Supplementary Materials

SAGCI-System: Towards Sample-Efficient, Generalizable, Compositional, and Incremental Robot Learning

Jun Lv*[1], Qiaojun Yu*[1], Lin Shao*[2], Wenhai Liu[3], Wenqiang Xu[1] and Cewu Lu[1]

With an RGB-D camera mounted at the wrist, the robot receives point clouds at each time step. Our pipeline takes the raw point clouds as inputs, outputs a URDF to initially represent the surrounding environment. To achieve this, we first segment the raw point cloud into several parts, generate meshes and physical properties of each part. Here each part would correspond to a link in the URDF. Then we need to estimate the joint relationship between parts to produce a URDF file. (See Fig. 1)

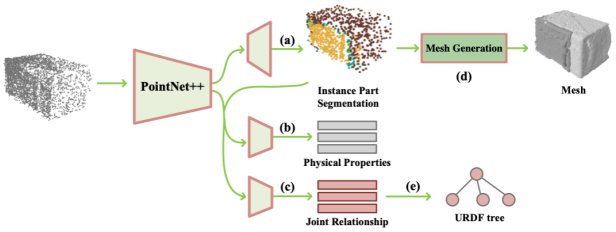


Fig. 1: To accomplish initial modeling of the environment, there are: (a) Part-level instance segmentation; (b) Mesh generation; (c) Physical property estimation; (d) Joint relation estimation; (e) URDF tree generation.

We use URDF models from SAPIEN dataset [1] and render 100 random views for each URDF model to generate a dataset. How to generate ground-truth data for each task will be introduced in the following paragraph.

a) Part-level instance segmentation model: We develop a part-level instance segmentation model similar to [2], which proposes a network to segment a finite, but unknown number of moving objects. We use the same computing paradigm as [2] to achieve instance segmentation, but only take static point cloud as input. For more details, we refer to

*The authors contributed equally

[1]Jun Lv, Qiaojun Yu, Wenqiang Xu, Cewu Lu are with Department of Computer Science, Shanghai Jiao Tong University, China. {lyujune_sjtu,yqjllxs,vinjohn,lucewu}@sjtu.edu.cn

[2] Lin Shao is with Artificial Intelligence Lab, Stanford University, USA. lins2@stanford.edu

[3]Wenhai Liu is with School of Mechanical Engineering, Shanghai Jiao Tong University, China. sjtu-wenhai@sjtu.edu.cn

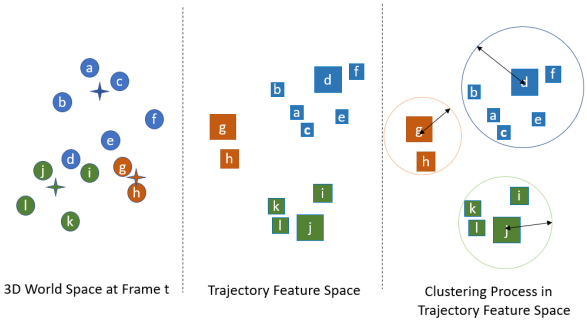


Fig. 2: A figure of the segmentation process adopted from [2]. Left: Points represent points in a point cloud. Stars represent ground-truth object centers. Same color indicates same object. Middle: Each square represents the features in feature space each associated with a point on the left. The size of the squares represents the corresponding point's probability of being an object centroid. Right: The segmentation process cycles through the squares starting with those having the highest probability to be an object centroid. A sphere centered at one of those squares with radius $\hat{\mathcal{B}}$ then segments trajectories and corresponding points.

[2]. The model takes in the point clouds $\{\mathcal{P}_t^i\}_{i=1}^N \in \mathcal{R}^{N\times 3}$, and outputs a part-level instance segmentation mask denoted as $\{\mathcal{M}_t^i\}_{i=1}^N$ where $\mathcal{M}_t^i \in \{1, 2, ..., K\}$. K is the number of parts. We then segment the raw point cloud $\{\mathcal{P}_t^i\}_{i=1}^N$ into a set of groups denoted as $\{\mathcal{G}_t^j\}_{j=1}^K$. A visualization of the process is shown in Fig.2. We discuss the detailed processes.

Here we adopt the same notations used in [2]. Given the received point cloud $\{\mathcal{P}_t^i\}_{i=1}^N$, we utilize PointNet++ [3] to regress the centroid $\{\zeta_t^i\}_{i=1}^N \in \mathcal{R}^{N\times 3}$, boundary $\{\mathcal{B}_t^i\}_{i=1}^N \in \mathcal{R}^{N\times 1}$, and confidence score $\{\mathcal{S}_t^i\}_{i=1}^N \in \mathcal{R}^{N\times 1}$ per point, similar to point segmentation. For each point $\mathcal{P}_t^i$ with a mask $M_t^i = k$ indicating which part the point $\mathcal{P}_t^i$ belongs to, centroid ζ_t^i denote the 3D centroid position of the part k in the world coordinate. $\mathcal{B}_t^i$ is a radius estimate of the sphere that encloses all points which belong to the same part. $\mathcal{S}_t^i$ contains the probability that it is nearest to the object centroid.

During training stage, we adapt a pixel-wise loss $\mathcal{L}^{seg}$ as defined in [2], which measures the L2 distance between the predicted $\widehat{\zeta^i}_t$, $\widehat{\mathcal{B}^i}_t$, and $\widehat{\mathcal{S}^i}_t$ and the corresponding ground truth values ζ_t^i, $\mathcal{B}_t^i$, and $\mathcal{S}_t^i$. Ground truth ζ_t^i is directly calculated respect to the ground-truth part segmentation. $\mathcal{B}_t^i$ is half of the distance between the ζ_t^i and the nearest neighbour part centroid. As for the ground truth $\mathcal{S}_t^i$, we sort the points by the distance to the part centroid, the top D

pixels per part will be labeled as 1, while others will be labeled as zero.

While generating the segmenting masks, we adapt the same method as [2]. We use the Figure in [2] to visualize the inference process. Given predicted centroid $\{\widehat{\zeta_t^i}\}_{i=1}^N$, boundary $\{\widehat{\mathcal{B}_t^i}\}_{i=1}^N$, and confidence score $\{\widehat{\mathcal{S}_t^i}\}_{i=1}^N$, we first choose the point with highest confidence score $\widehat{\mathcal{S}_t^k}$, where $k \in \{1, 2, ..., N\}$. Then all points falling inside the sphere centered at $\widehat{\zeta_t^k}$ with the radius $\widehat{\mathcal{B}_t^k}$ are assigned to the same part, which becomes the group $\mathcal{G}_t^j$. Then all points assigned to this part are removed from the set of unsegmented points before segmenting the next group. We keep iterating the process until there are no points that have confident scores higher than a threshold.

After the above process, we have initially segment the raw point clouds into a finite but unknown number of different groups.

b) Mesh Generation: Once finishing the part-level instance segmentation, we need to generate a watertight mesh for each point cloud group in $\{\mathcal{G}_t^j\}$. How to generate meshes from point clouds has a rich literature. Here we adopt one approach called ManifoldPlus [4] and find it works well in our cases. There are other options, but it is not our main focus. We first estimate the normals of raw point clouds within each group $\mathcal{G}_t^j$. For each point in the group $\mathcal{G}_t^j$, we create a small triangle centered at the point with the triangle surface orthogonal to the point's normal. We feed these triangles into the ManifoldPlus [4] algorithm to generate a watertight manifold surface in OBJ format. For more details of Manifoldplus, please refer to [4].

c) Physical property estimation model: Based on the raw input point clouds of each group, our model also estimates the physical proprieties of each link. Once finishing the part-level instance segmentation $\mathcal{G}^j$, we can use max-pooling to extract part-level feature from point-level feature generated by PointNet++. Then we can estimate the physical properties α^{sim} of each link from the part-level feature. A visualization of the network architecture is shown in Fig.1. We apply L2 distance to supervise the physical properties estimation, which is $\mathcal{L}^{phy}$.

