Supplementary Material for GAPartNet: Cross-Category Domain-Generalizable Object Perception and Manipulation via Generalizable and Actionable Parts

https://pku-epic.github.io/GAPartNet

1. Dataset and Data Annotation

1.1. Data Annotation

To construct a large-scale part-centric interactive dataset, great effort is needed to clean up and annotate existing object shapes. We first identify the issues with the existing database, and then we develop a systemic pipeline for annotating the large-scale dataset.

Data sources. GAPartNet dataset is constructed based on two existing datasets, PartNet-Mobility [11] and AKB-48 [4]. Focusing on the GAParts we define, we select 23 object categories from PartNet-Mobility and 4 object categories from AKB-48. Most of the 3D object shapes in GAPartNet are from PartNet-Mobility. Since the texture of shapes in PartNet-Mobility is all synthetic, to mitigate the sim-to-real gap, we further leverage the shapes from AKB-48 whose texture is scanned from the real world.

Note that both PartNet-Mobility and AKB-48 have the object categories *Box, Bucket, TrashCan*. Although they use the same category names, their shapes can be very different. A TrashCan from PartNet-Mobility and one from AKB-48 can have significant differences in geometry, the same as Box and Bucket. So we do not merge them together into one object category but keep their original categories.

Issues with Existing Database. The original PartNet-Mobility [11] and AKB-48 [4] lack of directly usable information we need for our new annotations. First, they do not provide directly usable consistent semantic annotations to similar parts across object categories. For example, some handles on *Door* are labeled as *door*, while some doors on *StorageFurniture* are labeled as *frame*. Secondly, their original annotations are not as fine-grained as we need. Specifically, fixed handles, *i.e*. line fixed handles and round fixed handles, are not annotated as individual parts, since they are attached to either base bodies or other movable parts. Their meshes are merged with others which leaves rare semantic cues to re-separate them. Finally, there are a lot of meshes of parts that we care about are imperfect, which seriously limits either the quality of our pose annotations or the quality of rendered images.

Data Annotation Effort. To address these issues, we first manually go over all objects to re-separate the meshes of fixed handles from the original 3D object shapes. We also modify the kinematic chains to re-merge these meshes into new links and add corresponding fixed joints, which provides more consistent annotations and is beneficial for following robotic tasks. In this step, more than 1,000 fixed handles are re-separated and re-merged. Secondly, we go over all 1,166 objects in GAPartNet and clean all original semantic annotations to align with our GAPart class definition. Thirdly, we manually use MeshLab and some heuristics to modify imperfect meshes, not only cutting the redundant meshes off but also fixing the one-sided meshes to facilitate the annotating and rendering. In this step, more than 100 object instances are modified.

Finally, with the 1,166 3D object shapes with new semantic annotations and modified meshes, we use a lot of heuristics to fit the oriented tight bounding boxes of all 8,489 GAParts, corresponding to their canonical orientations, and add our pose annotations. With our effort, GAPartNet is capable of detection, segmentation, pose estimation, and manipulation on cross-category generalizable and actionable parts.

1.2. Dataset Rendering

We use the SAPIEN 2.0 environment [11] to render a large-scale dataset from our GAPartNet objects, consisting of partial point clouds, part semantic segmentation masks, part instance segmentation masks, NPCS maps, and part pose annotations, which covers all the data needed for the proposed part segmentation, part pose estimation and part-based object manipulation tasks.

Environment Settings. We turn on the ray-tracing mode of SAPIEN to get more sense of reality. During rendering, we randomize the joints' poses of the articulated objects and randomly pick a camera position within a reasonable perspective. Specifically, we manually set the range of camera position for each object category to get desirable views of each object, making sure we do not look at the back of a StorageFurniture, or from beneath an Oven, neither from too far nor too close. In the meantime, we randomly dim the ambient light within [10%, 90%] and randomly rotate the camera within $\pm 5^\circ$.

The output image resolution is set to 800×800. For each

object, we render 32 RGB images. Along with each RGB image, we also obtain the segmentation masks and the depth image using built-in features of the SAPIEN environment. Additionally, we compute NPCS maps and oriented tight bounding boxes as part pose annotations for all GAParts.

