Supplementary Material for PartManip: Learning Cross-Category Generalizable Part Manipulation Policy from Point Cloud Observations

Haoran Geng [1,2*] Ziming Li [1,2*] Yiran Geng [1,2] Jiayi Chen [1,3] Hao Dong [1,2] He Wang[1,2†]
[1]CFCS, Peking University [2]School of EECS, Peking University
[3]Beijing Academy of Artificial Intelligence
https://pku-epic.github.io/PartManip

1. More Details about Our Benchmark

1.1. Objects and Robot Assets

We select 494 object instances from GAPartNet [11] dataset. GApartNet provides large-scale articulated objects with rich annotations. For doors and drawers, our benchmark requires a handle on it. For the button, there is no constraint. Because there are too many buttons on some remote or phone, we randomly select a maximum of 5 buttons on each object. The total benchmark contains 494 objects and 1432 different target parts.

1.2. Environment Settings

We carefully set the environment parameters in Isaacgym. To simulate real-world physics, we set the contact offset = 1e-3, which means that our gripper is harder to manipulate the part by scratching or rubbing the edge of the part. Also, we set a slight recovery force (= 0.1) to avoid the agent moving the part to a successful state by slightly touching it. The stiffness of the cabinet dof(slider joint for drawer and button, rotation joint for door) is set to 20. The damping of the cabinet dof is set to 200. The friction coefficient of cabinet dof is set to 5.

For robot controlling, we use pos control mode. This means that we need to input each joint angle of the Franka arm to the network at each step. We find that in our tasks, using pos control is easier for imitation learning.

1.3. Reward Weight

For each task, we tune the reward weight to train a human-like policy. We list rotation weight λ_r , handle distance weight λ_d, part moving weight λ_p, and tips closure weight λ_t in Table 2.

rotation weight. For the closing task and pressing button task, we don't need the gripper to grasp the handle, so we set $\lambda_r = 0$. For the other tasks, we set it to 0.2.

handle distance weight. The value of $\frac{\lambda_d}{\lambda_p}$ is important for opening task. If the value is too small, the policy wouldn't learn to open the part using the handle. If the value is too large, the policy wouldn't try to open the door, but only learn to grasp the handle. The λ_{df} is set to non-zero for the opening task because after the part is opened, the gripper can move away from the handle.

part moving weight. For the grasping task, we don't require the agent to manipulate part, so we set $\lambda_p = 0$.

tip closure weight. For the opening and closing tasks, we focus on using the handle to finish the task but don't require the final grasp pose of the gripper. So we set $\lambda_t = 0$.

1.4. Initialization and Success Criteria

For each task, we initialize the gripper at a certain distance (*i.e.*, 50cm) away from the center of the target part, facing the object. The initialization of objects and the success criteria for each task are shown below:

OpenDoor: The door should be opened to more than 30 degrees from the initial closed state.

OpenDrawer: The drawer should be opened to more than 20% of the maximum opening length from the initial closed state.

CloseDoor: The door should be closed to less than 1 degree from the initial opening angles of 45 degrees.

CloseDrawer: The drawer should be closed from the initial opening length of 30 cm to less than 1 cm.

PressButton: The button should be pressed for more than 50% of the maximum pressing depth.

GraspHandle: The robot should close two tips to less than a threshold from different sides of the handle, while the center of the two tips is inside the handle bounding box.

2. More Details about Our Method

Our pseudo code is shown in algorithm 1 and algorithm 2.

For state-based policy training, we update E epochs in one step. In one epoch, the state and actions buffer D_{π_θ} is divided in B minibatch to compute one gradient step. So after sampling once, the network update $E * B$ time.

