Robotica (2024), **42**, pp. 4055–4069
doi:10.1017/S0263574723001790

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Active learning of the collision distance function for high-DOF multi-arm robot systems

Jihwan Kim and Frank Chongwoo Park

Department of Mechanical Engineering, Seoul National University, Seoul, South Korea
Corresponding author: Frank Chongwoo Park; Email: fcp@snu.ac.kr

Received: 17 March 2023; **Revised:** 31 October 2023; **Accepted:** 6 December 2023;
First published online: 12 January 2024

Keywords: path planning; high-DOF robot systems; robot collision; machine learning; active learning

Abstract

Motion planning for high-DOF multi-arm systems operating in complex environments remains a challenging problem, with many motion planning algorithms requiring evaluation of the minimum collision distance and its derivative. Because of the computational complexity of calculating the collision distance, recent methods have attempted to leverage data-driven machine learning methods to learn the collision distance. Because of the significant training dataset requirements for high-DOF robots, existing kernel-based methods, which require $O(N^2)$ memory and computation resources, where N denotes the number of dataset points, often perform poorly. This paper proposes a new active learning method for learning the collision distance function that overcomes the limitations of existing methods: (i) the size of the training dataset remains fixed, with the dataset containing more points near the collision boundary as learning proceeds, and (ii) calculating collision distances in the higher-dimensional link $SE(3)^n$ configuration space – here n denotes the number of links – leads to more accurate and robust collision distance function learning. Performance evaluations with high-DOF multi-arm robot systems demonstrate the advantages of the proposed active learning-based strategy vis-à-vis existing learning-based methods.

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 [1–3] potential field methods [4–8] and related hybrid approaches involving graph search and optimization [9, 10] 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's 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 [11], spheres [12], capsules [13], and other simple convex shapes [14] 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, for example, 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, that is, 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

© The Author(s), 2024. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

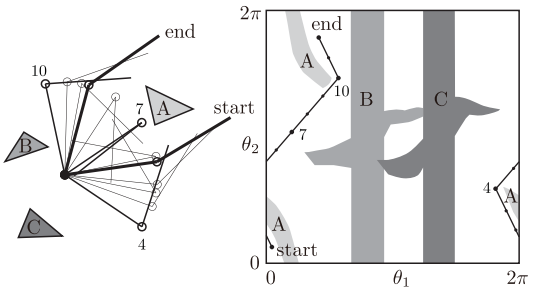


Figure 1. *The configuration space for a 2R planar robot [15]: The robot with workspace obstacles (left), and corresponding collision regions in the configuration space (right).*

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 Fig. 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, for example, support vector machine (SVM) models [12, 16], kernel perceptron models [17], and neural network (NN) models [7, 16, 18–21]. 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 ref. [17]. 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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

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 as follows:

- 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 2) followed by the problem formulation of collision distance learning and details of the neural network model (Section 3). Next, we describe our novel methods (Section 4). To validate the proposed methodology, numerical evaluations are performed for two high-DOF multi-arm robot systems. We also conducted a real-world experiment using a 7-DOF robot arm and obstacles to further validate the effectiveness of our approach. (Section 5).

2. Related works

A variety of techniques based on machine learning have been proposed to estimate collision distances and their derivatives. In ref. [7], 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 ref. [12], 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 ref. [18] to predict the self-collision cost by inputting the Cartesian joint positions as a concatenated vector. Similarly, the neural network model employed in ref. [19] utilizes the positional encoding vector of the joint configuration as its input. In ref. [16], 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 ref. [20], they take an extended configuration, which includes both joint and workspace configurations, as inputs. GraphDistNet [21] utilizes a graph neural network 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 [17] 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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

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. Collision distance learning

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 f_{GT} 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 $q_i \in \mathbb{R}^N$, that is, $d_i = f_{\text{GT}}(q_i)$. The f_{GT} 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 $f_{\text{GT}}(q)$ 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 f_{GT}. Finally, we optimize the parameters θ of the model f_θ by minimizing the mean-squared error 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 Eqs. (1) and (2).

$$\theta^* = \text{argmin}_\theta L(\mathcal{D}, f_\theta) \tag{1}$$

$$= \text{argmin}_\theta \frac{1}{|\mathcal{D}|} \sum_{(q_i,d_i)\in\mathcal{D}} ||f_\theta(q_i) - d_i||^2 \tag{2}$$

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 3.2), the input representation of the model (Section 4.1), and the active learning-based training method (Section 4.2).

