2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

SDF-Based Graph Convolutional Q-Networks for Rearrangement of Multiple Objects

Hogun Kee, Minjae Kang, Dohyeong Kim, Jaegoo Choy, and Songhwai Oh

***Abstract*— In this paper, we propose a signed distance field (SDF)-based deep Q-learning framework for multi-object rearrangement. Our method learns to rearrange objects with non-prehensile manipulation, e.g., pushing, in unstructured environments. To reliably estimate Q-values in various scenes, we train the Q-network using an SDF-based scene graph as the state-goal representation. To this end, we introduce SDFGCN, a scalable Q-network structure which can estimate Q-values from a set of SDF images satisfying permutation invariance by using graph convolutional networks. In contrast to grasping-based rearrangement methods that rely on the performance of grasp predictive models for perception and movement, our approach enables rearrangements on unseen objects, including hard-to-grasp objects. Moreover, our method does not require any expert demonstrations. We observe that SDFGCN is capable of unseen objects in challenging configurations, both in the simulation and the real world.**

I. Introduction

Learning robot manipulation skills in a situation where a variety of objects are placed is a challenging problem in robotics. There are many studies where only objects in direct contact with the robot are placed in the workspace. Still, these methods can easily fail if the composition of objects changes or a situation different from the training data comes out. Therefore, it is important to learn manipulation skills to perform tasks reliably even in the presence of obstacles or a large number of objects.

In this study, we address the rearrangement problem in multiple object situations. Unlike the problem of moving a single object to the target position, increasing the number of objects causes the remaining objects to act as obstacles while one object moves toward its goal. Despite its difficulties, solving the problem of rearranging objects to a given set of goals allows us to cover a wide variety of robotic tasks such as object singulation, sorting objects by type, or aligning objects in a shape. These derived tasks can be solved simply by setting the goal configurations according to the task.

We are interested in learning the non-prehensile action policy for a planar rearrangement task. To move all the objects to the target locations, a sequence of robot actions according to the current observation and target positions is required. Previous studies have solved this decision-making problem through task and motion planning (TAMP) [1] , [2] or reinforcement learning (RL) [3].

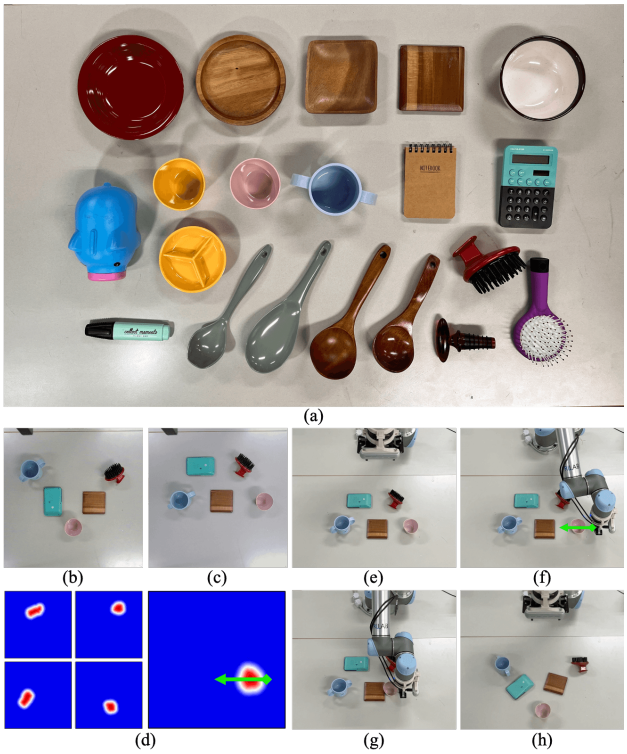

Fig. 1. Real world evaluation is performed on the various objects which are slippery, round and thin and difficult to grasp (a). The goal configuration is given as an RGB image (b) and the initial configuration is given as (c). The SDFs of the objects in the current scene are obtained (d), where the green arrow represents the pushing action. Three subsequent snapshots (e-g) showing the one-step push action. At every step, the robot receives a camera image at the initial position (e). When the action is determined, the robot arm moves to the pre-push position (f) and pushes with a fixed distance (g). Finally, the achieved goal configuration is shown in (h).

In the context of task and motion planning (TAMP), various methods have been studied to solve this problem efficiently, but they are limited to a single type of objects or objects for which the 3D models are known. Although there have been studies of rearranging objects seen for the first time, the lack of prior knowledge of objects was supplemented by grasp prediction models such as Contact-GraspNet under the assumption that all objects are graspable [4], [5].