To annotate the mass value of objects in the training set, we calculate the volume of the object's link and multiply the volume by the density of the material generating each link's mass value. Estimating the physical properties based on raw point clouds is usually not accurate. These initial estimated values would be modified in the interactive perception stages.

d) Joint estimation model: The mesh file and the corresponding physical properties constitute the descriptions of the **link** element in the URDF. Next we discuss how to estimate the **joint** element of the URDF.

We estimate the joint information between two parts from the part-level feature as shown in Fig.1. The part-level instance segmentation, physical property, and joint estimation models share the PointNet++ weights to extract point-level features.

First, given K links, we would develop a joint relationship model that takes the segmented point clouds of the link u and the link v as inputs, where $u, v \in \{1, 2, ..., K\}$, to estimate their pairwise joint relationship denoted as $\mathcal{J} \in \mathcal{R}^{K \times K \times 4}$ and joint spatial description $\mathcal{C} \in \mathcal{R}^{K \times K \times 9}$. The joint type $\mathcal{J}$ contains four classes, which are "**None**", "**Fixed**", "**Revolute**", "**Prismatic**". Joint spatial description $\mathcal{C}$ have nine dimensions, in which 6 for revolute joint (joint position and orientation) and 3 for prismatic joint's orientation. Note that we already generate the mesh file of each link. We collect each mesh's 3D position in the world coordinate. We then define their local coordinate and their relative transformation. If there is a prismatic joint between two links, we only need to estimate the prismatic movement direction.

For the **joint limit** element of the URDF, we set default values and these values are online verified and modified in the interactive perception stage.

Since there are always errors in part-level instance segmentation results, we would need to online generate the ground truth annotation of the joint description. Given two point clouds sets $\mathcal{G}_t^u$ and $\mathcal{G}_t^v$ in $\{\mathcal{G}_t^j\}_{j=1}^K$ from part-level instance segmentation network, we first find out the corresponding links in the ground-truth URDF. We calculate the IoU between the predicted instance segmentation masks and the ground-truth part instance segmentation masks and select the parts associated with the highest IoU scores. Then we can obtain ground truth joint relationship $\mathcal{J}(u, v)$ and ground truth joint spatial description $\mathcal{C}(u, v)$ between the relative links in the ground-truth URDF file. We use the cross-entropy loss to supervise the joint type classification $\mathcal{L}_{\mathcal{J}}^{joint}$, and an L2 distance to measure the predicted and ground truth joint spatial description $\mathcal{L}_{\mathcal{C}}^{joint}$.

e) URDF generation: After estimating the relationship between each pair in K links, we get a complete directed $K \times K$ graph. We need to find a directed tree to generate a valid URDF file. URDF maintains a directed tree structure to describe a single object, so there are several properties: 1) given K nodes, there is only one node that has no parent, which is root node; 2) except the root node, each node has one and only one parent, which means there are $K - 1$ links on the tree; 3) no cycles are allowed. Considering these properties, we propose a method is similar to Kruskal algorithm [5] to generate URDF. We treat the probability of "**None**" between two links as the edge weight, then find the next lowest-weight edge. The low probability of "**None**" between two links indicates that there is a high probability that there is joint between these two links. If adding this edge to the tree will form no cycles and the child node of this edge still have no parent, the edge will be accepted. Once the edge is accepted, the associated pair of two links will not be selected again. Given K node, once $K-1$ edges are accepted, the algorithm terminate. Then we construct the URDF structure by iterating the joint and the link from the root node to the leaf node.

B. Interactive Perception

Although we have estimated a URDF file based on the raw point clouds, there are always modeling errors during

the above-mentioned method. We propose a pipeline to train the robot to use the Interactive Perception (IP) to verify and modify the URDF. We first discuss what modeling parameters are re-estimated through the IP and then introduce the pipeline of how to update these modeling parameters.

a) Model Parameter: We would update the following model parameters: (1) the joint type $\mathcal{J}$; (2) the joint spatial descriptions $\mathcal{C}$; (3) each link's mask segmentation $\mathcal{M}$ and the corresponding mesh file; (4) the physical attributes α^{sim}. We denote model parameter set as $\mathcal{Z} = \{\mathcal{J}, \mathcal{C}, \mathcal{M}, \alpha^{sim}\}$, and $\mathcal{E}$ to describe the URDF tree structure.

After receiving the raw point cloud $\{\mathcal{P}_t^i\}_{i=1}^N$ at the time step t, our system generates an action denoted as a_t^{IP}. To learn the policy of generating the action a^{IP}, we formulate it as a reinforcement learning problem and discuss the policy network at the end of this subsection. After the action a_t^{IP} are executed, we can observe the difference between the real world and the simulation world. Minimizing their difference would encourage a more accurate modeling parameter $\mathcal{Z}$. To accomplish this, we first need to establish the correspondence between the real world and the simulation.

Note that, we would optimize the link from the root node on URDF tree to the leaf node if more than one joint state has been changed. And the leaf node would only be updated after its parent node is optimized.

b) Correspondence in Simulation: In the simulation, given model parameters $\mathcal{Z}$, $\mathcal{E}$ and action a_t^{IP}, we could directly calculate the new point position of $\{\mathcal{P}_t^i\}_{i=1}^N$ at time $t+1$, denote as $\{\overline{\mathcal{P}}_{t+1}^i\}_{i=1}^N$, via the forward function $\mathcal{F}^{Sim}$ in Eqn. 1.

$$\overline{\mathcal{P}}_{t+1}^i = \mathcal{F}^{Sim}(\mathcal{P}_t^i, \mathcal{Z}, \mathcal{E}, a_t^{IP}) \tag{1}$$

Given model parameter $\mathcal{Z}$, $\mathcal{E}$, and action a_t^{IP}, after the action executed on a specific joint, the joint state would change δq in the simulation through Eqn. 2, like open the microwave by δq degree or open the drawer by δq centimeters. Note that in the simulation, we could directly record and measure these changes.

$$\delta q = \textit{Simulation.step}(\mathcal{J}, \mathcal{C}, \mathcal{M}, \alpha^{sim}, \mathcal{E}, a_t^{IP}) \tag{2}$$

Based on these changes, we calculate the new positions of these point clouds at the next time step denoted as $\overline{\mathcal{P}}_{t+1}^i$ via Eqn. 3 4 5. For the revolute joint, with Rodrigues' rotation formula [6], we can obtain the transformation matrix $\mathbf{T} \in \mathcal{R}^{4\times4}$ from the joint spatial description $\mathcal{C}$ and the joint state change δq. For the prismatic joint, the child link of the joint moves along the joint orientation with δq, which is denoted as $\mathbf{T}$. We also have $\mathcal{M}$ which is the binary mask for link of the relative joint which applied action.