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:

$$y = \phi(Wx + b) \tag{3}$$

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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

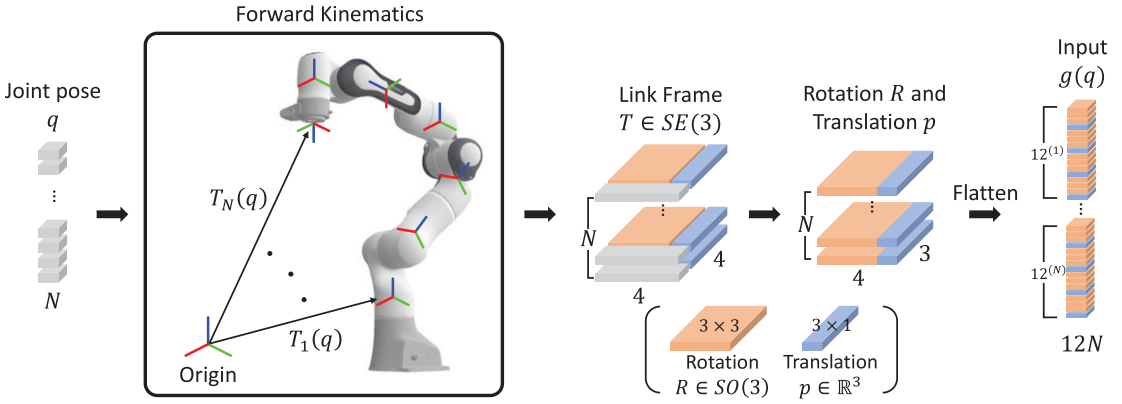


Figure 2. *An illustration of the link SE(3) configuration space representation mapping g. The joint configuration q is mapped to the collection of link frames $T_i(q)$ and transformed to an input vector g(q).*

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, 128, 128, 128), each followed by a ReLU activation function. The model can be formally expressed as:

$$f_\theta(q) = (NN_\theta \circ g)(q) \tag{4}$$

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

$$h^{(i)} = \phi(W^{(i)}h^{(i-1)} + b^{(i)}),\ i = 2, 3, 4 \tag{6}$$

$$h^{(1)} = \phi(W^{(1)}x + b^{(1)}) \tag{7}$$

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 [7, 16] or the position vector representation of joints [12, 18]). Our choice is detailed in Section 4.1.

4. Methods

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 Fig. 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 and 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

$$g(q)_{12(n-1)+4(i-1)+j} = T_{ij}^{(n)}(q). \tag{8}$$

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, for example, three-parameter representations or quaternion representations for the rotation matrix R.

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

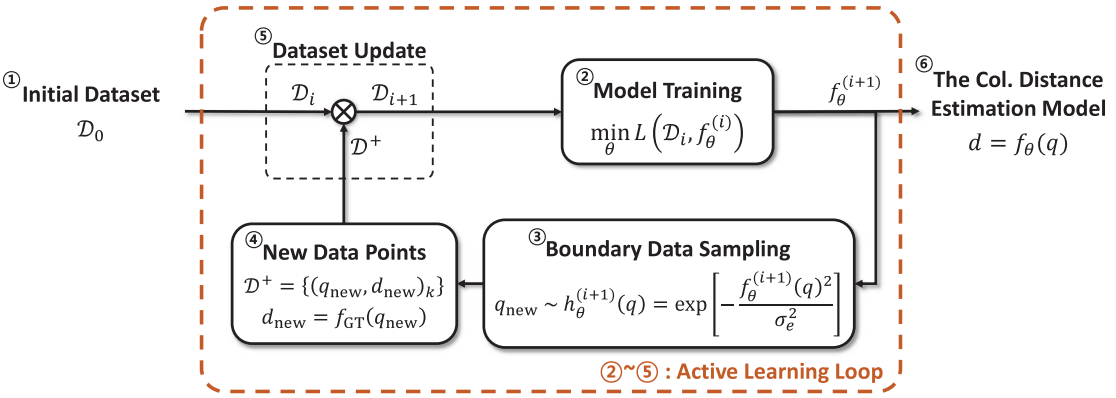


Figure 3. *An illustration of the active learning-based training procedure. The process starts with (1) an initial dataset and proceeds to (2) train the model using this data. Once trained, (3) the model generates new data points near the trained collision boundary. (4) The true collision distances of 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.*

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 better learning performance than other non-redundant representations (Section 5.4).