Point Cloud Sampling. Using camera intrinsics, 2D RGB images, and depth images, we do back-projection to obtain dense, partial point clouds. We sample 20,000 points for each dense point cloud using Farthest-Point-Sampling (FPS). While sampling the point clouds, we also generate corresponding ground truth of semantic segmentation, instance segmentation, and NPCS maps. These 20,000-point point clouds and their annotations are computed offline for speeding up the following 3D tasks.

2. More Details on Part Segmentation and Part Pose Estimation

2.1. Details on Network Architecture

Architecture. The vision network has a similar architecture as PointGroup [3]. Please refer to the original PointGroup paper for details. In our work, we set the cluster radius to 0.03 and the cluster point number threshold to 5 to get good segmentation results in the GAPartNet dataset. The input point cloud $\mathbf{P}$ is first voxelized into a $100 \times 100 \times 100$ voxel grid. The backbone UNet consists of an encoder and a decoder, both with a depth of 7 (with channels of [16, 32, 48, 64, 80, 96, 112]), and outputs a point-wise feature $\mathbf{F}$ with K channels, where $K = 16$. After grouping, each mask proposal C_i' is normalized and voxelized again into a $50 \times 50 \times 50$ voxel grid and passed through the $Scoring$ module, which consists of a 2-depth UNet (with channels of [16, 32]) for point-wise feature extraction, an ROI Pooling layer for foreground feature merging, and a linear layer for confidence score $\mathbf{S}_i$ prediction. During inference, points with binary classification scores below 0.4 are filtered out as background, and proposals with fewer than 5 points or a score lower than 0.09 are discarded. Finally, Non-Maximum Suppression (NMS) with an IoU (Intersection over Union) threshold of 0.3 is applied to get the final segmentation masks $\mathcal{C}$.

For domain adversarial learning, we introduce a Gradient Reverse Layer (GRL) with $\alpha = 0.3$ for the negative gradients and three domain discriminators with similar architectures as the $Scoring$ module mentioned above for domain classification. We place the three discriminators at the 2-nd, 4-th, and 6-th decoder layers of the backbone UNet, so the three discriminators can take different features from the three layers of the backbone for domain classification. Each discriminator takes the queried points and the corresponding features as input and predicts the domain labels. The domain discriminators are only used during the training procedure, and the proportion of classification is set to 0.05 in our implementation.

For each segmentation mask C_i, the point-wise feature $\mathbf{F}_{C_i}$ queried from $\mathbf{F}$ is passed through the $NPCS\text{-}Net$, consisting of a 2-depth UNet (with channels of [16, 32]) and three Multilayer Perceptrons (3-MLP) for poise-wise NPCS prediction. Note that in practice, we use 9 different groups of 3-MLP to predict NPCS coordinates in 9 channels, and we only supervise the channel corresponding to the ground truth semantic label.

2.2. Details on Supervision

Symmetry-aware Part Pose Estimation. Since each part class in GAPart has different symmetry patterns, they should be handled case by case. We design the symmetry-aware NPCS loss as follows:

Type 1 (*i.e.*, line fixed handle, hinge handle): we tolerate the $180°$ symmetry along the z axis for this symmetry type.

Type 2 (*i.e.*, hinge door, hinge lid): we tolerate the $180°$ symmetry along the y axis for this symmetry type.

Type 3 (*i.e.*, slider button, slider lid, round fixed handle): we tolerate the rotation along the z axis and flipping along the x-y plane for this symmetry type. In our implementation, we split the continuous rotation angles into 12 discrete angles for supervision.

Type 4 (*i.e.*, hinge knob): we tolerate the rotation along the z axis for this symmetry type. In our implementation, we split the continuous rotation angles into 12 discrete angles for supervision.

Type 5 (No symmetry, *i.e.*, slider drawer) : we do not tolerate any symmetry for this symmetry type.