$$\overline{\mathcal{P}}_{t+1}^i = \mathcal{F}^{Sim}(\mathcal{P}_t^i, \mathcal{J}, \mathcal{C}, \mathcal{M}, \alpha^{sim}, \mathcal{E}, a_t^{IP}) \tag{3}$$

$$= \mathcal{F}^{Sim\prime}(\mathcal{P}_t^i, \mathcal{J}, \mathcal{C}, \mathcal{M}, \delta q, \mathcal{E}) \tag{4}$$

$$[\overline{\mathcal{P}}_{t+1}^i \ 1]^T = \begin{cases} \mathcal{P}_t^i, & \mathcal{M}_t^i = 0 \\ \mathbf{T}[\mathcal{P}_t^i \ 1]^T & \mathcal{M}_t^i = 1 \end{cases} \tag{5}$$

c) Correspondence in Real World: Through the RGB-D camera, the robot in the real-world receives a new point cloud at the next time step denoted as $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$. We train a scene flow [7] model that takes raw point clouds $\{\mathcal{P}_t^i\}_{i=1}^N$ and $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$ as inputs, and outputs the scene flow $\{\mathcal{U}_t^i\}_{i=1}^N$. Note that there is rich literature discussing about the scene flow including the learning-based approaches [2, 7] and non learning-based approaches [8, 9] with different input modality. Calcuating the scene flow is not our focus in this work. After receiving the scene flow $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$, we then calculate the predicted positions of the raw point clouds at the next time step, denoted as $\{\mathcal{P}_t^i + \mathcal{U}_t^i\}_{i=1}^N$. Each point $\mathcal{P}_t^i + \mathcal{U}_t^i$ searches for the nearest point in $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$. In practice, if the distance between a point $\mathcal{P}_t^i + \mathcal{U}_t^i$ and its corresponding nearest point is larger than a given threshold, we would not use the point $\mathcal{P}_t^i + \mathcal{U}_t^i$. We denote these found points in $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$ as $\{\widetilde{\mathcal{P}}_{t+1}^i\}_{i=1}^N$. $\{\widetilde{\mathcal{P}}_{t+1}^i\}_{i=1}^N$ is point-wise correspondence to $\{\mathcal{P}_{t+1}^i\}_{i=1}^N$. We denote the point-wise real world forward function $\mathcal{F}^{real}$ as follows.

$$\widetilde{\mathcal{P}}_{t+1}^i = \mathcal{F}^{real}(\mathcal{P}_t^i, \mathcal{U}_t^i, \{\mathcal{P}_{t+1}^i\}_{i=1}^N) \tag{6}$$

We adapt the same network architecture and training scheme as [7]. We generate the training data on synthetic URDF models. Given ground truth joint relationship $\mathcal{J}$, joint spatial description $\mathcal{C}$, part-level instance segmentation $\mathcal{M}$, URDF tree $\mathcal{E}$ and a random joint state change δq, ground truth scene flow annotation can be generated via function $\mathcal{F}^{sim\prime}$ in Eqn. 4.

d) Model parameter optimization: Up to now, we can compute point-wise estimation of $\mathcal{P}_t^i$'s position at the next time step in both simulation and real world by $\mathcal{F}^{sim}$ and $\mathcal{F}^{real}$, which denotes as $\widetilde{\mathcal{P}}_{t+1}^i$ and $\overline{\mathcal{P}}_{t+1}^i$ respectively. A accurate model parameter leads to a small difference of them. We denoted the distance between $\{\widetilde{\mathcal{P}}_{t+1}^i\}_{i=1}^N$ and $\{\overline{\mathcal{P}}_{t+1}^i\}_{i=1}^N$ as the $\mathcal{L}_{t+1}$ defined in Eqn. 7.

$$\mathcal{L}_{t+1}(a_t^{IP}, \mathcal{Z}, \mathcal{E}) = \frac{1}{N}\sum_{i=1}^{N} \|\widetilde{\mathcal{P}}_{t+1}^i - \overline{\mathcal{P}}_{t+1}^i\|^2 \tag{7}$$

To simplify the optimization, we decouple the optimization process to two stages. First, we optimize the joint type $\mathcal{J}$, joint spatial description $\mathcal{C}$, part segmentation $\mathcal{M}$, and joint state change δq. Second we optimize the physical properties α^{sim}.

We first optimize $\mathcal{J}, \mathcal{C}, \mathcal{M}$ and δq respect to $\mathcal{L}^{tran}$ defined in Eqn. 8.

$$\mathcal{L}_{t+1}^{tran}(\mathcal{J}, \mathcal{C}, \mathcal{M}, \delta q, \mathcal{E}) = \frac{1}{N}\sum_{i=1}^{N} \|\widetilde{\mathcal{P}}_{t+1}^i - \overline{\mathcal{P}}_{t+1}^i\|^2 \tag{8}$$

Since we consider one joint at a time, only the $\mathcal{J}$, $\mathcal{C}$, $\mathcal{M}$ and δq of the one relative joint and its child link in $\mathcal{E}$ would be optimized during this interactive perception step. Leveraging the differentiability through Eqn. 9, we could have the optimized $\mathcal{J}'$, $\mathcal{C}'$, $\mathcal{M}'$, and $\delta q'$ respect to the gradient of $\mathcal{F}^{sim\prime}$ in Eqn. 4. $\mathcal{J}'$, $\mathcal{C}'$, $\mathcal{M}'$, and $\delta q'$ should

better fit the new point clouds in the real world after a_t^{IP} is executed.

$$\begin{aligned}\mathcal{J}' &= \mathcal{J} - \lambda_{\mathcal{J}} \frac{\partial \mathcal{L}_{t+1}^{tran}}{\partial \mathcal{J}} \\ \mathcal{C}' &= \mathcal{C} - \lambda_{\mathcal{C}} \frac{\partial \mathcal{L}_{t+1}^{tran}}{\partial \mathcal{C}} \\ \mathcal{M}' &= \mathcal{M} - \lambda_{\mathcal{M}} \frac{\partial \mathcal{L}_{t+1}^{tran}}{\partial \mathcal{M}} \\ \delta q' &= \delta q - \lambda_{\delta q} \frac{\partial \mathcal{L}_{t+1}^{tran}}{\partial \delta q}\end{aligned} \tag{9}$$

Now, we will introduce how to optimize the physical properties α^{sim}. Note that $\mathcal{J}'$, $\mathcal{C}'$, $\mathcal{M}'$, and $\delta q'$ results in a smaller difference between simulation and the real world. Given the optimized $\mathcal{J}'$, $\mathcal{C}'$, $\mathcal{M}'$ and the old α^{sim}, we can obtain $\delta q''$ through the nimble simulation in Eqn. 10.

$$\delta q'' = \textit{Simulation.step}(\mathcal{J}', \mathcal{C}', \mathcal{M}', \alpha^{sim}\mathcal{E}, a_t^{IP}) \tag{10}$$

The optimized $\delta q'$ may be different from $\delta q''$ due to the imperfect α^{sim}. We have $\mathcal{L}_{t+1}^{phy}$,

$$\mathcal{L}_{t+1}^{phy} = \|\delta q'' - \delta q'\|^2 \tag{11}$$

Thanks to the differentiablity of nimble simulation, we can also optimize α^{sim} through the gradient of $\mathcal{L}_{t+1}^{phy}$ by,

$$\alpha^{sim\prime} = \alpha^{sim} - \lambda_{\alpha^{sim}} \frac{\partial \mathcal{L}_{t+1}^{phy}}{\delta q''} \frac{\delta q''}{\partial \alpha^{sim}} \tag{12}$$

Till now, we have optimized the modeling parameter $\mathcal{Z}$ to $\mathcal{Z}' = \{\mathcal{J}', \mathcal{C}', \mathcal{M}', \alpha^{sim\prime}\}$ after one interactive perception step. After that, we need to modify the URDF file.

