Supplementary Material: Visually-Grounded Library of Behaviors for Manipulating Diverse Objects across Diverse Configurations and Views

Jingyun Yang* **Hsiao-Yu Tung*** **Yunchu Zhang*** **Gaurav Pathak**
Ashwini Pokle **Christopher G Atkeson** **Katerina Fragkiadaki**
Carnegie Mellon University
{jingyuny,htung,yunchuz,gauravp,apokle,cga,kfragki2}@andrew.cmu.edu

A Additional Results

A.1 Test Time Rollout Samples

To better understand the performance of our framework visually, we prepare test time rollouts of our framework as well as those of various baselines. Please look at the supplementary video for these qualitative results.

A.2 More Baselines on Grasping

We add evaluation and discussion on state-of-the-art grasping methods, 6D-GraspNet [1] and DexNet [2]. We also include one ablated model to show the influence of excluding non-top grasps on task performance. We include the following three extra baselines:

DexNet [2]: a state-of-the-art top-grasp method which, given a top-down depth map, generates grasps as the planar position, angle, and depth of a gripper relative to an RGB-D sensor. We use the publicly released code and model provided by [2], fine-tune the model with the same amount of interactive labels as our model, and evaluate it in our test environments.

6DoF Grasp-Net [1]: a state-of-the-art grasping method that describes the grasp as a full 6-degrees-of-freedom (DoF) pose. The system learns a grasp generator that proposes potential grasp points given a point-cloud from the depth image, and an evaluation module which evaluates the quality of the proposed grasp. The system is trained with 10M grasps generated from randomly generated boxes and cylinders, and ShapeNet models of bowls, bottles and mugs. We used the publicly released code and model provided by [1], fine-tune the model with the same amount of interactive labels as our model, and evaluate it in our test environments.

Our Model with Only Top-grasps: an ablated baseline using the proposed methods and including only the top-grasps. The baseline is similar to Pinto et al. [3], which predicts which of the 18 top-grasps the robot should execute based on a top-down RGB image patch around the target object. To handle cameras with arbitrary views and not just top-down views, which might be unavailable in many real world scenarios, we use the proposed 2D-to-3D feature encoder to compute view-invariant features as opposed to using the 2D-to-1D feature encoder used in Pinto et al. [3].

We compare the final task performances of our method and the aforementioned baselines in Table 1. All the three baselines perform worse than the proposed model. We found 6D-GraspNet performs much worse than what is reported in the original paper with our setup, and we attribute this to the fact that our setup is more difficult than the one proposed in the original 6D-GraspNet paper, since we randomize objects' initial and goal locations, and camera poses. We found 6D-GraspNet sensitive to camera pose change. Besides, many proposed grasps turn out to be unstable when an object is placed too close or far away from the robot. Dexnet performs reasonably well on objects that can be grasped with a top-down grasp, but fails completely on objects that require a side-grasp, e.g., plates. Aside from the flat objects that cannot be solved with top-grasps, *DexNet* fails very often on objects that are thin and long, and can only be robustly grasped by touching the two short edges. In

* Equal contribution.

5th Conference on Robot Learning (CoRL 2021), London, UK.

these scenarios, *DexNet* tends to grasp along the long edges of objects, which results in grasps that are not robust. On the ablated model, the grasping performance drops by 10% when side-grasps are excluded from the set of behaviors used by our method. In particular, the model without side-grasps fails to grasp flat objects such as plates and flat bowls.

	Proposed Method	Proposed Method w/ Only Top-grasps	DexNet [2]	6Dof-GraspNet [1]
Grasping	**0.78**	0.67	0.72	0.36

Table 1: Success rates on grasping unseen objects.

A.3 Are the single policies overfitting or underfitting?