To overcome these limitations, we propose a signed distance field (SDF)-based deep reinforcement learning framework. Reinforcement learning methods provide a promising avenue for learning manipulation skills from raw pixel observations. Instead of using raw camera images, we define the state and goal spaces as the set of SDFs of objects and

This work was partly supported by Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01190, [SW Star Lab] Robot Learning: Efficient, Safe, and Socially-Acceptable Machine Learning, 50%) and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2022R1A2C2008239, General-Purpose Deep Reinforcement Learning Using Metaverse for Real World Applications, 50%). *(Corresponding author: Songhwai Oh.)*

H. Kee, M. Kang, D. Kim, J. Choy, and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul, 08826, Korea (e-mail: {hogun.kee, minjae.kang, dohyeong.kim, jaegu.choy}@rllab.snu.ac.kr, songhwai@snu.ac.kr.)

action space based on the SDFs. Then we construct a scene graph having the SDFs of objects as graph nodes. Finally, we propose a new Q-network structure, signed distance field-based graph convolutional network (SDFGCN), to estimate the Q-values as the function of our states and actions. With our SDF-based Markov decision process (MDP) definition and Q-network, our agent can train the push policy for tabletop rearrangement on unseen objects. In our settings, the robot arm moves objects through a non-prehensile action (e.g., pushing) in situations where hard-to-grasp objects, such as bowls, plates, etc., are placed, as shown in Figure 1. These objects are round, thin, or slippery so that stable grasping is not guaranteed. Since the grasp predictive model cannot guarantee the movement of objects through pick-and-place, previous grasp-based methods cannot be used to find solutions for these sets of objects, while our method performs robustly with pushing action based on the signed distance field of objects.

In summary our contributions are as follows:

- We present an RL framework that solves the problem of rearranging unseen objects using SDF-based state and action spaces. These SDF-based representations allow the robot to reliably push objects without a 3D model and better understand the geometric properties of the scene.
- We propose SDFGCN, a new Q-network structure that can estimate Q-values from SDF-based scene graphs. We demonstrate that SDFGCN can effectively learn Q-values in unstructured environments showing successful sim-to-real transfer.
- To the best of our knowledge, SDFGCN is the first model-free reinforcement learning method for non-prehensile rearrangement problems on unknown objects.

II. Related Work

In this section, we review related studies regarding 1) methods for learning robotic manipulation skills through reinforcement learning, and 2) methods for rearranging objects in multi-object environments.

Reinforcement Learning in Robotics. Reinforcement learning methods have enabled robot controllers to find solutions for many complex tasks such as connector insertion [6], ball throwing [7], fabric folding [8], [9], and more. The RL methods can learn the policies without human demonstrations or expert data, but it requires a lot of interaction with the environment. To increase data efficiency and improve exploration, it is crucial to define the appropriate state and action spaces for the task. In this work, we focus on the studies of reinforcement learning using raw sensory input, which require less effort to match the state space. Fujita et al. [10] receives RGB observations from the wrist camera and performs the targeted grasping in the cluttered scene. To improve the perception module, Wang et al. [11] obtains segmentation masks of foreground, robot, and object from camera input and achieves effective policy learning through object-level reasoning. Some studies train two action spaces of grasping and pushing simultaneously and combine two robot motions to achieve task goals [12], [13]. Hundt et al. [14] shows safe and efficient exploration by finding a safe action space instead of exploring the entire action space. Instead of using raw sensory observation, our method calculates the SDFs of objects placed in the scene and use the obtained SDFs to define the state and action spaces.

Multi-Object Rearrangement Many researchers tried to solve rearranging objects in multi-object environments. Li et al. [3] uses the reinforcement learning method to learn the policy for block stacking. Based on the rapidly-exploring random trees (RRT) [15], Huang et al. [16] trained a pushing policy for large-scale rearrangement planning with up to 100 objects and packing factors of up to 40%. However, these methods use objects of the same shape, and the state includes the 3D positions of blocks. As a study conducted with various objects, Song et al. [1] performs a large-scale sorting task using Monte Carlo tree search (MCTS). However, as a tradeoff to reduce runtime, the action space is defined too simply and requires many steps to achieve a goal configuration.