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 Fig. 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, f_{GT}. 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.

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 Fig. 4. We create a robot environment consisting of a 2R planar robot and obstacles, along with the ground-truth collision distance function f_{GT} for this system (Fig. 4(a)). We begin by initializing an initial dataset containing 500 data points. 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 Fig. 4(b). To improve the model, we employ an active learning-based training method that involves boundary data sampling and dataset update procedures. The target distribution $h_\theta^{(1)}(q)$ for boundary data sampling of the first model is illustrated in Fig. 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 Fig. 4(d) and (e), respectively. As the active learning loop continues, the updated dataset increasingly includes more near-collision data points. The plot in Fig. 4(g)

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

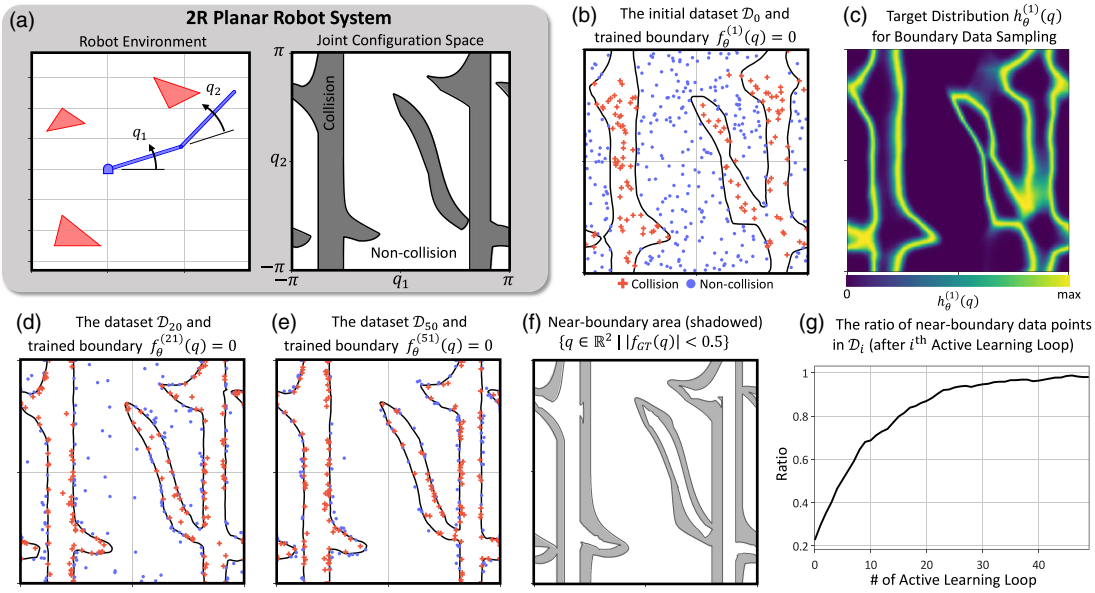


Figure 4. *(a) A 2R planar robot system and its joint configuration space. The gray region in the joint configuration space indicates the collision area, while the remaining space represents the non-collision area. (b) The initial dataset, $\mathcal{D}_0$, is uniformly sampled in the joint configuration space. The black line delineates the collision boundary of the model trained using the initial dataset. (c) The target (unnormalized) distribution $h_\theta^{(1)}(q)$ for the boundary data sampling procedure. The sampling probability for new data points is higher near the trained collision boundary. (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.*

represents the ratio of near-boundary data points in the dataset, with the near-boundary region displayed in Fig. 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:

$$q_{\text{new}} \sim h_\theta(q) = \exp\left[-\frac{|f_\theta(q)|^2}{\sigma_e^2} \right] \tag{9}$$

As shown in Fig. 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 (MCMC) [22], 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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

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 $\theta \rightarrow f_\theta^{(0)}$
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 $f_{\text{GT}}(q_{\text{new}})$
  $\mathcal{D}^+ \leftarrow \{(q_{\text{new}}, d_{\text{new}})_k\}$ ▷ 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 $h_\theta(q) = \exp\left[- \frac{|f_\theta(q)|^2}{\sigma_e^2} \right]$
  Initialize $q \leftarrow$ sample from Uniform($q_{\min}, q_{\max}$)
  $p_q \leftarrow h_\theta(q)$
  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($u_{\min}, 1$) ▷ Relaxed rejection
    if $\frac{p_{q^+}}{p_q} > u$ **then**
      $q \leftarrow q^+$
    end if
  end for
  return q
end function

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 [23].

