Generalizing Object-Centric Task-Axes Controllers using Keypoints

Mohit Sharma[1] and Oliver Kroemer[1]

***Abstract*— To perform manipulation tasks in the real world, robots need to operate on objects with various shapes, sizes and without access to geometric models. To achieve this it is often infeasible to train monolithic neural network policies across such large variations in object properties. Towards this generalization challenge, we propose to learn modular task policies which compose object-centric task-axes controllers. These task-axes controllers are parameterized by properties associated with underlying objects in the scene. We infer these controller parameters directly from visual input using multi-view dense correspondence learning. Our overall approach provides a simple and yet powerful framework for learning manipulation tasks. We empirically evaluate our approach on 3 different manipulation tasks and show its ability to generalize to large variance in object size, shape and geometry.**

I. INTRODUCTION

Manipulation tasks in the real world involve objects of varying, and often unknown, shapes and sizes. Learning to perform manipulation tasks across a wide range of objects, without access to their underlying geometric models, is a challenging problem. Recent work has shown how simple keypoint representations can be used to obviate the need of known geometric models [1], [2]. These keypoint representations, which are learned purely from visual data, are easy to acquire and provide accurate and robust intra-category generalization capabilities. Such keypoint representations have been utilized to formulate optimization problems, whose solutions results in a one-step $SE(3)$ action that is performed by the robot [1], [2]. Alternately, they have also been used for state estimation [2], [3], wherein the keypoints are often directly used as inputs to monolithic neural networks that output the action to be performed at each step.

Instead of using monolithic policies for task learning, recent work [4] has proposed a more modular approach by defining task-axis controllers for each possible subtask. These controllers are attached to different objects (or their parts) in the scene, such as the normal of a table or middle of the door handle. This object-centric nature of controllers provides important invariances to certain object properties such as a controller that reaches close to an object will be invariant of its position. More importantly, these controllers are reusable across multiple different tasks and provide a structured action space for the robot to explore and act. This approach results in improved sample complexity and

[1]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, {mohits1, okroemer}@cs.cmu.edu

This work was supported by NSF Award No. CMMI-1925130, IIS-1956163, Office of Naval Research Grant No. N00014-18-1-2775, ARL grant W911NF-18-2-0218 as part of the A2I2 program, and Nvidia NVAIL.

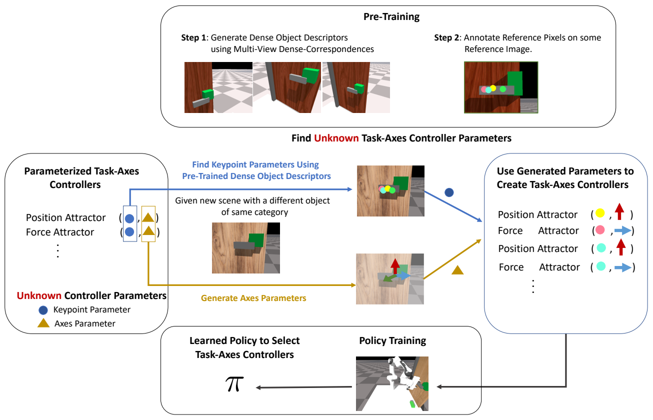
Pre-Training
Step 1: Generate Dense Object Descriptors using Multi-View Dense-Correspondences
Step 2: Annotate Reference Pixels on some Reference Image.
Find Unknown Task-Axes Controller Parameters
Parameterized Task-Axes Controllers
Position Attractor (●,▲)
Force Attractor (●,▲)
Unknown Controller Parameters
● Keypoint Parameter
▲ Axes Parameter
Find Keypoint Parameters Using Pre-Trained Dense Object Descriptors
Given new scene with a different object of same category
Generate Axes Parameters
Use Generated Parameters to Create Task-Axes Controllers
Position Attractor
Force Attractor
Position Attractor
Force Attractor
Learned Policy to Select Task-Axes Controllers
Policy Training
π

Fig. 1: Overview of our proposed approach. We extend task-axes controllers to operate on visual input and use them to present a simple and generalizable approach for learning manipulation tasks.

much better generalization for manipulation tasks [4], and is referred to as *object-centric task-axes controllers*.

One limitation of [4] is that they do not infer the controller parameters directly from the observed data. Instead, they use heuristics to define the set of possible controller parameterizations for each task. These controller parameters include both the *position targets*, *i.e.*, 3D positions for relevant objects or other semantically meaningful points on the object such as edges or corners, as well as the *relevant axes*, *i.e.*, the axes along which the controller acts.

