Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

ArticuBot: Learning Universal Articulated Object Manipulation Policy via Large Scale Simulation

Yufei Wang*[1], Ziyu Wang*[2], Mino Nakura†[1], Pratik Bhowal†[1], Chia-Liang Kuo†[3],
Yi-Ting Chen[3], Zackory Erickson‡[1], David Held‡[1]
[1]Robotics Institute, Carnegie Mellon University
[2]IIIS, Tsinghua University
[3]Department of Computer Science, National Yang Ming Chiao Tung University
*†Equal Contribution, ‡Equal Advising

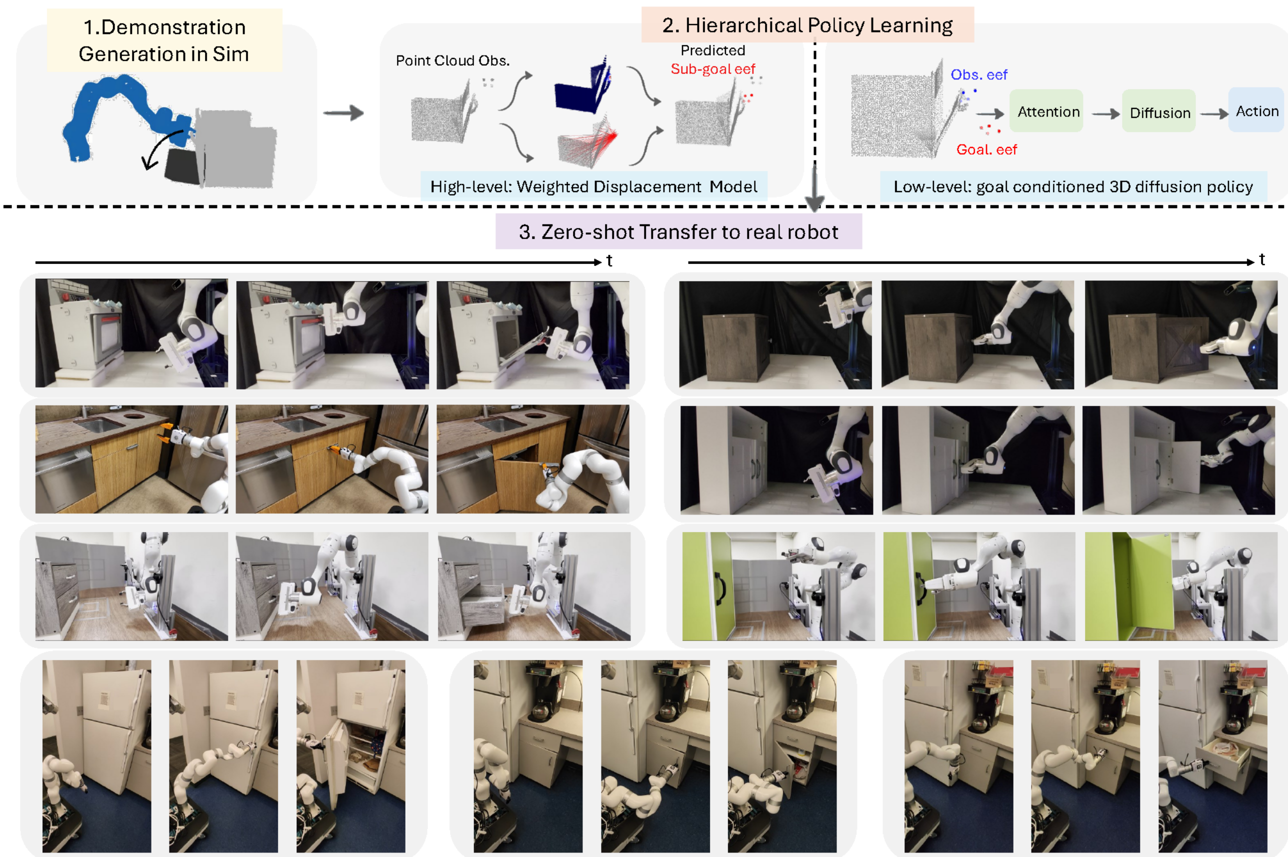

Fig. 1: Overview and real-world results of ArticuBot. **Top**: We generate thousands of demonstrations using a physics-based simulator, and distill them into a hierarchical policy with point cloud observations. **Bottom**: The learned policy can transfer zero-shot to table-top Franka arms in two different labs and a mobile X-Arm, and can open diverse unseen articulated objects in both labs, real kitchens and lounges.

Abstract—**This paper presents ArticuBot, in which a single learned policy enables a robotics system to open diverse categories of unseen articulated objects in the real world. This task has long been challenging for robotics due to the large variations in the geometry, size, and articulation types of such objects. Our system, ArticuBot, consists of three parts: generating a large number of demonstrations in physics-based simulation, distilling all generated demonstrations into a point cloud-based neural policy via imitation learning, and performing zero-shot sim2real transfer to real robotics systems. Utilizing sampling-based grasping and motion planning, our demonstration generalization pipeline is fast and effective, generating a total of 42.3k demonstrations over 322 training articulated objects. For policy learning, we propose a novel hierarchical policy representation, in which the high-level policy learns the sub-goal for the end-effector, and the low-level policy learns how to move the end-effector conditioned on the predicted goal. We demonstrate that this hierarchical approach achieves much better object-level generalization compared to the non-hierarchical version. We further propose a novel weighted displacement model for the high-level policy that grounds the prediction into the existing 3D structure of the scene, outperforming alternative policy representations. We show that our learned policy can zero-shot transfer to three different real robot settings: a fixed table-top Franka arm across two different labs, and an X-Arm on a mobile base, opening multiple unseen articulated objects across two labs, real lounges, and kitchens. Videos and code can be found on our project website: https://articubot.github.io/.**

I. INTRODUCTION

Robotic manipulation of articulated objects, such as cabinets, drawers, fridges, microwaves, has wide applications as such objects are ubiquitous in both industrial and household settings. Having a single robotics policy that can generalize to manipulate diverse articulated objects has long been challenging due to the large variations in the geometry, shape, size and articulation types of such objects. Many prior works have studied the problem of articulated object manipulation [58, 31, 10, 19, 21, 53, 15, 32]. However, few have demonstrated generalization to manipulating many different articulated objects in the real world without simplifying assumptions (e.g., using a suction gripper [10]). In this paper, we aim to learn a generalist articulated object manipulation policy that can open various kinds of articulated objects in the real world with commercial robotic manipulators equipped with a parallel jaw gripper, purely from visual observations, without assuming access to knowledge of the articulation parameters.

Motivated by a recent trend of success in scaling up robot learning with large datasets, we aim to learn a universal articulated object opening policy following this paradigm: generating thousands of demonstrations in physics-based simulation, distilling the generated data into a generalizable policy by imitation learning, and then performing zero-shot sim2real transfer. This is a paradigm that has been applied in previous work to learn general policies for different robotics tasks, such as grasping [12, 45], locomotion [26, 25, 67], assembly [47, 46], and deformable object manipulation [51, 16].

In this paper, we investigate various ways to realize such a system to learn a generalist policy for articulated object manipulation. We first build an efficient data generation pipeline that combines sampling-based grasping, motion planning, and action primitives. Using the pipeline, we have generated a large dataset consisting of thousands of (42.3k) demonstrations over 322 articulated objects. We also show that using a hierarchical policy representation, in which a high-level policy predicts sub-goal end-effector poses and a low-level policy predicts delta end-effector transformations, performs much better than the non-hierarchical version when imitating the generated large dataset. We also explored various design choices for the policy representations to study which architecture scales up the best when learning with a large number of demonstrations. We show that a weighted displacement model that leverages the underlying 3d scene structure can scale and generalize better than models that do not incorporate such 3D reasoning.

Our final policy, trained with 42.3k trajectories and 322 objects in simulation, can transfer zero-shot to the real world to open diverse unseen real articulated objects. Furthermore, although our policy is only trained on a Franka arm in simulation, we show that it can transfer zero-shot to two different embodiments in the real world: a table-top Franka arm, and a mobile-base X-Arm. This is achieved by using the policy to learn actions in the robotic arm's end-effector space instead of the joint space. Our final policy is successfully deployed in 3 different real-world settings: two table-top Franka arms in two different labs, and a mobile-base X-Arm in various real kitchens and lounges. This single policy is able to open 20 different unseen real-world articulated objects such as cabinets, drawers, microwaves, ovens, and fridges in these different test settings, in a zero-shot manner. See Fig. 1 for a visualization of some of the different real-world articulated objects that our policy is able to open.

In summary, our contributions are:

- A system that presents a single policy trained on thousands of demonstrations generated in simulation, that can zero-shot transfer to the real world and generalize to open various articulated objects with 2 robot embodiments: a table-top Franka, and a mobile base X-Arm.
- We show that using a hierarchical policy representation is better than the non-hierarchical version to achieve object-level generalization.
- We present a weighted displacement policy representation that scales up well with the number of demonstrations, outperforming alternative policy representations.
- A large articulated object manipulation simulation dataset that contains 42.3k demonstration trajectories for 322 articulated objects, and a pipeline for quickly generating additional demonstrations.

