof system. (b) Three 7-dof robot arms resulting in a 21-dof system.

When employing our active training procedure, we replace k samples with newly generated data points at each active learning iteration, resulting in $k \times N_{\text{active}}$ additional data points throughout the training process, where N_{active} denotes the number of active learning iterations. However, the performance improvement of our model is not merely attributed to using more data points. Rather, it is primarily due to generating new data points near the collision boundary regions, which have a more significant impact on model performance. While updating the dataset with uniformly sampled points within

joint limits shows modest improvement, our active methodology demonstrates superior performance gains by sampling near collision boundaries. Detailed experimental results comparing these approaches are presented in the Appendix A.2.

Evaluation Metrics. In this section, we present the numerical metrics used to evaluate the performance of our collision distance learning models. Since collision distance estimation models are typically used for collision label classification, we evaluate classification performance using two metrics: *accuracy* with the threshold $\epsilon = 0.0$, and *Area Under the Receiver Operating Characteristic* (AUROC). AUROC is a widely used performance metric for classification tasks that measures the model's ability to distinguish between different classes. It quantifies the overall performance of a classifier by calculating the area under the ROC curve, which represents the trade-off between the true positive rate and the false positive rate at various classification thresholds. AUROC provides a single scalar value that summarizes the classifier's discriminative power across all possible thresholds, making it robust to threshold selection. Higher AUROC values indicate better classification performance, with a perfect classifier achieving an AUROC of 1. We also report the *near-accuracy* and *near-AUROC* metrics, which evaluate the classification performance only on data points in the test dataset with collision distances within the range of -0.1m to 0.1m, since accurately classifying collisions in the close-to-collision region is critical for actual robot manipulation tasks.

3.5.2 Evaluation Results

We conduct a comparative analysis of the collision classification performance of our methods against baseline models. Table 3.1 presents the main results, highlighting the effectiveness of the active learning-based training framework and the link $SE(3)$ configuration space representation. The top-performing metrics for each target environment

are indicated in bold, all of which are achieved by the SE3NN model trained with the *active* procedure.

Compared to other input representations like the joint configuration and the Cartesian coordinates of joint positions, the use of the link $SE(3)$ configuration space representation results in a significant improvement in collision classification performance in both robot systems. Furthermore, our SE3NN model demonstrates better performance than other baseline models, ClearanceNet and DiffCo, which both use the joint configuration as inputs.

The proposed active learning-based training framework also demonstrates improved performance compared to the *none* training procedure. Although the improvements in terms of accuracy and AUROC may seem modest, the near-accuracy and near-AUROC metrics show significant improvement compared to the baseline models. This indicates that our method enables the model to make more accurate collision classifications in the close-to-collision region, where collision classification is critical. In contrast, the classification performance metrics of the DiffCo model are even worse in some cases when the *DiffCo* training procedure is applied. This may be due to the fact that the boundary data sampling strategy of DiffCo targets the update of the kernel perceptron model for dynamic changes in the environment for low-dof systems, rather than constructing datasets for high-dof robot systems that require over a million data points for collision distance learning.

Table 3.1: Collision distance estimation performance comparison: SE3NN with active learning-based training versus baseline methods

Env.	Model	Training	Accuracy (ϵ=0.0)	AUROC	near-Accuracy (ϵ=0.0)	near-AUROC
Fig. 3.5(a) (Two arms)	JointNN	*none*	0.9765	0.9942	0.8799	0.9491
		active	0.9809	0.9960	0.9018	0.9643
	PosNN	*none*	0.9810	0.9965	0.9025	0.9681
		active	0.9828	0.9971	0.9117	0.9735
	ClearanceNet	*none*	0.9627	0.9855	0.8128	0.8845
		active	0.9707	0.9900	0.8516	0.9186
	DiffCo	*none*	0.9541	0.9828	0.7786	0.8641
		DiffCo	0.9328	0.9827	0.7024	0.8674
	SE3NN (ours)	*none*	0.9862	0.9981	0.9293	0.9830
		active	**0.9886**	**0.9987**	**0.9416**	**0.9877**
Fig. 3.5(b) (Three arms)	JointNN	*none*	0.9691	0.9868	0.8189	0.8875
		active	0.9765	0.9918	0.8609	0.9280
	PosNN	*none*	0.9799	0.9943	0.8808	0.9485
		active	0.9821	0.9955	0.8937	0.9587
	ClearanceNet	*none*	0.9443	0.9507	0.6944	0.7202
		active	0.9532	0.9619	0.7377	0.7746
	DiffCo	*none*	0.9453	0.9589	0.7019	0.7557
		DiffCo	0.9447	0.9600	0.7105	0.7642
	SE3NN (ours)	*none*	0.9826	0.9958	0.8969	0.9619
		active	**0.9858**	**0.9971**	**0.9157**	**0.9738**

Table 3.2: Training time, inference time, and GPU memory requirements: Comparison of SE3NN and baseline methods

Model	Fig.3.5(a) (Two arms)			Fig.3.5(b) (Three arms)			Required GPU memory (GB)
	Training time (h)	Inference time (ms)		Training time (h)	Inference time (ms)		
		CPU	GPU		CPU	GPU	
JointNN	1.18	0.076	0.186	1.26	0.079	0.184	1.364
PosNN	16.4	0.079	0.183	23.9	0.076	0.186	1.378
ClearanceNet	34.9	0.233	0.139	44.9	0.239	0.148	1.384
DiffCo	0.13 / 2.96*	0.128	0.097	0.22 / 6.83*	0.878	0.098	23.166
SE3NN	16.9	0.082	0.181	23.8	0.079	0.189	1.378

*In case of Diffco, training time when utilizing the none / DiffCo training procedure.

3.5.3 Time and Memory

The results of measuring the training time, inference time and the required GPU memory of the models are presented in Table 3.2. The measured training times are all based on using the proposed *active* training procedure. To ensure a fair performance comparison, we also trained using the *none* training procedure for the same number of epochs as the *active* training procedure ($N_{\text{active}} \times N_{\text{epoch}}$). Thus, there is no significant difference in the training time between the two training procedures. For Diffco, we have provided both training times in the table as there was a difference in the training time between the *none* training procedure and the *DiffCo* training procedure. The required GPU memory was measured as the maximum GPU memory utilized during the model training.

Training PosNN and SE3NN requires more time compared to JointNN because these

two models involve calculating the robot's forward kinematics during the training process. For ClearanceNet, we set the batch size to the value used in [31], which is 191. Hence, compared to other models with a batch size of 10,000, ClearanceNet requires more gradient steps (assuming the same number of epochs), resulting in longer training times. The training time for DiffCo is notably shorter compared to other neural network-based models. However, it is due to the fact that we utilized only 7.5% of the training data points for training DiffCo models. The kernel perceptron model's training process involves computing the distance between all training data points, which requires $O(N^2)$ memory where N is the number of training data points. Consequently, this limitation hinders the use of large datasets. In our DiffCo model implementation, we were able to utilize a maximum of 75,000 data points under the 24GB graphics processing unit memory constraint.

The inference time represents the average time taken by each model to perform collision distance estimation 10,000 times in repetitions. Both CPU and GPU execution scenarios were considered separately. Due to the simplicity of the neural network architectures and the limited number of layers used, all models achieved inference times within 1ms. This level of performance is compatible with a 1kHz control frequency, ensuring efficient real-time operation.

3.5.4 Compared to Other Representations of SE(3)

In the proposed approach, we utilize the redundant representation of $SE(3)$, which involves the 12-dimensional flattened vector of a rotation matrix and a translation vector. To evaluate the effectiveness of our redundant representation, we compare our approach to other representations, namely quaternions and Euler angles. In mathematical notation,

Table 3.3: Performance of other non-redundant input representations

Env.	Model	Accuracy (ϵ=0.0)	AUROC	near-Accuracy (ϵ=0.0)	near-AUROC
Fig. 3.5(b) (Three arms)	QuatposNN	0.9819	0.9955	0.8930	0.9586
	EulerposNN	0.9818	0.9955	0.8923	0.9586
	SE3NN	**0.9826**	**0.9958**	**0.8969**	**0.9619**

we introduce two distinct mappings: $g_{\text{quat}} : \mathbb{R}^N \mapsto \mathbb{R}^{7N}$ representing the quaternion representation and $g_{\text{Euler}} : \mathbb{R}^N \mapsto \mathbb{R}^{6N}$ for the Euler angles representation. The input vectors for these mappings are given by:

$$g_{\text{quat}}(q)_{7(n-1):7n} = [\texttt{to\_quaternion}(R^{(n)}(q)), p^{(n)}(q)] \in \mathbb{R}^7, \tag{3.5.10}$$

$$g_{\text{Euler}}(q)_{6(n-1):6n} = [\texttt{to\_Euler\_angles}(R^{(n)}(q)), p^{(n)}(q)] \in \mathbb{R}^6. \tag{3.5.11}$$

Here, $R^{(n)}$ denotes the rotation matrix and $p^{(n)}$ the translation vector for the n^{th} link frame, represented by $T^{(n)}$. QuatposNN utilizes $g_{\text{quat}}(q)$, while EulerposNN employs $g_{\text{Euler}}(q)$ for their input mappings.

Table 3.3 shows the comparison results. The performances of these different representations do not show a big performance gap; however, it is noteworthy that the proposed redundant representation exhibits slightly better performance compared to the other representations.

3.6 Real-world Experiments

To validate the performance of our collision distance estimation, we conducted real-world experiments using a 7-dof Panda robot arm (Figure 3.6). The workspace was designed with complex obstacles like shelves and tables, introducing complexity to the

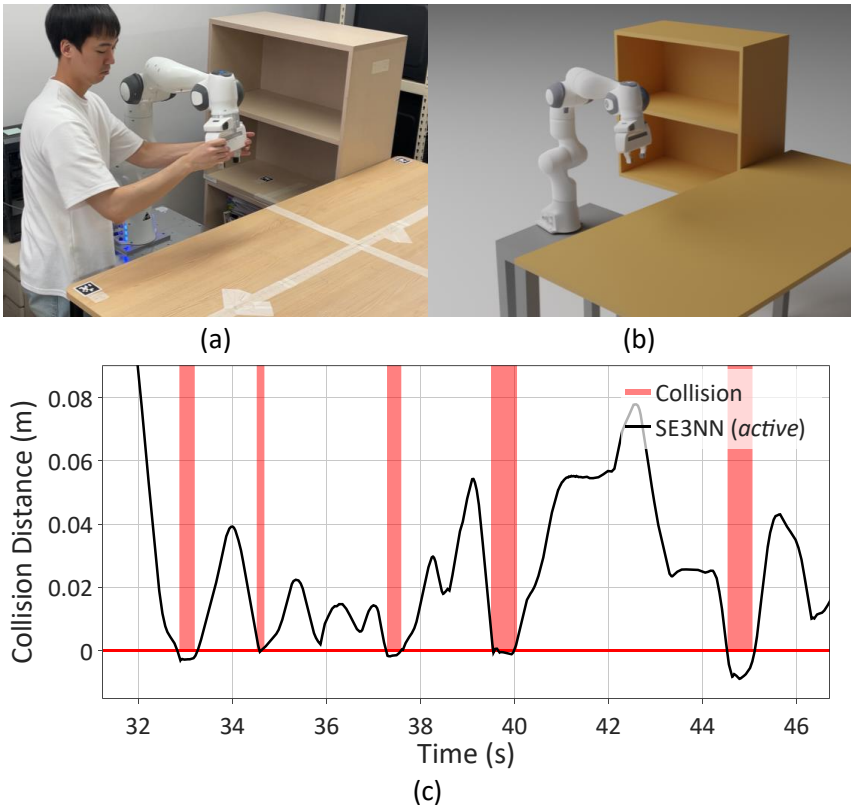

Figure 3.6: An illustration of the real robot experiment. (a) A 7-dof single arm robot system with obstacles and (b) the corresponding simulation environment. (c) The plot demonstrates the collision labels and estimated collision distances of the proposed model (SE3NN with the active training procedure)

robot arm's workspace. To safely generate close-to-collision trajectories, we conducted a human-guided demonstration as illustrated in Figure 3.6 (a). During this demonstration, the robot arm followed guided trajectories that came close to, and occasionally collided with, various parts of the obstacles. The guided trajectory encompassed motions that closely swept the inner space of the shelves and the top of the tables.

We employed the *active* training procedure to train the SE3NN model, following the same data generation and model training process as described in Section 3.4 and 3.5. To calculate the ground-truth collision distance of the dataset, we constructed a simulation environment for the real-world robot system, as illustrated in Figure 3.6 (b). The collision distance estimated by the trained SE3NN model and the real-world collision labels are plotted in Figure 3.6 (c). The plot demonstrates that the model successfully predicts negative collision distance when the robot collides with obstacles. Throughout the entire trajectory, we conducted collision distance estimation with a frequency of 1kHz, resulting in a total of 84,668 collision distance estimations. The proposed model exhibited a collision classification accuracy of 95.9% with a threshold of $\epsilon = 0.0$.

3.7 Conclusion

In this paper, we have presented an active learning strategy for learning the collision distance function for high-dof multi-arm robot systems operating in complex environments. The proposed method rests on two key ideas – an active learning-based training method with a boundary data sampling strategy that efficiently provides near-boundary data points for the training dataset, and using the link $SE(3)$ configurations as inputs to the model instead of the usual joint configurations. Our methods enable the model to accurately learn complex collision distance functions for high-dof robot systems. We validate our approach on two high-dof multi-arm robot systems with two and three 7-dof

robot arms, respectively. Our results show significant improvement in collision classification performance over the existing state-of-the-art. The SE3NN model with the *active* training procedure, which is the model to which both the active learning-based training method and the link $SE(3)$ configuration space representation are applied, demonstrate the best performance in every performance metric.

The proposed methods successfully demonstrate novel collision distance estimation performance when applied to a static environment, but if the environment changes, e.g., changing the robot base position, the training process must be repeated. In future work, we will address this problem through transfer learning or other learning-based methods; it may be possible to quickly fine-tune or retrain the model with some new data points for different environments.

4

PairwiseNet: Pairwise Collision Distance Learning

4.1 Introduction

Motion planning algorithms such as RRT [2, 3, 4] and its many variants [6, 7, 8, 9] all require the *collision distance* - the minimum distance between the robot and its nearest obstacle (including other links for self-collision avoidance). Among these, some even require the derivatives of the collision distances. It is well-known that calculating this distance involves finding the minimum distance between each robot link and the obstacles, which can be computationally intensive, especially for high-dof robots with complex geometries.

To alleviate the computational burden, one possible solution is to train a collision distance function using data. By collecting sufficient data consisting of robot configurations and their corresponding collision distances, machine learning models such as kernel perceptron models [26], support vector machines (SVM) [27], and neural networks

[18, 28, 30, 31, 32], can be used to learn the collision distance function. This learned function can then be used to quickly determine if a given configuration is collision-free. While these data-driven approaches have demonstrated satisfactory results for low-dof robot systems, often they perform poorly for higher-dof robots. The challenge lies in the fact that the collision distance function for higher-dof robots is complex and highly non-convex.

Another challenge faced by existing data-driven methods is their sensitivity to small environmental changes. For example, the addition of new obstacles or a change of the robot's base position can lead to a completely different collision distance function; for many of these methods, the entire training procedure must be repeated, from data collection to model training. (One possible exception is [26], which proposes an efficient model update strategy for dynamic environment updates, but their method is limited to low-dof robot systems and still requires an additional training procedure.)

We present PairwiseNet, a collision distance estimation method that provides a promising alternative to existing data-driven approaches used for predicting the *global* collision distance. Instead of directly estimating the global collision distance, PairwiseNet focuses on estimating the *pairwise* collision distance: the minimum distance between two elements in the robot system. The PairwiseNet model takes as input the point cloud data of two geometric shapes and their relative transformation and outputs the minimum distance between these two shapes. To estimate the global collision distance, PairwiseNet first predicts the minimum distances for every possible pair of elements in the system. It then selects the minimum of these pairwise distances as the estimate for the global collision distance (see Figure 4.1). The efficient parallel batch computation of the neural network enables the rapid prediction of minimum distances between pairs of elements.