The single policies have similar performances on the training and test data. For example, in the pushing task, the *Abstract 3D + Image* baseline achieves a success rate of 0.69 in training set and 0.70 in test set; the *Abstract 3D* baseline achieves a success rate of 0.81 in training set and 0.83 in test set. This means that these models are underfitting. The Abstract 3D baselines do not perform well because they do not have visual input and lack information about the object's shape; the Abstract 3D + Image baseline does not perform well because it operates in 2D image space where the representation (of objects, their poses, and appearance) can change dramatically due to camera pose change, making the learning problem difficult; the Contextual 3D baselines need much more compute due to the 4D bottleneck and do not learn even with strong supervision from imitation learning. For all the baselines, we have grid-searched on the number of parameters used in the model and picked the best. Models with more parameters are in general more difficult to train and do not necessarily give better results.

A.4 Failure cases of the proposed method

Our method fails typically when (1) the selector makes the incorrect prediction, e.g., grasping a cucumber with the wrong grasping angle (please see 2:13 of the supplementary video for the example) and (2) when all the behaviors in the library do not work, e.g., when trying to pour grapes from a container, although the trained selector chooses the correct pouring behaviors, the grapes did not successfully get out of the container due to strong friction (please see 3:31 of the supplementary video for the example). The first issue can be solved by improving the 3D feature representations. One way is to improve the 3D resolution with memory-efficient structures such as point clouds. Another approach is to improve the features using recent advances in self-supervised feature learning. The second issue can be improved by scaling up the behaviors in the library or automatically adapting existing behaviors in the library. Both are interesting future avenues to explore.

A.5 Visualizing Behavior Clusters

To understand the learned affordance-aware visual feature representations, we run T-SNE on the feature representations of test-time objects and initial configurations in the pushing task and show their behavior assignments as well as their relative positions in the embedding space in Figure 1. In the figure, each image shows one sample in the test set where a test-time object with its initial position and pose is displayed. The border color of each image denotes the behavior assignment of the test-time sample. As seen in the visualization, the learned feature representation in the affordance-aware behavior selector represents objects that are close in affordance in neighboring regions of the feature space. It is also robust to variations of object colors, sizes, and semantic categories. For example, the behavior trained on knives (last row in the figure) is predicted to be able to handle both very thin keyboards and knives; the behavior trained on bottles (fourth row in the figure) is predicted to be able to handle both bottles and small cans.

B Limitations and Future Work

The set of behaviors in the current library is fixed, and we look forward to exploring how to add new behaviors in future work. One direction is detecting missing behavior and automatically learning

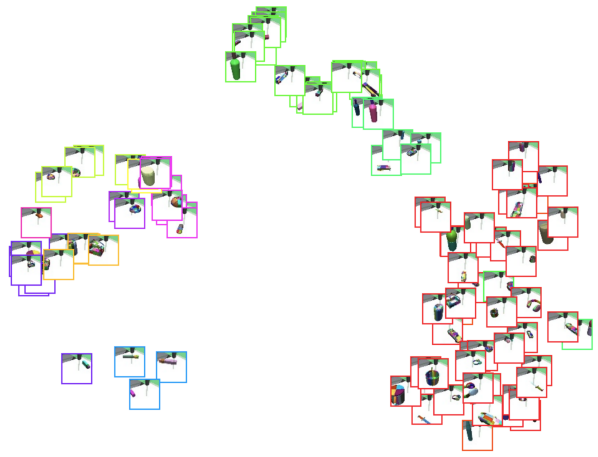

Figure 1: T-SNE visualization of feature representations of test-time objects and their initial configurations. Each image shows one sample in the test set where a test-time object with its initial position and pose is displayed. The border color of each image denotes the behavior assignment of the test-time sample.

appropriate new behavior. Another direction is to scale up the library of behaviours so you have diverse enough behavior to solve most tasks. Recent advances in large-scale imitation learning from human demonstration [4] will be a promising direction to investigate. Another limitation of our current framework is that it only supports manipulating single objects. For behaviours that involve multiple objects and parts, we would need a scene graph representation that considers the spatial interactions across different objects.

C Additional related work

C.1 Building libraries versus collecting more data for the single-policy approach

To improve the performance and capacity of the single-policy approach, the cost is not only about collecting more data, it is also about scaling up the size of your models, computation, training schema, and parameter search if you aim to get a general image-to-action model. Unfortunately, this often needs an unreasonable amount of data and computation to train [5, 6], and we haven't seen much success in achieving general robot manipulation. Another solution is to put engineering-effort into designing specialized architectures or representations for your application, e.g., the DexNet architecture is specialized for top-down grasp. We see this type of approach can achieve better results compared to the end-to-end approach with deep neural networks. However, it is often restricted to the specific domain it is designed for.