The design of NPCS loss $\mathcal{L}_{\text{NPCS}}$ is similar to [10]. We use soft-L1 loss and for each tolerated symmetry pattern, we supervise the minimal loss in the set. For more implementation details, please refer to [10].

Loss Function. The whole training procedure of the network can be divided into four stages.

For the first stage (0-5 epochs), we only supervise the semantic prediction and the offset prediction branches with the same loss functions $\mathcal{L}_{\text{sem}}$ and $\mathcal{L}_{\text{off}}$ as PointGroup [3]. Please refer to [3] for more details.

For the second stage (5-10 epochs), we add the score loss $\mathcal{L}_{\text{sco}}$ for the proposals' IoU prediction, following the design of [3].

For the third stage (10-15 epochs), we add the symmetry-aware NPCS loss $\mathcal{L}_{\text{NPCS}}$ for the NPCS prediction, as introduced above.

For the fourth stage (after 15 epochs), we introduce our domain adversarial learning strategy after the part segmentation network can output good proposals and corresponding proposal scores, similar to [2]. The total loss in this stage can be formulated as

$$\mathcal{L} = \mathcal{L}_{\textbf{QRB}\text{-adv}} + \mathcal{L}_{\text{sem}} + \mathcal{L}_{\text{off}} + \mathcal{L}_{\text{sco}} + \mathcal{L}_{\text{NPCS}},$$

where $\mathcal{L}_{\textbf{QRB}\text{-adv}}$ denotes the domain adversarial loss.

2.3. Details on Pose Fitting and Joint Prediction

Pose Fitting. Given a predicted 3D part mask with its NPCS map, we use RANSAC [1] for outlier removal and Umeyama algorithm [8] to estimate the 7-dimensional rigid transformation.

Joint Parameter Prediction. We simplify the joint parameter prediction process thanks to the unified definition of our GAParts. After estimating the bounding box for each part, we can leverage the definition of the GAPart to directly calculate the joint parameters. For example, given the bounding box of a slider button, we can directly query its prismatic joint parameter, which is along the z axis in the part canonical space.

2.4. Training Procedure

Our model is trained in an end-to-end manner with maximum training epochs of 200. We use the Adam optimizer with a batch size of 32 and a learning rate of 0.001. The whole training procedure takes around 1.5 days on a single NVIDIA GeForce RTX 2080 Ti GPU. Note that the domain adversarial training is very unstable, we thus use five seeds to train it and select the best one. What's more, to boost performance, we progressively use the multi-resolution training strategy, which improves the performance.

2.5. Seen/Unseen Object Categories Splitting

17 Seen Categories. Box, Bucket, Camera, CoffeeMachine, Dishwasher, Keyboard, Microwave, Printer, Remote, StorageFurniture, Toaster, Toilet, WashingMachine, Bucket (AKB-48), Box (AKB-48), Drawer (AKB-48), Trashcan (AKB-48).

10 Unseen Categories. Door, KitchenPot, Laptop, Oven, Phone, Refrigerator, Safe, Suitcase, Table, TrashCan.

2.6. Baseline Experiments

PointGroup [3]. The PointGroup baseline is modified from [3]. We add our NPCS prediction branch to the vanilla PointGroup. The final loss can be formulated as $\mathcal{L}_{\text{PointGroup}} = \mathcal{L}_{\text{seg}} + \mathcal{L}_{\text{NPCS}}$, where $\mathcal{L}_{\text{NPCS}}$ is the same as our method.

AutoGPart [5]. Following AutoGPart [5], we introduce a similar intermediate supervision for generalizable part segmentation. We build a parametric supervision model $\mathcal{M}(\cdot|\theta)$ to find a proper intermediate part segmentation supervision, which can be learned through a "propose, evaluate, update" strategy. We use each object category as each "sub-domain" in AutoGPart and use the same hyperparameters for the intermediate auxiliary loss. We still add our NPCS prediction branch to the network for part pose estimation. The final loss can be formulated as $\mathcal{L}_{\text{AGP}} = \mathcal{L}_{\text{seg}} + \mathcal{L}_{\text{intermediate}} + \mathcal{L}_{\text{NPCS}}$. For more details about the intermediate auxiliary loss and the training strategy, please refer to [5].

