Library of behaviors and tools for robot manipulation

Yunchu Zhang

CMU-RI-TR-22-41

July 26, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Christopher G. Atkeson, *chair*
Katerina Fragkiadaki, *co-chair*
Oliver Kroemer
Thomas Weng

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To all my family members, friends and people who inspired me.

Abstract

Learned policies often fail to generalize across environment variations, such as, different objects, object arrangements, or camera viewpoints. Moreover, most policies are trained and tested in simulation environments, and the simtoreal gap remains large under weak visual representations that do not disentangle the scene from the objects in it.

We first propose a visually-grounded library of behaviors approach for learning to manipulate diverse objects across varying initial and goal configurations and camera placements. Our key innovation is to disentangle the standard image-to-action mapping into two separate modules that use different types of perceptual input: (1) a behavior selector which conditions on *intrinsic* and *semantically-rich* object appearance features to select the behaviors that can successfully perform the desired tasks on the object in hand, and (2) a library of behaviors each of which conditions on *extrinsic* and *abstract* object properties, such as object location and pose, to predict actions to execute over time. We test our framework on pushing and grasping diverse objects in simulation as well as transporting rigid, granular, and liquid food ingredients in a real robot setup. Our model outperforms image-to-action mappings that do not factorize static and dynamic object properties.

We then propose an end-to-end learning framework that jointly learns to choose different tools and deploy tool-conditioned policies with a limited amount of human demonstrations directly on a real robot platform. It is important to correctly switch between and deploy suitable tools in object rearrangement and cleaning tasks in complex scenes. We evaluate our method on parallel gripper and suction cup picking and placing, sweeping with a brush, and household rearrangement tasks, generalizing to different configurations, novel objects, and cluttered scenes in the real world. Finally, we show a long-horizon planning framework that could utilize our multiple tool setup to manipulate elastoplastic objects successfully, such as a dough.

Acknowledgments

I would like to thank my advisor, Prof. Katerina Fragkiadaki for your support and guidance throughout my MSR journey. I can still remember our first meeting in your office : your passion and love for research inspired me to work with you on solving fundamental problems in computer vision, robotics and learning. During the past two years, your guidance pushed me to become a better researcher and my skill set was enriched a lot. From you, I learned paper analysis, effective experimental design and and project management.

I would like to thank my co-advisor, Prof. Chris Atkeson for your support and guidance throughout my MSR journey. My goal to become a roboticist was mainly inspired by the Baymax movie and the research projects conducted by Prof. Atkeson's group. My dream to work and learn from you came true when you agreed to accept me as your student. It is my pleasure to have the opportunity to work with you together. I really appreciate your insight thoughts, constructive suggestions in our emails and meetings. A lot of my research confusions on robot learning were resolved by you.

Besides, I really appreciate the rest of my committee members. Thank you to Prof. Oliver Kroemer for taking time out of your busy schedules to serve on my thesis committee. I really enjoy our discussions every time and your ideas are always novel and insightful. Thank you to my friend Thomas Weng. I appreciate all your kind words, encouragement and suggestions during my master. I have learned a lot from you and your research.

I would like to thank Kevin Zhang and Jacky Liang for their help on setting up our lab's Franka robot workspace. I could not have succeeded without their patient and careful instructions.

I would like to thank my collaborators in Katerina's Lab. Thank you to Adam Harley, Ashwini Pokle, Ayush Jain, Fish Tung, Gabe Sarch, Hao Zhu, Jing Wen, Jingyun Yang, Mayank Singh, Mihir Prabhudesai, Nikos Gkanatsios, Paul Shydlo, Shamit Lal, Wen-Hsuan Chu, Xian Zhou, Yiming Zuo, and Zhaoyuan Fang. I will miss the time working with you in the lab. Also, I would like to thank my collaborators outside the lab. Thank you to Carl Qi, Sha yi, Xingyu Lin and Xiaofeng Guo.

Thanks to all my friends for bearing with my complaints during these two

years when I was down.

Last but not least, I would like to thank for my parents and all my family members. It is your unconditional support and love that makes me earn this degree possible.

Funding

This material is based upon work supported by Sony AI, NSF CAREER award and Amazon faculty award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of Sony, Amazon.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

While robotics research has made great progress in solving novel and complicated tasks with data-driven methods, there are still many problems that remain, such as non-robust transfer from simulation to real world, sample inefficiency when training with deep networks, and weak generalization across multiple distinct tasks. Embracing prior knowledge (e.g library of behaviors) and exploring the object/scene affordance representation may be a possible way to address those problems. We assume that humans use libraries of task-specific behaviors with correspondence tools that they have been taught or shown to finish the most of daily tasks. Although those primitives' generalization ability are weak, robots can make analogies and generalize behaviors with shared affordance among different scenes.

1.1 Thesis Organization

The thesis is organized as follows:

- Visually-Grounded Library of Behaviors for Manipulating Diverse Objects across Diverse Configurations and Views [chapter 2] : a visually-grounded library of behaviors approach for learning to manipulate diverse objects across varying initial and goal configurations and camera placements.
- MultiTool Transporter Networks for Object Rearrangement from Demonstrations [chapter 3] : a neural model that given a visual image of the scene, predicts

a tool identity, as well as image-conditioned action parameters for close-loop policy.

- PASTA: Planning with Spatial-Temporal Abstraction from Point Clouds for Deformable Object Manipulation [chapter 4]: a framework that incorporates both spatial abstraction (reasoning about objects and their relations to each other) and temporal abstraction (reasoning over skills instead of low-level actions).

This material is based upon work supported by Sony AI, NSF CAREER award and Amazon faculty award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of Sony, Amazon.

Chapter 2

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

2.1 Introduction

Object manipulation in unstructured environments is challenging since methods to manipulate objects largely depend on the object's visual appearance. One approach to capture the dependence between actions and visual features is to learn a direct mapping from image to actions with deep neural networks [14, 43, 82]. Despite their flexibility, such end-to-end image-to-action mappings have been shown to be data aggressive [29], and cannot easily generalize across objects, camera viewpoints, or scene configurations [7].

Approaches that abstract away object details and encode only a subset of their properties, e.g., their 3D locations and velocities [3] or 3D keypoints [37, 48] make

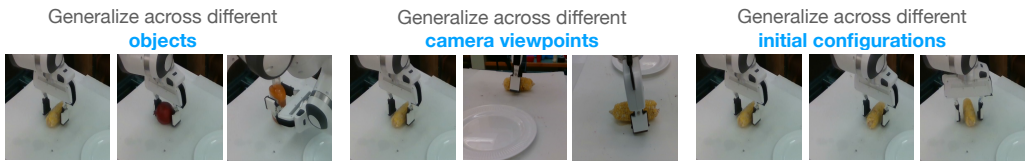

Figure 2.1: We propose a novel policy representation that generalizes to unseen objects, object placements and camera views.

the state-to-action mapping easier to learn with less data. However, this abstraction may substantially limit the range of objects that a policy can handle, since useful information (object shapes, softness, weight, and material, for example) for the downstream task may be ignored. The challenging question is: how can we design a framework for object manipulation that uses **abstract** representations for sample efficient behavior learning, but at the same time is capable of utilizing **semantically-rich** representations for handling diverse objects and views.

We propose Visually-grounded library of BEhaviors (V-BEs), a hierarchical framework for vision-based object manipulation. Our main contribution is that the two levels of our policy hierarchy use different visual representations. At the lower level of the hierarchy, a behavior library contains a set of distinct behaviors each of which operates on an **abstract** object state representation that captures dynamic properties of objects in the environment, such as object and gripper positions over time. Behaviors can be redundant, use different state abstractions, be closed loop or open loop, and be learned or manually engineered with different algorithms [2, 10, 33]. At the higher level of the hierarchy, a selector is trained using environment interactions to associate objects to behaviors that can successfully manipulate them. This selector takes as input RGB-D images of the scene, maps them to **semantically-rich** and view-invariant 3D object feature representations [68], and outputs the behavior that best suits the presented object and task. The decomposition of visual information into a perceptually rich selector network and behaviors with state abstraction makes the behaviors easier to learn with reinforcement learning or be designed by an engineer, and the selector easier to learn by trial-and-error learning as it does not bear the burden of producing low-level actions. At test time, we use 3D object detectors to localize the object, and present its location and size to the low-level controllers when necessary.

Our second contribution is proposing a visual representation for our behavior selector that is robust to changing camera viewpoint (with respect to a non-changing robot torso), as well as object placements. We use the differentiable 2D-to-3D neural networks of Tung et al. [68] to map an input RGB-D image alongside camera intrinsics and extrinsics to transform 3D visual scene feature maps from the current camera viewpoint to the world frame, ensuring that visual features are bound to a static world frame—that coincides with the frame of the agent's body—and do not change dramatically while the camera moves. The 3D visual feature representations learn based on environment interactions to embed objects close to the behaviors that can manipulate them. They are further trained from the auxiliary task of view prediction using a multi-view camera setup, which encourages the model to complete missing information from the current view so the 3D visual feature maps are consistent across views and through occlusions.

We test the proposed model in both simulated and real robot setups. In simulation, we consider two manipulation tasks: pushing and grasping. We show that our model can manipulate diverse objects and generalize the learned skills to unseen objects with varying object starting positions, initial poses, goal locations, and camera viewpoints. We show that our model outperforms existing end-to-end image-to-action mappings [29] and state abstracted object-to-action mappings that use only 3D object locations [3]. We ablate the contribution of our higher-level 3D feature representation used in our selector and then compare against 2D pretrained baselines. Lastly, we test our model in a real robot setup where the robot transports rigid, granular, and liquid objects onto a plate. We provide supplementary materials including additional qualitative results at https://yunchuzhang.github.io/vbes.github.io/.

2.2 Related Work

Libraries of Behaviors: Libraries of behaviors have been considered in many previous works, and library elements have been called action, motion, behavioral, or motor primitives, parameterized policies, options, chunks, macros, subroutines, behavioral units, and skills, as well as many other terms. However, most previous

work either assumes that the environment state is known and does not use perception, or has a different approach to perception than ours. Work of Neumann et al. on modular policies [41] defines a motor primitive as a mapping from a robot's joint angles and velocities and not the locations of objects. Rather, object locations are used as context for selecting a motor primitive. This means i) motor primitives are "blind" to the object 3D location throughout the episode, and ii) object appearance is discarded which means the library of primitives cannot handle object variability. Work of de Silva [59] on parameterized policies similarly considers as the contextual vector the desired goal location of a dart, and learns a mapping from this location to policy parameters, where each policy operates only over the robots' joints and is represented as a Dynamic Motion Primitive (DMP) [55, 57, 63]. To extend DMPs to tasks involving physical interactions, Kroemer et al. [26] parameterize DMPs based on the distance of an object part from the gripper, but only consider known object 3D shapes. Strudel et al. [62] use a depth frame as input to a meta-policy that selects skills to execute. However, the use of modularity over skills is for temporal sequencing: there is no mixture of behaviors per manipulation task, rather an image to action mapping is learned with behavior cloning from demonstrations.

Deep Reinforcement and Imitation Learning for Object Manipulation: Current work in (deep) reinforcement learning typically considers only one monolithic policy learned via reinforcement learning [2, 25, 71], imitation learning [51, 58, 82], or a combination of the two [8, 50]. Most approaches take as input 3D object and gripper centroids, orientations, and velocities, and ignore contextual visual information [42]. Existing methods are commonly trained in a fixed environment and evaluated in terms of their ability to discover a policy rather than their ability to generalize to previously unseen circumstances, e.g., novel objects and views, which is the goal of this work.

