Robotica (2024), **42**, pp. 1094–1112
doi:10.1017/S0263574724000109

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

End-to-end deep learning-based framework for path planning and collision checking: bin-picking application

Mehran Ghafarian Tamizi[1†], Homayoun Honari[2†], Aleksey Nozdryn-Plotnicki[3] and Homayoun Najjaran[1,2]

[1]Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada, [2]Department of Mechanical Engineering, University of Victoria, Victoria, BC, Canada, and [3]Apera AI, Vancouver, BC, Canada
Corresponding author: Homayoun Najjaran; Email: najjaran@uvic.ca

Received: 24 May 2023; **Revised:** 18 October 2023; **Accepted:** 20 December 2023; **First published online:** 13 February 2024

Keywords: Path planning; artificial neural network; collision checking; bin-picking; imitation learning; data aggregation

Abstract

Real-time and efficient path planning is critical for all robotic systems. In particular, it is of greater importance for industrial robots since the overall planning and execution time directly impact the cycle time and automation economics in production lines. While the problem may not be complex in static environments, classical approaches are inefficient in high-dimensional environments in terms of planning time and optimality. Collision checking poses another challenge in obtaining a real-time solution for path planning in complex environments. To address these issues, we propose an end-to-end learning-based framework viz., Path Planning and Collision checking Network (PPCNet). The PPCNet generates the path by computing waypoints sequentially using two networks: the first network generates a waypoint, and the second one determines whether the waypoint is on a collision-free segment of the path. The end-to-end training process is based on imitation learning that uses data aggregation from the experience of an expert planner to train the two networks, simultaneously. We utilize two approaches for training a network that efficiently approximates the exact geometrical collision checking function. Finally, the PPCNet is evaluated in two different simulation environments and a practical implementation on a robotic arm for a bin-picking application. Compared to the state-of-the-art path-planning methods, our results show significant improvement in performance by greatly reducing the planning time with comparable success rates and path lengths.

1. Introduction

Bin-picking, which includes object detection, motion planning, and grasping, is a crucial part of automation in industry. Bin-picking has been one of the trending research topics in recent years because of the challenges in image processing and path planning [1, 2]. Industrial bin-picking consists of a 2D/3D camera, which is used to collect the environmental information (object and obstacle detection), and a conveyor with the bins. The vision system is used to detect object positions and obstacle shapes, while the motion planning module calculates the path to the grasp position based on this information [3]. An efficient and real-time path for the manipulator can greatly improve the efficiency of mass production and assembly lines.

In a typical cycle of bin-picking operations, a combination of machine vision, inverse kinematics, and other algorithms are used to find the best grasping configuration for the robot. Once a configuration is found, path-planning algorithms generate a path from the robot's fixed home state to the grasp and then to a place state. Finally, the robot physically moves to complete the task. The serial and critical-path nature of path planning and movement means that planning time and execution time are essential factors

[†]MGT and HH contributed equally to this work.

that impact the overall cycle time and automation economics. It is crucial to ensure that robot paths are collision-free, considering both obstacles in the environment and self-collision.

Classical planners can be used for bin-picking, but they often fail to take advantage of the repetitive nature of the task. In a static environment, where a robot cell operates for extended periods, path planning is limited to fixed initial configurations and a fixed subspace of possible end-effector poses within a bin. This paper proposes a more efficient path planner in terms of the planning time for bin-picking tasks. We introduce the Path Planning and Collision Checking Network (PPCNet), a deep neural network-based tool specifically designed for repetitive tasks like bin-picking. The PPCNet generates waypoints along the path and estimates collisions faster, making it a reliable solution for autonomous systems such as robotic arms. By using PPCNet, we can generate a safe and efficient path in a relatively short amount of time. Our proposed method outperforms conventional approaches in terms of planning time and stays competitive in path quality and success rate, making it a viable solution for industrial bin-picking applications.

The following are the major features of this research:

- The proposed PPCNet end-to-end framework is an easy-to-implement and efficient approach in the context of path planning.
- The proposed post-processing dataset generation improves the quality of PPCNet paths.
- Collision detection is a critical and time-intensive aspect of path planning. PPCNet offers a faster solution, reducing planning time significantly through its ability to quickly detect collisions.
- We employ and compare two different methods for training the collision checking network to determine the best and safest approach for collision detection.

The paper is structured as follows: In Section 2, we provide a comprehensive overview of various path-planning techniques, discussing their advantages, disadvantages, and limitations for this problem. Section 3 outlines the problem definition. We then introduce our framework, PPCNet, in Section 4, where we also discuss the training process. In Section 5, we evaluate and compare the performance of our approach against four state-of-the-art path planners. Finally, we conclude our study in Section 6.

2. Related work

In this section, the related work will be discussed. In the first section, an overview of various path-planning techniques will be provided. Furthermore, the collision-checking methods are investigated in the second section. In each section, the general advantages, disadvantages, and limitations of the methods are mentioned.

2.1. Path planning

Pioneering collision-free path planning for robotic arms engaged in a wide spectrum of tasks, ranging from welding in industrial contexts to delicate medical applications, remains a central focal point in the realm of robotics [4]. path-planning methods can broadly be classified into two categories: classical and learning-based [5]. Classical path-planning approaches encompass a wide range of techniques, including artificial potential field [6], bio-inspired heuristic methods [7], and sampling-based path planners [8]. In contrast, learning-based path planners primarily utilize various machine-learning techniques to plan for the robot's path. These methods have been gaining popularity in recent years due to their ability to handle complex and dynamic environments.

Sampling-based methods are widely used in path planning. In this regard, they can be divided into two categories: single-query algorithms and multi-query algorithms. While single-query approaches aim to find a path between a single pair of initial and goal configurations, the multi-query ones try to be efficient when there are multiple pairs [9]. The Rapidly-exploring Random Tree (RRT) [10] and Probabilistic

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Road Map (PRM) [11] are two of the most well-known algorithms for single-query and multi-query approaches, respectively. PRM is a roadmap algorithm that aims to build an exhaustive roadmap of the configuration space to plan the path between any two configurations given to it. While PRM has been used for path planning of robotic arms before [12, 13], it has several limitations. Firstly, it generates paths that are far from the optimal solution. Secondly, generating the roadmap is computationally expensive [14], making it inefficient for path planning in relatively high-dimensional environments, even for static tasks such as bin-picking [15].

Furthermore, given an initial and goal configuration pair, RRT grows a tree of waypoints from the initial configuration in order to connect the two configurations. An important characteristic of RRT algorithm is its probabilistic completeness, meaning that it is guaranteed to find a path if there exists one [9]. Due to RRT's capability in non-convex high-dimensional spaces, it has been extensively employed in many studies for path planning of robotic arms [16–18]. To increase the performance of RRT in different aspects, such as planning time and optimality, there are many variants of this method in the literature. In ref. [19], the bi-directional RRT (Bi-RRT) is introduced for path planning of the 7-DOF manipulator. Moreover, in ref. [20] this algorithm was used for distributed picking application. This method simultaneously builds two RRTs; one of them attempts to find the path from the initial configuration to the final configuration, and the other one attempts to find the path in the opposite direction and tries to connect these two trees in each iteration. Although this method is faster than RRT, they both often produce suboptimal paths in practice and the quality of the path depends on the density of samples and the shape of the configuration space. In order to enhance the performance of the Bi-RRT method for manipulators, a pioneering study by Xu et al. [21] presents a novel modified version of this technique. The key innovation of their work lies in the integration of a sparse dead point saved strategy, effectively reducing the frequency of collision checks.

