Graph-Transporter: A Graph-based Learning Method for Goal-Conditioned Deformable Object Rearranging Task

Yuhong Deng[1,2], Chongkun Xia[2], Xueqian Wang[2,†] and Lipeng Chen[1]

***Abstract*—Rearranging deformable objects is a long-standing challenge in robotic manipulation for the high dimensionality of configuration space and the complex dynamics of deformable objects. We present a novel framework, Graph-Transporter, for goal-conditioned deformable object rearranging tasks. To tackle the challenge of complex configuration space and dynamics, we represent the configuration space of a deformable object with a graph structure and the graph features are encoded by a graph convolution network. Our framework adopts an architecture based on Fully Convolutional Network (FCN) to output pixel-wise pick-and-place actions from only visual input. Extensive experiments have been conducted to validate the effectiveness of the graph representation of deformable object configuration. The experimental results also demonstrate that our framework is effective and general in handling goal-conditioned deformable object rearranging tasks.**

I. INTRODUCTION

Manipulating deformable objects like towels and USB charging cables is an integral part of everyday life for humans. Although the task of robotic manipulation has been investigated for decades [1] [2] [3], most works are focused on rigid objects.Different from rigid object manipulation, deformable objects pose several new challenges. The first challenge is the high dimensionality of the state space [4], which leads to the increased complexity of the state observation during deformable object manipulation. The second challenge lies in the complex and non-linear dynamics of deformable materials [5], which makes the state of deformable objects hard to predict under forces or actions.

Recently, goal-conditioned deformable object rearranging task has become a research focus, where the robot is required to rearrange a deformable object from an initial configuration to a goal configuration. There are two main solutions for such tasks:

The first solution is to establish a data-driven dynamic model [6] which can predict the resulting object states under certain manipulation actions. The robot can then determine a proper manipulation action corresponding to the current and goal configurations using sampling-based methods with the dynamic model. However, if the initial and goal configurations of the deformable object differ considerably, a sequence of manipulation actions, rather than a single one, is required for completing the rearranging task. To tackle the multi-step manipulation planning problem, generating and introducing additional sub-goal configurations for the rearranging task has been proved to be an effective strategy [7] [8]. However, this line of work can still result in extra effort and introduce accumulative errors.

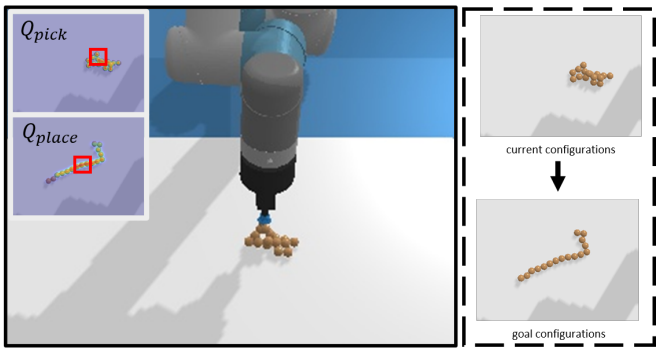


Fig. 1: We propose a novel general framework (Graph-Transporter) to output pixel-wise pick-and-place actions from only visual input for goal-conditioned deformable object rearranging tasks. Our framework can learn multi-task policies at the same time.

TThe second solution is to use imitation or reinforcement learning methods to map the raw observations directly to a policy for a specific task [9] [10] without modeling the object dynamics explicitly. However, this line of work tends to generalize only to a specific set of deformable object manipulation tasks, such as folding cloth, untangling rope-knot [11], hanging a small towel on a hanger, and relocating cloth and rope in a scene with obstacles [12]. Seita et al. [13] proposed a goal-conditioned transporter network, which can be a general learning framework for several goal-conditioned deformable objects rearranging tasks. However, the transport network needs to be retrained for each specific deformable object rearrangement task, which is time-consuming and inconvenient. In contrast, We aim at a general framework that can learn a multi-task policy from demonstrations of different deformable object rearranging tasks together.