Methods that do aim to generalize across objects learn a mapping from images-to-actions [29], depth-to-actions [35] or pointcloud-to-actions [40, 47]. They have been successful in various tasks including object grasping [35, 44] and object pushing from a fixed camera view [1, 29]. Seminal works of [29, 65] train an image-to-action deep neural network by imitating trajectories obtained from planners that operate in a low-

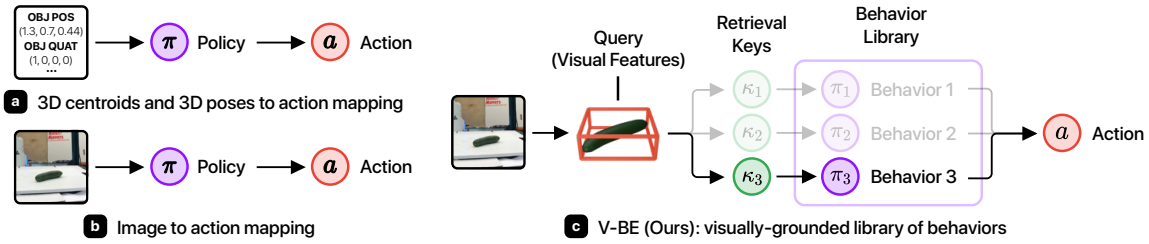

Figure 2.2: Comparison of our model with previous approaches. In contrast to state-to-action or image-to-action mapping, the proposed framework decomposes a policy into a behavior selection module and a library of behaviors to select from. The decomposition enables these modules to work on different representation: the selection module operates in a semantically-rich visual feature space, while the behaviors operates in an abstract object state space that facilitate efficient policy learning.

dimensional state space. Nevertheless, such monolithic image-to-action deep mappings don't generalize well across different camera viewpoints [7, 46, 52] and objects. Works that attempt to transfer visuomotor policies learned in simulation to the real-world often require identical placement of the camera in the real world [3, 22]. Florence et al. [13] train manipulation policies parameterized by the 2D or 3D locations of a designated set of visual descriptors obtained from RGB-D images, and show their policies generalize across objects of the same object category, e.g., shoes of different shapes. The proposed work further allows generalization across object categories, and no explicit descriptor selection or optimization is necessary. Furthermore, our framework permits each behavior to use a different state representation.

Hierarchical Reinforcement Learning (HRL): Most existing HRL methods [27, 61] focus on discovering behaviors and temporally sequencing them for long-horizon tasks. This paper instead uses a hierarchical vision-based policy architecture to improve model generalization across objects, configurations, and viewpoints.

2.3 Methods

Problem Setup We consider the problem of manipulating (e.g. grasping, pushing, etc.) diverse sets of objects from various initial configurations and camera viewpoints.

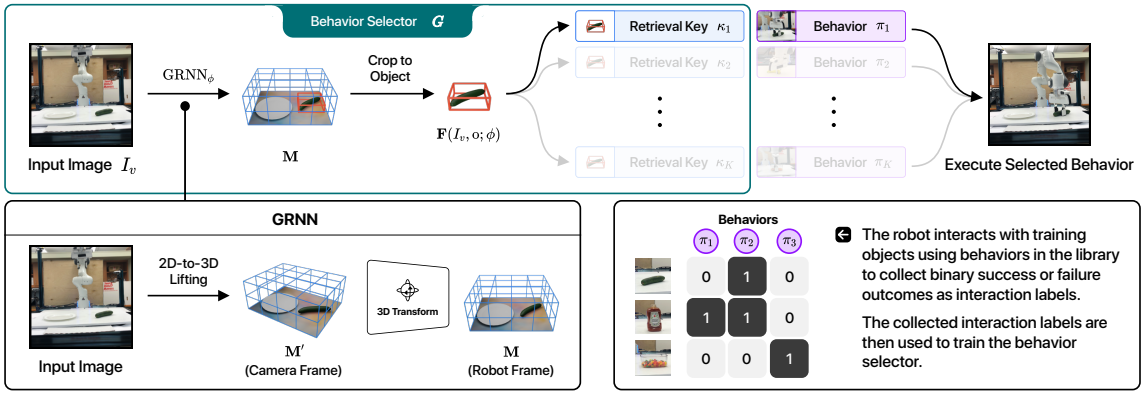

Figure 2.3: Overview of the proposed framework.

At training time, the agent has access to training objects $\mathcal{O}_{\text{train}}$, RGB-D images from J viewpoints $v_1, \ldots, v_J \in \mathcal{V}$, and it can interact with the training object from randomly selected viewpoints. At test time, the agent has to perform the same manipulation task as at training time on test objects $\mathcal{O}_{\text{test}}$ from selected viewpoints. Performance is measured by the success rate of manipulating all test objects with random initial configurations and camera viewpoints.

Our framework is comprised of two major components — (1) a library of behaviors $\Pi = \{\pi_i \mid i = 1, 2, \ldots, K\}$ and (2) a selector function G. Given an RGB-D image I of the scene captured from camera view v, denoted by $I_v = \{I, v\}$, and an object 3D bounding box o, the selector G obtains the probability of successfully manipulating the object when applying behavior π_i on the object by computing a query object feature representation $\mathbf{F}(I_v, \text{o})$ and compares it with learned key embeddings $\kappa_i = \kappa(\pi_i)$, $i = 1, ..., K$ associated with each behavior π_i,

$$\text{G}(I_v, \text{o}, \pi_i; \phi, \kappa) = \sigma(\langle \mathbf{F}(I_v, \text{o}; \phi), \kappa_i \rangle) \in [0, 1], \tag{2.1}$$

where ϕ is the learnable neural network parameters for function $\mathbf{F}$, $\langle \cdot, \cdot \rangle$ is the inner product operation and σ is the sigmoid function. We will detail the feature dimension of $\mathbf{F}(I_v, \text{o}; \phi)$ and κ in the next section. At test time, we select the behavior π_i that has the highest probability of leading to successful manipulation.

Each behavior π_i uses its own abstract state representation for the objects to manipulate, for example, 3D object locations and 3D poses, and does not necessarily take

into account other aspects of the input image. Each behavior is designed or learns to handle only a subset of object shapes and orientations. To handle a diverse set of objects and orientations, our selector learns to pick different behaviors for different objects or orientations. One advantage of the modularity of our policy architecture is that each behavior can use its own state abstraction, such as object 6D-poses, 3D bounding boxes, part-based 3D boxes, or 2D or 3D object keypoint locations [48]. Our framework supports integrating a wide range of existing models, representations, and controllers as selectable behaviors.

In the rest of this section, we detail the architecture and training details of the behavior selector in Section 2.3.1, and describe how we acquire a library of behaviors for grasping and pushing—the two robot manipulation tasks we evaluate our framework on—in Section 2.3.2.

2.3.1 Visually-Grounded Behavior Selector (G)

The behavior selector G is a classifier that learns object-centric visual feature representation $\mathbf{F}(I_v, \mathrm{o}; \phi)$ for the object box o in image view I_v, and behavioral key embeddings κ^i $(i = 1, \ldots, K)$ for the behaviors in the library to retrieve behaviors compatible with a particular object.

Training the selector with interaction labels. We learn the object feature representation and behavioral keys through trial-and-error. In each trial, our agent applies a randomly sampled behavior π_i on an object o in the workspace which results in binary success or failure outcome $\ell \in \{0, 1\}$, which we call *interaction labels*. Agent interactions are organized as tuples of the form $(I_v, \mathrm{o}, \pi_i, \ell)$. With the interaction experience as training data $\mathcal{T}$, we train the feature encoder f and behavioral keys $\kappa_i = \kappa(\pi_i)$ $(i = 1, \ldots, K)$ with the loss:

$$\mathcal{L}_{\text{afford}}(\kappa, \phi) = \sum_{(I_v, \mathrm{o}, \pi_i, \ell) \in \mathcal{T}} \mathrm{BCE}(\ell, \mathrm{G}(I_v, \mathrm{o}, \pi_i; \phi, \kappa)) = \sum_{(I_v, \mathrm{o}, \pi_i, \ell) \in \mathcal{T}} \mathrm{BCE}(\ell, \sigma(\langle \mathbf{F}(I_v, \mathrm{o}; \phi), \kappa_i \rangle)) \tag{2.2}$$

where $\text{BCE}(y, p) = -y \cdot \log(p) - (1 - y) \log(1 - p)$ is the binary cross entropy loss.

3D object feature representation. Our object feature representation $\mathbf{F}(I_v, \text{o}; \phi)$ is computed using Geometry-aware Recurrent Neural Networks (GRNNs) of Tung et al. [68], which are end-to-end differentiable architectures that map a single RGB-D image or a set of multi-view images to 3D feature representation of the scene the image(s) depict. Given a posed image I_v, GRNNs obtain scene-centric 3D feature representation $\mathbf{M} = \text{GRNN}_\phi(I_v) \in \mathbb{R}^{H \times W \times D \times C}$ through differentiable 2D-to-3D operations, 3D convolution-based refinement, and 3D rotation operations that align the 3D feature representations with the robot's coordinate frame (as opposed to camera frame), with W, H, D denoting the width, height and depth of the scene map and C denotes the feature dimension of the 3D scene feature map.

From the scene map $\mathbf{M}$, we obtain object-centric feature representation $\mathbf{F}(I_v, \text{o}; \phi) = \text{crop}(\mathbf{M}, \text{o}) \in \mathbb{R}^{64 \times 64 \times 64 \times 32}$ by cropping the scene map using a fixed-size axis-aligned box o, centered around the object we wish to manipulate. The feature cropping operation is similar to the one used in Mask-RCNN [20]. The retrieval policies keys are also learned in the same representation space. Thus, both $\mathbf{F}(I_v, \text{o}; \phi)$ and κ_i, $(i = 1, \ldots, K)$ have length $64 \times 64 \times 64 \times 32$ in the experiments.

3D object detector. We learn a detector with 3D Mask-RCNN built on top of the GRNNs feature encoder [68]. We use groundtruth 3D object boxes at training time, and predicted 3D object boxes at test time, where we train our representation to detect objects in 3D.

View prediction and occupancy prediction as an auxiliary task. We use view prediction and occupancy prediction as an auxiliary task to help our image encoder generalize better in its ability to select behaviors. These two self-supervised prediction tasks have been shown to provide a useful pretraining or co-training objective for 3D object detection in [18]. Given an input posed image I_v^n and a query view q^n, the

overall self-supervised prediction loss reads:

$$\mathcal{L}_{\text{self-pred}}(\phi, \theta, \eta) = \sum_{n=1}^{N} \underbrace{\|P_\theta(\text{GRNN}_\phi(I_v^n), q^n) - \bar{I}_q^n\|_2^2}_{\text{view prediction loss}} + \underbrace{\|\text{Occ}_\eta(\text{GRNN}_\phi(I_v^n)) - \text{occ}^n\|_1}_{\text{occupancy prediction loss}}, \tag{2.3}$$

where $P_\theta(\mathbf{M}, q)$ is a projection function that projects a 3D feature map $\mathbf{M}$ from the query viewpoint q to a 2D feature map and decodes it to a target image $\bar{I}_q^n$ using an image decoder with neural network weights θ, $\text{Occ}_\eta(\mathbf{M}) \in \mathbb{R}^{64\times64\times64}$ is a voxel occupancy prediction function that predicts a 3D occupancy map from an input 3D feature map $\mathbf{M}$ using a single 3D convolution layer with weights η, and occ^n is the estimated occupancy map computed from all available input views in the n^{th} data point by voxelizing the unprojected point clouds from all available depth images. We train the model from unlabelled multi-view images captured around the table by simply moving the cameras, capturing the images, and recording the corresponding camera locations.

The final objective for training the affordance-based visual features is

$$\underset{\kappa,\phi,\theta,\eta}{\text{minimize}}\, \mathcal{L}(\kappa, \phi, \theta, \eta) = \mathcal{L}_{\text{self-pred}}(\phi, \theta, \eta) + \lambda_{\text{a}} \cdot \mathcal{L}_{\text{afford}}(\kappa, \phi), \tag{2.4}$$

where λ_{a} is a hyperparameter for balancing the two losses.

2.3.2 Building a Library of State Abstracted Behaviors

Any existing behaviors, whether engineered or learned using reinforcement or imitation learning, can be included in our library. This flexibility is a contribution of our modular architecture. In this paper, we consider three common manipulation tasks: pushing, grasping, and transporting. We build appropriate behavior libraries for each.

In pushing, the behaviors are deterministic goal conditioned policies $a_t = \pi(s_t, g)$ that map a state of the environment and the robot $s_t = [s_t^e, s_t^r]$ and a goal state g to an action a_t at time step t. The environment state s_t^e is the 3D object centroid and the robot state s_t^r is the gripper 3D location, pose, and whether it is opened or

closed. Actions include 3D translation, opening (position control), and closing (force control) of the gripper. A goal state g is a target centroid location for the object. We use a total of 25 goal conditioned policies – one is trained from the whole set of objects, while the others are trained on disjoint subsets of object configurations organized based on object category and initial poses. We train all policies using deterministic policy gradients (DDPG) [50] with goal relabelling (HER) [2] while randomizing initial and goal object 3D locations.