RRT* is the optimal version of the RRT and finds the asymptotically optimal solution if one exists [14]. While that is a strong feature of RRT*, it is inefficient in high-dimensional spaces and has a relatively slow convergence rate. As a result, it is not a suitable solution for the real-time path planning of a robotic arm. To overcome the problems mentioned above, biased sampling is one of the promising methods which can improve the performance of sampling-based path planners, due to the adaptive sampling of the configuration space of the robot. Batch-informed trees (BIT*) [22] and informed RRT* [23] are other significant variants of the RRT* that employ an informed search strategy and batch-processing approach to find the optimal path in a more efficient way by focusing the search on promising areas of the tree. Although this method is able to find the optimal solution, it is not suitable for real-time path planning since it requires a large amount of computational resources, which can lead to delays in the system's response time.

As mentioned above, since collision-checking is computationally expensive and the configuration space gets exponentially bigger with the increase of dimensionality, most of the classical sampling-based methods suffer from high computation and low convergence rates, making them impractical for use in a complex environment with high-dimensional configuration space and real-time applications such as bin-picking [24]. Ellekilde et al. [15] present a new algorithm for planning an efficient path in a bin-picking scenario based on using lookup tables. Although this method is almost instantaneous and faster than sampling-based planners, it is memory inefficient.

To this end, learning-based path planners have emerged to deal with the limitations of classical methods. Learning-based techniques can solve the long run-time problem of sampling-based methods by providing biased sampling nodes, allowing them to run faster and more efficiently [25]. For instance, Qureshi et al. [26] proposed a deep neural network-based sampler to generate nodes for sampling-based path planners. This work shows a significant improvement in the performance of sampling-based path planners in terms of planning time.

Reinforcement learning (RL) is a subcategory of learning-based techniques for path planning in an unknown environment [27–30] where the agent learns the optimal behavior through interactions with the environment and receiving reward signals. RL can be a promising solution when there is limited information regarding the environment. However, especially in the context of path planning, using RL

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

poses several limitations. Due to the trial-and-error nature of RL, the agent needs to strike a good balance between exploration and exploitation during its learning process. Hence, it requires a vast amount of data to learn a viable policy. Therefore, in addition to being sample-inefficient, it requires powerful hardware and significant memory for processing, especially when neural networks with a large number of parameters are employed. Moreover, when using DRL to solve the path-planning problem, the long horizon nature of it poses a great challenge in having an effective learning process. In its basic form, the reward shaping used to solve these problems are sparse functions where the agent receives a positive signal only when it has reached the goal. However, the DRL algorithm using this reward shaping scheme can often exhibit instability during training [31]. Furthermore, devising dense reward functions requires expert knowledge and engineering and a careful training routine since it can be susceptible to convergence to suboptimal policies [32, 33]. To this end, imitation learning is an alternative to these limitations. In imitation learning, the training dataset is provided by an expert during the execution of the task. For instance, in ref. [34], a recurrent neural network is trained by using demonstration from the virtual environment to perform a manipulation task.

Neural planners are relatively new methods in the path-planning context. These planners are based on deep neural networks, and their purpose is to solve the path-planning problem as an efficient and fast alternative to sampling-based methods. Bency et al. [35] proposed a recurrent neural network to generate an end-to-end near-optimal path iteratively in a static environment. Moreover, in refs. [36] and [37], motion planning networks (MPnet) is introduced. This method has a strong performance in unseen environments as the path can be learned directly.

2.2. Collision approximation

Collision detection is a vital component of path planning in robotics and can consume a significant amount of computation time. In fact, it has been reported that it can take up to 90% of the total path-planning time [8]. There are several methods for performing collision checking in path planning. Analytical methods like GJK (Gilbert-Johnson-Keerthi) algorithm [38] use mathematical equations and geometric models to determine if two objects will collide. They are often fast and efficient but can struggle with complex shapes and interactions. Grid-based methods like Voxel Grid [39] divide the environment into a grid of cells and check for collisions by looking at the occupied cells. This is often faster than analytical methods but can result in a loss of precision. Another simple method for collision detection is by approximating objects with overlapping spheres [40]. In this approach, the number of spheres used to represent each object is a crucial hyperparameter. Additionally, this method is more instantaneous than other traditional collision detection algorithms.

Recently, machine-learning techniques have been utilized to overcome the limitations of traditional collision detection methods in robotics. For example, support vector machines [41] have been used to calculate precise collision boundaries in configuration space. Gaussian mixture models were applied in another study [42], resulting in the path being generated five times faster than bi-directional RRT. K-Nearest neighbors is another machine-learning technique used in modeling configuration space for collision detection [43]. Fastron, a machine-learning model, was proposed by Das et al. [44] to model configuration space as a proxy collision detector, decomposing it into subspaces and training a collision checker for each region. The majority of previous studies in this field rely on decomposing the configuration space, which requires significant engineering and simplifications.

3. Problem definition

In this section, we will define the path-planning problem for the bin-picking application. Besides, the notation that we use in this paper will be introduced.

The first important concept in path planning is the configuration space (C_{space}). C_{space} includes all positions and configurations of the robot. We can define q as the specific configuration for an n-joints robotic arm:

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

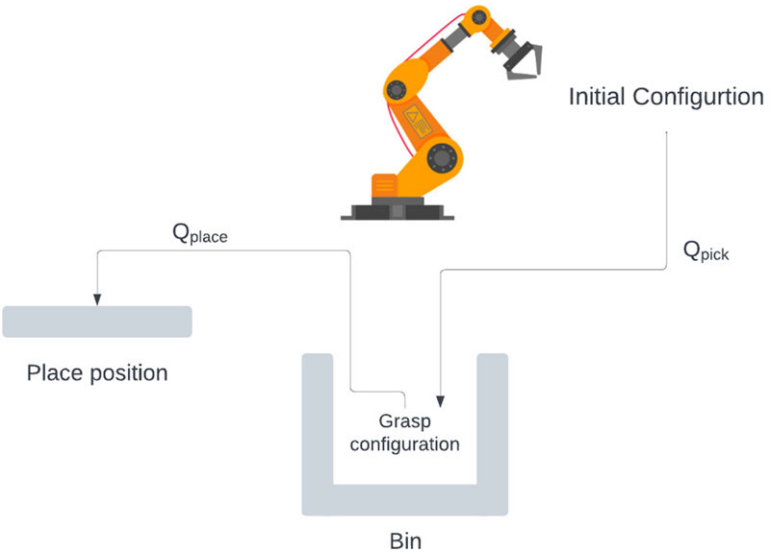

Figure 1. *Path planning for pick-and-place operation.*

$$q = \left[\theta_1 \ \ldots \ \theta_n\right]^{\mathrm{T}} \tag{1}$$

where θ_i is the angular position of the i-th joint of the arm. C_{free} can be defined as the subspace of C_{space} in which the robot is collision-free. Let τ be a trajectory and the first and last element of it be $q_{initial}$ and q_{target}, respectively. The path-planning problem is to find all of the elements between $q_{initial}$ and q_{target} in a way that all of the segments in the trajectory lie entirely in the C_{free} space. In the context of path planning, a segment between two configurations is the space of the configurations inside the linear combination of the two configurations. In the other words:

$$\left\{\alpha q_i + (1-\alpha) q_{i+1} \mid \alpha \in [0,1]\right\} \subseteq C_{free}, \forall i \in \{1, \ldots, N-1\} \tag{2}$$

where N is the number of waypoints.

Bin-picking includes two different path-planning problems. The first one is finding the path between the initial configuration of the robot and the grasp configuration which is located inside the bin (Q_{pick}); the second one is finding the path between the grasp position and the place configuration (Q_{place}). Fig. 1 shows the pick-and-place operation overview. Now the problem is to design a planner that can satisfy the following objectives:

- generating a safe and obstacle-free path.
- answering the queries in real time.
- generating a high-quality path with a high probability of success.

