Robotica (2025), **43**, pp. 1807–1826
doi:10.1017/S0263574725000517

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Deep learning-based collision detection framework for robot tasks in clutter

Giacomo Golluccio, Daniele Di Vito, Gianluca Antonelli and Alessandro Marino

Department of Electrical and Information Engineering, University of Cassino and Southern Lazio, Cassino, Italy
Corresponding author: Giacomo Golluccio; Email: giacomo.golluccio@unicas.it

Received: 21 May 2024; **Revised:** 14 March 2025; **Accepted:** 14 March 2025; **First published online:** 15 May 2025

Keywords: robot manipulators; collision detection; deep learning; grasping; point clouds; depth images

Abstract

In this work, the problem of reliably checking collisions between robot manipulators and the surrounding environment in short time for tasks, such as replanning and object grasping in clutter, is addressed. Geometric approaches are usually applied in this context; however, they can result not suitable in highly time-constrained applications. The purpose of this paper is to present a learning-based method able to outperform geometric approaches in clutter. The proposed approach uses a neural network (NN) to detect collisions online by performing a classification task on the input represented by the depth image or point cloud containing the robot gripper projected into the application scene. Specifically, several state-of-the-art NN architectures are considered, along with some customization to tackle the problem at hand. These approaches are compared to identify the model that achieves the highest accuracy while containing the computational burden. The analysis shows the feasibility of the robot collision checker based on a deep learning approach. In fact, such approach presents a low collision detection time, of the order of milliseconds on the selected hardware, with acceptable accuracy. Furthermore, the computational burden is compared with state-of-the-art geometric techniques. The entire work is based on an industrial case study involving a KUKA Agilus industrial robot manipulator at the Technology & Innovation Center of KUKA Deutschland GmbH, Germany. Further validation is performed with the Amazon Robotic Manipulation Benchmark (ARMBench) dataset as well, in order to corroborate the reported findings.

1. Introduction

Nowadays, the application scenarios of robots are becoming increasingly dynamic, often with strict time constraints, making the motion planning task more challenging. In fact, while in highly structured environment the robot motion might be offline planned, this approach is deemed to fail when coping with unstructured and dynamic scenarios where the time factor is demanding.

In these contexts, the collision detection represents a crucial factor both for motion planning and grasping tasks. Thus, it is necessary to equip the robot with proper sensor technology to perceive the environment and detect collisions in short time in order to have the robot safely replanning its trajectory accordingly. Indeed, as per the literature, the time required to detect a collision could be a bottleneck for the motion planners, depending on the complexity of the task [1].

For example, the authors in [2] solve an objects relocation in cluttered environment problem, and the planner spends a non-negligible amount of time to find a free path, if it exists, using the classical geometrical approaches for the collision detection. Furthermore, as stated in [3] and [4], a generic motion planner could be not capable to adopt a replan strategy in short time if the environment changes during the motion of the robot.

Modern learning-based approaches have consistently demonstrated their superiority in terms of computational time over classical methods, which often heavily rely on model descriptions, whether partially or entirely [5]. In fact, data-driven algorithms based on depth images are showing promising results in

© The Author(s), 2025. Published by Cambridge University Press.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

the fields of robotic grasping and state estimation through classification. In particular, grasping objects in cluttered scenes poses a significant challenge as it requires simultaneously considering the appropriate grasp strategy while avoiding collisions with the surrounding environment. It is very usual to decide a priori a list of possible grasping configurations considering the geometric and physical models of the objects [6, 7]; for instance, the authors in [8] demonstrate that, using neural networks on RGB-D images, it is possible to find the optimal grasp for an object on the Cornell Grasping Dataset. Similarly, in [9], the considered pipeline based on multimodal grasp planner takes as input the RGB and depth images and provides in output the best grasp depending on the arrangement of the objects into the scene. In the study conducted in [10], the authors address the challenge of training a robotic grasping model using depth images and gripper force feedback and propose an algorithm that effectively reduces the amount of data required for training the model. In [11], an end-to-end network that generates a distribution of 6-DoF parallel-jaw grasps from a depth in an efficient way is addressed; they achieved 90% of accuracy on objects never seen before, using about 17 million of simulated grasp. Differently, in [12], the authors proposed a grasping method using foundation models to help the robot understand objects better, such as finding the best grasp points and avoiding certain parts.

On this topic, it is also critical to generate grasps that are robust and that require a reasonable amount of computation time. In fact, in [13], they propose the generative residual convolutional neural network (GR-ConvNet), a model able to predict grasp configurations in real time.

Lastly, the authors of [14] present an attention-based approach using two-dimensional encoding to enhance local patterns and to reallocate the distribution of grasping-specific features. Going deep, it uses RGB-D images as input and outputs image-size feature maps, determining the grasp configurations.

A literature review focusing on grasp detection technology, considering significant aspects of analytic and data-driven methods, is reported in [15].

Traditionally, image-based solutions have dominated the field, also relying on convolutional neural networks (CNNs) [16]. Recently, 3D point clouds and 3D voxel data representations are also becoming increasingly adopted. For instance, in 2017, a challenge problem in computer vision using 3D data was addressed by [17], where PointNet, a point cloud-based architecture capable of classifying and segmenting 40 classes of objects in the ModelNet40 dataset [18] was proposed. It is important to mention that this approach requires significant computational power and memory resources. In [19], an end-to-end trainable architecture for point cloud for the 3D detection called VoxelNet is proposed. It is capable to operate directly on sparse points, outperforming the state-of-the-art LiDAR-based 3D detection methods; whereas, in [20], VoxNet is introduced. The network employs volumetric representation to effectively process 3D data, enhancing robustness in object recognition.

A widely adopted geometric-based collision detection method in motion planning is the *Flexible Collision Library* (FCL) [21]. This library combines various techniques to efficiently and accurately compute collision checks. Fast and accurate collision checking methods between objects can be achieved by utilizing object mesh information in the scene [22]. In the majority of approaches, polyhedral models are commonly used, and many techniques rely on the use of *bounding volume hierarchies* (BVHs). The complexity of BVH approaches is influenced by the complexity of the polyhedral models. Alternatively, other methods based on the nearest-neighbor technique are employed; however, these methods may face challenges in high-dimensional space, making them unfeasible in certain scenarios [23]. Therefore, approximation-based modeling is not applicable in high-precision situations [24].

Computer vision researchers have considered learning-based techniques that work well with 3D object geometries. In [25], the authors demonstrate that tools for collision detection based on robotics motion planning may be accelerated by performing collision checks considering Machine Learning model. Their model, named Fastron, is capable to represent the configuration space of robot manipulators, which can be used as a proxy collision detector, by replacing the standard geometric collision approaches. Results show that, in a simulated environment, their method outperforms FCL, but it requires a number of samples strictly related to the dimensionality of the space of interest.

Differently from the above approaches, the authors in [26] propose *CollisionGP*, a Gaussian process-based algorithm for modeling a robot's configuration space and query collision checks. In [27],

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

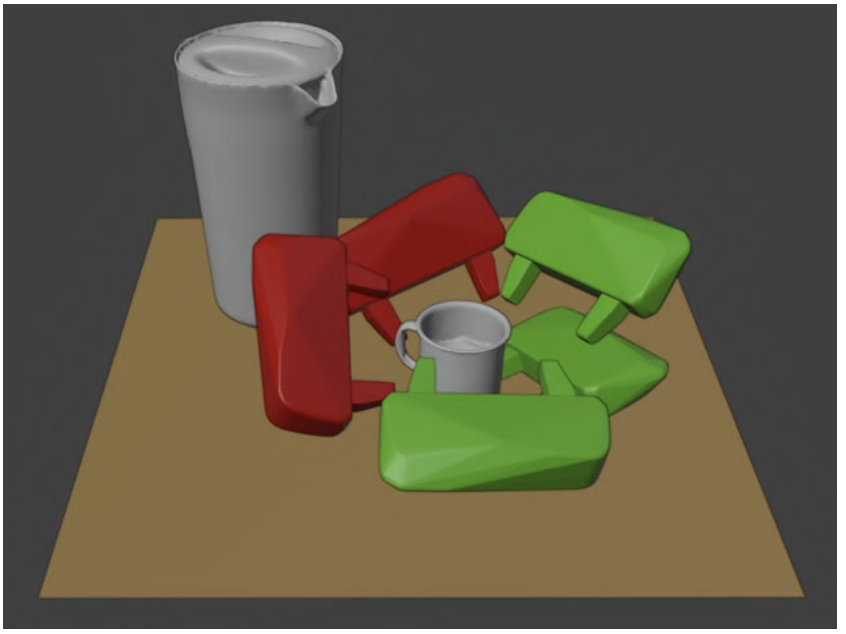


Figure 1. *3D Rendering of an example scenario requiring the robot planning for grasping a cup: the successful (green gripper mesh) and unsuccessful (red gripper mesh), i.e., colliding with surrounding objects/environment, attempts are represented, respectively.*

the authors propose Path Planning and Collision Checking Network (PPCNet), an end-to-end framework that reduce planning time significantly.

The authors in [28] allow for quickly computing distances for any point in the robot's workspace. They proposed an algorithm that combines deep learning-based approach for distance computations with geometric collision checkers.