In grasping, we design controllers $\pi(a_t|g; p^{grasp}, q^{grasp})$ which given a 3D grasping point $p^{grasp} \in \mathbb{R}^3$ relative to the center of the object and a grasping 3D angle $q^{grasp} \in \mathbb{R}^2$, move the gripper (open loop) to the grasping 3D point location, close it, and move it to the desired goal location. The grasping angle q^{grasp} consists of two numbers describing the yaw of the gripper and the elevation angles between the gripper and the table surface. When the elevation angle is smaller than 90 degrees (not top-down grasps), we constrain the gripper to point toward the center of the object on the x-y plane. We manually select 30 different controllers including top-down grasps with different yaw orientations (top-grasps) and grasps from the side with different elevation angles of the gripper (side-grasps). We empirically found that these parameterized controllers are quite stable and can be shared across multiple objects. More details are provided in the supplementary materials.

2.4 Experiments

Our experiments aim to answer the following questions: (1) Does the proposed library-based approach outperform existing methods that use a single combined perception and policy module, either using 2D images, 3D object locations, or 3D scene feature maps as input? (2) Is the proposed view-invariant and affordance-aware 3D feature representation a necessary choice for the selector? (3) Does the method work on a real robot? We test our model on grasping and pushing a wide variety of objects in the MuJoCo simulator [64] and further test a transporting task on a real-world Franka Panda robot arm.

2.4.1 Simulation Experiments

Our simulated environment consists of a Fetch Robot equipped with a parallel-jaw gripper. The robot is positioned in front of a table of height $0.4m$. To obtain the visual observations, on each episode we choose 3 random cameras from cameras placed at 30 nominal different views including 10 different azimuths ranging from 0° to 360° combined with 3 different elevation angles from 20°, 40°, 60°. All cameras are looking at the center of the table top, and are 0.5 meter away from that point. All images have size 128×128.

Task Descriptions: In the grasping task, the agent has to grasp an object and move it to a specified target location above the table. We use 274 distinct object meshes from 6 categories in ShapeNet [5] including toy buses, toy cars, cans, bowls, plates, and bottles. The materials and densities of all objects are identical. We randomly split the dataset into 207 training objects, and 67 testing objects. After augmenting the meshes with random scaling from 0.8 to 1.5 and random rotations around the vertical z-axis, we get a total of 800 distinct object configurations (object instance and pose), 600 for training and 200 for testing. At the start of each episode, an object is placed in an area of $30cm \times 16cm$ around the center of the table, and a goal is sampled uniformly $10 \sim 30cm$ away from the gripper's initial position. An episode is successful if the object centroid is within $5cm$ of the target at the final timestep.

In the pushing task, the agent has to push an object placed on the table to a specified target location. We use 100 objects from 12 categories in ShapeNet [5]: baskets, bowls, bottles, toy buses, cameras, cans, caps, toy cars, earphones, keyboards, knives, and mugs. After augmentation and splitting to train and test sets, we obtain 615 training object configurations and 200 for testing. The initial and the goal position of the object are both uniformly sampled to be within $15cm$ of the center of the table along both x-axis and y-axis, although we resample if that location is already in the goal area. An episode is successful if the object centroid is within $5cm$ of the goal within 50 timesteps.

Baselines: We compare our method with various learning and non-learning based methods for object manipulation:

We compare our method with various learning and non-learning based methods for object manipulation:

1. *Single Behavior w/Abstract 3D State (Abstract 3D) [2, 50]:* a policy takes as input ground truth 3D bounding box of the object and gripper and outputs actions.
2. *Single Behavior w/Abstract 3D State and 2D Images (Abstract 3D + Image):* a policy takes both RGB-D images and the ground truth 3D bounding box as inputs and outputs actions. Our architecture resembles that of [81],

   but we further include ground truth object position as extra inputs to the model. For fair comparisons to other methods, the model only takes as input the current state as opposed to the states in 5 past steps, as in [81].
3. *Single Behavior w/3D Feature Tensor (Contextual 3D):* a policy takes as input RGB-D images and the ground truth 3D bounding box and outputs actions. Different from (b), the model first transforms the image into a view-invariant 3D feature tensor using GRNNs [68], then converts the 3D feature tensor into a feature vector though three 3D-convolutional layers and a fully connected layer, and concatenates it with the rest of the inputs to predict actions.
4. *Ours, Library of Behaviors w/ Visual Selector (V-BEs):* Our model takes the same input as (b) and (c). The 3D bounding boxes are used as input to all the behaviors. The RGB-D images are transformed into 3D affordance-aware visual features and treated as input to the selector.

We train the baselines with different learning methods including behavior cloning [29], DDPG-HER [2, 33] and DAGGER [54]. We report the best performance we got by training the model with these different methods. We also attempt to make all the models have similar number of parameters so the comparison is fair. However, larger networks are empirically harder to train and do not converge well, so we instead increase the number of parameters in smaller networks until their performance saturates. For pushing, we found that using DDPG-HER is enough to lean a good *Abstract 3D* policy from scratch. For *abstract 3D + Image*, we found it is critical to use behavior cloning from expert demonstrations to obtain good policies. The

	Single Behavior			Library of Behaviors	Ablation study on the selector's visual feature representation	
	Abstract 3D [2, 50]	Abstract 3D + Image	Contextual 3D	V-BEs (Ours)	V-BEs w/ 2D features	V-BEs w/o Fine-tuning on Interaction Labels [17, 68]
grasping	0.30	0.35	0.20	**0.78**	0.46	0.31
pushing	0.83	0.70	0.10	**0.88**	0.81	0.46

Table 2.1: Success rates on grasping and pushing unseen objects. We also ablate the proposed method with selectors operating on varying representations.

expert demonstrations are obtained from trained expert policies on single objects. For *Contextual 3D*, we include DAGGER to enforce behavior cloning during execution. To train the grasping policies, we further include human demonstrations in the replay buffer when training it with DDPG-HER. Both *abstract 3D + Image* and *Contextual 3D* are trained with DAGGER since offline behavior cloning is insufficient.

2.4.2 Single Behavior versus a Library of Behaviors

We compare the proposed model with models that do not use a library-based approach, i.e., single behavior approaches. As shown in Table 2.1, our method outperforms all the single behavior baselines. *Abstract 3D* performs well, but since it does not use any visual information, its performance saturates at around 0.8 for pushing and 0.3 for grasping. *Abstract 3D* performs poorly for grasping. The learned behaviors do not transfer well to new objects. Adding a 2D image helps, but not dramatically (see *Abstract 3D + Image* in Table 2.1). Although 3D feature maps obtained from GRNNs are semantically rich and can handle varying viewpoints, the mapping to actions is harder to learn due to the higher dimensionality of the 3D scene map, resulting in under-fitting models. Our model takes advantage of both abstract and semantically rich representation and thus can handle better object variability and transferability. The combinatorial nature of the proposed method allows the model to capture the multi-modality in trajectory generation.

2.4.3 The Necessity of Building the Selector with the Proposed 3D Representations

Next, we show the importance of using view-invariant 3D visual feature representations and fine-tuning the selector with interaction labels. We compare our method with two baselines: (a) a model with a selector that learns the visual affordance features using 2D visual features extracted from 2D CNNs, and (a) a model with a selector that operates over 3D visual feature representation learned only with the view and occupancy prediction loss, as suggested in [17, 68], without fine tuning with interaction labels. See Table 2.1 for the results. Our method significantly outperforms these two baselines, which shows the importance of both proposed components. To fully test the power of existing 2D CNNs, we also tested 2D feature selector with existing VGG network [60] pretrained on ImageNet and fine-tuned on our interaction labels. However, the performance (a success rate of 0.78 on pushing) does not differ too much with shallower 2D CNNs trained from scratch.

2.4.4 Transporting Task on a Real Robot

We test our model on a 7-DOF Franka robot arm equipped with a parallel-jaw gripper (using [80]'s software stack) for a transporting task, where the robot needs to transport various rigid, granular, or liquid food ingredients from random initial positions and poses onto a plate (see Figure 2.4). We set up 4 Intel RealSense RGB-D cameras that have full view of the workspace around the the center of the table. In each trial, an object is placed in a 50cm $\times$ 30cm region on the table, and the goal is transport all objects to a plate 25cm to the left of the starting region. Granular objects and liquids are placed inside containers in the beginning of the trial. An episode is considered successful if the object is successfully transported into the plate. For granular objects and liquid, an episode is considered successful if at least half of the total quantity ends up in the plate.

We construct our library of behaviors with 26 controllers, in which 13 of them are pick-and-place controllers from various grasping angles and the other 13 are pick-and-pour

controllers using the same grasping angles as the pick-and-place ones. At evaluation, we randomly select 3 views from 4 possible camera views to obtain RGB-D images as inputs to the learned selector. We use 20 rigid objects, 20 granular objects, and 12 bottles of liquids for our experiment, and we split them into 38 training object and 14 testing objects. To train our model, we collect a total of 3510 interaction labels by running the 26 behaviors on the training objects with random initial poses and using the labels to finetune the visual selector with the objective specified in Equation (2.4).

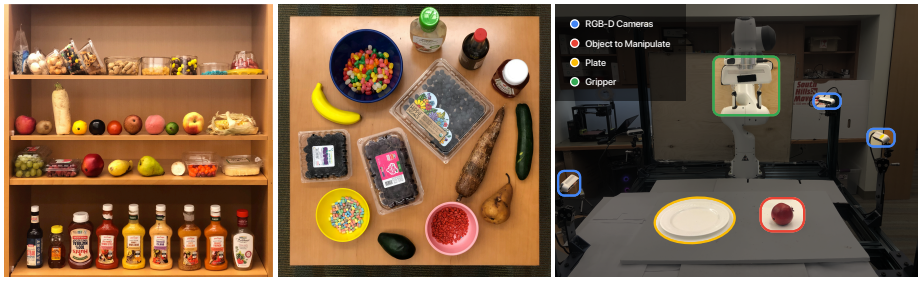

Figure 2.4: Real robot experimental setup. We set 4 cameras around the table that cover different viewpoints of the workspace. 3 out of 4 cameras are shown in the right most image. An extra topdown camera is used, but it is clipped from the image.

Our model achieves a success rate of **88.6%** on the test set. We compare our model with two baselines: 1) an image-to-action model trained with behavior cloning (*Image*), and 2) a hierarchical model that uses a library of behaviors and a selector with 2D representations (*V-BEs with 2D features*). We use the data collected during the interaction label collecting process as the data used to train both baselines. For the *Image* baseline, we are not able to get it to work at all, while we did make it work in simulation where there is more data. It may need more data to learn a general and robust policy in the real world [30]. For the *V-BEs with 2D features* baseline, we get a success rate of **38.0%** which is worse than the proposed model. This again shows the importance of operating the selector in the proposed view-invariant and object-centric 3D feature space. Sample executions and transporting objects in clutter are visualized in Figure 2.5 and included in this video.

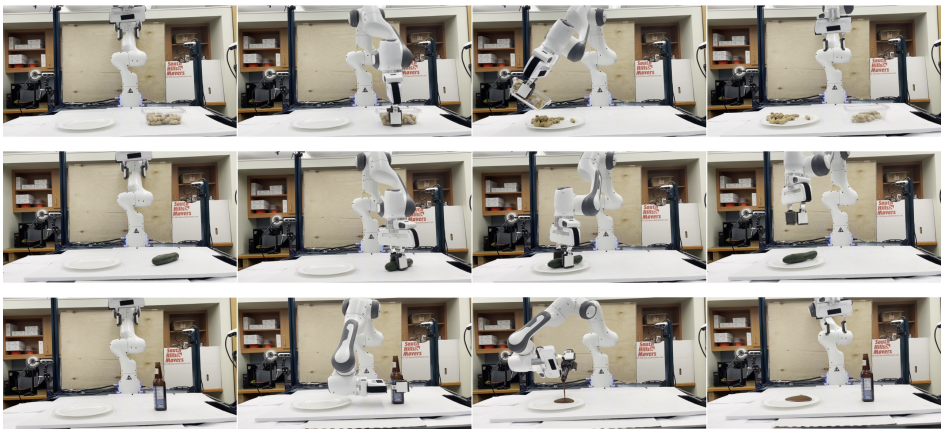

Figure 2.5: Sample real robot executions. Each row shows an execution trial during test time. Our robot can successfully transport rigid, granular, and liquid food ingredients to a target plate.