Another approach is to use the grasp generation method, such as [4], [5]. Based on grasp generation, Qureshi et al. [4] performed rearrangement with the stable grasping of unseen objects. It generates a scene graph to create a sequence of object selection and placement actions based on scene segmentation. Goyal et al. [5] learns an optical flow model and plans to minimize the flow between current and goal scenes. Although these methods can be performed on unseen objects, the premise is that all objects must be able to grasp. In contrast, we train the push behavior policy for rearranging objects using SDF-based scene graphs. With SDF-based state and action representations, our method enables robust rearrangement of unseen objects without grasp generation methods.

III. Background

A. Goal-Conditioned Deep Q-Learning

The goal of the deep Q-learning is to estimate the state-action value function $Q(s, a, g)$ with the Q-network. The Q-network can be parameterized by θ and we can formulate the greedy policy by choosing the action that maximizes the predicted Q-value: $\pi(s, g) = \arg\max_a Q(s, a, g; \theta)$.

Following the Bellman equation, θ can be optimized by minimizing the Bellman loss:

$$L(\theta_i) = \mathbb{E}_{s,a,r,s',g}\left[(y - Q(s, a, g; \theta_i))^2\right] \tag{1}$$

at iteration i, with the target value

$$y = r(s, a, s', g) + \gamma \max_{a'} Q(s', a', g; \theta_i^-). \tag{2}$$

Here, r is the reward function, and γ is the discount factor. θ^- is the parameters of the target network, which is the lagged version of the original Q-network.

B. Fast Marching Method

In this paper, we use the fast marching method (FMM) [17] to compute the signed distance field from the segmentation masks. FMM is a method for solving boundary value problems of the Eikonal equation:

$$F(x)|\nabla T(x)| = 1,$$

where F is the speed function, and T is the time function; this describes the evolution of a closed curve at a point x on the curve. By solving this equation, we can obtain the

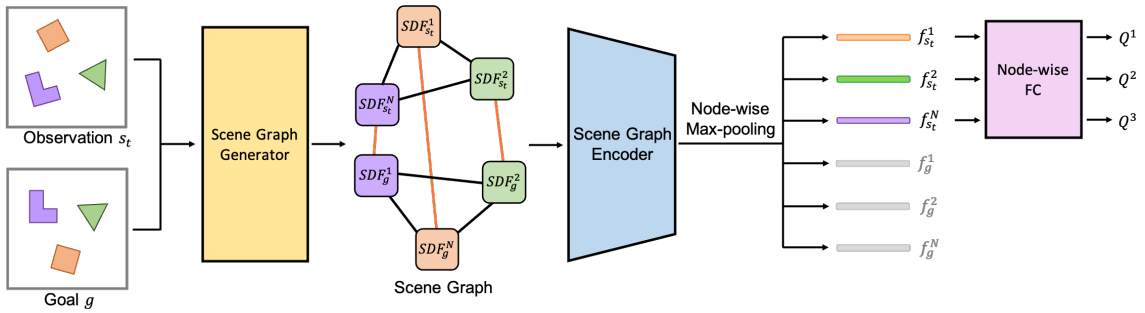

Fig. 2. SDFGCN consists of two parts: a scene graph generator and a scene graph encoder. The scene graph generator creates an SDF-based scene graph $G(s_t, g)$ from the current image s_t and goal image g. Then, based on the node features extracted through the scene graph encoder, we can get object-wise Q-value using a shared fully connected layer. The Q-value of each object becomes 8 dimensions, which is the number of angle bins in the pushing direction.

time at which the contour crosses a point x, which means the distance from the object's surface.

Given a segmentation mask m, FMM can be applied to compute a signed distance field in the form of a 2D matrix $SDF = FMM(m)$, where $SDF \in \mathbb{R}^{H \times W}$ and H and W denote the height and width of m. The value of a pixel has a distance to the closest surface of the object, and the sign encodes whether the point is inside (negative) or outside (positive) of the surface.

IV. Problem Description

In this paper, we address the planar rearrangement task on unseen objects from RGB vision, as depicted in Figure 1. Since our goal is to learn a push policy, we define the object-centric push based on SDF which can represent geometric properties of an object such as the center, size and shape. We obtain SDFs of objects in the scene and find the best push action according to the SDFs. To match the goal configuration, we also consider the SDFs of objects in the goal scene.