4. Path planning and collision checking framework

In this section, we will delve into the details of the proposed framework for path planning in bin-picking tasks. As illustrated in Fig. 2, the proposed method is comprised of two consecutive networks, namely the planning network and the collision checking network. Our choice of a two-network architecture is underpinned by several key considerations. First and foremost, safety is of importance in robot motion, and ensuring collision-free paths is a fundamental requirement. To expedite the planning process, we

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

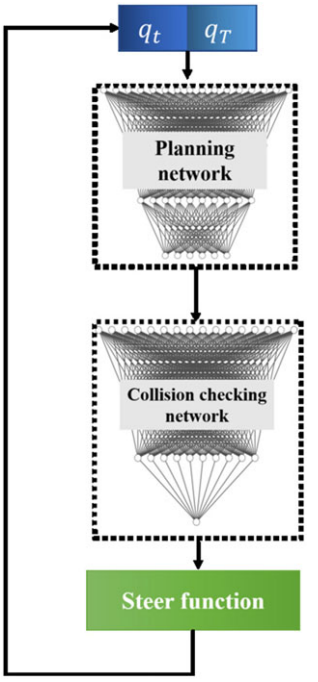

Figure 2. *Path planning and collision checking network (PPCNet).*

have incorporated a collision checking network, which offers significantly faster collision checks compared to traditional analytical methods. Secondly, the inherent nature of function approximation in our planning network introduces an element of inaccuracy, leading to occasional outputs prone to be in-collisions. Given this limitation, a subsequent verification step is essential to assess waypoint feasibility regarding potential collisions. The collision checker network serves this vital role, acting as a safeguard to ensure safety. By harnessing both networks, our approach strikes a balance between efficiency and safety, guaranteeing swift and secure robot motion planning.

The first network is a modified version of MPNet [37]. The original version of the MPNet consists of two networks. The first one is the planning network which generates the waypoints in a sequential manner by imitating the behavior of an expert planner. The second one, the encoder network, encodes the environment to a fixed-length vector in order to give environment information to the planning network for better planning.

The primary obstacles of concern in this paper are the bin, other permanent structures, and the robot itself. These are considered to be fully known and encoded within the planning environment such that unlike the MPNet, our proposed planner does not use a depth map or other sensory readings of the environment to encode information for the planning network. This allows for faster planning times, which is a key focus of our research. In a full bin-picking solution, the process that determines the grasp configuration will select grasps that are at low risk of collision during grasping and robot path.

In our work, the planning network takes the current and goal configurations and generates the next joint space configuration to move into. Each configuration is represented by an n-dimensional column vector. Therefore, the path-planning network's input is a $2n$-dimensional vector which is obtained by the concatenation of the current and goal configuration. In each iteration of decision-making, after getting the next configuration from the path-planning network, the feasibility of the segment is checked using the second network. The second network, namely the collision checking network, gets the next waypoint and estimates how likely it is that the segment between the two configurations will be inside C_{free}. By following this approach, the process of decision-making (followed by collision-checking) in the configuration space can be done in a more efficient manner.

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

4.1. Training and dataset generation

The main purpose of the path-planning network is to clone the behavior of an "expert" planner. In cases like this, where it is feasible for an expert to demonstrate the desired behavior, imitation learning is a useful approach. Moreover, while RRT-based path planners are not themselves sequential decision-makers, they can be modeled as:

$$q_{t+1} = f(q_t, q_{target}) \tag{3}$$

The purpose of the neural planner is to learn the function f. Behavior cloning, which focuses on using supervised learning to learn the expert's behavior, is the most basic type of imitation learning. The process of behavioral cloning is rather straightforward. The expert's demonstrations are broken down into state-action pairs, which are then used as independent and identically distributed (i.i.d.) training data for supervised learning. An important concern when using supervised learning for behavioral cloning is the risk of encountering out-of-distribution data. During the inference process, the neural planner may make decisions that were not previously observed in the dataset. Consequently, even minor inaccuracies in the replicated behavior can accumulate rapidly, resulting in a substantial failure rate in identifying viable paths. To mitigate this issue, we propose the data aggregation algorithm (DAGGER) in our study [45].

The training process of the neural planner and the collision checker is explained in Algorithm 1. At the beginning of the process, some initial demonstrations from the expert planner are required. To do so, *RandomGoalConf*(T) function samples T goal configurations in the collision-free space of the bin. The generated goal configurations are given to the expert planner to generate the paths using *GeneratePath* function which requires initial and goal configurations. The expert planner outputs two datasets: the neural planner dataset (D) and the collision checker network dataset (C). D only consists of the final path that the expert planner generates; however, C includes every segment which was checked by the expert planner using the geometrical collision checker. In the case of an RRT-based algorithm, the final path will be stored in D and the whole RRT tree and all the instances of its corresponding steer function are stored in C.

To train the neural planner, the demonstration dataset generated by the expert planner should be of high quality. To generate a high-quality dataset, post-processing is applied to the paths generated by the expert planner. As shown in Fig. 3, the expert planner generates a collision-free path for a random query. Followed by that, the binary state contraction (BSC) is utilized to remove redundant and unnecessary waypoints, resulting in a shorter overall path. The sparsity of waypoints is desirable as it simplifies the subsequent processing, such as collision checking, communication with the robot controller, and trajectory planning. While the output of BSC is similar to the lazy state contraction presented in ref. [36], BSC is computationally more efficient and reduces the run time. After BSC, the resampling function is used to ensure the smoothness and efficiency of the planned path. This involves discretizing the waypoints at regular intervals, which are guaranteed to be collision-free as they already exist on the collision-free path. Importantly, these additional waypoints do not increase the overall length of the path but do serve to reduce the mean and variance of the target step magnitudes, $||q_{t+1} - q_t||$. While larger steps require fewer neural network forward passes and collision checks during the inference phase, smaller steps offer several advantages. Firstly, they minimize the deviation from the planned path by reducing the magnitude of individual errors $||\hat{q}_{t+1} - q_{t+1}||$. Secondly, they are more likely to be collision-free as they occupy less physical space. Finally, they offer a more normalized training target for the neural network, where the problem is reduced to predicting a direction in the limit. However, BSC is a crucial step before resampling, as it relies on dividing large steps into regular steps, and a dense path of short steps cannot be divided. Fig. 4 illustrates this procedure in a 2D environment, highlighting how this method enhances the smoothness and quality of the path. The *PostProcess* procedure in Algorithm 1 presents the use of BSC and resampling on the generated paths.

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

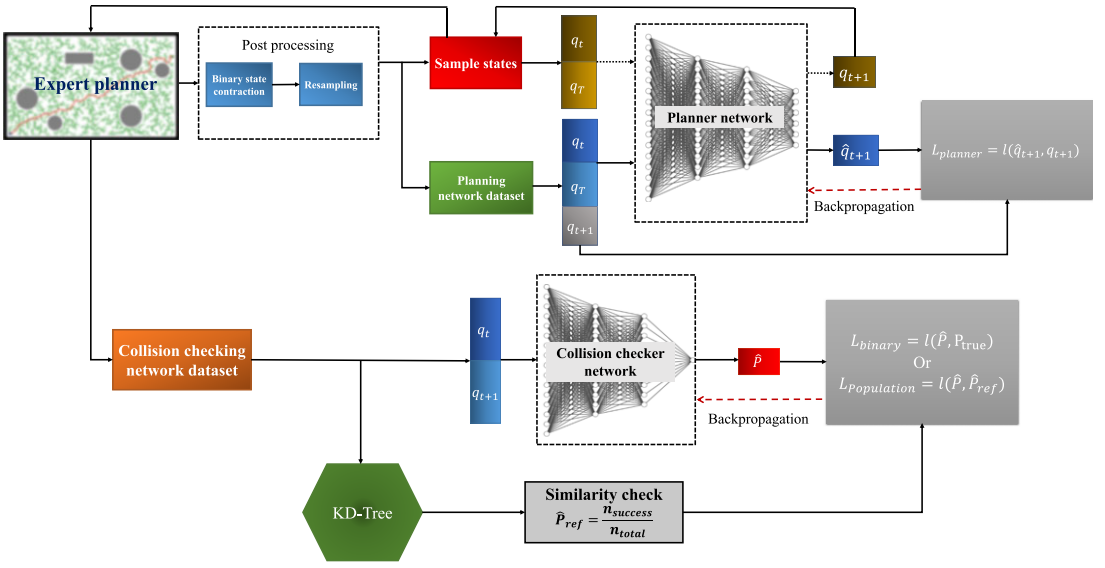

Figure 3. *End-to-end training process of PPCNet:* ***top row:*** *the imitation learning and data aggregation processes for training the planner network. In this process, based on random queries and the data aggregation process, the planning network dataset is constructed. The dataset outputs the tuple* (q_t, q_T, q_{t+1}) *indicating the current, goal, and next configurations. Finally, the loss function* $L_{planner}$ *is used to backpropagate through the network's weights.* ***bottom row:*** *population-based probability estimation and collision checker network training processes. In this process, for calculating the population-based probability, KD-tree is employed. Furthermore, using the collision checking network dataset and the respective loss function (*L_{binary} *or* $L_{Population}$*), the network is updated.*