II. RELATED WORK

A. *Robot Learning for Articulated Object Manipulation*

There is a rich body of prior work studying the problem of articulated object manipulation [31, 53, 10, 62, 58, 32, 19, 61, 20, 15, 50, 56]. Most of these prior works show major results in simulation, with limited real-world manipulation results [31, 53, 62, 58, 19, 61, 20, 50]. In contrast, our work aims to learn a manipulation policy that can transfer and generalize to diverse real-world articulated objects. Eisner et al. [10] shows a number of tests on real-world articulated objects in a table-top lab setting with a suction gripper to simplify grasping. Our policy works with the standard parallel jaw gripper which is more commonly equipped with robotic manipulators and perform grasping of the handles for opening. We also show results with a mobile manipulator in real kitchens and lounges. Gupta et al. [15] proposes a system that integrates various modules for perception, planning, and action and shows that it can open various cabinets and drawers with a mobile manipulator in real kitchens. Our method does not employ layered modules, instead, we directly learn a policy via imitation learning that maps sensory observations to actions. Our method also generalizes to a more diverse range of articulated objects such as fridges, microwaves, and ovens. Xiong et al. [56] builds a mobile base manipulator for articulated object manipulation and learn object-specific policies via imitation learning and reinforcement learning directly in the real world. We learn our manipulation policies by constructing much larger demonstration datasets in simulation and performing sim2real transfer, and we learn a single policy that can generalize to various articulated objects. Some prior works learn to first predict the articulation parameters and then use the predicted articulation parameters for manipulation [19, 61, 20]. Our policy directly learns how to manipulate the object without explicitly inferring the articulation parameters. Another line of work [21, 5, 30, 28] focus on reconstructing the articulated objects from real-world images to simulation. Our work focuses on manipulation rather than real2sim reconstruction. A recent work [32] learns specific grasps for articulated objects that are useful for downstream manipulation. Our policy learns not only the grasping, but also the opening; further, we compare to this prior work and show significantly improved performance.

B. Sim2real Policy Learning

Learning a policy via simulation training and then transferring to the real world (sim2real transfer) has been applied to many different domains in previous work, including legged locomotion [26, 6, 38], grasping [12, 45, 8], in-hand object reorientation [1, 4, 37], catching objects [63], deformable object manipulation [51, 59], and more [9, 13]. No prior work has demonstrated the learning of a generalizable policy for articulated object manipulation via sim2real transfer. Many of these prior works use reinforcement learning and teacher-student learning to learn the policy in simulation [26, 38, 6, 37, 63, 51]. In contrast, we generate demonstrations in simulation using a combination of techniques including sampling-based grasping, motion planning, and action primitives, and learn the policy via imitation learning. Some recent work [52, 49, 18] attempts to automate simulation policy learning for many tasks. In contrast, our work focuses on learning a generalizable policy for articulated object manipulation.

C. Robotic Foundation Models

Many recent works aim to develop a foundation model for robotics, where a single model can perform multiple tasks or generalize to different settings [2, 3, 24, 48, 11, 33, 8, 27]. Most of them perform imitation learning with a large set of demonstrations collected in the real world [2, 3, 24, 48, 11, 33, 27]. Instead, we generate demonstrations and learn the policy in simulation, and then we perform sim2real transfer to deploy it in the real world. Most of these works do not focus on tasks involving articulated objects and do not demonstrate the policy working for manipulating diverse articulated objects [2, 3, 24, 48, 27, 8], while our paper focuses on building a generalizable policy specifically for articulated object manipulation. Etukuru et al. [11] shows the most diverse real-world test settings for articulated object manipulation among these previous works. However, they train two different policies for drawer and cabinet opening. In contrast, we train a single model that can be applied to opening various categories of articulated objects. Besides, their system requires a specialized gripper, whereas our method works for general parallel jaw grippers and transfer across two different grippers.

III. Problem Statement and Assumptions

The task we are considering is for a robotic arm to open an articulated object within the category of drawers, cabinets, ovens, microwaves, dishwashers, and fridges. We assume that the object should have a graspable handle, so it can be opened in the fully closed state. We aim to learn a policy π, that takes as input a sensory observation and robot proprioception o, and outputs actions a that opens the articulated object. We assume the robot arm is equipped with a common parallel jaw gripper instead of a suction gripper or a floating gripper, which are often assumed in prior works for simplification [10, 58]. We also assume access to a pool of articulated object assets to be used in simulation, as well as annotations to handles (in simulation only). For effective sim2real transfer, we use point clouds as the sensory observations. We assume the name of the target object to manipulate, such that we can run a open-vocabulary segmentation method, e.g., Grounded SAM [39], to segment the object and obtain object-only point clouds.

IV. ArticuBot

Fig. 2 gives an overview of our system, which consists of 3 stages. The first is large-scale demonstration generation, in which we combine methods from motion planning, sampling-based grasping, and action primitives to generate thousands of demonstrations in simulation. The second is hierarchical policy learning, in which we perform imitation learning on the generated demonstrations to distill them into a vision-based policy. Finally, we deploy our simulation-trained policy zero-shot to the real world, on two table-top Franka arms in two different labs and an X-Arm on a mobile base in real kitchens and lounges, opening real-world cabinets, drawers, microwaves, fridges, dishwashers, and ovens.

A. Demonstration Generation in Simulation

First we describe our procedure for automatically generating thousands of demonstrations in simulation. We use the PartNet-Mobility [55] dataset, which contains hundreds of articulated objects. Among these, we use the categories of storage furniture, microwave, oven, dishwasher, and fridge. The majority of the assets in these categories have annotations of handles; we filter out assets that do not have such annotations, since the position of the handle is needed for generating the demonstrations.

The process of opening an articulated object can be decomposed into two substeps: grasping the handle, and then opening it along the articulation axis. Our demonstration generation pipeline follows these two substeps as well (see Fig. 2 top left for an illustration of the process): we first perform sampling-based grasping to generate hundreds of end-effector grasping poses on the handle. For each generated grasp, we approach the grasping pose using collision-free motion planning. After performing the grasp, since we have the ground-truth articulation information of the object in simulation, we move the end-effector along the articulation axis for a fixed distance to open it. We detail each of these three steps below.

Simulation Initialization: We use a Franka arm in simulation for generating the demonstrations. The base of the Franka Arm is initialized at the world origin. We randomize the position, orientation, and size of the object, as well as the initial joint

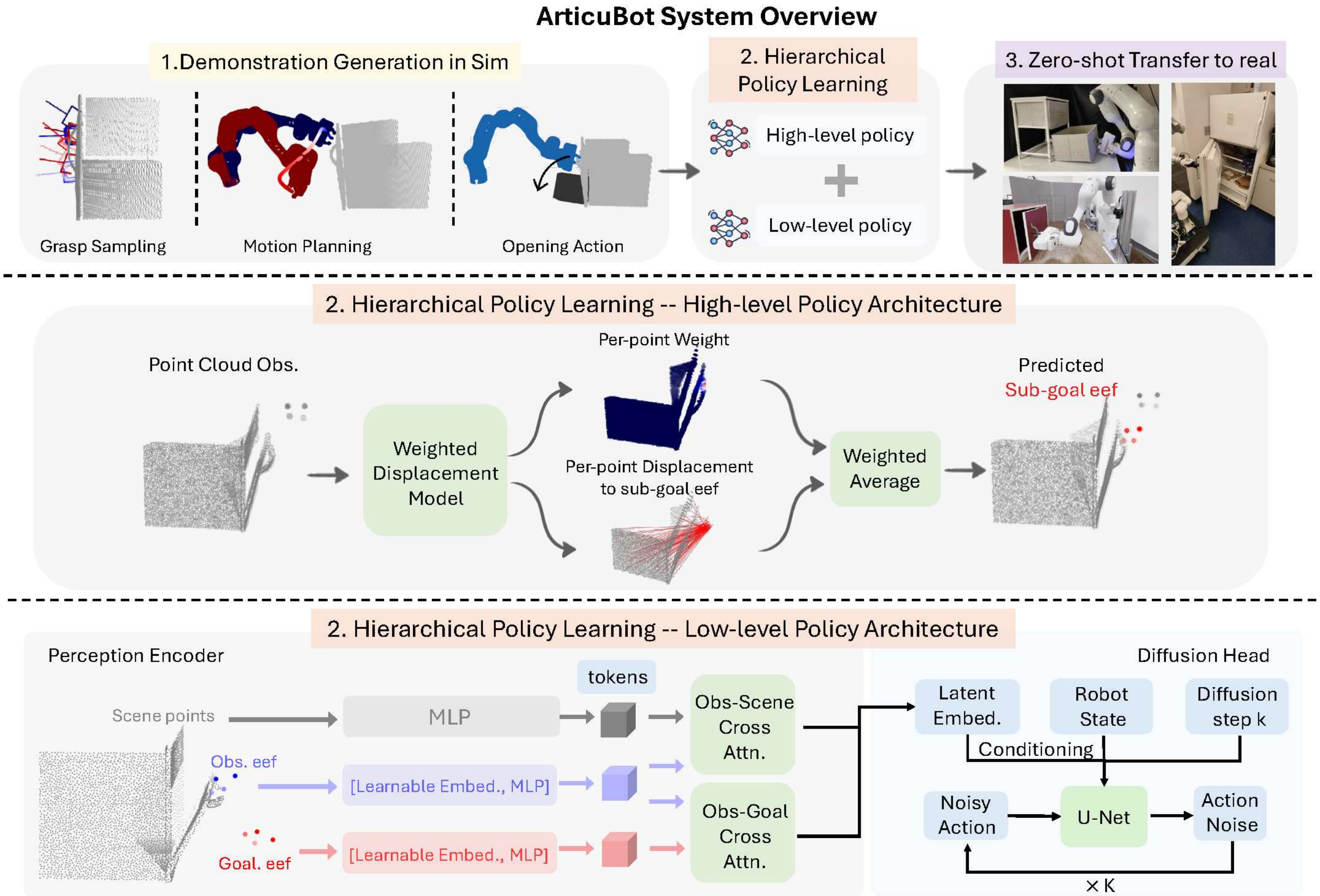

Fig. 2: System overview of ArticuBot. **Top**: We combine sampling-based grasping, motion planning, and opening actions to efficiently generate thousands of demonstrations in simulation. These demonstrations are distilled into a hierarchical policy via imitation learning, and then zero-shot transferred to real world. **Middle**: We propose a weighted displacement model for the high-level policy, which predicts the sub-goal end-effector pose. The weighted displacement model predicts the displacement from each point in the point cloud observation to the sub-goal end-effector, as well as a weight for each point. The final prediction is the weighted average of each point's prediction. **Bottom**: We propose a goal-conditioned 3D diffusion policy for the low-level policy, which first applies attention between the current end-effector points, the scene points, and the goal end-effector points to obtain a latent embedding, and then performs diffusion on the latent embedding to generate the action, which is the delta transformation of the robot end-effector.

angle of the Franka Arm, to increase diversity in the generated demonstrations. The detailed parameters for the randomization can be found in Appendix D.