3. More Details on Part-based Object Manipulation

3.1. Interaction Policy

(1) Round Fixed Handle: For a round fixed handle, we use the gripper to approach the handle from the positive direction of the z axis, open the gripper to a width that exceeds the side length of the bounding box, and then close the gripper to complete the grasping.

(2) Line Fixed Handle: The interaction policy for a line fixed handle is similar to a round fixed handle. Note that we want the opening direction of our gripper and the line fixed handle to be perpendicular, so we turn the opening direction parallel to the y axis of the predicted bounding box.

(3) Hinge Handle: The interaction policy for a hinge handle is similar to a line fixed handle. After approaching and grasping the hinge handle, we can rotate it along the predicted axis of the revolute joint.

(4) Slider Button: For a slider button, we close the gripper, approach the button from the positive direction of the z axis, and then press the button.

(5) Hinge Knob: For a hinge knob, we clamp the knob like a round fixed handle and rotate the end-effector to complete the manipulation.

(6) Slider Drawer: A gripper approaches an open drawer along the z axis to fetch something in the drawer, and approaches a drawer against the x axis to open it. More often than not, we expect to grab a handle hopefully located on the front face of a drawer.

(7) Hinge Door: For a hinge door with a handle on the front face, we try to grab the handle to open the door. After grabbing the handle, the gripper rotates around the predicted shaft of the door to complete the opening or closing. For a door without any handles, if the door is not closed, we use the gripper to clamp the outer edge along the y axis of the bounding box to open the door.

(8) Hinge Lid: for a hinge lid, we use an interaction policy similar to a hinge door.

(9) Slider Lid: for a slider lid with a handle, we grab the handle to open the lid. Otherwise, we use the gripper to clamp the edge of the lid along the x-y plane of the bounding box, and then move up and down along the z axis to open and close the lid.

3.2. Simulation Experiments

Benchmark Settings. We set up our interaction environment using the SAPIEN [11] simulator, modified from the ManiSkill challenge [7]. We benchmark our method on 4 tasks, *i.e.*, using a single Franka gripper to open a

drawer, open a door, manipulate a handle, and press a button. These tasks exemplify robot manipulation under the motion constraint of a prismatic or a revolute joint. For evaluation, we randomly pick unseen objects that contain doors, drawers, handles, and buttons from seen object categories. Considering the limitation of the single gripper, we select such objects that, given the ground truth of their segmentation and pose, can be opened successfully using the heuristics under our benchmark setting. Furthermore, to evaluate the cross-category generalizability of our method, we also randomly pick unseen objects from previously unseen object categories. Compared to the ManiSkill Challenge [7], we limit our observation to a first-frame-only partial point cloud of the object, with only one point around the part center indicating which part to interact with. Given the initial state of the robot, it performs the whole manipulation only based on the observation at the first time step. The action space of the robot is the motor command of the 6 joints of the robot to determine the pose of the gripper, and we use position control to open or close grippers. A success in opening the drawer, opening the door, using the handle, and pressing the button is defined as manipulating the part for 90% of the motion range within 1,000 steps with a stable stop at the end. For each task, we use 20 objects from seen categories and 20 from unseen categories to construct our benchmarks, respectively. Overall, we conduct 4 manipulation tasks in the simulator with 160 objects from 6 seen object categories and 6 unseen object categories.

Part-pose-based Manipulation Heuristics. We use the interaction policy based on the heuristics introduced in Sec. 3.1 to open drawers, open doors, manipulate handles, and press buttons. Specifically, when we get the part pose, we can immediately get the grasping pose with our policy. Then we use a motion planning library (*i.e.*, mplib, provided by SAPIEN [11]), to move our gripper to the grasping pose. Then, with our interaction policy and axis predicted from our method, we design the end-effector trajectory just along the trajectory of the part moving and interpolate the trajectory with a time step of $\frac{1}{250}$. With the IK (Inverse Kinematics) algorithm and a PID controller, we solve the poses of joints and move the end-effector along the trajectory. All of our implementations are decoupled from ROS and can be easily implemented in other simulators.