2.5 Conclusion

We have presented V-BEs, a hierarchical policy architecture where 3D object visual feature representations are used to select from a library of behaviors. The proposed modular architecture supports both low-level behaviors and the selector to be learnt in a data efficient manner. We have shown results on pushing and grasping diverse objects in simulation and in the real world, across diverse viewpoints. Our method outperforms image-to-action monolithic policies of previous works, as well as policies that operate on 3D locations and velocities alone. Since our framework is sample-efficient and simple to run, we can easily deploy complex transporting skills on a real robot arm.

2.6 Limitations

There are still some limitations for our work. First, 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 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 [74] 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.

Chapter 3

MultiTool Transporter Networks for Object Rearrangement from Demonstrations

3.1 Introduction

Humans use tools in their daily lives to accomplish different tasks, e.g., open a wine bottle with a wine-opener, cut wood with a saw, open dough with a rolling pin, etc.. the use of tools revolutionized the human productivity and it is believed to dramatically accelerated human's progress. In contrast to humans, robots either only use their grippers to accomplish various tasks or use tailored tool-like grippers, without the ability to switch tools on- the-fly during a task. In this paper, we explore how to enable robots to take full advantage of multiple tools. We propose a framework that jointly learns how to select the most suitable tool and how to best utilize the selected tool, from limited human demonstrations.

We propose MultiTool Transporter Networks, a neural model that given a visual image of the scene, predicts a tool identity, as well as image-conditioned action parameters

for close-loop policy. Our model is trained directly in the real world from few human demonstrations. The high sample efficiency of our network is thanks to the inductive bias of Transporter Networks [78] which use convolutional networks to predict image-anchored action parameters, i.e., image-anchored waypoints, regarding where the tool should make contact, be pushed to or rotate at, as opposed to workspace-anchored gripper poses at every timestamp. Such dense alignment between localized image features and actions permits to learn from very few examples. Our robot is equipped with multi-tool mounts and a tool cabinet that permit it to automatically, without any human intervention, exchange tools during task execution, as shown in Figure 3.1.

We test our framework on parallel gripper and suction cup picking and placing, sweeping with a brush, and household rearrangement tasks. Our results outperform directly image-to-action mapping baseline and Transporter network method that does not have a visually conditioned tool-changing mechanisms.

In summary, our contributions are:

1. A general end-to-end neural framework for reasoning about tools and tool-conditioned polices from a small number of human demonstrations.
2. A real world robot implementation of a tool-switching robot equipped with low-cost self-assemble suction gripper, and parallel gripper's tool adaptor for different tools including suction gripper, brush, which allows the robot to easily and steadily switch between different tools.
3. An framework for collecting real world human demonstrations for tool usage.

To the best of our knowledge, this is the first work that carries our visually conditioned tool prediction and usage for scene rearrangement on a real robot platform. Our neural model and tool CAD files for 3D printing will be publicly available to allow researchers to easily reproduce our work.

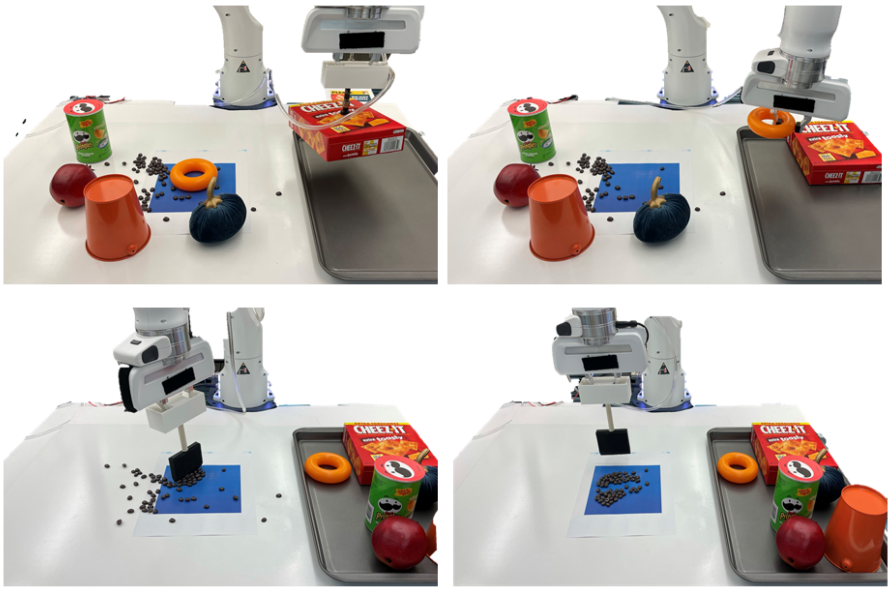

Figure 3.1: **Given an image, MultiTool Transporter Networks predict tools to use and actions for tool-conditioned re-arrangement of the visual scene. Our robot automatically switches tools during task execution using a tool cabinet and appropriate tool mounts.**

3.2 Related work

3.2.1 Tool Manipulation

To enable robots to understand and manipulate tools, seminal works focus on predicting affordance and functional regions [9] for task specific grasping with self-supervision from physical interactions in a simulator [69]. [6] shows that manipulation of tools can be performed through learning and planning. However, these works only focus on how to utilize a single tool's different parts, in other words, how the tool should be grasped. In contrast, our work takes advantage of multiple tools and demonstrates generalization capabilities, through jointly learning a task-aware tool selection model and a manipulation policy from limited amount of human demonstrations.

Moreover, previous works [9, 49] try to learn the synergy between different tool

grasping points and tasks with self-supervised learning method. The learned latent representations could be used to provide structured and condense understanding of tool objects. However, our method does not rely on specific representations and motion primitives, instead, it only uses a general heatmap to predict the actions, and generalizes better to novel objects during test time.

3.2.2 Tool design for robot tool usage

There has been works that aim to advance the design of tools to extend the range of tasks a single robot can achieve. This mainly refers to changing a robot's tools so it adapts to different tasks and scenarios more easily. [16] uses an electro-mechanical actuator to help switch between different tools, providing robust and precise tool-switching performance. However, this is not a visually conditioned framework, rather, the tool selection and switching process has been manually engineered and the tool-switching system comes at a high cost. Some other works [39],[21] try to develop some low cost interfaces module that act as adaptors between different tools and the original robot end-effector. Unfortunately, the tool switching process requires a second hand either by human or another robot arm. And the tools here [21] can not connect with the end-effector tightly, which may cause the robot to fail at many manipulation tasks that require high torque to support the loads. In our work, we share the same idea as [39] and design a series of low cost exchangeable mechanical tool modules for robots. Those tools could be connected to end-effector tightly by grasping two opposite sides of the tool mounts. We also build up a tool cabinet for hanging diverse tools, which allows the robot to switch tools with a single arm.

3.3 Method

Given a set of tools $T_1, ..., T_n$ and the current scene observation I_t, our goal is to select the most suitable tool for scene rearrangement, and sequentially predict the action parameters for using the tool to complete the task. After each interaction, a

new image is observed and a new action is predicted, till the goal is achieved.

We first describe Tansporter Networks [78], a neural model for visually conditioned scene-rearrangement in Section 3.3.1. Then, we present Multitool Transporter networks in Section 3.3.2, our demonstration collection framework in Section 3.3.3 and our hardware multi-tool switching platform in Section 3.3.4.

3.3.1 Background: Transporter Networks

Transporter networks decompose a scene rearrangement task as a sequence of pick and place sub-tasks. The model takes as input one or more RGBD images re-projected from a bird's eye view, and predicts a pick location as a spatial heatmap. Then it crops a small image feature region around the argmax location, and rotates it with $360/k$ angle to form k queries. Next, it independently featurizes the k queries and the overhead image through a ResNet, and take the argmax of the cross-correlations between the overhead image and the k queries to arrive at the final place location.

Transporter networks can model any behaviour that can be effectively represented as two consecutive poses for the robot gripper, such as pushing, sweeping, rearranging ropes, folding, and so on. Inverse kinematics are used to move the gripper to the predicted poses. For more details and visualizations of the basic transporter framework, please see [78].

3.3.2 Multitool Transporter networks

The architecture of our model is illustrated in Figure 3.2. We follow [78] and assume most scene rearrangement tasks can be treated as a sequence of two-stage gripper pose sub-tasks, such as, pick an object up (pick pose) and place it at another location (place pose), or push a group of small particles from the initial position (pick pose) to the goal position (place pose). Each sub-task's action a consists of a tool type T_i, a picking $P_{initial}$ and placing pose P_{final} of the tool. Overall, we have a policy that

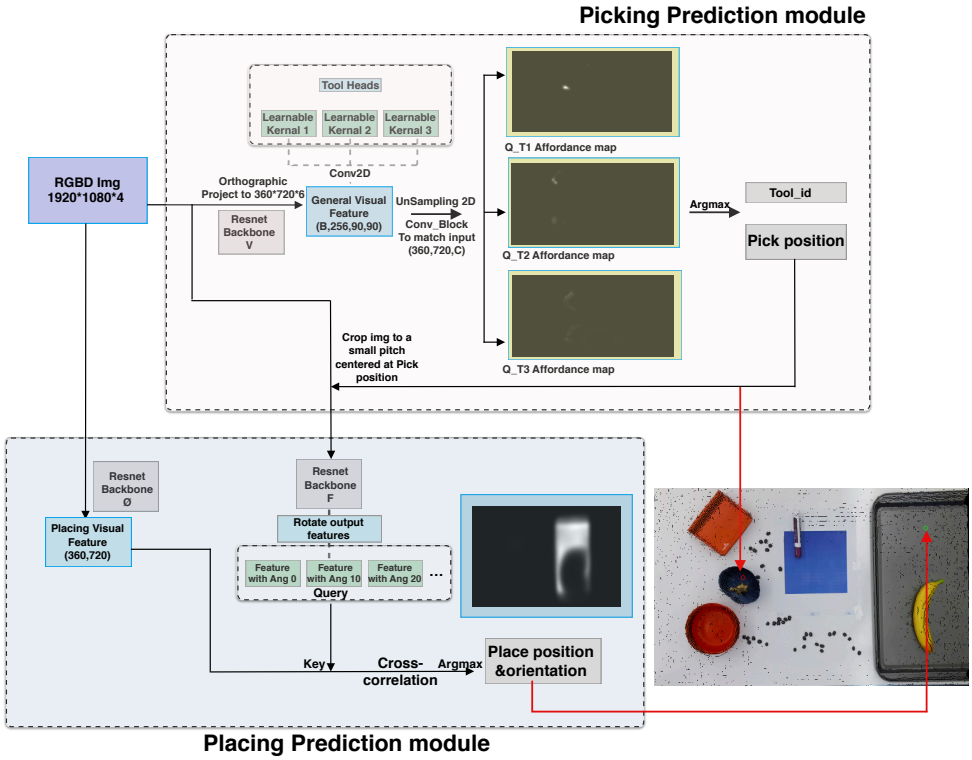

Figure 3.2: **Multitool Transporter network architecture**. Our method is composed of two modules: the affordance-aware picking prediction module and selection-conditioned placing prediction module. The picking prediction module infers which tool should be used and the picking location. The placing prediction module does cross-correlation between the rotated picking prediction module's output's features and placing feature to get the placing locations.

maps from task states to an action:

$$f(I_t) \rightarrow a = (T_i, P_{initial}, P_{final}) \in \mathbb{A} \tag{3.1}$$

$\mathbb{A}$ is the set of possible actions. Time steps can be indicated by a subscript t.