Sampling Based Grasping: Given an articulated object from PartNet-Mobility and a link (i.e., a door) we want to open, we first obtain a point cloud of the link's handle using the annotations from the dataset. We perform farthest point sampling on the handle point cloud to get $m_1 = 15$ candidate grasping positions. For each grasping position, to generate the grasping orientation, we align the z-axis of the robot end-effector (which is the direction that points from the root of the hand to the finger) with the normal direction of that handle point. We set the y direction of the end-effector (which is the direction along which the finger opens and closes) to be horizontal if the handle is vertical (i.e., its height is larger than its width), and vice versa. We also sample $m_2 = 8$ random small angle perturbations ($< 30°$) about the y axis to increase the diversity of our grasp pose candidates. This generates in total $m_1 \times m_2 = 120$ grasping pose candidates. See Fig. 2 (top left) for an illustration of the sampled grasps.

Motion Planning for reaching the grasping pose: For each of the grasp candidates, we first use inverse kinematics (IK) to compute a target joint configuration of the robot arm. We solve the IK for $m_3 = 80$ times and filter out solutions that have collisions between the robot arm and the environment (e.g. collisions with the floor or the target object). Among the collision-free solutions, we choose the one solution that has the shortest distance in the joint angle space to the current joint configuration, so as to minimize the distance of the path needed to reach the target joint configuration. We then run three different motion planning algorithms, RRT* [22], BIT* [14] and ABIT* [43], to generate the path to reach the target configuration. We smooth the resulting path from each algorithm by shortcutting unnecessary waypoints and using B-spline smoothing [17]. We keep the path that has the shortest length in terms of the total end-effector movement. See Fig. 2 (top left) for a visualization of the motion planned path.

Generating Opening Actions: Next, we generate demonstrations in simulation of the robot executing the opening action. After the grasping pose is reached via motion planning, we close the gripper to form a grasp. Using the ground-truth articulation information of the object, we can compute an idealized end-effector trajectory that opens the object perfectly. Formally, let T_{eef}^{init} represent the end-effector's pose after it grasps the handle, and let $T_{door}(\theta)$ represent the pose of the door at joint angle θ. We compute the idealized trajectory based on the fact that the relative pose between the robot end-effector and the door should remain unchanged during the trajectory of opening the door, i.e., $T_{rel} = T_{door}^{-1}(\theta)T_{eef}$ should be a constant for any joint angle θ. Assume the door is at joint angle θ_{init} when the robot grasps it, then the pose of the robot end-effector when the door is opened at joint angle θ can be computed as: $T_{eef} = T_{door}(\theta)T_{door}^{-1}(\theta_{init})T_{eef}^{init}$. We can then compute a trajectory for the end-effector pose that opens the object with increasing values of θ, e.g., from $0°$ to $90°$ with an interval of $1°$. IK is then performed for the end-effector to reach each of the computed poses along the trajectory to open the object.

Filtering: Some of the trajectories will fail to fully open the door due to various reasons such as: no collision-free joint angles can be found at the sampled grasping pose, motion planning failed to find a collision-free path to reach the grasping pose, the grasping pose does not result in a firm grasp of the handle, or the end-effector slips off the handle partway during opening. We filter out all trajectories where the final opened angle (radians for hinge doors and centimeters for drawers) is smaller than a threshold, e.g., if the door is opened less than 60 degrees. From the remaining successful trajectories, we choose the single best trajectory according to the following two metrics: 1) the stability of the grasp, which is approximately measured as the number of handle points that are between the end-effector fingertips, and 2) the length of the motion planned trajectory (in the end-effector space) to reach the grasping pose. Each trajectory is first ranked using these two metrics, and the final rank is the sum of the two individual ranks. The trajectory with the highest rank is kept as the final best trajectory for opening the door.

By employing the above data generation pipeline, and executing each of the $m_1 \times m_2$ trials in parallel, we can generate optimal trajectories for opening an articulated object. Using a CPU with 128 virtual cores, one optimal opening trajectory can be generated within 2 minutes. Using this approach, we have generated 42.3k successful opening trajectories for 322 objects in PartNet-Mobility.

B. Policy Learning with a Hierarchical Policy Representation

We now describe how we distill the above generated trajectories into a vision-based neural policy via imitation learning. Formally, the above approach gives us a set of generated demonstrations $\{\tau_i\}_{i=1}^N$, where each trajectory τ_i is a list of observation-action pairs: $\tau_i = \{(o_1^i, a_1^i), ..., (o_T^i, a_T^i)\}$. The observations include point clouds of the scene and the robot proprioception (end-effector pose and finger open/close). We perform segmentation on the scene point cloud to remove the background and leave only the target object. In simulation, this can be achieved using the ground-truth segmentation masks provided by the simulator; in the real world, we use an open-vocabulary object segmentation model, e.g., Grounded SAM [39]. See Fig. 2 (middle) for an example object point cloud in simulation. The actions represent the delta transformation of the end-effector, which includes the delta translations, delta orientations, and delta finger movement. We use the robot base frame as our reference frame, i.e., all point cloud observations, and robot actions, are expressed in the robot base frame.

Our goal is to find a neural network policy π, parameterized by θ, to minimize the following imitation learning loss:

$$\mathcal{L}_\theta = \sum_{i=1}^{N}\sum_{t=1}^{T} ||\pi_\theta(o_t^i) - a_t^i||_2^2 \tag{1}$$

The goal for the policy is to be able to generalize to open various kinds of different objects, which possess diverse geometries, shapes, and articulation types. We hypothesize that, to achieve object-level generalization in such a case, it is inefficient to just learn to predict actions as the low-level delta transformations of the end-effector. Instead, we propose to use a hierarchical policy representation, which consists of a high-level policy and a low-level policy. The high-level policy will learn to predict the sub-goal end-effector poses, e.g. intermediate waypoints of where the gripper should be at various key frames in the trajectory. The low-level policy still learns to predict the low-level delta transformations of the end-effector at each timestep, but it is additionally conditioned on the high-level prediction of the sub-goal end-effector pose, which helps the low-level policy to better generalize across diverse objects. We now detail how each of the policies work.

High-Level Policy. Intuitively, the high-level policy aims to predict "where" the robot should move to. Specifically, the high-level policy π_θ^H learns to predict the sub-goal end-effector pose given an observation. The sub-goal end-effector pose is defined as the pose of the robot end-effector at the end of each substep for a given task. In our case, the task of opening an articulated object (e.g., a cabinet) can be decomposed into two substeps: grasping the handle and opening the door. Thus for this task, the sub-goal end-effector poses are the poses where the robot has grasped the handle, and when it has fully opened the door. Formally, the high-level policy π_θ^H is learned via minimizing the following loss:

$$\mathcal{L}_\theta = \sum_{i=1}^{N}\sum_{t=1}^{T} ||\pi_\theta^H(o_t^i) - a_{pose_t}^i||_2^2 \tag{2}$$

where $a_{pose_t}^i$ is the sub-goal end-effector pose at timestep t, which is represented as its 3D position, orientation, and the gripper finger opened width.

We propose a new representation for the high-level policy, termed the *weighted displacement model*. Existing 3D neural policies, e.g., Perceiver-Actor [41], or 3D Diffuser Actor [23], often generate the sub-goal end-effector pose in free SE(3) space. Instead, we aim to predict the sub-goal end-effector pose by grounding the prediction on the observed 3D structure of the scene. To do so, we design the policy to learn to predict the "offset" from observed points in the scene to the sub-goal end-effector pose. This learned offset thus closely grounds the prediction in the observed 3D scene structure. See Fig. 2 (middle) for an overview of the weighted displacement model.

Specifically, instead of representing the sub-goal end-effector pose as a position and an $SO(3)$ orientation (e.g., a quaternion or a 6D orientation representation [65]) and forcing the network to learn the connection between $SO(3)$ orientations and the 3D point cloud observation, we propose to represent the sub-goal end-effector pose as a collection of K points that are naturally in 3D. In our case, we use $K = 4$: the first point is located at the root of the robot hand, the second and third points at the parallel jaw gripper fingers, and the fourth point at the grasping center when the finger closes.

In this way, a sub-goal end-effector pose can be represented as $\{ee_i\}_{i=1}^4$, where ee_i is the 3D position of the i^{th} point. Given a point cloud of the scene with M points $P = \{p_j\}_{j=1}^M$ and the current robot end-effector points $\{ee_i^{obs}\}_{i=1}^4$, we propose to let the policy π_θ^H learn to predict the displacement from each point p_j in the scene point cloud to the sub-goal end-effector points $\{ee_i^{goal}\}_{i=1}^4$: $\delta_j = [\delta_j^1, \delta_j^2, \delta_j^3, \delta_j^4]$, where $\delta_j^i = ee_i^{goal} - p_j$. At inference time, the final predicted sub-goal end-effector pose is the averaged prediction from all points in the scene: $ee_i(\theta) = \sum_{j=1}^M (p_j + \delta_j^i(\theta))$. This proposed model converts the prediction of the end-effector pose from $SE(3)$, especially $SO(3)$, to a list of vectors just in the 3D space, which is less complex. We note this per-point prediction requires us to use a network architecture that can generate per-point outputs given a point cloud input. Many point cloud processing networks can do so [35, 36, 64]; we choose to use PointNet++ [36] in our case. As the model predicts the displacement from existing points in the scene instead of the absolute positions, and PointNet++ is a translation-invariant architecture, our proposed model is thus invariant to the translation of the robot end-effector and the object, which makes it more robust in real-world settings.