For the library of behaviors approach, the main effort is in creating the behaviors. Fortunately, when designing new behaviors for the library approach, there is no restriction on the input-output the model should use and the algorithm to process them, which makes the engineering work relatively simple. Aside from engineered solutions, recent advances in deep RL and large-scale imitation learning from human demonstration will be a promising approach to scale up the behaviors. Since any behavior, no matter whether it is learned or engineered, can be put in the library, we can potentially combine the efforts from both traditional robot controls and modern deep learning approaches.

Another key issue is how one can update the model to include a new set of skills. For the single-policy approach, one will need to retrain the whole system with new data, and it is not guaranteed that tasks that have been tackled before can be successfully tackled after the retraining. On the contrary, explicitly maintaining a library of behavior makes it simple to incrementally add new skills without breaking existing ones. The only thing that needs to retrain is the selector, which is relatively simple to train.

C.2 Learning to grasp from 2D images

By scaling up real-robot interactions for supervised image-to-action policies, existing work [6, 7] have shown expressive results for grasping diverse objects [7] and can operate under diverse camera viewpoints [6]. However, [7] only works on images in a top-down view from a fixed over-the-shoulder camera. [6] focused on a reaching task where the goal is to touch a target object. They added a fixed script after the touching event that commands the robot to close the gripper and pick the object up, but they have not shown the grasping behavior alone is achieving SOTA results. Both works require significantly more data and computation compared to the proposed model.

C.3 Multi-task and skill learning

Existing works in multi-tasks or skill learning from image inputs [**?** 8, 9, 10] have shown impressive results in learning behaviors from demonstrations or interactions, but they often assume image inputs from a fixed camera view. [6] shows their reaching behavior can generalize to varying views by learning from multi-view data. Our work aims to advance the solution for learning general manipulation skills that can handle various object types, shapes, visuals, initial configurations, and camera viewpoints. Our proposal is to combine readily available behaviors, instead of learning policies from scratch, and to advance the visual representation so the resulting models can be invariant to viewpoint change. We have put genuine effort into the neural architecture and algorithms that can get the right visual representation. In the current paper, we assume the behaviors are given, and we focus on the design choice for the selector. We defer the question of how to automatically develop new behaviors to include in the library in our future work.

C.4 Behavior selection with a classifier

Many strong robotics works can be viewed as classifiers over a set of skills for flying drones [11], grasping [12, 13, 14], transporting [8] and even marble mazes for rolling balls [15]. To generate motion in a continuous space, some works extend the classifier to operate in a continuous space by using a spatial argmax operator over a continuous 2D heatmap [16], and some works use a learned generator to generate candidate behaviors [2, 1] for the selector to choose from. In particular, Transporter [8] generates a score over picking and placing for a discrete set of candidate translations and rotations location in a heatmap. Picking or placing objects at a certain translation and rotation location can be viewed as a behavior, and the generated score could be used by a classifier. Also, to achieve better performance and generalization, a key factor is to choose the right representation space for the classifier to operate on. Our selector works in a representation space that is object-centric and view-invariant, and that is how it can generalize across object appearances, shapes, initial/goal configurations and camera viewpoints. We have empirically shown that naively using 2D representation from 2D CNN layers, which are widely applied in existing work, often cannot achieve good performance.

D Experimental Setup for Pushing

The training objects we use for pushing consists of 60 distinct object meshes from ShapeNet. The detailed makeup of the set of objects is as follows: 6 cameras, 3 caps, 6 baskets, 3 keyboards, 3 earphones, 6 bottles, 6 bowls, 6 cups, 3 cans, 6 buses, 6 cars, and 6 knives.