Compared to the complex and highly non-convex function of the global collision distance, the minimum distance function between a pair of elements is simpler and

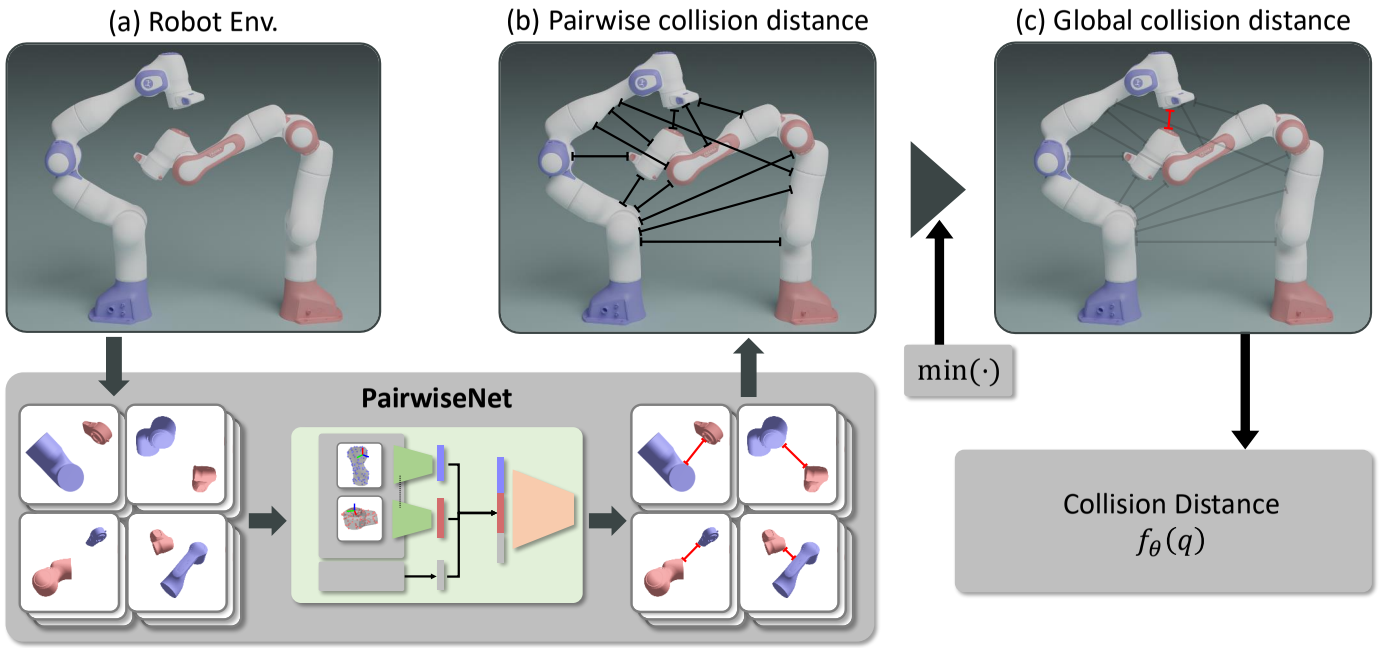

Figure 4.1: An illustration of the global collision distance estimation through PairwiseNet. (a) Robot environment at a given joint configuration. (b) Pairwise collision distances for all element pairs are determined through PairwiseNet. (c) The smallest of these distances becomes the global collision distance.

easier to train. By breaking down the challenging task of learning the global collision distance into smaller sub-problems of the pairwise collision distance learning, our PairwiseNet achieves significant performance improvements for high-dof robot systems.

Another advantage of PairwiseNet is its applicability to any system composed of known shape elements (shape elements that are sufficiently trained for estimating pairwise collision distance). The trained PairwiseNet model can be used without the need for additional training or modifications in such systems. For example, consider a scenario in which a sufficiently large dataset containing pairwise collision distances between the links of a Panda robot is available. In this case, the PairwiseNet model trained using this dataset can be applied to any system consisting of multiple Panda robots, regardless of the number of robots or their respective positions, as this is possible because the collision distance estimation for such systems can be broken down into pairwise collision distance estimations for each element pair, and these pairwise distances are already known by the trained PairwiseNet model. As long as the system is exclusively comprised of shape elements that have been learned during training, the trained PairwiseNet model is applicable to any such system. Even in cases where the system undergoes changes, such as changing the robot's base position or adding another robot arm, if the geometric shape elements of the system remain unchanged, the trained PairwiseNet model remains applicable to the changed system without any modifications.

Our approach has been evaluated in high-dof multi-arm robot manipulation systems, ranging from the two-arm (14-dof) to four-arm (28-dof) systems, as well as a single-arm robot with obstacles. The results demonstrate that our approach outperforms existing learning-based methods in terms of collision distance regression error, collision checking accuracy, and notably the False Positive Rate with the collision-free guaranteed threshold (Safe-FPR). Moreover, our approach performs better even when using a single trained PairwiseNet model for all multi-arm systems.

4.2 Related Works

Several machine learning-based methods for collision distance estimation have been proposed due to their computationally efficient inference procedures for collision distances and derivatives. [8] used SVM classifiers to identify whether each pair of parts of a humanoid robot was in a safe or dangerous self-collision status given a specific joint configuration. Only the minimum distances of the dangerous pairs of parts were estimated using a capsule-based BV algorithm, simplifying the calculation of collision distances and derivatives. [27] also employed an SVM classifier for a 14-dof dual-arm robot manipulation system. The SVM classifier inputs a vector consisting of the positions of all joints in the system and outputs a collision label of either <1 for a collision or >1 for a collision-free state. In [28], SVM and neural network models were trained to predict the collision label of a humanoid manipulation system at a given joint configuration. Separate collision classifier models were trained for every sub-part pairs, such as the left arm and right leg, resulting in a total of 10 sub-models used for collision label predictions.

Similar to [27], [30] utilized joint positions as inputs for their multi-layer perceptron neural network model. Meanwhile, [18] employed a positional encoding vector of the joint configuration as input for their neural network model. [31] trained a neural network model to estimate the collision distance using an extended configuration containing both joint and workspace configurations as input, with the model outputting the collision distance of the system. DiffCo [26] is a collision classifier model based on kernel perceptron that generates both the collision score and its derivative. DiffCo also utilizes an efficient active learning strategy that adjusts the trained collision score function for dynamic updates in the environment. Similarly, CollisionGP [29], a Gaussian process-based collision classifier model, has been proposed. CollisionGP determines the

collision query for a given joint configuration and also measures the uncertainty of the model in its prediction. Recently, GraphDistNet [32] was proposed as a Graph Neural Network (GNN) model for collision distance estimation. The model inputs the information on geometric shapes, which are represented as graphs for both the manipulator links and obstacles. GraphDistNet then utilizes the geometric relationship between the two graphs to predict the collision distance.

Similar to our method, some works [44, 45] approached the challenge by decomposing a complex problem into several simpler sub-problems. [44] proposed a novel configuration space decomposition method. This method separates the robot into disjoint components and trains a classifier for the collision-free configuration space of each component. Since the components near the base link have a relatively low-dimensional configuration space, training classifiers for these components is easier than training a single classifier for the whole system. [45] trained a collision predictor for generating collision-free human poses. They focused on the fact that collisions only affect local regions of the human body. Therefore, they designed a set of local body parts, and the collision prediction was accordingly decomposed into these local parts.

The effectiveness of these existing methods has been experimentally demonstrated only for low-dof robot systems; their performance degrades substantially for high-dof systems operating in complex environments. In particular, most learning-based collision distance estimation methods establish a collision-free guaranteed threshold to ensure that no collisions occur during actual manipulation. However, existing methods often suffer from a high false positive rate, resulting in overly cautious collision detection when utilizing the collision-free guaranteed threshold. In comparison, our method demonstrates effective collision distance estimation performance even in high-dof robot systems and maintains low false positive rates when using the collision-free guaranteed threshold.

4.3 Learning Pairwise Collision Distance

4.3.1 Problem Formulation

We assume the availability of a simulator environment of the target system, which includes the robot kinematics and geometric shapes of links and obstacles. We aim to determine the optimal model parameter ψ for the pairwise collision distance estimation model f_ψ, which can predict the collision distance between any pair of geometric shapes. The model takes the point cloud data of two geometric shapes $\mathcal{P}_i$, $\mathcal{P}_j$ (expressed in each corresponding object coordinates) and the relative transformation $T_{ij} \in SE(3)$ as input, and outputs the estimated pairwise collision distance $\hat{d}_{ij}$ between the two shapes.

$$\hat{d}_{ij} = f_\psi(\mathcal{P}_i, \mathcal{P}_j, T_{ij}) \tag{4.3.1}$$

After training the model, the global collision distance can be determined by the procedure as shown in Figure 4.2. First, a set of element pairs and corresponding transformations $\mathcal{S}(q) = \{(\mathcal{P}_i, \mathcal{P}_j, T_{ij}(q))\}_{i,j}$ in the given joint configuration q is extracted from the target robot system. Next, PairwiseNet determines the pairwise collision distance between each element pair in $\mathcal{S}(q)$, and the minimum distance found among these is taken as the global collision distance of the robot system. The global collision distance estimator function F_ψ can be expressed in the form of

$$\hat{d}_{\text{col}}(q) = F_\psi(q; f_\psi, \mathcal{S}) \tag{4.3.2}$$

$$= \min_{(\mathcal{P}_i, \mathcal{P}_j, T_{ij}(q)) \in \mathcal{S}(q)} f_\psi(\mathcal{P}_i, \mathcal{P}_j, T_{ij}(q)) \tag{4.3.3}$$

where $\hat{d}_{\text{col}}(q)$ is the estimated global collision distance in the joint configuration q. Using the batch computation of the neural network model, we can efficiently estimate the minimum distances of element pairs.

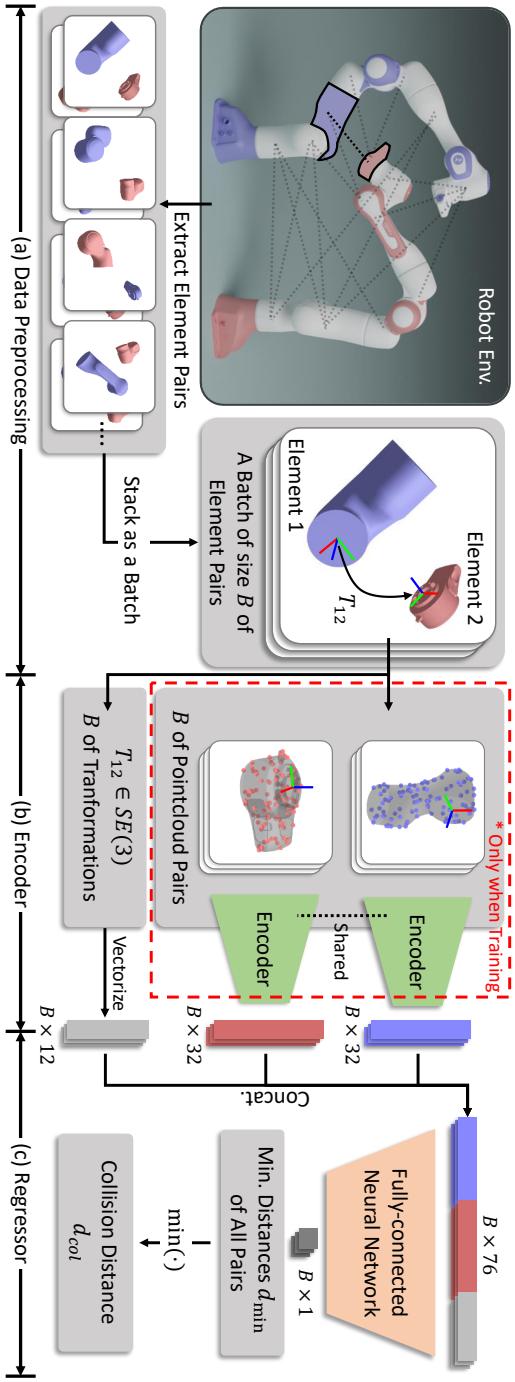

Figure 4.2: An illustration of estimating the global collision distance via PairwiseNet.

4.3.2 Strategy for Decomposing System Elements

PairwiseNet achieves robust collision distance estimation performance by dividing the robot system into multiple elements and calculating the pairwise collision distance between these elements. Therefore, to apply PairwiseNet, the robot system must be divided into these elements as a preliminary step. Each element must be a rigid body with an unchanging shape, and in the case of the multi-arm systems in Section 4.4, simply each link was treated as a separate element. Since PairwiseNet does not require each element to be convex, non-convex links of the Panda robot arm can be used without additional decomposition.

However, PairwiseNet's superior performance is attributed to the fact that the pairwise collision distance functions between elements are much easier to learn than the global collision distance function. Therefore, if an individual element's shape is complex and highly non-convex, learning the corresponding pairwise collision distance may become difficult, potentially diminishing PairwiseNet's effectiveness. Hence, a well-considered balance must be found between the complexity of decomposing the system and the performance of PairwiseNet. In our experiments (see Section 4.5), we decompose obstacles into several rectangular shape elements. Detailed information about this process is provided in Section 4.5.1.

4.3.3 Network Architecture

PairwiseNet consists of two main components: an *encoder* that creates a shape feature vector from a point cloud data of a geometric shape, and a *regressor* that predicts the minimum distance between two shape feature vectors and a transformation (Figure 4.3). The encoder employs two EdgeConv layers from Dynamic Graph Convolutional Neural Network [46] to extract 32-dimensional shape feature vectors from the point cloud data.

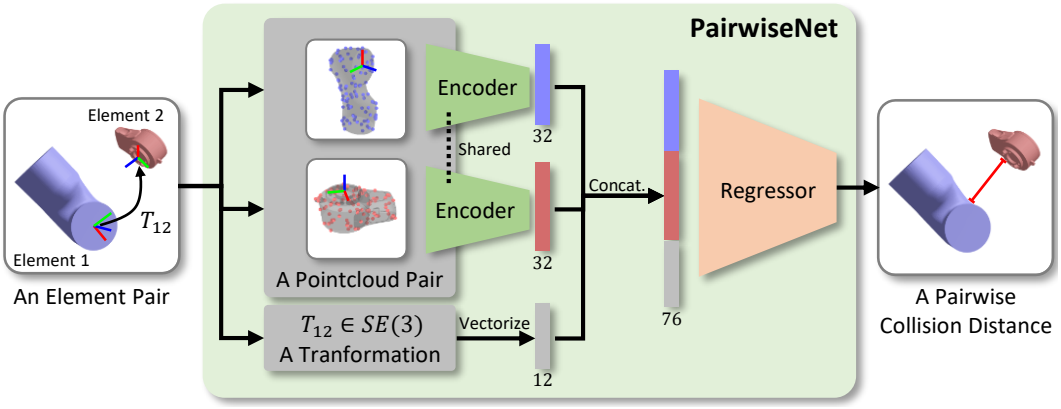

Figure 4.3: An illustration of the architecture of PairwiseNet.

The regressor then combines the two shape feature vectors and the transformation into a single vector and uses four fully connected layers with hidden state dimensions of $(128, 128, 128)$ to output the minimum distance. The training of PairwiseNet uses the mean-squared error (MSE) between the estimated and actual collision distances as the loss function

$$L = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathcal{P}_i, \mathcal{P}_j, T_{ij}, d_{ij}) \in \mathcal{D}_{\text{train}}} ||f_\psi(\mathcal{P}_i, \mathcal{P}_j, T_{ij}) - d_{ij}||^2 \tag{4.3.4}$$

where $d_{ij} \in \mathbb{R}$ is the ground-truth pairwise collision distance, and $\mathcal{D}_{\text{train}}$ denotes the training dataset.

4.3.4 Efficient Inference Strategy of PairwiseNet

Our approach includes an efficient inference strategy for the global collision distance calculation by eliminating the need to run the encoder, a deep neural network that transforms the point cloud data into feature vectors. Since the point cloud data of element pairs remains unchanged regardless of the joint configuration, shape feature vectors of

element pairs can be calculated and saved once for each robot system before calculating the collision distance. Using these pre-calculated shape feature vectors, PairwiseNet is able to estimate the collision distance only using the regressor, a simple neural network composed of fully-connected layers. Implemented in PyTorch [47], PairwiseNet is capable of performing collision distance estimation for the joint configuration in less than 0.5ms. Details on the inference time for PairwiseNet can be found in Section 4.6.