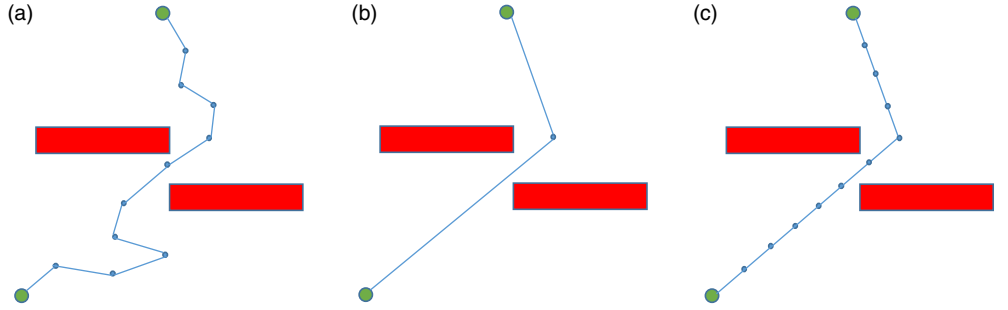

Figure 4. *Post-processing procedure: (a) The generated path by the planner, (b) Path after binary state contraction, (c) Path after resampling.*

4.1.1. Path planner training formulation

Once the dataset has been generated, it is time to train the network. The goal of the network is to predict the next configuration, $\hat{q}_{t+1}$, which should be as close as possible to the actual next configuration, q_{t+1}, in the training dataset. The network uses the dataset to learn the underlying patterns and relationships between the input and output data.

To achieve the goal of accurate prediction, the network must minimize the difference, or loss, between the predicted and actual configurations. This is done by comparing the predicted output $\hat{q}_{t+1}$ to the actual output q_{t+1} and adjusting the network's parameters to minimize the difference. This process is known

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

as backpropagation and it is typically done using optimization algorithms such as stochastic gradient descent.

$$L_{planner} = \frac{1}{N_t} \sum_{j=1}^{N_d} \sum_{i=1}^{T} ||\hat{q}_{j,i} - q_{j,i}||^2 \tag{4}$$

In this equation, N_t is the number of trajectories and N_d is the number of all individual waypoints in the training dataset.

4.1.2. Collision checker training formulation

To ensure the safety of the robot, we experimented with two different approaches for training the collision checker and evaluated their performance. As mentioned earlier, the collision checker network takes the current and next configurations as input and estimates the probability of the segment being in collision. Therefore, the training set for this network should include segments and corresponding labels indicating whether they are in collision or not. The optimization function utilized to train the collision checker for both of the approaches is called binary cross-entropy loss:

$$L(\hat{p}, p_{true}) = \hat{p} \log p_{true} + (1 - \hat{p}) \log (1 - p_{true}) \tag{5}$$

where $\hat{p}$ is the probability estimation made by the neural network and p_{true} is the true probability. The reason for using binary cross-entropy is due to the fact that, in essence, the collision checker is a binary classification network. Therefore, the standard approach in the literature is to use cross-entropy due to it giving a better probability distribution over the training data.

Binary labels. The most basic approach to tackle the optimization of the network is to use the true labels given by the analytical collision checker. In that case, p_{true} in Eq. (5) becomes either 0 (collision-free segment) or 1 (in-collision segment). In Fig. 3, this approach has been named as L_{binary}. The binary cross-entropy loss function measures the dissimilarity between the network's predicted probability distribution and the true probability distribution in the binary classification problem.

Population-based labels. The sparsity of the binary labels hinders the performance of the collision checker since it skews the output of the network towards the more common label. The motivation for this method is to make the binary labels continuous. In addition to the mentioned reasoning, making the labels continuous can help the network be better optimized in corner cases. For example, a collision-free segment inside the bin is mostly surrounded by in-collision segments. Hence, the collision-free segment will not be of much help in optimizing the network compared to the massive existence of the in-collision segments. Making the labels continuous will amplify the effect of the collision-free segment. To make the binary labels continuous, we give a regional estimation of the probability of a specific segment being collision-free by retrieving the segments similar to it whose centers lie within the hypersphere (with a specific radius) surrounding the center of the segment. Finally, the population-based probability (denoted as $\hat{p}_{ref}$ to indicate it being a reference probability) estimates the probability of a segment being collision-free by dividing the number of collision-free segments by the total number of segments:

$$\hat{p}_{ref} = \frac{\sum_{k=1}^{K} \mathbb{1}[p_{true}^{(k)} == 0]}{K} \tag{6}$$

where $\mathbb{1}$ is the identity-indication function. The advantage of the proposed approach is that the probability labels generated by this method can provide a well-behaving continuous function where near the obstacles, the probability gets near zero and the further it gets from them, it approaches one. In practice, we utilize KD-Tree method (built with the centers of the segments) to find similar segments. Moreover, the more data this approach gets, the better the estimation will be. Therefore, our proposed algorithm stores every segment that was checked by the analytical collision checker during the path-generation process with the expert planner. In the case of an RRT-based expert planner, this corresponds to storing the whole tree and all of the in-collision segments generated by it. While this approach may have higher computational cost compared to the binary labels approach, our results indicate that it can achieve better

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Algorithm 1. Training process of PPCNet

Require expert planner π^*, number of initial demonstrations T, number of rollouts in each iteration T', number of state samples S, required policy success rate ζ
Initialize Planner and Collision Dataset: $D, C \leftarrow \emptyset$
Generate initial demonstrations:
  $GoalConf \leftarrow RandomGoalConf(T)$
  $D, C \leftarrow GeneratePath(\pi^*, HomeConf, GoalConf)$
Apply post-processing: $PostProcess(D)$
$SuccessRate \leftarrow 0$
for $i = 1, \ldots$ **do**
  $\pi_i \leftarrow TrainPolicy(D)$ ▷ Start of DAGGER
  $GoalConf \leftarrow RandomGoalConf(T')$
  $D_{\pi_i} \leftarrow GeneratePath(\pi_i, HomeConf, GoalConf)$
  $S_i \leftarrow SampleStates(D_{\pi_i}, S)$
  $D_i, C_i \leftarrow GeneratePath(\pi^*, S_i, GoalConf)$
  $D_i \leftarrow PostProcess(D_i)$
  $D \leftarrow D \cup D_i$. ▷ End of DAGGER
  $C \leftarrow C \cup C_i$.
  $\eta_i \leftarrow TrainCollisionChecker(C)$
  $SuccessRate \leftarrow TestPolicy(\pi_i, \eta_i)$
  if $SuccessRate > \zeta$ **then**
   return π_i, η_i
  end if
end for

prediction performance. For this approach, $\hat{p}_{ref}$ is used instead of p_{true} in Eq. (5), and the corresponding loss function is called $L_{population}$.