To tackle this issue, we first introduce a graph structure to represent the configuration of the deformable object. Compared with the previous CNN-oriented image feature, the graph feature is better at handling the sparse information of the deformable object. It improves the model efficiency and makes it possible to learn policies from demonstration data of a variety of different tasks. We use an unsupervised learning method to detect keypoints from visual input. A representation graph containing vertex and edge features is then established from the keypoints. And we use the representation graph to generate a state vector $\mathbf{V}_G$ by GCN [14] and a mask $\mathbf{M}$ related to the position of keypoints. We proposed a novel

1 Tencent Robotics X Lab, Shenzhen, China. {francisdeng, lipengchen}@tencent.com

2 The Center for Intelligent Control and Telescience, Tsinghua Shenzhen International Graduate School, Shenzhen, China {xiachongkun, wang.xq}@sz.tsinghua.edu.cn

† Corresponding author

framework, Graph-Transporter, to output pixel-wise actions from only visual observations, where the model learns to process the graph feature from manipulation policy learning. Similar to the transporter network [13], [15], the output of our framework is the Q-value heatmap and each Q value represents the action confidence at the corresponding pixel position. The mask $\mathbf{M}$ is used as the mask and the difference of $\mathbf{V}_{\mathrm{G}}$ of current and goal configurations are used as the convolution kernel in the action Q-value heatmap generation. Our framework is trained in an imitation learning fashion.

We have established a dataset and quantitative metrics in simulation and conducted simulation experiments of multiple deformable objects rearranging tasks. The results demonstrate the proposed graph representation method and our framework are effective and general for goal-conditioned deformable object rearranging tasks. The contributions of this paper can be summarized as follows:

- We propose a method to represent the configurations of deformable objects with graph structure and the model learns to process the graph in the manipulation policy learning.
- We propose an effective and general learning framework that utilizes graph features and graph convolution for goal-conditioned deformable object rearranging tasks.
- We build a dataset and quantitative metrics to evaluate the effectiveness of our framework.

The paper is organized as follows. The related work is reviewed in Sec. II. The problem is formulated in Sec. III. Sec. IV establishes our dataset and the proposed framework (Graph-Transporter) is introduced in Sec. V. Experimental results are presented in Sec. VI.

II. RELATED WORK

A. Multi-Task Policy of Deformable Object Manipulation

The application of deformable object manipulation ranges from manufacturing to the service industry [16] [17] [18]. Recently, learning and data-driven approaches have paved the way toward equipping robotics with more advanced capabilities in deformable object manipulation [19]. Designing a learning policy for a specific deformable object manipulation task has been widely investigated. However, the generalization of these algorithms is limited to specific tasks. Learning multi-task policies for deformable object manipulation has achieved some progress recently. Zeng et al. [15] proposed the transporter network to infer a manipulation sequence from only visual input, which can induce a spatial displacement corresponding to the visual input. Their transporter architecture performs well on several rearranging tasks of rigid objects and Seita et al. [13] further improved the network and applied it to additional tasks of deformable objects. Shridhar et al. [20] proposed CLIPort that can produce multi-task policies conditioned on natural language. However, these works must be trained on demonstration data of a specific task to learn a corresponding manipulation policy. We notice that manipulation policies on different rearranging tasks enjoy great similarities. In our framework, the model can learn more general and effective policies that can be used on multiple different deformable object rearranging tasks and the model is trained on the dataset of multiple tasks.

B. State Representation of Deformable Objects

Considering the high dimensionality of the configuration space of deformable objects, an effective representation method is necessary. Most representation strategies rely on keypoint detection. Early works focus on tracking the keypoints of deformable objects [21] [22]. Miller et al. [23] introduced predefined geometric constraints in the process of detecting feature points to improve detection performance. Some solutions also use CNN feature directly to represent deformable objects, which can result in information redundancy because there are typically no deformable objects in most areas of the image. Ma et al. [5] use graph to represent the deformable object and improve modeling accuracy in data-driven dynamic modeling. Since understanding the geometric structure of a deformable object is useful, we group keypoints into a graph structure rather than using them directly, which provides rich semantics including keypoint positions and the topological relation between keypoints. The graph structure is more effective in representing the state of a deformable object by using vertex and edge features. In our model design, the processing of the graph (graph convolution network) is learning from end-to-end manipulation policy learning instead of dynamic modeling separated from manipulation tasks, which can exploit the potential of graphs in manipulation tasks more fully.