Baselines for Object Manipulation. (1) Where2act [6] (Oracle input for the first two tasks). 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 or drawer as a special indicator), and we directly select this point as the point to be interacted with. Then, after motion direction selection, the action is performed to finish the task. We constrain $N_{\text{w2a}} = 10$ action steps to finish these tasks. (2)ManiSkill [7] (Oracle baseline). ManiSkill provides a method for similar vision-based tasks in a reinforcement learning setting. To satisfy the settings in this baseline, we further provide oracle inputs (*i.e.*, per-frame point cloud observations and ground truth part masks). We also design similar dense rewards for each task and train the policy with the same hyper-parameters as ManiSkill. Please refer to [7] for more details.

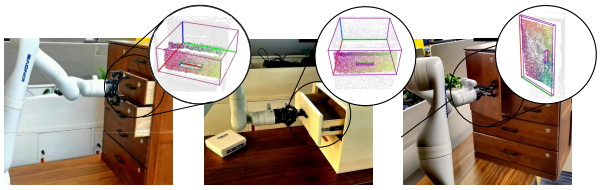

Figure 1. **More Qualitative Results for Part-based Object Manipulation in the Real-world.**

3.3. Real-world Experiments

Implementation Details. To evaluate the robustness and generalizability of our method, we use two robot arms (*i.e.*, KINOVA and FRANKA) to manipulate previously unseen objects with only partial point cloud observations in the real world. We use similar motion planning and a similar end-effector trajectory as what we do in the simulator. A partial point cloud of the target object is acquired from the RGB-D camera (Okulo P1 ToF sensor in our experiments). To set up the interaction environment, we place the object and the robot arm in a proper position for interaction and use ArUco markers to calibrate the camera sensor. We also provide a point indicating the part to interact with, just like in the simulator. During manipulation, we first estimate the bounding box of the target part and calculate the trajectory using the heuristics, then use the control API provided by the robot arm to follow the trajectory and finish the task. Overall, we conduct 4 manipulation tasks, *i.e.*, opening doors, opening drawers, lifting lids, and pressing buttons, in the real world with 11 objects from 2 seen object categories and 3 unseen categories.

4. Visualization of GAPartNet Dataset

Exemplar objects of each GAPart class from seen categories and unseen categories in the GAPartNet dataset are shown in Fig. 2.

5. More Results of Part Segmentation and Part Pose Estimation

We visualize more results of part segmentation and part pose estimation in Figs. 3 to 5.

Success Rate(%)	Drawer		Door		Handle		Button	
	Seen	Unseen	Seen	Unseen	Seen	Unseen	Seen	Unseen
Where2act [6]	69.9	54.5	44.4	18.2	78.7	49.2	82.2	80.9
ManiSkill [7]	32.9	26.6	27.8	28.3	53.9	42.1	65.5	54.5
Ours	**95.0**	**90.0**	**70.0**	**55.0**	**90.0**	**85.0**	**100.0**	**95.0**

Table 1. **Results for Cross-category Object Manipulation in SAPIEN Simulator [11].**

		Train Set	Test Set	
			S.C.	U.C.
Opening Door	w/o Pose	26.1±4.9	22.0±2.3	18.1±2.8
	w/ Pose	**58.3±3.9**	**37.9±2.5**	**18.3±2.9**
Opening Drawer	w/o Pose	59.8±4.2	40.9±4.5	18.4±3.3
	w/ Pose	**91.2±5.2**	**87.1±6.7**	**35.6±3.8**

Table 2. **Part poses improve RL success rate.** S.C.=Unseen objects in seen categories. U.C.=Unseen objects in unseen categories. A larger benchmark for RL with # instances: 258/63/77 for doors, 138/57/96 for drawers.

6. More Results of Part-based Object Manipulation

For the simulation experiments, the quantitative results are shown in Tab. 1. Our method significantly outperforms the baselines on all 4 tasks, showing good generalizability and proving the effectiveness of our part-pose-based manipulation policy. More qualitative results are provided in the supplementary video 04:30-04:46.