We formulate the rearrangement problem as a goal-conditioned MDP. The goal is to place N objects $O = \{o_1, o_2, ..., o_N\}$ in the same position as the placement of the goal scene.

a) State space: At each timestep, the agent receives an RGB image of the current scene from the fixed mounted camera. Then, we get the SDFs of objects from the RGB image using the unseen clustering network (UCN) [18] and FMM method. The set of all SDFs become a state. A more detailed description is in the proposed method section.

b) Goal space: For each episode, the goal configuration is given as an RGB image. Similar to the state space, we can obtain the SDFs of objects from the goal image. Then a goal is defined as the set of SDFs of objects in the goal configuration.

c) Action space: The action is given by (o, θ), where o denotes the object to push among all detected objects in the scene, and θ is the pushing direction discretized into eight angle bins. Once the object and pushing direction are determined, the starting point of the pushing action is obtained as the intersection of the surface of the object and the straight line passing through the object's center at the angle of θ. The robot arm pushes the object for a fixed distance from its surface.

d) Reward function: The reward function is obtained by the sum of the reward functions of each pair of objects. For m detected objects in current scene s and n detected objects in goal g, the agent finds k object pairs $((o_1^s, o_1^g), \ldots, (o_k^s, o_k^g))$, where k is the number of matching objects. The number of objects detected in current scene and the goal scene can be different.

For the i^{th} pair of objects (o_i^s, o_i^g), x_i^s and x_i^g denote the center of the i^{th} object in the scene s and the goal g. Then the reward function of the object is given as the difference in distance to its goal position, $r_i(s, a, s'|g) = C * (||x_i^s - x_i^g|| - ||x_i^{s'} - x_i^g||)$, where C is a constant value. Finally, the one step reward is calculated as the sum of the rewards of all object pairs, $r(s, a, s'|g) = \sum_i r_i(s, a, s'|g)$.

V. Proposed Method

We propose SDFGCN, a Q-network that can estimate Q-values on the set of SDFs. Since the number of SDFs depends on the number of objects, SDFGCN generates a graph-structured input, SDF-based scene graph, and finds the Q-function based on it. The structure of SDFGCN is illustrated in Figure 2. SDFGCN consists of two main components: a scene graph generator and a scene graph encoder.

We first describe how we generate the scene graph having SDFs as node features. Then we model the Q-network as a graph convolutional network suitable for estimating Q-values from scene graphs. Finally, we explain how we get the pushing policy with this Q-network.

A. SDF-based Scene Graph

Figure 3 shows an overview of our scene graph generation. At each timestep t, we use the pre-trained UCN to detect objects in the current scene image s_t and get the segmentation masks of objects. Then applying the FMM method on the segmentation masks, we can obtain a set of object SDFs of the current scene $SDF(s_t) = [SDF_1^{s_t}, SDF_2^{s_t}, \ldots, SDF_m^{s_t}]$, where m denotes the number of detected objects. Each SDF is in the form of a 2D matrix, $SDF_i^{s_t} \in \mathbb{R}^{H \times W}$, where H and W denote the same height and width of the current image.

After obtaining the set of SDFs in the current scene, the scene graph generator generates a scene subgraph having the SDFs as nodes. For arbitrary scene s, the subgraph $G^{sub}(s) = (V, E)$ has a node feature $V_i = (SDF_i^s, f^s)$ for each object i, where f^s is the scene label which is a binary value indicating whether s is the goal scene or not. For the current scene s_t, the scene label has the value of 0. Then

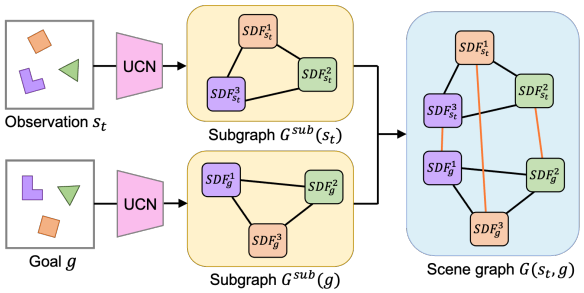

Fig. 3. Overview of the scene graph generator. Our scene graph generation consists of merging two scene subgraphs. The subgraph of each scene is a complete graph using the SDFs of objects as nodes. The SDFs are obtained through UCN and FMM Method and is the same resolution as the observed RGB image.