4.4 Collision Distance Learning for Multi-arm Robot Systems

4.4.1 Experimental Setting

Training Dataset for PairwiseNet. We collected pairwise collision distance data from dual-arm robot manipulation systems. For the diversity of the dataset, we utilize dual-arm robot systems with various relative positions between the two arms as illustrated in Figure 4.4. We sample θ and ϕ with eight equally spaced values within the range $[0, 2\pi)$, and sample R with five equally spaced values within the range $[0.1\text{m}, 1.0\text{m}]$. In total, we use 320 different combinations of (R, θ, ϕ), resulting in 320 different dual-arm robot systems. For each system, we sampled joint configurations uniformly within the joint limits and extracted a set of element pairs $\mathcal{S}(q)$ at each joint configuration q (Figure 4.2(a)). To obtain the ground-truth pairwise collision distance $d_{ij} \in \mathbb{R}$ between the element pair, we use the collision distance estimation algorithm implemented in PyBullet [43]. If the two elements collide, the collision distance is the negative of their penetration depth (the distance by which one convex object enters into the interior of another during a collision [36]). The resulting training dataset $\mathcal{D}_{\text{train}}$ contains 3 million data points.

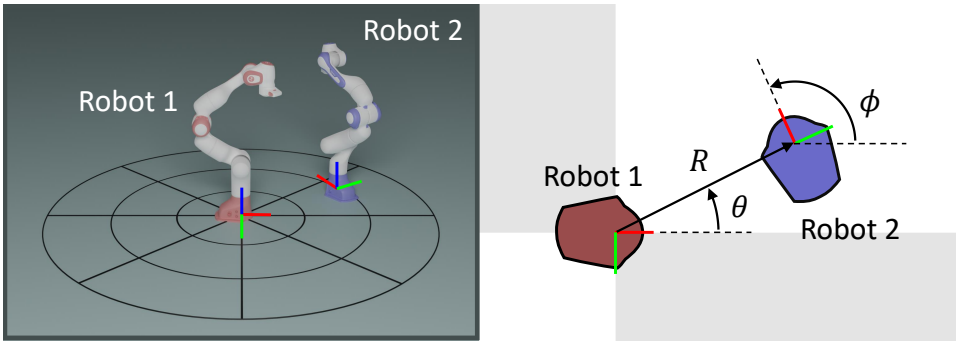

Figure 4.4: An illustration of multi-arm robot systems for generating the training dataset. Training data points are generated from dual-arm robot environments with various relative positions between two arms.

Target Systems. For the test environments, we selected three high-dof multi-arm robot systems as illustrated in Figure 4.5. We employed 7-dof Franka Emika Panda robot arms for our test environments. For the test dataset of each target environment, we sampled one million joint configurations from a uniform distribution within the joint limits.

Baselines. We trained our method and other existing collision distance estimation methods.

- **Capsule**: A bounding volume method with capsule-shape collision primitives used in [8, 38].
- **JointNN**: A fully-connected neural network model that directly uses joint configurations as inputs (the input representation used in [8, 28]).
- **PosNN**: A fully-connected neural network model that uses joint positions as inputs (the input representation used in [27, 30]).

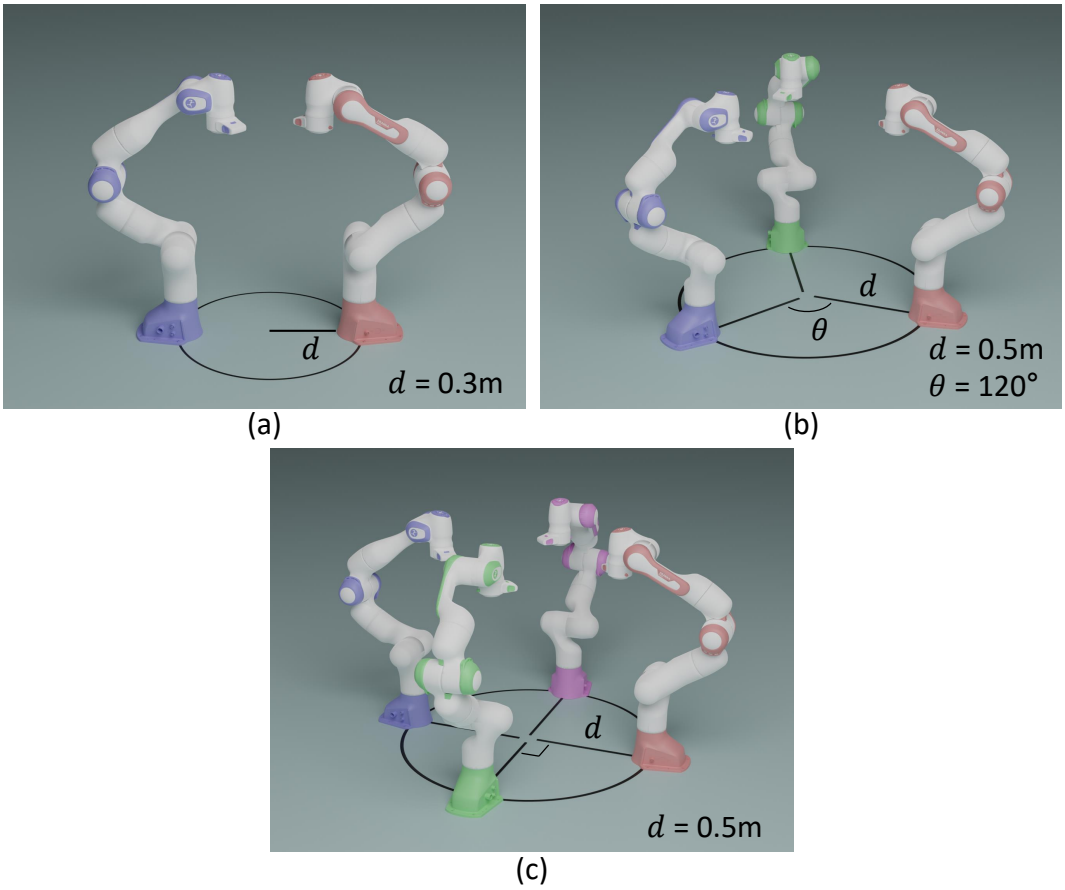

Figure 4.5: Test environments for the collision distance learning performance evaluation. We selected (a) two arms, (b) three arms, and (c) four arms robot systems.

- **jointNERF** [18]: A fully-connected neural network model that uses positional embedding vectors of joint configurations as inputs.
- **ClearanceNet** [31]: A neural network model that takes joint configurations as inputs and utilizes two fully-connected layers, each followed by a dropout layer.
- **DiffCo** [26]: A kernel perceptron model that takes joint configurations as inputs and outputs the collision score.

Existing collision distance learning methods were trained on one million uniformly sampled data points within the joint limits for each target system. For the DiffCo model, we were limited to a dataset size of 75,000 data points, as this was the maximum feasible size for kernel perceptron training on our hardware (AMD Ryzen Threadripper 3960X, 256GB RAM, and NVIDIA GeForce RTX4090 with 24GB VRAM).

Evaluation Metrics. We evaluate the performance of collision distance learning using four metrics: MSE, AUROC, Accuracy, and Safe-FPR. These metrics target both the collision distance regression and collision classification, with a robot configuration being classified as a collision if the collision distance is below the threshold ϵ (for DiffCo, if the collision score is above the threshold). MSE represents the mean squared error between the ground truth and estimated global collision distance. AUROC is the area under the receiver operating characteristic curve for the collision classification. Accuracy is the classification accuracy of collisions with the threshold $\epsilon = 0$. Lastly, in order for the trained collision distance estimation model to be used in actual path planning tasks, a sufficiently conservative threshold must be used to ensure that collisions cannot occur. However, the more conservative the threshold used, the more false alarms will occur where non-collision robot configurations are incorrectly classified as collisions. Safe-FPR is used to evaluate performance in these situations, representing the false alarm rate

when using the least yet sufficient conservative threshold that can classify all collision configurations in the test dataset as collisions.

4.4.2 Results

Performance Evaluation. Table 4.1 presents the evaluation results of PairwiseNet and other existing methods. Notably, PairwiseNet achieves significantly lower MSE values compared to the baseline methods. In all three environments (two, three, and four arms), PairwiseNet's MSE is at least an order of magnitude smaller than that of the best-performing baseline. For instance, in the four-arm environment, PairwiseNet's MSE (0.46e-4) is approximately 1/15 of the next best result (Capsule at 6.66e-4). Furthermore, PairwiseNet greatly surpasses the baselines in terms of safe-FPR, consistently achieving the lowest values across all environments. This indicates a superior ability to avoid false collision alarms, which is crucial for efficient robot operation. Importantly, we observe that the performance gap between PairwiseNet and the baselines widens as the complexity of the system increases. This is particularly evident in the four-arm environment, where PairwiseNet maintains high performance while other methods show more significant degradation. These results underscore PairwiseNet's robustness and scalability in handling higher-dof systems, a critical advantage for complex robotic applications.

Following the quantitative metrics, Fig. 4.6 presents the Detection Error Trade-off (DET) curves, which visualize the relationship between false positive rates (FPR) and false negative rates (FNR) across different detection thresholds. DET curves are used to evaluate binary classifiers, where curves closer to the origin (bottom-left) indicate superior performance as they represent lower error rates for all threshold choices. The curves clearly demonstrate PairwiseNet's superior detection capabilities, as its curve (purple

solid line) consistently lies closest to the origin across all test environments. The safe-FPR metric can be visualized in these DET curves; it represents the minimum false positive rate while maintaining zero false negatives, corresponding to the point where the curve intersects the x-axis (FNR = 0) closest to the origin. We indicate PairwiseNet's safe-FPR values as purple dots on its curves.

Generalizability. Existing learning methods have used individual models trained on each system's specific dataset to estimate collision distance in the three target robot systems, as the global collision distance function (2.2.4) differs for each system. In contrast, the pairwise collision distance function (4.3.1) remains the same even when robot base positions or the number of robot arms change. Consequently, PairwiseNet can estimate collision distance using a single model for all three target robot systems, and its performance even surpasses that of existing methods that use individual models for each system.

We extended our validation of PairwiseNet to include robot arm systems arranged asymmetrically and in an irregular manner, in addition to the three multi-arm robot systems used in our experiments. These additional systems are illustrated in Figure 4.7. In each systems, the robot arms are arranged in a more irregular and complex manner than in the previously used systems, resulting in a greater variety of relative positional relationships between the arms.

We used the trained PairwiseNet model that was originally used for our multi-arm robot systems. Although these new robot systems have complex arrangements of robot arms, they still consist of Panda robot arms, which have been sufficiently trained. Therefore, PairwiseNet can be applied directly to these systems without the need for additional training.

Table 4.1: Collision distance estimation performances of PairwiseNet versus baseline methods

Env.	Methods	MSE	AUROC	Accuracy ($\epsilon = 0$)	safe-FPR
Fig. 4.5 (a) (Two arms)	Capsule	5.47e-4	0.9995	0.9776	0.0247
	JointNN	3.63e-4	0.9955	0.9794	0.3200
	PosNN	2.71e-4	0.9970	0.9823	0.1476
	jointNERF	2.98e-4	0.9962	0.9808	0.2371
	ClearanceNet	1.11e-3	0.9853	0.9621	0.4570
	DiffCo*	-	0.9824	0.9818	0.3141
	PairwiseNet (ours)	**0.24e-4**	**0.9998**	**0.9941**	**0.0200**
Fig. 4.5 (b) (Three arms)	Capsule	5.96e-4	0.9993	0.9775	0.0241
	JointNN	9.69e-4	0.9902	0.9721	0.2679
	PosNN	5.41e-4	0.9951	0.9801	0.1336
	jointNERF	8.22e-4	0.9920	0.9747	0.2213
	ClearanceNet	4.63e-3	0.9499	0.9395	0.6067
	DiffCo*	-	0.9603	0.9453	0.5858
	PairwiseNet (ours)	**0.24e-4**	**0.9997**	**0.9944**	**0.0189**
Fig. 4.5 (c) (Four arms)	Capsule	6.66e-4	0.9986	0.9468	0.0694
	JointNN	1.59e-3	0.9718	0.9183	0.6988
	PosNN	7.18e-4	0.9885	0.9478	0.5371
	jointNERF	1.30e-3	0.9778	0.9280	0.6260
	ClearanceNet	6.67e-3	0.8738	0.8202	0.9965
	DiffCo*	-	0.8811	0.8306	0.9874
	PairwiseNet (ours)	**0.46e-4**	**0.9994**	**0.9858**	**0.0650**

*DiffCo outputs the collision score from -1 (collision-free) to 1 (collision).

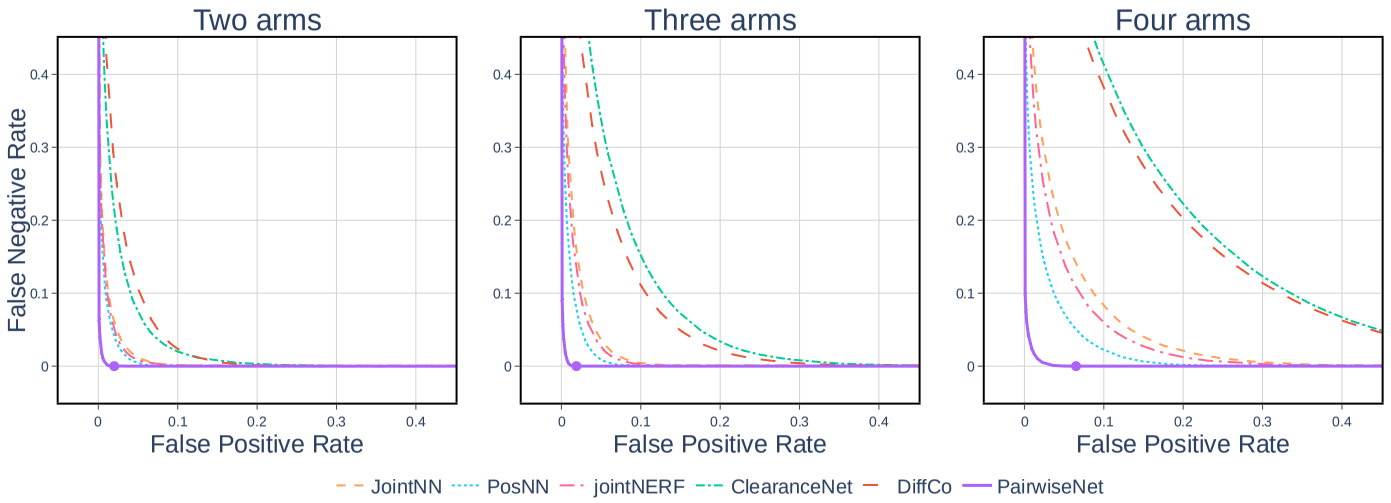

Figure 4.6: Detection Error Trade-off (DET) curves for collision detection methods. Curves closer to the origin indicate better performance, with PairwiseNet (purple solid line) achieving superior detection accuracy across all test environments.

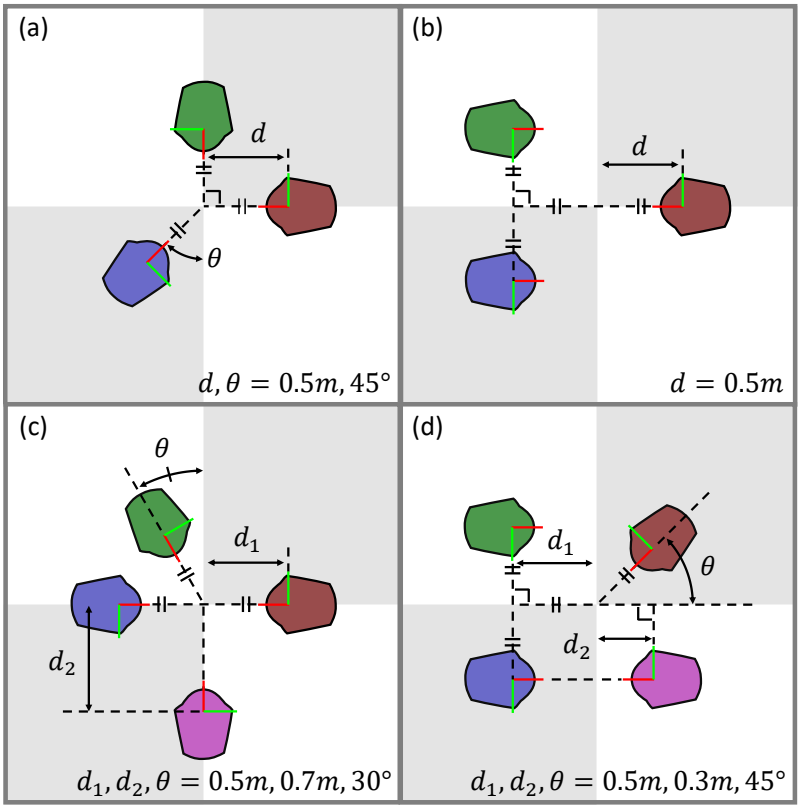

Figure 4.7: Illustrations of the top views of various base positions within multi-arm robot systems.