However, not all points in the scene are of equal importance for the task and for predicting the sub-goal end-effector pose. In the task of opening an articulated object (e.g., a cabinet), the points on the handle are probably more important compared to the points on the side of the cabinet. Therefore, we propose for the network to also learn a weight for each point in the scene point cloud when predicting the sub-goal end-effector pose. Formally, in addition to predicting the per point displacement δ_j, the network also predicts a weight w_j for each point p_j. At inference time, the final prediction of the sub-goal end-effector points is then the weighted average of the displacement from each point: $ee_i(\theta) = \sum_{j=1}^M w_j(\theta)(p_j + \delta_j^i(\theta)), i = 1, 2, 3, 4.$

We term this high-level policy representation the *weighted displacement model*. We train it with the following two losses, which supervises the per-point displacement prediction, and the weighted average prediction:

$$L = \lambda_1 \frac{1}{M} \sum_{j=1}^{M} ||\delta_j - \delta_j(\theta)||_2^2 + \lambda_2 \frac{1}{4} \sum_{i=1}^{4} ||ee_i^{goal} - ee_i(\theta)||_2^2 \tag{3}$$

$$ee_i(\theta) = \sum_{j=1}^{M} w_j(\theta)(p_j + \delta_j^i(\theta)) \tag{4}$$

Low-level Policy. The low-level policy π_θ^L learns to predict the delta transformation of the end-effector, given the observation o and the sub-goal end-effector pose $\{ee_i^{goal}\}_{i=1}^4$, i.e., "how" to actually move the end-effector to solve the task. It is learned to minimize the following loss:

$$\mathcal{L}_\theta = \sum_{i=1}^{N} \sum_{t=1}^{T} ||\pi_\theta^L(o_t^i, \{ee_k^{goal}\}_{k=1}^4\}) - a_t^i||_2^2, \tag{5}$$

where a_t^i is the delta transformation of the end-effector, including the delta translation, delta rotation, and delta finger movement (open/close). We represent the delta rotation using the 6D rotation representation [65]. We note that the low-level policy is not trained to reach the sub-goal end-effector pose; it is trained to solve the task, and the sub-goal end-effector pose is just an additional input that helps guide the low-level policy to learn how to move. Given that part of the demonstration trajectories are generated from a motion planner, which can be highly multi-modal, we employ a diffusion policy as the low-level policy representation, which is known for their ability to handle multi-modalities.

Specifically, we modify 3D Diffusion Policy (DP3) [60] such that it can be conditioned on the sub-goal end-effector pose. See Fig. 2 (bottom) for an overview of the low-level policy architecture. As in DP3, the network has two parts: a point cloud encoder that encodes the point cloud observation into a latent embedding, and a diffusion head on the latent embedding that generates the actions. We modify the encoder architecture to incorporate the sub-goal end-effector pose.

Formally, given the current scene point cloud observation $P = \{p_j\}_{j=1}^M$, the current end-effector pose represented with 4 points $\{ee_i^{obs}\}_{i=1}^4$, the sub-goal end-effector pose $\{ee_i^{goal}\}_{i=1}^4$, we treat each point as a token and perform attention among them to generate the final latent embedding. For the scene point cloud $P = \{p_j\}_{j=1}^M$, we use an MLP applied to each point in the point cloud to obtain a per-point feature $\{f_j\}_{j=1}^M$, which will be used as the features for cross attention later. For the current end-effector points $\{ee_i^{obs}\}_{i=1}^4$, its feature for attention includes the following: the first part is a learnable embedding v_i^{obs} for each of the 4 points. The second part is a feature vector produced by an MLP, where the input to the MLP includes each point's position ee_i^{obs}, the displacement to the corresponding point in the sub-goal end-effector pose $\delta_i = ee_i^{goal} - ee_i^{obs}$, and the displacement to the closest scene point:

	10 objs	50 objs	100 objs	200 objs	322 objs
With camera randomization	1116 121k	4656 502k	8749 958k	17893 1.97M	22918 2.55M
Without camera randomization	750 80k	3669 403k	6444 702k	11795 1.28M	15998 1.76M

TABLE I: Dataset Statistics. Top: # of trajectories. Bottom: total # of observation-action pairs in the trajectories.

$\delta'_i = p_k - ee_i^{obs}, k = \arg\min_j ||p_j - ee_i^{obs}||$. The final feature vector for each point is $f_i^{obs} = [v_i, \text{MLP}^{obs}(ee_i^{obs}, \delta_i, \delta'_i)]$. Intuitively, the displacement to the corresponding sub-goal points help the model to learn how to reach towards the goal; and the displacement to the closest scene points help the model to learn to avoid collision.

We then perform cross attention between the scene point cloud and the current end-effector points with Rotary Position Embedding (RoPE) [44], which generates the updated features for current end-effector points as $\{f_i^{\text{obs-scene}}\}_{i=1}^4$. We generate the features for the goal end-effector points in the same way as for the current end-effector points: $f_i^{goal} = [v_i^{goal}, \text{MLP}^{goal}(ee_i^{goal}, \delta_i, \delta'_i)]$. We perform cross attention between the current end-effector points and the goal end-effector points, also with Rotary Position Embedding (RoPE) [44], which produces another set of updated features for the current end-effector points $\{f_i^{\text{obs-goal}}\}_{i=1}^4$. The final latent embedding used for diffusion is the concatenation of the above two features: $[f_1^{\text{obs-scene}}, f_1^{\text{obs-goal}}, ..., f_4^{\text{obs-scene}}, f_4^{\text{obs-goal}}]$. This latent embedding is used as the conditioning for an action generation UNet diffusion head, which takes as input this latent conditioning, the robot state (which includes the 3D position, 6D orientation of the end-effector, and finger width), a noisy version of the action, a denoising time step, and predicts the noise. At test time, we use DDIM [42] as the denoising scheduler to generate the actions.

C. Zero-shot Transfer to Real Robotic Systems

After the high-level and low-level policies are trained in simulation, we transfer them zero-shot to real-world robotic systems. During inference, at each time step, given the current point cloud observation and end-effector pose, we first run the high-level policy to obtain a predicted goal end-effector pose, and then run the low-level policy with the point cloud observation, current end-effector pose, and the predicted goal end-effector pose to move the end-effector. We repeat this process until the object is fully opened, or a pre-defined episode length is reached, or the robot is going to collide with the environment. We discuss the details of our robot systems and real-world pipeline in Sec. VI.

V. Simulation Results

A. Experiment Setups

We use Pybullet [7] as the underlying physics simulator; any simulator that supports rigid-body dynamics and fast parallelization can be used. We use the PartNet-Mobility [55] dataset for the assets of the articulated object. We extracted 332 objects from 5 different categories: storage furniture, microwave, dishwashers, oven, and fridge, which have annotations for handles. Among these, 322 are used for training and 10 unseen objects are used for testing. For each object, we generate 75 demonstrations for opening it. Each demonstration has a different configuration, where we vary the position, orientation, size of the object, and the initial pose of the end-effector (randomization details in Appendix D).

In order to study the object-level generalization abilities of different methods, we first generated 15,998 demonstration trajectories with 1.76M observation-action pairs for these 322 training objects, without any camera randomizations when rendering the point clouds. For efficient sim2real transfer, we generate additional demonstrations with camera pose randomizations. The datasets with camera randomizations has in total 22,918 trajectories and 2.55M observation-action pairs. We partition both types of datasets into different sets, in which we vary the number of objects in each of these sets (objects and trajectories are randomly sampled) to study the scaling behavior of different methods. The detailed statistics of the partitioned datasets can be found in Table I.

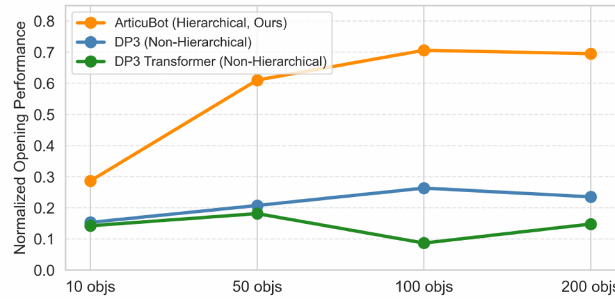

Fig. 3: Comparison of hierarchical and non-hierarchical policies.

For evaluation, we test each of the 10 objects with 25 different configurations, resulting in a total of 250 test scenarios. The evaluation metric is the **normalized opening performance**: the ratio of the increase in the opened joint angle of the object achieved by a method, to the increase in the opened joint angle of the object in the demonstration, which is calculated as $\frac{\theta_f - \theta_0}{\theta_{demo} - \theta_0}$, where θ_0 is the initial opened angle, θ_f is the final opened angle achieved by the method and θ_{demo} is the final opened angle in the demonstration. A value of 1 indicates that the method performs as well as the demonstration, while 0 means the method has not contributed to opening the object. For each method, we run the evaluation 3 times (a total of 750 trials) and report the mean and standard deviation of the normalized opening performances of the 3 runs. In the following, we compare to different baselines and prior methods to answer different research questions.