4.1.3. End-to-end training process

In this section, an overview of the end-to-end process of training the integrated model which consists of both the planner network and the collision checker network will be discussed. The training procedure is outlined in Algorithm 1.

Initially, a set of trajectories are sampled using the expert planner as the initial demonstration, and the planner and collision dataset are filled. In each iteration, the path planner's policy is trained using the demonstrations from an expert planner. Consequently, DAGGER process (as explained in Section 4.1) is executed which results in the expansion of both of the datasets. Finally, the collision-checking dataset is trained and the model's performance is evaluated in the *TestPolicy* function. The function uses some random goal configurations and attempts to path plan between them (following the process in Algorithm 2). If the success rate of the model is satisfying, the path planner and the collision checker network weights are returned.

4.1.4. Path-planning process

In this section, the end-to-end path-planning process of the PPCNet is explained. As illustrated in Fig. 2, the two networks, the planning network and collision checking network, do the decision-making sequentially. In the context of the bin-picking application, given the start, pick, and place configurations, the proposed model picks the object from the bin and places it in the place configuration. As outlined in Algorithm 2, in each iteration of the decision-making process the path-planning network outputs the

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Algorithm 2. Path-generation process using PPCNet

Require Trained neural planner π and collision checker η, and initial and goal configurations q_{start}, q_{target}
$path \leftarrow \{q_{start}\}$
$q_{current} \leftarrow q_{start}$
for $i = 1, \ldots, s_{max}$ **do**
  if $Steer(q_{current}, q_{target}, \eta) == q_{target}$ **then**
    $path \leftarrow path \cup q_{target}$
    if $IsFeasible(path)$ **then**
      return $path$
    else ▷ Apply Patching
      $segments \leftarrow InCollision(path)$
      for (q_s, q_e) *in segments* **do**
        $path_{alt} \leftarrow \pi^*(q_s, q_e)$
        if $Failed(path_{alt})$ **then**
          return failure
        end if
        $path \leftarrow Patch(path, path_{alt})$
      end for
      return $path$
    end if
  end if
  $q_{next} \leftarrow \pi(q_{current}, q_{target})$
  if $Steer(q_{current}, q_{next}, \eta)$ **then**
    $q_{next} \leftarrow Steer(q_{current}, q_{next}, \eta)$
    $path \leftarrow path \cup q_{next}$
    $q_{current} \leftarrow q_{next}$
  end if
end for

next configuration that the robotic arm must transition into. The current configuration and the next configuration are then checked with the collision-checking network. To do so, the *Steer* function is used. As explained in Algorithm 3, the path between the two waypoints is discretized into equal segments with lengths close to the resolution step size of the expert planner. This is done in order to have the segments be close to the collision checker training dataset's distribution. Furthermore, in each segment, the collision-free probability of the segment is checked. In order to decide whether a segment is collision-free or not, a safety threshold is needed. If the probability is more than the threshold, the algorithm continues to the next segment for collision checking. If the probability is less than the threshold, then the segment is deemed to be in collision. In that case, if the in-collision segment is the first segment of the path, then the planner has failed to output a good waypoint. Otherwise, if the in-collision segment is not the first segment, the planning network was successful and the tail of the last collision-free segment will be the output of the *steer* function. Moving on, if the *steer* function outputs success, then the next waypoint toward the goal configuration is added to the path. In the case that the steer function outputs failure, the planner network attempts to generate a new waypoint. For that purpose, stochasticity is needed inside the network. Therefore, dropout layers are used during inference to have the network be able to recover from failure cases [36]. Furthermore, in each iteration, if it is possible to directly connect the current configuration to the target configuration, then the target configuration is added to the path and it will be the final path generated by the PPCNet. At this point, the feasibility of the path is checked with the geometrical collision checker using *IsFeasible* function. If the path is collision-free, it will be returned;

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Algorithm 3. Steer Function ($q_{current}, q_{next}$)

Require collision checking network η, safety threshold $\bar{\eta}$
$segments \leftarrow Discretize(q_{current}, q_{next})$
$q_{final} \leftarrow q_{current}$
for (q_i, q_e) *in segments* **do**
  if $\eta(q_i, q_e) > \bar{\eta}$ **then**
    $q_{final} \leftarrow q_e$
  else
    if $q_{final} == q_{current}$ **then**
      return failure
    else
      return success, q_{final}
    end if
  end if
end for

otherwise, the algorithm will attempt to patch the in-collision segments using the expert planner. To this end, the *InCollision* function outputs the segments in the path that are in collision. It is important to note that if some consecutive segments were in collision, all of the segments will be counted as one bigger in-collision segment and the head and tail of the bigger segment will be considered. This is done in order to have lesser expert planner calls and be more computationally efficient. Furthermore, for each segment, the expert planner attempts to generate an alternative path between the two configurations. If for all of the segments, the patching was done with success, then the proposed path planner would be successful and the path would be returned.

5. Experimental results

In this section, we will discuss the performance of our proposed algorithm, which employs PyTorch [46] to implement the planner network and collision checker network. The simulation environment we used is Pybullet planning [47, 48]. To evaluate the effectiveness of our approach compared to the state-of-the-art path-planning techniques, we compared it against Bi-RRT [19], informed RRT* [23], BIT* [22], and MPNet [37] on three different environments (Fig. 5). The first environment consists of a universal robot UR5 and a bin, the second one involves a UR5 robot and a wall as an obstacle, and the third one features a Kinova Gen3 robotic arm, which we implemented on both simulation and practical settings. The system used for training and testing has a 3.700 GHz AMD Ryzen 9 processor with 96 GB RAM and NVIDIA TITAN RTX GPU. Moreover, Table I shows the hyperparameter selection for each of the planners.

5.1. Comparative study

Table II presents a comparative analysis of the performance of PPCNet against other planning methods. Our experiments demonstrate that PPCNet exhibits remarkable performance in terms of planning time in all three environments (More than 70% improvement). Fig. 6 provides a visual illustration of the planning time comparison. The hyperparameters for the competing RRT-based algorithms were tuned such that they produce paths of comparable length to PPCNet, while ensuring that they find a path for each planning task in less than 10 s and 1000 iterations. However, the results show that BIT* and Informed RRT* methods take longer to plan and have lower success rates than PPCNet.

PPCNet has shown its ability to generate paths with comparable length and success rates compared to other algorithms, which is particularly important in evaluating the effectiveness of path-planning

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

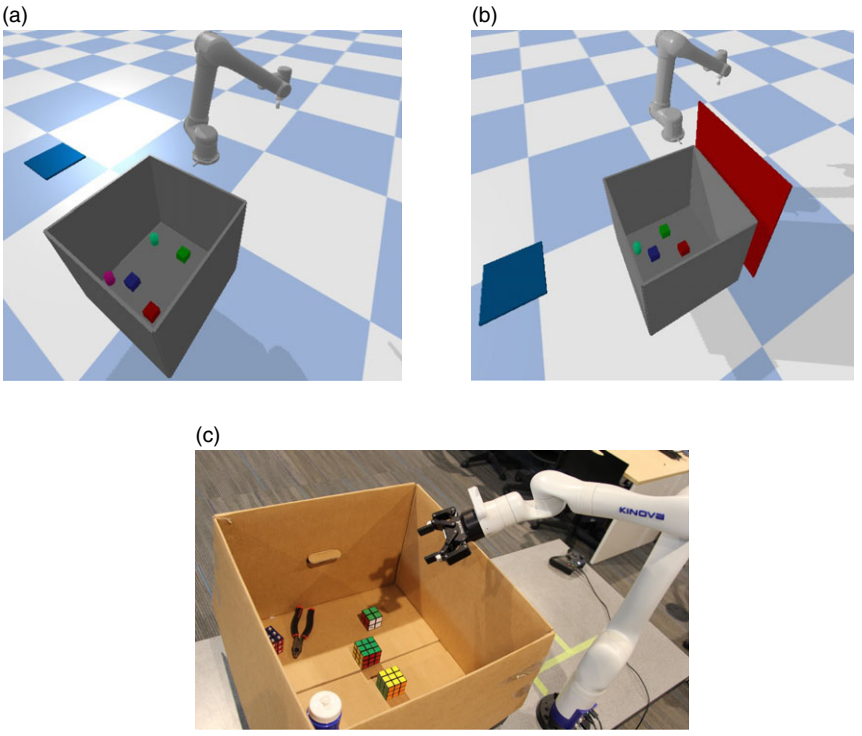

Figure 5. *Experimental environments: (a) UR5 scene, (b) UR5 scene with a wall as an obstacle, (c) Real-world implementation on Kinova Gen3.*