To increase the diversity on the objects, we perform the following augmentation on the set of object meshes. We generate 4 additional copies of each mesh with randomized scale and color. For scaling, we first scale the whole objects to become larger or smaller, then we scale on one of the dimension to make the object taller or longer. we first uniformly sample a scalar $x_1 \in [0.707, 1.414]$ to apply on the whole object; then, we randomly select a dimension $d \in \{0, 1, 2\}$ and sample a scalar $x_2 \in [0.707, 1.414]$ to apply on this dimension; after sampling these two numbers, we scale the dimension d to clip$(x_1 x_2, 0.707, 1.414)$ and other dimensions to x_1. After the augmentation step, we will have a total of 300 different objects.

At training time, we place objects in their canonical pose at environment reset for most objects. To help our behaviors cover scenarios in which long objects are rotated, we give access to scenarios in which keyboards, buses, cars, and knives are rotated $45°$, $90°$ and $135°$ around the z-axis at environment reset in addition to the default canonical pose. As a result, if we define initial object

configuration as a tuple of (what object is used, what pose of the object is used), then we have a total of $60 \times 5 + (3 + 6 + 6 + 6) \times 5 \times 3 = 615$ initial object configurations.

Test Time Setup The test objects we use for pushing consists of 40 distinct object meshes from ShapeNet. The detailed makeup of the set of objects is as follows: 3 cameras, 2 caps, 3 baskets, 2 keyboards, 2 earphones, 3 bottles , 3 bowls, 3 cups, 2 cans, 3 buses, 3 cars, and 2 knives.

To augment the test objects, we generate 4 additional copies of each mesh, randomizing the scale and the color of each generated object. The scaling procedure of the test objects is the same as that of training objects. In addition to random scaling and color assignment, we also randomly rotate each of the test objects during augmentation. To randomly rotate an object, we first flip the object to place a random side of it on top. To ensure that the flipping is meaningful, we require that cars, cups, and bowls can only be flipped 0° (i.e. canonical pose) or 180° (i.e. upside down); we also require that buses cannot be flipped to the side such that the side with the smallest surface area is facing the floor. After flipping, we randomly rotate the object along z-axis. Please note that this rotation process is very different from training time – at training time, we only rotate selected objects 45°, 90° and 135° around the z-axis; at test time, we rotate all objects and the rotation process has a lot more randomness than during training. After the augmentation, there are a total of 200 initial object configurations that an agent can encounter at test time.

E Implementation Details

E.1 Constructing Behaviors for Grasping

We manually select 30 different controllers including grasps with the gripper-hand pointed downwards and moving along a vertical axis with different yaw orientations (top-grasps) and grasps from the side with different elevation angles of the gripper, and the hand moving towards the centroid of the object (side-grasps). We list all the controllers in Table 2.

E.2 Learning Behaviors for Pushing

We use 25 pushing behaviors trained with subsets of the training objects. Among these 25 behaviors, 24 of them are trained on all cameras, all caps, all baskets, keyboards rotated 0°, keyboards rotated 45°, keyboards rotated 90°, keyboards rotated 135°, all earphones, all bottles, all bowls, all cups, all cans, buses rotated 0°, buses rotated 45°, buses rotated 90°, buses rotated 135°, cars rotated 0°, cars rotated 45°, cars rotated 90°, cars rotated 135°, knives rotated 0°, knives rotated 45°, knives rotated 90°, and knives rotated 135°, respectively. The last behavior is the same as the policy used in the *Abstract 3D* baseline.

E.3 Abstract 3D Baseline

For the *Abstract 3D* baseline, the policy takes a concatenation of object absolute position, object relative position, object pose, gripper position, gripper finger pose, and object size as input and outputs a deterministic action. For the actor network of the model, we apply a tanh activation to normalize the action between $[-1, 1]$. Both actor and critic networks consist of 3 fully connected layers of 256 hidden units with ReLU activations.

The baseline is trained on 60 randomly sampled training objects in the set of training objects. During training, rollout experience is collected by running a vectorized environment consisting of 60 individual environments, each using one of the 60 selected objects. In the training of this RL policy, as well as all the following model-free RL policies, we terminate the training when the model reaches a 95% success rate for 15 consecutive epochs or has not improved for 15 consecutive epochs after the 100th epoch.