Algorithm 1 State-based Expert Training

Input: robot states S, handle bounding box b_{handle}, part bounding box b_{part}, state $s = (S, b_{part}, b_{handle})$, policy network θ_p (*i.e.*, actor θ_p^a and critic θ_p^c),
for $t = 0, 1, 2 \ldots$ **do**
  Transfer observation to canonical space
  Sample trajectories $D_{\pi_\theta} = \{(s_i.a_i)\}_{i=1}^n$
  for $e = 1, 2, \ldots, E$ **do** ▷ PPO update
    for $b = 1, 2, \ldots, B$ **do**
      update policy network θ_p, according to: $\mathcal{L}_{RL}$
Select highest success rate $\theta_p \rightarrow \theta_{\text{expert}}$

while $|D_{demo}| <$ buffer size **do**
  sample trajectory t_i by θ_{expert}
  append t_i to D_{demo}

Algorithm 2 Vision-based Student Training

Input: partial point cloud $P \in \mathbb{R}^{N\times 3}$, robot states s, vision backbone θ_b, policy network θ_p (*i.e.*, actor θ_p^a and critic θ_p^c), expert policy θ_{expert}, demonstration buffer D_{demo}
Pre-training: update the vision backbone θ_b and actor MLPs θ_p^a, according to $\mathcal{L}_{\text{BC}}$
for $t = 0, 1, 2 \ldots$ **do**
  Sample trajectories $D_{\pi_\theta} = \{(s_i, o_i, a_i)\}_{i=1}^n$
  for $e = 1, 2, \ldots, E$ **do**
    for $b = 1, 2, \ldots, B$ **do**
      augment point cloud observation o_i as $\mathcal{A}(o_i)$
      update backbone θ_p and the actor of policy network θ_p^a, according to: $\lambda_{\text{DA}}\mathcal{L}_{\text{DA}} + \lambda_{\text{adv}}\mathcal{L}_{\text{adv}}$

For state-to-vision distillation, we use point cloud augmentation. We use point cloud jittering with a distance of 0.1 and a strong color augmentation, which changes the GAParts color to a random color during a specific episode. Although we randomly choose a color, we fix it during one episode. This technique works well and improves performance in the unseen category.

There is a potential problem for expert distillation. The output of the actor can be an arbitrary value in $\mathbb{R}^n$. Here n is the output dimension of the actor. Because we use the pose control, and the joint angle is in the range $(-\pi, \pi)$. If the value a_i in i_{th} dimension is out of this range, it would shift to a_i' satisfied $a_i' = a_i + 2k\pi, k \in \mathbb{Z}$. Because of this, multiple outputs would correspond to one action in the simulator so the L2 loss of expert action and student action is not positively associated with the similarity between expert action and student action. To tackle this problem, we add an additional Tanh layer and scale the action to $(-\pi, \pi)$. We use the scaled action to compute dagger loss and update the network.

3. More Details about the Experiment Setting

3.1. Training Details

We train our state-based policy on Nvidia GeForce RTX 2080Ti for 6 hours. For each task, we use all of the data in our dataset claimed in the paper. The PPO hyperparameter is shown in Table 3.

For actor and critic networks, we use 3 hidden layers of MLP. The hidden layer dimension of the MLP is 512, 512, 64. For vision-based policy, we use the Sparse-Unet backbone. If the input point cloud has more than $N = 20,000$ points, we first downsample it to 20000 points using FPS (Farthest Point Sampling). Then we voxelize the input point cloud into a $100 \times 100 \times 100$ voxel grid. The backbone U-Net has an encoder and decoder, both with a depth of 6 (with channels of [16, 32, 48, 64, 80, 96, 112]) and outputs a $N \times K$ per-point feature $\mathbf{F}$ where $K = 16$. We speed up the 3D Sparse UNet inference speed by introducing batch voxelization for point clouds and high parallelization of sparse convolution, thanks to the latest high-performance third-party code base like open3d and sparse

Because PPO is an on-policy RL algorithm, for each N step, we update the policy. To leverage the fast convergence of PPO, we want to update as frequently as we can. On the other side, due to the noisy gradient of RL, the batch that is used to compute the gradient should not be too small, which is equal to $T_{env} * N/M$. Here T is the training environment number, and M is the minibatch size. Empirically we find that the batch size near to 2000 is fine. For six tasks, due to the number of training data, we choose proper minibatch and nsteps. The task-specific hyperparameters are shown in Table 2.