The MH algorithm starts with an arbitrary point from the joint configuration space. A new point is proposed from an easily sampled distribution, for example, 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.

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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

Table I. *Hyperparameters.*

Name	Value
N_{active}	20
N_{epoch}	1000
k	100,000
σ_e	0.1
σ	0.05
u_{min}	0.8
max_step	1000

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 Fig. 4. Once the majority of training data points are located in the near-boundary area (Fig. 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.

5. Collision distance learning performance

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 refs. [7, 16]).
- **PosNN**: A neural network model that takes the Cartesian coordinates of the joint positions as inputs (the input representation used in refs. [12, 18]).
- **ClearanceNet** [20]: A neural network model composed of two fully connected layers with each followed by a dropout layer, taking the joint configuration as inputs.
- **DiffCo** [17]: 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 3.2. ClearanceNet and DiffCo, which are existing learning-based collision estimator models, are also used as baselines. Details on implementing ClearanceNet and DiffCo are consistent with those specified in refs. [17, 20].

We employ two distinct training procedures for the neural network models: the training procedure denoted as *active* is the proposed active learning-based training framework described in Section 4.2, while the training procedure denoted *none* corresponds to the standard training procedure using only the initial training dataset (without active learning). The hyperparameters used in these training procedures are presented in Table I. 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 is 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 Fig. 5. The first system features two Franka-Emika Panda robot

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

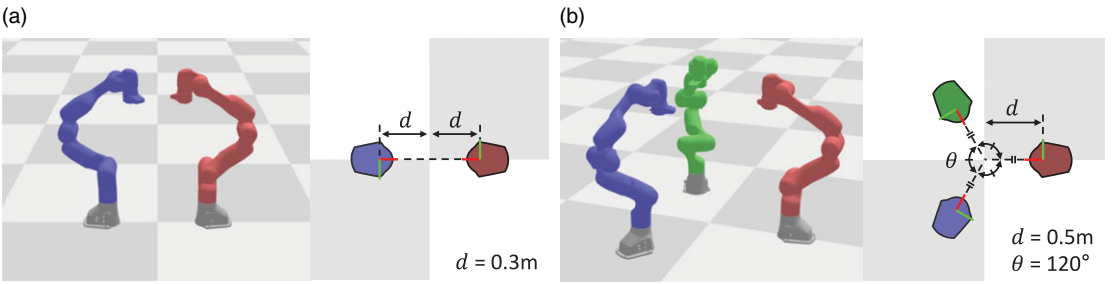


Figure 5. *An illustration of the target multi-arm systems. (a) Two 7-DOF Franka-Emika Panda robot arms resulting in a 14-DOF system. (b) Three 7-DOF robot arms resulting in a 21-DOF system.*

arms, each with 7-DOF, placed 0.6 m apart and facing each other. The second system features three Panda robot arms arranged equidistantly on a circle of radius 0.5 m 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 [24]. Using the collision distance calculation algorithm implemented in PyBullet [25], 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, 256 GB RAM, and NVIDIA GeForce RTX4090 with 24 GB VRAM).

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 tradeoff 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 -0.1 to 0.1 m, since accurately classifying collisions in the close-to-collision region is critical for actual robot manipulation tasks.

5.2. Evaluation results

We conduct a comparative analysis of the collision classification performance of our methods against baseline models. Table II 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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

Table II. *Collision distance learning performance.*

Env.	Model	Training	Accuracy ($\epsilon = 0.0$)	AUROC	near-Accuracy ($\epsilon = 0.0$)	near-AUROC
Figure 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	*none*	0.9862	0.9981	0.9293	0.9830
		active	**0.9886**	**0.9987**	**0.9416**	**0.9877**
Figure 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	*none*	0.9826	0.9958	0.8969	0.9619
		active	**0.9858**	**0.9971**	**0.9092**	**0.9701**

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.