Table 4.2: Collision distance estimation performances of PairwiseNet for various multi-arm systems

Env.	MSE	AUROC	Accuracy ($\epsilon = 0$)	safe-FPR
Fig. 4.7 (a) (Three arms)	0.24e-4	0.9997	0.9943	0.0341
Fig. 4.7 (b) (Three arms)	0.17e-4	0.9999	0.9987	0.0048
Fig. 4.7 (c) (Four arms)	0.43e-4	0.9991	0.9859	0.0611
Fig. 4.7 (d) (Four arms)	0.27e-4	0.9995	0.9931	0.0292

We have presented the results of PairwiseNet's collision distance estimation performance in Table 4.2. Upon examining these results, we observed that PairwiseNet performed well across all robot systems and metrics, regardless of the complexity of the arrangement of robot arms. The consistency in performance across these varying configurations demonstrates the robustness of PairwiseNet, affirming that there is no significant difference in the quality of collision distance estimation between these different scenarios.

4.5 Collision Distance Learning for a Single-arm Systems with Obstacles

We perform experiments in a 7-dof single-arm system with complex obstacles (Figure 4.8). This workspace is populated with obstacles such as shelves and tables that add complexity to the robot arm's operational landscape. To validate the collision distance estimation performance in this system, we generate a human-guided demonstration with the robot arm occasionally sweeping close to, and sometimes colliding with, tables and shelves for the test dataset.

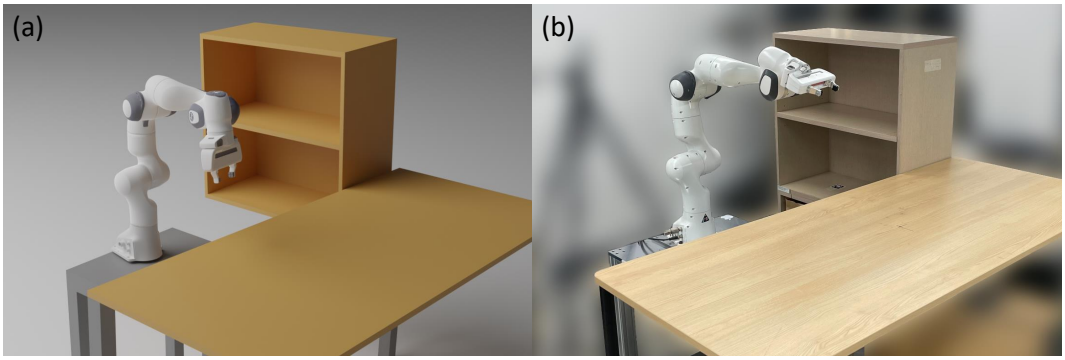

Figure 4.8: (a) Simulation and (b) real-world environments for a single-arm system with obstacles.

4.5.1 Experimental Setting

In this experiment, we decomposed tables into a tabletop and four legs, and shelves into six plates. This strategy was employed not only to simplify complex obstacles and make the pairwise collision distance more tractable but also to leverage the symmetry of the obstacle structure. For example, the bottom, middle, and top plates of the shelf have the same shape, so dividing them into separate elements allows for more efficient use of training data. Furthermore, the fact that the decomposed elements of the tables and shelves take the form of flat rectangular shapes can be beneficial to the learning process. Apart from this, the training methods of PairwiseNet and baselines, as well as the dataset generation and model architectures used, are identical to those employed in the experiments for multi-arm robot systems.

4.5.2 Results

The collision distance estimation results are presented in Figure 4.9. Note that Figure 4.9 displays only four snapshots from the demonstration; the complete video can be

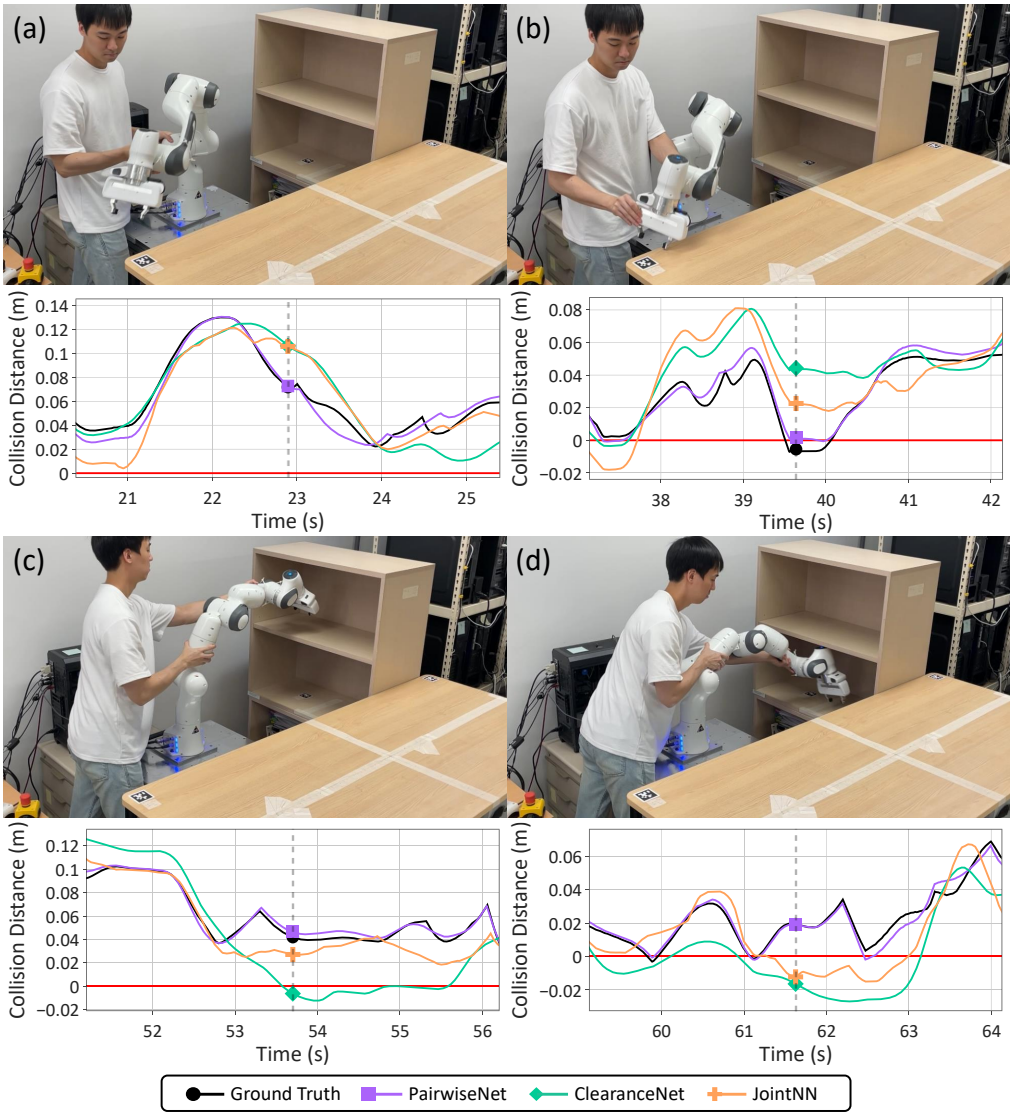

Figure 4.9: Collision distance estimation results for a single-arm system with obstacles. The top images display a human-guided robot arm, while the corresponding plots at the bottom illustrate the ground truth and estimated collision distances from PairwiseNet and other baselines at time $t =$ (a) 22.9s, (b) 39.6s, (c) 53.7s, and (d) 61.6s, respectively.

accessed in the supplementary[1] video. Compared to the evaluated baselines, PairwiseNet consistently exhibits the highest accuracy in estimating the actual collision distance. It reliably detects collisions between the robot arm and obstacles in the majority of cases. In contrast, the other baselines either fail to trigger collision alarms when a collision occurs (Figure 4.9 (b)) or produce false collision alarms when no collision is present (Figure 4.9 (c), (d)). PairwiseNet stands out by consistently and accurately identifying collisions between the robot and obstacles.

4.6 Inference Time of PairwiseNet

PairwiseNet incorporates an efficient inference strategy – using only the regressor network during the inference process, and computing multiple element pairs as a single batch. Thus, despite calculating collision distances pairwise like traditional non-data-driven methods, it demonstrates an inference speed as fast as other existing data-driven approaches. We have compared the inference speed of PairwiseNet with that of standard non-data-driven methods and displayed the results in Table 4.3. The standard collision distance calculation algorithms used for comparison are from the Flexible Collision Library (FCL) [39].

The test environments were multi-arm robot systems with two, three, and four Franka Emika Panda robot arms, containing 64, 192, and 384 collision pairs respectively. For each robot arm, we prepared three different geometric representations: original mesh, simplified convex mesh (created using convex hull), and capsule primitives for comprehensive comparison. More details about these geometric representations are provided in Appendix B.2. We measured the inference time for estimating collision distances for

[1] https://youtu.be/N5Q8ZXbB6Uc

1,000 joint configurations using both classical FCL methods with these different geometric representations and our PairwiseNet approach with both CPU and GPU implementations. A key advantage of our learning-based collision distance estimator is its ability to analytically compute gradients with respect to joint configurations. We also measured the inference time of PairwiseNet's gradient computation, which is represented under the PairwiseNet (gradient) entries in Table 4.3.

Additionally, PairwiseNet is capable of receiving multiple joint configurations as a single batch input; it can simultaneously compute the collision distances for all the joint configurations. This is an efficient feature for tasks that require simultaneous collision checks for multiple joint configurations. In our experiments, we also measured the time taken to calculate both the collision distances and their gradients for 1,000 joint configurations at once through PairwiseNet, and this is represented under the PairwiseNet (batch) and PairwiseNet (gradient, batch) entry in Table 4.3. PairwiseNet was implemented using PyTorch [47]. FCL were implemented using the `python-fcl` library. Thus, all the time measurements are conducted within Python code. The experiments were carried out in an environment equipped with an AMD Ryzen 9 7950X (16 cores, 32 threads), NVIDIA RTX 4090, and 125GB of RAM.

While FCL with simplified geometric representations (convex meshes and capsule primitives) demonstrates reasonable computational efficiency for real-time applications, PairwiseNet achieves comparable performance even when running on CPU. When utilizing GPU acceleration, PairwiseNet significantly outperforms these classical methods. Notably, as the number of collision pairs increases with system complexity (from two to four arms), the computational cost of FCL scales roughly linearly, while PairwiseNet

Table 4.3: Inference time comparison: PairwiseNet versus classical collision distance estimation methods

Methods	Inference time for 1000 joint poses (s)					
	Two arms (64 pairs)		Three arms (192 pairs)		Four arms (384 pairs)	
FCL (original mesh)	13.87		43.97		92.63	
FCL (convex mesh)	0.0714		0.1682		0.3265	
FCL (capsule)	0.0294		0.0582		0.1130	
	CPU	GPU	CPU	GPU	CPU	GPU
PairwiseNet	0.0774	0.0630	0.1480	0.0646	0.2065	0.0671
PairwiseNet (batch)	0.0226	0.0001	0.1066	0.0003	0.2192	0.0003
PairwiseNet (gradient)	0.1813	0.1484	0.3038	0.1380	0.3974	0.1407
PairwiseNet (gradient, batch)	0.0697	0.0004	0.2257	0.0004	0.4727	0.0004

maintains relatively consistent inference times, particularly in GPU operations. This advantage becomes even more pronounced when leveraging PairwiseNet's batch processing capability, which enables simultaneous evaluation of multiple configurations. Furthermore, as a learning-based method, PairwiseNet offers the unique ability to analytically compute gradients with respect to joint configurations – a crucial feature for optimization-based motion planning. The gradient computation speed of PairwiseNet is remarkably efficient, comparable to the basic collision distance calculations of classical methods. This efficiency extends to batch gradient computations as well, where GPU acceleration enables extremely fast parallel processing of multiple configurations. These results demonstrate PairwiseNet's practical viability for real-world robotics applications, particularly in scenarios requiring rapid collision checking or gradient-based optimization.

4.7 Conclusion

In this paper, we present PairwiseNet, a novel collision distance estimation method that estimates the minimum distance between a pair of elements instead of directly predicting the global collision distance of the robot system. By simplifying the problem into smaller sub-problems, our approach achieves significant performance improvements for high-dof robot systems compared to methods that directly predict the global collision distance. Additionally, PairwiseNet is capable of handling environmental changes such as robot base repositioning without requiring additional training or fine-tuning. We evaluate and compare the collision distance estimation performance of PairwiseNet for both high-dof multi-arm robot systems and single-arm systems in the presence of obstacles, and validate its accurate collision distance estimation and generalization to environmental changes.

5

Planning with Collision Distance Estimator

5.1 Introduction

In this chapter, we demonstrate practical applications of our proposed novel methods for learning collision distance in high-dof robot systems. Our approach stands out for its exceptional accuracy in complex, high-dof configurations, where existing methods often struggle. We showcase how our learning-based collision distance estimator can be applied to various planning tasks, leveraging the general advantages of such estimators:

- Rapid calculation of collision distances
- Batch processing capability for simultaneous estimation of collision distances across multiple joint configurations
- Ability to compute derivatives of collision distance with respect to joint configurations

To highlight both the general benefits of learning-based collision distance estimators and the unique strengths of our method, we present two challenging applications:

- Offline trajectory optimization (Section 5.2): This task demonstrates the estimator's batch processing capability and efficient derivative calculation in a high-dof setting (a 28-dof four-arm robot system).
- Real-time collision avoidance (Section 5.3): This application showcases the estimator's fast inference speed for collision distances and their derivatives in a complex environment (a single-arm robot system with obstacles).

These examples, performed in scenarios that are particularly challenging for existing methods, illustrate the versatility, efficiency, and superior accuracy of our proposed collision distance estimator in addressing complex path planning challenges.

5.2 Offline Trajectory Optimization for a Four-arm Robot System

In this section, we perform trajectory optimization to generate collision-free paths. This gradient-based optimization process requires the derivative of collision distances with respect to joint configurations. It benefits significantly from the batch calculation capability of learning-based collision distance estimators, as collision distances and their derivatives for all joint configurations along the path must be estimated at each optimization step.

Our experiments are conducted in a complex, high-dof robot system, presenting a challenge that is nearly impossible for existing methods due to their limited applicability to high-dof systems. This scenario showcases the unique strengths of our approach in handling such challenging environments.

The trajectory optimization for collision-free paths is a constrained optimization problem with equality and inequality constraints. Given the start and end poses, the objective is to minimize the path length while ensuring that joint configurations remain within their limit ranges and collision distances stay above a safe threshold ϵ along the entire path $q(t)$. Mathematically, this can be expressed as:

$$\begin{aligned} &\underset{q(t)}{\text{minimize}} \quad \texttt{length}[q(t)] \\ &\text{subject to} \quad \begin{cases} q(0) = q_i,\ q(T) = q_f \\ q_{\text{min}} \leq q(t) \leq q_{\text{max}} \\ \dot{q}_{\text{min}} \leq \dot{q}(t) \leq \dot{q}_{\text{max}} \\ F_\psi(q(t); f_\psi, \mathcal{S}) \geq \epsilon \end{cases} \quad \forall t \in [0, T], \end{aligned} \tag{5.2.1}$$

where q_i and q_f are the start and end poses, respectively. The function `length` can represent either the path length in the joint configuration space or the path length in task space (i.e., the path length of the end-effectors in the system). In our experiments, we use a combination of both lengths for the optimization objective to ensure smoothness of the optimized path.

We modeled the trajectory $q(t)$ as a natural cubic spline curve defined by m via-points q_m with corresponding timestamps $t_m \in (0, T)$. This approach uses third-order polynomials between via-points, ensuring a smooth path that passes through each point. To verify the inequality constraints in (5.2.1) at each optimization step, we randomly sampled 10,000 samples of t from the interval $[0, T]$.

The optimization problem (5.2.1) presents significant challenges due to its highly non-convex nature and numerous local minima. These difficulties are particularly pronounced in complex systems like our four-arm robot system, where the arms' movements are intricately intertwined. Hence, the suitable choice of initial trajectory $q(t)$ is

a crucial factor in the optimization process. Our initial attempts to use a straight path in joint configuration space as the initial trajectory often failed to yield feasible solutions. To address this, we adopted a two-step approach: first, we employed the RRT-connect algorithm [3] to generate suboptimal via-points, then used these as the initial via-points for $q(t)$ in our optimization process.