III. PROBLEM FORMULATION

Given a current image $\boldsymbol{I}_t$ and a goal image $\boldsymbol{I}_{\mathrm{g}}$, where t denotes the time instant, the task of rearranging a goal-conditioned deformable object is to obtain a manipulation action

$$\boldsymbol{a}_t = \mathcal{F}(\boldsymbol{I}_t, \boldsymbol{I}_{\mathrm{g}}),$$

where $\mathcal{F}$ is the manipulation model that should be developed. This forces the visual image of the deformable object to become as

$$\boldsymbol{I}_{t+1} = \mathcal{D}(\boldsymbol{I}_t, \boldsymbol{a}_t).$$

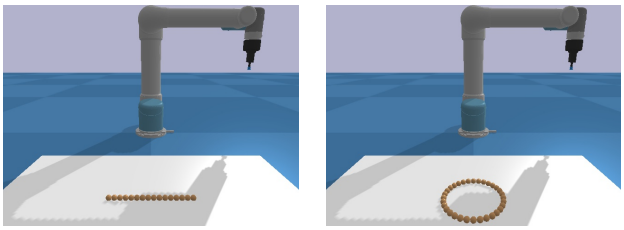
where $\mathcal{D}$ denotes the dynamics of the deformable object.

The above procedure should be repeated until the distance between $\boldsymbol{I}_t$ and $\boldsymbol{I}_{\mathrm{g}}$ in the latent space is less than a certain threshold α.

In this work, our goal is to design an end-end manipulation model $\mathcal{F}$ that can output pixel-wise pick-and-place manipulation actions from only visual input. The challenge of this task mainly lies in two aspects. One is that the target state is not a specific form (a specific shape or a specific location) but a variety of different random forms, which indicates that our model needs to learn more efficient manipulation policies for general rearranging tasks rather than specific policies for a certain rearrangement task. The other is that the task is very complex that requires the model to generate manipulation sequences but without generating any sub-goal images.

IV. DATASET

As our goal is to obtain a framework that can learn a general manipulation policy for multiple goal-conditioned deformable object rearranging tasks, we modify and optimize the simulation environment in [15] and come up with our own set of data, which is more convenient for evaluating the generalization and effectiveness of our framework. The dataset



(a) Rope (b) Rope ring

Fig. 2: Simulation Environment in PyBullet [24]: two kinds of deformable objects with different topologies, rope and rope ring, are generated.

is composed of 1000 deformable object rearranging tasks, while each requires a random number of manipulation actions. In addition, two kinds of deformable objects with different typologies, rope (Fig. 2(a)) and rope ring (Fig. 2(b)), are involved in our task. The established dataset is designed with the explicit goal of training and testing a manipulation policy to rearrange the deformable objects to goal configurations.

We propose a quantitative definition of the success of a goal-conditioned rearrangement task, which makes our evaluation more informed. The definition is detailed in Sec. VI-B.

A. Simulation Environment

Real robotic experiments are usually subjected to a restricted experimental environment and thus not scalable. Safety and cost also limit the number of robot datasets collected in the real environment, and the training of robotic algorithms often requires thousands of iterations. Therefore, we resort to the PyBullet robotic engine [24], which can provide satisfying realistic visual renderings and the requirement of deformable object simulation to generate our large-scale demonstration dataset.

For each demonstration trial, one selected deformable object is placed on the platform, and a UR5 robotic manipulator with an end-effector is placed in front of the platform for object manipulation (Fig.2). Images are captured with a RGB-D camera, which is fixed on the top of the platform.

B. Dataset Generation

We have generated 1000 deformable objects rearranging tasks. For each task, the initial and goal configurations are given as two random images. The goal of the task is to generate a sequence of manipulation actions to rearrange the deformable object to the goal configuration image from the initial image. As shown in Fig. 3, The dataset generation is completed during rearranging the deformable objects with manipulation actions produced by an oracle agent (the design of the oracle agent is briefed in Sec. V-D). We record and save the demonstration data during the rearranging process. At each time instant t, the structure of the record data is $\boldsymbol{I}_t, \boldsymbol{I}_g, \boldsymbol{a}_t\}$, where $\boldsymbol{I}_t$ is the current image at time t, $\boldsymbol{I}_g$ is the goal image and $\boldsymbol{a}_t$ is the manipulation action that the robot should take according to $\boldsymbol{I}_t$ and $\boldsymbol{I}_g$. The demonstration data will be collected at every time instant until the task is finished.