In this work, we extend [4] to allow it to infer the controller parameters directly from visual input. Thus we avoid the use of fixed heuristics to find position-target parameters for the controllers. This is important since such heuristics are often defined as functions of object properties, and thus assume direct knowledge of an object's shape, size, and overall geometry, which may not be easily available in the real world. Instead, we propose to use keypoint representations based on dense object descriptors to infer these parameters directly from visual data. Additionally, instead of using heuristics to provide axes-parameters we populate them automatically for each of the task-axes controllers. This results in a simple approach that allows the robot to learn complex manipulation tasks directly through interaction.

Our overall contributions include: 1) We extend object-centric task-axes controllers to infer the controller parameters directly from visual input. 2) Since learning both task-specific controller parameters and task-specific controller combinations (i.e. task-policy) together is a challenging problem, we propose to solve this problem by learning to bootstrap controller parameters using dense correspondence learning. 3) We empirically validate our approach on multiple manipulation tasks and show its generalization abilities across objects with different shape, size and geometry.

II. Related Work

Task Frames: Task frames (or task-spaces) have long been used by robotics community for robust task execution. Early works of [5]–[7] formalized the notion of task-axes and constraint based task-frames for manipulation tasks. For robust task execution, roboticists often design specific motions relative to some fixed task-frame or task-axes [8]–[11]. More recent works have proposed techniques to learn to select the appropriate task frame for the given task [12]–[16]. These methods use Imitation Learning (IL) combined with manually defined heuristics such as inter-trial variance between demonstrations to rank proposed task-frames. Recent work [4] have also proposed using Reinforcement Learning (RL) to choose multiple different controllers both sequentially and in parallel to complete a task. Each controller in [4] is defined with respect to some task axes or target keypoint. To avoid controllers at each step from interferring with each other null-space projections are employed. In this work, we further extend this line of research by learning controller targets and axes from visual observations.

Manipulation with Keypoints: We use keypoints to define targets for the different task-axes controllers. For this we build upon the recent work on using keypoints for manipulation learning [1], [2], [17]–[19]. Most closely related to our work are [1], [18], [20], all of which use keypoint representations for manipulation tasks. In [1] the authors use supervised learning to detect keypoints, while [20] uses self-supervised learning. However, both [1], [20] use keypoints to solve a task-specific optimization problem, whose solution in $SE(3)$ is used to perform the task in an open-loop manner. Concurrent to our work kPAM [1] was extended to use the notion of oriented keypoints [18] (kPAM 2.0), *i.e.* keypoints with local orientations. In addition to local orientations, [18] also uses object-centric actions with respect to these keypoints. Although the use of keypoints and local frames attached to keypoints is common between [18] and our work, there are some major differences. First, our approach disassociates keypoints and task-axes, *i.e.*, while kPAM 2.0 only uses local axes around a keypoint, our framework allows for any axes (e.g. global axes) with a keypoint. Second, instead of a fixed manually defined control policy [18], we use RL to learn a policy that composes parameterized task-axes controllers for task completion.

Learning with Parameterized Actions: Learning to select the task-axes controller to execute at each step (task-policy), as well as the target and axes parameter for this controller is closely related to learning with parameterized actions. Parameterized action spaces consider the problem where each action of an MDP is parameterized by a low dimensional input. In [21], parameterized action-MDPs were referred to as PAMDPs. PAMDPs are challenging to solve since they require a bilevel optimization algorithm, wherein the outer loop searches over the continuous action parameters, while the inner loop optimizes for the discrete action selection to complete the task. Since the inner loop requires solving an RL problem, this bilevel optimization is challenging to perform for high-dimensional spaces. Alternative works [22] propose to avoid this bilevel optimization by exploring both, the parameter-space and the action-selection space together. However, [22] uses uniform distributions to explore over action parameters, which is unsuitable when these parameters need to be inferred from high dimensional input, *e.g.*, selecting a pixel (keypoint) from an image. In current work, we avoid the computational complexity of this bilevel optimization by bootstrapping the controller parameters using multi-view correspondence learning.

III. Preliminaries

Manipulation tasks often involve different objects and manipulating them through contact to achieve desirable effects. The use of object-centric task-axes controllers provides a structured action space for the robot to both explore and act in. We briefly describe the different types of controllers and their associated parameters.