After one interactive perception step and optimization, if the function value $\mathcal{L}_{t+1}$ is lower than a threshold after optimization, $\mathcal{Z}'$ is accepted to modify URDF file. The joint type and joint spatial description would be directly changed by $\mathcal{J}'$ and $\mathcal{C}'$. Also the optimized $\mathcal{M}'$ of the link would be accepted, but it may cause other link in URDF vanishes due to the over segmentation of part-level instance segmentation in initial modeling stage. Some areas are eliminated from the initial segmentation, which are points belong to $\mathcal{M}$ (the segmentation mask before the optimization) but not belong to $\mathcal{M}'$ (the segmentation mask after the optimization)

$$\mathcal{M}^* = (\neg \mathcal{M}') \cap \mathcal{M} \tag{13}$$

if $\mathcal{M}^*$ is larger than a threshold, it would be separated into a new link in URDF and be verified and modified in the future interactive perception step.

e) Policy network: We develop a deep reinforcement learning algorithm which tasks as inputs the raw point cloud $\{\mathcal{P}_t^i\}_{i=1}^N$ and model parameter set $\mathcal{Z}$, and outputs an action denoted as a_t^{IP}. Here the a_t^{IP} contains a discrete action which is the joint id on the generated URDF, and a continuous action that determines the goal state change of the corresponding joint value. One challenge is that the action space in our setting is hybrid. We develop an algorithm to tackle the hybrid action space. Note that the action space also contains the fixed joint type and random pushing motion to encourage exploration. So it's acceptable that the interactive perception policy network outputs an action of changing a fixed joint by certain degree. Our system would ask the robot to push the corresponding child link by certain forces. How to generate the associated robot manipulation actions will be discussed in III-D.

Because the number of joints is discrete and the joint's angle/value is continuous, we combine the DQN [10] and DDPG [11] algorithms. In the nimble simulation setting, a rigid object without any articulation is set to have a virtual joint with the world. Our pipeline is also suitable for the rigid objects.

We denoted the modeling quality improvement in Eqn.14 as the reward of taking the action a_t given the state s_t to

Algorithm 1 Interactive perception policy network

Initialize replay memory D to capacity N.
Initialize action-value function $Q(s, a|\theta^Q)$ with random weights θ^Q.
Initialize actor $\mu(s|\theta^\mu)$ with random weights θ^μ
Initialize target action-value function $Q(s, a|\theta^{Q'})$ with random weights $\theta^{Q'}$.
Initialize target actor $\mu(s|\theta^{\mu'})$ with random weights $\theta^{\mu'}$
Initialize target network $\theta^{Q'}$ and $\theta^{\mu'}$ with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
while $i \leq M$ **do**
  Randomly select a URDF to get the initial state s_1.
  while $t \leq T$ **do**
    Select angle $\delta q_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise.
    With probability ϵ select a random joint a_t^{IP}
    Otherwise select $a_t^{IP} = \arg\max_n Q(s_t, \delta q_t; \sigma)$
    Execute action a_t^{IP} in the emulator and observe reward r_t and state s_{t+1}
    Store transition $(s_t, a_t^{IP}, r_t, \delta q_t, s_{t+1})$ in D
    Sample random mini-batch of transitions $(s_j, a_j^{IP}, r_j, \delta q_j, s_{j+1})$ from D
    Set $y_j = r_j + \gamma \arg\max_n Q(s_{j+1}, \delta q_{j+1}; \theta^Q)$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j^{IP}; \theta^Q))$ with respect to the network parameters θ^Q
    Update the actor policy using the sample policy gradient:

$$\frac{\partial J}{\partial \theta^\mu} = \frac{1}{N} \sum_i \frac{\partial Q(s, \delta q|\theta^Q)|_{s=s_j, \delta q=\mu(s_j)}}{\partial \delta q} * \frac{\partial \mu(s|\theta^\mu)|_{s=s_j}}{\partial \theta^\mu}$$

    Update target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end while
end while

train the network to learn how to make the decision a_t.

$$r_t = \mathcal{L}_{t+1}(a_t^{IP}, \mathcal{Z}, \mathcal{E}) - \mathcal{L}_{t+1}(a_t^{IP}, \mathcal{Z}', \mathcal{E}') \quad (14)$$

The observation s consists of four parts. It first contains the point cloud of the whole object $R^{3\times1024}$. Besides the xyz coordinate values, each point also has a two dimensional mask vector. The discrete mask of $(1, 0)$ represents that the point belonging to the parent link. $(0, 1)$ indicates that the point belonging to the child link. The rest points have the mask vector of $(0, 0)$. The mask feature in total has a shape of $R^{2\times1024}$. The third part of the observation is the category ("**Fixed**", "**Revolute**", "**Prismatic**") of the joint represented by one-hot $R^{3\times1024}$, and the fourth part of the observation is the parameters of the joint (xyz, rpy) $R^{6\times1024}$. When there are N joints, the observation state of the URDF is $s_t \in \mathcal{R}^{N\times14\times1024}$. For two different joints, the differences between these $R^{14\times1024}$ are the mask vectors.

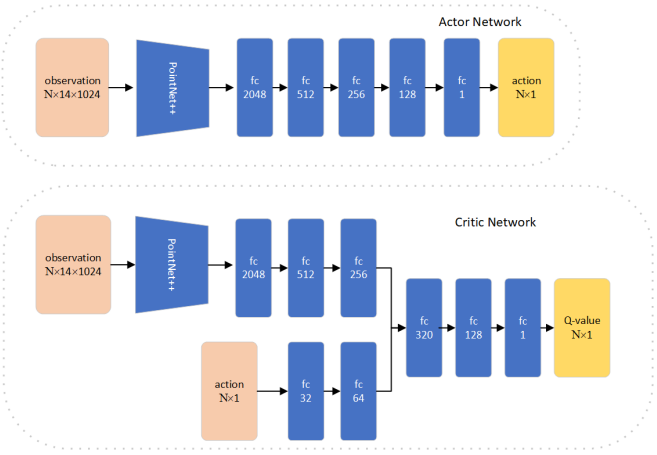


Fig. 3: Interactive perception policy network architecture.

The policy network structure is shown in Fig. 3. When the number of the joint that need to be optimized in URDF is N, the shape of the observation is $(N, 14, 1024)$. In actor network, we first use PointNet++ to extract the global features in the 3D point cloud, and then use the fully connected network to further extract the features and determine the changing joint angle/value $\delta q_t = \mu(s_t|\theta^\mu)$, and the dimensionality of the action is $(N, 1)$. In critic network, given N state-action pairs, for each state, we use PointNet++ to extract the global features in the 3D point cloud, and for the corresponding action, we use the fully connected network to extract the features. We concatenate the two features together to calculate the Q-value. The shape of the output Q-value is $(N, 1)$, in which the joint that needs to be optimized is selected through the largest Q-value.

C. Sim-Real Gap Reduction Through augmenting differential simulation with neural networks

Even if the analytical model parameters have been determined, rigid-body dynamics alone often does not exactly predict the motion of mechanisms in the real world [12]. In the interactive perception stage, our pipeline would optimize the modeling parameters to reduce the differences between simulation and the real world. However there are always discrepancy left. To address this, we propose a simulation that leverages differentiable physics models and neural networks to allow the efficient reduction of the sim-real gap. We denote s_t^{sim} and s_t as current state of simulation and real world respectively.

We develop a neural network model denoted *NeuralNet* as shown in Eqn. 16 to predict the residual change of the next state based on the current state, the current action, and the calculated next state from the Nimble simulation. Note that the calculated next state value from the Nimble simulation may vary after each parameter's updating. The neural network model would adjust the residual values based on the calculated next state. In our implementation, the neural network model is a two-layer fully-connected neural network.