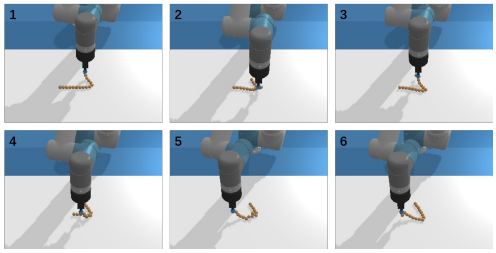


Fig. 3: The dataset generation is completed during rearranging the deformable objects guided by an oracle agent (Sec. V-D).

TABLE I: Dataset split

	rope task	rope-ring task
train	400	400
val	50	50
test	50	50

C. Dataset Analysis

The constructed dataset is composed of 1000 rearranging tasks, of which the task for rope and task for rope-ring manipulation each account for half. The training set accounts for 80% of the total data, and the test and eval set accounts for each 10% of the total data. The spilt of the dataset is shown in TABLE. I. It is worth noting that our framework can handle multiple goal-conditioned rearranging tasks of rope or rope-ring at the same time without requiring separate training for each specific task.

We analyze the distribution of the length of the generated manipulation sequence of all 1000 tasks, which reflects the difficult distribution of tasks in our dataset (the sequence length is equal to the number of manipulation actions required for the corresponding rearranging task). As shown in Fig. 4, the sequence length is mainly distributed from 4 to 19, which indicates that our framework must be able to deal with the planning of long sequences and short sequences at the same time. In addition, many tasks require more than 15 manipulation actions to complete, which also illustrates the complexity of the tasks in our dataset.

V. METHODOLOGY

A. System Overview

As shown in Fig. 5, we introduce Graph-Transporter that utilizes graph features and graph convolution to address the problem of goal-conditioned deformable object rearranging. We use the pick-and-place actions as the manipulation primitives to complete the task for portability.

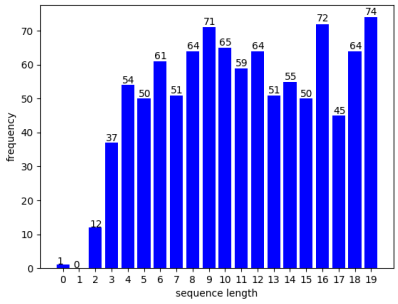


Fig. 4: The distribution of sequence length in our dataset.