Controller Parameters: We use multiple controllers including position, force and rotation controllers. Each controller is associated with some underlying object in the scene and acts along some specific task axes. These task axes are centered at some task-specific target positions, such as object centers or other semantically useful points on the object *e.g.* edges or corners. Thus each controller has a small set of parameters associated with it. More specifically, position controllers are parameterized with (x_d, u), where $x_d \in \mathbb{R}^3$ is some position target, $u \in \mathbb{R}^3$ is the target axes along which the controller acts. Similarly, force controllers require (f_d, u), where $f_d \in \mathbb{R}$ is the force target to achieve. While rotation controllers require (r_d, u) where $r_d \in \mathbb{R}^3$ is the desired rotation target and u represents some axes of the end-effector which needs to be aligned with r_d. Finally, since each controller is either implemented as a PD, PI or PID controller, they also require suitable gain parameters. In this work, we focus on learning parameters that can be directly inferred from visual input, *i.e.*, the position targets (x_d), rotation target r_d and target-axes (u). We fix the other parameter values. From hereon we use the term *learned controller parameters* to refer to this initial set of parameters.

IV. Learning Controller Parameters

In [4], desired position targets x_d are defined using task-specific heuristics such as middle of the door handle or the middle of the wall as target positions. Task-specific knowledge is also required for task-axes parameter u and rotation targets r_d. For instance, for the door opening task only rotation controllers that align the downward end-effector axes with the door normal are used. By contrast, we want to infer these parameters from visual input and learn the appropriate parameters based on task interaction instead of heuristics or a priori information.

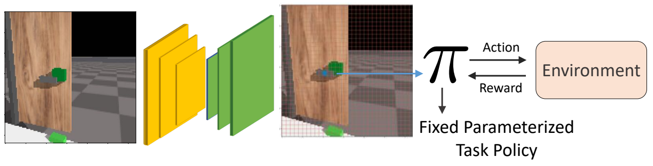

Fig. 2: Overview of proposed approach to learn keypoint parameters using the bootstrapped task policy.

As noted previously, to learn both — relevant x_d, u, r_d for each task-axes controller as well as the composition of these controllers for task completion, is a challenging problem. This is because it requires solving a bilevel optimization problem with an inner loop that involves solving an RL problem [21]. This challenge is further magnified when controller parameters have to be inferred from high dimensional inputs such as images, *e.g.*, 3D controller position target parameters can lie anywhere on the image space.

To overcome these challenges we propose to bootstrap learning the controller position parameters x_d through multi-view correspondence learning and a few human annotations. While we simultaneously infer a set of candidate axes parameters for both u and r_d from the given objects in the scene. Using these bootstrapped parameters we learn a task-axes controller composition policy π using RL. Using π we can subsequently learn x_d parameters for novel objects for which no human annotations are available. In the following subsections we explain the proposed approach in more detail.

A. Learning to Bootstrap Controller Parameters

We now discuss our approach to bootstrapping the position target and target-axes parameters. First, we look at the case of 3D position parameters x_d, and subsequently we look at the axes parameters u and r_d.

1) Keypoint Parameters: We refer to the 3D position target parameters (x_d) of each controller as keypoint parameters. These keypoint parameters are associated with semantically meaningful parts of underlying objects such as middle of the door handle, center of the button, edge of a block. To bootstrap learning the keypoint parameters, we propose to use multi-view dense correspondence learning. Specifically, we use DenseObjectNets [2], [17] which learn dense object descriptors for each pixel from multi-view data in a completely self-supervised manner. Not only does this setup avoid the need of any expensive manual data-labeling procedure, prior works have shown that the learned object descriptors are quite robust to the presence of mild occlusions and importantly, lead to category-level generalizations [2]. As we show empirically in Section VI, such category level generalization allows us to infer consistent controller targets irrespective of the object's size and position as well as variation in its shape and geometry.

To train dense object descriptors, we use a small set of objects (≈ 10) relevant to each task family and learn dense descriptors on these objects. All of these objects belong to the same category, *e.g.*, for the door opening task we learn descriptors across doors with varying sizes of door handles

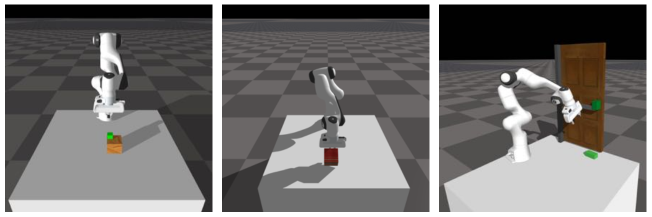

Fig. 3: Tasks used to evaluate our proposed approach. From *left* to *right*: Button Press, Block Tumble, and Door Opening.

as well as their locations on the door frame. Thus, the learned dense object descriptors should generalize to other novel objects that belong to the same category, *e.g.*, door handles with more complex shapes. Figure 6 visualizes some learned object descriptors. We refer to this dense-object network model as our *bootstrapped* keypoint model ϕ.

