IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2023, London, UK. Cite as RA-L paper.

Sim2Real Learning of Obstacle Avoidance for Robotic Manipulators in Uncertain Environments

Tan Zhang[1,2], Kefang Zhang[1], Jiatao Lin[1], Wing-Yue Geoffrey Louie[3], Hui Huang[1]*

***Abstract*—Obstacle avoidance for robotic manipulators can be challenging when they operate in unstructured environments. This problem is probed with the sim-to-real (sim2real) deep reinforcement learning, such that a moving policy of the robotic arm is learnt in a simulator and then adapted to the real world. However, the problem of sim2real adaptation is notoriously difficult. To this end, this work proposes (1) a unified representation of obstacles and targets to capture the underlying dynamics of the environment while allowing generalization to unseen goals and (2) a flexible end-to-end model combining the unified representation with the deep reinforcement learning control module that can be trained by interacting with the environment. Such a representation is agnostic to the shape and appearance of the underlying objects, which simplifies and unifies the scene representation in both simulated and real worlds. We implement this idea with a vision-based actor-critic framework by devising a bounding box predictor module. The predictor estimates the 3D bounding boxes of obstacles and targets from the RGB-D input. The features extracted by the predictor are fed into the policy network, and all the modules are jointly trained. This makes the policy learn object-aware scene representation, which leads to a data-efficient learning of the obstacle avoidance policy. Our experiments in simulated environment and the real-world show that the end-to-end model of the unified representation achieves better sim2real adaption and scene generalization than state-of-the-art techniques.**

***Index Terms*—Planning under Uncertainty; Collision Avoidance; Reinforcement Learning; robotic manipulator, unified representation**

I. Introduction

MOTION planning is a fundamental task in robotics, which aims at navigating a robot from one location to another in an environment that contains objects without collisions. When the environment changes, the planning task becomes notoriously difficult. Mathematical models for such a motion planning task are usually characterized with high-dimensional and continuous states [1]. Robots with high degrees of freedom increase the computational complexity of such a motion planning task.

In recent years, deep learning (DL) has rapidly evolved and gained wide popularity in various domains, such as computer vision and video games [2]. Using deep learning techniques, advanced methods (e.g., [3]) can adapt knowledge from other motion planners to speed up the exploration for a given robot planning task. The recent advances in deep reinforcement learning (DRL) enabled the development of vision-based controllers for robots. Such models perform end-to-end learning, mapping the observations directly to actions. Compared with a large number of DRL applications in navigating aerial [4] and mobile vehicles [5], [6], the training of the obstacle avoiding moving policy for robotic manipulators using DRL is limited to a simulated environment, handling manipulators either in 2D space [7] or with discrete actions [8]. In real-world scenarios, such DRL training can hardly be conducted, because the trial-and-error search of a control policy through physical agent-environment interaction is then prohibitively time consuming [9], [10].

Recent works have shown promising results in sim2real learning of navigation and manipulation policies, where the DRL-based policy learning is conducted in a simulator and then transferred to the real-world environment [6], [4]. However, such sim2real adaptation is still not trivial due to the lack of high-fidelity replication of real-world scenes and the imprecise simulation models [11]. Various approaches have been attempted to bridge the sim2real gap in domain adaptation. Some works focus on domain randomization by generating randomized configurations in a simulator [12], [13]. Other works used pixel-level or feature-level domain adaptation or multi-task learning [14], still requiring a significant amount of unlabeled real-world data. Furthermore, many works tried either to make the simulation environment more realistic [5], [4], [6] or to manually create physical scenes that are similar to simulated environments [8]. However, the domain discrepancy between the simulated and real worlds is difficult to eliminate.

In order to achieve sim2real model adaption, a unified representation for both simulated and real environments is desired. The unified representation learns the shared representation and exploits their generality in different scenes. In this work, we propose the use of 3D bounding boxes to represent the obstacle and target objects in both the virtual and

This paper was supported in parts by NSFC (U2001206, U21B2023), GD Talent Plan (2019JC05X328), DEGP Innovation Team (2022KCXTD025), Shenzhen Science and Technology Program (KQTD20210811090044003, RCJC20200714114435012, JCYJ20210324120213036), and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ).

[1]Tan Zhang, Kefang Zhang, Jiatao Lin, and Hui Huang* (corresponding author) are with the Visual Computing Research Center, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518061, China. (zhangtan@sztu.edu.cn; zhangkefang45@gmail.com, zeup300 @gmail.com, huihuang@szu.edu.cn)

[2]Tan Zhang is also with the Sino-German College of Intelligent Manufacturing, Shenzhen Technology University, Shenzhen, 518118, China.

[3]Wing-Yue Geoffrey Louie is with the Department of Electrical and Computer Engineering, Oakland University, Rochester, MI 48309, USA. (louie@oakland.edu)

real worlds. Such a representation is agnostic to the geometry and appearance of the underlying objects. To realize this idea using DRL, we opt for the shared representation learning inspired by [15] that proposed the concept of successor representation for RL to improve its generalization.
We work with a vision-based actor-critic framework in which a bounding box predictor module is devised. Specifically, after receiving the environment input information, the system generates the bounding box of the targets and obstacles and feeds it to the deep reinforcement learning system, and then outputs the robot's action.