methods. Additionally, the post-processing method employed in dataset generation has allowed PPCNet to outperform its expert planner (Bi-RRT) in terms of path length. The BSC function plays a critical role in eliminating redundant and unnecessary waypoints, resulting in a reduction in path length and an improvement in path quality. The combination of the BSC function and resampling has significantly enhanced the quality of paths generated by PPCNet. Furthermore, our findings show that while the addition of the collision checking to the PPCNet has caused significant improvement compared to the MPNet in the expense of the marginal reduction in the PPCNet's success rate.

The success rate falling short of 100% in a known environment can be attributed to two primary factors. First, the limitations inherent to function approximation become apparent when the planner's output lacks precision, especially when navigating near obstacles. The paths generated by the expert planner occasionally skirt very close to an obstacle's edge. Replicating such precision in the planner network becomes challenging, leading to deviations from the expert's path and resulting in failures. Secondly, our specific objectives, which revolves around planning time and path length, have prompted us to establish defined thresholds. For instance, in pick-and-place tasks, we have set a limit of 300ms for overall planning time and a maximum of 100 waypoints. Any result exceeding these thresholds is classified as a failure. Adjusting these thresholds can potentially improve our success rates, although it is essential to balance them with the efficiency of our approach.

Finally, the study's results indicate that the population-based training method for the collision-checking network performed better than the binary classification approach, resulting in a more accurate collision-checking model. This is evident from the decrease in the planning time due to a reduction in the need for patching. Hence, the tradeoff between the computational cost of training the population-based collision checker against the accuracy it offers can be observed.

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Table I. *Hyperparameters selection for planners.*

Methods	Hyperparameters	Values
Bi-RRT	Max iterations	1000
	Max planning time	5 s
	Resolution	0.1 rad
Informed RRT*	Number of batches	500
	Resolution	0.05 rad
	γ	500
	Goal probability	0.1
	Max iterations	1000
	Max planning time	10 s
BIT*	Max iterations	1000
	Max planning time	10 s
	Number of samples	500
	η	20
	Goal probability	0.1
MPNet	Max iterations	100
	learning rate	0.0001
	Expert planner	Bi-RRT
	Network architecture	Fully connected 6 layers, 300 neurons per layer
PPCNet	Post-processing resample step size	0.1745 rad
	Max iterations	100
	Optimizer	Adam
	Similarity distance	0.4 rad
	Dagger iterations	30
	Expert planner	Bi-RRT
	Safety threshold ($\bar{\eta}$)	0.8
	Network architecture (planning and collision checking network)	Fully connected 6 layers, 300 neurons per layer

5.2. Real-world implementation

In this section, the motion profile generated by the Kinova Gen3 robot based on the path created by the PPCNet in the real-world bin-picking example is discussed. Due to the path-planning architecture of our approach, the main trajectory-planning process for deploying the robot in the real world relies on the specific software platform, parametric settings, and the architecture of the particular robot in use. Hence, to demonstrate the practical applicability of our method, and to underscore its potential for real-world robotic operations, we made use of the Kinova Kortex API [49] for our Kinova Gen3 robot to perform bin-picking operations. This facilitates the conversion of our established paths into actionable trajectories. For safety considerations during the implementation phase on the Kinova Gen3 robotic arm, we capped the maximum velocity for all six actuators at $36^{\circ}/s$. This constraint served as a determinant to compute the trajectory time for each distinct path segment. Figs. 7 and 8 present motion profiles from a randomly selected bin-picking scenario. As evident, the robot's motion, guided by the generated path, exhibits a combination of smoothness and safety. The Cartesian space motion profile illustrates that the generated collision-free path maintains a safe distance from the bin, ensuring its safety. Furthermore,

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

Table II. *Planning time and path length comparison of the proposed method and BI-RRT for 500 random pick-and-place queries.*

Environments	Methods	Planning time (s)	Path length (rad)	Success rate (%)
UR5	Bi-RRT	0.701 ± 0.246	7.516 ± 2.147	100
	Informed RRT*	10.186 ± 3.192	5.304 ± 1.983	79.8
	BIT*	8.621 ± 3.594	5.278 ± 2.315	77.6
	MPNet	0.384 ± 0.079	7.305 ± 2.298	94.4
	PPCNet Collision: Binary	0.101 ± 0.031	5.887 ± 1.038	90.2
	PPCNet Collision: Population	0.093 ± 0.033	5.679 ± 1.503	93.4
UR5 with a wall	Bi-RRT	0.749 ± 0.182	7.925 ± 3.307	100
	Informed RRT*	10.856 ± 3.425	5.813 ± 2.056	73.6
	BIT*	9.299 ± 3.919	5.514 ± 2.747	70.4
	MPNet	0.392 ± 0.106	7.874 ± 3.011	93
	PPCNet Collision: Binary	0.106 ± 0.027	6.142 ± 1.979	89.8
	PPCNet Collision: Population	0.099 ± 0.034	6.016 ± 1.749	92.2
Kinova Gen3	Bi-RRT	0.645 ± 0.281	6.812 ± 2.749	100
	Informed RRT*	9.535 ± 4.122	5.124 ± 2.205	79.2
	BIT*	8.496 ± 3.082	5.099 ± 2.464	79.6
	MPNet	0.227 ± 0.151	6.714 ± 2.812	90.6
	PPCNet Collision: Binary	0.090 ± 0.029	5.315 ± 1.482	88.2
	PPCNet Collision: Population	0.0857 ± 0.032	5.188 ± 1.335	89.8

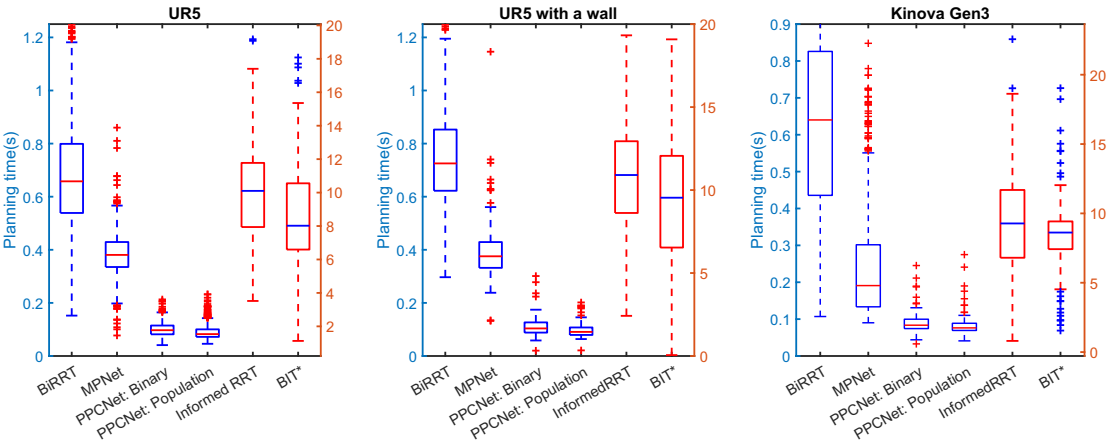

Figure 6. *Planning time comparison between different methods.*

there are no abrupt changes in the robot's joint velocity and acceleration in all of our randomly selected real experiments on Kinova Gen3 robotic arm.