Researchers from NVIDIA and Berkeley introduced a NN called *SceneCollisionNet*, which leverages raw point clouds of objects and scenes [29]. The network takes the pose of the object as an additional input and generates an output indicating the likelihood of object collision. This is achieved by utilizing a voxel representation within the network architecture. An accuracy of 93% is obtained considering 2 million of collision queries 10 times faster than the classic FCL library. Furthermore, in [30], the authors render synthetic partial point clouds to train CabiNet, a collision model to predict collisions for poses of SE(3) objects in the scene.

In this work, the focus is on the collision checking between the robot gripper and the surrounding environment for tasks such as replanning and object grasping in clutter. The problem is illustrated in Figure 1, where, in an example scenario involving a cup and bottle and a grasping task, the gripper mesh is projected near the cup, leading to both collision (red) and non-collision (green) configurations.

More in detail, in the case under investigation, mesh-based approaches may not be satisfying due to the time required for mesh reconstruction and collision checking. However, it is also possible to perform collision detection using neural network models working with depth images and point clouds. The idea is to assign a desired pose to the gripper and project it onto the camera-captured real scene to check collisions occurrences in real-time.

A collision checker based on deep learning is therefore proposed and an in-depth analysis of the latter is performed in terms of accuracy and computational time using different NN architectures, i.e., fully convolutional neural networks on depth images, PointNet-based methods on point clouds, and an hybrid approach resulting by merging features from both depth images and point clouds. The training of NN models is performed using two datasets of 60k samples each, containing depth images and point clouds, respectively. Additionally, a comparison is made considering the computational time between

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

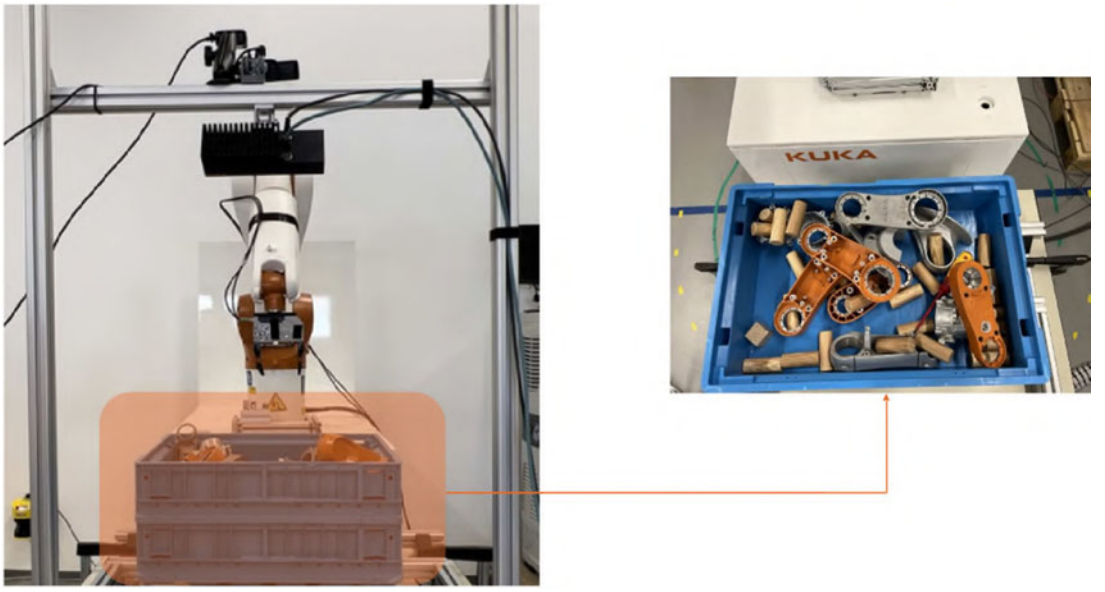


Figure 2. *The experimental setup composed by the KUKA Agilus robot manipulator with a two fingers gripper and a bin with several objects. The camera is top-mounted in order to have a complete view of the bin. Such setup is used to build the KUKA dataset.*

the best NN model resulting from the previous analysis and geometric approaches, i.e, methods based on distance and KD-Tree and OcTree-based algorithms.

The results demonstrate the superior time performance of the learning-based approach, with the best approach being a customized Point Cloud-based architecture while offering acceptable accuracy. As case study, an industrial application is considered using an experimental setup with a KUKA Agilus robot manipulator, shown in Figure 2, available at the laboratory of Technology & Innovation Center (TIC) of KUKA Deutschland GmbH in Augsburg, Germany.

The proposed methods are further validated using a different dataset, i.e., *ARMBench* [31], confirming the performance obtained with the KUKA dataset, thus proving the generality of the reported findings.

As an example to demonstrate the applicability of the approach in a practical industrial scenario, experiments involving the same setup adopted to build the KUKA dataset were performed, consisting of 10 collision checking evaluations across different objects configurations [32].

Video of the experiments is available as multimedia material attached to this paper. Alternatively, the video can be accessed at the following link: https://youtu.be/0gtcNsYazLA.

2. Proposed approach and synthetic data generation

The proposed architecture for the collision detection is shown in Figure 3.

In detail, the depth image of the scene from the camera is exploited to build the data structure, which consists of depth images or point clouds and a digital representation of the gripper.

Then, data are sent to a pre-trained neural network whose output is a binary value γ (0 in case of non-collision, 1 otherwise), which can be used inside a possible planner or grasping algorithm to evaluate the collision occurrence.

Assumption 1. *The proposed approach considers the gripper pose without taking into account the robot configuration. This is suitable for industrial bin-picking applications, as the one considered in the paper, in which the gripper approaches the objects from the top and mainly oriented along the vertical line*

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

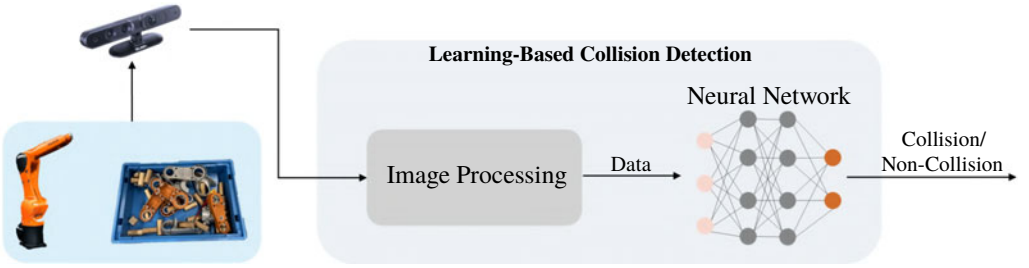


Figure 3. *Learning-based collision detection: proposed architecture.*

toward to the bin, and the remaining part of the robot structure does not collide with the environment by design.

In particular, such an assumption holds true by design for SCARA or gantry robots, for example, and for anthropomorphic manipulator as well by starting from proper high-elbow configurations (see Figure 2). In fact, SCARA robots have limited DoFs, typically 4 DoFs, and move primarily in a 2D plane with an added vertical motion for picking. Then, given their constrained movement, the arm does not collide with the environment by design. Gantry robots, operating along fixed Cartesian coordinates, are designed for predictable motion within a defined workspace. Therefore, the assumption holds well because only the gripper moves downward and the rest of the structure is clear from obstacles. For anthropomorphic manipulators, instead, as mentioned above, the proposed approach can be applied only if a high-elbow configuration is enforced, ensuring that the robot arm stays clear of obstacles.

Concerning the data, two different datasets are built: (*i*) $\mathcal{D}_{\text{depth}}$ for the case of depth images and (*ii*) $\mathcal{D}_{\text{pcl}}$ for the case of point clouds. In fact, the learning-based models are capable to extract features from both the depth map and point cloud, learning geometric information about the environment.

With regard to the gripper for which collisions with the environment are required to be checked, its desired configuration is considered

$$\boldsymbol{\eta}_{\text{g,d}} = \begin{bmatrix} \boldsymbol{p}_{\text{g,d}} \\ \boldsymbol{\phi}_{\text{g,d}} \end{bmatrix} \in \mathbb{R}^6, \tag{1}$$

where $\boldsymbol{p}_{\text{g,d}}$ and $\boldsymbol{\phi}_{\text{g,d}}$ are the desired gripper position and orientation, respectively, with the subscript "g" meaning gripper, and the subscript "d" meaning desired. As outlined in the following sections, this configuration is utilized in combination with the gripper mesh to load its digital depth image or point cloud representation into the environment's representation.