Compared with state-of-the-art neural networks and RL-based approaches to handle the manipulator obstacle avoidance problem, the contributions of our work include:

1) First, we propose unified representation and show how they can be applied to RL algorithms in an end-to-end manner. The RL states are not pre-determined: the bounding boxes are dynamic, i.e., they change over time as the detected bounding boxes change. This problem setting precludes the direct application of the trained policy in simulation to the real world by making the simulation more realistic [4] or creating physical scenes that are similar to the simulation [8]. We use unified representation of obstacles and targets in both simulated and real worlds based on 3D bounding boxes, which allows a seamless transferring of the knowledge learnt from simulation to new scenarios in the real world.
2) Second, we perform policy generation after every box generation and feed the updated robot state back to the network for a new representation learning: unlike typical sim2real approaches to policy generation [7] [8], where the network is only tasked to produce a policy, the proposed obstacle avoidance approach integrates two tasks: representation learning, which is directly impacted by how the control policy is generated, and control policy learning. It is noted that representation learning is directly impacted by how the control policy is generated.
3) At last, we conduct extensive simulated and real-world experiments to show the notable generalization ability of our method. The experiments demonstrate that the proposed soft actor-critic (SAC) [16] with unified representation outperforms other RL algorithms in terms of success rate. The experiments show that the proposed method performs better than other state representation (e.g. direct training without bounding boxes) and other training method (e.g. the bounding box is trained in advance, separately from the controller's model).

II. Related Work

Obstacle avoidance represents a fundamental problem for autonomous robotic manipulators, which operate in the real world and interact with existing objects [6]. In order to learn effective collision prediction models, the robot needs to see a number of real-world collisions during training, which comes at a high cost. A typical method is to train policies in the simulation and transfer it to the real world. However, due to the imprecise simulation models and lack of high-fidelity replication of real-world scenes, policies learned in the simulation often cannot be directly applied to real-world systems [11]. To overcome this limitation, many studies investigated the so-called sim-to-real (sim2real) [17] problem, where the core is to find accurate models of the environment to facilitate the optimization of control policies in the real world.

A. Motion Planning for Manipulators in Simulated Environments

Recent developments in DL have allowed the researchers to apply various DL architectures to robotic control and planning. An early learning-based method [18] used a sample-based motion planner to obtain the state of obstacles and manipulators. Then, a learning-based method was used to generate efficient paths to push the obstacles away and move toward the target. Other works [19], [20] used a learning-based algorithm in motion planning for a simulated environment to explore collision-free trajectories by modeling the robot arm links with cylinders. However, the success of the DL approach in manipulator motion planning tasks that require high precision is still rather limited [7].

Robot exploration when obstacles exist in the environment is inherently a sequential decision-making problem under uncertainty; thus, it is appropriate to design obstacle avoidance as a reinforcement learning (RL) problem. Obstacle avoidance using DRL techniques is so far still limited to robotic manipulators, handling either only point robots [4], [21], [6], [5] or manipulators with discrete actions [8]. These methods cannot adapt to higher dimensional manipulators with continuous actions. A previous work [8] proposed an obstacle avoidance method for redundant manipulators using a Q- learning technique. This approach trains the obstacle avoidance model in randomized simulations with only a single obstacle, and then transfers the trained model to the real-world environment that has a good replica of the virtual setup. More recently, some preliminary research has applied DRL techniques for robot manipulators, where the learned policy was used to avoid unpredictable obstacles. One study [22] combined a normalized advantage function (NAF) and Q-learning for manipulator obstacle avoidance in a structured 3D environment. Similarly, the techniques proposed in other works [23], [7] used a deep deterministic policy gradient (DDPG) framework for a 4-DoF manipulator in simulation only. Both methods [23], [7] handled only simple obstacles in the environment. Besides, the method in [7] used a 2D simulator for training, which makes the trained agent completely agnostic to the perception of depth. Hence, this approach may not reach a target object when implemented on a real robot in a 3D environment.

B. Sim2Real for Manipulators

To overcome the discrepancy between the simulated and real worlds, training has been performed on a large variety of simulated scenarios [5], [6]. On the other hand, the policies trained in simulation were fine-tuned to enable a successful

use in the real world [4], and common feature representations of real and synthetic data were obtained [24].

In the context of sim2real, model-based RL is often used to explore control policies [5], [4], [6]. A common method for sim2real approaches is an accurate 3D rendered simulator. A previous work [5] used an asynchronous advantage actor-critic(A3C) to train the control policy for visual navigation in the AI2-THOR framework, which provides an environment with high-quality 3D scenes and a physics engine. Similarly, another work extended deep RL methods to real-world robotic applications through collecting a large number of 3D synthetic indoor environments [4]. However, some limitations still exist. Firstly, despite the existence of high-quality 3D scene models, the real-world scenes are usually different from environments on which the robot was trained, and thus the generalization to new scenes is weak. Secondly, the reconstruction is time-consuming, which makes it difficult to quickly adapt the simulated model to the real world.