For the real-world experiments, more qualitative results are provided in Fig. 1 and the supplementary video 04:47-05:12.

7. More Discussions

7.1. Real Depth Signal and Sim-to-Real Gap

In our experiments, we find that depth quality is crucial to our perception and downstream manipulation. Actually, for our real-world experiments, we have to spray the contrast aid paint onto the transparent lid and use a structural sensor to closely scan the remote and the calculator for obtaining good and detailed geometry. For diffuse objects with okay depth quality, we argue that further leveraging domain adaptation would be beneficial; however, for certain metallic or transparent objects, their depth will be incomplete, falling into a completely different problem. We leave a more fundamental solution to predict/refine geometry for future works.

7.2. Outlier Part Shapes

In our work, GAParts are defined to be functional parts with similar geometry and actionability. So how can our framework tackle the parts with outlier shapes?

Here we take the curvy or irregular handles on doors as an example. For certain handles, their perception is basically an out-of-distribution perception problem and can theoretically be tackled within our framework; however, we admit the pose of those outliers may not be so informative, which may lead to failure in manipulation heuristics. We argue that the function and actionability of outlier door handles, *e.g.*, revolving to open, is still the same as the regular ones. So learning a manipulation policy based on actionable information instead of relying on heuristics would be promising (see our further discussion in Sec. 7.3) and can potentially handle those outliers.

7.3. Part Information for Manipulation in RL

By definition, the GAPart carries abundant information about the part's pose, function, actionability, *etc.*, which is valuable to facilitate manipulation policy learning in RL. Here we provide a pilot study and some preliminary results to showcase the usefulness of GAPart information. We conduct experiments on learning cross-category manipulation policy from state observations for opening door and opening drawer tasks using PPO under dense rewards, as shown in Tab. 2. We take proprioceptive information and the bounding box of the door/drawer as state input. The distinction between w/ and w/o pose is whether an additional state input, ground truth handle pose, is used. The results demonstrate that oracle GAPart information can significantly benefit policy learning. This would hopefully shed light on more advanced RL designs in future research, such as incorporating part pose estimation into reward functions and leveraging the part pose to canonicalize visual signals.