Given ϕ, we infer keypoint parameters for each controller by using a *reference* image (I_r) from the dataset collected for training dense descriptors. This reference image is used as representative for the object category being manipulated. To extract keypoints we manually label a set of reference pixels on I_r, denoted as $P := \{p_r^1, p_r^2, \cdots\}$. These reference pixels p_r^i encode semantic information about the scene which is relevant for the task. For instance, for the door open task we label pixels near the edge and middle of the door handle since these keypoints afford grasping and rotating the door handle compared to pixels closer to the handle's rotation joint. Similarly for the block tumble task we label pixels near the edge instead of the middle of the block. Figure 5 shows example keypoints for the tasks considered in our current work. Assuming known camera parameters we get 3D position targets $\{x_d^1, x_d^2, \cdots\}$ from $\{p_r^1, p_r^2, \cdots\}$ directly. To get position targets for a new image, which contains a novel object of the same category as I_r, we use pixels that are closest to the reference pixels in the descriptor space *i.e.*, $p^i = \arg\min_p \|\phi(p_r^i) - \phi(p)\|_2$, where p is any pixel on the image, and p^i is used to get the *i'th* controller target x_d^i.

2) Axes Parameters: Both rotation target r_d and target-axes u parameters require valid 3D axes. To find these axes we initially extract a relevant set of 3D axes for the given scene. We refer to this as the *candidate axes* set, $A := \{a^1, a^2, \cdots\}$. There exist multiple approaches to extract these candidate axes. For instance, possible candidate axes include both the global (world) axes as well as object axes. Additionally, we can also extract candidate axes by using the local geometry of the object around each inferred keypoint *e.g.*, using the axes normal to surface or along the surface. In this work, we directly use the object axes and the global axes as our candidate axes.

Given A the set of candidate axis, we can find u for each controller by finding the most relevant axes from A. However, this requires task-specific knowledge *e.g.*, in the form of user defined priors as used in [4]. Instead, we avoid this by associating each position target with every axis $a^j \in A$. Additionally, each axes in A can be used to create a unique rotation target r_d. One drawback of this approach is that it results in a combinatorial number of controllers, since we associate every position target with each axes in A. This can result in a large action space for π to explore. In Section VI, we empirically validate how this choice affects the sample complexity of using task-axes controllers for different manipulation tasks. Figure 1 visualizes the pipeline of our overall approach.

B. Learning Controller Parameters via Learned Task-Policy

Using the above approach we can bootstrap controller parameters and learn a task-specific manipulation policy π. This policy learns to compose different task-axes controllers to achieve the overall manipulation task. As we show empirically, this combination of bootstrapped controller parameters and learned task-policy can now be used on novel objects of varying shapes and sizes. This impressive level of generalization is a result of both the category level generalization provided by dense-keypoint parameters and the object-centric nature of task-axes controllers.

However, in some scenarios we may want to use the learned task policy on objects of a different category as compared to objects used to learn the bootstrapped keypoint parameters. For such cases, the keypoints p^i inferred by the bootstrapped dense-object network (ϕ) can be unsuitable. Rather than re-training the dense-object network on this new set of objects and relying on human annotations to get corresponding reference pixels P, we can instead utilize the learned task policy π and learn keypoint parameters directly from the image space. This is possible since π is implicitly parameterized by the controller target parameters x_d. Hence, we can learn x_d's value for the new set of objects based on the feedback from evaluating π. This assumes that π is approximately similar across both sets of objects, *i.e.*, the set of objects used during bootstrap learning and the new set of objects. This assumption holds for the tasks we consider, *e.g.*, to learn a door opening policy, we only need to reach close to the handle, grasp it from a location which affords grasping, rotate the handle and pull it.