While our approach did not involve explicit trajectory optimization algorithms, our results suggest that the path generated by PPCNet exhibits the potential for conversion into a smooth trajectory.[1] The

[1]The video of the proposed algorithm's sample bin-picking path can be accessed through the URL: https://youtu.be/lYxLWPi0XuA

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

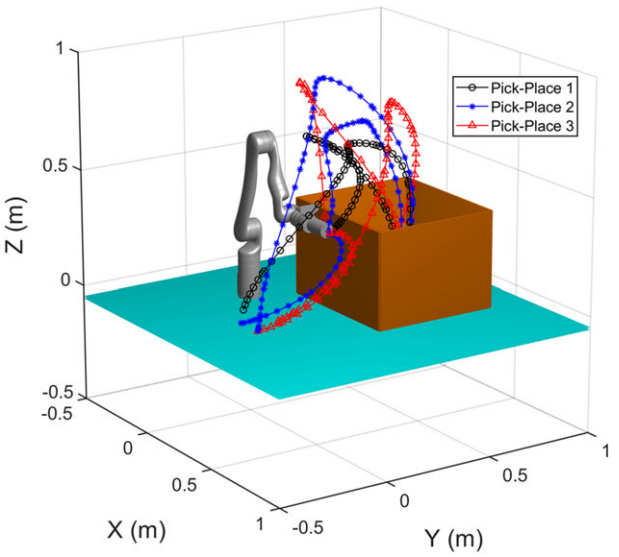

Figure 7. *Cartesian space path for a randomly selected bin-picking scenario with the Kinova Gen3 robotic arm. In this scenario, the robot should pick three different objects and place them on the table.*

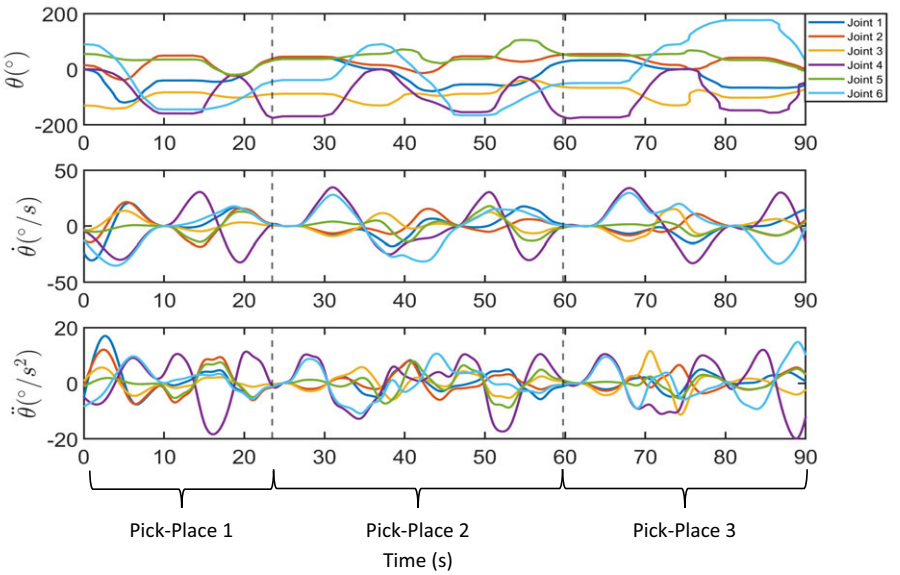

Figure 8. *Motion profile for a randomly selected bin-picking scenario with the Kinova Gen3 robotic arm.* ***top row****: joints angles plot* ***middle row****: joints velocities plot* ***bottom row****: joints accelerations plot.*

field of trajectory planning offers various methods and techniques to optimize different aspects of the trajectory, as documented in the literature [50–52]. Leveraging these established methods in conjunction with PPCNet could lead to enhanced trajectory refinement and further improvements in the overall performance.

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

6. Conclusions

In this paper, we introduced a novel path-planning framework, PPCNet, and examined its efficacy for bin-picking applications. The robot path planning for the bin-picking application is typically a computationally expensive process because the traditional algorithms need to compute the path in its entirety every time the target changes. Further, most existing algorithms lack the ability to learn from past experiences. The proposed PPCNet utilizes deep neural networks to find a real-time solution for path-planning and collision-checking problems. Specifically, we propose an end-to-end training algorithm that leverages the data aggregation algorithm to generate i.i.d. data for supervised learning. Using an expert planner in the training process, data aggregation can assist the agent in encountering unseen states. Another novel component of the proposed PPCNet algorithm includes two variations of the collision-checking network for approximating the exact geometrical collision function. We evaluate our proposed framework, PPCNet, by conducting simulation on two robotic arms namely UR5 and Kinova Gen3 in a bin-picking application. We have also examined the algorithm in a real-world scenario using Kinova Gen3 robot. Our results indicate that our approach significantly reduced planning time while producing paths with path lengths and success rates comparable to those of the state-of-the-art path-planning methods. In our research, we chose Kinova Gen3 based on the available resources to showcase PPCNet's applicability in real-world examples. However, the implementation of this algorithm on other robotic systems is feasible due to its model-free nature.

There is potential for future work in PPCNet's adaptation to dynamic environment such as scenarios involving human-robot interaction. In this context, the method can be enriched by extracting crucial data from human behavior to determine goal configurations and facilitate collision avoidance within PPCNet. Furthermore, a promising avenue for research is the incorporation of human motion data within the learning from demonstration subroutine. This is especially effective in tasks like human-robot handovers where the anticipation of human-like robot motion by human provides a higher level of adaptability. Moreover, another important future work is considering robot or environment constraints which could facilitate motion planning resulting in smoother and better motion profiles.

Supplementary material. The supplementary material for this article can be found at http://dx.doi.org/10.1017/S0263574724000109.

Author contribution. **Mehran Ghafarian Tamizi** and **Homayoun Honari**: Material preparation, data collection, analysis, and writing original draft. **Aleksey Nozdryn-Plotnicki**: writing original draft, review, and editing. **Homayoun Najjaran**: review and editing, supervision, and funding acquisition.

Financial support. This work was supported by Apera AI and Mathematics of Information Technology and Complex Systems (MITACS) under IT16412 Mitacs Accelerate.

Competing interests. The authors have no relevant financial or non-financial interests to disclose.

Ethical approval. None.

References

[1] D. Buchholz, D. Kubus, I. Weidauer, A. Scholz and F. M. Wahl, "Combining Visual and Inertial Features for Efficient Grasping and Bin-Picking," **In:** *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China (IEEE, 2014) pp. 875–882.

[2] Y. Domae, H. Okuda, Y. Taguchi, K. Sumi and T. Hirai, "Fast Graspability Evaluation on Single Depth Maps for Bin Picking with General Grippers," **In:** *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China (IEEE, 2014) pp. 1997–2004.

[3] D. Buchholz. *Bin-picking, Studies in Systems Decision and Control*, 44, (Springer, New York, 2016), 3–12.

[4] P. Boscariol, A. Gasparetto and L. Scalera, "Path Planning for Special Robotic Operations," **In:** *Robot Design: From Theory to Service Applications* (G. Carbone and M. A. Laribi, eds.) (Springer, Cham, 2022) pp. 69–95.

[5] M. G. Tamizi, M. Yaghoubi and H. Najjaran, "A review of recent trend in motion planning of industrial robots," *Int J Intel Robot Appl* **7**(2), 253–274 (2023).

[6] Z. Long, "Virtual target point-based obstacle-avoidance method for manipulator systems in a cluttered environment," *Eng Optimiz* **52**(11), 1957–1973 (2020).

[7] L. Tian and C. Collins, "An effective robot trajectory planning method using a genetic algorithm," *Mechatronics* **14**(5), 455–470 (2004).

[8] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access* **2**, 56–77 (2014).