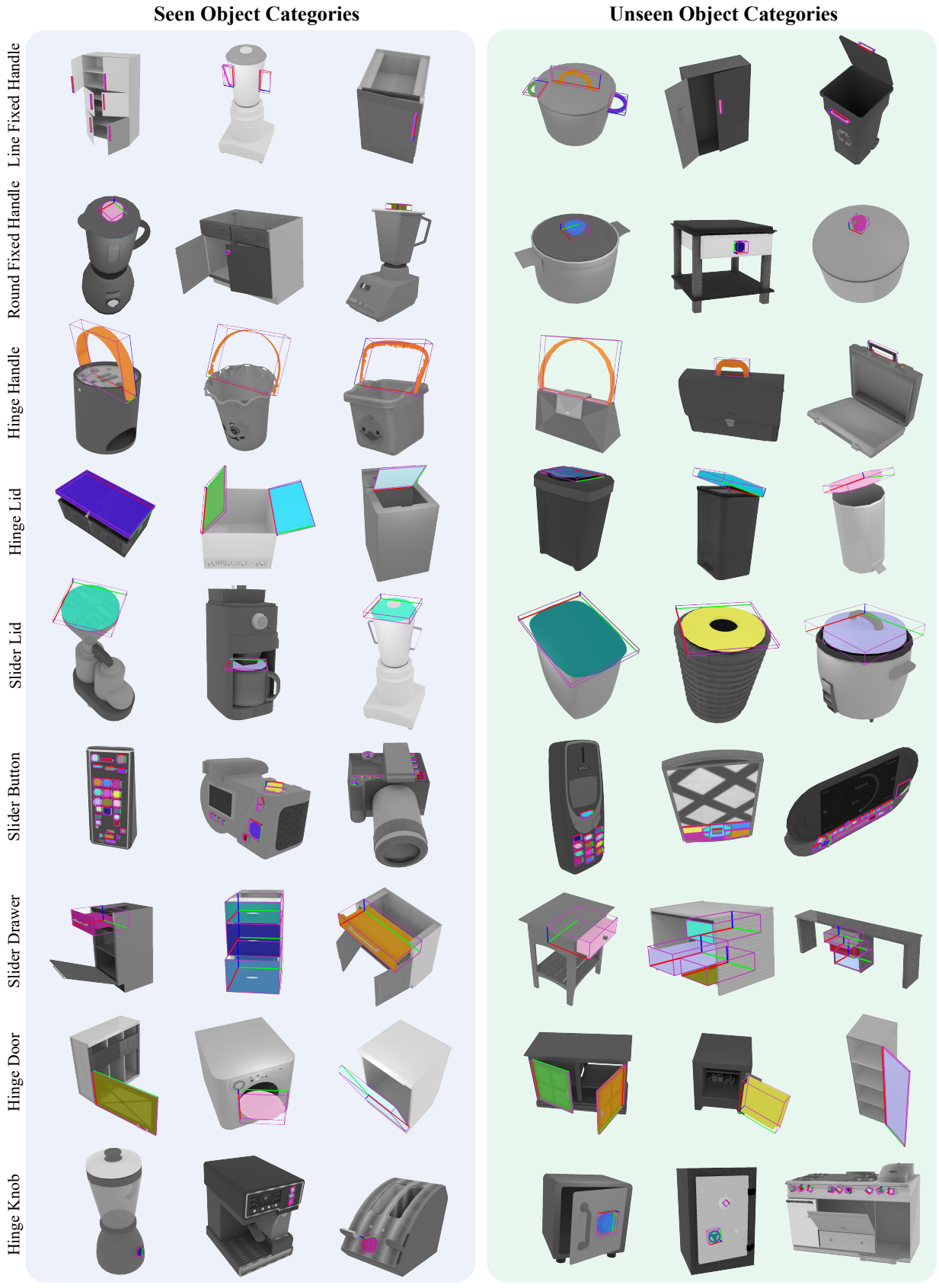

Figure 2. **Exemplar Objects of Each GAPart Class from Seen Categories and Unseen Categories.** We show objects in gray scale, GAPart segmentation masks in color, and GAPart poses using oriented tight bounding boxes.

Qualitative Results of Part Segmentation and Part Pose Estimation (Seen Category Unseen Instance)

Figure 3. **Part Instance Segmentation and Pose Estimation Results on the Unseen Instances from the Seen Categories.** Here we compare our method on part instance segmentation task with PointGroup [3], SoftGroup [9], and AutoGPart (modified from [5]).

Qualitative Results of Part Segmentation and Part Pose Estimation (Unseen Category Unseen Instance)

Figure 4. **Part Instance Segmentation and Pose Estimation Result on the Unseen Instances from the Unseen Categories.** Here we compare our method on part instance segmentation task with PointGroup [3], SoftGroup [9], and AutoGPart (modified from [5]).

Qualitative Results of Part Segmentation and Part Pose Estimation (Real World)