B. Is a Hierarchical Policy Needed?

Our first set of experiments aims to answer whether it is beneficial to use a hierarchical policy. We compare our proposed **hierarchical policy** with the following two non-hierarchical baselines:

- **3D Diffusion Policy (DP3)** [60], a diffusion policy that

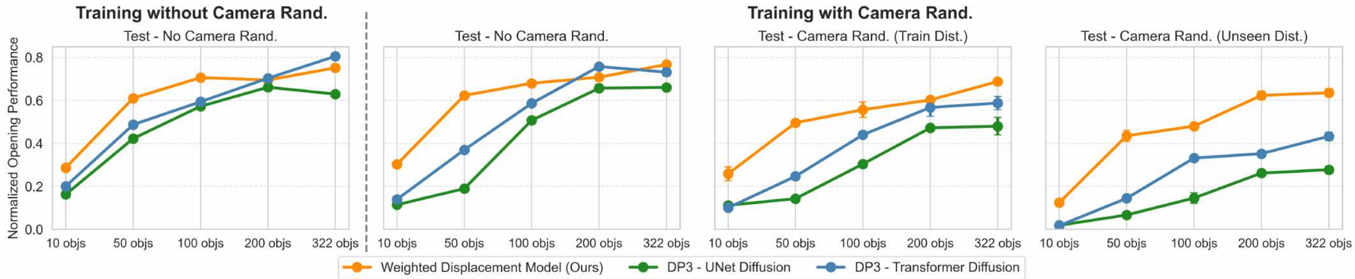

Fig. 4: Comparison of different high-level policies. Leftmost: Train and test without camera randomizations. Right: Train with camera randomizations, and test with no camera randomization, with camera randomizations from training distribution, and with camera randomizations from an unseen test distribution.

takes 3D point cloud as input and outputs delta end-effector transformations as the actions.

- **DP3 Transformer**, which replaces the simplified PointNet encoder in DP3 with a transformer-based encoder (the same one used in our low-level policy in Sec. IV-B).

We compare these two baselines with our method that uses a hierarchical policy on datasets without camera randomizations to study the object-level generalization abilities of them. The results are shown in Fig. 3. As shown, the performance of using a non-hierarchical policy only gets a normalized opening performance below 0.25, which is much lower than that of using a hierarchical policy. Furthermore, we observe that the non-hierarchical policies do not experience significant improvement in performance as the number of training objects and trajectories increase. These results show that it is very challenging to achieve object-level generalization if we just learn low-level delta end-effector transformations, regardless of how much training data we use; using a hierarchal policy achieves much better object-level generalization performances.

C. Comparison of Different High-level policies

We now investigate the performance of different high-level policy architectures. We compare our proposed **Weighted Displacement Model** to the following baselines:

- **DP3 - UNet Diffusion**: this baseline builds upon DP3 and diffuses the sub-goal end-effector points. We modify the simplified PointNet encoder in DP3 to an attention-based encoder (similar to our low-level policy), as we find this provides better performance in early experiments.
- **DP3 - Transformer Diffusion**: In addition to using the attention-based encoder, we also modify the UNet diffusion head in DP3 to be a transformer-based architecture, such that the diffusion head conditions on not only a latent embedding, but also the 3D point cloud features.
- **3D Diffuser Actor (3DDA)** [23]: this baseline also diffuses the sub-goal end-effector points conditioned on 3D point cloud features, but employs a different architecture compared to DP3 - Transformer Diffusion.

Please see Appendix G for more details about these baselines.

We use a fixed low-level policy for all experiments in this section. We first compare all method's performances when trained on the datasets without camera randomizations. The results are shown in the left subplot of Fig. 4. As shown, our proposed weighted displacement model performs consistently better than other methods when the number of training objects ranges from 10 to 100. When training with all 322 objects, DP3 - Transformer Diffusion achieves the best performance, outperforming the weighted displacement policy by 5%. We also find that all methods' performances generally improve as the size of the dataset increases (except for DP3 - UNet diffusion when the number of training objects increases from 200 to 322, and weighted displacement model when the number of training objects increases from 100 to 200). We trained 3DDA with 200 objects and found it to perform poorly, achieving a performance of only 0.135, much lower than the performance of alternative methods (> 0.6). Therefore, we omit the training of it on other datasets to save computation.

Since our primary focus is sim2real transfer of the learned policy to the real world, we also compare these methods on the dataset with camera randomizations, as it is hard to place the camera at the exact pose in the real world as in simulation, and we want the policy to be robust to camera pose changes. For evaluation, we have three different settings: test on a fixed camera pose, test on random camera poses sampled from the training distribution, and test on random camera poses sampled from a test distribution not seen during training. The results are shown in the right 3 subplots in Fig. 4. Interestingly, we find that when tested with camera randomizations, our proposed weighted displacement model performs much better than the compared methods, for all different sizes of training datasets. The performance gap is especially large when tested with unseen camera randomizations. Although DP3 - Transformer still achieves good performance when tested with a fixed camera pose with datasets more than 200 objects, its performance degrades drastically when tested with randomized cameras. In contrast, the performance drop for weighted displacement model when tested under camera randomizations is much smaller. We also find DP3 - UNet diffusion to perform poorly in this setting, which could be due to that it compresses the 3D scene into a single latent embedding vector, losing some of the needed 3D information for making the prediction. Similarly, we find that training with more data is generally helpful for achieving a higher performance.

D. Ablation Studies

In this subsection, we examine some of the design choices in our method to understand their contributions. We compare

Ablations (trained with 200 objs)	Normalized Opening Performance
ArticuBot (Ours)	$\mathbf{0.7 \pm 0.01}$
Weighted Displacement Model w/ Point Transformer	0.6 ± 0.02
Unweighted Displacement Model	0.66 ± 0.01
Weighted Displacement Model w/ 6D orientation	0.53 ± 0.01
Replacing low-level policy with a motion planner	0.24 ± 0.02

TABLE II: Performance of different ablation studies.

our full method to the following ablations:

- **Weighted Displacement Model w/ Point Transformer [64]**: instead of using PointNet++, we use the Point Transformer architecture for the weighted displacement high-level policy.
- **Unweighted Displacement Model**: This ablation does not learn a weight for each point in the weighted displacement model; instead, the prediction is simply the average of all point's predictions.
- **Weighted Displacement with 6D orientation:** Instead of predicting the offset to the 4 goal end-effector points, this ablation predicts the 3D offset to the goal end-effector position, and the 6D orientation of the goal end-effector, from each scene point. The final prediction is the average of each point's prediction.
- **Replacing low-level policy with a motion planner:** This ablation does not use a low-level policy for moving the robot end-effector. We first predict a goal end-effector pose for grasping using the high-level policy and then use a motion planner to reach it. After grasping, we run the high-level policy again to predict a goal end-effector pose for opening the door. We compute the corresponding joint angles using inverse kinematics and use a joint PD controller to reach it.

More details of these ablations can be found in Appendix I. We compare to these ablations when training with 200 objects without camera randomizations. The results are shown in Table II. We find that using a Point Transformer, not predicting the per-point weights, or predicting a per-point goal end-effector 6D orientation instead of per-point displacements to the goal end-effector points, all lead to worse performance, supporting the effectiveness of our design choices in ArticuBot. Predicting a per-point 6D orientation and averaging them leads to a large drop in performance because it is difficult to correctly compute the average of multiple 6D orientations; in ArticuBot, we avoid this by representing the goal end-effector pose as a collection of points and averaging the displacement to these points. Replacing the low-level policy with motion planning and an IK controller results in very poor performance. We hypothesis it could be due to two reasons: 1. The high-level policy may predict poses with minor collisions, and motion planning often fails due to the inability to find collision-free paths. 2. During door-opening, the joint PD controller takes the shortest joint-space path to the goal pose, ignoring necessary kinematic constraints (e.g., following an arc to open a revolute door), causing the gripper to slip off. Experimentally, the motion planning failure rate is 17%. Among successful grasps, 97% of the failures are due to the gripper falling off the handle during opening (possibly due to ignoring object kinematic constraints). This shows the importance of using a learned low-level policy.

Method	Grasping Sucess Rate	Normalized Opening Performance
ArticuBot (Ours)	$\mathbf{0.88 \pm 0.01}$	$\mathbf{0.75 \pm 0.01}$
AO-Grasp	0.11 ± 0.0	0.08 ± 0.0
ArticuBot (Ours), After Grasping	-	$\mathbf{0.86 \pm 0.01}$
FlowBot3d - w/o Mask, After Grasping	-	0.2 ± 0.01
FlowBot3d - w/ Mask, After Grasping	-	0.57 ± 0.01

TABLE III: Comparison with prior articulated object manipulation methods.

E. Comparison with Prior Articulated Object Manipulation Methods

We also compare our system with prior methods that aim to learn a single policy for generalizable articulated object manipulation. We compare to two state-of-the-art prior methods that focus on each stage of manipulating an articulated object:

- **AO-Grasp** [32]: this method focuses on the grasping of the articulated objects for downstream manipulation. It learns an Actionable Grasp Point Predictor that predicts the grasp-likelihood scores for each point in the point cloud, which is combined with pretrained ContactGraspNet [45] to generate 6D grasps.
- **FlowBot3D** [10]: this method predicts the flow for each point on the articulated object, and moves the robot end-effector along the maximal flow direction to open the object. The original paper performs grasping by using a suction gripper to attach the robot end-effector to the maximal flow point on the object's surface.

We compare ArticuBot with AO-Grasp in terms of **grasping success rate**, i.e., if the method generates a firm grasp of the object that enables downstream manipulation. As AO-Grasp only generates a 6D grasp pose, we use motion planning to move the robot end-effector to reach the grasping pose. Although AO-Grasp does not open the object, we still compare with it in terms of **normalized opening performance** in the following way: After grasping, we assume access to ground-truth articulation information of the object and use our designed opening action (See Sec. IV-A) to open the object. Note such information is not available in the real world, and ArticuBot also does not use such information in the learned policy. We compare with FlowBot3D in terms of **normalized opening performance after grasping**: starting from the state where the robot gripper has already firmly grasped the handle of the object, how well does the method open the object. We used pre-trained checkpoints provided by the authors of AO-Grasp and FlowBot3D for the comparison.