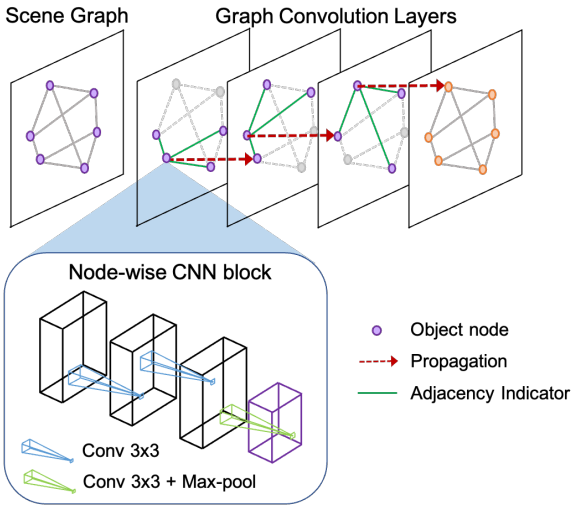

Fig. 4. Architecture of the scene graph encoder. The graph encoder of SDFGCN consists of CNN layer-based graph convolution layers. Since our scene graph has a node feature in the form of image, the network propagates the features through CNN blocks. We construct a node-wise CNN block with three Conv layers with kernels of size 3×3, and the kernels of the node-wise CNN blocks are shared in the same GCN layer. Finally, the encoder network consists of two CNN layer-based GCN layers.

all pairs of nodes in the subgraph are connected by edges to form a complete graph.

Similar to the process of generating a subgraph of the current scene, we can obtain a set of SDFs of the goal scene $SDF(g) = [SDF_1^g, SDF_2^g, \ldots, SDF_n^g]$ and generate a subgraph $G^{sub}(g)$ from the goal image g. The difference is that the number of detected objects is n and the scene label has the value of 1.

The next step is to combine the two subgraphs to form a full scene graph $G(s_t, g)$. Before combining two subgraphs, we need to determine the correspondence between the objects in two scenes. Inspired by [4], we use similarity scores between two sets of objects to find the correspondence. Using the segmentation masks generated above, we get the local RGB information of each object patch and feed it to a pre-trained ResNet model to extract the object's visual features. We then use these features to compute the L2 norm similarity scores for all object pairs in the form of an $m \times n$ matrix. The Hungarian algorithm [19] determines the optimal object correspondence at the lowest cost. If m and n are different, some nodes will be left without matching nodes. Finally, we connect new edges between the corresponding object nodes to combine the two subgraphs and get a full scene graph.

B. Graph Convolutional Network on SDF Nodes

To estimate the goal-conditioned Q-value from the state-goal representation in the form of a scene graph, we need a neural network that can operate on graphs. We consider a multi-layer graph convolutional network (GCN) for this.

A GCN is the generalization of a convolutional neural network (CNN) to graphs of arbitrary structures with the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G with self-loops added. I_N is the $N \times N$ identity matrix, $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$ and $W^{(l)}$ is the trainable weight matrix of l^{th} convolution layer. $\sigma(\cdot)$ is an activation function and $H^{(l)}$ denotes the matrix of activations in the l^{th} layer.

The majority of existing approaches using GCN represent embedding vectors as one-dimensional vectors. They assumed $H^{(l)}$ to be $N \times d_l$ matrix and $W^{(l)}$ to be a $d_l \times d_{l+1}$ matrix. In these works, the multilayer perceptron (MLP) is used to construct the graph convolution layer that propagates information along edges.

In our method, since the SDF-based scene graph has two-dimensional SDF images as node features, MLP is not suitable for propagation. As CNN has shown great success in image processing, we use the CNN layers as the propagation layers in the scene graph encoder instead of MLPs. Figure 4 shows the operation of the graph encoder. We construct the graph encoder with two sets of CNN blocks and propagation layers, where each CNN block consists of three 3×3 convolution layers. The input graph node features are given to CNN blocks, and each CNN block is followed by a CNN-based propagation layer. The network learns to extract local node features from the CNN blocks and shares the information with neighboring nodes through the propagation layers. After the last propagation layer, the outputs of graph encoder are flattened and passed into two linear layers to estimate object-wise Q-values.

C. Training SDFGCN

We use the Q-learning algorithm [20] to train the goal-conditioned Q-function $Q^{\pi}(s, a, g)$. The overall architecture of the Q-network is shown in Figure 2. Since the number of detected objects in the current and goal scenes, m and n, may not be constant, we set the maximum number of objects to $N \geq \max(m, n)$. After creating an SDF-based scene graph, we add some empty nodes for each scene to have N nodes on each subgraph and $2N$ nodes on the entire scene graph.