We extend Transporter Networks by adding tool-specific information. Specifically, we first use a ResNet-style [19] backbone to extract general visual features $\mathbf{v}$ from the visual image, which are tool-agnostic and contain entangled visual information relevant for all tasks. Then, for each tool T_i, a learnable tool-specific filter Φ_{T_i} implemented as a 1x1 convolution operation is applied to each spatial location to extract tool-conditioned affordance features $\mathbf{a}_{T_i}$ — information relevant only to that specific tool:

$$\mathbf{a}_{T_i} = \Phi_{T_i} \circledast \mathbf{v} \tag{3.2}$$

From $\mathbf{a}_{T_i}$, an upconvolutional network estimates the heatmap $\mathbf{Q}_{T_i} \in \mathbb{R}^{1 \times H \times W}$, where $\mathbf{Q}_{T_i}(x, y)$ indicates whether the spatial location (x, y) suits the specific tool or not. Finally, we obtain the selected tool and the pick location simply by taking an argmax over the collection of heatmaps:

$$T_i, P_{initial} = \underset{(x,y,T_i)}{\operatorname{argmax}} \mathbf{Q}_{T_i}(x, y) \tag{3.3}$$

Note that inside affordance-aware picking prediction module, all the weights are shared except the tool-specific filters Φ_{T_i}'s. This formulation forces all the tool-specific information to be stored in Φ_{T_i}'s, while the rest of the network remains tool-agnostic and shared across all tools. In contrast to learning separate networks from tool-specific data, this inductive bias on the architecture allows the network to maximize information sharing and learn generalizable representations.

We keep the architecture of our selection-conditioned placing module similar as in [78], to learn the placing locations and rotations, we partially crop the I_t around initial picking prediction to get $f_{initial}$, then rotate it with $360/k$ angle to form k queries. Those queries and I_t are then feed into two different ResNet to get two embeddings. After doing cross correlation between those embeddings, the placing

predictions (location and angle) could be obtained by:

$$P_{final} = \underset{(x,y)}{\operatorname{argmax}} Q_{goal}((x,y)|I_t, T_t, f_{final}) \tag{3.4}$$

We train our framework end to end using supervision from demonstations for the gripper pose parameters as follow:

$$L_{initial} = CE(Label_{init}, Q_{init}((x,y)|o_t, T_t, P_{initial})). \tag{3.5}$$

$$L_{final} = CE(Label_{final}, Q_{final}((x,y)|o_t, T_t, P_{final})). \tag{3.6}$$

where CE denotes softmax cross entropy loss.

Specifically, we do not use the predicted placing angle θ in the sweeping task, since given the initial and final positions, the angle between positive axis and a vector pointing from start to end will indicate the optimal value, which can be defined as a primitive.

3.3.3 Demonstration Collection

The field of reinforcement learning has come to recognize that "seeding" a policy with human demonstrations can speed up learning, or sometimes just make learning possible at all. There are several challenges, however. It is tedious and sometimes costly to collect large numbers of human demonstrations, clean up the typically noisy data, and then map human behavior on to robot capabilities. Robots don't typically have hands as capable as humans, aren't as compliant, and don't have the limb and body degrees of freedom and range of motion of a human, in addition to more limited perceptual capabilities. Our approach is to force humans to act using a robot-like hand whose behavior can be more easily captured. This reduces the "human2robot" capability gap, focuses attention on the relevant human actions (they only involve the robot-like hand), and help us avoid the inclusion of sub-optimal actions using simple filters. We constrain human demonstrators' actions by asking them to use two

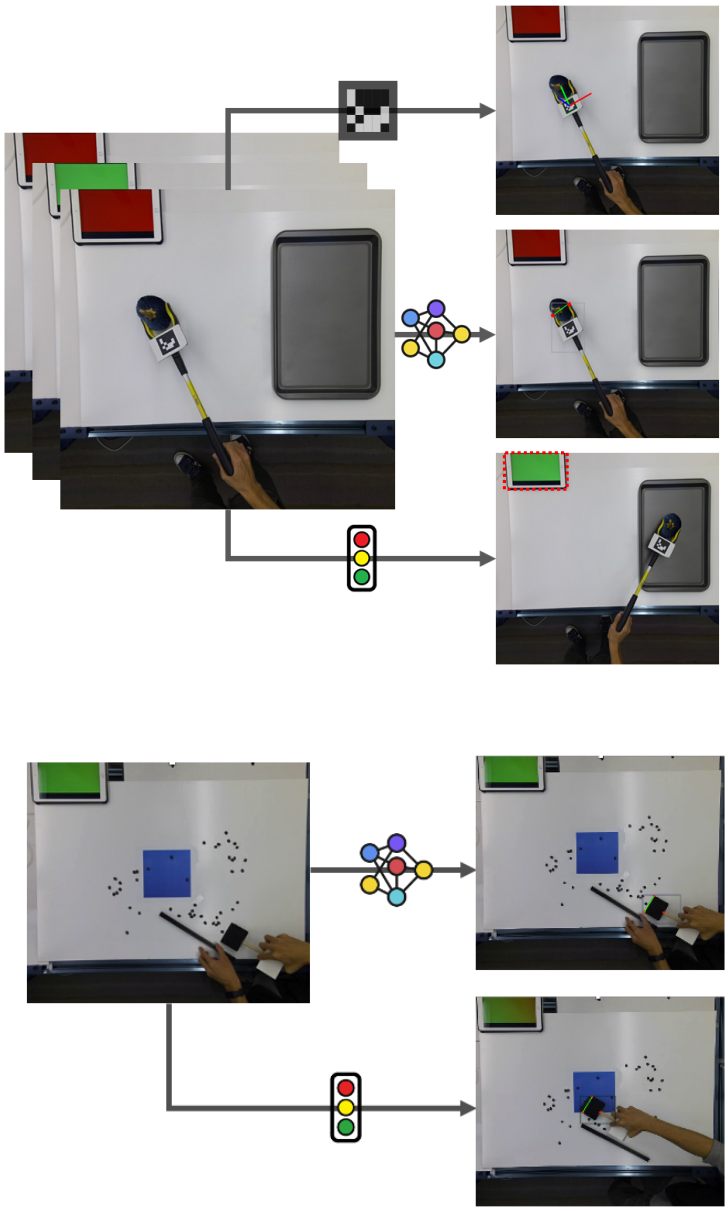

Figure 3.3: **Dataset Collection**. Our demonstration data is collected by recording a grabber hand's actions manipulated by human experts. The attached ARTag is used to get the orientation of the grabber, and a pre-trained detector is used to get the accurate pick and place points in the workspace. Also, a ipad made signal lights is used to label the pick/place key frames we need. We use the pre-trained detector's output keypoints to get orientation and translation for pick and place points

fixed tools as shown in Figure 3.3. The first fixed tool – grabber is used to collect the parallel grasping and suction grasping demos. And the second fixed tool – brush with self designed adaptor is used to collect the sweeping demos.

We put in frame a screen that alternates between two colors (red and green) to indicate the occurrences of keyframes and the keyframe indicator is generated by human demonstrator during the demonstration. More specifically, when the tools reach the pick/place positions, the human demonstrator uses a remote to prompt the keyframe indicator screen to change color. The timesteps at which the indicator screen changes color are extracted as the keyframes. To obtain robust training signals from the extracted keyframes, we train detectors with heavy augmentations for the two tools by labelling a tiny portion of the keyframes (1,000 pictures). We then use the detectors to generate pseudo-labels for all the demonstrations.

For parsing pick and place demos, we use the detector's detection results for gripper as the pseudo-labels of the pick/place locations, and the pose of the Aruco tag (attached on the grabber) as pick/place rotation angles. For sweeping demos, we use the detector's 3 keypoints outputs to get both location and rotation. To get the tool affordance supervision, we design the collection process with pre-defined orders (e.g collect parallel grasping first then suction grasping and last for sweepings).

3.3.4 Hardware Setup

The hardware setup is shown in Fig 3.4. We use a 7-DoF Franka Emika robot arm equipped with a parallel-jaw gripper. To enable the robot to automatically switch between tools, we built a tool cabinet mounted at the back of the workspace, which includes a vacuum-suction end-effector and a brush end-effector. A control PC is connected to the robot to control the primitive behaviors (robot position and trajectory control during pick and place, and pushing) by generating high-frequency control commands [79]. A Raspberry Pi 4B+ controls the suction gripper. Finally, a Linux system PC is connected to both the control PC and the Raspberry Pi for overall control and synchronization. We obtain RGB-D images (resolution 1280x720)

using one low cost Azure Kinect depth camera mounted above the robot, providing a bird's eye view.

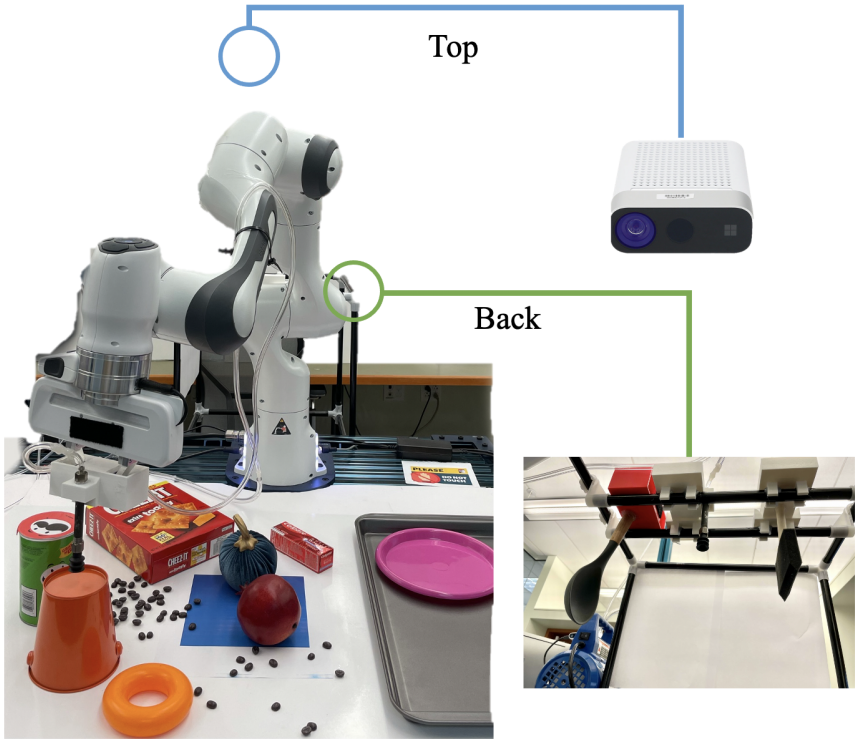

Figure 3.4: **Hardware Setup.** We set up one Azure Kinect camera on top of the robot workspace to get RGB-D streams. And we put a tool shelf at the back of Franka arm for placing different tools. For each tools, we attached one 3D printed tool adaptor to help robot pick up tools easily and steadily.

3.4 Experiments

Our experiments aim to answer the following questions: (1) Does the proposed affordance based picking part correctly select which tools to use? (2) Does the tool-conditioned policy achieve almost the same performance as task-specific learning policy given limited demonstrations? (3) Whether our method could work well in complex environment with generalization ability to random positions, novel objects,

Figure 3.5: **The objects used in our experiments.**

cluttered and unseen scenes? Qualitative results are included in our supplementary video.

3.4.1 Parallel gripper and suction cup P&P (pick-and-place)

On these two tasks, the robot is required to pick up the objects and place them on the plate. 3.3 shows the objects we use in training and testing. For parallel gripper and suction cup P&P, we test our method and all baselines in three settings with increasing difficulties. First, we test single-object P&P with objects in the training set but with randomized object and container locations. We then replace the training objects with test objects, testing single-object P&P while still randomizing object

	50 Gripper Demos			**Mixed Demos**		
	Random Positions	+Novel Objects	+Cluttered Scenes	Random Positions	+Novel Objects	+Cluttered Scenes
Transporter[78]	0.88	0.84	0.85	-	-	-
Ours	**0.93**	**0.91**	**0.90**	**0.95 (0.95)**	**0.90 (0.86)**	**0.85(0.75)**

Table 3.1: SUCCESSFUL RATES OF TASK ON PARALLEL GRIPPER. Mixed demos denote demos contains objects on the left phrase of 3.5. Numbers outside parentheses denote success rate by using all possible tools for this task. Numbers in parentheses denote task success rate when it is restricted to use parallel gripper.

	20 Suction Demos			**Mixed Demos**		
	Random Positions	+Novel Objects	+Cluttered Scenes	Random Positions	+Novel Objects	+Cluttered Scenes
Transporter[78]	**0.95**	0.70	0.65	-	-	-
Ours	0.92	**0.89**	**0.83**	**0.93 (0.93)**	**0.79 (0.68)**	**0.77 (0.63)**

Table 3.2: SUCCESSFUL RATES OF TASK ON SUCTION CUP. Mixed demos denote demos contains objects on the left phrase of 3.5. Numbers outside parentheses denote success rate by using all possible tools for this task. Numbers in parentheses denote task success rate when it is restricted to use suction gripper.