2.1. Depth images data structure

The procedure for generating $\mathcal{D}_{\text{depth}}$ is reported in Figure 4, and it works as follows: (*i*) the camera acquires the full bin in the environment and provides as output the depth image $\boldsymbol{I}_s \in \mathbb{R}^{W \times H}$ with $W = 640$ and $H = 480$; (*ii*) in parallel, starting from the mesh of the gripper denoted as M_g and available offline from the 3D-CAD model, i.e., not acquired online through camera sensor, a virtual camera (with the same intrinsic and extrinsic parameters of the real camera) generates the depth images $\boldsymbol{I}_g \in \mathbb{R}^{W \times H}$ relative to the desired configuration of the gripper $\boldsymbol{\eta}_{\text{g,d}} \in \mathbb{R}^n$ with $n = 6$. Then, the depth images $\boldsymbol{I}_s$ and $\boldsymbol{I}_g \in \mathbb{R}^{W \times H}$ are combined into a single image $\boldsymbol{I}_c \in \mathbb{R}^{W \times H}$ by the Π block in the picture as follows:

$$\boldsymbol{I}_c = \Pi(\boldsymbol{I}_s, \boldsymbol{I}_g)\,. \tag{2}$$

The function Π considers the minimum depth value for each pair of pixels between the scene $\boldsymbol{I}_s$ and the gripper $\boldsymbol{I}_g$. Thus, the objects pixels that are closer to the camera have a minor depth value and result predominant compared to the depth value for the pixels belonging to the far away objects and covered by the gripper.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

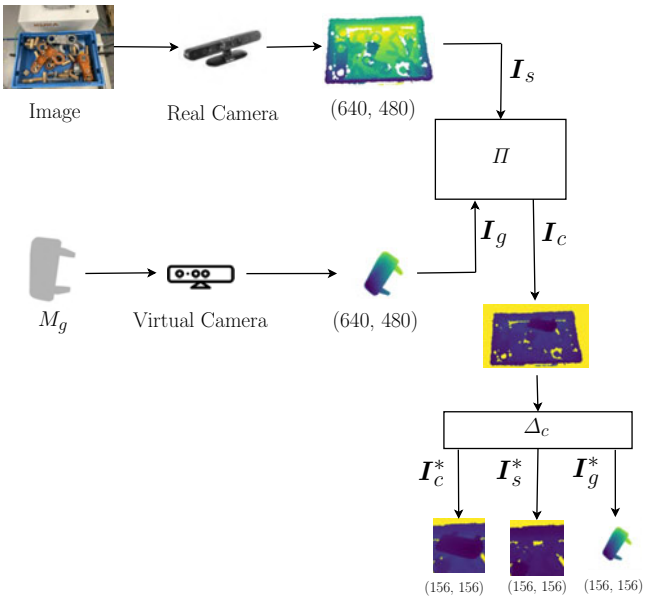


Figure 4. *Dataset $\mathcal{D}_{depth}$ generation: the camera acquires the scene and the gripper is projected into the scene; the output is represented by three depth images, i.e., gripper, scene (without gripper), and complete scene (with gripper).*

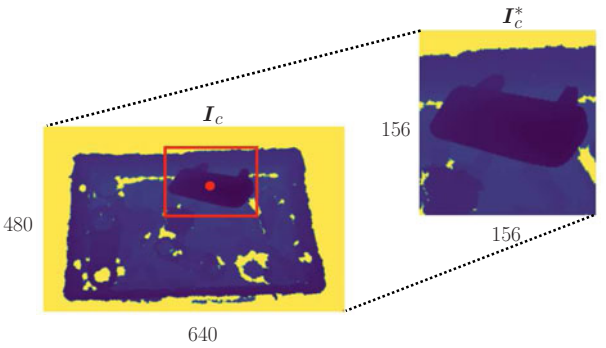


Figure 5. *Application of the crop filter on the complete depth image $\boldsymbol{I}_c$; it identifies the center of the gripper and creates a bounding box around it, reducing the images from* (640,480) *to* (156,156).

Finally, in order to reduce the computational load, a crop filter is applied by neglecting parts of the images with no gripper points, i.e., ensuring that the entire gripper is contained without loss of information about its close surrounding space. With reference to Figure 4, this operation is performed by the the block Δ_c, which generates cropped images $\boldsymbol{I}_s^*$, $\boldsymbol{I}_g^*$, and $\boldsymbol{I}_c^* \in \mathbb{R}^{W^* \times H^*}$, having, in this particular case, $W^* = 156$ and $H^* = 156$, respectively.

The effect of the crop filter is reported in Figure 5.

Summarizing, the dataset $\mathcal{D}_{\text{depth}}$ is

$$\mathcal{D}_{\text{depth}} = \{[\boldsymbol{I}_c^* \ \boldsymbol{I}_s^* \ \boldsymbol{I}_g^*]_1, [\boldsymbol{I}_c^* \ \boldsymbol{I}_s^* \ \boldsymbol{I}_g^*]_2, \ldots, [\boldsymbol{I}_c^* \ \boldsymbol{I}_s^* \ \boldsymbol{I}_g^*]_{N_s}\}, \tag{3}$$

where N_s is the number of samples.

It is worth remarking that, in this work, only collisions between the gripper and rest of the scene is considered, the remaining structure of the robot is not considered.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

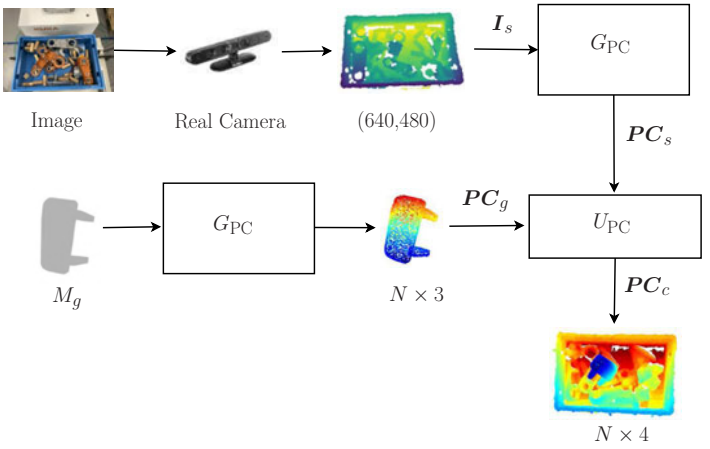


Figure 6. *Dataset $\mathcal{D}_{pcl}$ generation: the physical camera captures the scene, producing a depth image; then, the PointCloud Generator block G_{PC} generates point clouds for both the scene and the gripper; subsequently, the two point clouds are combined using the concatenation block U_{PC} to produce the complete point cloud.*

2.2. Point clouds data structure

The dataset $\mathcal{D}_{\text{pcl}}$ is generated in a different way, according to Figure 6. As well-known, a point cloud is represented by a set of 3D points $\{P_i | i = 1, \ldots, N\}$, where each $P_i \in \mathbb{R}^3$ has coordinates $[x\, y\, z]$ and makes possible to add some extra feature channels (e.g., color and other details). As in the previous case, the real camera acquires the scene of the cluttered environment (objects within a bin), providing the depth image $\boldsymbol{I}_s$. Then, the point cloud generator block G_{PC} extracts two point clouds that are $\boldsymbol{PC}_s \in \mathbb{R}^{N\times 3}$ and $\boldsymbol{PC}_g \in \mathbb{R}^{N\times 3}$, with N number of points, for the scene and the gripper, respectively; the block U_{PC} combines them to generate the complete point cloud $\boldsymbol{PC}_c \in \mathbb{R}^{N\times 4}$. Due to the different natures of the point clouds (the first one comes from the camera, whereas the second one from the mesh model), a downsampling and padding are applied. The complete point cloud has dimensions of $N \times 4$ due to the inclusion of an extra fourth column, with values set to either 0 or 1 to indicate association to the background scene or to the gripper, respectively.

More in detail, the point cloud extraction executed by the block G_{PC} is obtained through a conversion of the depth image. In fact, each pixel (u, v) in the depth image with depth value $d(u, v)$ is mapped to a 3D point (x, y, z) using the camera's intrinsic parameters in eq. (4). Given the intrinsic matrix $\boldsymbol{K}$

$$\boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4}$$

where f_x,f_y are the focal lengths in pixels, and c_x, c_y are the coordinates of the principal point, respectively, the 3D coordinates can be computed as

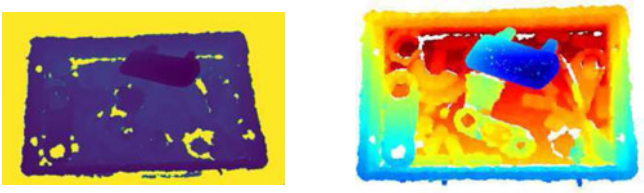
$$\begin{cases} z = d(u, v) \\ x = (u - c_x)\dfrac{z}{f_x} \\ y = (v - c_y)\dfrac{z}{f_y} \end{cases}. \tag{5}$$

The latter transformation in eq. (5). maps the 2D pixel coordinates and depth values into a 3D point obtaining, therefore, the entire point cloud. An example of the acquired-processed depth image and the relative point cloud is reported in Figure 7.

Finally, the dataset $\mathcal{D}_{\text{pcl}}$ resulting from the current point cloud dataset generation is

$$\mathcal{D}_{\text{pcl}} = \{\boldsymbol{PC}_{c_1}, \boldsymbol{PC}_{c_2}, \ldots, \boldsymbol{PC}_{c_{N_s}}\}, \tag{6}$$

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press



I_c PC_c

Figure 7. *On the left, the complete (scene and gripper) depth image $\boldsymbol{I}_c$ is reported, whereas in the right part the complete point cloud $\boldsymbol{PC}_c$ is shown.*

the labels of which are generated using Open3D [33], an open-source library that supports the rapid development of software for handling 3D data, providing a wide selection of data structures and algorithms in C+ + and Python.