The concept of successor representation for RL was first proposed in [16] to improve the generalization for the learning in different domains and was combined with deep networks [25]. Recent work [24] used DRL to train a mobile robot to navigate in a simulator with rendered depth images, and then transferred to the real world with depth images. Nonetheless, the existing approaches are limited to similar scenes. Besides, the robot needs to be trained in a very similarly rendered simulated environment, which largely limits its generalization ability to new scenes where objects have new positions and geometries. Moreover, it is computationally expensive to calculate the distances between the robot and obstacles [22], [23]. Thus, it remains hard to generalize existing methods to deal with unseen complex environments in the real world.

In order to narrow the reality gap, we propose to use the 3D bounding box-based representation for obstacles and targets in both simulated and real environments. This method reduces the discrepancy between the same objects in different domains and different objects. The presented reinforcement learning-based model predicts the robot's actions using the data acquired from the robot's sensors. Its Vision Module is based on a built-in convolutional network or a pre-trained Mask R-CNN networks and outputs information of basic features, raw features, or a bounding box. The Mask R-CNN network is an object detection model, i.e. it estimates the position and the size of each object on the image, thus acquiring the 2D bounding box of each object. We then combine the 2D bounding box and the depth information to learn the 3D bounding box of each object.

To generalize the obstacle avoidance control policy in uncertain environments, we use a deep reinforcement learning framework. Specifically, we use the soft actor-critic method (SAC) [16] to model the obstacle avoidance problem. This method takes the distance between the end-effector and target as a reward, without considering the distance between the robot links and obstacles. We show that the trained policy can be extended to allow a more general, robust and rapid transfer across tasks and scenes.

III. Problem Description

Fig. 1 shows an overview of the proposed obstacle avoidance method for robotic manipulators. In summary, the Vision Module generates the 3D bounding boxes of targets and obstacles. Then, the DRL model uses these bounding boxes and the states of the robot to determine the next robot action to reach the target. Robot actions are represented by a set of joint angles that are fed into the robot low-level controller in the exploration model. We train the model in randomized simulated environments, such that it can effectively generalize to unseen scenes. We will first describe the formulation of our model and reinforcement learning algorithm, and then we present the training details.

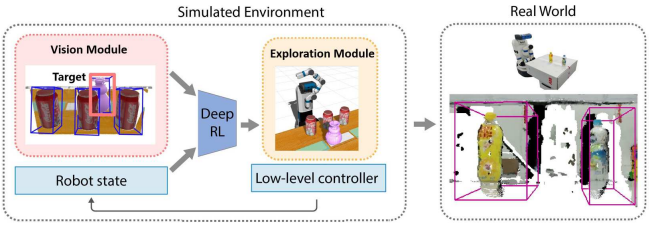


Fig. 1. An overview of our vision-based manipulator obstacle avoidance system.

A. Path Planning for Robot Manipulators

In the path planning problem for a robot manipulator, the configuration is defined as a position representation of the robot in the workspace. In other words, the configuration space $\mathbf{G}$ is a set of all possible joint angles of the robot manipulator. To be specific, $\mathbf{G}$ is defined as a subset of n-dimensional vector space, where n is the number of the joints of the robot manipulator. The set $\mathbf{G}$ consists of free space $\mathbf{G}_{\text{free}}$ and collision space $\mathbf{G}_{\text{collide}}$. $\mathbf{G}_{\text{free}}$ is comprised of all possible configurations in which the robot does not collide with any obstacles or itself, while $\mathbf{G}_{\text{collide}}$ in which the robot arm manipulator collides with obstacles or itself.

B. Collision Detection Using the 3D Bounding Box

The goal of the robot's controller is to choose the velocity settings so that the robot, starting from point $\mathbf{P}^0$, reaches destination point $\mathbf{P}^*$ in the shortest possible time. Both points are given as a four-element vector (Eq. (1)), relative to the center.

$$\mathbf{P} = \{x, y, z, \varphi\} \tag{1}$$

where x, y, z are the positions on axes X, Y, Z, and φ is the angle about the vertical axis.

To solve this problem, the robot's controller finds the best move to the target using the information from the visual sensors. The main task of the controller is to estimate the setting vector $\mathbf{Y}_{\text{vision}}$ based on the input data $\mathbf{X}_{\text{vision}}$, as can be expressed by Eq. (2).

$$\mathbf{X}_{\text{vision}} \xrightarrow{\text{model}} \mathbf{Y}_{\text{vision}} \tag{2}$$

The robotic manipulator was equipped with a depth camera. The camera generates an image tensor $\mathbf{X}_{\text{vision}}$, containing integers in the range [0, 255] and the depth value. $\mathbf{Y}_{\text{vision}}$ is the coordinates of the target object's bounding box on the image. In the environment, both targets and obstacles are detected as

boxes. Thus, the collision checking between two boxes only has fifteen cases: three faces of the first box, three faces of the second box, and nine edge combinations between the first box and second box. Based on the two neighboring boxes, the collision check between one robot and one obstacle can be implemented as a repeated two bounding box checking.