The results are shown in Table III, tested without camera randomizations. The grasping success rate of AO-Grasp is much lower than ArticuBot. We find that AO-Grasp often proposes grasps at the edges of the point cloud (e.g., the side wall of a drawer; see Appendix H for visuals). This likely stems from its training data, which includes many objects in a partially opened state where edge grasps are valid. However, in

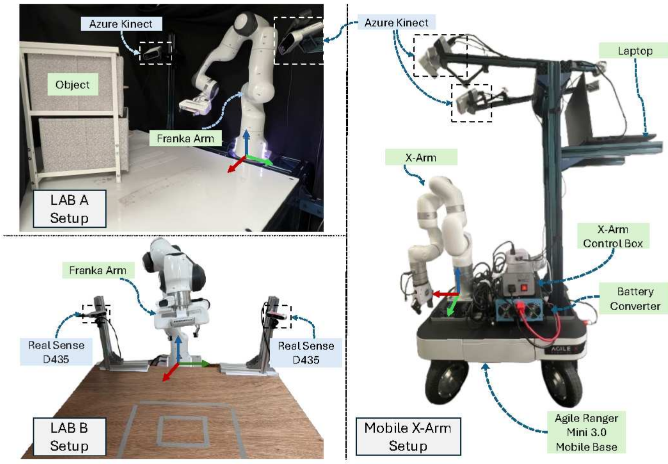

Fig. 5: The three different real robot setups.

Fig. 6: Real-world test objects for table-top and mobile-base experiments.

our test cases, objects are usually closed or not open enough for such grasps. Additionally, many detected edges are fake edges that result from partial observations from the camera rather than true graspable edges. For FlowBot3D, we find its performance to be reasonable (0.57) when provided with the segmentation mask of the target link (door or drawer) to open, and much lower (0.2) without such masks. Note that ArticuBot does not use a segmentation mask for the target link. To form a fair comparison, we also evaluated our policy's performance after grasping. In such a case, the performance of ArticuBot further improved from 0.75 to be 0.86 (See Table III), outperforming FlowBot3D by a large margin.

F. Comparison of Different Low-level Policies

We also performed experiments to test the performance of different low-level policy architectures (e.g., using a different diffusion head, or using a different action space). The detailed results and analysis can be found in Appendix C. In summary, we do not observe huge performance differences (within 5%) between these different methods. Our hypothesis is that the goal end-effector points provide a strong conditioning for the low-level policy; with such information as input, the differences in the policy architectures and action spaces may not matter too much.

G. Additional Experiments and Evaluations

We show some preliminary experiments in Appendix E that our hierarchical policy learning approach works on more manipulation tasks beyond articulated object manipulation. We also study how robust the policy is to the orientation and position of the handles, with the results shown in Appendix F.

VI. Real-world Experiments

A. Setups

We deploy our learned policies to three different real robot settings: table-top fixed Franka arms in two different labs, and an X-Arm on a moble base in real lounge and kitchens, to test its robustness and generalization ability in the real world. We note that our policy in simulation is only trained on the Franka arm. The policy can transfer zero-shot cross embodiment to an X-Arm because the policy learns actions in the robotic arm's end-effector space (sub-goal end-effector pose and end-effector delta-transformations) instead of the joint space.

For robust sim2real transfer, we generate more demonstrations in simulation with augmentations on the point cloud observations to make the policy robust to noisy point clouds obtained from real-world depth sensors. Specifically, we add the following two augmentations to the depth map in simulation [8]: the first is edge artifacts that models the noise along object edges, and the second is random holes in the depth map to model random depth pixel value loss in real-world depth cameras. Details of these augmentations can be found in the Appendix J. We also randomize the camera poses closer to where they are located in the real world. Combined with the non-augmented demonstrations, we generated in total 42.3k trajectories with 4.7M observation-action pairs, and trained a single weighted displacement model high-level policy on this dataset. We find the low-level policy to transfer well without needing to be trained on such point cloud augmentations. We detail the 3 different robot setups as below, visualized in Fig. 5.

Table-top Franka Panda Arm in Lab A: The first setup has a fixed-base table-top Franka Arm. The table has a length and width of 110 cm. The robot arm is located near one corner of the table. We use two Azure Kinect cameras, each mounted on one side of the robot looking at the center of the table, to get the point cloud of the objects. The robot is controlled via the Deoxys library [66] with a joint position controller, i.e., given a target pose, we first use a IK solver to obtain the target joint angle, and then use the joint position controller to reach that joint angle. We test 9 different articulated objects, including drawers, cabinets, microwaves, toy oven and toy fridges in this workspace (as shown in Fig. 6), all purchased from local stores and not seen during training.

Table-top Franka Panda Arm in Lab B: We also deploy our policy in a different lab to more thoroughly test its robustness in a different setting. The table used in this lab has a width and length and width of 100cm and 80cm. The robot is placed at the center of one edge of the table. Two Intel RealSense D-435 RGBD cameras, one mounted on each side of the

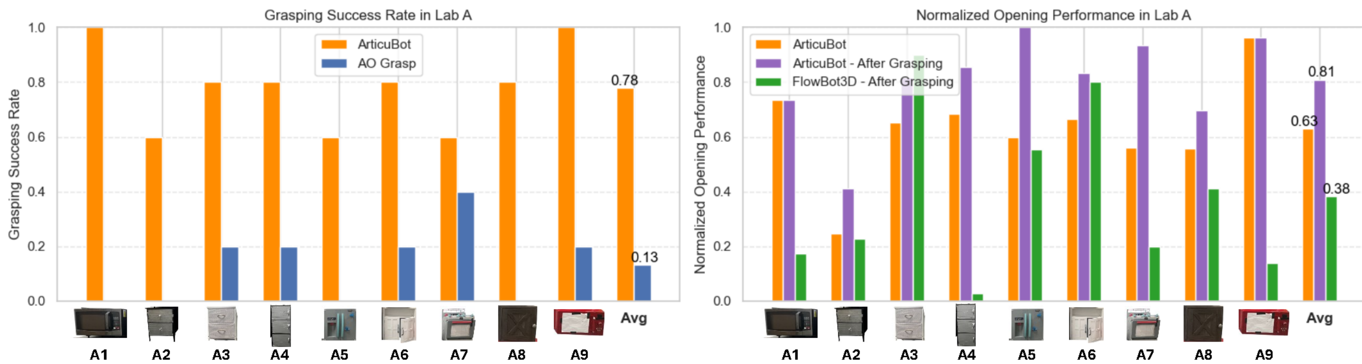

Fig. 7: Comparison of ArticuBot with FlowBot3D and AO-Grasp on 9 test objects in Lab A with table-top Franka. We omit OpenVLA in the plot as it achieves a performance of 0.

robot, are used to capture the objects' point clouds. The robot is controlled using a end-effector position controller. Four different objects are tested in this workspace, shown in Fig. 6, all purchased from local stores and not seen during training.

X-Arm on a mobile base: To test our policy in real lounges and kitchens, we additionally build a mobile manipulator, where we assemble an X-Arm onto a Ranger Mini 3.0 mobile base, following the design in Xiong et al. [56]. We mount two Azure Kinects on manually built frames on the mobile base for capturing point cloud observations (see Fig. 5). We use the company-provided end-effector position control python API for controlling the X-Arm. We test this mobile X-Arm in 4 different kitchen, lounge and offices on 7 objects (See Fig. 6). The X-Arm and both Azure Kinects are connected to a Lenovo Legion Pro 7 Laptop. The laptop has a built-in NVIDIA GeForce RTX 4090 GPU, which is used for running the trained policies. The X-Arm, the Azure Kinects, and the laptop are all powered by the battery that comes with the Ranger Mini 3.0 mobile base, which gives a 48V DC output, and we use a bettery inverter to convert it to a standard 120V AC output to power these devices.

For both table-top settings, the objects are placed near the center of the table with some variations in the position and orientation, such that the robot arm can open it as much as possible within its joint limits. In Lab A, the Franka arm is randomly initialized at one of two fixed locations, one with the end-effector closer to the table, and the other with the end-effector higher in the air. In Lab B, the arm is randomly initialized such that the end-effector is 30 to 60 centimeters away from the object. For the mobile X-Arm, we manually tele-operate the base to be near the target object, and the base remains fixed when the X-Arm is opening the object. We randomly initialize the X-Arm at different joint angles.

We perform camera-to-robot base calibration in all settings, and all point cloud observations are transformed into the robot's base frame. We use GroundingDino [29] and EfficientSAM [57] to segment the object given a text of the object name, and obtain an object-only point cloud in the real world. For removing the robot from the point cloud, we first render a canonical robot point cloud from the robot urdf and mesh files, transform it to the current point cloud observation using the robot's current joint angles, and then project it to the 2D depth image using known camera extrinsics and intrinsics to obtain a robot mask. All pixels within the robot mask are removed. We perform additional radius and statistical outlier removing to remove some remaining outlier points from the noisy depth cameras. More details of the real-world perception pipeline can be found in Appendix K.

Robot Test Settings	Grasping Sucess Rate	Normalized Opening Performance
ArticuBot - Tabletop Franka Lab A	0.78	0.63
ArticuBot - Tabletop Franka Lab B	0.85	0.59
ArticuBot - Mobile X-Arm	0.90	0.54

TABLE IV: Performance of ArticuBot under all three robot setups.