	Random Layouts		Novel Objects		Novel Goal	
	95% completed	100% completed	95% completed	100% completed	95% completed	100% completed
Human Demonstrations	8.5 ± 0.9	10.2 ± 0.9	8.4 ± 0.8	10.3 ± 0.9	8.2 ± 0.9	10.3 ± 0.9
Circular	17.4 ± 0.7	18.5 ± 0.5	17.5 ± 0.7	18.5 ± 0.6	17.2 ± 0.7	18.4 ± 0.7
Horizontal + Vertical	19.2 ± 0.6	19.6 ± 0.5	19.1 ± 0.6	19.4 ± 0.6	19.2 ± 0.5	19.5 ± 0.6
Ours	**11.8 ± 1.3**	**13.7 ± 1.2**	**11.9 ± 1.3**	**13.5 ± 1.3**	**11.7 ± 1.3**	**13.7 ± 1.2**

Table 3.3: TIMES OF PUSHES OF TASK ON BRUSH

	Pick and Place		Sweep		
	Tool Selection	Execution	Tool Selection	95% completed	100% completed
Ours	**0.94**	**0.91**	**0.98**	**11.8 ± 1.2**	**13.7 ± 1.2**

Table 3.4: Performance on Rearrangement Task

and container locations. Finally, we select random subsets of test objects and form cluttered scenes with them. testing multi-object P&P. We use the success rate (the ratio of successful P&P attempts to the total number of attempts) as the metric. For each combination of method and setting, we collect results from 100 test trials.

For these tasks, we compare our method with Transporter [78], since it shows strong performance by learning from limited number of demonstrations. However, it requires to train a new model for each single task. As shown in Table 3.1, our method outperform Transporter on parallel gripper for both 20 demos and 50 demos. This suggests that our architecture retains a strong performance, when a single policy is trained. In Table 3.2, our method is even better than baseline since with the tool/task head added, the model could focus on learning to recognize flatten surface instead of learning to map certain shape/color/contour pattern of the object to actions, which will have better generalization ability to novel objects at novel scenes.

3.4.2 Brush Sweeping

In this task, the robot needs to use the brush to push all the beans to the target area. There are 40 beans on the table. The color can be black, grey or blue. The shape of the target area is square or round, and the area is 225 square centimeters. For this task, we compare the policy learned from our method against 3 baselines:

- Circular sweeping: a heuristic method that sweeps towards the target area, with starting positions initialized along a 3/4 circle large enough to cover all objects;
- Horizontal + vertical sweeping: a heuristic method that first sweeps all objects horizontally toward the middle (from both sides) and sweeps vertically to bring all objects into the target area;

- Human demonstrations: human testers were asked to sweep greedily only in straight lines, which serves as an upper bound of the performance. The demonstrations we collected are also used for training.

All models are trained with 750 human-demonstrated pushes. We use a simple OpenCV HSV filter to detect the status of the current trial and measure success. This allows us to measure success at different success thresholds. We report the number of pushes performed (the lower the better) before 95% and 100% of the beans are pushed into the target area. The maximum number of attempts is set to 30. Therefore if the robot can't sweep all the beans to the target area within 30 attempts, the number of pushes is 30.

We test our method and all baselines in 3 settings. First, we test the model with the same beans during training, but in randomized layouts. Second, we test the model on new beans (grey and blue), also in random layouts. Finally, we change the shape of target area from square to round, which is also unseen from the demonstrations. For each method under each setting, we test the model with 20 random scenes. The performance of all methods are shown in Table 3.3.

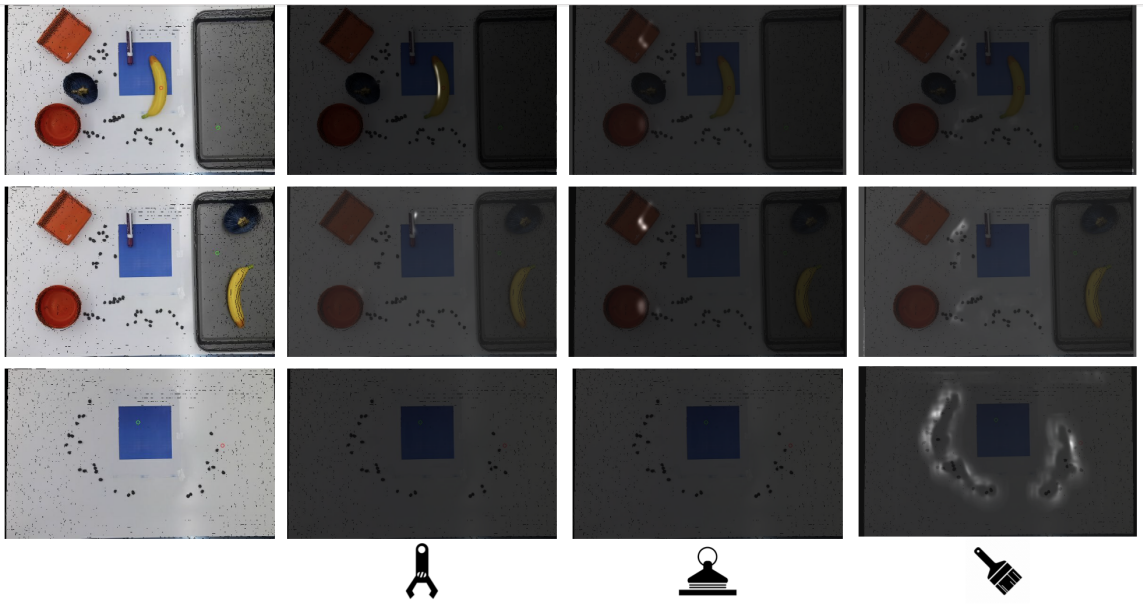

Figure 3.6: **Visualizations of heatmaps on different tools**

3.4.3 Scene Rearrangement

Scene rearrangement task requires the robot to move different objects with a suitable tool. In our settings, the robot needs to use one of the three tools: parallel gripper, suction cup and brush to rearrange the objects on the table. During training, we use 41 demos from suction, 56 demos from parallel grasping, 87 demos for mixing parallel grasping and suction grasping, and 246 sweeping demos. To be more clear, each demo means single step action a_t as we defined 3.3.2.

We test our model by asking robot to transfer all the objects to the tray, and push all the beans to the blue square area. As we could see from table 3.4, our method could not only select correct tool with a successful rate greater than 90%, but could jointly learn a good representation and policy as shown in Figure 3.6. The visualizations for initial and final heatmaps show our model could discrete tool clearly and could generalize to novel object and random unseen scenes.

3.5 Conclusion

We explored neural models and hardware platforms for robots to learn to select and use tools for visual scene rearrangement. We empirically showed the proposed model and robot platform learn from a limited number of human demonstrations, and can generalize to novel object arrangements, novel objects, novel camera views, and even cluttered scenes. We extend transporter networks with the ability of tool selection and switching during task completion. We believe our work will inspire researchers towards automated visually-conditioned tool development, selection and policies, which would support generalizable robot platforms that can tackle diverse tasks in low cost setups.

Chapter 4

PASTA:Planning with Spatial-Temporal Abstraction from Point Clouds for Deformable Object Manipulation

4.1 Introduction

Consider a typical cooking task of making dumplings from dough. People plan over which piece of dough to manipulate and which tool to use in sequence, incorporating both spatial and temporal abstractions. A spatial abstraction reasons about objects, parts, and their relations to each other, such as reasoning about pieces of dough instead of reasoning about individual dough atoms; such a spatial abstraction enables efficient planning and compositional generalization. On the other hand, a temporal abstraction incorporates abstract actions represented as a set of skills such as deciding which tool to use at different task stages, instead of making plans at low-level actions such as joint torques. Temporal abstractions allow planning at the skill level, enabling more efficient optimization for solving long-horizon tasks. An autonomous robot

that operates in unstructured environments should be able to reason about world dynamics using high-level spatial and temporal abstractions instead of reasoning only over the atoms, infinitesimally small timesteps, and low-level robot actions.

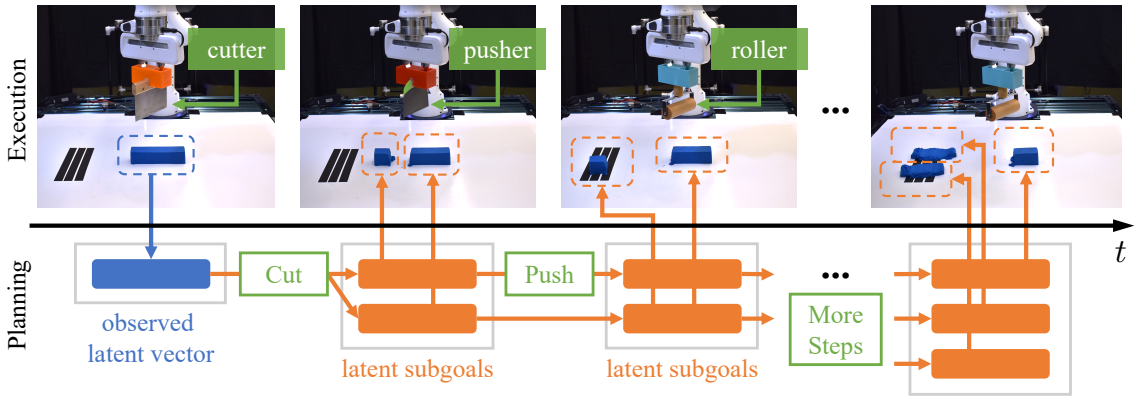

Figure 4.1: **Long-horizon dough manipulation with diverse tools.** Our framework is able to solve long-horizon, multi-tool, deformable object manipulation tasks that the agent has not seen during training. The illustrated task here is to cut a piece of dough into two with a cutter, transport the pieces to the spreading area on the left (with a high-friction surface) using a pusher, and then flatten both pieces with a roller.

4.2 Method

Given a point cloud of the dough P^{obs} and a goal point cloud P^{goal}, our objective is to execute a sequence of actions $a_1, ..., a_T$ that minimizes the distance between the final observed point cloud and the goal $D(P_T^{obs}, P^{goal})$ where P_i^{obs} is the segmented observation point cloud at time i. We aim to solve long-horizon tasks that require chaining multiple skills in novel scenes with more objects than training. To do so, we present a general framework that incorporates spatial and temporal abstractions for learning and planning with skills from high-dimensional observations, as summarized in Figure 4.2. We use point clouds as input to all our modules to enable easier transfer from simulation to the real world and to enable robustness to changes in viewpoint.

We assume access to an offline dataset of demonstration trajectories $\mathcal{D}_{demo}$, where

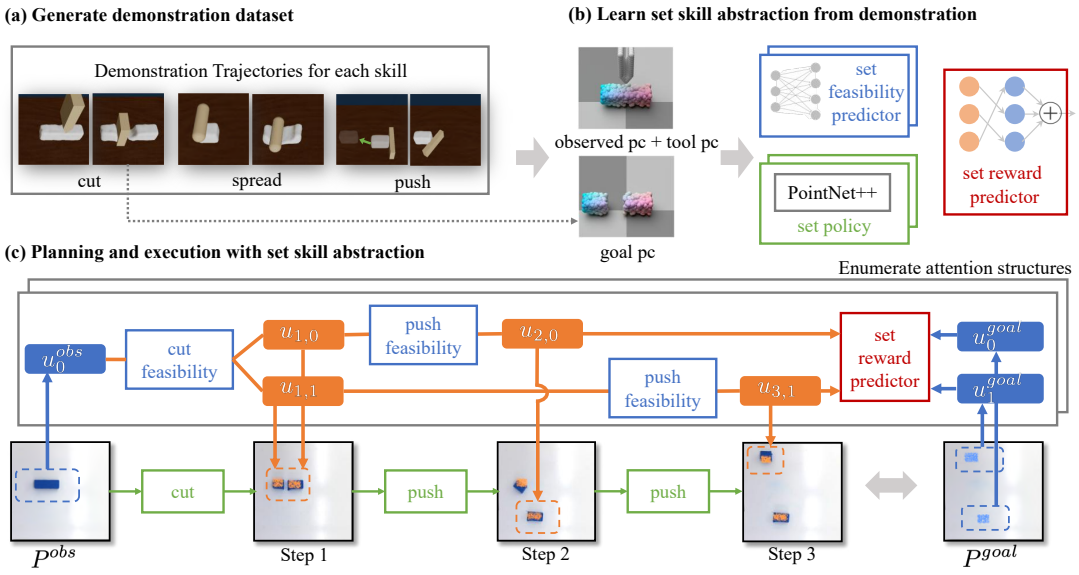

Figure 4.2: **Overview of our proposed framework PASTA.** (a) We first generate demonstration trajectories for each skill in a differentiable simulator using different tools. (b) We then sample point clouds (pc) from the demonstration trajectories to train our set skill abstraction modules. (c) We map point clouds into a latent set representation and plan over tool-use skills to perform long-horizon deformable object manipulation tasks. P^{obs}, P^{goal} are the observation and target pc; $u_{i,j}$ denotes component j at step i. The example shows our method performs the CutRearrange task, which requires cutting the dough into two pieces with a knife and transporting each piece to its target location.

each trajectory demonstrates one of the K skills using one tool. We can learn skill policies by imitation learning from these demonstration. To chain these skills to solve long-horizon tasks, we train a set of skill abstraction modules, which can be used for efficient planning in a latent space. Below, we first describe how we generate this latent space and use an object-centric representation to generalize our planner to scenes with more objects.