PPO params	value / type
learning rate	3e-4
optimizer	Adam
gamma	0.99
lambda	0.95
desired kl	0.01
clip range	0.1
entropy coef	0.01
init noise std	1

Table 3. **PPO Hyperparameters of Policy Training**

3.2. More Details and Results of the Baselines

For opening the door and drawer, thank the previous exploration, We compare our policy with many baselines. For [1, 4, 7, 8], they focus on tasks like opening drawers and doors and we can modify their method to our OpenDoor and OpenDrawer tasks. And for the other four tasks, we also compare with some possible methods if they can be easily modified to fit our framework. More results are shown in

	robot state	part bounding box	handle bounding box	Point Cloud	part mask	handle mask
Ours (state-based)	✓	✓	✓			
Where2Act [2]	✓			✓	✓	
VAT-Mart [9]	✓			✓	✓	
Maniskill [3]	✓			✓	✓	✓
Ours (vision-based)	✓			✓		

Table 1. **Comparison with Other Methods.**

name	opening door	opening drawer	closing door	closing drawer	pressing button	grasping handle
Train environment number	363	246	363	246	215	88
minibatchs	2	3	2	3	2	2
nsteps	20	20	20	20	20	40
λ_r	0.2	0.2	0	0	0	0.2
λ_d	2	1.3	1	1	1	1
λ_p	1	1	1	1	100	0
λ_t	0	0	0	0	10	1
λ_{df}	1	2	0	0	0	0

Table 2. **Task Specific Hyperparameters of State-based Policy Training**

task	method	Training Set	ValIntra Set	ValInter Set
Closing Door(%)	Where2act [2]	77.3±0.1	54.6±0.0	51.5±0.2
	PPO [6]	35.5±1.1	37.6±0.9	15.4±0.5
	DAgger [5]	84.5±2.5	79.4±1.1	69.9±2.3
	Ours	**88.7±1.0**	**88.4±2.9**	**87.0±1.6**
Closing Drawer(%)	Where2act [2]	89.9±0.2	90.5±0.1	89.9±0.3
	PPO [6]	69.9±5.9	75.2±2.6	59.3±2.1
	DAgger [5]	95.9±1.2	97.3±1.1	91.5±0.2
	Ours	**99.6±0.6**	**97.9±2.1**	**98.6±1.2**
Pushing Button(%)	Where2act [2]	15.5±0.2	16.2±0.1	19.3±0.3
	PPO [6]	25.5±0.2	21.6±1.1	7.9±5.5
	DAgger [5]	32.8±2.2	**41.2±6.6**	29.8±1.2
	Ours	**89.6±2.9**	**79.6±4.2**	**66.6±4.2**
Grasping Handle(%)	Where2act [2]	27.7±0.1	25.4±0.2	13.9±0.3
	PPO [6]	15.7±2.2	13.2±0.6	9.9±3.5
	DAgger [5]	45.6±2.2	35.5±2.1	29.8±2.9
	Ours	**79.8±2.4**	**70.0±2.4**	**56.4±2.9**

Table 4. **More Results of Method Comparison and Baselines**

Table 4.

PPO [6]. We directly use the PPO algorithm to learn a vision-based policy to handle each task. The detailed PPO parameter and training strategy is the same as the state-based expert training in our method.

Where2Act [2]. We input the part mask as an extra dimension in our task as a baseline, and others remain the same. We modified the where2act interaction pipeline to finish our tasks. We use a similar pulling motion for the first three tasks and a pushing motion for the fourth task. Giving only a point to indicate the part to be interacted with makes it challenging for where2act to perform proper actions, especially for opening drawers and doors. We thus provide additional information (*i.e.*, the handle center of the target door and drawer), and this method needs to select one point from the given points. Then, after motion direction selection, the action is performed to finish the task. We constrain $N_{\text{w2a}} = 10$ actions to finish these tasks.

ILAD [10]. Due to we have designed a dense reward in our task, we use our dense reward instead of their extra Q functions to compute the advantage in the third term of g_{ILAD}. The demonstrations are also collected by expert policy as the GAIL [7] baseline implementation. We don't input part 6D pose into the network as a fair comparison to our method.