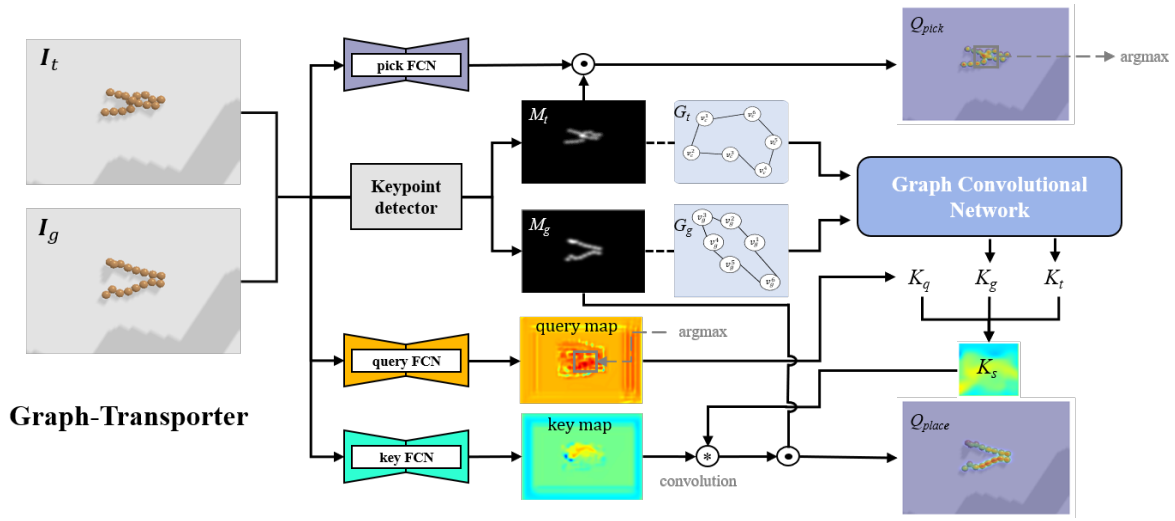


Fig. 5: The architecture of Graph-Transporter

First, we use an unsupervised learning method to extract the keypoints of the deformable object and then construct a graph $G = (V, E)$ to represent the configurations of the deformable object. Then we encode the graph feature to be a state vector by a GCN model. A Gaussian mask related to the positions of keypoints is also generated at the same time. Finally, the graph state vector, Gaussian mask, and RGB images of the current and goal configurations are fed into a FCN-based architecture to generate two Q-value heatmaps for picking and placing. It is worth noting that both Gaussian masks and graph state vectors are generated from RGB images, i.e. our model is an end-to-end formulation from RGB images to manipulation actions.

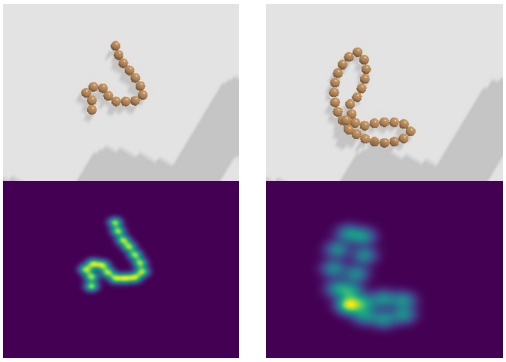
B. Unsupervised Graph Feature Learning

The motivation of Graph-Transporter is to use the graph structure to represent the deformable object configuration with a large degree of freedom. Considering that a lot of image sequences can be obtained from our dataset, we use self-supervised learning methods commonly used in image sequences (video) to detect keypoints. We borrow the architecture from [25] to design our keypoint detector.

The key idea of the keypoint detector is that the information of keypoints should be sufficient for the image reconstruction. The model consists of three parts: an encoder f_e to extracting the feature $f_e(\boldsymbol{I})$, a point detector f_{kp} to detect keypoints $f_{kp}(\boldsymbol{I})$ and generate a Gaussian heatmap $\mathcal{H}_{f_{kp}(I)}$ related to keypoints coordinates, and a decoder f_d to reconstruct the image from the extracting feature and Gaussian heatmap. The source image $\boldsymbol{I}_{src}$ and target image $\boldsymbol{I}_{tgt}$ (two adjacent images in each demonstration sequence in our dataset) are passed through f_e and f_{kp} to get features and Gaussian heatmap, and then the target image will be reconstructed by the decoder f_d. The loss during training is designed as the mean square error of image reconstruction:

$$loss = \left\| f_d\left(f_e\left(\boldsymbol{I}_{src}\right), f_e\left(\boldsymbol{I}_{tgt}\right), \mathcal{H}_{f_{kp}(\boldsymbol{I}_{src})}, \mathcal{H}_{f_{kp}(\boldsymbol{I}_{tgt})}\right), \boldsymbol{I}_{tgt} \right\| \tag{1}$$

We set the number of keypoints as sixteen, as sixteen keypoints are sufficient to represent a deformable object in our task and can ensure that the noises in the image will not be recognized as keypoints. The example results of keypoints detection are shown in Fig. 6. The Gaussian mask with fixed-variance Gaussian distribution around each keypoint will be also used in action generation.



(a) Gaussian mask of a rope (b) Gaussian mask of rope ring

Fig. 6: Result of keypoint detection.

Then we construct a graph $G = (V, E)$ based on the detecting keypoints $f_{kp}(\boldsymbol{I})$. Because the position is the most important feature in rearranging tasks, we encoder the position of each keypoint in $f_{kp}(\boldsymbol{I})$ as the feature vector of the corresponding vertex in the graph to build up V. Then we calculate the adjacency matrix E. Considering that a node on a deformable object tends to have a topological constraint relationship with its two adjacent nodes on the left and right, we think each vertex in the graph to hold an adjacency relation with its two nearest vertexes. The elements related to each vertex and its two nearest vertexes are set to be 1 while the other elements are set to be 0 in the adjacency matrix E.