To learn keypoint parameters for a new set of objects, we learn a new neural network ψ_θ, with weights θ, by rolling out π and learning to optimize the underlying task reward $J(\theta) = \mathbb{E}_{\tau\sim\pi(\cdot|s,\psi_\theta(I_s))}\left[r(\tau)\right]$, where $\psi_\theta(I_s)$ denotes the keypoint parameters inferred from image I_s at state s, and π represents the learned task policy. We do not update π during this keypoint learning stage. $J(\theta)$ can be optimized using any policy gradient method *e.g.* Reinforce [23], $\nabla J(\theta) = \mathbb{E}_\tau\left[r(\tau)\nabla\log\psi_\theta(s)\right]$. We represent ψ_θ using a fully-convolutional network, with the last layer of the network containing as many channels as the number of keypoint parameters to infer. We use a softmax over each last channel and sample the appropriate pixel to get p^i which is then used to infer the controller target x_d^i. Figure 2 provides an overview of our approach.

V. Experimental Setup

With our experiments we aim to evaluate: 1) How well does our proposed approach using bootstrapped controller parameters perform? Since a combinatorial mix of axes and keypoint parameters leads to a large action space, we investigate its affect on task performance and sample complexity. 2) How well does the combination of dense-object net based keypoint model and object-centric task-axes controller generalize to new objects of varying shapes, sizes and geometry? 3) How well can a learned task policy be utilized to learn keypoint controller parameters for new set of objects?

A. Tasks

We evaluate our approach on 3 different manipulation tasks (Figure 3) of increasing complexity – Button Press, Block Tumble, and Door Opening.

Button Press: In this task, a 7-DoF Franka Panda arm is used to push down a button which is positioned on a box placed infront of the robot (Figure 3 left). Instead of using only one type of button object, we use multiple objects with different shapes and sizes. These variations include the button position on top of the box object, sizes of both the button as well as its underlying box object.

Block Tumble: In this task, a 7-DoF Franka Panda arm is required to tumble a block along a particular axis (Figure 3 middle). This task is particularly interesting since there exist multiple different ways to accomplish it. For instance, the robot can tumble the block by applying a downward force anywhere along its edge. To test generalization for this task, we vary the size of the block between 0.07m to 0.16m.

Door Opening: In this task the Franka robot needs to open a door by first turning its door handle and then pulling the door beyond an opening threshold. In contrast to [4], which tests generalization by only varying the position of the door handle on the door frame, we vary both the size of the door handle and the location of the door handle on the door. Additionally, we also change the door shape and evaluate on cuboidal, cylindrical and more complex door handle shapes.

B. Compared Approaches

We compare our proposed approach against multiple different methods. In the below sections we refer to task-axes controllers using the shorthand TAC.

1) **EE-Space:** We verify the utility of the structured action space provided by object-centric task-axes controllers by comparing against an approach that directly controls the robot via end-effector delta targets.
2) **TAC (Manual)** We also evaluate our approach against manually specified controller parameters. We note that we do not aim to use the *smallest* possible number of controllers and their parameters. Instead, we specify the controllers and their parameters only to provide some useful priors for overall task learning.
3) **TAC (Keypoints):** We evaluate one version of our proposed approach in which we only infer the keypoint parameters *i.e.* the target positions for each position or force controller. We reuse the axes specified for TAC (Manual) with the inferred keypoints.

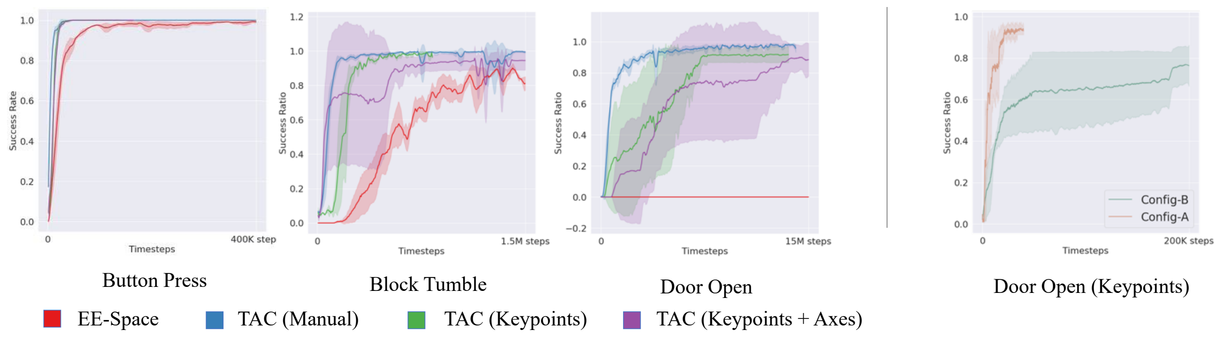

Fig. 4: *Left:* Task Success Rate for all 3 environments with bootstrapped controller parameters. *Right:* Task success rate on two different configs (door-handle types) for learning keypoints using a learned policy for the Door-Open Task. Figure 8 visualizes both configs. The dark line shows mean sucess-ratio, shaded region plots std across 5 seeds.