2.3. Dataset complexity

Regarding the complexity of the scenarios, the idea is to take into consideration different cases. A basis made of 250 highly diverse cases (bins either empty or filled with objects in a completely random manner) is considered. By randomly sampling these scenarios, the gripper was placed in the scene with random positions and orientations, resulting in a total of 60k configurations, evenly split between 50% collision and 50% non-collision cases. This was done to avoid bias and ensure maximum generalization. In particular, the collision label is set, as detailed in Section 3.1 computing the minimum distance between the gripper and the bin (with all the objects inside) point clouds, respectively. If the minimum distance is less than an arbitrary desired threshold (1 cm in our case), then the scenario is labeled as a *collision case*, *non-collision* otherwise.

Samples from the dataset are shown in Figure 8. Cases A-B are the easiest, whereas C-D are medium in terms of complexity. The harder cases for the collision checker are E-F since they represent scenarios (objects and bin) never used in the training dataset.

3. Collision checker

In the following, the approaches compared for solving the collision checking problem are detailed. First, geometric approach is reported since it is used as baseline. Then, the learning-based methods are described remarking their customizations.

3.1. Geometric-based collision checking

As baseline for collision checking, an algorithm based on libraries Open3D[1] and Trimesh[2] is taken into consideration. The latter is a mesh-based library capable of handling triangular meshes for collision detection based on the known general purpose FCL library [21].

More in detail, in this work, the procedure consists of building an accurate mesh M_s from the point cloud of the scene $\boldsymbol{PC}_s$ through the library Open3D using the *ball pivoting* algorithm. It utilizes the continuity property of spherical balls with a given radius [34]. The mesh of the gripper M_g is then projected into the scene with a desired pose $\boldsymbol{\eta}_{\mathrm{g,d}}$ and, finally, the collision checker available from the library Trimesh is executed on the generated objects to compute the collision condition γ, i.e., collision/non-collision state, and the minimum distance between the nearest two points belonging to the scene and to the gripper, respectively. The entire flow of the algorithm is reported in Algorithm 1. It is worth mentioning that this procedure is meant to be used only as baseline, since it cannot be used in real time due to the time related to the mesh reconstruction as detailed in Section 4.

[1] http://www.open3d.org/
[2] http://trimsh.org/

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press



Figure 8. *Dataset samples of different complexity: (easy)* (a), (b) *show a bin with no items and wooden objects, respectively; (medium)* (c), (d) *contain industrial objects; (difficult)* (e), (f) *take into consideration plastic fishes and different shape of the bin not seen at the training phase.*

3.2. CNNs-based collision checking

This work considers three different architectures based on CNNs: two fully convolutional neural networks (FCNNs), one utilizing a single depth image and the other utilizing two depth images, respectively, and the well-known ResNet18 network customized for this application. These learning-based models differ from the basic versions as they are adapted to handle various data representations, i.e., depth images versus RGB images, incorporating additional layers to accommodate these differences. These first two architectures are reported in Figure 9, while the ResNet18-based architecture is shown in Figure 10.

3.2.1 FCNNs - 1 depth image (FCNN1D) and 2 depth images (FCNN2D)

Both the FCNN models consist of four blocks, each made of three convolutional layers (CONV), followed by max pooling (MP) and dropout (D) layers. The last layer is a convolutional layer with a kernel

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

Algorithm 1: Mesh-based Collision Checker.

Input:
$\boldsymbol{I}_s$ // depth scene
M_g // mesh gripper
$\boldsymbol{\eta}_{\text{g,d}}$ // desired pose
Output:
γ // collision 0/1
d_{min} // minimum objects distance

1 ApplyTransformation(M_g, $\boldsymbol{\eta}_{\text{g,d}}$); // Assigning a desired pose
2 $\boldsymbol{PC}_s \leftarrow$ PointCloudGenerator($\boldsymbol{I}_s$); // Building point cloud from depth image
3 $M_s \leftarrow$ CreateMesh($\boldsymbol{PC}_s$); // Reconstructing the mesh of the scene
4 γ, $d_{\text{min}} \leftarrow$ CollisionChecker(M_s, M_g); // Collision check, minimum distance between meshes

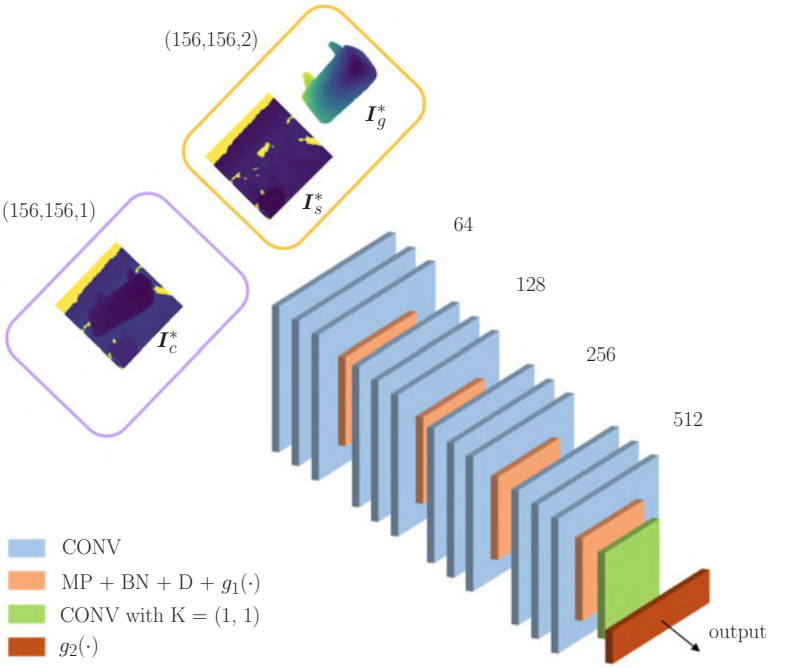


Figure 9. *Fully convolutional neural network (FCNN) architecture: FCNN1D takes one single-channel depth image* (156, 156, 1) *as input (left violet input part); FCNN2D takes two depth images* (156, 156, 2)*, i.e., multi-channel depth image (right yellow input part); both of them return collision/non-collision state as output.*

of size 1 (CONV with K = (1,1)), and it is used for compressing the output as a binary classifier. Furthermore, $g(\cdot)$ represents the activation function. In this case, there are two activation functions $g_1(\cdot)$ and $g_2(\cdot)$.

The difference between the two models is represented by the input. In particular, as shown in Figure 9, the FCNN1D takes as input $\boldsymbol{I}_c^*$, an image (156×156) with 1 channel that contains the gripper projected into the scene. Instead, the model FCNN2D considers two depth images $\boldsymbol{I}_s^*$ and $\boldsymbol{I}_g^*$, representing the scene and gripper which, combined together, result in a depth image with input size of (156,156) and 2 channels, i.e., the image is composed by two layers and, therefore, threaded by the neural network as a multi-channel depth image.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

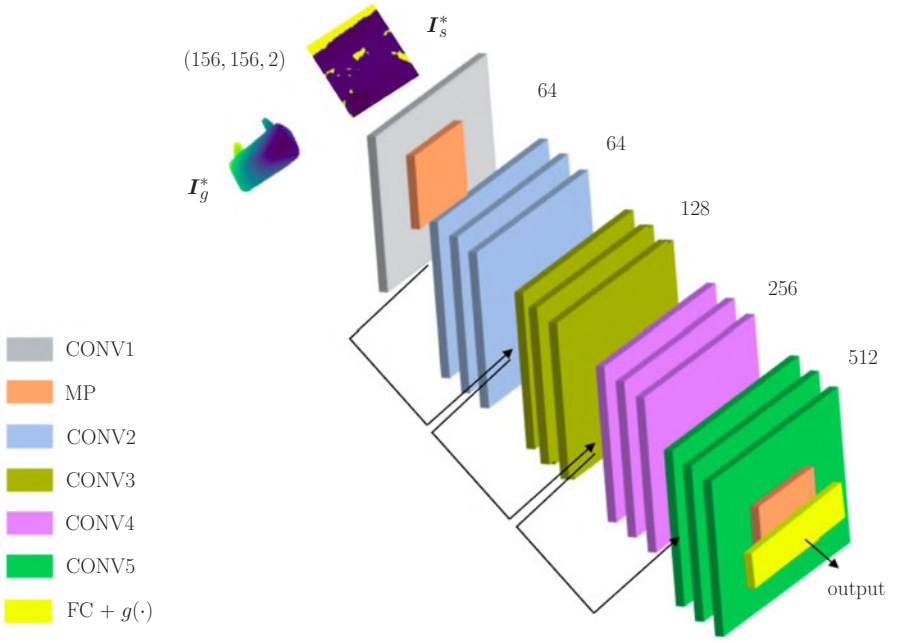


Figure 10. *ResNet18 architecture: it takes two depth images* $\boldsymbol{I}_s$, $\boldsymbol{I}_g$ $(156, 156, 2)$ *as input, i.e., a multi-channel depth image, and classifies the output in collision/non-collision.*

3.2.2 ResNet18

It is a convolutional neural network made of 18 layers (see Figure 10). In the transfer learning context, there is the possibility to use a pre-trained version of the network, which is trained on more than 1 million of images present in the ImageNet database [35]. It is capable of classifying more than 1000 different objects present in the dataset. The original version of the network considers images $(224, 224)$, but a customized version is implemented in this work. In the detail, only the deep architecture is used (not pre-trained) and adapted to work with the images of the dataset $\mathcal{D}_{\text{depth}}$. A well-known problem with very deep neural networks is the vanishing gradient when it is back-propagated to earlier layers because the repeated multiplication may make the gradient very small. Based on this, ResNet uses the concept of *residual blocks*, which incorporate shortcut skip connections in order to jump over some layers.