C. Model Architecture

As shown in Fig. 5, Graph-Transporter uses three parallel Fully Convolution Networks (FCN), one for picking Q-value heatmap generation and the other two for placing Q-value heatmap generation. First, the current image I_t and the goal image I_g are passed through the point detector, which is pretrained for detecting keypoints by unsupervised

learning. Then we can get representation graphs (G_t and G_g) and Gaussian masks (M_t and M_g) related to the detecting keypoints in I_{t} and I_{g}. We design a 2-layers GCN model to encode G_t and G_g and the two encoded graph features are fattened into two feature vectors K_t and K_g. The calculating process of the GCN layer is defined as:

$$h_i^{l+1} = \sigma \sum_{j \in N_i} \frac{1}{c_{ij}} h_j^l w^l \tag{2}$$

where h_i^{l+1} represent the features of node i in layer $l+1$, N_i represent all neighbor nodes of i in the graph, c_{ij} is the normalization factor calculated from the adjacency matrix of graph, w^l is the weight of layer l, and σ represent the activation function.

At the same time, the current image I_{t} and the goal image I_{g} are concatenated together as input, and then passed through three FCNs in parallel. For the picking point, we multiply the output of the pick FCN by current mask M_t to generate a picking Q-value heatmap Q_{pick}. For the placing point, we borrowed the attention mechanism, the output of query FCN is treated as a query map while the output of key FCN is treated as a key map. After getting the Q_{pick}, we can get the picking position. Then we crop an area K_q near the picking position on the query map. A cross-convolution operation is applied on the key map by using the convolution kernel $K_s = K_q + K_g - K_t$. Finally, we multiply the result of cross-convolution by goal mask M_g to generate placing Q-value heatmap Q_{place}.

Algorithm 1 imitation policy for rearranging task

unit number = n
index = [0,1,..,n-1]
bestmap = [0,1,..,n-1]
min = ∞
max = 0
pick = 0
place = 0
for i in index **do**
  remap = [i,i+1,..,n-1,0,1,..i-1]
  dis = MSE(P_c[remap],P_g)
  if dis<min **then**
    min = dis
    bestmap= remap
  end if
end for
for j in index **do**
  dis1 = MSE(P_c[bestmap][j],P_g[j])
  if dis1>max **then**
    max = dis1
    pick= j
    place= j
  end if
end for
p_{pick} = P_c[bestmap][pick]
p_{place} = P_g[place]
return p_{pick}, p_{place}

D. Imitation Learning

Since we have complete information on the configurations of the deformable object in simulation, we can obtain the best action behavior for the robot to rearrange the deformable object. We adopt an imitation learning methodology to train the robot to mimic the best action behavior.

The simulation of deformable objects in PyBullet is realized by discretizing the deformable objects into some small rigid object units that are constrained by each other. We can get the position of every unit during simulation. These position data can be the guidance for the oracle agent to output manipulation action. Denote P_c and P_g are the current and the goal positions of every unit respectively. The generation process of the imitation action $[p_{pick}, p_{place}]$ (p_{pick} is the picking position, p_{place} is the placing position) for a rope-ring rearranging task is shown in Alg. 1. First, the algorithm needs to find the node correspondence that minimizes the distance between P_c and P_g, which can ensure rotational consistency. After that, we need to find which two corresponding nodes have the largest distance under this node correspondence relationship, and the positions of these two nodes are the pick position and place position. The generation process of imitation action for the rope rearranging task is similar, the only difference is that there are only two types of node corresponding relationships.

VI. EXPERIMENTS

This section presents experiments to show the performance of our method.

A. Ablation Studies on Graph Feature

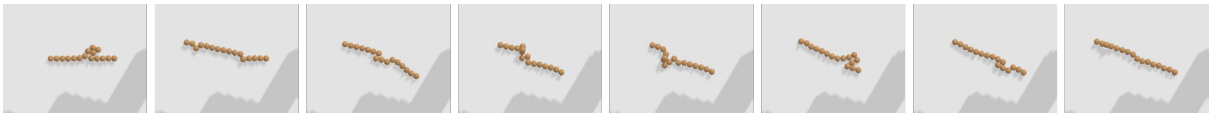
To verify the effect of introducing graph features, we conduct experiments to compare the performance of our model (with graph features) and the goal-conditioned transporter [13] (without graph features) on the test data.