Env	EE-Space	TAC (Manual)*	TAC (Keypoints+Axes)
Button Press	0.98 (0.01)	1.0 (0.0)	1.0 (0.0)
Block Tumble	0.487 (0.26)	0.96 (0.03)	0.932 (0.04)
Door Opening	0.0 (0.0)	0.97 (0.01)	0.94 (0.05)

TABLE I: Mean (std) for each method on all three different manipulation tasks.

4) **TAC (Keypoints+Axes):** We evaluate our proposed approach wherein both the keypoints as well as the axes parameters are inferred for each scene.

Metrics: We show qualitative and quantitative results for two scenarios. First, we show results for learning task policy using bootstrapped controller parameters. Second, we use this learned task policy to show results for learning controller parameters using direct interactions. For both scenarios we compare approaches using the success ratio metric.

RL Training: For training the task policy we use Proximal Policy Optimization (PPO) [24] based on stable-baselines [25]. While for learning controller parameters we found a simple Reinforce [23] based approach to be sufficient. All results are run and reported for 5 different seeds.

VI. Results

A. Learning Task Policy

1) Training Results: Figure 4 plots success ratio for all approaches and tasks. As seen above, we observe that for the simplest task *i.e.* button press, all methods are able to learn the task quickly. Also, since the underlying task is not complex, each method has little variance across multiple seeds. For the Block Tumble task (Figure 4 middle), although all methods perform well on the training task, methods that use task-axes controllers are much more sample efficient. This is true even when we use a much larger set of controllers *i.e.* the TAC (Keypoints + Axes) approach. This is because most of the keypoints used in the task (Figure 5) can be used to accomplish the task. Also, since there is only one axes along which the block needs to be flipped, the robot is quickly able to find this relevant axes using the provided dense rewards. Thus, inferring the keypoints and axes parameters does not affect the sample complexity significantly.

Figure 4 (right) plots success ratios for the door opening task. From this figure, we observe that the EE-space is unable to solve the overall task. Similar results were also observed in [4]. The main reason for this failure is the overall task complexity, especially since task completion requires different subtasks (reaching, grasping, turning the handle and pulling it back) to be performed in sequence. On the other hand, TAC based methods perform well on the task. However in contrast to the previous tasks, TAC (Keypoints+Axes) does require more samples as compared to TAC (Manual). Additionally, only inferring the keypoints and not the axes (TAC-Keypoints) is still quite sample efficient. This indicates that the agent in TAC Keypoints+Axes does spend initial time exploring different axes which can be used to accomplish the task. This is possible because there exist multiple ways to grasp the handle, *e.g.*, it is possible to grasp the handle both along the vertical and the horizontal axes. However, using the vertical axes is not robust, since it can easily collide with the door frame. Thus, as a large number of actions are not particularly useful for the task, the agent will have to interact and learn the most suitable and robust ways to achieve it.

2) Generalization Results: Table I shows the generalization performance for three different methods. This generalization performance was recorded on 15 different environment settings with varying object sizes and shapes. We note that for TAC (Manual) we only used primitive shapes (cuboids and cylinders) since we need to manually provide keypoint parameters. See Figure 7 for test configurations used for TAC (Keypoints+Axes), for door open task. As seen in Table I, methods that use TAC are able to generalize quite well across all tasks. On the other hand, the EE action-space provides good generalization capabilities only for the simplest task (Button Press). While even for the moderately complex Block Tumble task its performance reduces significantly. One reason for this is the lack of any inductive bias in the EE-space, and since we train on a small set of objects only, the EE-space policy fails to generalize to large variations in objects. Figure 7 shows some objects with complex underlying geometry that were never used either for dense descriptor or policy training. While TAC (Manual) cannot be applied to such objects, we show in the attached video results that our method is successfully able to zero-shot generalize to such large variations as well.