4.2.1 Planning with Set Representation

Given an observation and a goal point cloud P^{obs}, P^{goal}, we can plan subgoals and the sequence of skills using our trained abstraction modules, such that we can use our skills to follow each subgoal sequentially to reach a given target. To do so, we need to optimize for the sequence of skills to apply, the attention for each skill (i.e. find $\hat{U}^o \subseteq U^o$), as well as the latent subgoals for each skill (i.e. the exact value for each latent vector in $\hat{U}^o$). For our most simple approach, we run a three-level nested optimization: In the top-level, we exhaustively search over the combinations of skills to apply at each step, i.e. $k_1 \ldots, k_H$, where k_h is the index of the skill to apply at the high-level step h. We only keep the sequences that end with the same set cardinality as the goal by ensuring that $\sum_{h=1}^{H} M_{k_h} - N_{k_h} = N_g - N_o$, where M_{k_h} and N_{k_h} are the number of observation and goal components for the skill applied at step h and N_o and N_g are the number of components in the observed and target point clouds.

In the second-level optimization, we search over different attention structures. We formally define the attention structure at step h to be I^h, which consists of a list of indices, each of length N_{k_h}, such that I^h selects a subset from U^{h-1} to be the input to the feasibility predictor, i.e. $\hat{U}^{h-1} = U^{h-1}_{I^h} \subseteq U^{h-1}$. Assume that we have N_h components before applying skill k_h, i.e. $|U^{h-1}| = N_h$ and skill k_h takes K_h components as its observation. We can search over all $C^{K_h}_{N_h}$ combinations of attention structures. For components not considered by the skill, its latent vector will remain the same at step h. The combination of each skill attention yields an attention structure $\mathbf{I}$ for the whole plan, as illustrated in Fig. 4.2(c). For this level of optimization, we use a sampling-based procedure to avoid an exhaustive search over topologically

equivalent attention structures. See appendix for how we do this efficiently.

In the low-level optimization, for each attention structure $\mathbf{I}$, we follow the optimization in DiffSkill [34]. We first sample multiple initializations for the set of latent subgoals $\mathbf{U}$, where each latent vector in the set is initialized from our generative model. We can then perform gradient descent to further optimize the latent subgoals on the following objective:

$$\arg\min_{\mathbf{k},\mathbf{I},\mathbf{U}} C(\mathbf{k},\mathbf{I},\mathbf{U}) = \prod_{h=1}^{H} f_{k_h}(\hat{U}^{h-1},\hat{U}^{h}) \exp(-R(U^H, U^g)), \tag{4.1}$$

where $\mathbf{k}$ is the skill sequence, $\mathbf{I}$ is the attention structure of the plan, $\mathbf{U}$ is the set of all latent subgoals, $\hat{U}^h = \{u_i^h\}_{i=1\ldots M_{k_h}}$ are the latent subgoals at step h, $\hat{U}^0 = \hat{U}^o \subseteq U^o$ is the attended observed set, and U^g is the goal set. Finally, we can use our policy to execute our plan by following each subgoal. More details could be checked in our CORL-2022 Submission website.

4.3 Experiments

Figure 4.3 shows our real world setup. We use a Franka robot with a top-down Azure Kinect camera capturing the RGB-D observation of the workspace. The robot is equipped with a tool station that allows an automatic change of tools. For real world "dough", we use Kinect Sand as a proxy because of its stable physical property. We transfer the feasibility predictor and reward predictor of PASTA directly from simulation and define heuristic controllers for the skills. For evaluation, we first generate a desired target point cloud and then reset the dough to its initial shape and record its point cloud. Next, given the current and the target point cloud, we use the planner to generate a sequence of skills and subgoals and then execute the plan with our controller. Finally, we record the achieved point cloud and report the normalized improvement EMD. We compare with the Flat3D method and also report performance of human on these tasks.

We evaluate on three of the simulation tasks: CutRearrange, CRS, and CRS-Twice.

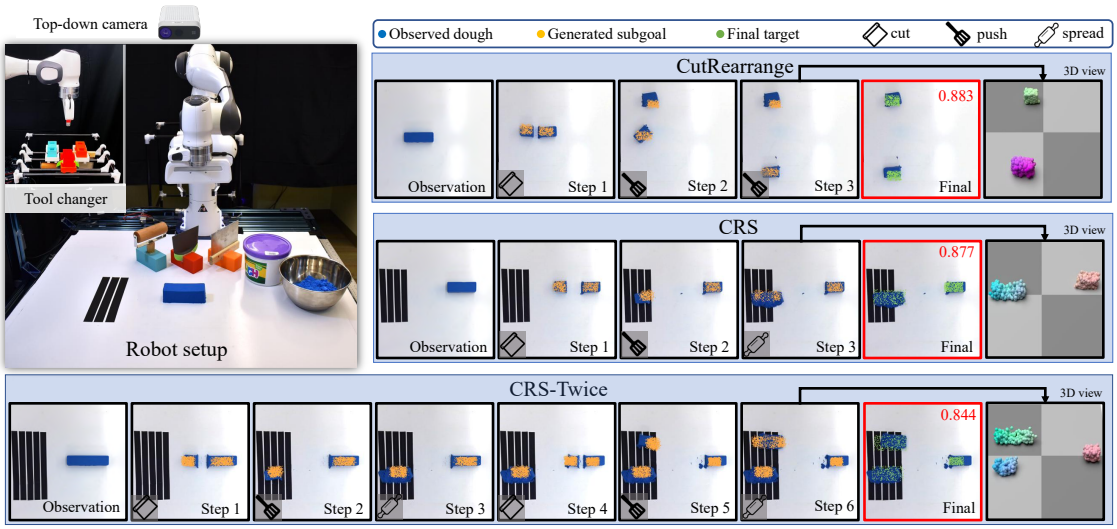

Figure 4.3: **Real world setup and execution with planned subgoals**. Our workspace consists of a Franka robot, a top-down camera, and a novel tool changer behind the robot that allows the robot to automatically switch tools. For each task, we show frames after executing a skill overlaid with the decoded point cloud subgoal; we report the final performance in red and overlay the ground truth target in green in the final frame. Additionally, we include a 3D view of the last generated subgoal to show the shape variations.

For each task we evaluate on the same four initial and target shapes for each method and report the performance in Table 4.1. Figure 4.3 shows the key frames from the execution of PASTA. We overlay the planned subgoals as well as the final goal for qualitative comparison. PASTA performs on-par with human in the real world, highlighting the robustness of our planner and the advantage of using 3D representation for sim2real transfer.

MethodTask (Horizon)	CutRearrange (3)	CRS (3)	CRS-Twice (6)
Flat 3D	0.351 ± 0.478	0.007 ± 0.429	-
PASTA (Ours)	**0.836 ± 0.029**	**0.854 ± 0.016**	**0.795 ± 0.035**
Human	0.910 ± 0.014	0.863 ± 0.018	0.895 ± 0.013

Table 4.1: Normalized improvement on real world tasks. Each entry shows the mean and std of the performance over 4 runs. Flat 3D does not produce any meaningful plan for CRS, so we do not evaluate it on CRS-Twice.

4.4 Conclusions and Limitations

In this work, we propose a planning framework named PASTA that incoporates both spatial and temporal abstraction by planning with a 3D latent set representation with attention structure. We demonstrate a manipulation system in the real world that uses PASTA to plan with multiple tool-use skills to solve the challenging deformable object manipulation tasks, and we show that it significantly outperforms a flat 3D representation, especially when generalizing to more complex tasks.

Limitations: First, we rely on an unsupervised clustering method for entity decomposition and a point cloud VAE for mapping an observation to our latent set representation. This design is specific to the tasks we show and may not generalize to other tasks without retraining or parameter tuning. We hope that our framework can be incorporated in the future with self-supervised methods for learning the spatial abstraction. Second, manipulating deformable objects like dough is very challenging with a significant sim2real gap. This paper provides a starting point for planning with multiple tools towards this challenge; we hope our work can inspire more works in this exciting area.

Chapter 5

Conclusions

In this thesis, we first propose a visually-grounded library of behaviors approach for learning to manipulate diverse objects across varying initial and goal configurations and camera placements. We test our framework on pushing and grasping diverse objects in simulation as well as transporting rigid, granular, and liquid food ingredients in a real robot setup. Our model outperforms image-to-action mappings that do not separate static and dynamic object properties.

We then propose an end-to-end learning framework that jointly learns to choose different tools and deploy tool-conditioned policies with a limited amount of human demonstrations directly on a real robot platform. We empirically showed the proposed model and robot platform learn from a limited number of human demonstrations, and can generalize to novel object arrangements, novel objects, novel camera views, and even cluttered scenes. Finally, we show a long-horizon planning framework that could utilize our multiple tool setup to manipulate elastoplastic objects in the real world successfully, such as a dough.

Appendix A

V-BEs supplementary materials

A.1 Additional Results

A.1.1 More Baselines on Grasping

We add evaluation and discussion on state-of-the-art grasping methods, 6D-GraspNet [40] and DexNet [35]. 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 [35]: 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 [35], fine-tune the model with the same amount of interactive labels as our model, and evaluate it in our test environments.

6DoF Grasp-Net [40]: 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 [40], 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. [45], 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. [45].

We compare the final task performances of our method and the aforementioned baselines in Table A.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 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 [35]	6Dof-GraspNet [40]
Grasping	**0.78**	0.67	0.72	0.36

Table A.1: Success rates on grasping unseen objects.

A.1.2 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.1.3 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.1.4 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 A.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.

A.2 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 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 [74] will be a promising direction to investigate. Another limitation of our current framework

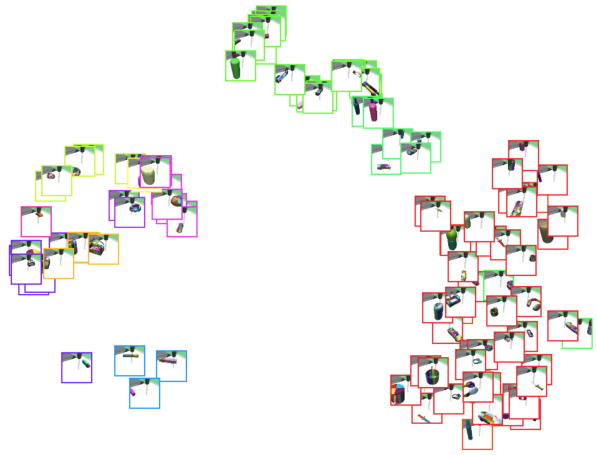

Figure A.1: T-SNE visualization of feature representations of test-time objects and their initial configurations. Each image shows one sample in the test set where a test-time object with its initial position and pose is displayed. The border color of each image denotes the behavior assignment of the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

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.

A.3 Additional related work

A.3.1 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 [31, 56], 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.

A.3.2 Learning to grasp from 2D images

By scaling up real-robot interactions for supervised image-to-action policies, existing work [24, 56] have shown expressive results for grasping diverse objects [24] and can operate under diverse camera viewpoints [56]. However, [24] only works on images in a top-down view from a fixed over-the-shoulder camera. [56] 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.