The imitated error is the most significant performance metric in imitation learning. Considering that the imitated action is a pixel-wise pick and place action, we use the pixel distance between the picking and placing positions output by the model and the picking and placing positions output by the oracle agent to be the metric to evaluate our model. We also divide the pixel distance by a factor of the image size for normalization. The definition is shown as follows:

$$e_{pick} = \frac{\|p_{pick} - o_{pick}\|}{\sqrt{w^2 + h^2}} \tag{3}$$

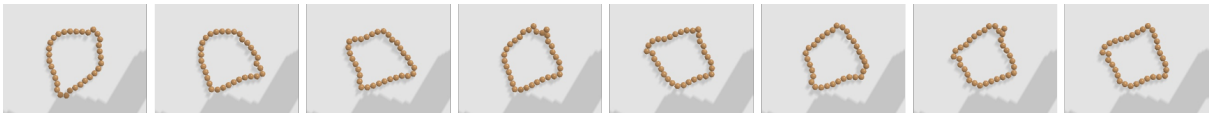
$$e_{place} = \frac{\|p_{place} - o_{place}\|}{\sqrt{w^2 + h^2}} \tag{4}$$

where p_{place}, p_{pick} are the output actions, o_{place}. o_{pick} are the imitated actions. w and h are the width and height of the image respectively. It should be noticed that we adjust the positions of the camera and the UR5 manipulator to ensure that the camera is directly above the plane where the deformable object is located. So we do not use the visual representation from multi-view image synthesis as used in [15], but directly use the raw image from the camera (the deformable object is not occluded in the image) in our experimental setup. This setting does not affect the performance of the model but makes experiments faster.



(a) The robot needs to rearrange the shrunk rope into a straight line, whose direction and position are given by the goal image.

(b) The robot needs to rearrange the rope to a "V" shape, where the angle and direction are given by the goal image.



(c) The robot needs to rearrange the rope ring into a square shape, where the right angle is the challenge.

Fig. 7: The experimental results show that our framework can handle multiple deformable object rearranging task without training on each specific task.

TABLE II: Comparison of two models on two types of deformable object manipulation tasks.

imitation error	rope		rope ring	
	e_{pick}	e_{place}	e_{pick}	e_{place}
Graph-Transporter	0.045	0.031	0.068	0.020
Transporter without graph	0.051	0.036	0.085	0.038

The result is shown in TABLE. II. In both tasks, the performance of Graph-Transporter is better than Transporter without graph. The reason is that our graph structure well represents the configurations of the deformable object, which helps the robot learn the rearranging actions better. The mask obtained from the graph structure eliminates invalid regions in the image and speeds up the convergence of the model. At the same time, the convolution kernel constructed by the feature vector obtained by encoding the graph structure brings information about the configuration change of the deformable object for action generation.

B. Evaluation Experiment

We also conduct experiments to evaluate the performance of our framework on rearranging tasks. The robot is given random goal configurations by only visual input, and the robot needs to rearrange the deformable object to the goal configurations without any sub-goal input. We define that the robot completes a rearranging task within 20 manipulation actions as a success and the rest as a failure. The situation of completing a rearranging task is that the average pixel distance between the corresponding units in the current and goal configurations is less than 10 (We have found that this threshold is reasonable through most experiments. The two configurations that satisfy this situation are similar enough in the experiment). The success rate of rearranging rope reaches 73% in 100 testing tasks and the success rate of rearranging the rope ring reaches 68% in 100 testing tasks. Considering that our task is a highly generalized task, this result can illustrate the effectiveness of our framework. The key to the performance of our architecture is to introduce graph features that can better represent the sparse information of deformable objects in the image in the process of Q-value heatmap generation so that the model can learn more general rearranging policies without expanding the size.

(a) self-occlusion rope (b) self-occlusion rope ring

Fig. 8: Failure case.

Three examples of the experiment are shown in Fig. 7. The first task is straightening the crumpled rope. The second task is rearranging the rope into a V-shape. The third task is rearranging the rope ring into a rectangle. All of these tasks are successfully completed in 20 manipulations.