Since we use object-centric push, the action space is defined along the objects. After extracting the features of each object node through the scene graph encoder, we estimate the Q-values by leaving only the first m nodes from the current scene s_t, and discard the remaining nodes including empty nodes. As we optimize the Bellman equation with gradient descent on mini-batches, we use zero padding for empty nodes to form the states in the same shape. Since the gradient flows according to the adjacency matrix, we can prevent gradients from flowing into empty nodes by constructing the adjacency matrix with a value of 0 for the empty nodes.

We also use hindsight experience replay (HER) [21] by replacing real goals with achieved goals in selected transitions during replay.

VI. EXPERIMENTS

In order to evaluate the performance of our system, we devised three sets of experiments. In the first experiment, the agent learns to solve the rearrangement problem with a fixed number of objects. We compared our method with other methods, including a rule-based algorithm and DQN methods with different structured Q-network. Second, we show a generalization of our method for changing the number of objects in the scene. Third, we demonstrate a sim-to-real generalization of our method in several challenging real-world scenarios, including unseen rearrangement tasks.

A. Experimental Setup

We use a 6-DoF Universal Robots UR5 mounted with a Robotiq 2F-85 Gripper at the end effector to perform multi-object rearrangement tasks both in simulation and in the real world. We use a MuJoCo simulator as the simulation environment. An RGB camera is mounted in the environment to capture downsampled RGB images to 96×96 resolution. In the real world experiments, the RGB images are captured from an Intel RealSense D435 camera mounted on the wrist of the robot. The hardware setup is illustrated in Fig. 5.

In the simulated environment, we use 3D object models from the ShapeNet dataset for training and test sets. The training set contains 32 models, and the test set includes 16 models. At the beginning of each episode, the objects are randomly placed on the table in a stable position without collision. The goal scene is obtained by capturing a specific arrangement from the same camera view. For training, we uniformly sample the goal positions of objects within a feasible region to avoid collisions.

B. Comparison to Baseline Methods

To validate our approach, we first compared our proposed method with other baseline methods in the simulated environment. We consider the following algorithms:

Rule-based agent: We implement a rule-based agent using the objects' ids and ground truth positions. The agent randomly selects an object which has not reached the goal and pushes the object in the nearest direction toward the goal point.

Pose-input GCN: The pose-input method, which is the ablation over the input representation, uses the detected position of each object as a node feature. The Q-network is constructed with GCN using a MLP as propagation.

Ground truth pose-input GCN: This method uses the ground truth positions as node features instead of the detected positions. The structure of the Q-network is the same as Pose-input GCN.

Segmentation GCN: This method is another ablation of SDFGCN over the input representation. It uses the binary segmentation masks as node features instead of the SDFs.

SDF-input CNN: This is the ablation of SDFGCN over the graph convolution layers. The Q-network is constructed with CNNs without GCNs.

SDFGCN-separate: This is the ablation of SDFGCN over the graph structure. We generate a scene graph without connections between object nodes in the same scene. The scene graph only contains edges between corresponding object nodes and does not propagate between nodes in the same scene.

Methods	Train Set			Test Set		
	SR	Steps	Error	SR	Steps	Error
Rule-based agent	68%	57.7	30.7	69%	59.1	28.8
Pose GCN	40%	45.1	30.1	33%	**33.7**	33.7
GT-Pose GCN	41%	42.0	31.5	51%	44.5	31.3
Segmentation GCN	18%	44.3	31.4	16%	41.6	26.3
SDF CNN	75%	41.1	30.0	70%	37.3	30.3
SDFGCN-Separate	67%	39.1	**25.9**	65%	36.2	**26.2**
SDFGCN	**80%**	**35.0**	29.5	**80%**	34.6	31.3

TABLE I: Performance comparison between SDFGCN and baseline methods in MuJoCo simulator. SR is the success rate, Steps is the average number of pushing steps, and Error is the average position error (in mm) between the desired arrangement and the achieved arrangement. All the Q-networks are trained on random rearrangements of 3 objects.

In this experiment, all agents are trained with random tabletop scenes containing three objects randomly selected from the training set. Then we tested for each agent in 200 different scenes containing three unseen test objects. The objects in the test set are all different in category, shape, size, and color from the objects in the train set.

The results are shown in Table I. We measured success rates, number of steps and distance errors as performance metrics. We note that the average number of steps and average distance error are calculated only on successful scenarios. We can see our proposed method, SDFGCN achieves a success rate of 80% at rearranging three objects which is the highest performance among those methods. We find that both the SDF-input and GCN layers are quite important to the Q-network structure. The position-input methods failed to learn the optimal Q-functions despite using ground truth information. Many of the failure cases are due to unexpected collisions. We argue that the SDF-based state representation helps SDFGCN perform reliably over the size and shape of objects. Compared to the CNN model, SDFGCN learns Q-values from sets of SDFs more efficiently due to permutation invariance.