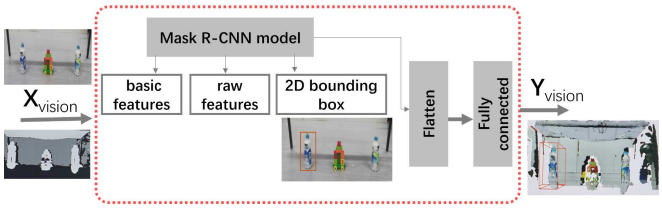


Fig. 2. The Vision Module with a vision processing method. Fully connected layers are optional, trainable layers, which process the output of preceding blocks; the module receives a vector $\mathbf{X}_{\text{vision}}$ and outputs a vector $\mathbf{Y}_{\text{vision}}$.

C. *Reinforcement Learning*

We model the manipulator exploration problem in the framework of Markov decision process (MDP). An MDP can be represented by *S, A, P, R, γ*, where *A and S* denotes the actions and states, respectively. *P* is the transition model that represents the probability of state transitions. Further, *R* represents the reward that is calculated when an action *a* is taken at state *s*. *γ* is the discount factor.

In order to acquire the optimal policy, the optimal value function (i.e., estimate of the maximum return) is approximated by value-based methods like deep Q-network (DQN) as function approximation. The other way to get the optimal policy is using the policy gradient methods to compute the optimal policy directly from the agent experience. Representative methods of the policy gradient are REINFORCE [26], actor–critic method [27], deterministic policy gradient (DPG) [28], deep DPG (DDPG) [29], etc. This paper aims at devising techniques to utilize the samples effectively for better learning, for example replay memory and HER, we present a DRL which enables the manipulator to collect and store relevant observations gathered over time.

This method is based on a soft actor-critic (SAC) architecture [16]. This approach utilizes two models; the actor (policy gradient method) is used to control the agent behavior, while the critic (action-value function method) evaluates the actor's decision. Both models are trained in parallel: the actor policy $\pi(\mathbf{s}, \mathbf{a})$ is trained to maximize the value of rewards received for actions taken by the agent and the critic function $Q(\mathbf{s}, \mathbf{a})$ is optimized to minimize reward function approximation error. Here, **s** refers to the state vector (in our case, equivalent to the input vector containing the information from the environment: visual observations X_{vision}), and **a** is the action vector (which refers to the joint actions). The reward function keeps the robot links stay away from the obstacles.

IV. Method

A. *Vision Module*

We trained our model in a robotics simulator. The general idea is that the information of all the objects (including the geometry and positions) in the physics engine is streamed to the RL framework. The RL framework, in turn, issues control commands based on the object information and sends them back to the robot in the physics engine. The use of the physics engine to model the environment is advantageous because it is highly scalable and can be executed safer than in the real world. The image from the camera is processed by the Vision Module first. This module is responsible for feature extraction from an image and dimensionality reduction. Its output is concatenated with data from depth sensors in the module that also performs further processing.

Fig. 2 shows the Vision Module. Mask R-CNN performs image analysis in a single-shot manner. As an output, it estimates the coordinates of the target object's bounding box on the image. In this module, Mask R-CNN delivers three different layers of output information: basic features, raw features, and a 2D bounding box. In addition, the images are pre-processed with bounding boxes of detect objects to modify the depth image received from the depth camera in order to improve depth estimation of objects. The Vision Module returns a vector $\mathbf{Y}_{\text{vision}}$.

We used 3D bounding boxes to represent the information of the obstacles and targets in the environment. The advantage of this representation is that it is invariant to the shape and appearance of the objects, thus unifying the scene representation of the simulated world and real world. Therefore, the model trained for certain objects can be used for unseen objects with complicated shapes. In this paper, we considered one target and multiple obstacles in one scene, and the obstacles in the tested scenarios were fewer than those in the training scenarios.

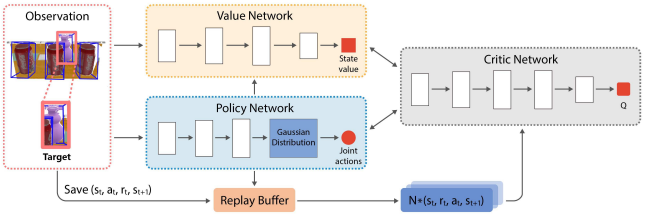


Fig. 3. Network architecture.

B. *Exploration Module*

For the exploration module, we adopted the soft actor-critic (SAC). SAC is an off-policy RL algorithm that optimizes a stochastic policy and can be used for continuous actions. It is composed of three neural networks, including a state value network to obtain the value of the next state, a critic network to obtain Q value of actions from the actor network and an actor (policy) network to generate the policy. As can be seen in Fig. 3, the input to the state value network is the current observation, including the vision information of all the objects and the robot's joint states. The model consists of several

convolutional layers, each followed by a ReLU activation function. The output is the joint actions for the next state.

C. Training

With the SAC, the model is trained by interacting with individual environments, using their copies of the global network. Agents train their individual network's weights independently, and at the end of each set of episodes, the global network is updated synchronously.