We use the following evaluation metrics as in simulation. **Grasping Success Rate**: We manually check if the robot gripper has a firm grasp of the object. **Normalized Opening Performance**: The opened distance of the object normalized by the maximal achievable opening distance of the object, subject to the workspace and robot joint limit constraints.

We compare to the following baselines with the table-top Franka Arm in lab A, on 9 test objects. **OpenVLA**: This is an open-sourced robotic foundation policy trained on Internet scale of real-world datasets. It takes a language instruction as input and output robot actions. A small portion of the datasets contain articulated object manipulation tasks. We also compare to **AO-Grasp** and **FlowBot3D** as described in Sec. V-E. We compare with AO-Grasp in terms of grasping success rate: we use our learned low-level policy to reach the grasping pose generated by AO-Grasp and manually check if the grasping is successful. We compare with FlowBot3D in terms of normalized opening performance: we first manually move the end-effector to grasp the handle of the object, and then apply FlowBot3D to open the object. We do not input the optional segmentation mask for the target link to open for FlowBot3D, as such masks are not readily available in the real world, and ArticuBot also does not use such masks. For table-top Franka arms, we run each method on each object for five trials and report the mean performance. For mobile X-Arm, we run ArticuBot for three trials on each object.

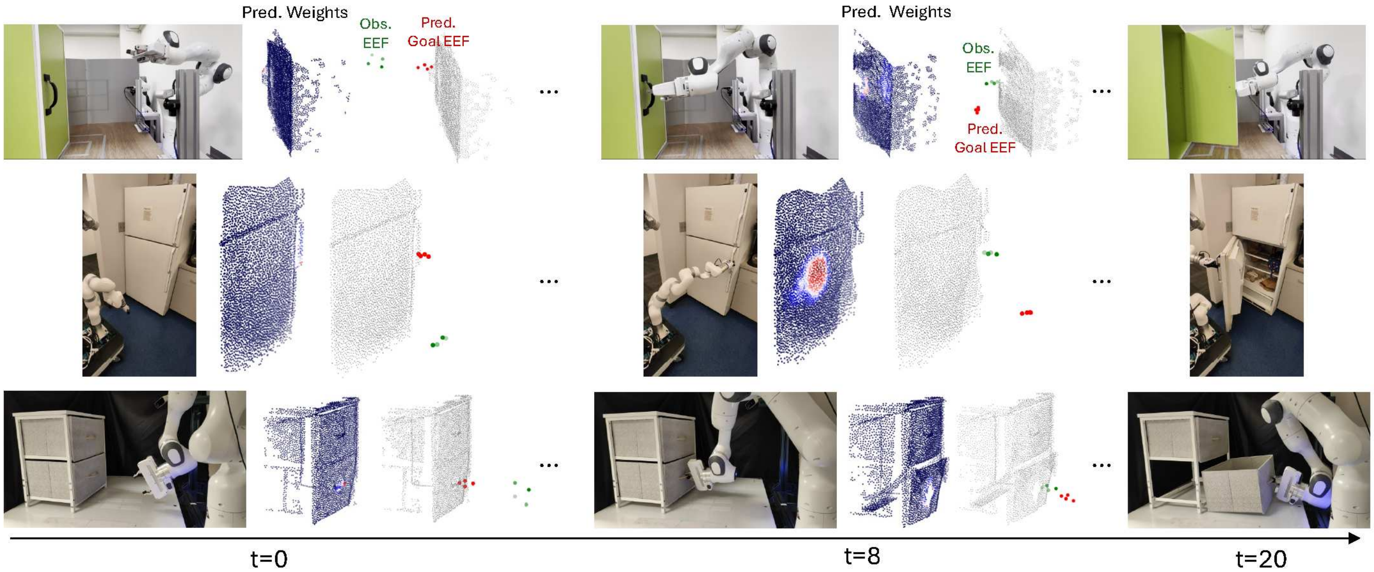

Fig. 8: Visualizations of the high-level policy's predictions (per-point weights and goal end-effector points) in three of the real-world test cases. The green points represent the observed current end-effector points, and the red points represent the predicted goal end-effector points.

B. Table-Top Franka Arm Results

The results for all test objects and compared methods in lab A are shown in Fig. 7; the results of ArticuBot in lab B are shown in Table IV. ArticuBot achieves an average grasping success rate of 0.78 and 0.85, and a normalized opening performance of 0.63 and 0.59 in Lab A and B, respectively, showing that it can generalize to open diverse real articulated objects with varying geometries, shapes and articulation types across both lab settings. We note that some of the test objects are quite challenging and require very precise manipulation, e.g., the knobs of object A2 and A8 are very small, with a diameter of only 2 cm, but ArticuBot can still precisely grasp and open it. As in simulation, we find the grasping success rate of AO-grasp to be low, where it tends to grasp at the "fake" edge in the point cloud due to partial observations, which are not actually graspable (See Appendix H for visuals of the grasps produced by AO-Grasp). FlowBot3D achieves a reasonable normalized opening performance of 0.38, starting from the state where the robot gripper already grasps the object. The performance is still lower than ArticuBot, even though ArticuBot performs the additional grasping step. The major failure case for FlowBot3D is that the predicted flow is in the wrong direction, e.g., it predicts upwards flows for opening a microwave (See Appendix H for visuals of the flows). If we compute the normalized opening performance for ArticuBot only in cases where the grasp is successful (i.e., the same starting conditions as FlowBot3D), the performance of ArticuBot further improves to 0.81 and outperforms FlowBot3D by a large margin (See Fig. 7). We also find OpenVLA fails to grasp or open any test objects, resulting in a grasping success rate and normalized opening performance of 0. This is likely due to its training data lacking sufficient demonstrations of articulated object manipulation, making generalization to our test cases difficult.

Fig. 8 (zoom-in for better views) visualizes ArticuBot's predictions on some of the real-world test objects. As shown, the learned per-point weights from the weighted displacement model concentrates on the handle of the object before grasping, and the predicted grasping end-effector pose is quite accurate on the handle, even though for most of the objects the handles are just a very small portion of the point cloud. We note that there is no explicit supervision for the model to learn to assign high weights to the handle; this is automatically learned by just minimizing the imitation learning loss. After grasping, the weights are more randomly distributed across the objects; but as shown in Fig. 8, ArticuBot generalizes to predict different opening end-effector poses for objects with different articulations (left opening revolute joints, right opening revolute joints, pulling out prismatic joints). We do notice a drop in performance compared to the results in simulation. We believe this is mostly due to the noisier point cloud observation from the depth sensors in the real world (e.g., see the noisy point cloud for the green cabinet obtained from the RealSense cameras in Fig. 8). See Fig. 1 for screenshots of the opening trajectories for more objects (and Appendix A for trajectories of all objects); please refer to the supplementary materials for videos. Common failure cases for table-top experiments include: 1. The robot arm runs into joint limits while opening the object, due to the limited space of the robot workstation. 2. Wrong end-effector pose predictions for grasping the handle, which we find to happen more for objects with small handles, e.g., A2. Appendix L provides visualizations of some of the failure cases.

C. Mobile X-Arm Results

Table IV shows the results with the mobile X-Arm. As shown, ArticuBot achieves a grasping success rate of 0.9 and normalized opening performance of 0.54, showing it can generalize to drawers, cabinets, and fridges in real kitchens and lounges. See Fig. 8 for a visualization of the high-level policy predictions from ArticuBot on a real-world fridge. See Fig. 1 for screenshots of some of the successful opening trajectories with the mobile X-Arm (and Appendix A for trajectories of all objects); please refer to the supplementary materials for videos. Some of the tested real-world objects are quite challenging, for example, cabinet C3 has a very small handle that protrudes only 2 cm from the surface, but ArticuBot is still capable of precisely grasping and opening it.

We do notice a drop in the normalized opening performance compared to the table-top Franka experiments. In our early experiments, we find that Unlike the Franka Arm, the X-Arm lacks impedance control and force sensing. This requires a more precise prediction for the opening end-effector pose. Small prediction errors, such as turning too sharply when opening a revolute door, would result in excessive force for the X-Arm and causes it to stop for motor protection, which is a common failure case in this setting. To partially mitigate this issue, we use the Fast-UMI [54] gripper in latter experiments, whose soft, compliant design enhances safety and partially helps prevent the arm from stopping due to excessive force. Another sim2real gap is that, in simulation, objects are isolated by itself; in real kitchens and lounges, objects are usually surrounded by other objects, which occlude its side and top, and only the front side is observable. This additional occlusion might have caused issues for the policy to transfer as well. Finally, the articulation of some objects are inherently ambiguous to judge from a single point cloud observation, e.g., although many dishwashers have a revolute door that open downwards, the dishwasher C2 in Fig. 6 has a prismatic joint that needs to be pulled out horizontally. ArticuBot tends to make more mistakes on such ambiguous objects. See Appendix L for visualizations of some of the failure cases of ArticuBot, and some basic failure recovery abilities of ArticuBot.

VII. Limitations

Our system currently has the following limitations: 1) Our weighted displacement policy does not handle multi-modal outputs since it is trained with a regression loss (Eq. (3)). This may create issues for it when working with cabinets with multiple doors and opening only a specific one is desired. 2) The current system does not support opening a user-specified door on a multi-door object, as the policy is not trained to be conditioned on any user input. Although our training data includes multi-door objects, demonstrations are generated for opening the closest door to the initial pose of the robot. The policy learns to open the closest door implicitly, rather than opening a user specified door. 3) The policy uses point cloud observations. Although this simplifies sim2real transfer, existing depth sensors in the real world do not work well for transparent or reflective objects, and thus our policy would also fail on such objects in the current form. 4) As mentioned above, the X-Arm itself does not support force sensing and impedance control, which requires more precise policy predictions to avoid excessive force. We think adding a force-torque sensor on the X-Arm to enable impedance control could help alleviate this issue; fine-tuning the policy in the real-world via reinforcement learning or a few demonstrations for more precise sub-goal end-effector pose predictions could also help. 5) Objects in real kitchens and lounges are usually occluded by neighboring objects, and we believe that adding this type of occlusion could further improve the sim2real performance of the policy. 6) ArticuBot does not use interaction history during the manipulation process. We think that incorporating interaction history with current visual observations could further improve performance, especially for objects whose articulation are ambiguous to judge just from visual observations. We leave addressing these limitations as important future work.