E.4 Abstract 3D + Image Baseline

The *Abstract 3D + Image* baseline in the pushing task takes 128×128 images of the environment's front-view as input. When the policy receives the 3D states and images as inputs, it first feeds the image into an encoding network composed of 4 CNN layers similar to those of [17]. The outputs

Type	$(\alpha, \beta, \gamma, \eta)$	Description
top-grasps	([0, 0, 1], [0, 0, -0.010], 90, -90) ([0, 0, 1], [0, 0, -0.010], 90, -60) ([0, 0, 1], [0, 0, -0.010], 90, -30) ([0, 0, 1], [0, 0, -0.010], 90, 0) ([0, 0, 1], [0, 0, -0.010], 90, 30) ([0, 0, 1], [0, 0, -0.010], 90, 60) ([0, 0, 1], [0, 0, -0.010], 90, -90) ([0, 0, 1], [0, 0, -0.023], 90, -60) ([0, 0, 1], [0, 0, -0.023], 90, -30) ([0, 0, 1], [0, 0, -0.023], 90, 0) ([0, 0, 1], [0, 0, -0.023], 90, 30) ([0, 0, 1], [0, 0, -0.023], 90, 60)	Top-grasps with varying yaw orientations and depths.
top-grasps	([0, -1, 1], [0, 0, -0.010], 90, 90) ([0, 1, 1], [0, 0, -0.010], 90, 0) ([1, 0, 1], [0, 0, -0.010], 90, 90) ([0, -1, 1], [0, 0, -0.023], 90, 90) ([0, 1, 1], [0, 0, -0.023], 90, 0) ([1, 0, 1], [0, 0, -0.023], 90, 90)	Top-grasps from the top edges of the bounding boxes.
side-grasps	([0, -1, 1], [0, 0, -0.023], 81, 90) ([0, 1, 1], [0, 0, -0.023], 81, 0) ([1, 0, 1], [0, 0, -0.023], 81, 90)	Top-grasps with slightly lower elevations.
side-grasps	([0, 1, 1], [0, 0, -0.023], 72, 0) ([0, 1, 1], [0, 0, -0.023], 63, 0) ([0, 1, 1], [0, 0, -0.023], 54, 0) ([0, 1, 1], [0, 0, -0.023], 45, 0) ([0, 1, 1], [0, 0, -0.010], 81, 0) ([0, 1, 1], [0, 0, -0.010], 63, 0) ([0, 1, 1], [0, 0, -0.010], 54, 0) ([0, 1, 1], [0, 0, -0.010], 45, 0) ([0, 1, 1], [0, -0.02, -0.004], 45, 0)	right-handed side-grasps with varying elevations. Note that the last controller is designed for grasping flat objects. The gripper starts from a low elevation and pushes a bit towards the center of the object while grasping.

Table 2: Controllers used in the proposed model.

of the CNN encoding layers are passed through a spatial softmax layer [18], flattened into a vector, and then concatenated to the abstract 3D state (note that compared to [17], we use the abstract 3D state instead of robot end-effector positions). In addition to the abstract 3D state, we also take the output of the spatial softmax layer, send it through a fully connected layer to perform an auxiliary object position prediction task, and then concatenate the prediction to the concatenated vector. The rest of the policy architecture consists of 3 fully connected layers identical to those of the *Abstract 3D* baseline.

In contrast to the *Abstract 3D* baseline, we find out that the *Abstract 3D + Image* baseline achieves the best performance when trained using behavior cloning. Concretely, we collect 15 successful expert demonstrations from each of the 615 objects, resulting in a total of 9225 demonstrations. During training, we optimize a composite loss defined by $\mathcal{L}(\theta) = \lambda_{l1}\mathcal{L}_{l1} + \lambda_{l2}\mathcal{L}_{l2} + \lambda_{aux}\mathcal{L}_{aux}$, where θ denotes all parameters in the network architecture, $\mathcal{L}_{l1} = \|\pi_\theta(o_t) - u_t\|_1$ is the L1 action loss, $\mathcal{L}_{l2} = \|\pi_\theta(o_t) - u_t\|_2^2$ is the L2 action loss, $\mathcal{L}_{aux}$ is the auxiliary task loss, $\lambda_{l2} = 1.0$, $\lambda_{l1} = 0.1$, and $\lambda_{aux} = 0.1$.