C. Generalization to Different Number of Objects

We performed a set of experiments by changing the number of objects from two to six. We trained SDFGCN in three ways depending on the number of objects in the training scenes.

SDFGCN-fixed is trained on three objects during the whole training and **SDFGCN-random** is trained on random number of objects between two and four. **SDFGCN-curriculum** is trained with curriculum learning, increasing the number of objects from two to four. For the curriculum learning, we first place only two objects in the training scenes and increase the number of objects when the success rate on the train set reaches 80%.

The results are shown in Table II. We note that all objects for evaluation are unseen during training and the average number of steps and final distance error are calculated only in successful scenarios. Although SDFGCN-fixed is trained

Configurations	Rule-based agent			SDFGCN-fixed			SDFGCN-random			SDFGCN-curriculum		
	SR	Steps	Error	SR	Steps	Error	SR	Steps	Error	SR	Steps	Error
2 Objects	78%	35.2	31.2	88%	20.3	33.0	87%	21.0	32.0	84%	24.0	30.8
3 Objects	69%	59.1	28.8	80%	34.6	31.3	76%	34.3	30.1	83%	34.7	28.6
4 Objects	37%	69.3	31.0	61%	47.5	29.6	68%	48.5	29.2	64%	48.9	28.9
5 Objects	25%	89.8	25.8	12%	58.2	30.0	60%	85.9	27.2	61%	77.3	26.9
6 Objects	20%	111.2	31.0	0%	-	-	35%	99.5	26.9	44%	100.9	28.7
Average	45%	72.9	29.6	48%	40.2	31.0	65%	57.8	29.1	67%	57.2	28.8

TABLE II: Generalization to a different number of objects in MuJoCo simulator. All methods except the rule-based agent use the SDFGCN as Q-networks; the only difference is the change in the number of objects placed in the training scenes. All evaluations are done with a new set of objects which are unseen during training. We cannot measure the average steps and errors for the SDFGCN-fixed on a 6 objects configuration because the agent fails to all scenarios.

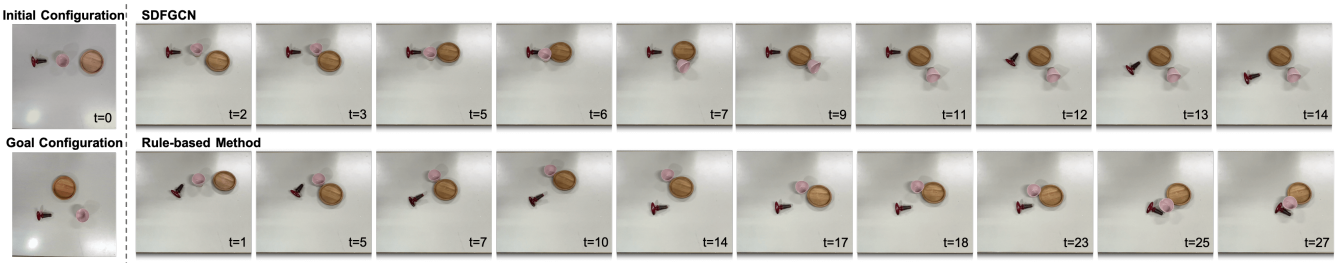

Fig. 5. An examples of rearrangement scenario performed with SDFGCN and the rule-based method. The initial configuration and goal configuration are given as RGB images. SDFGCN successfully swapped the pink cup and brown plate and found a successful push sequence, despite the pink cup falling down in the middle of the scenario. The rule-based method failed to swap, and all the objects were entangled.

Configurations	Rule-based agent			SDFGCN-fixed			SDFGCN-random			SDFGCN-curriculum		
	SR	Steps	Error	SR	Steps	Error	SR	Steps	Error	SR	Steps	Error
3 Objects	70%	11.0	22.7	80%	12.0	25.9	80%	10.8	28.1	80%	9.5	31.1
4 Objects	40%	16.3	21.3	40%	24.5	27.5	80%	23.5	29.1	80%	26.0	23.8
5 Objects	30%	18.7	24.3	20%	32.7	27.0	60%	19.0	28.6	50%	21.3	19.5

TABLE III: Performance of real robot evaluation as the number of objects increases. The Rule-based agent shows a short number of steps but has a low success rate. This means that the rule-based agent performs well in simple scenarios, while SDFGCN is capable of more difficult scenarios as well.