3.3. PointNet-based collision checking

When working with point clouds, due to the irregularity of structure, it is common to pass through a 3D voxel grids transformation [19]. Unfortunately, this procedure is really hard and expensive from the computational point of view. For this reason, this problem has been faced in literature.

3.3.1. (Standard) PointNet

The authors presented in [17] a deep learning framework, called PointNet, that receives in input unordered points. In its original version, the proposed architecture is capable of classifying and segmenting 40 different classes of objects, through reorganizations of the points received in input. In this work, the segmentation from the network is not considered, since the point cloud $\boldsymbol{PC}_c$ is built considering the information about the belonging of the points.

On the latter, initially a transformation through the T-Net block is applied (e.g., rotation about z-axis), which does not affect them in terms of shape or change of relevant features but perform an input transform and feature transform, respectively. This first T-Net aligns the input point cloud before any feature extraction takes place. Once aligned, the features are upscaled to 64 dimensions through a shared MLP,

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

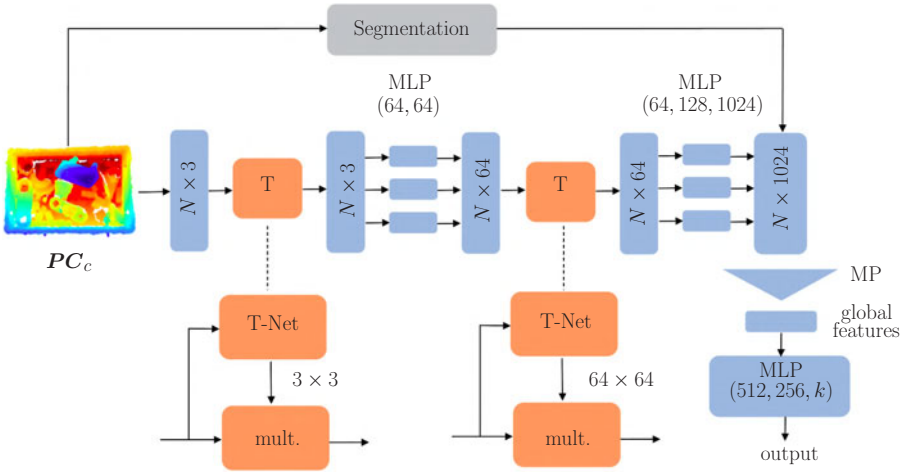


Figure 11. *PointNet and Fast-PointNet architectures with and without orange blocks, respectively: they take a point cloud* $\boldsymbol{PC}_c$ *as input and classify the output in collision/non-collision.*

marking the start of feature extraction. A second transformation with a second T-Net, i.e., feature transform, is then applied to the 64-dimensional features to derive the local features. In particular, these two T-Nets are neural networks designed to learn proper transformation matrices that perform normalization for a given input. During the forward pass, the input point cloud and then the features vector are multiplied by these learned transformation matrices, allowing PointNet to adaptively learn from various point cloud representations in an end-to-end manner. After the input and feature transform, another transformation to a different space with dimensionality 1024 is applied in the second MLP. To this regard, an important aspect is that each point has 1024 features and the network can learn the shape of the elements present in the data. The segmentation block contains additional information about the points, i.e., which mesh (gripper or bin) they belong to, and it is fused with the rest of features before the max pooling operations. With such architecture, even if the order of the input changes, the result is the same. Finally, there is a fully connected layer to map the features in one of the k categories to classify. In this particular case, as already mentioned, the classes are two, namely collision and non-collision, as shown in Figure 3. The overall architecture is reported in Figure 11, where every block is part of the current model.

3.3.2. Fast-PointNet

A custom version of PointNet, named Fast-PointNet, is proposed as well. In fact, since the camera used to acquire the data is fixed in the space, the T-Net block can be neglected. From the computational point of view, if the point cloud is composed of a large number of points, the multiplication with these matrices could take varying amounts of time. Due to this consideration, this custom version is faster than the previous architecture.

The segmentation block containing additional information about the points is considered as well. The latter block, as proven by tests, with further information about the points helps at improving the accuracy of the NN. The resulting architecture is shown in Figure 11 where, in particular, the orange blocks are neglected while the segmentation block is included in the model.

3.4. Hybrid collision checking

Finally, a combination of FCNNs and Fast-PointNet, entitled MixNet, is proposed. It is a network that fuses the depth image $\boldsymbol{I}_s^*$ of the entire scene and the point cloud of the gripper $\boldsymbol{PC}_g$. These two architectures are combined in such a way that the features extracted from the depth image of the scene $\boldsymbol{I}_s$ and the point cloud of the gripper $\boldsymbol{PC}_g$ are fused all together. The architecture is reported in Figure 12.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

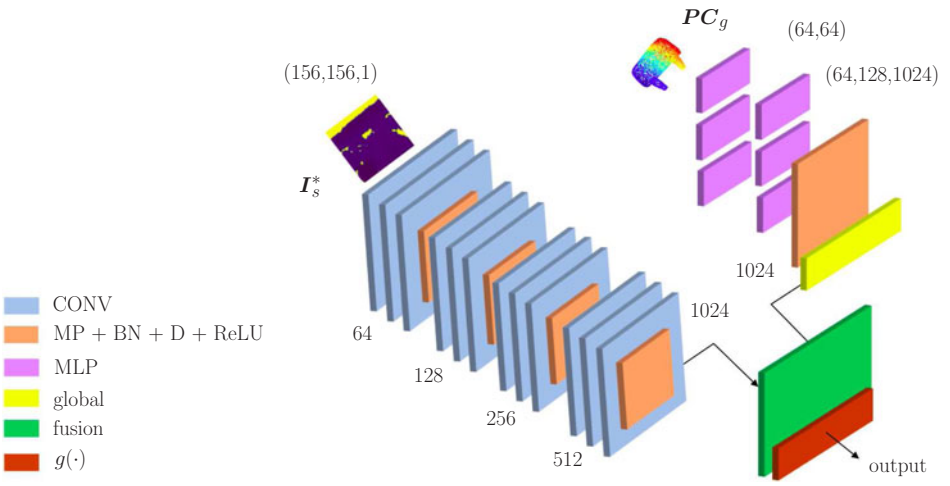


Figure 12. *MixNet Architecture: it takes a depth image* $\boldsymbol{I}_s^*$ (156, 156, 1) *and a point cloud* $\boldsymbol{PC}_g$ *as input and classifies the output in collision/non-collision.*

3.5. Learning-based collision checker

The entire flow for the learning-based collision checker, both for depth images and point cloud data structures, is reported in Algorithm 2. In particular, based on the approaches proposed above, the input of the algorithm is made of the depth image of the scene $\boldsymbol{I}_s$ provided by the camera, the mesh of the gripper M_g, the desired pose $\boldsymbol{\eta}_{g,d}$ for the gripper and the architecture to use, choosing among FCNN1D, FCNN2D, ResNet18, PointNet, Fast-PointNet, and MixNet. The output of the algorithm is, as described above, a scalar value γ that is the prediction of the neural checker, which contains the information about the collision. Then, the *ApplyTransformation* function loads the mesh of the gripper M_g with a specific pose $\boldsymbol{\eta}_{g,d}$. The *DepthGenerator* generates the synthetic depth image for the gripper $\boldsymbol{I}_g$. At the same time, the *PointCloudGenerator* builds the point clouds for the gripper $\boldsymbol{PC}_g$ and for the scene $\boldsymbol{PC}_s$; the latters are combined through the function $\Pi(\cdot, \cdot)$ as per eq. (2), which projects the gripper into the real scene, generating the complete depth image $\boldsymbol{I}_c$ (a filter to reduce the resolution is applied to the resulting point cloud). Differently, the *Union* function is applied to the point clouds, computing the final point-cloud image $\boldsymbol{PC}_c$. At this point, the architecture to use as a collision checker is selected, and the related samples are fed into the corresponding model.

4. Evaluation of the proposed models

As mentioned above, the models have been trained using a dataset consisting of 60k depth images and point clouds. It is made of balanced classes (50% collision and 50% non-collision). The chosen loss function is binary cross-entropy, and the training optimization is performed using the stochastic gradient descent (SGD) optimizer with momentum and regularization. Furthermore, the datasets $\mathcal{D}_{\text{depth}}$ and $\mathcal{D}_{\text{pcl}}$ are divided into three sub-datasets following the rule 70% for the training set, 15% for the validation set and 15% for the test set. After training the proposed models, the latter are used for collision prediction in scenes containing previously unseen objects, in order to evaluate the generalization capability of the network. In particular, all the learning-based models have been trained e validated on a laptop equipped with CPU 11th Gen Intel Core™ i7-11850H 2.50 GHz × 16 and GPU NVIDIA T1200.