VIII. Conclusion

This paper presents ArticuBot, a robotics system powered by a single learned policy that is able to open diverse categories of unseen articulated objects in the real world. ArticuBot consists of three parts: generating a large number of demonstrations in simulation, distilling all generated demonstrations into a point cloud-based neural policy via imitation learning, and performing zero-shot sim2real transfer. Using sampling-based grasping and motion planning, ArticuBot's demonstration generalization pipeline is fast and effective, generating a total of 42.3k demonstrations over 322 training articulated objects. For policy learning, ArticuBot uses a novel hierarchical policy representation, in which the high-level policy learns the sub-goal for the end-effector, and the low-level policy learns how to move the end-effector conditioned on the predicted goal. A novel weighted displacement model is used for the high-level policy that grounds the prediction into the existing 3D structure of the scene, outperforming alternative policy representations. Our learned policy can zero-shot transfer to three different real robot settings: a fixed table-top Franka arm across two different labs, and an X-Arm on a mobile base, opening multiple unseen articulated objects across two labs, real lounges, and kitchens.

Acknowledgments

This material is based upon work supported by the Toyota Research Institute, National Science Foundation under NSF CAREER Grant No. IIS-2046491, and NIST under Grant No. 70NANB24H314. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Toyota Research Institute, National Science Foundation, or NIST.

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[3] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[4] Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

[5] Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv preprint arXiv:2405.11656*, 2024.

[6] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11443–11450. IEEE, 2024.

[7] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[8] Murtaza Dalal, Min Liu, Walter Talbott, Chen Chen, Deepak Pathak, Jian Zhang, and Ruslan Salakhutdinov. Local policies enable zero-shot long-horizon manipulation. *arXiv preprint arXiv:2410.22332*, 2024.

[9] Murtaza Dalal, Jiahui Yang, Russell Mendonca, Youssef Khaky, Ruslan Salakhutdinov, and Deepak Pathak. Neural mp: A generalist neural motion planner. *arXiv preprint arXiv:2409.05864*, 2024.

[10] Ben Eisner, Harry Zhang, and David Held. Flowbot3d: Learning 3d articulation flow to manipulate articulated objects. *arXiv preprint arXiv:2205.04382*, 2022.

[11] Haritheja Etukuru, Norihito Naka, Zijin Hu, Seungjae Lee, Julian Mehu, Aaron Edsinger, Chris Paxton, Soumith Chintala, Lerrel Pinto, and Nur Muhammad Mahi Shafiullah. Robot utility models: General policies for zero-shot deployment in new environments. *arXiv preprint arXiv:2409.05865*, 2024.

[12] Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu. Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 2023.

[13] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *Conference on Robot Learning*, pages 967–977. PMLR, 2023.

[14] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[15] Arjun Gupta, Michelle Zhang, Rishik Sathua, and Saurabh Gupta. Opening cabinets and drawers in the real world using a commodity mobile manipulator. *arXiv preprint arXiv:2402.17767*, 2024.

[16] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[17] Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE international conference on robotics and automation*, pages 2493–2498. IEEE, 2010.

[18] Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. *arXiv preprint arXiv:2410.03645*, 2024.

[19] Ajinkya Jain, Rudolf Lioutikov, Caleb Chuck, and Scott Niekum. Screwnet: Category-independent articulation model estimation from depth images using screw theory. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13670–13677. IEEE, 2021.

[20] Ajinkya Jain, Stephen Giguere, Rudolf Lioutikov, and Scott Niekum. Distributional depth-based estimation of object articulation models. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1611–1621. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/jain22a.html.

[21] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5616–5626, 2022.

[22] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[23] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

[24] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[25] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[26] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal

locomotion over challenging terrain. *Science robotics*, 5 (47):eabc5986, 2020.

[27] Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. *arXiv preprint arXiv:2410.18647*, 2024.

[28] Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. Paris: Part-level reconstruction and motion analysis for articulated objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 352–363, 2023.

[29] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pages 38–55. Springer, 2025.

[30] Zhao Mandi, Yijia Weng, Dominik Bauer, and Shuran Song. Real2code: Reconstruct articulated objects via code generation. *arXiv preprint arXiv:2406.08474*, 2024.

[31] Kaichun Mo, Leonidas J Guibas, Mustafa Mukadam, Abhinav Gupta, and Shubham Tulsiani. Where2act: From pixels to actions for articulated 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6813–6823, 2021.

[32] Carlota Parés Morlans, Claire Chen, Yijia Weng, Michelle Yi, Yuying Huang, Nick Heppert, Linqi Zhou, Leonidas Guibas, and Jeannette Bohg. Ao-grasp: Articulated object grasp generation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13096–13103. IEEE, 2024.

[33] Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[34] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[36] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[37] Haozhi Qi, Brent Yi, Sudharshan Suresh, Mike Lambeta, Yi Ma, Roberto Calandra, and Jitendra Malik. General in-hand object rotation with vision and touch. In *Conference on Robot Learning*, pages 2549–2564. PMLR, 2023.

[38] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*, 9(89):eadi9579, 2024.

[39] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.

[40] Daniel Seita, Yufei Wang, Sarthak J Shetty, Edward Yao Li, Zackory Erickson, and David Held. Toolflownet: Robotic manipulation with tools via predicting tool flow from point clouds. In *Conference on Robot Learning*, pages 1038–1049. PMLR, 2023.

[41] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

[42] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[43] Marlin P Strub and Jonathan D Gammell. Advanced bit*(abit*): Sampling-based planning with advanced graph-search techniques. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 130–136. IEEE, 2020.

[44] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[45] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[46] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality. *arXiv preprint arXiv:2305.17110*, 2023.

[47] Bingjie Tang, Iretiayo Akinola, Jie Xu, Bowen Wen, Ankur Handa, Karl Van Wyk, Dieter Fox, Gaurav S Sukhatme, Fabio Ramos, and Yashraj Narang. Automate: Specialist and generalist assembly policies over diverse geometries. *arXiv preprint arXiv:2407.08028*, 2024.

[48] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[49] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023.

[50] Yian Wang, Ruihai Wu, Kaichun Mo, Jiaqi Ke, Qingnan Fan, Leonidas J Guibas, and Hao Dong. Adaafford: Learning to adapt manipulation affordance for 3d artic-

ulated objects via few-shot interactions. In *European conference on computer vision*, pages 90–107. Springer, 2022.

[51] Yufei Wang, Zhanyi Sun, Zackory Erickson, and David Held. One policy to dress them all: Learning to dress people with diverse poses and garments. *arXiv preprint arXiv:2306.12372*, 2023.

[52] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.

[53] Ruihai Wu, Yan Zhao, Kaichun Mo, Zizheng Guo, Yian Wang, Tianhao Wu, Qingnan Fan, Xuelin Chen, Leonidas Guibas, and Hao Dong. Vat-mart: Learning visual action trajectory proposals for manipulating 3d articulated objects. *arXiv preprint arXiv:2106.14440*, 2021.

[54] Ziniu Wu, Tianyu Wang, Chuyue Guan, Zhongjie Jia, Shuai Liang, Haoming Song, Delin Qu, Dong Wang, Zhigang Wang, Nieqing Cao, et al. Fast-umi: A scalable and hardware-independent universal manipulation interface. *arXiv preprint arXiv:2409.19499*, 2024.

[55] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020.

[56] Haoyu Xiong, Russell Mendonca, Kenneth Shaw, and Deepak Pathak. Adaptive mobile manipulation for articulated objects in the open world. *arXiv preprint arXiv:2401.14403*, 2024.

[57] Yunyang Xiong, Bala Varadarajan, Lemeng Wu, Xiaoyu Xiang, Fanyi Xiao, Chenchen Zhu, Xiaoliang Dai, Dilin Wang, Fei Sun, Forrest Iandola, et al. Efficientsam: Leveraged masked image pretraining for efficient segment anything. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16111–16121, 2024.

[58] Zhenjia Xu, Zhanpeng He, and Shuran Song. Universal manipulation policy network for articulated objects. *IEEE robotics and automation letters*, 7(2):2447–2454, 2022.

[59] Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, Zhiao Huang, Chuang Gan, and Shuran Song. Roboninja: Learning an adaptive cutting policy for multi-material objects. *arXiv preprint arXiv:2302.11553*, 2023.

[60] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Robotics: Science and Systems*, 2024.

[61] Vicky Zeng, Tabitha Edith Lee, Jacky Liang, and Oliver Kroemer. Visual identification of articulated object parts. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2443–2450. IEEE, 2021.

[62] Harry Zhang, Ben Eisner, and David Held. Flowbot++: Learning generalized articulated objects manipulation via articulation projection. *arXiv preprint arXiv:2306.12893*, 2023.

[63] Yuanhang Zhang, Tianhai Liang, Zhenyang Chen, Yanjie Ze, and Huazhe Xu. Catch it! learning to catch in flight with mobile dexterous hands. *arXiv preprint arXiv:2409.10319*, 2024.

[64] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[65] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753, 2019.

[66] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors. *arXiv preprint arXiv:2210.11339*, 2022. doi: 10.48550/arXiv.2210.11339.

[67] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. *arXiv preprint arXiv:2309.05665*, 2023.