In the paper, we present the final performance of training the framework on each training object using 5 different views facing $-90\,\text{deg}$, $-45\,\text{deg}$, $0\,\text{deg}$, $45\,\text{deg}$, and $90\,\text{deg}$ relative to the front of the table, and then testing the trained model on test objects using randomly selected views among these 5 views. In addition to this result, we also trained the same framework using single view training data and tested it on both single view test data and multi-view test data. The comparison of the performance of these variations on the pushing task are shown in Table 3.

As seen in the results, the baseline performs slightly worse when trained using multiple views than when trained with a single view. This shows that the generalization challenge posed by changing

viewpoints impairs the overall performance of the framework. Another insight obtained by testing the framework trained on single-view observations on both the single-view test setup and the multi-view test setup is that, while the framework takes both images and abstract states as inputs, the framework does not only rely on the abstract states but also utilizes information from the images, which is why there is a huge performance drop when the model is tested on images taken from perspectives it has never seen before.

Training Views Testing Views	Single-view Single-view	Single-view Multi-view	Multi-view Mutli-view
Success Rate	0.75	0.13	0.70

Table 3: Success rates on training and testing the *Abstract 3D + Image* baseline in different viewpoint setups in the pushing task.

E.5 Contextual 3D Baseline

In the Contextual 3D baseline, we leverage the DAGGER [19] algorithm to obtain the policy through behavior cloning, as we find out that a policy using 3D features as internal representations is very hard to learn directly via RL. To obtain expert labels for policy rollouts, we use distinct controllers (for grasping) and learned policies (for pushing) for each object in the training set. During behavior cloning training, we sample rollouts simultaneously from 15 randomly selected object configurations within the set of all possible training time object configurations and imitate the actions labeled by the experts.

As for the architecture of the *Contextual 3D* baseline, the policy reads multi-view RGB-D images of size $4 \times 64 \times 64 \times (3+1)$ and obtain the 3D feature map $\mathbf{M}_t \in \mathbb{R}^{64\times64\times64\times32}$ using GRNNs [20]. We then encode the feature map to an embedding vector of length 256 with four 3D convolution layers and concatenate the vector with a vector of environment and robot states with length 10. We then send this concatenated vector of length 266 through a MLP with layer output sizes 64, 32, and action space dimensionality to predict actions. Due to the speed of the training, we select the image size of 64 and a DAGGER training batch size of 16.

F Real Robot Experiment Setup

F.1 Task Description

In the real robot experiment, the task is to transport diverse food ingredients from one side of the table to a plate at the other side of the table. To ensure the diversity of the objects our model will be tested with, we use a total of 20 rigid food ingredients, 20 granular food ingredients, and 12 bottles of sauces from a local supermarket. We then split these objects to a training set and a test set, where 75% of objects in each category are placed in the training set and rest in the test set.

To collect the training set, we place each rigid and granular food ingredient in 3 different initial poses and each liquid ingredient in 5 different initial poses and record the success rate of running each of our 26 behaviors on every object and initial pose combination. This results in a total of 135 data points for training the behavior selector model.

At test time, we place each food ingredient in different initial poses in the same way as the training set and test the performance of our model by running one episode for each of the 45 object and initial pose combinations.

F.2 Building the Behavior Library

To build a behavior library for the real robot setup, we use a total of 26 controllers, 13 of which instances of pick-and-place behaviors and 13 of which pick-and-pour behaviors. The motivation of including both types of behaviors is that the pick-and-place ones are created for manipulating rigid objects, while granular objects and liquids need to be placed inside containers and poured onto the plate. Below we describe the 26 controllers we use in more detail:

- *Pick-and-place from Top:* we include 5 pick-and-place controllers that grasp the object from the top, pick it up, and place it at a specified position. The 5 controllers each uses a grasping pose rotated by $0°, 45°, 90°, 135°$, and $180°$ around the z-axis.
- *Pick-and-place from Rim:* we include 5 pick-and-place controllers that grasp the object from its rim, pick it up, and place it at a specified position. The 5 controllers each grasp the left, front-left, front, front-right, and right side of the object from the top.
- *Pick-and-place from Side:* we include 3 pick-and-place controllers that grasp the object horizontally from the side of it, pick it up, and place it at a specified position. The 3 controllers each grasp from the left, front-left, and front of the object from the side. We do not include other orientations because the gripper cannot reliably reach these orientations in most cases.
- *Pick-and-pour from Top:* we include 5 pick-and-pour controllers that grasp the container that includes the object of interest from the top, pick it up, and pour the object of interest from the container to a desired position. The grasping poses used by the 5 controllers are the same as those of the *Pick-and-place from Top* controllers.
- *Pick-and-pour from Rim:* we include 5 pick-and-pour controllers that grasp the container in the same way as the *Pick-and-place from Rim* controllers but pour the content of the container in the same way as the *Pick-and-pour from Top* controller.
- *Pick-and-pour from Side:* we include 3 pick-and-pour controllers that grasp the container in the same way as the *Pick-and-place from Side* controllers but pour the content of the container in the same way as the *Pick-and-pour from Top* controller.

To obtain object positions, we match the center and bounding box of the object detected from RGB images in 5 different views to get the object's 3D centroid and bounding box coordinates. Although this grasping process is open-loop, it works well due to the high accuracy of the low level Franka robot controller. Using visual feedback within an episode for closed loop control is an important topic for future work.

F.3 Training Details

We train our behavior selector from scratch for the real robot experiment. With a dataset of success labels collected by running the 26 behaviors on the training objects, we train the model for 5,000 update steps and use the trained model to assess the success rate of grasping test-time objects. To speed up data collection, we include some heuristics: when asked to pour something, we label failure for all grasping controllers; when grasping long objects, we label obviously infeasible grasping angles as failures.

F.4 Qualitative Samples

To better understand the performance on the real robot, we prepared test-time rollouts of our framework, which contain both success and failure cases. Please look into supplementary video for these qualitative results.

References

[1] A. Mousavian, C. Eppner, and D. Fox. 6-dof graspnet: Variational grasp generation for object manipulation. *CoRR*, abs/1905.10520, 2019. URL http://arxiv.org/abs/1905.10520.

[2] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. 07 2017. doi:10.15607/RSS.2017.XIII.058.

[3] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1509.html#PintoG15.

[4] S. Young, D. Gandhi, S. Tulsiani, A. Gupta, P. Abbeel, and L. Pinto. Visual imitation made easy, 2020.

[5] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[6] F. Sadeghi, A. Toshev, E. Jang, and S. Levine. Sim2real view invariant visual servoing by recurrent control. *CoRR*, abs/1712.07642, 2017. URL http://arxiv.org/abs/1712.07642.

[7] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293.

[8] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *CoRR*, abs/2010.14406, 2020. URL https://arxiv.org/abs/2010.14406.

[9] Y. Lee, J. Yang, and J. J. Lim. Learning to coordinate manipulation skills via skill behavior diversification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxB2lBtvH.

[10] K. Fang, Y. Zhu, S. Savarese, and L. Fei-Fei. Discovering generalizable skills via automated generation of diverse tasks. *CoRR*, abs/2106.13935, 2021. URL https://arxiv.org/abs/2106.13935.

[11] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017.

[12] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *CoRR*, abs/1509.06825, 2015. URL http://arxiv.org/abs/1509.06825.

[13] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019.

[14] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019.

[15] D. C. Bentivegna, G. Cheng, and C. G. Atkeson. Learning from observation and from practice using behavioral primitives. In P. Dario and R. Chatila, editors, *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 551–560. Springer, 2003. doi:10.1007/11008941_59. URL https://doi.org/10.1007/11008941_59.

[16] L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T.-Y. Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. URL https://yenchenlin.me/vision2action/.

[17] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *ICRA*, pages 1–8. IEEE, 2018.

[18] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[19] S. Ross, G. Gordon, and D. 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, 2011.

[20] H.-Y. F. Tung, R. Cheng, and K. Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.