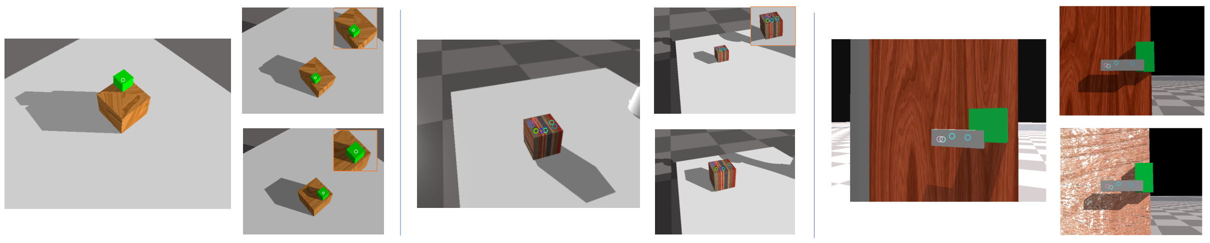

Fig. 5: Visualization for *reference* images and pixels (left image in each column) and corresponding pixels predicted using learned descriptors. For door open, one reference pixel is closer to door joint to show that our approach learns to not use it.

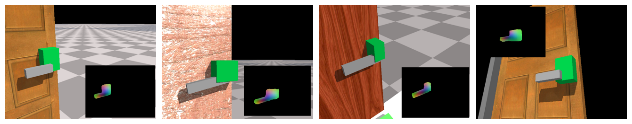

Fig. 6: Dense object descriptor results for Door Open Task.

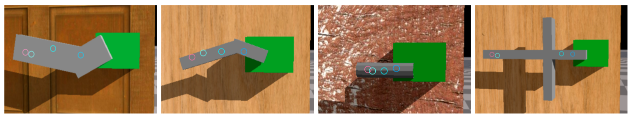

Fig. 7: *Qualitative Results* we show that our learned control policy although not trained on any of the above models does successfully transfer to them (see results in project-page).

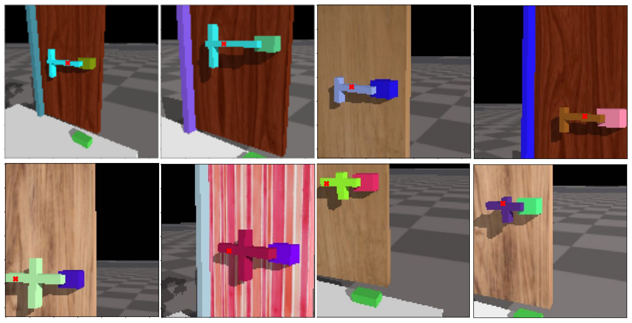

Fig. 8: Keypoint Parameters learned using a learned task policy. Top Row is Config-A, bottom row is Config-B.

3) Qualitative Results: We show qualitative results for both descriptors and keypoints. Figure 5 plots the keypoints used for each task. The left image in each column is the *reference* image annotated with *reference* keypoints. While the right column shows scenes with 2 *different* test set objects used to evaluate the learned policies. Figure 6 plots the learned descriptors for the door opening task. As seen above, the learned descriptors are able to approximately cluster semantically meaningful regions together, *e.g.*, the handle part close to its rotation joint, the middle and end of the door handles, as well as the right end of the hinge are all well estimated. While in Figure 7 we see that the reference pixels are also able to generalize to objects with very different shapes and geometry. We show policy results for these samples in our video. This is not surprising since as the above keypoints afford grasping and rotating the door handle, the underlying task-axes controllers should be able to generalize. See video results for all tasks at `https://sites.google.com/view/robotic-manip-task-axes-ctrlrs`.

B. Learning Controller Parameters with Learned Task Policy

We now show results (Fig 4 (right)) for learning controller parameters (keypoints) using the learned task policy. We only use one keypoint for this task policy, since it is sufficient to perform the door opening task. This task policy is trained on simple door handles only, visualized in Figure 6. While the keypoint policy is trained on 2 completely different door-handle configurations, visualized in Figure 8. Additionally, as seen in Figure 8, we ensure that the door handle can lie anywhere on the image space and not necessarily near the image center or always orthogonal to the camera. This makes the keypoint learning problem much more challenging.

Although both configurations (config-A and config-B) are visually quite similar, they result in very different sample complexities when learning keypoint parameters (Figure 4 (right)). This is because for config-A there exists a much larger region of the door handle that can be selected to perfectly execute the learned parameterized task-policy. This can be seen in Figure 8 (top-row) where different keypoints along the door handle have been selected to successfully open the door. While for config-B this valid region is quite small (to the left of the vertical handle bar). Additionally, small inconsistencies in keypoint predictions, such as visualized in Figure 8 (bottom-row right) fail to open the door. This makes the keypoint learning problem much more challenging, which consequently results in a much larger sample complexity.