[9] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control* (Cambridge University Press, England, 2017).

[10] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning, (1998). Technical report.

[11] L. E. Kavraki and J.-C. Latombe, Probabilistic roadmaps for robot path planning. Practical motion planning in robotics: Current approaches and future challenges, citeseer, (1998), 33–53.

[12] C. Rodríguez, A. Montaño and R. Suárez, "Planning manipulation movements of a dual-arm system considering obstacle removing," *Robot Auton Syst* **62**(12), 1816–1826 (2014).

[13] J. Rosell, R. Suárez, C. Rosales and A. Pérez, "Autonomous motion planning of a hand-arm robotic system based on captured human-like hand postures," *Auton Robot* **31**(1), 87–102 (2011).

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int J Robot Res* **30**(7), 846–894 (2011).

[15] L.-P. Ellekilde and H. G. Petersen, "Motion planning efficient trajectories for industrial bin-picking," *Int J Robot Res* **32**(9-10), 991–1004 (2013).

[16] T. Rybus and K. Seweryn, "Application of Rapidly-Exploring Random Trees (rrt) Algorithm for Trajectory Planning of Free-Floating Space Manipulator," **In:** *2015 10th International Workshop On Robot Motion and Control (RoMoCo)*, Poznan, Poland (IEEE, 2015) pp. 91–96.

[17] X. Wang, X. Luo, B. Han, Y. Chen, G. Liang and K. Zheng, "Collision-free path planning method for robots based on an improved rapidly-exploring random tree algorithm," *Appl Sci* **10**(4), 1381 (2020).

[18] T. Rybus, "Point-to-point motion planning of a free-floating space manipulator using the rapidly-exploring random trees (RRT) method," *Robotica* **38**(6), 957–982 (2020).

[19] J. J. Kuffner and S. M. LaValle, "An Efficient Approach to Single-Query Path Planning," **In:** *IEEE International Conference on Robotics and Automation*, San Francisco, USA (IEEE, 2000) pp. 473–479.

[20] M. A. Riedlinger, M. G. Tamizi, J. Tikekar and M. Redeker, "Concept for a Distributed Picking Application Utilizing Robotics and Digital Twins," **In:***2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Stuttgart, Germany (IEEE, 2022) pp. 1–4.

[21] J. Xu and J. Wang, "Effective motion planning of manipulator based on SDPS-RRTConnect," *Robotica* **40**(6), 1855–1867 (2022).

[22] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Batch Informed Trees (bit): Sampling-Based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs," **In:** *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA (IEEE, 2015) pp. 3067–3074.

[23] J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal Sampling-Based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic," **In:** *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, USA (IEEE, 2014) pp. 2997–3004.

[24] T. F. Iversen and L.-P. Ellekilde, "Benchmarking motion planning algorithms for bin-picking applications," *Indus Robot* **44**(2), 189–197 (2017).

[25] R. Cheng, K. Shankar and J. W. Burdick, "Learning an Optimal Sampling Distribution for Efficient Motion Planning," **In:** *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA (IEEE, 2020) pp. 7485–7492.

[26] A. H. Qureshi and M. C. Yip, "Deeply Informed Neural Sampling for Robot Motion Planning," **In:** *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain (IEEE, 2018) pp. 6582–6588.

[27] I. Aleo, P. Arena and L. Patané, "SARSA-Based Reinforcement Learning for Motion Planning in Serial Manipulators," **In:** *The 2010 International Joint Conference on Neural Networks (IJCNN)*, Barcelona, Spain (IEEE, 2010) pp. 1–6.

[28] M. Duguleana, F. G. Barbuceanu, A. Teirelbar and G. Mogan, "Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning," *Robot Com-INT Manuf* **28**(2), 132–146 (2012).

[29] L. Chang, L. Shan, C. Jiang and Y. Dai, "Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment," *Auton Robot* **45**(1), 51–76 (2021).

[30] Z. Li, H. Ma, Y. Ding, C. Wang and Y. Jin, "Motion Planning of Six-DOF Arm Robot Based on Improved DDPG Algorithm," **In:** *2020 39th Chinese Control Conference (CCC)*, Shenyang, China (IEEE, 2020) pp. 3954–3959.

[31] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations," **In:** *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, Australia (IEEE, 2018) pp. 6292–6299.

[32] J. Xie, Z. Shao, Y. Li, Y. Guan and J. Tan, "Deep reinforcement learning with optimized reward functions for robotic trajectory planning," *IEEE Access* **7**, 105669–105679 (2019).

[33] G. Peng, J. Yang, X. Li and M. O. Khyam, "Deep reinforcement learning with a stage incentive mechanism of dense reward for robotic trajectory planning," *IEEE Trans Sys Man Cyber Syst* **53**(6), 3566–3573 (2022).

[34] R. Rahmatizadeh, P. Abolghasemi, A. Behal and L. Bölöni, Learning real manipulation tasks from virtual demonstrations using LSTM, (2016). arXiv preprint arXiv: 1603.03833, 2016.

https://doi.org/10.1017/S0263574724000109 Published online by Cambridge University Press

[35] M. J. Bency, A. H. Qureshi and M. C. Yip, "Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation," **In:** *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China (IEEE, 2019) pp. 3965–3972.
[36] A. H. Qureshi, A. Simeonov, M. J. Bency and M. C. Yip, "Motion Planning Networks," **In:** *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada (IEEE, 2019) pp. 2118–2124.
[37] A. H. Qureshi, Y. Miao, A. Simeonov and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Trans Robot* **37**(1), 48–66 (2021).
[38] 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 Robot Auto* **4**(2), 193–203 (1988).
[39] O. S. Lawlor and L. V. Kalée, "A Voxel-Based Parallel Collision Detection Algorithm," **In:** *Proceedings of the 16th International Conference on Supercomputing*, New York, USA (Association for Computing Machinery, 2002) pp. 285–293.
[40] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Trans Graph* **15**(3), 179–210 (1996).
[41] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering* **1**(1), 046–057 (2015).
[42] J. Huh and D. 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)*, Stockholm, Sweden (IEEE, 2016) pp. 63–69.
[43] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *Int J Rob Res* **35**(12), 1477–1496 (2016).
[44] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Trans Robot* **36**(4), 1096–1114 (2020).
[45] S. Ross, G. Gordon and A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," **In:** *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011) pp. 627–635.
[46] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, Automatic differentiation in pytorch, (2017).
[47] C. R. Garrett, Pybullet Planning (2020), https://pypi.org/project/pybullet-planning/.
[48] E. Coumans and Y. Bai, Pybullet, a python module for physics simulation for games, robotics and machine learning, (2016). http://pybullet.org.
[49] Kinova Kortex API, GitHub Respository (2019), https://github.com/kinovarobotics/ros_kortex.
[50] P. Boscariol, R. Caracciolo and D. Richiedei, "Energy Optimal Design of Jerk-Continuous Trajectories for Industrial Robots," **In:** *The International Conference of IFToMM ITALY*, Cham (Springer, 2020) pp. 318–325.
[51] F. Lozer, L. Scalera, P. Boscariol and A. Gasparetto, "An Experimental Setup to Test Time-Jerk Optimal Trajectories for Robotic Manipulators," **In:** *International Conference on Robotics in Alpe-Adria Danube Region*, Cham (Springer, 2023) pp. 309–316.
[52] J. Ichnowski, M. Danielczuk, J. Xu, V. Satish and K. Goldberg, "Gomp: Grasp-Optimized Motion Planning for Bin Picking," **In:** *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France (IEEE, 2020) pp. 5270–5277.

Cite this article: M. Ghafarian Tamizi, H. Honari, A. Nozdryn-Plotnicki and H. Najjaran (2024). "End-to-end deep learning-based framework for path planning and collision checking: bin-picking application", Robotica **42**, 1094–1112. https://doi.org/10.1017/S0263574724000109