In the real world, our robot would take action a_t based on the state s_t and gather the transition tuple (s_t, a_t, s_{t+1}). Meanwhile in the simulation, our robot would also take the same action a_t based on the state s_t and gather the transition tuple $(s_t^{sim}, a_t, s_{t+1}^{sim})$. Here s_t^{sim} and s_t are the same. We define a loss denoted as $\mathcal{L}^{aug}$ to measure the difference between s_{t+1}^{sim} and s_{t+1}.

$$s_{t+1}^{Nimble} = \textit{Nimble.step}(s_t^{sim}, a_t) \quad (15)$$

$$\delta s_{t+1} = \textit{NeuralNet.forward}(s_t^{sim}, a_t, s_{t+1}^{Nimble}) \quad (16)$$

$$s_{t+1}^{sim} = s_{t+1}^{Nimble} + \delta s_{t+1} \quad (17)$$

$$\mathcal{L}^{aug} = \|s_{t+1}^{sim} - s_{t+1}\| \quad (18)$$

Although we have augmented the differentiable simulation with neural networks, the augmented simulation remains differentiable as shown below.

$$\frac{\partial \delta s_{t+1}}{\partial s_t^{sim}}, \frac{\partial \delta s_{t+1}}{\partial a_t} = \textit{NeuralNet.backward} \quad (19)$$

$$\frac{\partial s_{t+1}^{sim}}{\partial s_t^{sim}} = \frac{\partial s_{t+1}^{Nimble}}{\partial s_t^{sim}} + \frac{\partial \delta s_{t+1}}{\partial s_t^{sim}} \quad (20)$$

$$\frac{\partial s_{t+1}^{sim}}{\partial a_t} = \frac{\partial s_{t+1}^{Nimble}}{\partial a_t} + \frac{\partial \delta s_{t+1}}{\partial a_t} \quad (21)$$

D. Model-Based Manipulation Learning

In this subsection, we discuss how our system produces robotic manipulation actions to successfully accomplish a given task denoted as $\mathcal{T}$. For example, the robot may receive a task request from Sec.III-B to open the microwave by θ degree, which is the action a^{IP} defined in Sec.III-B during the interactive perception stage. We first train the robot to accomplish the task $\mathcal{T}$ in the differential simulation. We then record these robotic manipulation sequences from the simulation denoted as $\{(s_t^{sim}, a_t^{sim})\}_{t=0}^{T-1}$, and utilize these manipulation sequences to guide the robotic execution in the real-world. Note that the manipulation tasks discussed in this work still require the experts to provide the corresponding goal states. In this work, we did not consider how to accomplish the long-horizon manipulation task and leave it for future work.

a) Manipulation in the Simulation: Based on the modeling of the environment, we propose two-levels procedures to guide the robot in the simulation to reach the target goal $\mathcal{G}(\mathcal{T})$ which indicates the success of the task $\mathcal{T}$.

Object-centric setting. The first level is to find a feasible path to reach the goal $\mathcal{G}(\mathcal{T})$ under the object-centric setting. In this setting, the robot is not loaded into the simulation, and we would directly control the objects. We denote the state and the action in this object-centric setup to be x_t and u_t. Here $x_t = [q_t, \dot{q}_t]$ which contains the object current joint value q_t and the joint velocity $\dot{q}_t$. u_t represents the external forces exerted directly to control the objects. In our setting, there is only one object and one robot arm in the manipulation scene. If there are multiple objects/parts in the scene and there are N_r robot arms ($N_r \in \{1, 2, ...\}$), we would restrict that there are only at most N_r objects/parts which can be actively controlled simultaneously.

We formulate the problem of finding a feasible path to reach the goal as an optimal control problem with the dynamic function denoted as $\mathcal{F}^{sim}$ and the cost function l^o shown as below. Here T is the maximal time step.

$$\mathcal{L}^o(\mathcal{T}) = \sum_{t=0}^{T-1} l_t^o(x_t, u_t; \mathcal{T}) \tag{22}$$

$$\text{s.t. } x_{t+1} = \mathcal{F}^{sim}(x_t, u_t) \tag{23}$$

$$x_0 = x_{\text{init}} \tag{24}$$

Here we explain the loss function in Eqn 25 for the articulation object manipulation tasks. The l_{T-1}^0 measures the difference between the final state of these objects (x_{T-1}) and the goal state of these objects ($\mathcal{G}(\mathcal{T})$. The distance metric depends on the task $\mathcal{T}$. In our application, we measure the differences between the final 6D poses of these objects and their corresponding goal 6D poses and adopt the angle-axis representation. We add a penalty $\mathscr{C}$ when any collision is detected.

We omit each component's weight coefficient within the following loss functions within this subsection to simply the notation.

$$l_t^o(x_t, u_t, \mathcal{T}) = \begin{cases} \|u_t\|^2 + \mathscr{C} & t < T-1 \\ \|x_{T-1} - \mathcal{G}(\mathcal{T})\|^2 + \mathscr{C} & t = T-1 \end{cases} \tag{25}$$

We could find a solution to the optimization problem leveraging the differentiable simulation. For more details about the differentiation and how to solve the loss function, we refer to [13] and their code repository documentation [14]. We record the corresponding procedure sequences denoted as $\{x_t^*, u_t^*\}_{t=0}^{T-1}$. The sequences reflect how these objects should be transformed to reach the goal $\mathcal{G}(\mathcal{T})$ successfully, which are used to guide robot actions at the next level.

Robot-centric setting. After gathering the $\{x_t^*, u_t^*\}_{t=0}^{T-1}$ in the object-centric setting, we load the robot into the simulation and start the robot-centric procedure, in which we find the robotic action sequences to accomplish the task $\mathcal{T}$. Note that in the robot-centric setup, the state $s_t^{sim} = [x_t; q_t^r, \dot{q}_t^r]$ is composed of two components: the objects' state denote as x_t, which are the joint position q_t^o and joint velocity $\dot{q}_t^o$; the robot's joint position q_t^r and joint velocity $\dot{q}_t^r$. The action a_t^{sim} in the robot-centric setup is the robot control forces denoted as τ. Note that a_t^{sim} does not contain u_t used in the object-centric setup meaning that we does not directly control objects in the robot-centric settting.

We use the $\{x_t^*, u_t^*\}_{t=0}^{T-1}$ in the object-centric setting to guided the robotic manipulation process. For example, when we want to open a microwave, the simulation would return a sequence of actions. In this cases, these actions indicates that we would need to exert a stable torques on the joint of the microwave's door so that it would open to the specific degrees. After we know what types of forces are need to exert onto these objects in order to accomplish a given manipulation task, what is left for robots to figure out is how to exert these forces on the objects through contact manipulations. We would then infer the contact regions and types.

Robotic manipulations, which involve making contact with external objects, are divided into prehensile and non-prehensile manipulations [15]. Prehensile manipulation indicates that the contact forces from the robotic manipulators alone can stabilize the object, regardless of external forces such as gravity. Meanwhile, the could also exert forces on the object to change the object's status. Non-prehensile manipulation contains pushing the door, flipping the light switch, rolling the ball on the table, etc. Given the modeling and the task, our robot would decide what type of manipulations to take. Given the sampled points, we adopt UniGrasp [16] to rank where the grasping regions. For cases, such as opening a box without any handle. There are no regions to grasp the box to perform prehensile manipulation operations. The robot would switch to consider non-prehensile manipulation such as pushing. Denote the required pushing direction to be e_1. The pushing action is only feasible if the robot could find a contact point on the object's surface with the contact point's surface inward-pointing normal vector denoted as e_2. The pushing action is acceptable if 1) the angle between e_1 and e_2 is smaller than a given degree; 2) there are feasible inverse kinematic solutions for the robot to reach the contact region. Similarly, we would sample these contact candidates on the surface of the objects and rank their scores. Some visualization results in the real objects are shown in our project webpage https://sites.google.com/view/egci.

After gathering the grasping region candidates and the pushing region candidates, we could infer the contact point positions on the object and contact types according to the state and action $\{a_t^{sim}\}_{t=0}^{T-1}$. We add contact constraints in the objectives of Eqn. 26 denoted as Ψ_t at each time step. If the contact type is grasping, the robot is optimized to maintain grasping through the grasping point. We minimize the distance between the robot's two fingers' positions. If the contact type is pushing, the robot is optimized to keep making contact through the pushing point, leveraging the contact detection module from the simulation. The robot's

initial configuration is directly inferred based on the contact type and region and is calculated through inverse kinematics. Although there are multiple candidates, we parallelize the whole computation to speed up the process.