VII. Conclusion

In this paper we propose a modular architecture for learning manipulation tasks. Our approach uses task-specific keypoints on objects and task-axes controllers parameterized by these keypoints for task learning. As we show empirically, our approach improves generalization and works well on a large set of objects beyond the small set used during training. Additionally, a modular architecture allows us to reuse the control policy while retraining the perceptual network on a new set of objects and still perform the underlying task. This is in contrast to end-to-end DeepRL approaches where the perceptual network is closely tied with the control policy and it is not possible to update one without the other. Finally, our approach also improves the interpretability of the learned models. This is a consequence of both semantic keypoints and task-axes controllers which operate on semantic inputs.

References

[1] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, “kpam: Keypoint affordances for category-level robotic manipulation,” *arXiv preprint arXiv:1903.06684*, 2019.

[2] P. R. Florence, L. Manuelli, and R. Tedrake, “Dense object nets: Learning dense visual object descriptors by and for robotic manipulation,” in *Conference on Robot Learning*, 2018, pp. 373–385.

[3] A. Ganapathi, P. Sundaresan, B. Thananjeyan, A. Balakrishna, D. Seita, J. Grannen, M. Hwang, R. Hoque, J. E. Gonzalez, N. Jamali *et al.*, “Learning to smooth and fold real fabric using dense object descriptors trained on synthetic color images,” *arXiv preprint arXiv:2003.12698*, 2020.

[4] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, “Learning to compose hierarchical object-centric controllers for robotic manipulation,” *arXiv preprint arXiv:2011.04627*, 2020.

[5] M. T. Mason, “Compliance and force control for computer controlled manipulators,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[6] M. H. Raibert and J. J. Craig, “Hybrid position/force control of manipulators,” 1981.

[7] D. H. Ballard, “Task frames in robot manipulation.” in *AAAI*, vol. 19, 1984, p. 109.

[8] D. Berenson, S. Srinivasa, and J. Kuffner, “Task space regions: A framework for pose-constrained manipulation planning,” *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[9] J. E. King, M. Cognetti, and S. S. Srinivasa, “Rearrangement planning using object-centric and robot-centric action spaces,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3940–3947.

[10] T. Migimatsu and J. Bohg, “Object-centric task and motion planning in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[11] S. Manschitz, M. Gienger, J. Kober, and J. Peters, “Learning sequential force interaction skills,” *Robotics*, vol. 9, no. 2, p. 45, 2020.

[12] M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick, “Automatic selection of task spaces for imitation learning,” in *2009 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2009, pp. 4996–5002.

[13] J. Kober, M. Gienger, and J. J. Steil, “Learning movement primitives for force interaction tasks,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3192–3199.

[14] A. L. P. Ureche, K. Umezawa, Y. Nakamura, and A. Billard, “Task parameterization using continuous constraints extracted from human demonstrations,” *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1458–1471, 2015.

[15] L. Peternel, L. Rozo, D. Caldwell, and A. Ajoudani, “A method for derivation of robot task-frame control authority from repeated sensory observations,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 719–726, 2017.

[16] A. Conkey and T. Hermans, “Learning task constraints from demonstration for hybrid force/position control,” in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2019, pp. 162–169.

[17] P. Florence, L. Manuelli, and R. Tedrake, “Self-supervised correspondence in visuomotor policy learning,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 492–499, 2019.

[18] W. Gao and R. Tedrake, “kpam 2.0: Feedback control for category-level robotic manipulation,” *IEEE Robotics and Automation Letters*, pp. 1–1, 2021.

[19] K. Zakka, A. Zeng, J. Lee, and S. Song, “Form2fit: Learning shape priors for generalizable assembly from disassembly,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2020.

[20] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese, “Keto: Learning keypoint representations for tool manipulation,” 2019.

[21] W. Masson, P. Ranchod, and G. Konidaris, “Reinforcement learning with parameterized actions,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[22] M. Hausknecht and P. Stone, “Deep reinforcement learning in parameterized action space,” *arXiv preprint arXiv:1511.04143*, 2015.

[23] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[25] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines,” https://github.com/hill-a/stable-baselines, 2018.