A.3.3 Multi-task and skill learning

Existing works in multi-tasks or skill learning from image inputs [28, 77**?**] have shown impressive results in learning behaviors from demonstrations or interactions, but they often assume image inputs from a fixed camera view. [56] 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.

A.3.4 Behavior selection with a classifier

Many strong robotics works can be viewed as classifiers over a set of skills for flying drones [14], grasping [36, 44, 76], transporting [77] and even marble mazes for rolling balls [4]. 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 [73], and some works use a learned generator to generate candidate behaviors [35, 40] for the selector to choose from. In particular, Transporter [77] 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.

A.3.5 Task and Motion Planning (TAMP) for robot manipulation

Task and Motion Planning (TAMP), as described here [23], formulate the manipulation problem as the combination of high level symbolic, discrete reasoning and lower level continues motion trajectory planning. These systems typically assume known object states and known effects for the action operators [11, 15, 38, 66, 67]. However, it is difficult to estimate states or dynamics for unknown objects, novel camera viewpoints, or from partial observations. While recent works have made progress in learning certain components of the system, such as the logical states [75] from high image observations or learning action models [32, 70, 72] from interactions, those methods still have weak generalization ability to diverse objects across varying initial and goal configurations and camera placements. In contrast, our work learns a object-centric affordance aware 3D representation and can easily generalize to more complex scenes.

A.4 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 $\text{clip}(x_1 x_2, 0.707, 1.414)$ 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.

A.5 Implementation Details

A.5.1 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 A.2.

A.5.2 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.

A.5.3 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.

A.5.4 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 [81]. The outputs of the CNN encoding layers are passed through a spatial softmax layer [12], flattened into a vector, and then concatenated to the abstract 3D state (note that compared to [81], 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 deg, -45 deg, 0 deg, 45 deg, and 90 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 A.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.

A.5.5 Contextual 3D Baseline

In the Contextual 3D baseline, we leverage the DAGGER [53] 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 [68]. 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.

A.5.6 V-BEs (Our Method)

As for the architecture of the affordance-aware behavior selector, the 2D-to-3D image encoding using GRNNs [68] follows the exact neural architecture as in [68], which takes as input RGB-D images and outputs 3D feature map $\mathbf{M}_t$ of size $64 \times 64 \times 64 \times 32$. After obtaining the feature maps from the input RGB-D image, we compute the cosine-similarity between the object's feature representation and each behavior retrieval key κ_i for every feature dimension of the feature map. This will give us a vector of 32 similarity values for each (object, behavior) pair. We then feed this vector into a fully-connected layer with input size 32 and output size 1 to predict the logits used to compute the loss for training. While computing the loss during training, we also multiply a constant weight of 0.5 on all negative samples to offset the bias caused by the smaller number of positive samples. At test time, we directly use the logits that correspond to whether the (object, behavior) pair leads to execution success to select the best behavior for a given object. In other word, we compute the logit for every (object, behavior) pair and select the behavior with the highest predicted value on an object as selector output.

A.6 Real Robot Experiment Setup

A.6.1 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.

A.6.2 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.

A.6.3 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.

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 A.2: Controllers used in the proposed model.

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

Bibliography

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016. 2.2

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017. 2.1, 2.2, 2.3.2, 1, 2.4.1, **??**

[3] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 2.1, 2.1, 2.2

[4] Darrin C. Bentivegna, Gordon Cheng, and Christopher G. Atkeson. Learning from observation and from practice using behavioral primitives. In Paolo Dario and Raja 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. A.3.4

[5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2.4.1

[6] Hao Dang and Peter K. Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1311–1317, 2012. doi: 10.1109/IROS.2012.6385563. 3.2.1

[7] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning, 2019. 2.1, 2.2

[8] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, pages 767–782. PMLR, 2018. 2.2

[9] Kuan Fang, Yuke Zhu, Animesh Garg, Andrey Kurenkov, Viraj Mehta, Li Fei-Fei, and Silvio Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *The International Journal of Robotics Research*, 39 (2-3):202–216, 2020. 3.2.1

[10] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim. Optimization based controller design and implementation for the atlas robot in the darpa robotics challenge finals. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1028–1035, 2015. 2.1

[11] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971. A.3.5

[12] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016. A.5.4

[13] Peter Florence, Lucas Manuelli, and Russ Tedrake. Self-supervised correspondence in visuomotor policy learning, 2019. 2.2

[14] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017. 2.1, A.3.4

[15] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020. A.3.5

[16] David Gyimothy and Andras Toth. Experimental evaluation of a novel automatic service robot tool changer. In *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1046–1051, 2011. doi: 10.1109/AIM.2011.6027122. 3.2.2

[17] Adam W Harley, Fangyu Li, Shrinidhi K Lakshmikanth, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Embodied view-contrastive 3d feature learning. *arXiv preprint arXiv:1906.03764*, 2019. **??**, 2.4.3

[18] Adam W Harley, Shrinidhi K Lakshmikanth, Fangyu Li, Xian Zhou, Hsiao-Yu Fish Tung, and Katerina Fragkiadaki. Learning from unlabelled videos using contrastive predictive neural 3D mapping. *ICLR*, 2020. 2.3.1

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 3.3.2

[20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870. 2.3.1

[21] Zhengtao Hu, Weiwei Wan, and Kensuke Harada. Designing a mechanical tool for robots with two-finger parallel grippers. *IEEE Robotics and Automation Letters*, 4(3):2981–2988, 2019. 3.2.2

[22] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *CVPR*, pages 12627–12637, 2019. 2.2

[23] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. A.3.5

[24] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL http://arxiv.org/abs/1806.10293. A.3.2

[25] Ozsel Kilinc, Yang Hu, and Giovanni Montana. Reinforcement learning for robotic manipulation using simulated locomotion demonstrations, 2019. 2.2

[26] Oliver Kroemer and Gaurav S. Sukhatme. Learning relevant features for manipulation skills using meta-level priors. *CoRR*, abs/1605.04439, 2016. URL http://arxiv.org/abs/1605.04439. 2.2

[27] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016. 2.2

[28] Youngwoon Lee, Jingyun Yang, and Joseph 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. A.3.3

[29] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 2.1, 2.1, 2.2, 2.4.1

[30] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale

data collection. *CoRR*, abs/1603.02199, 2016. URL http://arxiv.org/abs/1603.02199. 2.4.4

[31] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre 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. A.3.1

[32] Jacky Liang, Mohit Sharma, Alex LaGrassa, Shivam Vats, Saumya Saxena, and Oliver Kroemer. Search-based task planning with learned skill effect models for lifelong robotic manipulation. In *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022. A.3.5

[33] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016. 2.1, 2.4.1

[34] Xingyu Lin, Zhiao Huang, Yunzhu Li, Joshua B Tenenbaum, David Held, and Chuang Gan. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *ICLR*, 2022. 4.2.1

[35] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio, and Ken 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. 2.2, A.1.1, **??**, A.3.4

[36] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019. A.3.4

[37] Lucas Manuelli, Yunzhu Li, Pete Florence, and Russ Tedrake. Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning, 2020. 2.1

[38] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. *Technical Report CVC TR-98-003/DCS TR-1165*, 1998. A.3.5

[39] Stephen McKinley, Animesh Garg, Siddarth Sen, David V. Gealy, Jonathan P. McKinley, Yiming Jen, Menglong Guo, Doug Boyd, and Ken Goldberg. An interchangeable surgical instrument system with application to supervised automation of multilateral tumor resection. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 821–826, 2016. doi: 10.1109/COASE.2016.7743487. 3.2.2

[40] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational

grasp generation for object manipulation. *CoRR*, abs/1905.10520, 2019. URL http://arxiv.org/abs/1905.10520. 2.2, A.1.1, ??, A.3.4

[41] Gerhard Neumann, Christian Daniel, Alexandros Paraschos, Andras Kupcsik, and Jan Peters. Learning modular policies for robotics. *Frontiers in computational neuroscience*, 8:62, 2014. 2.2

[42] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019. 2.2

[43] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 2.1

[44] Lerrel Pinto and Abhinav 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. 2.2, A.3.4

[45] Lerrel Pinto and Abhinav 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. A.1.1

[46] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017. 2.2

[47] Yuzhe Qin, Rui Chen, Hao Zhu, Meng Song, Jing Xu, and Hao Su. S4g: Amodal single-view single-shot se(3) grasp detection in cluttered scenes, 2019. 2.2

[48] Zengyi Qin, Kuan Fang, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*, 2019. 2.1, 2.3

[49] Zengyi Qin, Kuan Fang, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Keto: Learning keypoint representations for tool manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7278–7285. IEEE, 2020. 3.2.1

[50] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. 2.2, 2.3.2, 1, ??

[51] N. Ratliff, J. A. Bagnell, and S. S. Srinivasa. Imitation learning for locomotion and manipulation. In *2007 7th IEEE-RAS International Conference on Humanoid*

Robots, pages 392–397, 2007. 2.2

[52] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing solving sparse reward tasks from scratch. In *International Conference on Machine Learning*, pages 4344–4353. PMLR, 2018. 2.2

[53] 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, 2011. A.5.5

[54] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. In *In AISTATS*. Citeseer, 2011. 2.4.1

[55] Elmar Rückert and Andrea d'Avella. Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems. *Frontiers in computational neuroscience*, 2013. 2.2

[56] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real view invariant visual servoing by recurrent control. *CoRR*, abs/1712.07642, 2017. URL http://arxiv.org/abs/1712.07642. A.3.1, A.3.2, A.3.3

[57] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006. 2.2

[58] Maximilian Sieb, Xian Zhou, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation. In *Conference on Robot Learning (CoRL)*, 2019. 2.2

[59] Bruno Da Silva, George Konidaris, and Andrew Barto. Learning parameterized skills, 2012. 2.2

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2.4.3

[61] Jost Tobias Springenberg, Karol Hausman, Martin Riedmiller, Nicolas Heess, and Ziyu Wang. Learning an embedding space for transferable robot skills. 2018. 2.2

[62] Robin Strudel, Alexander Pashevich, Igor Kalevatykh, Ivan Laptev, Josef Sivic, and Cordelia Schmid. Learning to combine primitive skills: A step towards versatile robotic manipulation, 2019. 2.2

[63] Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *Robotics, IEEE Transactions on*, 28:1360–1370, 12 2012. doi: 10.1109/TRO.2012.2210294. 2.2

[64] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 2.4

[65] Tarik Tosun, Eric Mitchell, Ben Eisner, Jinwook Huh, Bhoram Lee, Daewon Lee, Volkan Isler, H Sebastian Seung, and Daniel Lee. Pixels to plans: Learning non-prehensile manipulation by imitating a planner. *arXiv preprint arXiv:1904.03260*, 2019. 2.2

[66] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. A.3.5

[67] Marc A Toussaint, Kelsey Rebecca Allen, Kevin A Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. *Robotics: Science and Systems Foundation*, 2018. A.3.5

[68] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2.1, 2.3.1, 3, **??**, 2.4.3, A.5.5, A.5.6

[69] Dylan Turpin, Liquan Wang, Stavros Tsogkas, Sven Dickinson, and Animesh Garg. Gift: Generalizable interaction-aware functional tool affordances without labels. *arXiv preprint arXiv:2106.14973*, 2021. 3.2.1

[70] Emre Ugur and Justus Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633, 2015. doi: 10.1109/ICRA.2015.7139553. A.3.5

[71] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015. 2.2

[72] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021. A.3.5

[73] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi 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/`. A.3.4

[74] Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. Visual imitation made easy, 2020. 2.6, A.2

[75] Wentao Yuan, Chris Paxton, Karthik Desingh, and Dieter Fox. Sornet: Spatial

object-centric representations for sequential manipulation. In *Conference on Robot Learning*, pages 148–157. PMLR, 2022. A.3.5

[76] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019. A.3.4

[77] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *CoRR*, abs/2010.14406, 2020. URL https://arxiv.org/abs/2010.14406. A.3.3, A.3.4

[78] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020. 3.1, 3.3, 3.3.1, 3.3.2, 3.3.2, ??, ??, 3.4.1

[79] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *CoRR*, abs/2011.02398, 2020. URL https://arxiv.org/abs/2011.02398. 3.3.4

[80] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020. 2.4.4

[81] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *ICRA*, pages 1–8. IEEE, 2018. 2, A.5.4

[82] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *ICRA*, pages 1–8, 2018. 2.1, 2.2