During training, the robot reaches a circular object (i.e., the target) while avoiding box-shaped obstacles. As shown in Fig. 3, the targets and the obstacles could have various sizes and positions. It is noticed that the Vision Module can be used to detect any desired targets automatically, without indicating the position components of the RL setup. The reward function is defined as follows:

$$R = w_1 R_a + w_2 R_c + w_3 R_d, \tag{3}$$

where w_1, w_2, w_3 are the weights of the action R_a, collision situation R_c and distance between the end-effector and the target destination R_d, respectively. The reward function is tuned by the weights according to their priority in the training process.

The magnitude of the actions R_a by the manipulator is computed as the square of the norm of the action vector, as follows:

$$R_a = -\|a\|^2 \tag{4}$$

such that it gives a negative contribution to the reward, thus encouraging smaller actions. If a collision is detected, then R_c is 0; otherwise, R_c is 1.

The distance R_d is calculated using the Huber loss function:

$$R_d = \begin{cases} 0.5 * d^2, & \text{for } d < \delta \\ \delta(|d| - 0.5 * \delta), & \text{otherwise} \end{cases} \tag{5}$$

where d denotes the Euclidean distance between the end-effector of the manipulator and the target, and the parameter δ determines the smoothness.

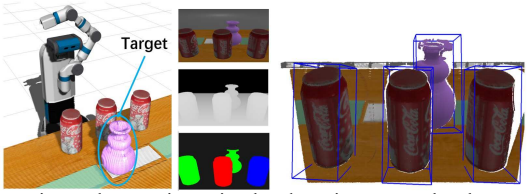


Fig. 4. Experimental setup in a simulated environment: the three middle images from top to bottom denote the scene from the perspective of our Fetch robot, the corresponding depth image, and the semantically segmented image, respectively; the right image represents the 3D bounding boxes of the observed objects.

When a manipulator completes an episode, a reward is given. This reward is independent on the information gain and moving distance at every step during the episode. The objective is to provide higher rewards to the robot that has moved closer to the target during the earlier steps of an episode. Therefore, the weights of the network are tuned to prefer the actions that maximize the total information gain of a robot in the shortest possible distance.

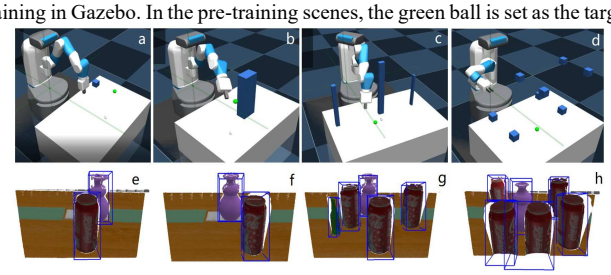


Fig. 5. Some training scenes. (a-d) are for pre-training and (e-h) are for training in Gazebo. In the pre-training scenes, the green ball is set as the target and the other boxes are obstacles. In the training scenes, the vase is set as the target and the other objects are obstacles.

V. Experiments

To investigate the feasibility of our proposed architecture for the manipulator motion planning task, we conducted three sets of experiments: 1) training and sim2real performance and scalability tests for our approach in simple and complex environments; 2) a comparison of our proposed approach with other baselines; 3) a real-world experiment with the deployment of our approach on a physical robotic manipulator reaching a target in a changing environment.

A. Training Setup

1) Environment Setup

The experiment involved several objects around the reachable workspace of a Fetch robot, as shown in Fig. 4. We obtained the 3D bounding boxes of all the objects. We trained the models on an NVIDIA GeForce GTX Titan X CPU. The values used for the hyper-parameters along with their definitions for the SAC training are summarized in Table I.

TABLE I
TUNING PARAMETERS FOR SAC TRAINING

Parameter	Definition	Value
ϕ	Policy network size	12×800×500×400×400×300×6
θ	Soft Q network size	18×800×500×400×400×300×1
ψ	Soft value network size	12×800×500×400×400×300×1
γ	Discount factor	0.95
M	Minibatch size	128
α	Learning rate	10^{-4}
$\|R\|$	Size of replay memory	10^5
w_1	Weight of the action magnitude	100
w_2	Weight of the collision check	200
w_3	Weight of the distance from the target	2000
τ	Soft target update factor	5×10^{-4}
σ	Discriminating parameter for the Huber loss	0.1
β	Entropy temperature parameter	0.2

2) Implementation

We trained our SAC controllers on different scenes, see Fig. 5 for examples. Fig. 5(a-d) represent the first-stage training, in which the robot learns motion planning policies for scenes with cubic boxes. For the pre-training, we use a robotics simulator MuJoCo [30] due to its fast dynamics simulation. All the obstacles were box-shaped and had diverse sizes and positions to change the difficulty of the scenes. It is noted that we set the maximum number of obstacles as ten. The current state of some "obstacles" are set as zero if there are less than ten obstacles.

Fig. 5(e-h) shows the scenes used for further training with the pre-trained weights using box-shaped scenes. In Fig. 3(e-h), we used the open-source simulator Gazebo combined with ROS (Robot Operating System) for training due to its good

support for sensing systems in virtual environments. At this stage, the obstacles were in various sizes and positions. As shown in Figs. 3 and 5(e-h), we used Vision Module to detect the bounding boxes of the objects. The robot can use the knowledge from pre-training and adapt to these irregular shapes, to learn a more advanced control policy while considering the inaccuracies caused by sensory systems.