We formulate the problem of directly finding the sequence of robotic actions $\{a_t^{sim}\}_{t=0}^{T-1}$ to accomplish the task $\mathcal{T}$ as an optimization problem. We adopt the Guided Policy Search formulation [17]. The cost function is denoted as l^r. The policy parameterized by θ is π_θ, which is a two-layer fully-connected neural network. To train the policy network, we adopt the mirror descent guided policy search (MDGPS) algorithm [18] to minimize the differences between the policy network's output and the global optimization results via supervised learning. The policy network could also be extended to incorporate multi-modality as inputs, such as images, tactile signals from various sensors. We leave it for future work.

$$\mathcal{L}^r(\mathcal{T}) = \sum_{t=0}^{T-1} l^r(s_t^{sim}, a_t^{sim}; \{x_t^*\}_{t=0}^{T-1}, \{u_t^*\}_{t=0}^{T-1}, \mathcal{T}) \quad (26)$$

$$\text{s.t. } s_{t+1}^{sim} = \mathcal{F}^{sim}(s_t^{sim}, a_t^{sim}) \quad (27)$$

$$s_0^{sim} = s_{\text{init}}^o \quad (28)$$

$$a_t^{sim} = \pi(s_t^{sim}) \quad (29)$$

Here we explain the loss function in Eqn 30

$$l^r(s_t^{sim}, a_t^{sim}; \{x_t^*\}_{t=0}^{T-1}, \{u_t^*\}_{t=0}^{T-1}, \mathcal{T}) \quad (30)$$

$$= \begin{cases} \|a_t^{sim}\|^2 + \|x_t - x_t^*\| + \mathscr{C} + \Psi_t & t < T-1 \\ \|x_{T-1} - \mathcal{G}(\mathcal{T})\|^2 + \mathscr{C} + \Psi_t & t = T-1 \end{cases} \quad (31)$$

where $\mathscr{C}$ the collision loss and Ψ_t is the contact loss. We denote the corresponding sequences as $\{s_t^*, a_t^*\}_{t=0}^T$. Note that we could switch back to the object-centric setting if some hard constraints are met, such as the robot reaching its joint limit or is near to a potential collision. We gather the object state in the above robot-centric as $\{x_t^{*r}\}_{t=0}^T$. These object states are introduced to guide the object-centric optimization process with the following modified loss function.

$$l_t^o(x_t, u_t, \mathcal{T}) = \begin{cases} \|u_t\|^2 + \mathscr{C} + \|x_t - x_t^{*r}\| & t < T-1 \\ \|x_{T-1} - \mathcal{G}(\mathcal{T})\|^2 + \mathscr{C} & t = T-1 \end{cases} \quad (32)$$

In the task of putting a cup into the microwave, we have specified the goal pose that the cup is inside the microwave. The initial pose of the cup is near the microwave and is reachable by the robot. The microwave door has already opened. There is enough space between the door and the microwave so that the robot can put the cup inside. However, there are multiple feasible trajectories to move the cup from its initial pose to the goal pose. After the first object-centric level, our pipeline infers the robot could perform prehensile manipulation to grasp the cup and put it inside the microwave. Leveraging the UniGrasp [16], we have calculated the contact points on the cup. However, the calculated trajectory in the object-centric level would return a trajectory. At some time step, the object is outside of the robot's workspace during the robot-centric level. The trajectory of the object's poses in the robot-centric level is leveraged in the next round object-centric level to guide the optimization process to find a feasible trajectory for the robot to execute.

b) Guided Manipulation in the Real world: After receiving the sequence of states $\{s_t^{sim*}, a_t^{sim*}\}_{t=0}^T$ and the policy π_θ, we would execute the learned policy $\pi_\theta(s^t)$ and the state will change to s_{t+1}. Meanwhile, we record the transition tuple in the real world $(s_t^r, a_t^r, s_{t+1}^r)$. If the current episode fails to accomplish the task $\mathcal{T}$ in the real world. These states would be added into the memory buffer which is used to improve the quality of the model as described in Sec. III-C.

In real robot implementation, our policy network's output would be adapted according to the different control paradigms of the real robot. Recent works have shown that it's promising to output forces in object pushing tasks to identify object mass and friction coefficients leveraging the differentiable simulation [19–21]. If the position control is implemented in the real robot, we could change the output of the policy network π_θ. The policy network would output the movement between this timestep and the next timestep denoted as $\pi_\theta(s_t) = q_{t+1} - q_t$ in joint space or $\pi_\theta(s_t) = FK(q_{t+1}) - FK(q_t)$ in Cartesian space, where FK is the forward kinematics. For the sim-real gap reduction in Sec. III-C, since the action is the movement displacement, we compare the difference between the next time step of object state which is $\|x_{t+1}^{sim} - x_{t+1}\|$. Our pipeline minimizes the difference of the object states in simulation (x_{t+1}^{sim}) and in the real world (x_{t+1}) after commanding the robot to move by the same distance. However sometimes the robot moves to different next states in simulation and in the real world due to object contact, collision, etc. Our pipeline also minimizes the differences between the robot next state in simulation and in the real world, along with the object next states minimization.

IV. Experiments

In this work, we develop a learning framework aiming to achieve sample-efficient, generalizable, compositional, and incremental robot learning. Our experiments focus on evaluating the following question: (1) how robust is our framework to scene flow error? (2) how effective is our proposed interactive perception framework? (3) whether our approach is more sample-efficient and has better generalization compared to other approaches? (4) How useful is our model in zero/few-shot learning, task composition/combination, long-horizon manipulations tasks?

A. Experimental Setup

a) Simulation: We use PyBullet [22] to simulate the real world which is different from our differentiable simulation based on Nimble [13]. We conduct our experiments using 6 object categories from the SAPIEN dataset [1], which are box, door, microwave, oven, refrigerator, and storage

furniture. We select 176 models in total, 143 models for training, and 33 models for testing.

b) Real World: We also set up the real world experiment. We amount an RGB-D camera RealSense on the wrist of the 7-Dof Franka robot.

B. Evaluating the Robustness

To better understand how scene flow affects the performance of IP, we use three scene flow results with different qualities. They are the predicted scene flow from our model, noisy scene flow in which the random noise ranging from 0cm to 5cm is added to each point of our predicted scene flow, and ground truth scene flow. We evaluate the IoU of motion segmentation (mIoU), joint type classification accuracy (Acc.), joint axis translation (Tran.) and rotation (Rot.) error before and after one optimization step. The results are shown in Tab. I. There are no significant differences between the results under "sf+noise" column and "sf" column, indicates interactive perception can effectively improve the precision on several metric even when the scene flow is relatively noisy. We also observe that if the ground-truth scene flow is provided in the process, our pipeline could achieve better performance. As an active agent, a robot might directly put markers to collect ground-truth scene flow. We leave how to gather such scene flow results as future works.

TABLE I: Robustness of interactive perception.

	Init	sf + noise	sf	sf GT
mIoU ↑	0.574	0.757	0.791	0.975
Acc. ↑	0.972	0.905	0.922	0.965
Rot. ↓	12.414	8.910	7.703	5.502
Tran. ↓	19.970	3.199	2.578	0.898

C. Evaluating the Performance of Interactive Perception

In this subsection, we compare the modeling qualities before and after the interactive perception operations. Given the initial modeling of the environment, we verify and modify each part on the predicted URDF with a sequence of interactions, and online optimize the URDF based on the novel signal created by each interaction step. To evaluate the performance, We compute the average precision under IoU 0.75 of instance part segmentation (AP75), joint type classification accuracy (Acc.), joint orientation error (Rot.), and joint position error (Tran.). Comparing the results of the "init" and "opt" in Tab. II, the pipeline can significantly improve the URDF quality after interactive perception.