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 III. 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*

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

Table III. *The training time, inference time, and required GPU memory.*

	Figure 5(a) (Two arms)			Figure 5(b) (Three arms)			
	Training time (h)	Inference time (ms)		Training time (h)	Inference time (ms)		Required GPU
Model		CPU	GPU		CPU	GPU	memory (GB)
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.

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 ref. [20], 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 24 GB 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 1 kHz control frequency, ensuring efficient real-time operation.

5.4. Compared to other non-redundant 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 non-redundant representations, namely quaternions and Euler angles. In mathematical notation, 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:

$$g_{\text{quat}}(q)_{7(n-1):7n} = [\texttt{to_quaternion}(R^{(n)}(q)), p^{(n)}(q)] \in \mathbb{R}^7, \tag{10}$$

$$g_{\texttt{Euler}}(q)_{6(n-1):6n} = [\texttt{to_Euler_angles}(R^{(n)}(q)), p^{(n)}(q)] \in \mathbb{R}^6. \tag{11}$$

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 IV 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 non-redundant representations.

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

Table IV. *Performance of other non-redundant input representations.*

Env.	Model	Accuracy ($\epsilon = 0.0$)	AUROC	near-Accuracy ($\epsilon = 0.0$)	near-AUROC
Figure 5(b)	QuatposNN	0.9819	0.9955	0.8930	0.9586
(Three arms)	EulerposNN	0.9818	0.9955	0.8923	0.9586
	SE3NN	**0.9826**	**0.9958**	**0.8969**	**0.9619**

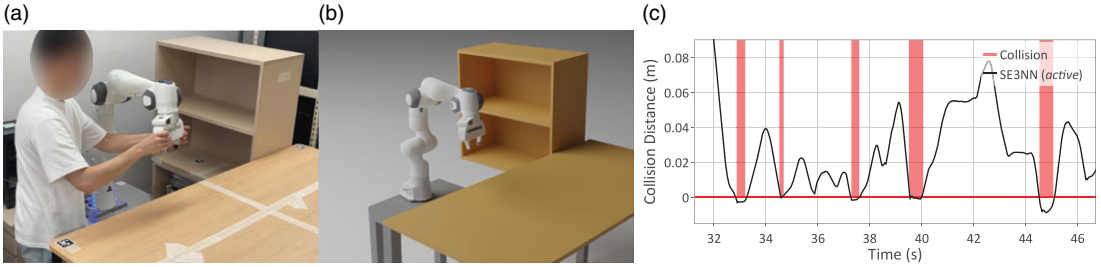


Figure 6. *An illustration of the real robot experiment. (a) A 7-DOF single-arm robot system with obstacles and (b) the corresponding simulation environment. (c) The plot demonstrates the collision labels and estimated collision distances of the proposed model (SE3NN with the active training procedure).*

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 (Fig. 6). The workspace was designed with complex obstacles like shelves and tables, introducing complexity to the robot arm's workspace. To safely generate close-to-collision trajectories, we conducted a human-guided demonstration as illustrated in Fig. 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 Sections 4 and 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 Fig. 6(b). The collision distance estimated by the trained SE3NN model and the real-world collision labels are plotted in Fig. 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 1 kHz, 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 $\epsilon = 0.0$.

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

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

active learning-based training method and the link $SE(3)$ configuration space representation are applied, demonstrates 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, for example, 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.

Author contributions. Jihwan Kim and Frank Chongwoo Park conceived and designed the study and experiments, and wrote the article.

Financial support. This work was supported in part by IITP-MSIT grant 2021-0-02068 (SNU AI Innovation Hub), IITP-MSIT grant 2022-0-00480 (Training and Inference Methods for Goal-Oriented AI Agents), KIAT grant P0020536 (HRD Program for Industrial Innovation), ATC + MOTIE Technology Innovation Program grant 20008547, SRRC NRF grant RS-2023-00208052, SNU-AIIS, SNU Interdisciplinary Program in AI, SNU-IAMD, SNU BK21+ Program in Mechanical Engineering, and SNU Institute for Engineering Research.

Competing interests. The authors declare no competing interests exist.

Ethical approval. Not applicable.