3) Baseline

For comparison, we trained the controller with seven other methods:

- **DQN (Deep Q-Network)**. Since it can handle only the discrete action space, we divided the range of the joint angle evenly into 20 values.
- **RRT (Rapidly-exploring Random Trees)**. We selected RRT to compare the query time with RL methods.
- **DDPG (Deep deterministic policy gradient)**. We used the similar parameter setting as SAC.
- **SAC-NHL (SAC** method that does **not** include **Huber loss)**. We used mean square error (MSE) as the loss to train the controller rather than the Huber loss.
- **SAC-NA (SAC** method that does **not** include the magnitude of the **actions)**. We used SAC to train the controller using the reward that does not consider the magnitude of the actions, see Eq. 3.
- **SAC-NUR (SAC** method that does **not** use the **unified representation)**. This is the scenario that does not use the unified representation for both training and testing environments.
- **SAC-SVC (SAC** method with **separate** training of **visual** output and **control** policy). This is the scenario that controller is trained separately from the 3D bounding box training. That is, the 3D bounding boxes are trained in advance.

4) Training Efficiency

To analyze the training efficiency of the SAC model for the manipulator obstacle avoidance and the advantage of using a reward function defined in Eq. 2, we compared the proposed SAC method with DQN, DDPG, SAC- NHL and SAC-NA using the scene with three obstacles. Fig. 6 shows the training progress, which shows that our proposed model outperforms the other methods in terms of the training speed and performance. Unlike DDPG, which achieved an average reward of only -25000, the SAC network learned policies with higher rewards. This is because overestimating the Q value cannot be solved by getting more exploration in the obstacle avoidance problem.

Conversely, with a longer training period, the effect is more severe, preventing DDPG from obtaining better performance. In addition, the results show that the method that uses MSE rather than the Huber loss does not converge. This is because the Huber loss is less sensitive to outliers in the data than the MSE. Also, the proposed approach performs better than the SAC-NA, which does not consider the magnitude of the action in the reward. This is because the weights in the proposed method tune the reward function according to the priority of the actions, collision and target reaching situation in the training process. We also compared our method with DQN, which mainly works for discrete action space. It can be seen that DQN performs better than other baselines and converges earlier than our method. This is because the training scenarios are relatively simple, and the robot's action space is larger. However, as DQN can easily fall into local minima in a small-scale discrete space, the convergence cannot reach a good result, which can be seen from Fig. 6.

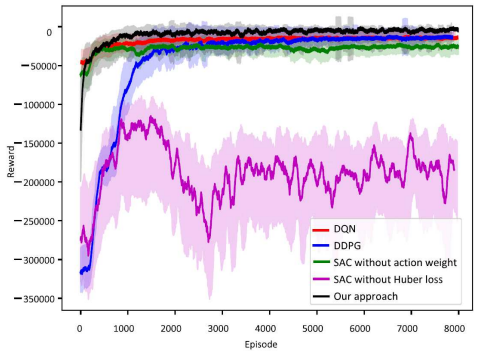


Fig. 6. The total distance and reward per episode for the scene in Fig. 5(c).

B. Evaluation in Simulation

To analyze the advantage of introducing the unified representation for the sim2real problem, we compared the control policy trained in Fig. 5(e-h) with that trained in the pre-training stage in Fig. 5(a-d). The latter is the regular sim2real method that directly transfers the policy from simulation to the real world. We tested these policies in three unseen scenes with various shapes and positions, see Fig. 7.

1) Performance Metrics

We adopted two metrics to measure the performance of different methods: the query time to reach the destination and the simulation success rate. The results are detailed in the next section.

2) Sim2Real Results

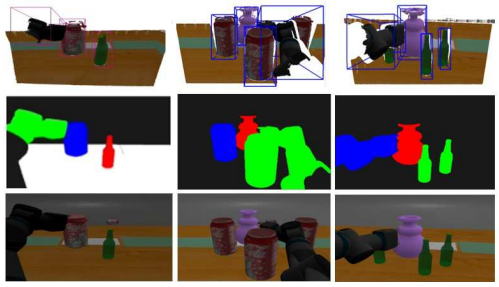
All the controllers were tested ten times for each scene in simulation in Fig. 7. The average evaluation results are listed in Table II. Fig. 7 shows three scenes, where the sizes and locations of the obstacles are different from those in the training scenarios. The task in Scene 1 is to reach the cola bottle in the first scene. The tasks for Scenes 2 and 3 are to reach the vase, as shown in Fig. 7(b) and (c), respectively.

Query time: We selected RRT to compare the performance. For our method, the results include both the time required to load the weights for the model and the time required to implement the joint actions. Table II presents the timing results for these two approaches and shows that compared with RRT, all the RL methods achieve shorter query time in all setups. This is because RL uses the trained control policy, while RRT performs online calculation and is a probabilistically complete motion planner that searches for feasible paths using a heuristic approach. Due to the computational overload, the search time marginally increases when the number of workspace obstacles increases. We anticipate that other sampling-based methods, such as

optimal RRT (RRT*) and probabilistic roadmaps (PRM) may have the same problem. Therefore, RRT is not guaranteed for real-time motion planning for robotic manipulators in complex scenes.