TABLE II: Performance of interactive perception.

	AP75 ↑		Acc. ↑		Rot. ↓		Tran. ↓	
	init	opt	init	opt	init	opt	init	opt
Door	**0.748**	0.714	0.975	**0.975**	13.713	**2.252**	15.806	**3.670**
Microwave	0.800	**0.939**	1.000	**1.000**	11.562	**4.230**	15.439	**3.788**
Box	**0.895**	0.822	0.877	**0.907**	10.143	**4.530**	15.241	**7.899**
Oven	0.773	**0.875**	1.000	**1.000**	23.525	**6.589**	18.731	**10.066**
Fridge	0.584	**0.814**	**0.989**	0.955	12.174	**5.331**	18.000	**7.820**
Storage	0.499	**0.519**	0.924	**0.949**	11.564	**9.831**	33.510	**7.404**
Overall	0.717	**0.781**	0.961	**0.964**	13.780	**5.461**	19.455	**6.775**

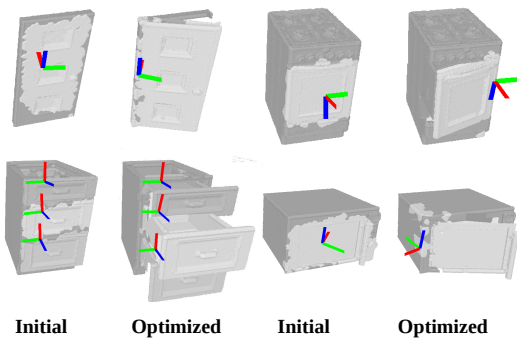
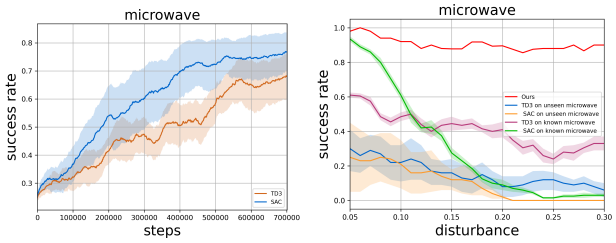


Fig. 4: Qualitative Comparison between the initial URDF and the optimized URDF. Different color indicates different part, and z-axis (blue) indicates the joint axis.

We visualize the initial URDF and the optimized URDF in Fig. 4. The segment and joint parameter have been improved after interactive perception.

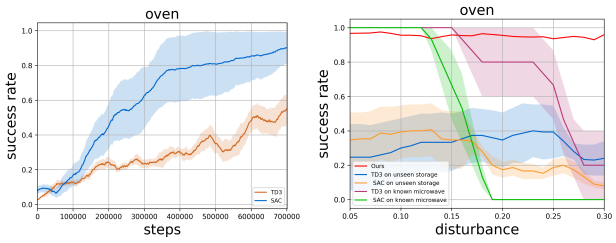
D. Evaluating the Sample-Efficiency and Generalization

We compare our proposed approach with two popular model-free RL algorithms: SAC [23] and TD3 [24] on articulation object manipulation tasks. The tasks are to teach the robot to open the articulation objects of six classes. We post the result of opening the microwaves and more results on other categories.



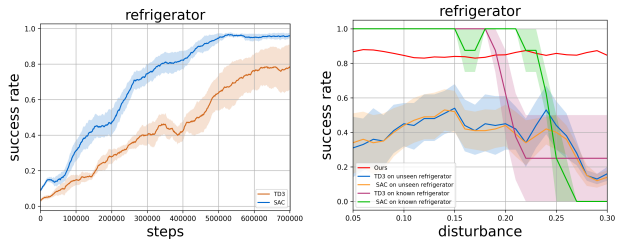
(a) Learning curve (b) Generalization Ability

Fig. 5: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the microwave



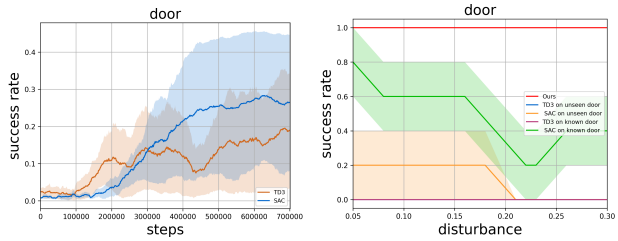
(a) Learning curve (b) Generalization Ability

Fig. 6: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the oven



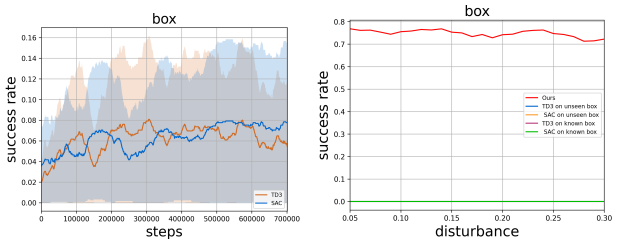
(a) Learning curve (b) Generalization Ability

Fig. 7: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the refrigerator



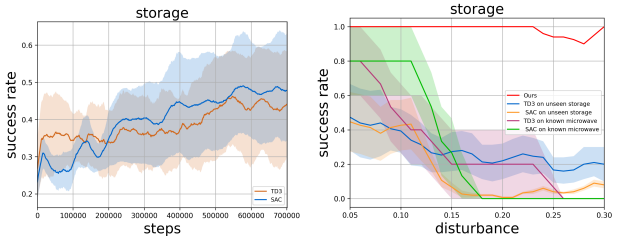
(a) Learning curve (b) Generalization Ability

Fig. 8: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the door



(a) Learning curve (b) Generalization Ability

Fig. 9: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the box



(a) Learning curve (b) Generalization Ability

Fig. 10: (a) Learning curves of SAC and TD3. (b) Comparison of SAC, TD3 and our method in terms of the generalization on the storage

For the model-free RL approaches, the observation in the microwave environment is 18 dimensions, including the handle's 6D pose, the microwave's revolute joint value, the robot end-effector's 6D pose, the distance between the robot's two fingers, and the handle's goal 3D position with the corresponding joint state of the microwave. The action space is seven dimensions, including the robot's end-effector's 6D pose and the opening width between its two fingers. The reward consists of three parts: 1) the first part is the negative distance from the handle's current 3D position to the target 3D position; 2) the second part is the negative distance from the robot's current revolute joint values to the corresponding target values; 3) the third part is the negative distance between the handle's 3D position and the end-effector's 3D position. Every time the environment is initialized or reset, the position and rotation of the microwave will be randomly set within a certain range, and the robot's gripper is also set randomly within a certain region. Fig. 10(a) shows the average success rate of the SAC and TD3 in the training stage. After training 700k steps, the average success rate of SAC models and TD3 models to open microwaves are about 80% and 70%, respectively. However, our pipeline achieves a success rate of around 90% after five interactive perception operations and training experiences/samples. Moreover, the modeling constructed by our approach could be adopted for other related tasks immediately.

To evaluate the generalization abilities of different approaches, we move the robot end-effector's 6D pose and the articulated object's 6D pose by a 6D offset/disturbance. As shown in Fig. 10(b), with the increasing disturbance to the robot end-effector's initial 6D pose, the performance of SAC and TD3 decreases rapidly, especially in manipulating unseen microwaves. The results show that, overall, our method performs comparably to the baseline methods on the small disturbance and outperforms them on the large disturbance by a large margin and on unseen microwaves, indicating a significantly better generalization ability.

E. Evaluating the compositionality and incrementality

To evaluate the compositionality and incrementality, we conduct two real-world experiments. The first one is to open the microwave, put the mug into it, and close it. And the second one is to open the drawer, take the tap out, and close the drawer.