ManiSkill [3] Winners, *i.e.*, *Shen et. al* [7], SilverBullet3D [4], *Wu et. al* [8], *Dubois et. al* [1]. We follow the ManiSkill [3] settings and follow the corresponding policy learning strategy to learn. If the method needs collected demonstrations as input, we provide the demonstrations collected by the state-based expert.

For these baselines, we analyze that their performances are limited due to the distribution shift in behavior cloning, lacking vital information with realistic sensory observation input and noisy reinforcement learning gradient.

3.3. More Results for a Single Camera Setting

Here we provide more experiment results for a single camera setting. For OpenDoor, the performances are 36.7±3.3, 33.9±2.4, 22.6±3.0 in the training set, Val-Intra set and Val-Inter set respectively. For OpenDrawer, the performances are 64.5±5.5, 60.2±4.4, 17.1±2.2 in the training set, Val-Intra set and Val-Inter set respectively.

3.4. Some Qualitative Results for the Failure Cases

Here we provide some qualitative results for failure cases. In Fig. 1, we show two failure cases. For the left one, the gripper fails to identify the handle and grasps the wrong position due to the thin and flat handle shape (yellow,

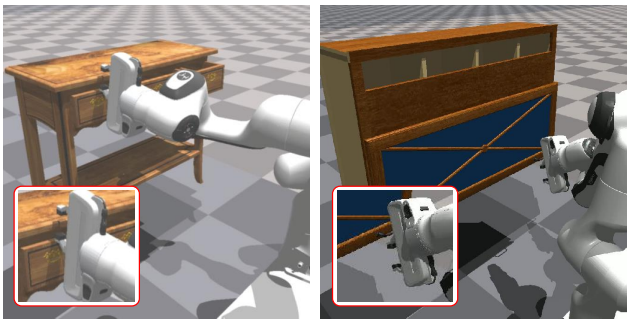

Figure 1. **Failure Cases**

zoom in to see), while for the right one, the door opening fails later for unstable grasping.

4. Real Experiment

We use the robot arm (FRANKA) to manipulate previously unseen real objects with only partial point cloud observations. A partial point cloud of the target object instance is acquired from the RGB-D camera (Okulo P1 ToF sensor in our experiments). To set up the interaction environment, we use aruco markers to calibrate the camera sensor and place the object and the robot arm in the proper positions, the same as the trained policy in the simulator. We also provide a point to indicate the part to interact with, just like we did in the simulator. During manipulation, we use the control API provided by the robot arm system to follow the trajectory (a sequence of joint angle and gripper position) and finish the tasks.

References

[1] Fabian Dubois, Eric Platon, and Tom Sonoda. Improving performance on the maniskill challenge via superconvergence and multi-task learning. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022. 2, 3

[2] 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. 3

[3] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021. 3

[4] Yingwei Pan, Yehao Li, Yiheng Zhang, Qi Cai, Fuchen Long, Zhaofan Qiu, Ting Yao, and Tao Mei. Silver-bullet-3d at maniskill 2021: Learning-from-demonstrations and heuristic rule-based methods for object manipulation, 2022. 2, 3

[5] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 3

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3

[7] Hao Shen, Weikang Wan, and He Wang. Learning category-level generalizable object manipulation policy via generative adversarial self-imitation learning from demonstrations. *arXiv preprint arXiv:2203.02107*, 2022. 2, 3

[8] Kun Wu, Yinuo Zhao, Zhiyuan Xu, Zhen Zhao, Pei Ren, Zhengping Che, Chi Harold Liu, Feifei Feng, and Jian Tang. A minimalist ensemble method for generalizable offline deep reinforcement learning. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*. 2, 3

[9] 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. In *International Conference on Learning Representations*, 2022. 3

[10] Yueh-Hua Wu, Jiashun Wang, and Xiaolong Wang. Learning generalizable dexterous manipulation from human grasp affordance. *arXiv preprint arXiv:2204.02320*, 2022. 3

[11] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Ipod: Intensive point-based object detector for point cloud, 2018. 1