References

[1] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning (1998).
[2] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings, vol. **2** (2000) pp. 995–1001.
[3] J. Huh and D. D. Lee, "Learning High-Dimensional Mixture Models for Fast Collision Detection in Rapidly-Exploring Random Tree," *IEEE International Conference on Robotics and Automation* (2016) pp. 63–69.
[4] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. **2** (1985) pp. 500–505.
[5] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard and A. Kheddar, "Real-Time (Self)-Collision Avoidance Task on a HRP-2 Humanoid Robot," *2008 IEEE International Conference on Robotics and Automation* (2008) pp. 3200–3205.
[6] A. Dietrich, T. Wimbock, A. Albu-Schaffer and G. Hirzinger, "Integration of reactive, torque-based self-collision avoidance into a task hierarchy," *IEEE Trans. Rob.* **28**(6), 1278–1293 (2012).
[7] C. Fang, A. Rocchi, E. M. Hoffman, N. G. Tsagarakis and D. G. Caldwell, "Efficient Self-Collision Avoidance Based on Focus of Interest for Humanoid Robots," *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)* (2015) pp. 1060–1066.
[8] J. J. Quiroz-Omaña and B. V. Adorno, "Whole-body control with (self) collision avoidance using vector field inequalities," *IEEE Rob. Autom. Lett.* **4**(4), 4048–4053 (2019).
[9] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968).
[10] K. Shin and N. McKay, "A dynamic programming approach to trajectory planning of robotic manipulators," *IEEE Trans. Autom. Control* **31**(6), 491–500 (1986).
[11] P. Cai, C. Indhumathi, Y. Cai, J. Zheng, Y. Gong, T. S. Lim and P. Wong, "Collision Detection Using Axis Aligned Bounding Boxes," **In:** *Simulations, Serious Games and Their Applications* (2014) pp. 1–14.
[12] N. B. F. Fernandez, S. S. M. Salehian and A. Billard, "Multi-Arm Self-Collision Avoidance: A Sparse Solution for a Big Data Problem," *Proceedings of the Third Machine Learning in Planning and Control of Robot Motion (MLPC) Workshop. CONF* (2018).
[13] A. E. Khoury, F. Lamiraux and M. Taix, "Optimal Motion Planning for Humanoid Robots," *IEEE International Conference on Robotics and Automation* (2016) pp. 3136–3141.
[14] J. S. Liu, W. H. Pan, W. Y. 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).
[15] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, 1st ed. (Cambridge University Press, USA, 2017).
[16] M. Koptev, N. Figueroa and A. Billard, "Real-time self-collision avoidance in joint space for humanoid robots," *IEEE Rob. Autom. Lett.* **6**(2), 1240–1247 (2021).
[17] Y. Zhi, N. Das and M. Yip, "Diffco: Autodifferentiable proxy collision detection with multiclass labels for safety-aware trajectory optimization," *IEEE Trans. Rob.* **38**(5), 2668–2685 (2022).

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press

[18] D. Rakita, B. Mutlu and M. Gleicher, "RelaxedIK: Real-Time synthesis of accurate and feasible robot arm motion," *Robotics: Science and Systems* (2018), pp. 26–30.
[19] M. Bhardwaj, B. Sundaralingam, A. Mousavian, N. D. Ratliff, D. Fox, F. Ramos and B. Boots, "Storm: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation,"*Conference on Robot Learning. PMLR* (2022) pp. 750–759.
[20] J. C. Kew, B. Ichter, M. Bandari, T.-W. E. Lee and A. Faust, "Neural Collision Clearance Estimator for Batched Motion Planning," *International Workshop on the Algorithmic Foundations of Robotics* (2020) pp. 73–89.
[21] Y. Kim, J. Kim and D. Park, "GraphDistNet: A graph-based collision-distance estimator for gradient-based trajectory optimization," *IEEE Rob. Autom. Lett.* **7**(4), 11118–11125 (2022).
[22] C. J. Geyer, "Practical markov chain monte carlo," *Stat. Sci.* **7**(4), 473–483 (1992).
[23] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *Am. Stat.* **49**(4), 327–335 (1995).
[24] E. G. Gilbert, D. W. Johnson and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Rob. Autom.* **4**(2), 193–203 (1988).
[25] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," (2016-2021) http://pybullet.org.

Cite this article: J. Kim and F. C. Park (2024). "Active learning of the collision distance function for high-DOF multi-arm robot systems", Robotica **42**, 4055–4069. https://doi.org/10.1017/S0263574723001790

https://doi.org/10.1017/S0263574723001790 Published online by Cambridge University Press