Obtained results are reported in Table I and show that the methods based on the point clouds outperform the others in terms of precision, F1-score and recall, and that Fast-PointNet represents the best choice for most of the main parameters. Regarding computational cost, Fast-PointNet holds a significant advantage over PointNet on both GPU and CPU. It also yields comparable results to other methods, especially in the more compelling scenario of GPU usage.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

Algorithm 2: Learning-based Collision Checker.

Input:
$\boldsymbol{I}_s$: // depth scene
M_g: // mesh gripper
$\boldsymbol{\eta}_{\text{ee,d}}$: // desired pose
Type: // architecture: FCNN1D, FCNN2D, ResNet18, Fast-PointNet, PointNet, MixNet

Output:
γ: // Collision 0/1

ApplyTransformation($M_g, \boldsymbol{\eta}_{\text{ee,d}}$); // Assigning desired pose to the gripper
$\boldsymbol{I}_g \leftarrow$ DepthGenerator(M_g); // Building gripper depth image from mesh
$\boldsymbol{PC}_g \leftarrow$ PointCloudGenerator(M_g); // Building gripper point cloud from mesh
$\boldsymbol{PC}_s \leftarrow$ PointCloudGenerator($\boldsymbol{I}_s$); // Building scene point cloud
$\boldsymbol{I}_c \leftarrow \Pi(\boldsymbol{I}_s, \boldsymbol{I}_g)$; // Combining the depth images (FCNN1D, FCNN2D, ResNet18 or MixNet)
$\boldsymbol{I}_c^*, \boldsymbol{I}_s^*, \boldsymbol{I}_g^* \leftarrow$ Filter($\boldsymbol{I}_c$); // Applying a crop filter on depth image
$\boldsymbol{PC}_c \leftarrow$ Union($\boldsymbol{PC}_s$,$\boldsymbol{PC}_g$); // Combining point clouds (PointNet, Fast-PointNet or MixNet)
data $\leftarrow \boldsymbol{I}_c^*, \boldsymbol{I}_s^*, \boldsymbol{I}_g^*, \boldsymbol{PC}_c, \boldsymbol{PC}_s, \boldsymbol{PC}_g$; // Processing input data for Neural Networks
$\gamma \leftarrow$ Net(data) // Query for the collision check

Table I. *KUKA dataset evaluation results: applied model with corresponding accuracy, precision, recall, F1-Score, CPU time, and GPU time ($\mu \pm \sigma$) are reported, respectively.*

Dataset	Model	Acc.[%]	Pre.	Rec.	F1-Sc.	T_{CPU} [ms]	T_{GPU} [ms]
KUKA	**Fast-PointNet**	**93.19**	0.94	**0.91**	**0.93**	143.80 ± 5.44	0.38 ± 0.04
	PointNet	91.36	0.92	0.90	0.91	221.00 ± 5.49	4.25 ± 0.11
	FCNN1D	90.98	0.92	0.88	0.90	5.75 ± 0.45	0.56 ± 0.10
	FCNN2D	90.94	**0.99**	0.82	0.90	4.68 ± 0.90	0.56 ± 0.13
	ResNet18	90.69	0.94	0.86	0.90	33.07 ± 2.10	1.55 ± 0.18
	MixNet	92.63	0.96	0.88	0.92	555.30 ± 68.53	1.40 ± 0.29

In general, the resulting performance is remarkable if compared to geometric approaches. For instance, the geometric-based (FCL-based) checker, used as baseline for the dataset generation, takes ≈1.90s for reconstructing the mesh from the related high resolution point cloud, making this approach not suitable for fast real-time applications.

4.1. Comparison of Fast-PointNet with geometric point cloud-based approaches

Since Fast-PointNet proves to be overall the most performing approach, a comparison with other classic geometric methods based on point clouds is addressed. The considered algorithms are:

- **NP_{dist}—Norm-based distance computation using NumPy**[3] [36]: it uses linear algebra functions to provide efficient low level implementations of standard algorithms for distance computation. It computes the distances among all the points for the two point clouds $\boldsymbol{PC}_s$ and $\boldsymbol{PC}_g$. Then, according to the camera resolution and kinematics characteristics of the gripper, a threshold-based heuristic is considered as decision-making for collision checking;
- **$O3D_{dist}$—Norm-based distance computation using Open3D**[4] [33]: for each point in the source point cloud $\boldsymbol{PC}_s$, it computes the distance to the target point cloud $\boldsymbol{PC}_g$, exploiting CUDA. Afterward, for detecting collision, the same approach used in NP_{dist} is applied;

[3] https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html
[4] https://www.open3d.org

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

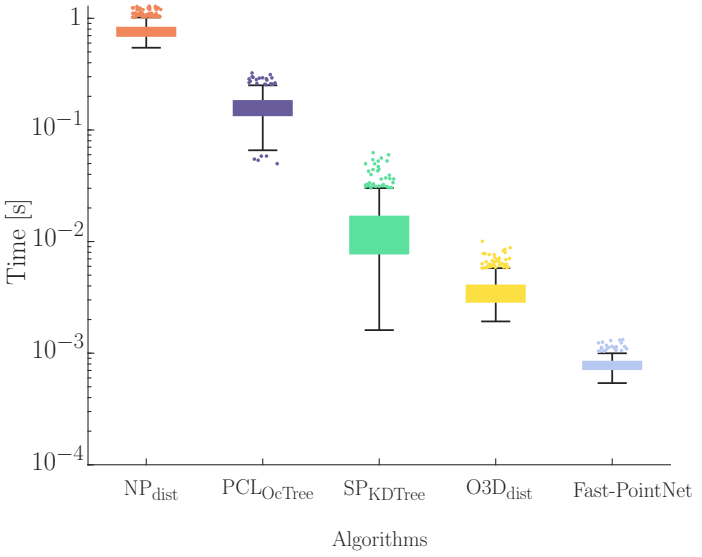


Figure 13. *Box charts relative to time complexity in log scale of: NP_{dist} (NumPy), $O3D_{dist}$ (Open3D), SP_{KDTree} (SciPy), PCL_{OcTree} (Point Cloud Library), and our proposed method Fast-PointNet.*

- **SP_{KDTree}—KDTree using SciPy**[5] [37]: the construction of a KD Tree is made of recursively, partitioning the points in the space, forming a binary tree. Each internal node in the tree functions has a hyperplane, splitting the space into two regions. Then, it considers a query to computes the distances of all for nearest neighbors provided by point cloud target using a multi-threading approach;
- **PCL_{OcTree}—OcTree using Point Cloud Library (PCL)**[6] [38]: it is a tree data structure where each internal node has eight children. This structure is mainly used for spatial partitioning of 3D point clouds due to their useful description of 3D space, which facilitates the quick finding of nearby points.

The time analysis results are reported in Figure 13. More in detail, the chart illustrates statistics on the time complexity (in s) required to correctly detect collisions between the robot gripper and its environment using point cloud representation. In particular, each element group represents the distribution of the times for the i-th method, highlighting its median, maximum, and minimum values, first and third quartile, and potential outliers. Specifically, approaches based on well-known algorithms like OcTree and KDTree, as well as methods based on geometric distances are compared along with with our best model based on point cloud, i.e., Fast-PointNet, remarking the time performance of the latter. In particular, due to the time gap between the Fast-PointNet and the other methods, it has been necessary to use a logaritmic scale on the chart to better display the results.

4.2. Evaluation on the Amazon Robotic Manipulation Benchmark (ARMBench) dataset

With the aim of corroborating the findings obtained with the KUKA dataset, results obtained with the *ARMBench* dataset are presented in this section. It is a large-scale dataset collected in an Amazon warehouse from containers with heterogeneous contents. In particular, it contains depth images with several annotations as, e.g., objects types and positions/orientation. Such depth images are adapted to the proposed framework, allowing for their integration in the same way as the data presented in Section 2.3. In detail, as for the KUKA dataset, a set of 60k scenarios has been generated by randomly projecting

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html
[6] https://github.com/strawlab/python-pcl/tree/master

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

Table II. *ARMBench dataset evaluation results: applied model with corresponding accuracy, precision, recall, F1-Score, CPU time, and GPU time ($\mu \pm \sigma$) are reported, respectively.*

Dataset	Model	Acc.[%]	Pre.	Rec.	F1-Sc.	T_{CPU} [ms]	T_{GPU} [ms]
AMAZON	**Fast-PointNet**	**91.23**	0.93	**0.89**	**0.91**	154.18 ± 7.95	0.52 ± 0.10
	PointNet	90.90	0.93	0.88	0.90	355.60 ± 15.55	9.81 ± 0.20
	FCNN1D	85.40	0.91	0.79	0.84	4.47 ± 1.18	0.57 ± 0.16
	FCNN2D	87.22	**0.95**	0.78	0.86	16.22 ± 1.31	0.54 ± 0.12
	ResNet18	85.95	0.90	0.81	0.85	14.35 ± 0.82	1.85 ± 0.15
	MixNet	88.60	**0.95**	0.81	0.87	34.17 ± 5.86	1.29 ± 0.25