Figure 5.1 illustrates three optimized trajectories. For each trajectory, the initial ($t = 0$) and final ($t = 5$) poses are predefined, with each pose set to have the four robot arms intricately intertwined. Planning in such a complex system necessitates highly accurate collision distance estimates. Our approach successfully demonstrates the ability to perform planning in these challenging scenarios, as evidenced by the resulting trajectories.

Figure 5.2 displays the ground truth collision distances along with the estimated collision distances from our approach and baseline methods for each optimized trajectory. Given that these trajectories involve four robot arms moving in intricate, intertwined configurations, the element pair determining the collision distance (i.e., the closest element pair) changes dynamically. This dynamic nature is evident in the ground truth collision distance plot (black line) in Figure 5.2, which exhibits abrupt changes rather than smooth transitions. Existing methods, which attempted to learn this complex global collision distance directly, show significant deviations from the ground truth in their predictions. In contrast, our methodology, which focuses on learning pairwise collision distances for each element pair, uniquely succeeds in accurately predicting these complex collision distances.

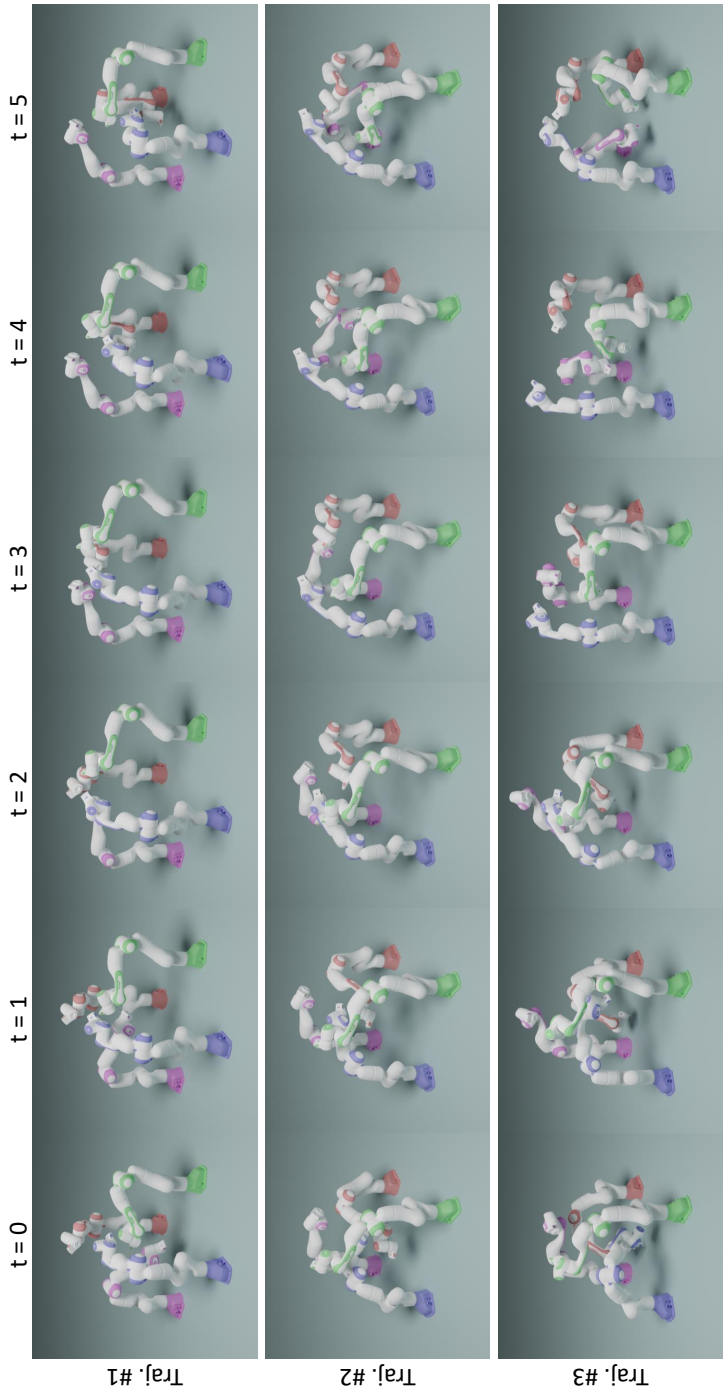

Figure 5.1: Optimized trajectories for the 28-dof four-arm robot system: three solutions demonstrating collision-free paths in a complex environment.

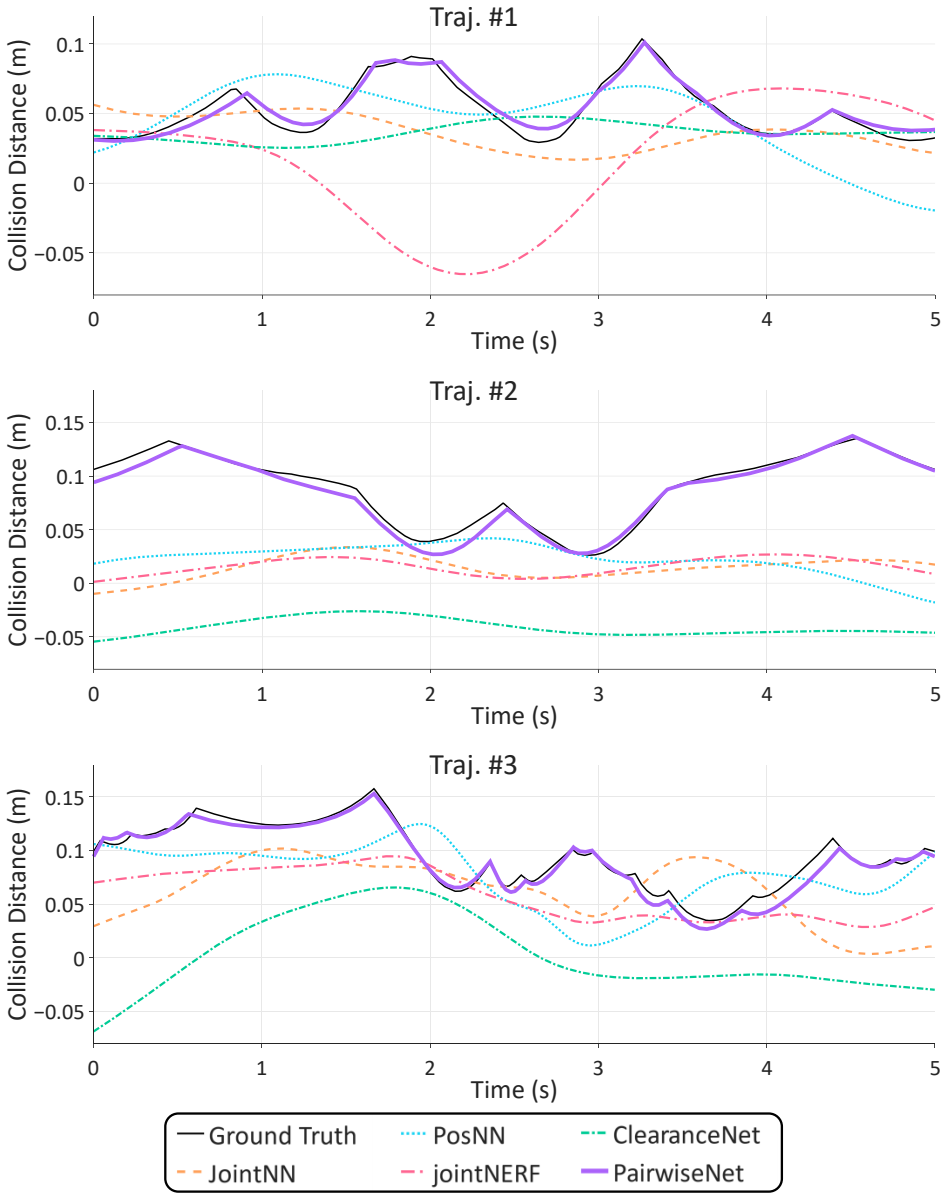

Figure 5.2: Collision distance plots for optimized trajectories: ground truth, PairwiseNet, and baseline methods.

5.3 Real-time Collision Avoidance for a Single-arm Robot System with Obstacles

In this section, we demonstrate the application of our collision distance estimator to real-time collision avoidance in a complex environment. This scenario showcases the fast inference speed of our method for calculating collision distances and their derivatives, which is crucial for reactive motion planning.

Our experimental setup consists of a single-arm robot system operating in an environment with multiple obstacles. This setting, while less complex in terms of degrees of freedom compared to the four-arm system, presents unique challenges due to the presence of external obstacles.

We define a repulsion potential for collision avoidance that pushes the robot arm away from obstacles. Traditionally, repulsion potentials need to be defined for each obstacle individually, which involves calculating the minimum distance and its derivative between every pair of elements. However, by employing our learning-based collision distance estimator, the repulsion potential becomes much simpler to define. In our experiment, we apply a repulsion torque when the estimated collision distance falls below a threshold ($\hat{d}_{\text{col}} < \epsilon$). This torque acts in the direction that increases the collision distance, easily computed as the derivative of our trained model with respect to the joint configuration ($\nabla \hat{d}_{\text{col}}(q)$).

Figure 5.3 illustrates this concept for a 2-dof planar robot, showing how the derivative directs movement towards increased collision distance. As the robot approaches the upper right triangular obstacle (Figure 5.3 (a)), the current joint pose is marked as a red dot in Figure 5.3 (b) and (c). Since the robot is near the obstacle, a repulsion force is applied in the direction of the collision distance derivative $\nabla \hat{d}_{\text{col}}(q)$. This direction guides the robot away from the obstacle. This can be conceptualized as a kind of spring

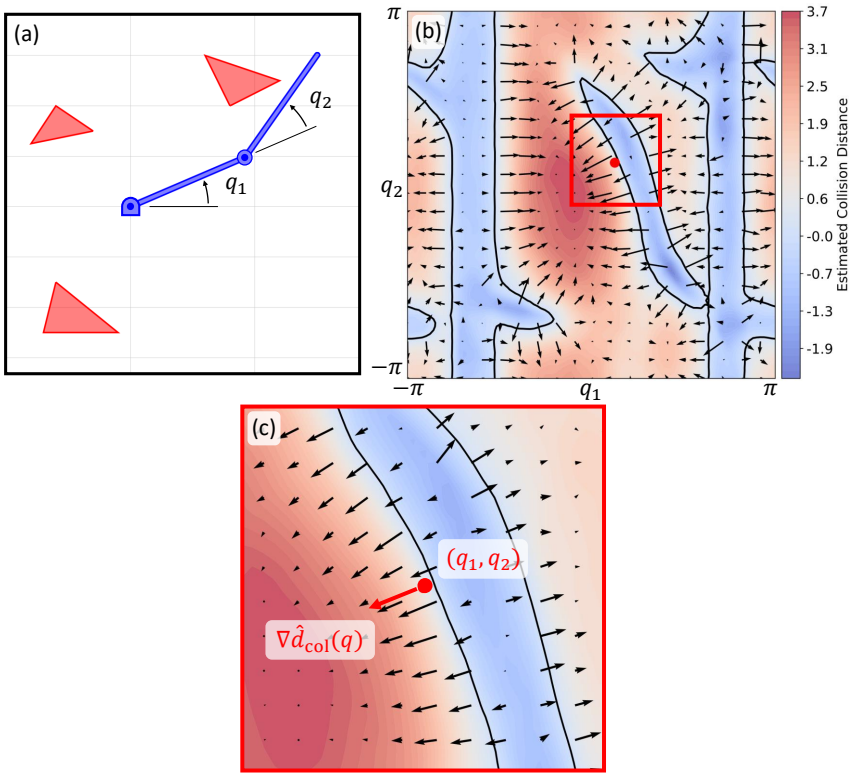

Figure 5.3: Illustration of a collision repulsion example for a simple toy case. (a) A 2-dof planar robot (blue) and obstacles (red). (b) Joint configuration space with collision distance contour plot. Colors denote the collision distance, increasing from blue (low) to red (high). Black lines indicate collision boundaries where $\hat{d}_{\text{col}} = 0$, and black arrows indicate the derivative $\nabla \hat{d}_{\text{col}}(q)$. (c) The red dot denotes the current joint pose, which is near the upper right obstacle. The direction of the collision distance derivative $\nabla \hat{d}_{\text{col}}(q)$ at this joint pose is the direction of increasing collision distance, which moves the robot away from the obstacle.

system attached at the collision boundary ($\hat{d}_{\text{col}} = \epsilon$) defined in the joint configuration space. We include a damping term to prevent the arm from bouncing off obstacles, ensuring smoother and more stable motion.

Mathematically, the control torque can be expressed as:

$$\tau = g(q) + \mathbf{1}_{\hat{d}_{\text{col}}(q)<\epsilon} \frac{\nabla \hat{d}_{\text{col}}(q)}{|\nabla \hat{d}_{\text{col}}(q)|} \left(K_p(\epsilon - \hat{d}_{\text{col}}(q)) - K_d \dot{\hat{d}}_{\text{col}}(q) \right), \tag{5.3.2}$$

where $g(q)$ denotes the gravity compensation torque, $\mathbf{1}_{\hat{d}_{\text{col}}(q)<\epsilon}$ represents an indicator function that equals 1 when the estimated collision distance $\hat{d}_{\text{col}}(q)$ is less than the threshold ϵ and 0 otherwise, $K_p \in \mathbb{R}$ and $K_d \in \mathbb{R}$ are the proportional and derivative gains respectively, and $\nabla \hat{d}_{\text{col}}(q)$ is the derivative of the estimated collision distance with respect to the joint configuration.

This control scheme requires real-time calculation of collision distance and its derivative, benefiting greatly from learning-based collision distance estimators which can provide these values rapidly and efficiently. Specifically, due to the superior accuracy of our method, we can effectively control the robot in our complex system, which involves a robot arm navigating among intricate, concave obstacles such as shelves - a scenario that poses significant challenges for existing approaches.

We conducted an experiment to validate the effectiveness of our collision avoidance system by freely pushing the robot towards obstacles while applying the aforementioned control scheme. Our control method only applies gravity compensation when the robot is far from obstacles, allowing it to move freely in response to external forces. However, as the robot approaches an obstacle and the collision distance falls below the threshold, the collision repulsion torque activates, ensuring that the robot maintains a safe distance from the obstacle while continuing to move. This behavior results in smooth, collision-free motion even when external forces that may occur collisions with obstacles are applied.

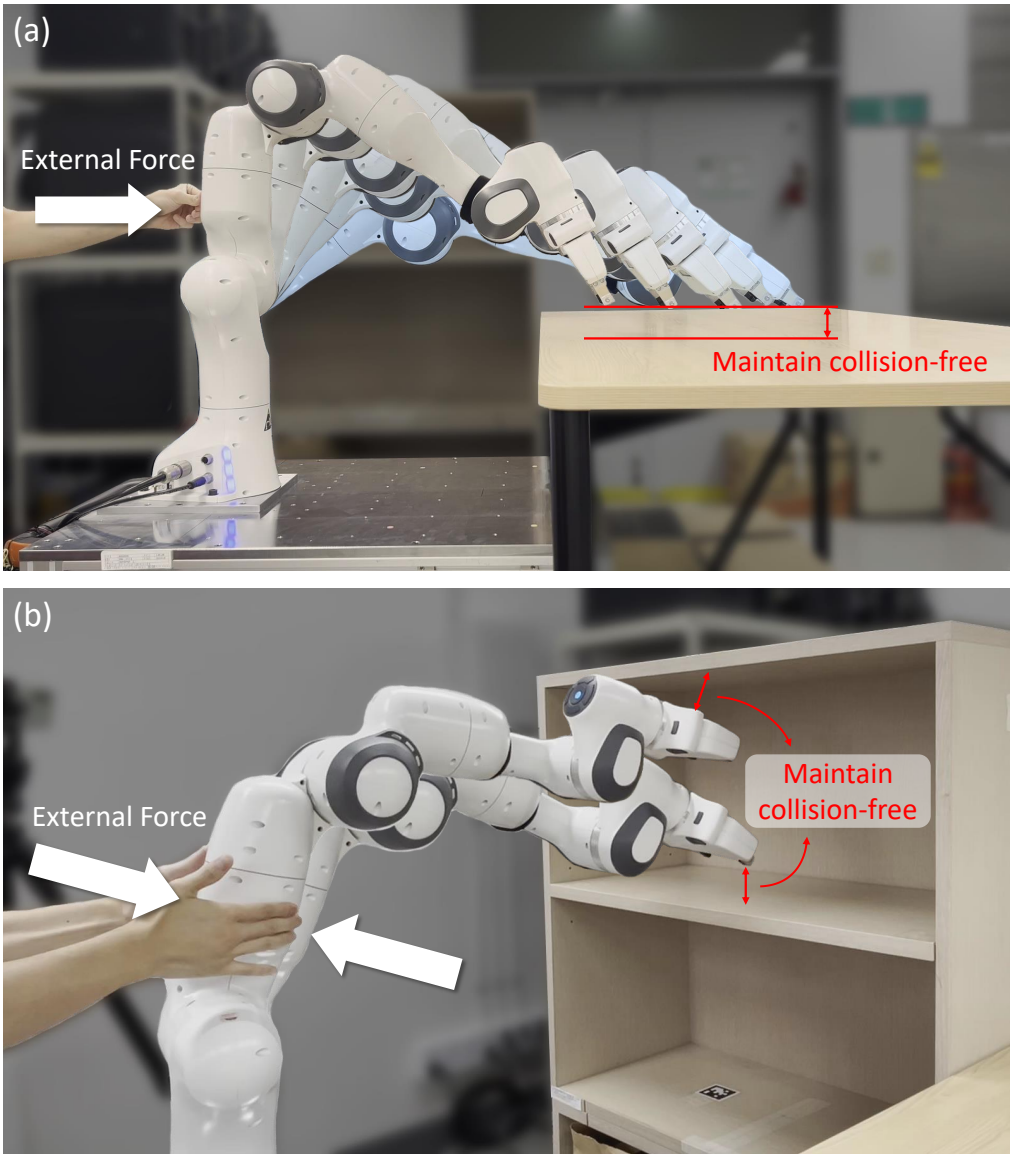

Figure 5.4: Real-time collision avoidance demonstration. (a) Robot arm maintaining safe distance from a table despite external forces. (b) Arm navigating within a complex shelf structure, demonstrating stable collision avoidance in confined spaces.