First, we need to scan the point cloud. We mount the RealSense D415 RGB-D camera on the robot wrist, then scan the point cloud from several different views. Since the camera is calibrated, we can fuse different views directly based on the camera pose, then utilize ICP to refine the point clouds. Based on the scanning, we can generate the URDF file, apply the model-based manipulation learning method to generate manipulation policy, and verify and modify the URDF via interactive perception. To put the mug in the microwave and take the tap out from the drawer, we choose Unigrasp [16] to generate a grasp pose. We adopted position control in these real-world experiments.

The URDF file is initially estimated before opening the microwave/drawer. The modeling is updated during the first task of opening the microwave/drawer. Once the URDF is

well-estimated, it reduces the modeling efforts for other tasks related to the same object(microwave/drawer), such as pushing the microwave/drawer, closing the microwave/drawer, putting an object inside the microwave, or taking an object outside the drawer. This allows the newly learned skills to be easily integrated into the existing modeling of the surrounding environment. Moreover, the newly learned skill can continue to improve the modeling quality, which helps contribute to the existing knowledge.

The simulation plays the role of "mental model" [25] to endow robots with the ability of forward predicting the action's effects and the backward inverting ability. The inverting ability allows robots to find a trajectory from the initial states to the goal states, satisfying the objectives or constraints. The model-based manipulation sequences are produced in the "mental model", making it easy to be analyzed and combined with other manipulation sequences.

F. Evaluating the object-centric and robotic-centric model-based approach

To evaluate the importance of our proposed model-based manipulation learning approach, we design comparison experiments. We remove the object-centric level. The whole optimization process becomes the following equation

$$\mathcal{L}^a(\mathcal{T}) = \sum_{t=0}^{T-1} l(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{33}$$

$$\text{s.t. } s_{t+1}^{sim} = \mathcal{F}^{sim}(s_t^{sim}, a_t^{sim}) \tag{34}$$

$$s_0^{sim} = s_{\text{init}}^o \tag{35}$$

$$a_t^{sim} = \pi(s_t^{sim}) \tag{36}$$

Here we explain the loss function in Eqn. 33

$$l(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{37}$$

$$= l^a(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{38}$$

$$= \begin{cases} \|a_t^{sim}\|^2 + \mathscr{C} + \Psi_t & t < T-1 \\ \|x_{T-1} - \mathcal{G}(\mathcal{T})\|^2 + \mathscr{C} + \Psi_t & t = T-1 \end{cases} \tag{39}$$

Although the contact type and contact regions are inferred from $\{x_t^*, u_t^*\}_{t=0}^{T-1}$ produced in the object-centric level, we put the robot at the same initial pose configurations with our proposed two-level (object-centric and robot-centric) approach. We add the contact loss component denoted as Ψ_t.

In addition to the above loss function, we also run another comparison experiment where the contact loss component is removed as shown in the following equation.

$$l(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{40}$$

$$= l^b(s_t^{sim}, a_t^{sim}; \mathcal{T}) \tag{41}$$

$$= \begin{cases} \|a_t^{sim}\|^2 + \mathscr{C} & t < T-1 \\ \|x_{T-1} - \mathcal{G}(\mathcal{T})\|^2 + \mathscr{C} & t = T-1 \end{cases} \tag{42}$$

Using $l^b(s_t^{sim}, a_t^{sim}; \mathcal{T})$ and $l^b(s_t^{sim}, a_t^{sim}; \mathcal{T})$ as the loss function, there are no success cases in the articulated object manipulation tasks described in Sec.IV-D. The results indicate that the contact-rich articulated object manipulation tasks are complicated and our proposed two-level(object-centric and robot-centric) model-based approach is effective.

V. NOTATION

We create a notation table shown in Table III summarizing the symbols used in this work.

Symbol	explanation
N	the number of raw points
$\mathcal{P}_t^i$	the 3D position of the raw point cloud at time step t
$\mathcal{M}_t^i$	the part segmentation of the point cloud at time step t
K	the number of parts
$\mathcal{G}_t^j$	point clouds group
ζ_t^i	centroid position
$\mathcal{B}_t^i$	part boundary
$\mathcal{S}_t^i$	confidence score
D	the number of points whose $\mathcal{S}_t^i$ is one
$\mathcal{L}$	loss function
α	physical properties
$\mathcal{J}$	joint relationship
$\mathcal{C}$	joint spatial description
$\mathcal{E}$	URDF tree structure
$\mathcal{Z}$	model patameter set
a_t	action at time step t
$\mathcal{F}$	correspondence function
δq	joint state change after a_t executed
$\mathbf{T}$	transformation matrix
$\mathscr{C}$	the collision penalty

TABLE III: Notation Table

REFERENCES

[1] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, L. Yi, A. X. Chang, L. J. Guibas, and H. Su, "SAPIEN: A simulated part-based interactive environment," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[2] L. Shao, P. Shah, V. Dwaracherla, and J. Bohg, "Motion-based object segmentation based on dense rgb-d scene flow," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3797–3804, 2018.

[3] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[4] J. Huang, Y. Zhou, and L. Guibas, "Manifoldplus: A robust and scalable watertight manifold surface generation method for triangle soups," *arXiv preprint arXiv:2005.11621*, 2020.

[5] Wiki, "Wiki kruskal algorithm," https://en.wikipedia.org/wiki/Kruskal%27s_algorithm.

[6] R. W. Brockett, "Robotic manipulators and the product of exponentials formula," in *Mathematical theory of networks and systems*. Springer, 1984, pp. 120–129.

[7] X. Liu, C. R. Qi, and L. J. Guibas, "Flownet3d: Learning scene flow in 3d point clouds," *CVPR*, 2019.

[8] E. Herbst, X. Ren, and D. Fox, "Rgb-d flow: Dense 3-d motion estimation using color and depth," in *Robotics and Automation (ICRA), 2013 IEEE Int. Conf. on*. IEEE, 2013.

[9] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers, "A primal-dual framework for real-time dense rgb-d scene flow," in *Robotics and Automation (ICRA), 2015 IEEE Int. Conf. on*, 2015.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] 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," *arXiv preprint arXiv:1509.02971*, 2015.
[12] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "Neuralsim: Augmenting differentiable simulators with neural networks," *arXiv preprint arXiv:2011.04217*, 2020.
[13] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics for articulated rigid bodies with contact," in *Proceedings of Robotics: Science and Systems (RSS)*, July 2021.
[14] nimble, "Nimble physics documentation," https://nimblephysics.org/docs.
[15] I. M. Bullock and A. M. Dollar, "Classifying human manipulation behavior," in *2011 IEEE International Conference on Rehabilitation Robotics*, 2011, pp. 1–6.
[16] L. Shao, F. Ferreira, M. Jorda, V. Nambiar, J. Luo, E. Solowjow, J. A. Ojea, O. Khatib, and J. Bohg, "Unigrasp: Learning a unified model to grasp with multifingered robotic hands," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2286–2293, 2020.
[17] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1–9.
[18] W. H. Montgomery and S. Levine, "Guided policy search via approximate mirror descent," *Advances in Neural Information Processing Systems*, vol. 29, pp. 4008–4016, 2016.
[19] C. Song and A. Boularias, "Learning to slide unknown objects with differentiable physics simulations," in *Proceedings of Robotics: Science and Systems (RSS)*, 2020.
[20] C. Song and A. Boularias, "A probabilistic model for planar sliding of objects with unknown material properties: Identification and robust planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.
[21] C. Song and A. Boularias, "Identifying mechanical models of unknown objects with differentiable physics simulations," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 749–760.
[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.
[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
[24] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.
[25] L. P. Kaelbling, "The foundation of efficient robot learning," *Science*, vol. 369, no. 6506, pp. 915–916, 2020.