C. Failure Case Analysis

We have found that the performance of our model will be affected when the deformable object has self-occlusion. The failure case is shown in Fig. 8. It can be difficult to detect effective keypoints when the object has a large degree of self-occlusion, which will make the representation of the graph feature incomplete. At the same time, it will also be hard to differentiate which part of the deformable object is on the top or on the bottom, which will influence the graph construction and action performance (picking and placing points may be unreachable). We aim at introducing the 3D point cloud to replace the RGB image in future work, where the adjacency relationship in the representation graph will be calculated based on the distance in 3D space instead of pixel distance in the image plane. A 3D graph feature has the potential to fully represent the configurations of a deformable object.

VII. CONCLUSION

In this paper, we propose a novel framework, Graph-Transporter, to solve the task of goal-conditioned deformable object rearranging. The configurations of the deformable object are represented as a graph in Graph-Transporter. The graph feature is used in a FCN-based architecture, which can generate pixel-wise pick-and-place actions with only visual input. Extensive experiments have been conducted demonstrating the effectiveness of representing the deformable object with graph structure and the rationality of our framework design for deformable object rearranging tasks. In future work, we will extend our framework to more complex deformable objects such as cloth and rubber.

REFERENCES

[1] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[2] C. Wang, S. Wang, B. Romero, F. Veiga, and E. Adelson, "Swingbot: Learning physical features from in-hand tactile exploration for dynamic swing-up manipulation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5633–5640.

[3] 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.

[4] H. Yin, A. Varava, and D. Kragic, "Modeling, learning, perception, and control methods for deformable object manipulation," *Science Robotics*, vol. 6, no. 54, p. eabd8803, 2021.

[5] X. Ma, D. Hsu, and W. S. Lee, "Learning latent graph dynamics for deformable object manipulation," *arXiv preprint arXiv:2104.12149*, 2021.

[6] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *arXiv preprint arXiv:2003.05436*, 2020.

[7] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, "Combining self-supervised learning and imitation for vision-based rope manipulation," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2146–2153.

[8] A. Wang, T. Kurutach, K. Liu, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," *arXiv preprint arXiv:1905.04411*, 2019.

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

[10] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to manipulate deformable objects without demonstrations," *arXiv preprint arXiv:1910.13439*, 2019.

[11] J. Grannen, P. Sundaresan, B. Thananjeyan, J. Ichnowski, A. Balakrishna, M. Hwang, V. Viswanath, M. Laskey, J. E. Gonzalez, and K. Goldberg, "Untangling dense knots by learning task-relevant keypoints," *arXiv preprint arXiv:2011.04999*, 2020.

[12] D. McConachie, A. Dobson, M. Ruan, and D. Berenson, "Manipulating deformable objects by interleaving prediction, planning, and control," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 957–982, 2020.

[13] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng, "Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks," *arXiv preprint arXiv:2012.03385*, 2020.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in neural information processing systems*, vol. 29, pp. 3844–3852, 2016.

[15] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," *arXiv preprint arXiv:2010.14406*, 2020.

[16] P. Jiménez, "Survey on model-based manipulation planning of deformable objects," *Robotics and computer-integrated manufacturing*, vol. 28, no. 2, pp. 154–163, 2012.

[17] I. Leizea, A. Mendizabal, H. Alvarez, I. Aguinaga, D. Borro, and E. Sanchez, "Real-time visual tracking of deformable objects in robot-assisted surgery," *IEEE computer graphics and applications*, vol. 37, no. 1, pp. 56–68, 2015.

[18] A. Kapusta, Z. Erickson, H. M. Clever, W. Yu, C. K. Liu, G. Turk, and C. C. Kemp, "Personalized collaborative plans for robot-assisted dressing via optimization and simulation," *Autonomous Robots*, vol. 43, no. 8, pp. 2183–2207, 2019.

[19] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, 2019.

[20] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 894–906.

[21] T. Tang, Y. Fan, H.-C. Lin, and M. Tomizuka, "State estimation for deformable objects by point registration and dynamic simulation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2427–2433.

[22] T. Tang and M. Tomizuka, "Track deformable objects from point clouds with structure preserved registration," *The International Journal of Robotics Research*, p. 0278364919841431, 2018.

[23] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 249–267, 2012.

[24] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016," *URL http://pybullet. org*, 2016.

[25] T. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih, "Unsupervised learning of object keypoints for perception and control," *Advances in neural information processing systems*, vol. 32, pp. 10 724–10 734, 2019.