Figure 5.4 presents a series of sequential snapshots from our experiment. We applied external forces to the freely movable robot arm, attempting to cause collisions with obstacles. As the arm approaches a table, we observe that repulsion torques activate in the direction that increases the collision distance, maintaining a collision-free state while the arm continues to move (Figure 5.4 (a)). Furthermore, even when the robot arm is positioned within a complex, concave structure like a shelf, it accurately estimates the distance to each wall of the shelf. This enables the arm to operate stably while avoiding collisions in this challenging environment (Figure 5.4 (b)). These results demonstrate the effectiveness of our approach in real-time collision avoidance, even in complex robot systems. The complete experiment can be viewed in the supplementary[1] video.

5.4 Conclusion

In conclusion, this chapter has demonstrated the practical effectiveness of our novel learning-based collision distance estimator in complex path planning scenarios. Through two challenging applications - offline trajectory optimization for a 28-dof four-arm robot system and real-time collision avoidance for a single-arm robot navigating obstacles - we have showcased the unique strengths of our approach. Our method has proven its capability in leveraging the general advantages of learning-based estimators, including rapid calculation of collision distances, efficient batch processing, and derivative computation. More importantly, it has demonstrated superior accuracy in complex, high-dof configurations where traditional methods often fall short.

The offline trajectory optimization task highlighted our estimator's ability to handle intricate multi-arm scenarios, efficiently processing multiple joint configurations simultaneously. The real-time collision avoidance experiment, on the other hand, emphasized

[1] https://youtu.be/PloLnzSc0d0

the method's fast inference speed and accuracy in robot systems with complex obstacles.

These results underscore the versatility and robustness of our approach in addressing complex robotic planning challenges. By providing accurate, real-time collision distance estimates even in highly complex scenarios, our method opens up new possibilities for advanced robot control and automation in challenging environments. The success in these applications suggests that our approach could significantly enhance the capabilities of robots in various fields, from industrial automation to service robotics, where precise navigation in complex environments is crucial.

6 Conclusion

6.1 Summary

This thesis has addressed critical challenges in collision distance estimation for complex, high-dof robot systems. Through the development of novel approaches, we have significantly advanced the state-of-the-art in both the methodology and practical application of collision distance estimation. Our contributions offer more accurate, efficient, and adaptable solutions, particularly for scenarios involving high-dof robot systems. The methods presented in this thesis have important implications for enhancing the capabilities of robots in various applications, from industrial automation to service robotics, where precise planning in complex environments is crucial.

The key contributions of this thesis are:

- **Active Learning of the Collision Distance Function**

  We proposed an innovative active learning strategy for high-dof robot systems that

overcomes the limitations of existing learning-based methods. Our approach focuses on sampling near-boundary configurations in high-dimensional configuration spaces, addressing the challenge of exponential growth in data requirements as the dof of the system increase. By maintaining a fixed-size training dataset that is continuously updated with these new data points, we ensure that the model learns from the most informative data points. Experimental evaluations on high-dof robot systems demonstrated substantial performance improvements over existing state-of-the-art methods, validating the effectiveness of our approach.

- **PairwiseNet: Pairwise Collision Distance Learning**

  Addressing the limitations of simplistic input representations and model structures in capturing complex collision distance functions for high-dof robot systems, we first explored a link $SE(3)$ configuration space representation. This approach, incorporating additional rotational information of links, showed improvements over simpler input representations. However, recognizing that this alone was insufficient to tackle the inherently complex and non-smooth nature of the global collision distance function, we developed PairwiseNet, a novel collision distance estimation method.

  PairwiseNet provides a promising alternative to existing data-driven approaches by focusing on predicting the pairwise collision distances rather than directly estimating the global collision distance. This approach simplifies the learning task by breaking down the complex, non-smooth global collision distance function into more manageable sub-problems. As a result of this innovative approach, PairwiseNet demonstrates superior performance compared to existing state-of-the-art methods in collision distance estimation for complex robotic systems.

  An additional advantage of PairwiseNet is its remarkable generalizability to minor

environmental changes, which allows its applicability to various robot configurations without requiring retraining. This flexibility allows PairwiseNet to adapt to systems with multiple arms or changed base positions, as long as the constituent shape elements remain consistent with those used during training. Our evaluations on high-dof multi-arm systems (from 14-dof two-arm to 28-dof four-arm configurations) and single-arm robots with obstacles demonstrated PairwiseNet's superior performance in terms of various performance metrics.

Furthermore, we demonstrated the practical applications of our collision distance estimation methods in challenging path planning scenarios. Our approach showcased exceptional accuracy and efficiency in complex, high-dof configurations where existing methods often struggle. We leveraged the key advantages of our learning-based estimators, including rapid calculation of collision distances, batch processing capabilities, and efficient computation of distance derivatives. Two significant applications were presented: offline trajectory optimization for a 28-dof four-arm robot system, and real-time collision avoidance for a single-arm robot system with obstacles. The success in these challenging scenarios underscores the versatility and superior performance of our collision distance estimation techniques in addressing complex path planning challenges.

6.2 Future Work

The generalizability of PairwiseNet is currently limited to systems that exclusively consist of known shape elements. Given the significant impact of generalization to handle unseen objects on PairwiseNet's practical applicability, we conducted additional experiments investigating its potential (detailed in Appendix B.6). Leveraging PairwiseNet's encoder architecture, which transforms object geometries into shape feature vectors, our model demonstrates modest performance in the environment containing unseen objects.

While these experimental results are currently limited to simple objects with dimensional variations, they reveal the encoder's ability to capture generalizable geometric features rather than merely memorizing specific configurations. Future work is aimed at enhancing the generalizability of PairwiseNet to systems with unseen objects, which could involve constructing diverse datasets containing link geometries of other robot arms and various objects, and incorporating techniques to handle unknown or novel shape elements [48].

A

Appendix: Active Learning of the Collision Distance Function

A.1 Hyperparameters of our Active Learning-based Training Procedure

Table A.1 presents the key hyperparameters employed in our active learning-based training procedure. The process involves $N_{\text{active}} = 20$ iterations, each consisting of model training and dataset update phases. During each iteration, the model is trained for $N_{\text{epoch}} = 1000$ epochs, ensuring sufficient learning from the current dataset. The parameter $k = 100000$ represents the number of new configurations sampled in each iteration; with the total dataset containing a million data points, this updates 10% of the dataset per iteration. The parameters of MCMC sampling (σ_e, σ, u_{min}, max_step) are carefully tuned to suitably and stably sample new data points near collision boundaries.

Since the parameter k, representing the number of newly sampled data points in

Table A.1: Hyperparameters for our active learning-based training procedure

N_{active}	N_{epoch}	k	σ_e	σ	u_{min}	max_step
20	1000	100000	0.1	0.05	0.8	1000

each iteration, is the core parameter of the proposed active learning-based training procedure, we conducted an additional ablation study to analyze how the choice of k influences learning performance. The experiments were performed on the two-arm environment shown in Figure 3.5(a) using the simplest JointNN model. We examined how performance changed during active learning iterations by varying the replacement ratio (the ratio of k to the total dataset size) from 5% to 80%.

Figure A.1 shows four performance metrics plotted against the number of active learning iterations on the x-axis. Each model was trained five times with different random seeds, with solid lines representing the mean performance and shaded regions indicating the standard deviation. The replacement ratios used in each model's training are indicated by colors, ranging from yellow (lower ratios) to purple (higher ratios). The results demonstrate that model performance improves with active learning iterations before eventually converging to a certain level. Performance tends to slightly improve with higher replacement ratios, which can be merely attributed to the increased volume of training data (initial dataset + $k \times N_{\text{active}}$ data points). While higher values of k can yield these modest performance gains, they also demand greater computational resources, necessitating a careful balance between these competing factors. We did not employ higher k values in our main experiments since improving performance simply by using more data diverges from our paper's focus on optimizing data distribution under computational constraints. However, using larger k values could be considered when computational resources are abundant and maximum performance is the primary goal.

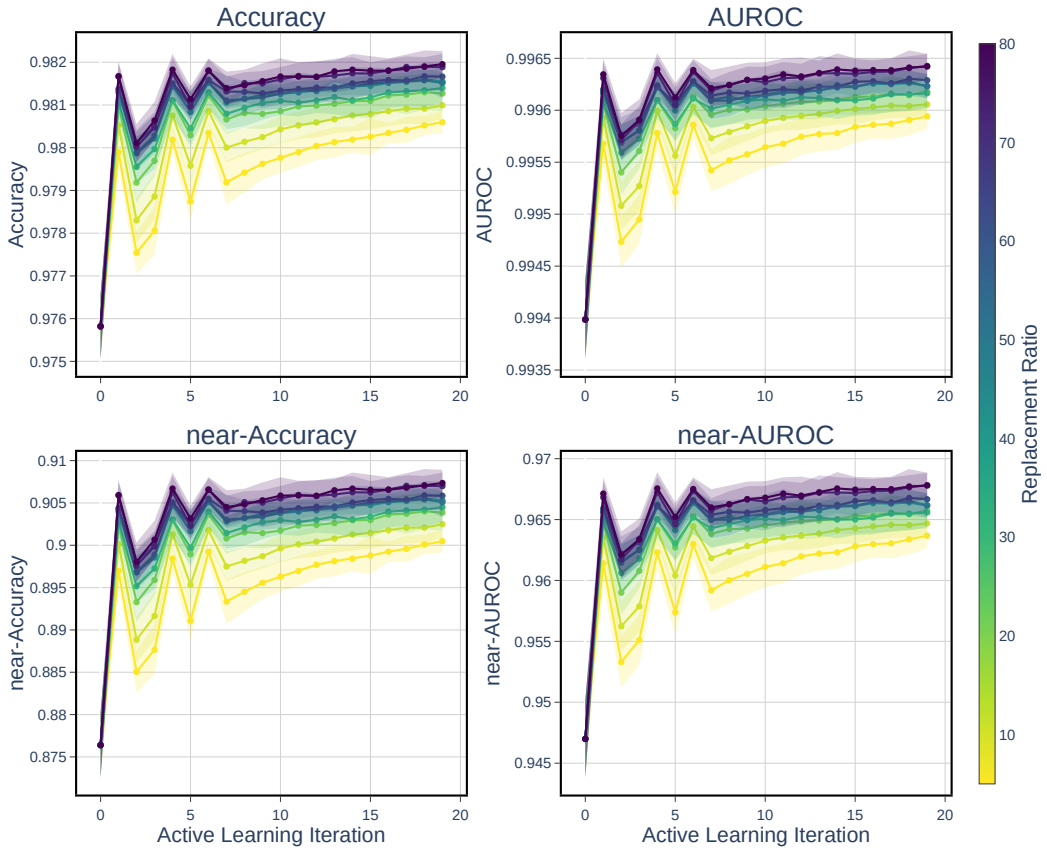

Figure A.1: Performance evaluation of model training with different replacement ratios over active learning iterations, showing Accuracy, AUROC, near-Accuracy, and near-AUROC metrics.

A.2 Impact of Exploration-Exploitation Balance on Model Performance

The exploration-exploitation trade-off is a widely studied challenge across many research domains, particularly in active learning, where finding the right balance between these competing factors is crucial. In our proposed active learning-based training procedure, generating data points near collision boundaries can be interpreted as exploitation, while sampling data points from a uniform distribution represents exploration. While our current approach relies on exploration only through the initial dataset, it may be beneficial to incorporate some exploration into our active learning-based training procedure. To investigate this possibility, we conducted additional experiments.

In each active learning iteration, k new data points are generated and incorporated into the dataset. We modified the sampling process by varying the proportion of these k points that come from uniform sampling (exploration) rather than boundary sampling. We tested exploration ratios ranging from 0% (equivalent to our original proposed method) to 100% (purely uniform sampling), using the same experimental setup and model configuration as described in Appendix A.1.

Figure A.2 presents the results across different exploration ratios. Each experiment was repeated five times with different random seeds, with solid lines showing the mean performance and shaded regions indicating the standard deviation. The results show that the original method (0% exploration ratio) and configurations with low exploration ratios (10% and 20%) achieve superior performance across all metrics. Performance tends to decline as the exploration ratio increases, with purely exploratory sampling (100%) showing the poorest results. This indicates that the performance improvements from our active learning-based training procedure stem primarily from effectively focusing

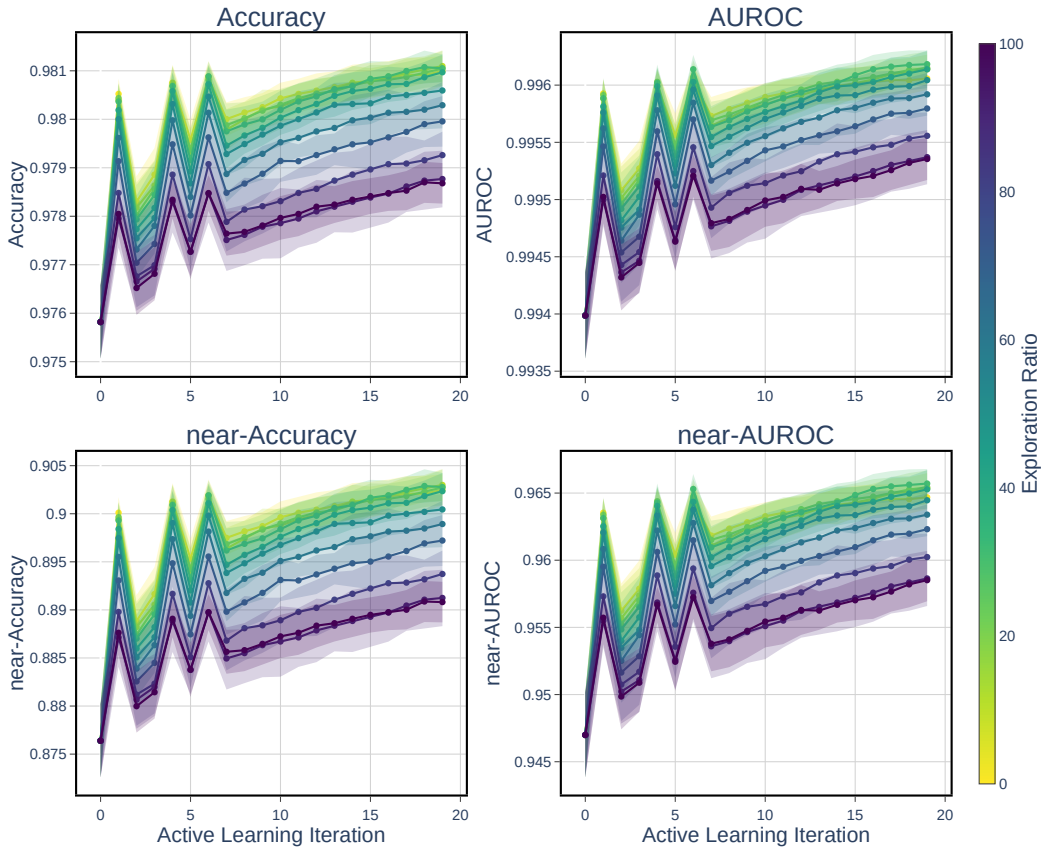

Figure A.2: Performance evaluation of model training with different exploration ratios over active learning iterations, showing Accuracy, AUROC, near-Accuracy, and near-AUROC metrics.

on critical regions rather than simply utilizing more data points. While pure exploitation performed well in our experiments, incorporating some level of exploration might be beneficial for robot systems with more complex collision area in the configuration space, where relying solely on exploitation could potentially be problematic.