Avoidance analysis and generalization: The success rate indicates whether the control policy reaches the target without colliding with the obstacles based only on the visual information of the current state. We used SAC-NUR as a baseline. As shown in Table II, our approach achieves better success rate in the three test scenes (70%, 60%, 60%) compared with SAC-NUR (50%, 40%, 40%), and SAC-SVC (70%, 60%, 50%). For comparison, we also present the success rate for SAC-NA, which performs worse (40%, 30%, 30%) than the proposed approach, as the policy does not reach a good result during training.

In addition, the success rate of the proposed approach in Scene 1 is higher than both in Scenes 2 and 3. This is because the latter two scenes are more complex than Scene 1, and the vase is an irregular object with a bounding box that may be changing. As this uncertainty is included in the training process using the bounding box representation, SAC-UR adapts the obstacle avoidance policy to the inaccurate modeling of the obstacles caused by the sensory system.



(a) (b) (c)

Fig. 7. Three scenes for testing: (a) a scene with a cola bottle target and a single green obstacle; (b) a scene with a vase target and some obstacles located in random positions; (c) a scene with a vase target and some new obstacles with unseen geometric shapes.

TABLE II
QUANTITATIVE EVALUATION AND GENERALIZATION FOR DIFFERENT METHODS UNDER VARIED ENVIRONMENT SETUPS. THE BOLD FONT HIGHLIGHTS THE BEST OR WORST RESULTS IN EACH COLUMN.

Methods	Scene 1		Scene 2		Scene 3		Sim2Real	
	Query time (s)	Success rate	Query time (s)	Success rate	Query time (s)	Success rate	Query time (s)	Success rate
DQN	0.22	—	0.27	—	0.27	—	—	—
DDPG	0.23	—	0.31	—	0.31	—	—	—
RRT	**2.51**	—	**13.63**	—	**55.45**	—	—	—
SAC-NHL	0.25	—	0.33	—	0.33	—	—	—
SAC-NA	0.23	40%	0.32	30%	0.36	30%	—	—
SAC-NUR	0.22	50%	0.28	40%	0.28	40%	1.25	30%
SAC-SVC	0.22	70%	0.28	60%	0.28	50%	1.25	40%
SAC	0.22	**70%**	0.28	**60%**	0.26	**60%**	1.27	**50%**

C. Testing in the Real World

We performed experiments in unseen real environments with our overall architecture using a physical Fetch robot. To validate the generalization ability of the model to different real-world settings, we tested our trained model in dynamic scenes where the objects have different geometries and keep changing their positions. To achieve this, we put obstacles and targets on a box mounted on a Turtlebot, as shown in Fig. 8(a) and 8(b). The Turtlebot rotates 360 degrees clockwise, which drives the obstacles and targets to rotate 360 degrees. A new scene was generated each time the Turtlebot rotated 30 degree to a new position. We used the depth sensor on the Fetch robot to obtain the 3D bounding boxes of the objects in the environment. The right image in Fig. 8(c-j) shows the result of the scene model. A magnet was mounted on the end-effector of the robot, and the target was picked when the robot reached it.

We also evaluated the performance of the proposed approach from the sim2real viewpoint. We trained a model in a similar way as the proposed approach, using geometries and locations as observations instead of the 3D bounding boxes. In the training scenario, we used all box-shaped objects. Then, we applied the new model to scenes where objects were represented as bounding boxes. Ten scenes in both simulated and real environments were tested.

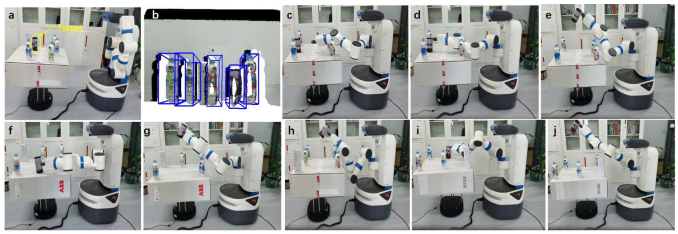


Fig. 8. Experiment Setup. (a-b) The real experimental setup (a) and the real time reconstruction and recognition (b) of the scene using a depth sensor. (c-j) The Fetch robot successfully reached the purple metal box as a target while the whole scene was changing.

The success rate is shown in Table II. It was found that the robot was able to reach the target in the real world for five times. In five times, the robot collided with the obstacles when two consecutive scenes required a larger magnitude of the actions. This was mainly because of the low quality of the point clouds captured by our low-cost depth sensor, which possibly resulted in the inaccurate location of the dynamic objects. Fig. 9 shows that the direct training without detecting the bounding boxes, SAC-NUR. As the point cloud of some obstacles is unclear, the collision occurred when the robot arm reaches the target.