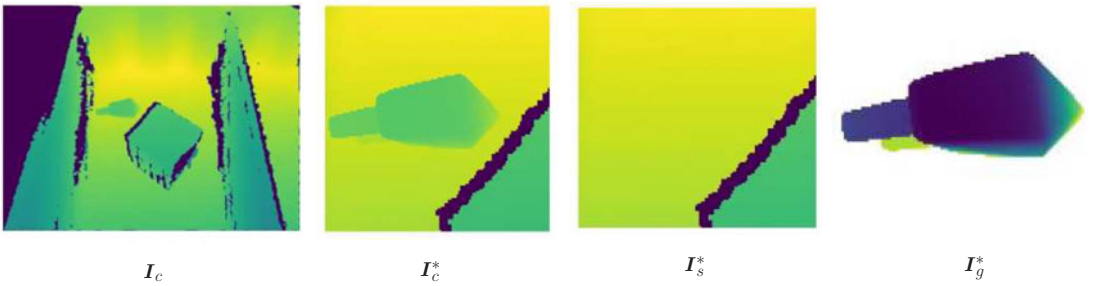


Figure 14. *Depth images based on Amazon ARMbench dataset: $\mathbf{I}_c$ is the complete depth image, i.e., scene and gripper; $\mathbf{I}_c^*$ is the complete depth image cropped around the gripper with a resolution of 156×156 px, $\mathbf{I}_s^*$ is like the previous one without the gripper; $\mathbf{I}_g^*$ is related to the gripper alone cropped with a resolution of 156×156 px.*

the gripper mesh into Amazon depth images, in such a way to have, as for the KUKA dataset, 50% of collision and 50% of non-collision cases. Examples of depth images generated using the ARMBench dataset are reported in Figure 14. The latter shows:

- I_c (top-left)—the complete depth image, namely the scene and gripper together;
- I_c^* (bottom-left)—the complete depth image cropped around the gripper with a resolution of 156×156 px;
- I_s^* (top-right)—the depth image of the scene alone cropped around the gripper with a resolution of 156×156 px;
- I_g^* (bottom-right)—the depth image of the gripper alone cropped with a resolution of 156×156 px.

The results are reported in Table II. Notably, they are very close to the performance obtained with the KUKA dataset, proving the generality and effectiveness of the reported findings and remarking the Fast-PointNet outperforming the others in terms of accuracy and time as well, even if, regarding the latter, it is comparable to FCCN1D-2D, as already observed for the KUKA dataset.

4.3. Industrial application with KUKA Agilus robot

The overall framework was finally exploited for the scenario under consideration on the real hardware. In detail, the test campaign was carried out in Python environment using a combination of PyTorch and robot operating system (ROS). The considered scenario is the same used to build the KUKA dataset and depicted in Figure 2.

The experiments consist of 10 different cases C_i ($i = 1, 2, \cdots, 10$), as reported in Figure 15, where, for each scenario, it is reported the front and top view of the setup and the corresponding depth image

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

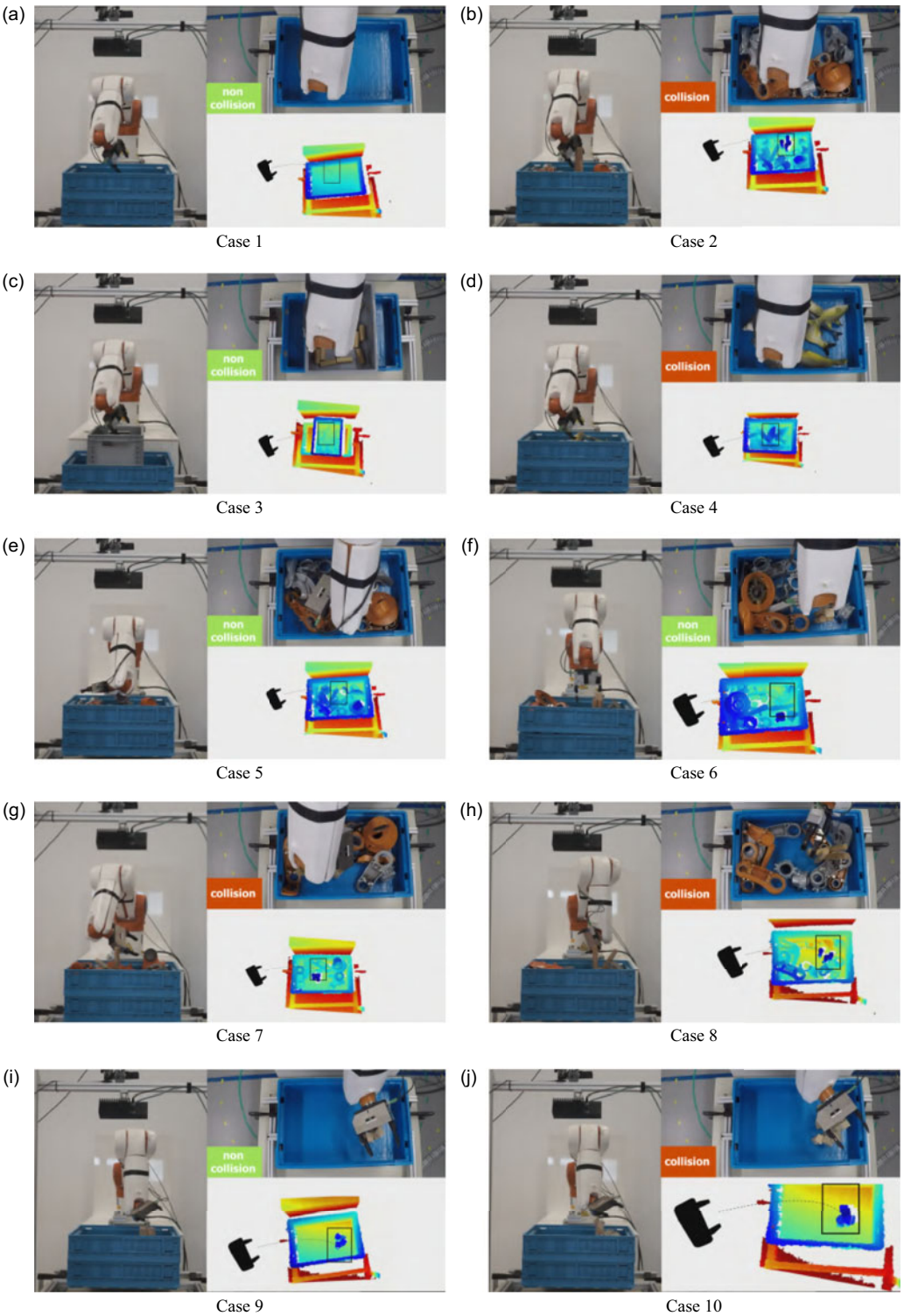


Figure 15. *Industrial application with KUKA Agilus robot in the Technology & Innovation Center at KUKA Deutschland GmbH in Augsburg (Germany). The learning-based approach is evaluated in* 10 *different real cases reporting the front, top views with the depth image showing a bounding box for the gripper target pose.*

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

of the bin alone with a bounding box showing the target pose of the gripper as projected in the depth image for collision checking. The case C_1 represents a straightforward scenario made of two sub-cases: non-collision and collision, respectively. In the first part an empty bin is considered, making it easy for the algorithms to detect the absence of collisions for the desired assigned configuration of the gripper. Then, a tower of objects is added in correspondence of the same desired position, in order to generate intentionally a collision with the gripper. In case C_2, the bin is filled with different objects, representing a more complex detection scenario compared to the previous one. The cases C_3 and C_4 involve different bins and objects in the scene, respectively, to test the generalization capacity of the models. Specifically, in the case C_3, the gray bin has a different orientation and height compared to the one used in previous cases. Furthermore, in C_4 the objects involved are plastic fish, and the collision occurs only with the initial part of the fingers, resulting in a hard collision to detect. In the previous cases, the final desired pose of the gripper is the same, while in the following remaining cases different final configurations are considered. In particular, C_5, C_6, and C_9 are such as that the non-collision occur by moving the gripper very close to the objects, while cases C_7, C_8, and C_{10} consider collisions with minor contact between the gripper and the environment.

5. Conclusion and future work

In this work, the problem of checking collisions between the robot gripper and the surrounding environment in short time for tasks, such as planning and object grasping in clutter, was considered. Starting from the mesh of the robot gripper, corresponding depth images or point-clouds were generated and combined with the depth image provided by the camera about the surrounding environment. Deep learning tools were, then, exploited to evaluate the occurrence of collisions. The validation tests showed the effectiveness of the learning-based approaches highlighting the Fast-PointNet method outperforming the other considered methods in terms of accuracy, reaching an accuracy of $\approx 93\%$ on a dataset made of 60k samples, with an evaluation time which is comparable with the methods based on depth images (order of 10^{-1} ms). A comparison was made with geometric approaches based on point clouds as well, remarking the higher performance of the Fast-PointNet over the other methods in terms of computational time. Furthermore, an additional dataset, i.e., the ARMBench by Amazon, was used to evaluate the learning based models obtaining results close to performances seen for the KUKA dataset and, therefore, showing the robustness and scalability of the proposed learning-based models.