Methods	SR	Steps	Error
Rule-based	70%	11.0	22.7
SDF CNN	70%	**9.4**	**21.8**
SDFGCN-Separate	60%	14.5	28.0
SDFGCN	**80%**	12.0	25.9

TABLE IV: Performance of real robot evaluation with three random objects. We found that edge connection of scene graphs has a significant impact on performance.

only on three objects, in scenarios with less than 5 objects, it performs similarly to the models trained on 2 to 4 objects. Training with a varying number of objects allows SDFGCN to generalize on scenarios with much more objects. We can see that SDFGCN is effective in estimating Q-function in situations where the number of objects changes.

D. Real Robot Experiments

We evaluated 10 scenes for each method. The initial and goal configurations for each scenario are predefined as RGB images. To replicate the initial configurations, we placed the objects as visually consistent as possible. The results are shown in Table III and Table IV.

The failure cases of SDFGCN usually occur when the agent is trapped in a loop and repeats the same states or objects are mismatched due to noisy feature extraction. On the other hand, the rule-based agent failed when the objects were entangled with each other. Once objects become entangled, the object detection does not work properly, and the entangled objects often do not separate from each other until the end of the scenario. Figure 5 shows that the rule-based agent fails due to entangled objects, but SDFGCN successfully rearranges objects while avoiding collisions. The video for additional examples of real experiments is included in the supplementary materials.

VII. Conclusions

We have proposed a deep Q-learning framework based on object SDFs for multi-object rearrangement. Our proposed Q-network, SDFGCN, generates a scene graph with SDFs as graph nodes and learns Q-values as a function of the scene graph. Using the CNN-based graph convolution layers as propagation layers, SDFGCN can learn Q-values from the sets of SDFs with permutation invariance. We evaluate SDFGCN on challenging problems in the simulation and real world experiments and show that SDFGCN is capable to unseen hard-to-grasp objects without grasp generation methods. However, we have observed that SDFGCN often fails due to the entanglement between objects. We found that object-centric push action is difficult to resolve these entanglements. Therefore, in future work, we plan to train SDFGCN in a combined action space of push and grasp to improve the rearrangement performance.

REFERENCES

[1] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9433–9440.

[2] S. H. Cheong, B. Y. Cho, J. Lee, C. Kim, and C. Nam, "Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7791–7797.

[3] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4051–4058.

[4] A. H. Qureshi, A. Mousavian, C. Paxton, M. C. Yip, and D. Fox, "Nerp: Neural rearrangement planning for unknown objects," *Robotics: Science and Systems*, 2021.

[5] A. Goyal, A. Mousavian, C. Paxton, Y.-W. Chao, B. Okorn, J. Deng, and D. Fox, "Ifor: Iterative flow minimization for robotic object rearrangement," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2022, pp. 14 787–14 797.

[6] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5548–5555.

[7] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2351–2358.

[8] R. Lee, D. Ward, V. Dasagi, A. Cosgun, J. Leitner, and P. Corke, "Learning arbitrary-goal fabric folding with one hour of real robot experience," in *Conference on Robot Learning (CoRL)*. PMLR, 2020.

[9] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning (CoRL)*. PMLR, 2018, pp. 734–743.

[10] Y. Fujita, K. Uenishi, A. Ummadisingu, P. Nagarajan, S. Masuda, and M. Y. Castro, "Distributed reinforcement learning of targeted grasping with active vision for mobile manipulators," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9712–9719.

[11] Y. Wang, G. N. Narasimhan, X. Lin, B. Okorn, and D. Held, "Roll: Visual self-supervised reinforcement learning with object reasoning," in *Conference on Robot Learning (CoRL)*. PMLR, 2020.

[12] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[13] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, "Deep reinforcement learning for robotic pushing and picking in cluttered environment," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 619–626.

[14] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager, ""good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6724–6731, 2020.

[15] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[16] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 211–218.

[17] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts." *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[18] Y. Xiang, C. Xie, A. Mousavian, and D. Fox, "Learning rgb-d feature embeddings for unseen object instance segmentation," in *Conference on Robot Learning (CoRL)*. PMLR, 2020.

[19] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in Neural Information Processing Systems*, vol. 30, 2017.