For simulated scenarios, there was three failed cases using our model. The statistic results are shown in Table II. It can be seen that the proposed method outperforms other methods in

transferring the model from simulation to the real world. Also, it can be seen that the control policy with trained bounding boxes (SAC-SVC) performs better than SAC-NUR. This demonstrate the effectiveness of the bounding box representation. The performance of SAC-SVC performs worse than our proposed method with an end-to-end training in complex scenarios (e.g. Scene 3 in simulated environment). This is caused by the accumulated errors. In summary, the experiment demonstrates the effectiveness of the unified representation based on an end-to-end training in learning real-world interactions and shows a robust generalization from simulation to the real world.

Fig. 9. Physical experiment with direct training without detection of 3D bounding boxes (SAC-NUR).

VI. Conclusions and Future Work

In this paper, we presented a DRL-based technique to train a manipulator in a virtual simulator to avoid obstacles. Our technique employs a SAC method together with a novel reward function and the Huber loss to significantly reduce the training time. These settings resulted in a fast response time and adaptation to unseen environments. We have evaluated our method in both virtual and physical environments. All the presented results were obtained upon a single CPU. The model update and query times may be improved with the use of multi-core or GPUs, which can be performed in future work.

In future work, we plan to explore the combination of the DRL approach with Long Short-Term Memory (LSTM) units to handle the purely vision-based navigation, in which the robotic manipulators reach the target based upon the visual information of the current state only. We believe that the system will be able to handle heavy occlusion among various objects in the scene with the use of the memory.

References

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Syst.,* pp. 1097–1105, 2012.

[3] A. D. Dragan, G. J. Gordon, and S. S. Srinivasa, "Learning from experience in manipulation planning: Setting the right goals," in *Rob. Res.,* Springer, 2017, pp. 309–326.

[4] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Robotics: Sci. Syst.*, 2017.

[5] Y.Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, F.F. Li, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *IEEE Int. Conf. Robot. Automat.* 2017, pp. 3357–3364.

[6] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," *RSS 2017 workshop on New Frontiers for Deep Learning in Robotics*, 2017.

[7] T. Jurgenson and A. Tamar, "Harnessing reinforcement learning for neural motion planning," in *Robotics: Sci. Syst.*, 2019.

[8] M. Duguleana, F. G. Barbuceanu, A. Teirelbar, and G. Mogan, "Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning," *Robot Comput. Integr. Manuf.*, vol. 28, no. 2, pp. 132–146, 2012.

[9] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *Int. J. Rob. Res.*, vol. 37, pp. 421–436, 2018.

[10] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *IEEE Int. Conf. Robot. Automat.*, 2016, pp. 3406–3413.

[11] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Eur. Conf. Artificial Life*, 1995, pp. 704–720.

[12] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuo-motor control from simulation to real world for a multi-stage task," in *Conf. Rob. Learning.*, 2017, pp. 334-343.

[13] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, K. Bousmalis, A. Irpan, et al., "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4243–4250.

[14] K. Fang, Y. Bai, S. Hinterstoisser, S. Savarese, and M. Kalakrishnan, "Multi-task domain adaptation for deep learning of instance grasping from simulation," in *IEEE Int. Conf. Robot. Automat.*, 2018, pp. 3516–3523.

[15] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Comput.*, vol. 5, no. 4, pp. 613–624, 1993.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Int. Conf. Mach. Learn*, *PMRL*, 2018, pp. 1861-1870.

[17] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," a*rXiv preprint arXiv:1511.07111*, 2015.

[18] Z. Wei, W. Chen, H. Wang, and J. Wang, "Manipulator motion planning using flexible obstacle avoidance based on model learning," *Int. J. Adv. Rob. Res.,* vol. 14, no. 3, 2017.

[19] N. Das, N. Gupta, and M. C. Yip, "Fastron: An online learning-based model and active learning strategy for proxy collision detection," in *Conf. Rob. Learning*, 2017, pp. 496-504.

[20] H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *IEEE Int. Conf. Robot. Automat.*, 2019, pp. 2118-2124.

[21] S. Singla and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge," *IEEE Trans. Intell. Transp. Syst*., 2018.

[22] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep reinforcement learning for collision avoidance of robotic manipulators," in 2018 *Europ. Contr. Conf.*, 2018, pp. 2063– 2068.

[23] S. Wen, J. Chen, S. Wang, H. Zhang, and X. Hu, "Path planning of humanoid arm based on deep deterministic policy gradient," in *IEEE Int. Conf. Rob. Biom.*, 2018, pp. 1755–1760.

[24] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IEEE Int. Conf. Intell. Robots Syst.*, 2017, pp. 2371–2378.

[25] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep successor reinforcement learning," *arXiv: Machine Learning*, 2016.

[26] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning,* vol. 8, no. 3, pp. 229–256, 1992.

[27] V. Konda and J.N. Tsitsiklis, "Actor-critic algorithms," in *Adv. Neural Inf. Process. Syst.,* 2000, pp. 1008-1014.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Int. Conf. Mach. Learn.*, *PMRL*, 2014, pp. 387-395.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Int. Conf. Learn. Represent.,* 2015.

[30] E. Todorov, T. Erez and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026-5033.