As future work, another learning-based approach, Vision Transformers (ViT), will be considered for comparison. ViT features a different architecture from CNNs, leveraging the self-attention mechanism commonly used in large language models (LLMs). Moreover, the framework will be extended to account for other parts of the robot's structure beyond the gripper, making the proposed collision-checking approach applicable to additional tasks. However, it is worth noting that the proposed collision detection method has a wide range of applications, not only in robotics but also in other fields, such as automotive, which could be the subject of further investigation.

Acknowledgments. This work was performed at KUKA, Germany. We would like to thank Dr. Jonas Schwinn and Dr. Dominik Joho for their helpful supervision and suggestions and Dr. Kirill Safronov, Mr. Giuseppe Monetti, and Dr. João Oliveira for their advices.

Supplementary material. The supplementary material for this article can be found at https://doi.org/10.1017/S0263574725000517.

Financial support. The authors declare that this work was supported by H2020-ICT project CANOPIES (Grant Agreement N. 101016906), by Project "Ecosistema dell'innovazione—Rome Technopole" financed by EU in NextGenerationEU plan through MUR Decree n. 1051 23.06.2022—CUP H33C22000420001, by Project CONCERTO-A COgNitive arChitecture for sEamless human-Robot inTeractiOn, CUP H53C24001050006, and from Project COM^3, CUP H53D23000610006, both funded by EU in NextGenera-tionEU plan through the Italian "Bando Prin 2022 - D.D. 104 del 02-02-2022" by MUR.

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

Conflicts of interest. The authors declare no conflicts of interest exist.

Ethical approval. Not applicable.

References

[1] J. Pan and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing," *The International Journal of Robotics Research* **35**(12), 1477–1496 (2016).

[2] G. Golluccio, P. Di Lillo, D. Di Vito, A. Marino and G. Antonelli, "Objects relocation in clutter with robot manipulators via tree-based Q-learning algorithm: Analysis and experiments," *Journal of Intelligent & Robotic Systems* **106**(2), 1–20 (2022).

[3] D. Di Vito, M. Bergeron, D. Meger, G. Dudek and G. Antonelli, "Dynamic Planning of Redundant Robots Within a Set-Based Task-Priority Inverse Kinematics Framework," *2020 IEEE Conference on Control Technology and Applications (CCTA)* (IEEE, Montreal, QC, Canada, 2020) pp. 549–554.

[4] J. Kim and F. C. Park, "Active learning of the collision distance function for high-DOF multi-arm robot systems," *Robotica* **42**(12), 4055–4069 (2024).

[5] P. Schmidt, N. Vahrenkamp, M. Wächter and T. Asfour, "Grasping of Unknown Objects Using Deep Convolutional Neural Networks Based on Depth Images," *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Brisbane, QLD, Australia, 2018) pp. 6831–6838.

[6] K. Huebner, K. Welke, M. Przybylski, N. Vahrenkamp, T. Asfour, D. Kragic and R. Dillmann, "Grasping Known Objects with Humanoid Robots: A Box-Based Approach," *2009 International Conference on Advanced Robotics* (IEEE, Munich, Germany, 2009) pp. 1–6.

[7] J. Bohg, A. Morales, T. Asfour and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on Robotics* **30**(2), 289–309 (2013).

[8] I. Lenz, H. Lee and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research* **34**(4-5), 705–724 (2015).

[9] S. D'Avella, A. M. Sundaram, W. Friedl, P. Tripicchio and M. A. Roa, "Multimodal grasp planner for hybrid grippers in cluttered scenes," *IEEE Robotics and Automation Letters* **8**(4), 2030–2037 (2023).

[10] L. Berscheid, T. Rühr and T. Kröger, "Improving Data Efficiency of Self-Supervised Learning for Robotic Grasping," *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, Montreal, QC, Canada, 2019) pp. 2125–2131.

[11] M. Sundermeyer, A. Mousavian, R. Triebel and D. Fox, "Contact-Graspnet: Efficient 6-Dof Grasp Generation in Cluttered Scenes," *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Xi'an, China, 2021) pp. 13438–13444.

[12] R. Mirjalili, M. Krawez, S. Silenzi, Y. Blei and W. Burgard, "Lan-grasp: Using large language models for semantic object grasping," arxiv: 2310.05239 (2024).

[13] S. Kumra, S. Joshi and F. Sahin, "GR-Convnet v2: A real-time multi-grasp detection network for robotic grasping," *Sensors* **22**(16), 6208 (2022).

[14] L. Chen, M. Niu, J. Yang, Y. Qian, Z. Li, K. Wang, T. Yan and P. Huang, "Robotic grasp detection using structure prior attention and multiscale features," *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **54**(11), 7039–7053 (2024).

[15] M. Dong and J. Zhang, "A review of robotic grasp detection technology," *Robotica* **41**(12), 1–40 (2023).

[16] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems* **25**, 1097–1105 (2012).

[17] C. R. Qi, H. Su, K. Mo and L. J. Guibas, "Pointnet: Deep Learning on Point Sets For 3D Classification and Segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI, USA, 2017) pp. 77–85.

[18] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, "3D shapenets: A deep representation for volumetric shapes," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA, USA, 2015) pp. 1912–1920.

[19] Y. Zhou and O. Tuzel, "Voxelnet: End-To-End Learning For Point Cloud Based 3D Object Detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT, USA, 2018) pp. 4490–4499.

[20] D. Maturana and S. Scherer, "Voxnet: A 3D Convolutional Neural Network for Real-Time Object Recognition," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, Hamburg, Germany, 2015) pp. 922–928.

[21] J. Pan, S. Chitta and D. Manocha, "Fcl: A General Purpose Library for Collision and Proximity Queries," *2012 IEEE International Conference on Robotics and Automation* (IEEE, Saint Paul, MN, USA, 2012) pp. 3859–3866.

[22] 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 Journal on Robotics and Automation* **4**(2), 193–203 (1988).

[23] M. Kleinbort, O. Salzman and D. Halperin, "Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning," arxiv: 1607.04800 (2016).

[24] P. Jiménez, F. Thomas and C. Torras, "Collision Detection Algorithms For Motion Planning," In: *Robot Motion Planning and Control* (Springer, 2005) pp. 305–343.

[25] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *IEEE Transaction on Robotics* **36**(4), 1096–1114 (2020).

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press

[26] J. Muñoz, P. Lehner, L. E. Moreno, A. Albu-Schäffer and M. A. Roa, "Collisiongp: Gaussian process-based collision checking for robot motion planning," *IEEE Robotics Automation Letters* **8**(7), 4036–4043 (2023).
[27] M. G. Tamizi, H. Honari, A. Nozdryn-Plotnicki and H. Najjaran, "End-to-end deep learning-based framework for path planning and collision checking: Bin-picking application," *Robotica* **42**(4), 1094–1112 (2024).
[28] D. Joho, J. Schwinn and K. Safronov, *IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Yokohama, Japan, 2024) pp. 15402–15408.
[29] M. Danielczuk, A. Mousavian, C. Eppner and D. Fox, "Object Rearrangement Using Learned Implicit Collision Functions," *2021 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Xi'an, China, 2021) pp. 6010–6017.
[30] A. Murali, A. Mousavian, C. Eppner, A. Fishman and D. Fox, "Cabinet: Scaling Neural Collision Detection for Object Rearrangement with Procedural Scene Generation," *2023 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, London, United Kingdom, 2023) pp. 1866–1874.
[31] C. Mitash, F. Wang, S. Lu, V. Terhuja, T. Garaas, F. Polido and M. Nambi, "Armbench: An Object-Centric Benchmark Dataset for Robotic Manipulation," *2023 IEEE International Conference on Robotics and Automation (ICRA)* (London, United Kingdom, 2023) pp. 9132–9139. https://www.amazon.science/publications/armbench-an-object-centric-benchmark-dataset-for-robotic-manipulation.
[32] G. Golluccio. *Robot Planning and Control Combined with Machine Learning Techniques* (University of Cassino and Southern Lazio, Cassino, 2023).
[33] Q.-Y. Zhou, J. Park and V. Koltun, "Open3D: A modern library for 3D data processing," arXiv: 1801.09847 (2018).
[34] S. Peng, C. Jiang, Y. Liao, M. Niemeyer, M. Pollefeys and A. Geiger, "Shape as points: A differentiable poisson solver," *Advances in Neural Information Processing Systems* **34**, 13032–13044 (2021).
[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," *2009 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Miami, FL, USA, 2009) pp. 248–255.
[36] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant, "Array programming with NumPy," *Nature* **585**(7825), 357–362 (2020).
[37] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **36**(11), 2227–2240 (2014).
[38] R. L. de Queiroz, D. C. Garcia, P. A. Chou and D. A. Florencio, "Distance-based probability model for octree coding," *IEEE Signal Processing Letters* **25**(6), 739–742 (2018).

Cite this article: G. Golluccio, D. Di Vito, G. Antonelli and A. Marino (2025). "Deep learning-based collision detection framework for robot tasks in clutter", Robotica **43**, 1807–1826. https://doi.org/10.1017/S0263574725000517

https://doi.org/10.1017/S0263574725000517 Published online by Cambridge University Press