A.3 Comparison with Existing Balanced Dataset Generation Method

Most data-driven methods for learning collision distance functions generate datasets randomly, without regard for the collision boundary. An exception is found in [27, 28], where they generate a balanced dataset. Their approach aims for a composition of 50% collision data points ($d_{\text{col}} < 1\text{cm}$), 35% near-collision (but non-collision) data points ($1\text{cm} < d_{\text{col}} < 5\text{cm}$), and 15% non-collision data points ($d_{\text{col}} > 5\text{cm}$).

This generation process relies on rejection sampling: for near-collision joint configurations, a joint configuration is first sampled from a uniform distribution within the joint limits. It is accepted if its collision distance falls within the near-collision range ($1\text{cm} < d_{\text{col}} < 5\text{cm}$) and rejected otherwise. The efficiency of this rejection sampling heavily depends on the disparity between the sampling distribution (uniform within joint limits) and the target distribution ($1\text{cm} < d_{\text{col}} < 5\text{cm}$).

If the volume of the near-collision region in the joint configuration space is relatively small, the probability of rejecting sampled data points increases significantly. Consequently, the number of required samples and collision distance calculations can far exceed the desired number of data points. In our multi-arm robot systems, we need to calculate collision distances more than 9 times the desired number of data points (one million) while generating the balanced datasets. This inefficiency can worsen further as the near-collision region's volume in the joint configuration space decreases, potentially

Table A.2: Performance comparison: our active learning-based training versus balanced dataset approach

Env.	Model	Training	Accuracy (ϵ=0.0)	AUROC	near-Accuracy (ϵ=0.0)	near-AUROC
Fig. 3.5 (Two arms)	JointNN	*none*	0.9765	0.9942	0.8799	0.9491
		balanced	0.9767	0.9942	0.8854	0.9552
		active	**0.9809**	**0.9960**	**0.9018**	**0.9643**
Fig. 3.5 (Three arms)	JointNN	*none*	0.9691	0.9868	0.8189	0.8875
		balanced	0.8887	0.9118	0.6422	0.7229
		active	**0.9765**	**0.9918**	**0.8609**	**0.9280**

leading to even higher sampling and calculation requirements.

Despite the considerable time required for generation, we produced a balanced dataset and conducted experiments to evaluate the performance of two models: one trained with the existing balanced dataset approach, and another trained using our proposed active learning-based training framework.

Table A.2 presents a performance comparison between our active learning-based training approach and the balanced dataset approach. The results reveal several key insights. Firstly, for both target environments, the balanced dataset approach shows no significant performance improvement compared to the uniformly sampled dataset (denoted as *none*). In fact, for the three-arm robot system, the balanced dataset approach performs worse than the uniform sampling. In contrast, our active learning-based training procedure successfully enhances performance across all metrics, with particularly notable improvements in the near-collision metrics (near-Accuracy and near-AUROC). It's worth noting that during our experiments, we observed that training with the balanced

dataset was relatively unstable, resulting in performance variations across different random seeds. The results presented in Table A.2 for the balanced approach represent the best outcomes from five training trials. In summary, our method of iteratively sampling points near the estimated collision boundary and updating the training dataset not only allows for more efficient computation in dataset generation but also leads to more stable training and more effective performance improvement. This approach proves superior to the balanced dataset method, which samples points near the ground truth collision boundary only once for the initial dataset.

B

Appendix: PairwiseNet

B.1 Hyperparameters of PairwiseNet and Baseline Methods

Table B.1 presents the hyperparameters employed in our experiments for training PairwiseNet and baseline methods. For PairwiseNet, we used a batch size of 1000, a learning rate of 1e-3, and trained for 2000 epochs. In contrast, ClearanceNet, one of our baseline methods, was trained with a smaller batch size of 191, a lower learning rate of 1.75e-4, and for 400 epochs, which are reported as the optimal values in [31]. Other neural network baselines were trained with a batch size of 10000, a learning rate of 1e-3, and for 10000 epochs. Specific to PairwiseNet's architecture, hyperparameters of EdgeConv layers are set to the default values provided by [46]. Each shape element in our point cloud data representation comprises 100 points, providing a detailed yet manageable geometric description.

Table B.1: Hyperparameters for training PairwiseNet and baseline methods

hyperparameter	value
batch size, learning rate, epoch for PairwiseNet	1000, 1e-3, 2000
batch size, learning rate, epoch for ClearanceNet	191, 1.75e-4, 400
batch size, learning rate, epoch for other NN baselines	10000, 1e-3, 10000
k for k-nearest neighbor of EdgeConv layers	5
# of points in the point cloud data of an shape element	100
hidden nodes of EdgeConv layers	64

B.2 Geometric Representations of Robot Links for Collision Checking

For efficient collision checking, we construct three different geometric representations of the robot links: original mesh, convex mesh, and capsule primitives. The original mesh representation preserves all geometric details of each robot link, including non-convex features, ensuring the most accurate collision distance calculations. For faster computation, we create simplified convex meshes by computing the convex hull of each original mesh, which reduces geometric complexity while maintaining a reasonable approximation of the link shape. As shown in Table B.2, this convex hull approximation significantly reduces the geometric complexity - the number of vertices and triangles in the convex meshes are about 1/300 of those in the original meshes. Figure B.1 illustrates these geometric representations on the three-arm robot system, demonstrating how the convex hull approximation (middle) simplifies the detailed original mesh geometry (left) while preserving the essential shape characteristics for collision checking.

For even faster collision distance calculations, we also construct capsule-shaped collision primitives for each robot link (right in Figure B.1). Following the methodology

Table B.2: Geometric complexity of original and simplified convex meshes for Panda robot links

	# of vertices / # of triangles	
	Original mesh	Convex mesh
Link 0	59517 / 20483	102 / 200
Link 1	37309 / 12516	152 / 300
Link 2	37871 / 12716	152 / 300
Link 3	42510 / 14233	152 / 300
Link 4	43506 / 14621	152 / 300
Link 5	54751 / 18327	152 / 300
Link 6	63895 / 21620	102 / 200
Link 7	35768 / 12082	102 / 200

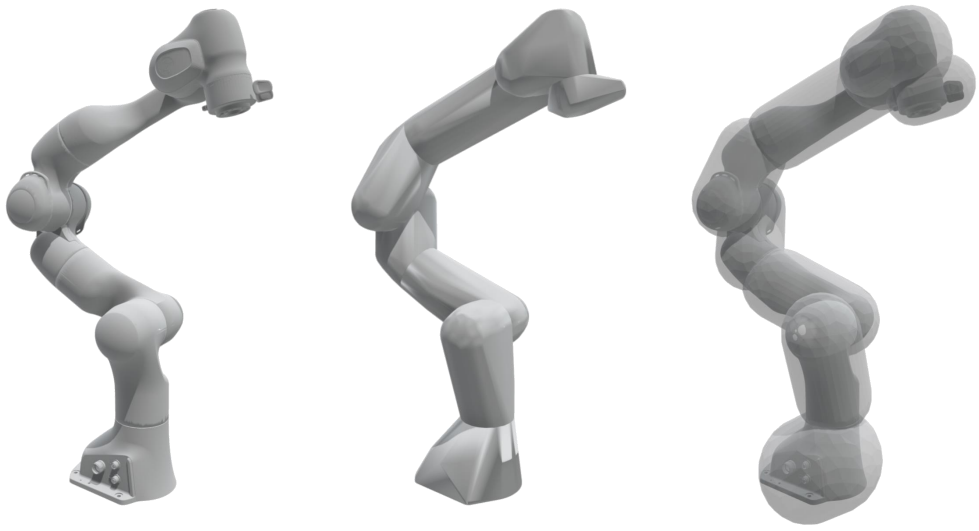

Figure B.1: Three geometric representations of Panda robot links.

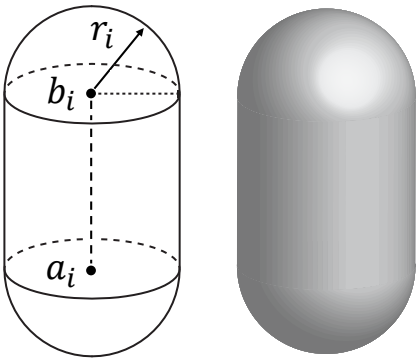

Figure B.2: Illustration of capsule geometry with optimized parameters. The capsule is defined by two end points (a_i and b_i) and a radius (r_i).

of [38], we formulate an optimization problem to create minimal volume capsules that encapsulate all vertices of the link meshes:

$$\underset{a_i,b_i,r_i}{\text{minimize}} \quad ||a_i - b_i||\pi r_i^2 + \frac{4}{3}\pi r_i^3 \tag{B.2.1}$$

$$\text{subject to} \quad \text{dist}(p, \overline{a_i b_i}) \leq r_i, \quad \forall p \in \mathcal{V}_i \tag{B.2.2}$$

Here, i denotes the link index, $\mathcal{V}_i$ represents the vertices of the i^{th} link mesh, and $a_i, b_i \in \mathbb{R}^3$ and $r_i \in \mathbb{R}$ refer to the two endpoints and the radius of a capsule, respectively. The line segment connecting the two endpoints is denoted as $\overline{a_i b_i}$. Using these capsule approximations, the collision distance calculation reduces to finding the minimum distance between capsules, which is computationally much simpler than mesh-based calculations.

The geometric approximations using convex meshes and capsule primitives, and our proposed PairwiseNet, are all approximations of the true collision distances computed from original detailed meshes. To evaluate the accuracy of these approximation methods, we measured the mean squared error (MSE) between the true collision distances

Table B.3: Comparison of collision distance estimation errors across different geometric approximations and PairwiseNet

	Approximation error (MSE)		
	Two arms	Three arms	Four arms
Convex mesh	4.55e-4	2.27e-4	1.61e-4
Capsule	1.38e-3	1.05e-3	3.37e-3
PairwiseNet	**0.24e-4**	**0.24e-4**	**0.46e-4**

and the approximated distances from each method. As shown in Table B.3, all methods demonstrate relatively small approximation errors across different multi-arm configurations. While the convex mesh approximation maintains reasonable accuracy by preserving the overall shape of the links, the capsule primitives show slightly larger errors due to their simplified geometric representation. PairwiseNet achieves comparable or better approximation accuracy, suggesting that it can serve as a novel alternative to classical methods that rely on simplified geometric representations.

B.3 The Collision-free Guaranteed Threshold

The collision-free guaranteed threshold ϵ_{safe} refers to a predefined distance value that is established in collision distance estimation methods. This threshold is set to ensure that during testing or actual operation, the estimated collision distance remains above this threshold for all valid configurations or movements of the robot system. In other words, if the estimated collision distance between the robot and any obstacles remains above the collision-free guaranteed threshold ($F_\psi(q; f_\psi, \mathcal{S}) > \epsilon_{\text{safe}}$), it is considered safe and collision-free. In our experiments, we set the collision-free guaranteed threshold to the least conservative value that allows us to classify all the collision configurations in the

Table B.4: The collision-free guaranteed thresholds of PairwiseNet and baseline methods

Methods	Two arms	Three arms	Four arms
Capsule	0.0	0.0	0.0
JointNN	0.2111	0.2015	0.2231
PosNN	0.1141	0.1189	0.1756
jointNERF	0.1661	0.1734	0.2001
ClearanceNet	0.2840	0.3713	0.4944
DiffCo	-1.2789	-1.4672	-0.9535
PairwiseNet	0.0150	0.0152	0.0184

test dataset as collisions. These thresholds are then utilized for measuring the Safe-FPR.

B.4 Training Complexity of PairwiseNet

We utilized 3 million data points (3 million shape element pairs with their corresponding distances) to train PairwiseNet for collision distance estimation in multi-arm robot systems and for single-arm systems with obstacles. For a more detailed description of the complexity of PairwiseNet's training process, additional information is presented in Table B.5.

- *Unique element pairs* refers to the count of element pairs in the system, excluding those with duplicate shapes (for example, the pair of the robot arm's seventh link with the top shelf plate and the pair with the middle shelf plate are considered the same since the shapes of the top and middle plates are identical). The more unique element pairs, the greater the number of pairwise collision distances PairwiseNet must learn, resulting in higher training complexity.

Table B.5: Training Complexity of PairwiseNet

Training Env.	Multi-arm	Multi-arm	Single arm w/ obstacles	Two arms w/ obstacles
Unique element pairs	36	36	70	64
Training data points	1,000,000	3,000,000	3,000,000	1,000,000
Gradient steps	2,860,000	4,286,000	4,286,000	2,860,000
Training time elapsed (h)	24.8	35.6	36.1	23.3
Validation loss (MSE)	1.46e-5	1.43e-5	8.03e-6	9.82e-6

- *Gradient step* refers to the number of times the learnable parameters were updated to minimize the loss during the training process.
- *Training time elapsed* refers to the total time taken to complete the training.
- The table also includes *Validation loss* to represent each training result.

The training time was measured on an environment with AMD Ryzen 9 7950X (16 cores, 32 threads), NVIDIA RTX 4090, and 125GB of RAM environment.

We conducted additional experiments to analyze the training complexity of PairwiseNet. Initially, for the existing multi-arm robot systems, while we began with 3 million data points for training PairwiseNet, we also conducted experiments using fewer data points (1 million) and fewer gradient steps. While the training time decreased, the learning results were comparable to those with the original 3 million data points.

Next, we examined the previously mentioned single-arm system with obstacles. Although there were many unique element pairs (70), since all obstacle elements were rectangular, they were relatively easier to learn. Additionally, we trained PairwiseNet with a two-arm robot system, adding non-rectangular household objects as obstacles (as shown in Figure B.3). Despite using merely a total of 1 million data points and fewer

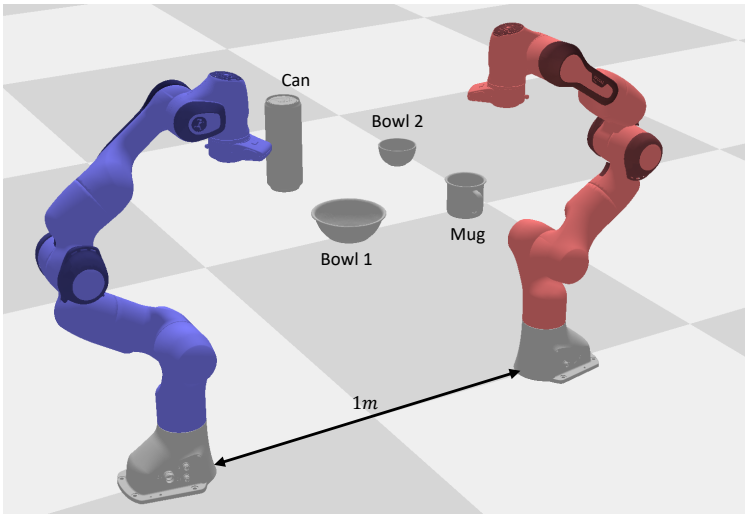

Figure B.3: A two-arm robot system with four household objects

gradient steps in this scenario compared to the original PairwiseNet, the validation loss was successfully reduced to 9.82e-6, confirming successful learning.

B.5 Comparison with Direct Point Cloud Distance Computation

This section compares PairwiseNet's approach to collision distance estimation with a direct method of computing pairwise distances between every point in the point clouds. While the direct method might seem intuitive, PairwiseNet offers several advantages in terms of efficiency and scalability. The following subsections will examine these advantages in detail, focusing on GPU memory consumption, inference complexity, and inference time on CPU.