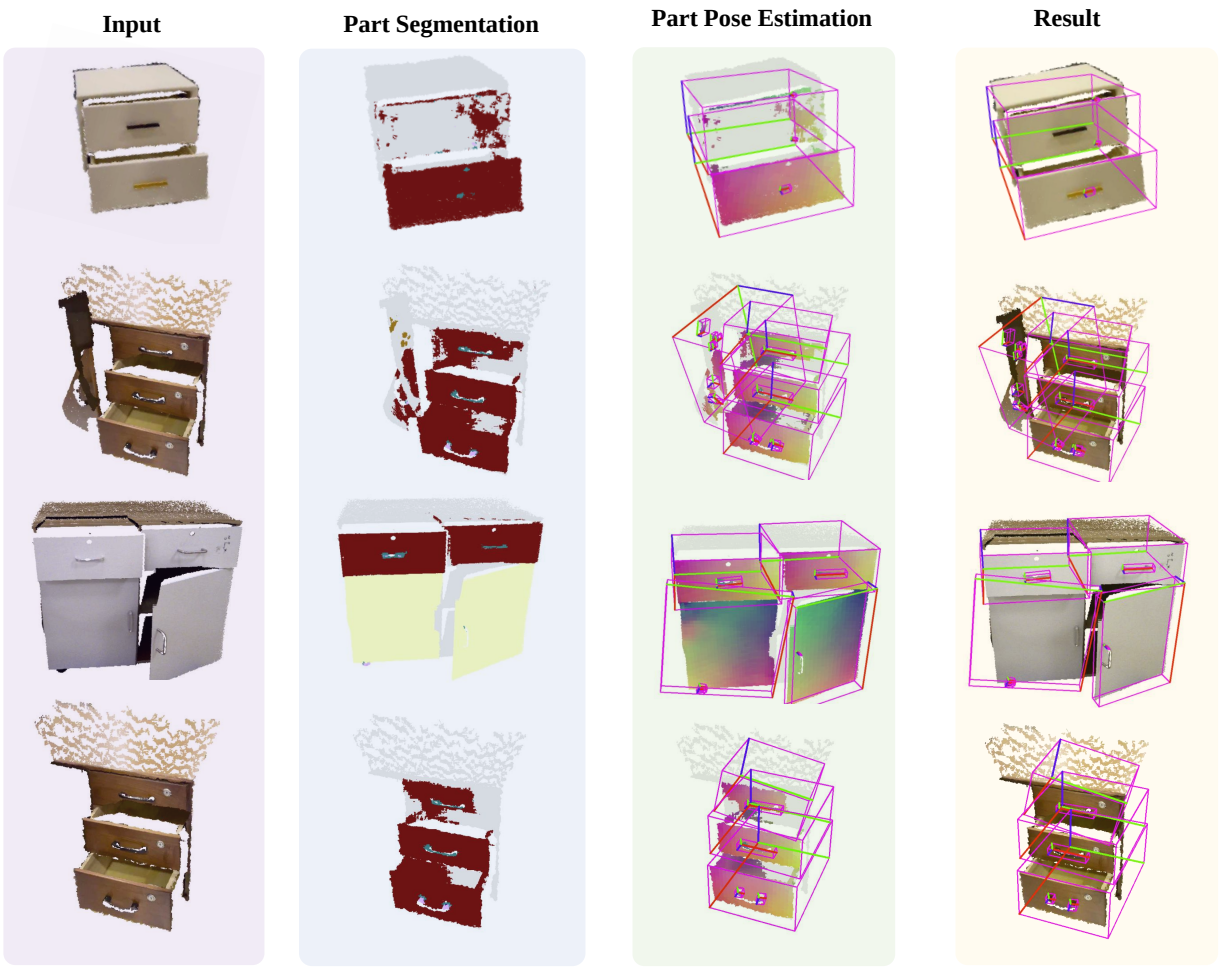

Figure 5. **Part Instance Segmentation and Pose Estimation Result on the Unseen Objects from the Real World.**

References

[1] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 3

[2] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016. 2

[3] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4867–4876, 2020. 2, 3, 7, 8

[4] Liu Liu, Wenqiang Xu, Haoyuan Fu, Sucheng Qian, Qiaojun Yu, Yang Han, and Cewu Lu. Akb-48: A real-world articulated object knowledge base. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14809–14818, 2022. 1

[5] Xueyi Liu, Xiaomeng Xu, Anyi Rao, Chuang Gan, and Li Yi. Autogpart: Intermediate supervision search for generalizable 3d part segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11624–11634, 2022. 3, 7, 8

[6] 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. 4, 5

[7] 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, 5

[8] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991. 3

[9] Thang Vu, Kookhoi Kim, Tung M Luu, Thanh Nguyen, and Chang D Yoo. Softgroup for 3d instance segmentation on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2708–2717, 2022. 7, 8

[10] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *CVPR*, pages 2642–2651, 2019. 2

[11] 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 *CVPR*, 2020. 1, 3, 4, 5