Table B.6: Inference time comparison: PairwiseNet versus direct point cloud distance computation

Method	Inference time for 1000 joint poses (s)					
	Two arms (64 pairs)		Three arms (192 pairs)		Four arms (384 pairs)	
	CPU	GPU	CPU	GPU	CPU	GPU
Direct distance	50.14	0.2325	151.4	0.3903	308.9	0.7966
Direct distance (Batch)	55.68	out of memory	174.7	out of memory		
PairwiseNet	0.1054	0.0988	0.1590	0.0998	0.2025	0.1045
PairwiseNet (Batch)	0.0362	0.0207	0.1070	0.0224	0.2121	0.0253

GPU Memory Consumption. PairwiseNet is designed for efficient memory usage, requiring only 67,585 parameters for the regressor network and the shape feature vectors, totaling approximately 270kB. This efficiency is achieved by encoding each point cloud data into a 32-dimensional feature vector, eliminating the need to store all point cloud data in memory. In contrast, computing pairwise distances between every point in two point clouds would necessitate storing all point cloud data in GPU memory, leading to significantly higher memory requirements. These higher memory requirements might harm the batch computation capability of collision distances and their derivatives, which is crucial for efficient planning and control algorithms. Our experimental results, as shown in Table B.6, demonstrate that direct distance computation in batch mode results in out-of-memory errors for all multi-arm robot systems, even when utilizing Geforce RTX4090 with 24GB VRAM. PairwiseNet's memory efficiency, on the other hand, allows for larger batch sizes, enabling more effective parallel processing of multiple configurations simultaneously, even for complex multi-arm scenarios.

Inference Complexity. The computational complexity and required memory usage of direct point cloud distance calculation grows quadratically $O(M^2)$ with the number of points M in each point cloud. This scaling makes the method less efficient for large point clouds. Conversely, PairwiseNet's inference complexity is independent of the number of points in the point cloud data, offering better scalability. This results in significantly longer inference times for direct computation compared to PairwiseNet. In our experiments with 100 points per cloud, as shown in Table B.6, direct computation on GPU took significantly longer than PairwiseNet – approximately 2.4 times longer for two arms, 3.9 times longer for three arms, and 7.6 times longer for four arms. Notably, the performance gap widens for more complex robot systems containing more element pairs, as evidenced by the increasing time difference across two-, three-, and four-arm configurations. This indicates that the inference time of direct point cloud distance calculation is more sensitive to the number of element pairs in the system than PairwiseNet. The performance gap becomes even more significant for batch mode operations, which are not feasible for direct point cloud distance calculation due to its large GPU memory requirements.

Inference time on CPU PairwiseNet maintains short inference times even on CPU, demonstrating its versatility across different hardware configurations. Our experimental results, as shown in Table B.6, highlight the stark contrast in CPU performance between PairwiseNet and direct point cloud distance calculation. For the two-arm configuration, PairwiseNet completes inference in just 0.1054 seconds, while direct calculation takes 50.14 seconds - a difference of nearly 476 times. This performance gap widens further for more complex systems: in the three-arm scenario, PairwiseNet takes 0.1590 seconds compared to 151.4 seconds for direct calculation (952 times slower), and in the four-arm case, the times are 0.2025 seconds and 308.9 seconds respectively (1525 times slower).

These results demonstrate that direct pairwise calculation struggles to achieve comparable efficiency without powerful GPU support, limiting its applicability in scenarios where GPU resources are constrained or unavailable.

PairwiseNet's design offers a balance between accuracy, computational efficiency, and memory requirements. It provides a practical and versatile solution that scales well with increasing point cloud size and remains efficient across various hardware configurations. This comparison underscores PairwiseNet's advantages in scenarios where computational resources or memory are constrained, making it a more adaptable solution for real-world robotics applications.

B.6 Generalization Performance on Unseen Objects

PairwiseNet's architecture suggests potential generalization capabilities to unseen objects through its encoder component, which transforms object geometries into shape feature vectors. While not the primary focus of this work, generalization to unseen objects would significantly enhance PairwiseNet's practical applicability by enabling collision detection across diverse real-world environments without requiring retraining. This potential makes it worthwhile to investigate PairwiseNet's performance on unseen objects.

To evaluate PairwiseNet's generalization capability to unseen objects, we designed two test environments as shown in Fig. B.4. Environment (a) consists of a Panda arm with three boxes of different sizes (10cm, 20cm, and 30cm), which was used during training. Environment (b) introduces two unseen boxes with different dimensions (15cm and 25cm) to test the model's ability to handle unseen objects. Table B.7 presents the performance evaluation results in both environments. When tested on environment (b), the performance shows some inevitable degradation but maintains reasonable accuracy, achieving an MSE of 2.43e-4 and AUROC of 0.9962. This maintained performance

on unseen objects suggests that PairwiseNet successfully learns generalizable geometric features rather than merely memorizing specific configurations. These results indicate the potential for broader generalization capabilities if the training dataset includes various shape primitives and other robot arm link geometries, enabling the model to handle diverse objects in real-world applications.

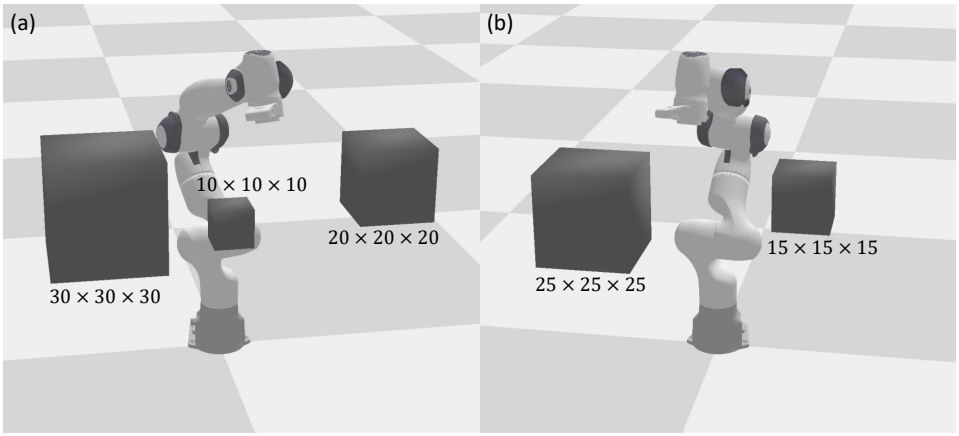

Figure B.4: Test environments for evaluating shape generalization capability: (a) Training environment with three boxes of size 10cm, 20cm, and 30cm alongside a Panda arm, and (b) Test environment with two unseen boxes of size 15cm and 25cm.

Table B.7: Performance evaluation of PairwiseNet's generalization capability between environments with different objects

	MSE	AUROC	Accuracy ($\epsilon = 0$)	safe-FPR
Fig.B.4 (a)	3.82e-5	0.9990	0.9845	0.1340
Fig.B.4 (b)	2.43e-4	0.9962	0.9698	0.1917

Bibliography

[1] KM Lynch. *Modern Robotics*. Cambridge University Press, 2017.

[2] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[3] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[4] Jinwook Huh and Daniel D Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–69. IEEE, 2016.

[5] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[6] Olivier Stasse, Adrien Escande, Nicolas Mansard, Sylvain Miossec, Paul Evrard, and Abderrahmane Kheddar. Real-time (self)-collision avoidance task on a hrp-2 humanoid robot. In *2008 ieee international conference on robotics and automation*, pages 3200–3205. IEEE, 2008.

[7] Alexander Dietrich, Thomas Wimbock, Alin Albu-Schaffer, and Gerd Hirzinger. Integration of reactive, torque-based self-collision avoidance into a task hierarchy. *IEEE Transactions on Robotics*, 28(6):1278–1293, 2012.

[8] Cheng Fang, Alessio Rocchi, Enrico Mingo Hoffman, Nikos G Tsagarakis, and Darwin G Caldwell. Efficient self-collision avoidance based on focus of interest for

humanoid robots. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1060–1066. IEEE, 2015.

[9] Juan José Quiroz-Omaña and Bruno Vilhena Adorno. Whole-body control with (self) collision avoidance using vector field inequalities. *IEEE Robotics and Automation Letters*, 4(4):4048–4053, 2019.

[10] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Kang Shin and Neil McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control*, 31(6):491–500, 1986.

[12] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. Path planning with modified a star algorithm for a mobile robot. *Procedia engineering*, 96:59–69, 2014.

[13] Tang XiangRong, Zhu Yukun, and Jiang XinXin. Improved a-star algorithm for robot path planning in static environment. In *Journal of Physics: Conference Series*, volume 1792, page 012067. IOP Publishing, 2021.

[14] Byeongho Lee, Yonghyeon Lee, Seungyeon Kim, MinJun Son, and Frank C Park. Equivariant motion manifold primitives. In *7th Annual Conference on Robot Learning*, 2023.

[15] Yonghyeon Lee, Byeongho Lee, Seungyeon Kim, and Frank C Park. Motion manifold flow primitives for language-guided trajectory generation. *arXiv preprint arXiv:2407.19681*, 2024.

[16] Yonghyeon Lee. Mmp++: Motion manifold primitives with parametric curve models. *IEEE Transactions on Robotics*, 2024.

[17] Evan Prianto, MyeongSeop Kim, Jae-Han Park, Ji-Hun Bae, and Jung-Su Kim. Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay. *Sensors*, 20(20):5911, 2020.

[18] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022.

[19] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[20] Ermano Arruda, Michael J Mathew, Marek Kopicki, Michael Mistry, Morteza Azad, and Jeremy L Wyatt. Uncertainty averse pushing with model predictive path integral control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 497–502. IEEE, 2017.

[21] Corrado Pezzato, Chadi Salmi, Max Spahn, Elia Trevisan, Javier Alonso-Mora, and Carlos Hernández Corbato. Sampling-based model predictive control leveraging parallelizable physics simulations. *arXiv preprint arXiv:2307.09105*, 2023.

[22] Seungyeon Kim, Byeongdo Lim, Yonghyeon Lee, and Frank C Park. Se (2)-equivariant pushing dynamics models for tabletop object manipulations. In *Conference on Robot Learning*, pages 427–436. PMLR, 2023.

[23] Seungyeon Kim, Young Hun Kim, Yonghyeon Lee, and Frank C Park. Leveraging 3d reconstruction for mechanical search on cluttered shelves. In *7th Annual Conference on Robot Learning*, 2023.

[24] Nikhil Das, Naman Gupta, and Michael Yip. Fastron: An online learning-based model and active learning strategy for proxy collision detection. In *Conference on Robot Learning*, pages 496–504. PMLR, 2017.

[25] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[26] Yuheng Zhi, Nikhil Das, and Michael Yip. Diffco: Autodifferentiable proxy collision detection with multiclass labels for safety-aware trajectory optimization. *IEEE Transactions on Robotics*, 38(5):2668–2685, 2022.

[27] Nadia Barbara Figueroa Fernandez, Seyed Sina Mirrazavi Salehian, and Aude Billard. Multi-arm self-collision avoidance: A sparse solution for a big data problem. In *Proceedings of the Third Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop*, 2018.

[28] Mikhail Koptev, Nadia Figueroa, and Aude Billard. Real-time self-collision avoidance in joint space for humanoid robots. *IEEE Robotics and Automation Letters*, 6(2):1240–1247, 2021.

[29] Javier Muñoz, Peter Lehner, Luis E Moreno, Alin Albu-Schäffer, and Máximo A Roa. Collisiongp: Gaussian process-based collision checking for robot motion planning. *IEEE Robotics and Automation Letters*, 8(7):4036–4043, 2023.

[30] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. Relaxedik: Real-time synthesis of accurate and feasible robot arm motion. In *Robotics: Science and Systems*, volume 14, pages 26–30. Pittsburgh, PA, 2018.

[31] J Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust. Neural collision clearance estimator for batched motion planning. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 73–89. Springer, 2020.

[32] Yeseung Kim, Jinwoo Kim, and Daehyung Park. Graphdistnet: A graph-based collision-distance estimator for gradient-based trajectory optimization. *IEEE Robotics and Automation Letters*, 7(4):11118–11125, 2022.

[33] Jihwan Kim and Frank Chongwoo Park. Active learning of the collision distance function for high-dof multi-arm robot systems. *Robotica*, pages 1–15, 2024.

[34] Jihwan Kim and Frank C Park. Pairwisenet: Pairwise collision distance learning for high-dof robot systems. In *Conference on Robot Learning*, pages 2863–2877. PMLR, 2023.

[35] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[36] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, page 209, 2001.

[37] Panpan Cai, Chandrasekaran Indhumathi, Yiyu Cai, Jianmin Zheng, Yi Gong, Teng Sam Lim, and Peng Wong. Collision detection using axis aligned bounding boxes. *Simulations, Serious Games and Their Applications*, pages 1–14, 2014.

[38] Antonio El Khoury, Florent Lamiraux, and Michel Taix. Optimal motion planning for humanoid robots. In *2013 IEEE international conference on robotics and automation*, pages 3136–3141. IEEE, 2013.

[39] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[40] Jing-Sin Liu, Wen-Hua Pan, Wen-Yang Ku, Y-H Tsao, and Y-Z Chang. Simulation-based fast collision detection for scaled polyhedral objects in motion by exploiting analytical contact equations. *Robotica*, 34(1):118–134, 2016.

[41] Charles J Geyer. Practical markov chain monte carlo. *Statistical science*, pages 473–483, 1992.

[42] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[43] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[44] Yiheng Han, Wang Zhao, Jia Pan, and Yong-Jin Liu. Configuration space decomposition for learning-based collision checking in high-dof robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5678–5684. IEEE, 2020.

[45] Qingyang Tan, Zherong Pan, and Dinesh Manocha. Lcollision: Fast generation of collision-free human poses using learned non-penetration constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3913–3921, 2021.

[46] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[48] Seungyeon Kim, Taegyun Ahn, Yonghyeon Lee, Jihwan Kim, Michael Yu Wang, and Frank C Park. Dsqnet: a deformable model-based supervised learning algorithm for grasping unknown occluded objects. *IEEE Transactions on Automation Science and Engineering*, 20(3):1721–1734, 2022.

국문초록

충돌 거리 추정은 로봇 경로 계획 및 장애물 회피에 있어 중요한 요소이다. 전통적인 기하학적 알고리즘은 단순한 시스템에는 효과적이지만, 복잡하고 고자유도를 가진 로봇 시스템에서는 계산 속도, 배치 처리, 미분 계산에 있어 한계를 보인다. 이러한 한계로 인해 충돌 거리 추정을 위한 학습 기반 접근법들이 개발되고 있다.

그러나 기존의 학습 기반 방법들은 복잡한 고자유도 로봇 시스템에 적용될 때 중요한 도전 과제에 직면한다. 이러한 과제들은 구성 공간의 지수적 증가로 인한 데이터셋 구축의 어려움, 충돌 거리 함수의 본질적 복잡성, 그리고 사소한 환경 변화에 대한 제한된 일반화 능력을 포함한다. 따라서 더욱 정교하고 적응 가능한 충돌 거리 추정 기술의 필요성이 대두되고 있다.

본 논문은 이러한 과제들을 해결하기 위한 두 가지 주요 기여를 제시한다. 첫째, 고자유도 로봇 시스템에서 효율적인 데이터셋 구축을 위한 능동 학습 전략을 소개한다. 이 방법은 충돌 경계 근처의 가장 정보가 풍부한 구성을 샘플링하는 데 중점을 두어, 훈련 데이터의 품질을 크게 향상시키고 특히 중요한 영역에서 모델 성능을 개선한다.

둘째, 쌍별 충돌 거리 학습을 위한 혁신적인 방법인 PairwiseNet을 제안한다. PairwiseNet은 전체 시스템의 충돌 거리를 직접 추정하는 대신, 로봇 시스템 내 요소 쌍 간의 최소 거리를 예측하는 데 중점을 둔다. 이 접근 방식은 학습 과제를 단순화하고 다양한 로봇 구성에 걸쳐 탁월한 일반화 능력을 보여준다.

다중 팔 구성 및 장애물이 많은 환경에서의 단일 팔 로봇을 포함한 고자유도 로봇 시스템에 대한 광범위한 실험을 통해 우리의 접근 방식의 효과성을 검증했다. 결과는 기존 방법들과 비교하여 충돌 거리 회귀 오류, 충돌 검사 정확도, 그리고 거짓 양성 비율에서 상당한 개선을 보여준다. 우리의 기여는 복잡한 로봇 시스템을 위한 충돌 거리 추정 분야의 최신 기술을 발전시키며, 더욱 정확하고 효율적이며 적응 가능한 솔루션을 제공한다.

주요어: 충돌 거리 추정, 능동 학습, 데이터셋 구성, 쌍별 충돌 거리 추정, 충돌 회피, 경로 계획.

학번: 2019-24947